SmartFace Embedded Toolkit

4.2.1

Generated by Doxygen 1.9.7

Chapter 1

Overview

SmartFace Embedded Toolkit (SFE Toolkit) is a modular, portable and easy-to-integrate SDK, that can be used in any facial recognition use case on various platforms.

1.1 Key Features

SFE Toolkit currently supports the following features.

1.1.1 Face Detection

SmartFace Embedded can detect faces of various sizes, orientations, facial hair types or ethnicities.

1.1.2 Face Quality Evaluation

SFE Toolkit provides functionality to measure the quality of detected faces to improve the quality of enrollment and accuracy of face recognition.

SFE Toolkit supports the following face attributes:

- Head pose deviation (yaw, pitch, roll)
- Brightness in the facial region
- Contrast in the facial region
- Sharpness in the facial region
- Uniform Lighting / Shadow in the facial region
- ICAO cropping of the facial region.

1.1.3 Face Recognition

- Face verification (1:1)
- Face identification (1:N)
- Face liveness evaluation

1.1.4 Iris Recognition

- Iris detection
- Iris verification (1:1)
- Iris identification (1:N)

1.1.5 Palm Recognition

- Palm detection
- Palm verification (1:1)
- Palm identification (1:N)
- Palm attributes

1.2 Platform support

SFE Toolkit supports the following platforms:

- Windows x86_64,
- Linux x86_64,
- Android ARMv7 oand ARM64
- Embedded Linux ARMv7 and ARM64

If you need to support another OS or architecture, please contact your sales representative at Innovatrics.

1.2.1 HW acceleration

- Ambarella CVFlow (Ambarella SDK 3.0)
  - CV28
  - CV25
  - CV22
  - CV2
- Rockchip RKNPU (RKNPU 1.6.0)
  - RK1808/RK1806
  - RV1109/RV1126
- Rockchip RKNPU2 (RKNPU2 1.5.2)
  - RK3566/RK3568
  - RK3588/RK3588S
  - RV1103/RV1106
  - RK3562
- NVidia GPUs and NVidia Jetson platforms with NVIDIA CUDA or TensorRT support
- NXP iMX8 (Tensorflow Lite -> VX delegate)

1.3 Package

Each of the SFE Toolkit packages consists of:

- root directory includes:
  - this readme file,
  - changelog,
  - EULA,
- `doc` directory includes documentation in HTML and PDF format,
- `lib` directory includes shared libraries compiled for one target CPU,
- `solver` directory includes a set of solvers compiled to be accelerated using a specific neural network accelerator,
- `include` directory includes header files with documentation of the C API,
- `bin` directory includes:
  - pre-built example applications demonstrating the usage and integration of the SFE Toolkit libraries,
  - pre-built becnhmark applications for benchmarking of the SFE Toolkit libraries,
  - License Manager application compiled for the target CPU,
- `example` directory includes sources of example applications,
- `benchmarks` directory includes sources of benchmark applications,
- `assets` directory includes images for example and benchmark applications.

1.4 Libraries

SFE Toolkit consists of the following libraries:

- libsfe_core.so (sfe_core.dll on Windows)
  - solver loading
  - image operations
  - error handling
- libsfe_face.so (sfe_face.dll on Windows)
  - face detection
  - face landmarks detection
  - face mask detection
  - face template extraction
  - 1:1 face template matching
  - 1:N face template identification
  - face liveness detection
  - face and image quality attributes
- libsfe_iris.so (sfe_iris.dll on Windows)
  - iris detection

  - iris template extraction
  - 1:1 iris template matching
  - 1:N iris template identification

- libsfe_palm.so (sfe_palm.dll on Windows)
  - palm detection
  - palm template extraction
  - 1:1 palm template matching
  - 1:N palm template identification

- libsfe_tattoo.so (sfe_tattoo.dll on Windows)
  - tattoo detection

1.4.1 API

SFE Toolkit libraries provide C API. We can also provide a binding with an example application for the following platforms:

- Python for Windows and Linux
- Kotlin for Android
- Swift for iOS
- .NET for Windows and Linux

1.5 Licensing

You will need a valid license file to use the SFE Toolkit.

1.5.1 Hardware ID

The hardware ID of your Embedded/Edge device or PC is required to generate the valid license.

Please use the License Manager application to generate a Hardware ID for your device:
```
./license_manager_cli -p
```

On Embedded platforms, the MAC address of the device's network adapter is used as the hardware ID.

NOTE: If you require a license based on a different type of hardware parameter of your edge device, please contact your Innovatrics sales representative.

1.5.2 License generating

Use the hardware ID or MAC address (without colons) of your device to generate a license at our Customer Portal.

1.5.3 License deployment

Copy this license file to one of the following locations on Linux PC and Embedded Linux:

- `/etc/innovatrics` (all users)
- `~/.innovatrics` (specific user)
- working directory

and on Windows PC:

- C:\ProgramData\Innovatrics\iengine.lic (all users)
- C:\Users<user>\AppData\Local\Innovatrics\iengine.lic (specific user)
- working directory

The name of the license file has to be renamed to `iengine.lic`.

License ENV variables

The license can be also set using environment variables:

- **ILICENSE** - path to the license file
- **ILICENSE_DATA** - base64 (RFC 4648) encoded license file content. Please note you have to disable line wrapping. To convert the license file to base64 correctly, please use the following command:
  ```
  cat iengine.lic | base64 -w0 > base64_license.txt
  ```

1.6 Example application

To run the pre-built example applications run the following commands in the package root directory:
```
export LD_LIBRARY_PATH=$PWD/lib
bin/example_face_identify
```

To display options run:
```
bin/example_face_identify -h
```

1.6.1 Python example

To run the Python example, run the following command:

On Linux:
```
cd example/python
export LD_LIBRARY_PATH=../../lib
python example_face.py
```

On Windows:

1. Copy `sfe_*` DLLs to `C:\Windows\System32` or working directory (`example/python`)
2. Copy DLLs from `bin/` directory to `python.exe` installation directory because there is an incompatible `onnxruntime.dll` installed in Windows 10 and 11.
   - `onnxruntime.dll`, `zlibwapi.dll`
   - OPTIONALLY: `onnxruntime_providers_*` and CUDA (`cudnn*`) DLLs to enable GPU acceleration using `cuda` or `tensorrt` runtime provider
3. Run `python.exe example_face` from `example/python` directory.

1.7 Benchmarks application

To run the pre-built benchmarks applications run the following commands in the package root directory:

```
export LD_LIBRARY_PATH=$PWD/lib
bin/benchmark_face
```

To display options run:

```
bin/benchmark_face -h
```

If you want to build the application, please follow the instructions in example/readme.md

1.8 Support

Please contact sfembedded-integration@innovatrics.com in case of any issues or questions.

Chapter 2

Changelog

All notable changes to this project are documented in this file.

2.1 [4.2.1] - 2026-03-16

2.1.1 Fixed

- Documentation missing in the package [BSDK-545]
- Fix minor documentation inconsistencies

2.2 [4.2.0] - 2026-03-09

2.2.1 Fixed

- Fixed incorrect Face extraction preprocessing that affected accuracy. [BSDK-486]

2.2.2 Breaking Changes

- Face templates extracted using prior v4.x toolkit must be re-extracted.

2.3 [4.1.0] - 2026-02-18

2.3.1 Added

- Optional Palm landmarks on detection: palm detections can now carry attached landmarks, so you can skip calling palm landmark extraction when the detector already provides them.
- C API: `SFEPalmLandmarks` and `SFEPalmDetectionData` with `has_landmarks` and `landmarks`.
- UniFFI: `PalmLandmarks` dictionary and `Modality::Palm` with optional `landmarks`.

2.3.2 Fixed

- Palm landmark extraction reuses landmarks already on the detection when present. Fixes [BSDK-484].

2.4 [4.0.4] - 2026-02-12

2.4.1 Added

- Face template export/import support [BSDK-463]
- C API: `sfeFaceTemplateExport()`, `sfeFaceTemplateImport()`
- UniFFI: `template_export`, `template_import`

2.5 [4.0.3] - 2026-02-09

2.5.1 Fixed

- Fixed incorrect score normalization in Face match/identification [BSDK-479]

2.6 [4.0.2] - 2026-02-04

2.6.1 Fixed

- Updated Android NDK to r28c to fix 16k page size issues [BSDK-459]
- Updated `SFEImageFormat` definition to match implementation
- More predictable identification performance on various platforms

2.7 [4.0.1] - 2026-01-30

2.7.1 Added

- Support for TFLite and TensorFlow models

2.7.2 Changed

- Fixed missing mutli-detection models in all packages
- Fixed missing models in Ambarella packages

2.8 [4.0.0] - 2026-01-29

2.8.1 Breaking Changes

Unified Detection API

All modality-specific detection types and functions have been replaced with a unified detection system:

- New `SFEDetection` struct replaces all modality-specific detection types (`SFEFaceDetect`, `SFEIrisDetect`, `SFEPalmDetect`, `SFEObjectDetect`, `SFETattooDetection`, `SFEPersonOrientedDetect`)
- New `sfeDetect()` function replaces modality-specific detection functions (`sfeFaceDetect`, `sfeIrisDetect`, `sfePalmDetect`, `sfeObjectDetect`, `sfeTattooDetect`)
- Detection modality is indicated by `SFEModality` enum field in `SFEDetection`
- Field names standardized: `bbox` → `bounding_box`, `bbox_confidence` → `confidence`

Unified Tracker API

All modality-specific tracker functions have been replaced with unified tracker functions:

- `sfeDetectionTrackerCreate()`, `sfeDetectionTrackerUpdate()`, `sfeDetectionTrackerFree()`, `sfeDetectionTrackerLost()`, `sfeDetectionTrackerRemoved()`
- Tracker creation now requires 5 parameters with new `new_track_thresh` parameter for minimum confidence to initialize a new track
- `max_time_lost` parameter type changed from `size_t` to `uint64_t`

Core API Changes

- Type renames:
  - `SFELicenseSource` → `SFELicenseType`
  - `SFE_LICENSE_SOURCE_*` → `SFE_LICENSE_TYPE_*`
  - `SFEBbox` → `SFEBoundingBox`
  - `SFEOrientedBbox` → `SFEOrientedBoundingBox`
- Function renames:
  - `sfeImageLoad()` → `sfeImageDecode()`
  - `sfeImageSave()` → `sfeImageEncode()`
  - `sfeSolverCreateWithParameters()` → `sfeSolverCreate()`
- Signature changes:
  - `sfeImageInfo()`: Now returns `SFEImageInfo` struct instead of separate output pointers
  - `sfeImageColorTranspose()`: Now returns new image via `out_image` parameter instead of in-place modification
  - `sfeLicenseValidate()`: No longer returns license source; use new `sfeLicenseType()` function instead

Face API Changes

- `SFEFaceDetectAccuracyType` → `SFEFaceDetectionAccuracyType`
- `sfeFaceLandmarks()`: Now accepts `const SFEDetection *` instead of `SFEBbox`
- `sfeFaceTemplateExtract()`: Now requires `const SFEDetection *` parameter
- `sfeFaceLivenessPassive()`: Now returns `SFEFaceLiveness` struct instead of float
- `sfeFaceDetectInputSize()`: Now returns `SFEDetectionInputSize` struct
- `sfeFaceCrop()`: `face_size_extension` parameter changed from `uint32_t` to `float`

Palm API Changes

- New `SFEPalmLandmarks` struct and `sfePalmLandmarks()` function for palm landmark extraction
- `sfePalmTemplateExtract()`: Now accepts `const SFEPalmLandmarks *` instead of detection
- `sfePalmAttributes()`: Now accepts `const SFEPalmLandmarks *` instead of detection
- `sfePalmLivenessPassive()`: Now accepts `const SFEDetection *`

Person API Changes

- `sfePersonAttributes()`, `sfePersonPose()`, `sfePersonTemplateExtract()`: Now accept `const SFEDetection *` instead of `SFEBbox`

Iris API Changes

- `sfeIrisTemplateExtract()`: Now accepts `const SFEDetection *`

Tattoo API Changes

- `sfeTattooTemplateExtract()`: Now accepts `const SFEDetection *`
- `sfeTattooTemplateVersion()`: Now returns `SFETattooTemplateVersion` struct

Other Breaking Changes

- `SFEEntityIdentificationCandidate` struct now includes `index` field

2.8.2 Added

- Unified detection API with `sfeDetect()` supporting all modalities
- New `multi_detect_bb` detection solver for Face, Person, Palm and QR detection
- Unified tracker API with `sfeDetectionTracker*()` functions
- `SFEDetection` struct with `SFEModality` discriminator for type-safe detection results
- `SFEImageInfo` struct for image metadata
- `SFEDetectionInputSize` struct for detection input size calculation
- `SFEPalmLandmarks` struct and `sfePalmLandmarks()` function
- `SFEFaceLiveness` struct for liveness results
- `sfeLicenseType()` function to query current license type

2.8.3 Changed

- Improved error messages with more detailed explanations
- Solvers are now validated for compatibility on load
- New face extraction models: balanced, fast, accurate (replacing legacy models)
- Updated `person_detect_fisheye` solver

2.8.4 Removed

- Modality-specific detection structs: `SFEFaceDetect`, `SFEIrisDetect`, `SFEPalmDetect`, `SFEObjectDetect`, `SFETattooDetection`, `SFEPersonDetectOriented`
- Modality-specific detection functions: `sfeFaceDetect()`, `sfeIrisDetect()`, `sfePalmDetect()`, `sfeObjectDetect()`, `sfeTattooDetect()`
- Modality-specific tracker functions: `sfeFaceTracker*()`, `sfeIrisTracker*()`, `sfePalmTracker*()`, `sfeObjectTracker*()`
- Deprecated type aliases: `SFEFaceEntity`, `SFEIrisEntity`, `SFEPalmEntity`, `SFEPersonEntity`, `SFEObjectEntity`, `SFETattooEntity` (use `SFEEntity`)
- Deprecated type aliases: `SFEIdentificationResult`, `SFE*TemplateIdentificationCandidate` (use `SFETemplateIdentificationCandidate`)
- Deprecated type aliases: `SFE*EntityIdentificationCandidate` (use `SFEEntityIdentificationCandidat`
- `SFEPalmQualityAttributes` and `sfePalmQualityAttributes()` (use `SFEPalmAttributes` and `sfePalmAttributes()`)
- Legacy face extraction models

2.9 [3.4.0] - 2026-01-19

2.9.1 Added

- Added support for fast evaluation of demographic attributes (BSDK-412)

2.10 [3.3.1] - 2026-01-13

2.10.1 Added

- Added `sfeLicenseType` function to get the license type (BSD-302)

2.10.2 Changed

- Python module now searches for native libraries also in system directories and LD_LIBRARY_PATH (BSDK-303)

2.10.3 Fixed

- Fixed an issue where Python examples from the Conan package could not be run without a prior pip installation (BSDK-303)
- Tattoo detection bounding box is clamped to the 0,1 range (BSDK-385)
- Empty tattoo detection bounding box is not propagated (BSDK-385)

2.11 [3.3.0] - 2025-12-10

2.11.1 Added

- Added demographic attributes (BSDK-229, BSDK-384)

2.11.2 Fixed

- Missing tattoo solvers are added into conan package (BSDK-383)

2.12 [3.2.4] - 2025-12-03

2.12.1 Changed

- Implementation of tattoo extraction uses real solver (BSDK-282)

2.13 [3.2.3] - 2025-11-28

2.13.1 Changed

- Implementation of tattoo detection uses real solver (BSDK-284)

2.14 [3.2.2] - 2025-11-05

2.14.1 Fixed

- Minor documentation fixes.

2.15 [3.2.1] - 2025-10-29

2.15.1 Changed

- All android libraries now use `16 kB page sizes`, as mandated by Google for architectures `x86_64` and `arm64-v8a` (BSDK-279)

2.16 [3.2.0] - 2025-10-14

2.16.1 Breaking Change

- Fixed Palm template extraction for `palm.extraction.resnet50`; template now reports version v1.1.0 and invalidates older templates. [BSDK-291]

2.17 [3.1.5] - 2025-10-10

2.17.1 Added

- Added new `face_detect_fast` model [BSDK-26]

2.18 [3.1.4] - 2025-10-06

2.18.1 Added

- Publishing aarch64-unknown-linux-gnu and aarch64-apple-darwin Conan packages to Artifactory [BSDK-238]

2.18.2 Changed

- Updated palm extraction model palm.extraction.resnet50 [BSDK-269]
  - Improved performance on Intel CPUs
  - Updated onnxruntime_solver to version 0.10.25

2.18.3 Fixed

- Revised outdated documentation and fixed run issues in code examples. [BSDK-251]

2.19 [3.1.3] - 2025-09-29

2.19.1 Fixed

- Added missing tattoo extraction methods into UniFFI API [BSDK-245]

2.20 [3.1.2] - 2024-09-25

2.20.1 Changed

- Updated `face_liveness_passive_nearby_fast.tflite.solver` with better accuracy on NXP NPUs [BSDK-234]
- tflite_solver updated to v0.10.11 [BSDK-234]

2.21 [3.1.1] - 2025-09-18

2.21.1 Added

- Added package for aarch64-unknown-linux-gnu [BSDK-238]
- Added package for aarch64-apple-darwin [BSDK-237]
- Added `tattoo` module to Kotlin

2.21.2 Changed

- Improved accuracy of `face_liveness_passive_nearby_fast` preprocessing [BSDK-268]
- Updated palm extraction model to palm.extraction.resnet50 (by upgrade onnxruntime_solver to version 0.↩ 10.24) [BSDK-262]

2.21.3 Fixed

- Missing sfeImageFree in example_face_identify.cpp

2.22 [3.1.0] - 2025-09-12

2.22.1 Added

- Added detection API for new `tattoo` modality [BSDK-244]
- Added extraction API for new `tattoo` modality [BSDK-245]

2.22.2 Changed

- Updated `face_liveness_passive_nearby_fast` [BSDK-233]
- Updated Person fisheye detection model
- Updated ilicense to v3.4.3 [BSDK-259]
- Updated onnxruntime_solver to version 0.10.22 [BSDK-245]

2.23 [3.0.2] - 2025-08-21

2.23.1 Breaking Change

- Removed deprecated `palm::metrix` functionality

2.23.2 Added

- New `core::OrientedBondingBox` type for oriented detectors (BSDK-213)
- Added `person::detect_oriented` (BSDK-213)

2.24 [3.0.1] - 2025-08-18

2.24.1 Added

- Added palm_liveness_fast model for Rockchip RKNN2 (BSDK-121)

2.24.2 Changed

- Updated Rockchip RKNN2 solver version to 0.10.7 (BSDK-121)

2.24.3 Fixed

- YoloX square padding preventing unnecesary upscales and errors with fixed shape solvers

2.25 [3.0.0] - 2025-08-12

2.25.1 Breaking Change

- The `AttributesResult.quality` is now in range <0,1> instead of <0-100>.

2.25.2 Changed

- Updated palm attributes model (BSDK-204)
- Updated palm liveness accurate model (BSDK-197)
- Updated palm liveness fast model (BSDK-198)
- Updated onnxruntime_solver to version 0.10.18
- Updated ILicense to version 3.4.2

2.26 [2.0.2] - 2025-07-18

2.26.1 Changed

- Updated Rockchip RKNN2 solver version to 0.10.6 (BFO-51)
  - Updated rknn_api_v2 to version 2.3.2
  - Fixed palm_template_extract_accurate model output name
  - palm_template_extract_accurate and palm_detect_accurate model converted with latest rknn_toolkit↩ _2.3.2 to be compatible with rknn librknnrt 2.3.2

2.27 [2.0.1] - 2025-07-08

2.27.1 Changed

- The palm accurate detector performance for rockchip RKNNv2 is improved

2.28 [2.0.0] - 2025-07-04

2.28.1 Breaking Change

- To ensure consistent terminology across SDKs
  - in C
    - Renamed enum `SFEHandSide` to `SFEHandPosition`
    - Renamed `hand_side` in `SFEPalmDetect` to `hand_position`
  - other languages
    - Renamed enum `HandSide` to `HandPosition`
    - Renamed `hand_side` in palm `Detection` to `hand_position`

2.28.2 Deprecated

- `sfePalmQualityAttributes` is deprecated and will be removed soon. Please use `sfePalm↩Attributes` instead.

2.28.3 Fixed

- Added missing palm solvers in Rockchip RKNN2 packages

2.28.4 Added

- Added new function `sfePalmAttributes` that provides `SFEPalmAttributes` with (BSDK-123):
  - `hand_orientation`
  - `hand_position`
  - `quality`

2.29 [1.15.7] - 2025-06-25

2.29.1 Added

- Updated Rockchip RKNN2 solver version to 0.10.4 (BFO-80):
- Added Rockchip RKNN2 palm accurate detector support
- Added Rockchip RKNN2 palm accurate extractor support
- Added logging support to ONNX solvers with configurable log level and log file output
- Added parameter `log_level` (empty for no log, or set to verbose, info, warning, error, fatal)
- Added parameter `log_file` (empty for stdout, or specify file path)
- Added parameter `profile_prefix` to dump JSON stats from inference
- Added `session_config` parameter to support advanced session settings in ONNX solvers

2.29.2 Changed

- Updated onnxruntime_solver to v0.10.14
- Updated fast palm detector model and postprocessing (BSDK-146)

2.29.3 Fixed

- Inference performance on Intel CPUs due to subnormal numbers (BSDK-182)
- Fixed NNAPI FP16 flag logic in ONNX solvers by correcting inverted condition

2.30 [1.15.6] - 2025-06-18

2.30.1 Fixed

- Upload conan packages into artifactory (BSDK-174)

2.31 [1.15.5] - 2025-06-17

2.31.1 Added

- Upload kotlin binding into maven releases repository (BSDK-174)
- Upload conan packages into artifactory (BSDK-174)

2.31.2 Fixed

- Python binding performance regression

2.32 [1.15.4] - 2025-06-13

2.32.1 autotoc_md88

- `sfePalmQualityAttributes` - Implementation of palm quality attributes (BSDK-157)

2.33 [1.15.3] - 2025-06-06

2.33.1 Added

- `sfePalmQualityAttributes` - API for Palm quality attributes (BSDK-168)
- Optional JVM bindings with Kotlin examples (BSDK-149)

2.33.2 Fixed

- Missing generated Python bindings

2.34 [1.15.2] - 2025-06-04

2.34.1 Fixed

- Solvers are now loaded on Android using `loadLibrary` (BSDK-152)

2.35 [1.15.1] - 2025-06-03

2.35.1 Added

- Licensing based on Android ID has been added for devices running Android 10 and older (i.e., below Android 11) (BSDK-152)

2.35.2 Fixed

- Removed unnecessary image resize before sfePalmDetect in example_palm_$*$ and benchmark_palm applications

2.36 [1.15.0] - 2025-05-15

2.36.1 Added

- New palm modality detection, extraction and liveness models:
  - `palm_detect_accurate` - High-accuracy palm detection model
  - `palm_detect_fast` - Optimized for real-time palm detection
  - `palm_template_extract_accurate` - High-accuracy palm template extraction
  - `palm_liveness_accurate` - Enhanced palm liveness verification

2.36.2 Changed

- Palm template format improvements:
  - Reduced template size to 522B for improved efficiency
  - Enhanced palm detection with 8 keypoints (previously 4)
  - Added `SFEHandOrientation` and `SFEHandSide` enums for better hand position tracking
- Updated `sfePalmTemplateExtract` API to require solver parameter for improved flexibility

2.36.3 Fixed

- Corrected const qualifier in tracking API parameters for better type safety
- Fixed parameter passing consistency in `sfePalmLivenessPassive` function [BSDK-133]

2.36.4 Breaking Changes

- Removed backward compatibility with palm templates from versions 1.13.0 through 1.14.6
  - This change is required due to the implementation of a new palm template algorithm
  - Users must re-extract palm templates with version 1.15.0 or later

2.37 [1.14.6] - 2025-04-25

2.37.1 Added

- rknn2 face_detect_accurate_mask solvers for w480h640, w720h960 and w960h1280 images (rknn2 0.10.3)

2.37.2 Changed

- Updated face passive liveness threshold in example_face_liveness according to documentation
- Updated Rockchip RKNN2 solver version to 0.10.3

2.38 [1.14.5] - 2025-04-11

2.38.1 Fixed

- A bug in reported version string

2.39 [1.14.4] - 2025-04-10

2.39.1 Added

- Added support for Palm liveness with examples and benchmark

2.39.2 Changed

- Reported version string in FFI/UniFFI APIs will now match the package version

2.40 [1.14.3] - 2025-03-28

2.40.1 Fixed

- Fixed C API definition of SFEFaceTracked, SFEIrisTracked, SFEObjectTracked, SFEPalmTracked
  - Removed unused ¨entity¨ field, use ¨uuid¨ field instead
  - Added ¨state¨ field and enum to indicate the tracking state
- Updated tracking to be more readable with terminal output

2.41 [1.14.2] - 2025-03-13

2.41.1 Changed

- TFLite library and solvers skipped in Kotlin packages build for Android
- TFLite solver updated to v0.10.9 [BFO-21]
  - Added face detectors with portrait input sizes 720x1280 and 1080x1920

2.42 [1.14.1] - 2025-03-05

2.42.1 Added

- Added Tracking API in C API and bindings [SE-333]
- Added Android ID based licensing [SE-255]

2.42.2 Changed

- Entity, SFETemplateIdentificationCandidate, SFEEntityIdentificationCandidate are now a Core types. Modality specific types are deprecated and will be removed in next major release. [SE-333]
- onnxruntime_solver updated to v0.10.6 [SE-255]
- tflite_solver updated to v0.10.7 [SE-255]

2.42.3 Fixed

- License with MAC address range HWID (IM0∗-IM0∗) is now checked against multiple MAC addresses if available.

2.43 [1.13.5] - 2025-02-18

2.43.1 Changed

- Updated Kotlin UniFFI bindings for better Java compatibility
  - API types change: ULong => Long, UInt => Int, UByte => Byte
- Cleaned up Kotlin and Java imports
- Android in ONNXRT configuration uses Float16 solvers to reduce package size [SE-510]
- Android packages are built with opt-level = ¨s¨ [SE-510]

2.44 [1.13.4] - 2025-01-21

2.44.1 Added

- Add support for Android armeabi-v7a
- Add suppurt for Android arm64-v8a GPU

2.44.2 Changed

- Updated onnxruntime solver to v0.10.4 (fixed qr/palm keypoints calculation [SE-507])
- Update Liveness thresholds documentation

2.45 [1.13.3] - 2024-12-04

2.45.1 Added

- Add suport for palm modality in Kotlin for Android [SE-447]

2.45.2 Fixed

- Ambarella face detector post processing had wrong relative coordinates calculation
- Fixed licensing impact on identification performance

2.46 [1.13.2] - 2024-11-28

2.46.1 Changed

- TFLite solver updated to version 0.10.6 [SE-470]
  - New passive liveness NEARBY FAST model

2.47 [1.13.1] - 2024-11-19

2.47.1 Added

- Added support for dynamic input shape palm detection [SE-479]
  - new palm_qrcode_detect.onnxrt.solver with dynamic input shape support
  - old solver renamed to palm_qrcode_detect_w640h480.onnxrt.solver

2.47.2 Changed

- TFLite solver updated to version 0.10.5 [SE-431]
  - Improved face_detector_accurate_mask∗ model performance
  - Default delegate is now correctly set to `vx` for NXP build, `gpu` for Android build

2.47.3 Fixed

- Jpeg encoding has wrong color order, BGR instead of RGB

2.47.4 Removed

2.48 [1.13.0] - 2024-11-07

2.48.1 Added

- Introduced new Palm modality:
  - sfePalmDetect
  - sfePalmTemplateExtract
  - sfePalmTemplateMatch
  - sfePalmTemplateVersion
  - sfePalmTemplateIdentify
  - sfePalmEntityIdentify
- Added assets, example and benchmark for palm modality

2.48.2 Changed

2.48.3 Fixed

2.48.4 Removed

2.49 [1.12.1] - 2024-11-06

2.49.1 Added

- Added C++ examples for face entity identification, liveness and iris identification

2.49.2 Changed

- Separated examples from benchmarks

2.49.3 Fixed

- sfeHwidGet now returns required buffer length when called with null buffer pointer.

2.49.4 Removed

2.50 [1.12.0] - 2024-10-25

2.50.1 Added

- Added new passive liveness NEARBY FAST model with significantly improved accuracy
  - ONNX Runtime solver
  - Updated documentation for new liveness solvers

2.50.2 Changed

- TFLite solver updated to 0.10.4
  - New passive liveness DISTANT FAST model with significantly improved accuracy.
  - TFLite VX delegate caching is enabled by default, and there is preset cache file path specific for each model

2.50.3 Fixed

- Iris matching score after normalization was wrongly mapped to [0.0, 1.0] range
- Fixed QualityAttributes to return zero values when an error occurs instead of returning a Result type

2.50.4 Removed

- Removed old passive liveness DISTANT and NEARBY model

2.51 [1.11.2] - 2024-10-04

2.51.1 Added

2.51.2 Changed

- TFLite solver update to 0.10.2 with default delegate `vx` for NXP build and `gpu` for Android build

2.51.3 Fixed

- Fixed sfeFaceSize mismatched output parameter type from size_t to float

2.51.4 Removed

2.52 [1.11.1] - 2024-08-05

2.52.1 Added

2.52.2 Changed

- Ambarella solver updated to v0.10.1 with a static EazyAI based implementation
- Ambarella vproc.bin location is now first searched for in current working directoryd Solver API version is now v0.10.x

2.52.3 Fixed

2.52.4 Removed

2.53 [1.11.0] - 2024-07-26

2.53.1 Added

- Integrated new passive liveness DISTANT FAST model with significantly improved accuracy.
  - Ambarella solver SDK2.5.8 cv22 and SDK3.0.6 - cv2, cv22, cv28

2.53.2 Changed

- Supported Solver API version is now v0.10.x
- Update of solver version because of major fix in generic_solver [SE-399]
- Improved documentation design

2.53.3 Fixed

2.53.4 Removed

2.54 [1.10.3] - 2024-06-12

2.54.1 Added

- Added reference to bindings API on documentation

2.54.2 Changed

- Fixed missing tensorflow_lite.so in Android AAR package

2.54.3 Fixed

2.54.4 Removed

2.55 [1.10.2] - 2024-05-28

2.55.1 Added

- Added TFLite GPU solvers to Android Kotlin package
- Added Python connector for Linux and Windows

2.55.2 Changed

2.55.3 Fixed

2.55.4 Removed

2.56 [1.10.1] - 2024-05-24

2.56.1 Added

- Added support for new passive liveness DISTANT FAST on Rockchip RKNNv1 (rockchip solver v0.9.1)

2.56.2 Changed

2.56.3 Fixed

2.56.4 Removed

2.57 [1.10.0] - 2024-05-20

2.57.1 Added

- Integrated new passive liveness DISTANT FAST model with significantly improved accuracy.
  - ONNX Runtime solver only
- Integrated new face detector ACCURATE model
  - ONNX Runtime solver only
- Add support for NN model acceleration on Android GPU via TensorFlow Lite GPU delegate

2.57.2 Changed

- Supported Solver API version is now v0.9.x
- Deprecating passive liveness DISTANT model. It will be removed in next major release.

2.57.3 Fixed

2.57.4 Removed

2.58 [1.9.1] - 2024-03-05

2.58.1 Added

- sfePersonTemplateType()

2.58.2 Changed

2.58.3 Fixed

2.58.4 Removed

2.59 [1.9.0] - 2024-02-29

2.59.1 Added

- Person template extract modality
  - Person template extraction
  - Person template matching
  - Person matching
- Uniffi API for Swift (iOS and macOS) and Kotlin (Android)
- Face solvers for Rockchip with RKNNv2

2.59.2 Changed

2.59.3 Fixed

2.59.4 Removed

2.60 [1.8.3] - 2024-02-23

2.60.1 Added

2.60.2 Changed

2.60.3 Fixed

- More robust solver dll loading on Windows

2.60.4 Removed

2.61 [1.8.0] - 2024-01-31

2.61.1 Added

- Added a new modality - Iris:
  - sfeIrisDetect
  - sfeIrisTemplateExtract
  - sfeIrisTemplateMatch
  - sfeIrisTemplateQuality
  - sfeIrisTemplateVersion
  - sfeIrisTemplateIdentify
  - sfeIrisEntityIdentify
- Added support for Ambarella cv28 architecture

2.61.2 Changed

- Deprecating SFEBBox in favor of SFEBoundingBox, field names will be changed in next major release
- Ambarella solvers unified for cv2, cv22, cv25 and cv28 - version 0.8.6
  - same accuracy on each architecture
- New licensing for solvers based on modalities: face, face_liveness, iris, person, object
  - old licenses won't work

2.61.3 Fixed

2.61.4 Removed

2.62 [1.7.2] - 2023-11-29

2.62.1 Added

2.62.2 Changed

2.62.3 Fixed

- Fixed padding in sfeLicenseSet

2.62.4 Removed

2.63 [1.7.1] - 2023-11-27

2.63.1 Added

2.63.2 Changed

2.63.3 Fixed

- Fix in identification API

2.63.4 Removed

2.64 [1.7.0] - 2023-11-17

2.64.1 Added

- Added licensing API to the sfe_core library:
  - sfeHwidGet
  - sfeLicenseValidate
  - sfeLicenseSet

2.64.2 Changed

2.64.3 Fixed

- Fixed error formatting on Windows - charset changed to Ascii, color disabled [SE-259]
- Fixed path to user specific license file location (∼/AppData/Local/Innovatrics) on Windows

2.64.4 Removed

2.65 [1.6.0] - 2023-11-07

2.65.1 Added

- sfeFaceEntityIdentify function - Identify entities with multiple templates.

2.65.2 Changed

- SFEIdentificationResult - Is now deprecated and will be removed in next major release. Use SFEFace↩ TemplateIdentificationCandidate instead.

2.65.3 Fixed

2.65.4 Removed

2.66 [1.5.3] - 2023-10-27

2.66.1 Added

2.66.2 Changed

2.66.3 Fixed

- Sharpness calculation was fixed
- Affine transformation for rectangular images was fixed

2.66.4 Removed

2.67 [1.5.2] - 2023-09-14

2.67.1 Added

2.67.2 Changed

- Solver for Ambarella SDK v2.5.8 updated to later version (v0.8.6) with improved performance.

2.67.3 Fixed

2.67.4 Removed

2.68 [1.5.1] - 2023-09-07

2.68.1 Added

- Ambarella SDK v2.5.8 is now supported:
  - Added new package containing special solvers that enable Ambarella SDK v2.5.8 support.

2.68.2 Changed

2.68.3 Fixed

2.68.4 Removed

2.69 [1.5.0] - 2023-08-25

2.69.1 Added

- Added new function sfeFaceCrop that returns face crop with its bounding box according given landmarks and face size extension.

2.69.2 Changed

2.69.3 Fixed

2.69.4 Removed

2.70 [1.4.2] - 2023-08-14

2.70.1 Added

2.70.2 Changed

- Integrated new passive liveness DISTANT model with significantly imporved accuracy
  - ONNX Runtime solver updated to v0.8.3
    - face_liveness_passive_distant.onnxrt.solver
  - Rockchip solver updated to v0.8.0
    - face_liveness_passive_distant_quant.rockchip.solver
  - Ambarella solver updated to v0.8.3
    - updated face_liveness_passive_distant_acc.ambarella.solver
    - New face_liveness_passive_distant_perf_dra3.ambarella.solver with better accuracy and comparable performance replaces face_liveness_passive_distant_perf.ambarella.solver

2.70.3 Fixed

2.70.4 Removed

2.71 [1.4.1] - 2023-07-10

2.71.1 Added

2.71.2 Changed

- Ambarella solvers updated to v0.8.1
  - New BALANCED template extraction model with improved accuracy - face_template_extract_↩ balanced_perf_dra3.ambarella.solver
- Android JNI wrapper now uses *android::util::Base64* class instead of *java::util::Base64* class to encode license data.

2.71.3 Fixed

- Fixed detection solvers in Android packages
- Fixed alignment issue on 32bit systems in sfe_face library function sfeFaceHeadPose
- Fixed SFESolverParameter ownership of name and value

2.71.4 Removed

2.72 [1.4.0] - 2023-06-23

2.72.1 Added

- Added new function sfeSolverCreateWithParameters which accepts collection of solver specific parameters (SFESolverParameter) on input. Supported solver parameters are documented in Solver documentation.
- Added support for TensorflowLite solver enabling NN acceleration on NXP iMX8M Plus chip with VX delegate support

2.72.2 Changed

- Ambarella solvers updated to v0.8.0
  - improved accuracy of BALANCED template extraction - face_template_extract_balanced_perf.↩ ambarella.solver
- function sfeSolverCreate is deprecated and will be replaced by sfeSolverCreateWithParameters in next major release version
- ONNX Runtime execution provider can be configured using solver parameter ¨runtime_provider¨, so on Linux x86, Windows x86 and Jetson platforms the same ONNX Runtime solver binary can be used for CPU, CUDA and TensorRT.

2.72.3 Fixed

2.72.4 Removed

- Removed onnxrt_cuda and onnxrt_tensorrt solvers.

2.73 [1.3.4] - 2023-06-06

2.73.1 Added

2.73.2 Changed

- Rockchip solver to v0.7.5 with faster input preprocessing (no channel swap, no CHW/HWC swap)

2.73.3 Fixed

2.73.4 Removed

2.74 [1.3.3] - 2023-06-02

2.74.1 Added

- Added support for passive liveness on Rockchip
- New documentation in HTML and PDF added to the package
- License Manager app added to the package

2.74.2 Changed

- Faster image resize
- Performance improvements (fast image resize, parallel warp) in all solvers

2.74.3 Fixed

2.74.4 Removed

2.75 [1.3.2] - 2023-05-18

2.75.1 Added

- New function sfeVersion() added into the sfe_core library

2.75.2 Fixed

- Fixed swapped roll and pitch calculation

2.76 [1.3.1] - 2023-05-09

2.76.1 Added

- Added support for passive liveness on Ambarella
- Added Android Java connector with sample application

2.76.2 Fixed

- Fixed missing liveness solvers in Android package

2.77 [1.3.0] - 2023-04-23

2.77.1 Added

- DISTANT and NEARBY passive liveness integrated into sfe_face library
- Face areas computation support (relative area, relative area in image)
- Added face quality attributes (sharpness, brightness, contrast, unique_intensity_levels) to sfe_face library
- Added support for calculating headpose angles

2.78 [1.2.4] - 2023-04-18

2.78.1 Added

- Added support for Ambarella CV22 and CV25 (armv8 CPU) with CVFlow support. Ambarella SDK 3.0.6 and newer is required.

2.79 [1.2.3] - 2023-04-14

2.79.1 Added

- Calculate detection input dimension based on face size
- Added support for ILICENSE and ILICENSE_DATA environment variables

2.80 [1.2.2] - 2023-04-03

2.80.1 Fixed

- Debugging symbols are now stripped

2.81 [1.2.1] - 2023-03-21

2.81.1 Added

- Face size api request to sfe_face lib

2.82 [1.2.0] - 2023-03-09

2.82.1 Added

- Introduced support for Android (armv7, armv8)
- Added pose estimation to sfe_person library

2.82.2 Changed

- The library sfe_person_attribute renamed to sfe_person
- sfe_face_detect, sfe_face_landmarks and sfe_face_template libraries merged into single sfe_face library
- The library sfe_object_detect renamed to sfe_object

2.83 [1.1.0] - 2023-02-20

2.83.1 Added

- Added license check to identification
- Added object detection for 80 object types including person, animals and vehicles
- Added 3 ONNX Runtime solvers for object detection
  - object_detect_fast
  - object_detect_balanced
  - object_detect_accurate

2.83.2 Changed

- SFEBBox parameter added into sfeGetPersonAttributes function

2.84 [1.0.3] - 2023-02-13

2.84.1 Added

- Support for older NVIDIA Jetson Jetpacks:
  - Jetpack 4.4: ONNX Runtime version 1.7.2
  - Jetpack 4.6: ONNX Runtime version 1.12.2

2.84.2 Fixed

- Undefined symbols in ONNX Runtime solvers related to licensing were fixed

2.85 [1.0.2] - 2023-01-30

2.85.1 Added

- Added Windows x86_64, Linux x86_64 and Jetson ARM64 support
  - ONNX Runtime solvers (CPU, CUDA and TensorRT)
- Added environment variables for threading optimization
  - ONNXRUNTIME_SOLVER_INTER_THREADS
  - ONNXRUNTIME_SOLVER_INTRA_THREADS
- Added person attributes detection
  - supported in ONNX Runtime solvers only

2.85.2 Changed

- SFEBBox structure moved to libsfe_core

2.86 [1.0.1] - 2023-01-23

2.86.1 Added

- Added face template identification
- Added SFEFaceLandmarkType enumeration

2.87 [1.0.0] - 2023-01-09

- First version for Rockchip RV1109 armv7 CPU with RKNN support

Chapter 3

Features

The face recognition process includes the detection of the face, detection of facial landmarks, face template extraction and 1:1 matching (verification) or 1:N matching (identification).

3.1 Face Detection

Face detection is a process of finding multiple faces in the input image.

Note: SFE Toolkit 4 uses a unified detection API (`sfeDetect()`) that works for all detection modalities (face, iris, palm, object, tattoo, person). The function automatically determines the detection type based on the solver provided. This unified API replaces the previous modality-specific detection functions.

SmartFace Embedded can detect faces of various sizes, orientations, facial hair types or ethnicities. The face can be partly occluded by glasses, sunglasses or a hat. IFace SDK can operate with many different lighting conditions and image qualities.

Face size is defined as the maximum value of the distance between inter eyes centers and distance between the center of the mouth and the center point between the eyes (nose root):

```
face_size = max(distance(left_eye_center, right_eye_center), distance(mouth↩
_center, eyes_center))
```

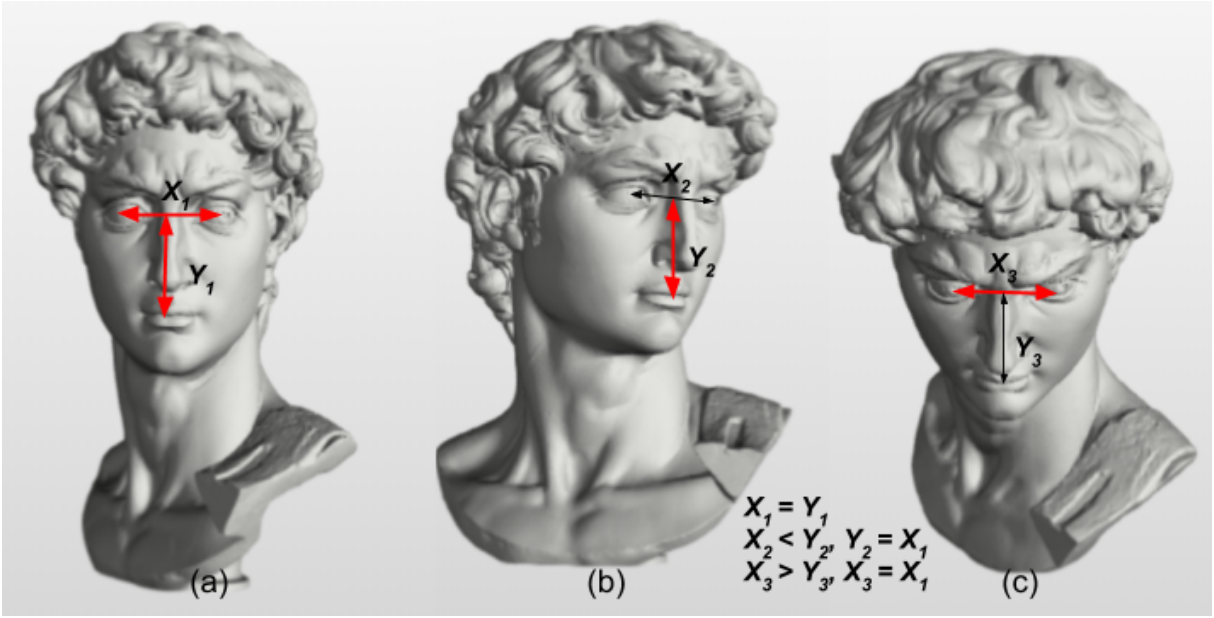

Figure 3.1 Face size: visualizations of various distances of inter eyes centers distances (X=distance(left_eye_center,right_eye_center)) and distances between the center of mouth and center point between eyes (Y=distance(mouth_center, eyes_center)). Face size is the maximum of these two distances. It can be seen from images (a), (b) and (c) that the face size (red arrow) defined in this way is invariant to the pose of the head (yaw, pitch, roll).

The face size has to be specified in absolute (pixel distance). In **SFE Toolkit** the minimum and maximum face size comes as input for [sfeFaceDetectInputSize] function. This function returns the recommended dimension of the image to be downscaled before face detection.

The face area is closely related to the face size. It is an area around a face defined by a bounding rectangle with width = 4 x face_size, height = width / 0.75 (according to ISO/IEC 19794-5 standard, section 9.2). The Point which is the geometric center between eyes is positioned in the face area in position X_Pos, Y_Pos, where Y_Pos = 0.6f $*$ width, and X_Pos relies on the head yaw rotation. The main reason for this is to have the whole head in the face area no matter what the head rotation is.

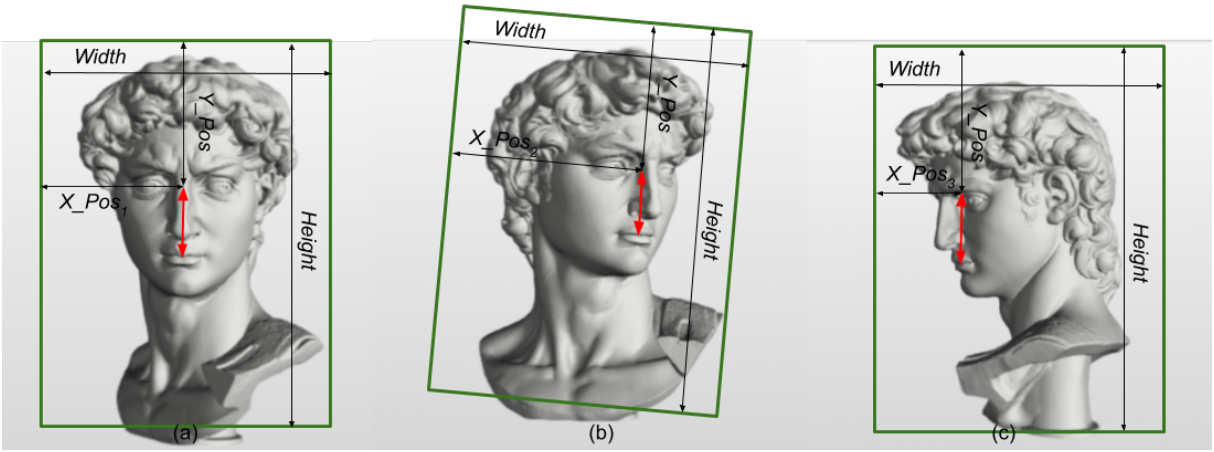

Figure 3.2 Face area: visualizations of face areas (green boxes) for the face in various positions. The face size is defined by a distance shown as the red arrow. There are different positions of the face area according to the face yaw rotation shown in (a),(b) and (c). Positions of the eyes' centers in the face areas are the same in the Y direction (Y_Pos) but change in the X direction (X_Pos1, X_Pos2, X_Pos3). Width and Height are the same in all three cases.

Face area size relative to image area size can specify the sizes of faces that should be detected.

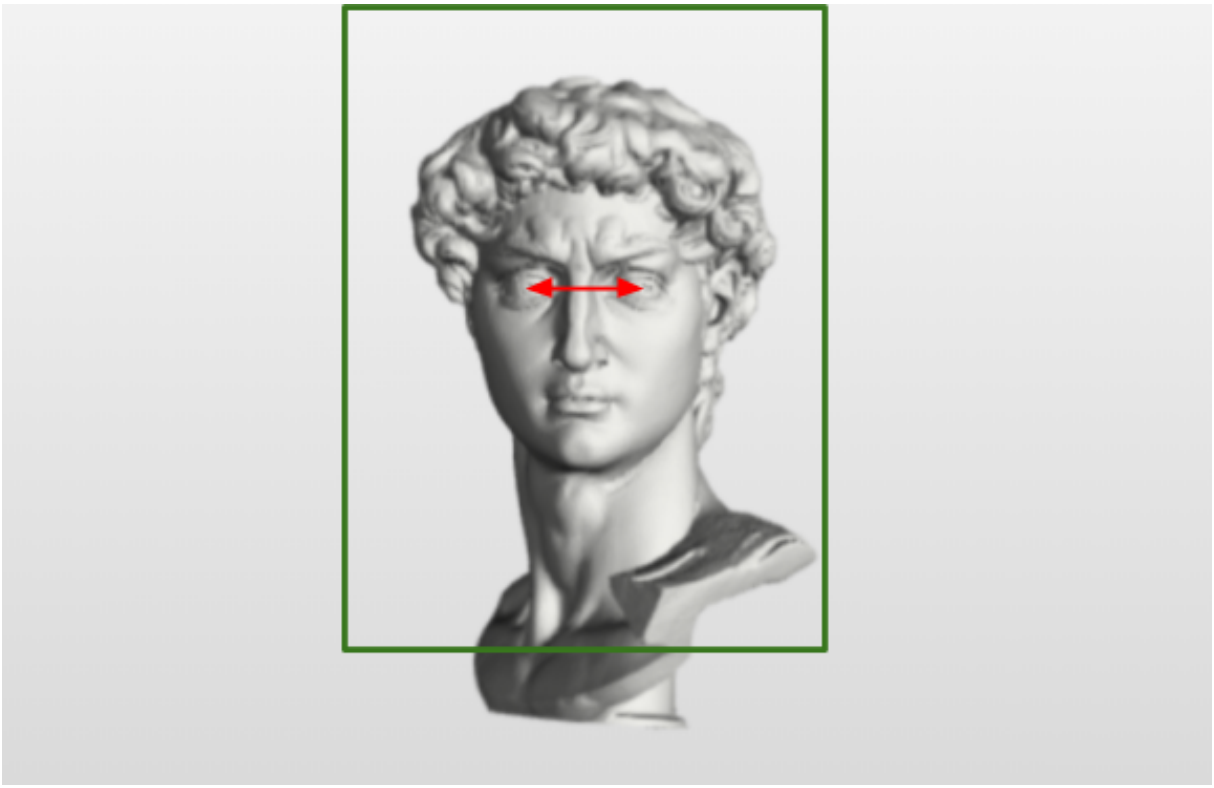

Figure 3.3 If image above has resolution of 730x470 pixels (image_area_size = 730 ∗ 470 = 343100) and face_size is 70 pixels (face_area_width = 70 ∗ 4 = 280, face_area_height = face_area_width / 0.75 = 373, face_area_size = 104440), then relative image size is relative_image_size = 104440 / 343100 = 0.304 (30.4%). The face area is marked with a green box and the face size with a red arrow.

Sometimes, when faces are close to image boundaries, their face areas can get out of the image boundaries. Some applications may want to find these faces (and filter them out). Due to this reason, the SFE Toolkit provides functionality to return the face area visible in the image.

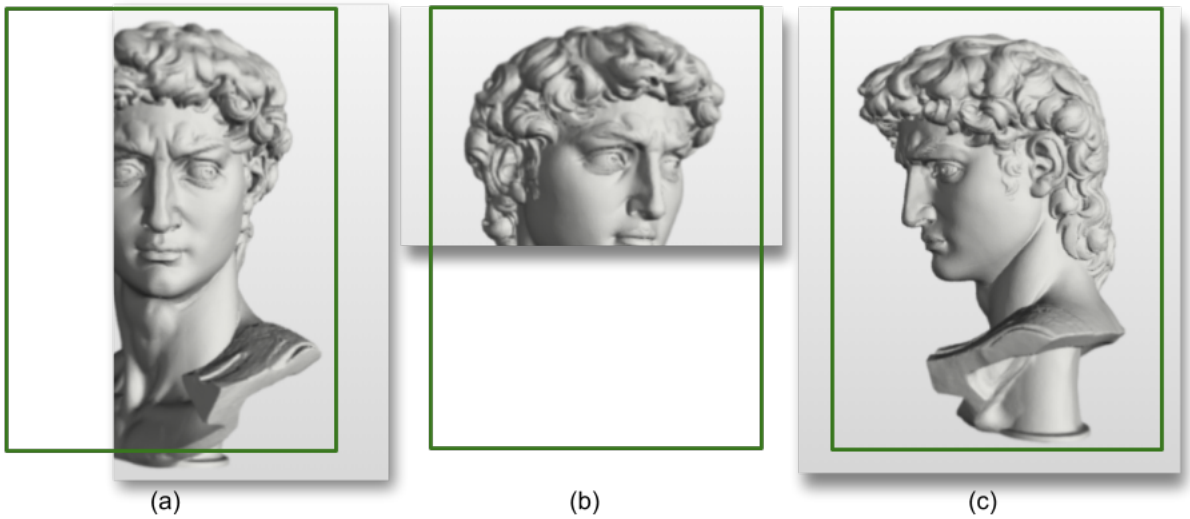

Figure 3.4 Example of face area visible in the image. Relative values: (a) 0.30, (b) 0.5, (c) 1.

In **SFE Toolkit** C API use function [sfeFaceArea] which returns actual face size, face area and face area relative to the image. Find more information in the Face Area section

SmartFace Embedded can detect faces of various rotations.

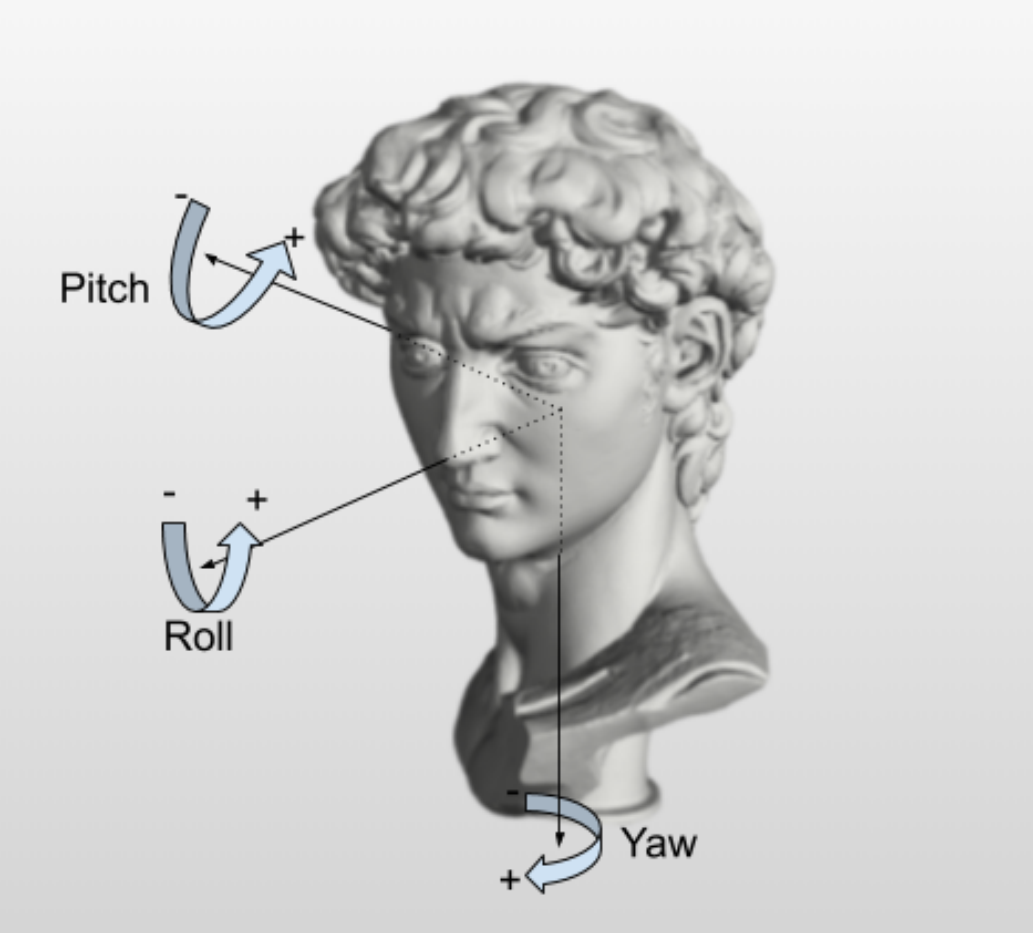

Figure 3.5 Possible face rotations (defined as in DIN9300).

Find more information in the Headpose Angles section

SmartFace Embedded has different face detection modes. Using different modes can adjust the trade-off between speed and accuracy of face detection. Face detection accuracy has two meanings:

- a ratio between false accepted faces and false rejected faces
- the precision of facial feature point detection.

Face detection returns an array of bounding boxes of detected faces with detection confidence. Face detection confidence is a value in the range <0.0, 1.0> and it refers to the confidence score of the face related to face detection. The higher the value of the attribute the better quality of the face. The decision thresholds are around 0.06, but it depends on the face image quality/camera angle etc. Detected faces are ordered using face detection confidence. Face detection confidence defines the order of faces. They are ordered in descending order, so the face with the highest confidence is the first (index 0) in the array.

In **SFE Toolkit** C API use function [sfeDetect] with a face detection solver. The unified `sfeDetect()` function automatically determines the detection modality based on the solver type.

Example

```
SFEDetection detected_face = {};
{ // STAGE 3: Detect face in the image
  size_t detection_count = 1;
  // Detect face in the image
  error = sfeDetect(detector_solver, resized_image, detection_threshold,
                    &detected_face, &detection_count);
```

```
    utils::checkError(error);

    if (detection_count == 0) {
      std::cout « "No face detected in the probe image." « std::endl;
      return 0;
    } else {
      std::cout « "Found " « detection_count
                « " face(s) in the probe image. Using the face with highest "
                   "confidence."
                « std::endl;
    }
  }
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_core.detect`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.core.detect`
- In Swift use `SFEToolkitCore.detect`
- In .NET use `Innovatrics.SFEToolkit.Core.Detect`

3.1.1 Recommended input image dimensions

Before face detection, the image size can be adjusted to fit the required face size detection range. This process improves accuracy. The input combination of required maximal and minimal face sizes is validated against the model constraints. If valid, the recommended width and height of the image are calculated. The original image can be resized by using a specific function from sfe_toolkit.

In **SFE Toolkit** C API use function [sfeFaceDetectInputSize]

Example

```
    // Solvers without width and height in name can accept dynamic input
    // image size. For best results we recommend to scale the input image to
    // a resolution recommended by the sfeFaceDetectInputSize function

    // Use sfeFaceDetectInputSize to get recommended input image dimensions
    // for given min_face_size/max_face_size
    SFEDetectionInputSize input_size{};
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &input_size);
    utils::checkError(error);
    recommended_width = input_size.width;
    recommended_height = input_size.height;
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.detect_input_size`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.detectInputSize`
- In Swift use `SFEToolkitFace.detectInputSize`
- In .NET use `Innovatrics.SFEToolkit.Face.DetectInputSize`

3.1.2 Dynamic model input shape

Face detection ONNX models used in the SFE Toolkit support dynamic input shape. It means you can choose any image resolution to pass into face detection. Please note the bigger the input image is the longer the face detection takes.

Based on the resolution of the input image and the required face size to be detected, we can recommend the size of the image for face detection. See more in this section

3.1.3 Static model input shape

Some NN conversion tools and NN inference engines do not support dynamic model input shapes, for example, Rockchip, and Ambarella. In this case, we provide multiple face detection models with the input shape set according to the customer's need, for example, Full HD (1920x1080), HD (1280x720), 640x360, etc.

3.2 Face Landmarks Extraction

Facial landmarks extraction is a process of detecting 23 landmarks (feature points) in a detected face. These points include the position of the eyes, eyebrows, nose, mouth, chin and edges of the face. Each detected landmark consists of the landmark type, x and y coordinates relative to the input image and confidence. All detected landmarks and their IDs can be seen In the picture below.

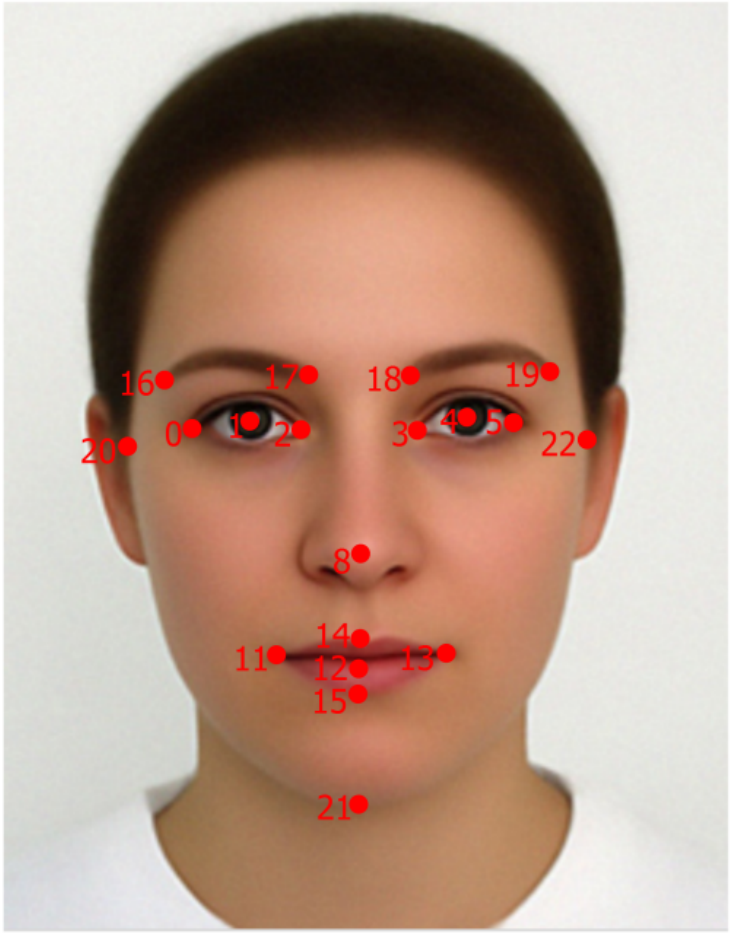

Figure 3.6 Facial landmarks and their IDs detected by SmartFace Embedded.

In **SFE Toolkit** C API use function [sfeFaceLandmarks] to extract the landmarks from the detected face.

Example

```
std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
{ // STAGE 4: Get face landmarks

  // Use landmarks detection solver to get relevant landmarks for
  // the detected face
  error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
                           landmarks.data());
  utils::checkError(error);
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.landmarks`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.landmarks`
- In Swift use `SFEToolkitFace.landmarks`
- In .NET use `Innovatrics.SFEToolkit.Face.Landmarks`

A list of all facial landmarks is defined in enum [SFEFaceLandmarkType]. The position of the output facial landmark is always relative to the input image origin. The image origin is pixel[0,0] in the upper left corner. The position of the feature point is returned as a pair of x and y coordinates with float precision. [SFEFaceLandmarks] includes also the confidence of detected landmarks.

Landmarks are used for face mask status detection, normalization of the face before the face template extraction, face liveness detection and other operations.

3.2.1 Face Crop

SmartFace Embedded provides functionality to crop the detected face from the input image for example for template re-extraction and migration purposes. Face crop includes the crop image of the detected face, its bounding box and used face size extension.

Face size extension defines the size of the crop as an extension of the detection bounding box. The crop will be centered on the detection bounding box and the face size will be multiplied by this value. The API accepts a float; typical values are 0, 1, 2, 3, 4 and 5. You should use the correct value depending on the post-processing required on the server side, for example:

- Template extraction requires face_size_extension=2
- Passive liveness detection requires face_size_extension=5
- Passive liveness distant fast model does not require face_size_extension

You can also limit the size of the crop by setting the maximum face size. Maximum face size defines the maximal size of the face in the crop area. Once a face is detected and its crop area is calculated using face_size_extension, this value will be used to determine the scale of the crop. If the actual face size is larger than the input max_face_size, the cropped image will be downscaled accordingly.

In **SFE Toolkit** C API use function [sfeFaceCrop]

Example

```
SFEFaceCrop crop_data = {};
DEFER(sfeImageFree(crop_data.crop_image));
{ // STAGE 5.2: Get face crop
  //  Defines the size of the crop as an extension of
  // the detection bounding box.
  float face_size_extension = 2.0f;
  SFEError error =
      sfeFaceCrop(image, landmarks[best_face_index].data(),
                  face_size_extension, (float)(face_size_max), &crop_data);
  utils::checkError(error);

  std::cout « "Face crop extension: " « crop_data.face_size_extension
            « ", width of cropped image: " « crop_data.crop_image.width
            « "px, height of cropped image: "
            « crop_data.crop_image.height « "px." « std::endl;
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.crop`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.crop`
- In Swift use `SFEToolkitFace.crop`
- In .NET use `Innovatrics.SFEToolkit.Face.Crop`

3.2.2 Face Mask Detection

Face mask detection determines the presence of the face mask on the given face. The presence of the mask is specified using mask confidence which is a value in the range $<0.0, 1.0>$. The recommended decision threshold is around 0.5.

In **SFE Toolkit** C API use function [sfeFaceMaskConfidence]

Example

```
error = sfeFaceMaskConfidence(landmarks.data(), &mask_confidence);
utils::checkError(error);
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.mask_confidence`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.maskConfidence`
- In Swift use `SFEToolkitFace.maskConfidence`
- In .NET use `Innovatrics.SFEToolkit.Face.GetFaceMaskConfidence`

3.3 Face Template Extraction

SmartFace Embedded can be also used for 1:1 (verification) and 1:N (identification) face recognition.

To perform 1:1 or 1:N matching it is necessary to extract a face template. A face template is a feature vector describing the face. The size of the face template is only 522 B so the memory requirements for identification are low and the matching is very fast. SmartFace Embedded supports multiple template extraction modes. Using different modes can adjust the trade-off between face template creation speed and face template quality. Face templates of higher quality give better results when used for 1:1 or 1:N face matching. Templates created with different template extraction modes are generally not compatible; exceptions are the V2 group (fast, balanced, accurate, accurate server), which can be matched with each other (see the compatibility matrix below). Retrieved template versions can be compared.

- *fast* - Face templates suitable for verification of fairly good accuracy are created when the *fast* mode is used. The performance of face template creation is very fast. Suitable for mobile/embedded devices.
- *balanced* - Face templates suitable for verification/identification of high accuracy are created when the *balanced* mode is used. The performance of the face template creation is somewhere in between *accurate* and *fast* modes. The *balanced* model is also capable of template extraction from faces with the face mask present.
- *accurate* - Face templates suitable for verification/identification of very high accuracy are created when the *accurate* mode is used. However, the performance of the face template creation is not as good as when *balanced* or *fast* mode is used.

You can decide which algorithm to use by choosing the appropriate face template extraction solver.

- face_extraction_fast
- face_extraction_balanced
- face_extraction_accurate

Please see the supported features table to understand which model is supported on your HW.

In **SFE Toolkit** C API use function [sfeFaceTemplateExtract] To use the correct template extraction mode, load the appropriate solver:

- face_extraction_fast
- face_extraction_balanced
- face_extraction_accurate

Example

```
{ // STAGE 1.4: Extract probe face template
  // Use template extraction solver to create new face
  // template
  error = sfeFaceTemplateExtract(template_solver, image, &detected_face,
                                 landmarks.data(), &probe_face_template);
  utils::checkError(error);
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.extract`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.extract`
- In Swift use `SFEToolkitFace.extract`
- In .NET use `Innovatrics.SFEToolkit.Face.TemplateExtract`

The face template includes the template version and can be obtained using C API.

- **Template version** is used during matching. Within the V2 group (fast, balanced, accurate, accurate server), templates from different extraction models can be matched; for other versions, the same minor version is required. When using multiple SmartFace Embedded Toolkits on multiple platforms or in conjunction with the SmartFace platform, see the compatibility matrix below for which template versions can be matched.

3.3.1 Face verification template compatibility

The face template has its system of version numbers. Major number change defines radical changes in the internal structure of the template document. Minor number change defines a change in extraction algorithm or minor data change of template. Except within the V2 group (see table below), templates of different minor versions are not compatible for matching.

Face template version compatibility (matching)

Group	Minor versions	Cross-version matching
Legacy	33, 36, 37, 38, 39, 40, 41	No (exact version only)
V2 (SFE)	52, 53, 54, 55	Yes (all four together)
Server/NIST	44, 45, 50, 51	No (exact version only)

Same minor version always matches. For the V2 group (Fast, Balanced, Accurate, AccurateServer), any pair among these four is compatible; when the two templates have different but compatible minor versions, the match score is the average of the probe and reference normalized scores (inno-gallery behaviour). Legacy and Server/NIST versions match only with the exact same minor version.

In **SFE Toolkit** C API use function [sfeFaceTemplateVersion]

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.template_version`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.templateVersion`
- In Swift use `SFEToolkitFace.templateVersion`
- In .NET use `Innovatrics.SFEToolkit.Face.Template.GetVersion`

3.3.2 Face Template Import/Export

Face templates can be exported to bytes for sharing between Innovatrics components or for storage. Import accepts multiple formats (auto-detected) and returns an `SFEFaceTemplate`.

Export format: iface-template protobuf only (single face embedding, modality Face, model version and quality in metadata). A buffer of 1024 bytes is sufficient for export.

Import formats (auto-detected):

- **Raw ICF:** exactly 522 bytes, starts with `ICF\0` — same as internal template layout.
- **iface-template protobuf:** exactly one embedding; modality must be Face; nonzero model version (high byte = major, low byte = minor); embedding 512 elements as quantized (char) or float (float is normalized then quantized the same as iface).

Unsupported format or invalid data returns an error.

Supported template versions (for import and matching):

- **Legacy (IFace SDK):** 33, 36, 37, 38, 39, 40, 41 (LegacyAccurate, LegacyFast, LegacyVisa, LegacyMask, LegacyBalanced, LegacyAccurateMask, LegacyWild).
- **V2 extraction (SFE):** 52 (Fast), 53 (Balanced), 54 (Accurate), 55 (AccurateServer).
- **Server/NIST (iface/inno-gallery):** 44 (Visa), 45 (Wild), 50 (Nist), 51 (NistP1).

Matching rules between versions are described in the compatibility matrix above.

In **SFE Toolkit** C API use functions [sfeFaceTemplateExport] and [sfeFaceTemplateImport]

Example

```
{ // STAGE 1.6: (optional) Export and re-import template
  std::vector<uint8_t> buf(1024);
  size_t size = buf.size();
  error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
  if (error && size > buf.size()) {
    buf.resize(size);
    size = buf.size();
    error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
  }
```

```
    utils::checkError(error);

    SFEFaceTemplate imported_template = {};
    error = sfeFaceTemplateImport(buf.data(), size, &imported_template);
    utils::checkError(error);

    float roundtrip_score = 0.f;
    error = sfeFaceTemplateMatch(&probe_face_template, &imported_template,
                                 &roundtrip_score);
    utils::checkError(error);
    std::cout << "Export/import round-trip match score: " << roundtrip_score
              << std::endl;
  }
```

Use the following functions in SFE Toolkit bindings:

- In Python use `sfe_face.template_export` and `sfe_face.template_import`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.templateExport` and `templateImport`
- In Swift use `SFEToolkitFace.templateExport` and `SFEToolkitFace.templateImport`
- In .NET use `Innovatrics.SFEToolkit.Face.TemplateExport` and `TemplateImport`

3.4 Face Template Verification

Verification is a comparison of two extracted face templates and it returns a matching score which refers to the probability that the two templates are extracted from the face of the same person. When the matching score is higher than a certain score threshold then the face images belong to the same person with high probability. The matching score range is <0.0, 1.0>. Its values can be interpreted as follows:

- Low values of the score, i.e. range <0, 0.6>, are normalized using FAR values and this formula: `score_L=-10*log(FAR)/100`. It means that score 0.3 is related to FAR=1:1000=10^-3, and score 0.5 is related to FAR=1:100000=10^-5 (evaluated on our large testing non-matching pairs dataset).
- High values of the score, i.e. range <0.8, 1.0>, are normalized using FRR values and this formula: `score_H=100/3*(FRR + 2)/100`. It means that a score of 0.8 is related to FRR=0.4, and a score of 0.9 is related to FRR=0.7 (evaluated on our large testing matching pairs dataset).
- Scores values in the range (0.6, 0.8) are weighted averages of *score_L* and *score_H*. This normalization helps the users to select the score threshold according to their needs. If it is too low e.g. threshold = 0.3, then the chance of falsely accepted non-matching faces is quite high (FAR=10^-3). When it is too high e.g. threshold = 0.9, then the chance of false rejected matching faces is quite high (FRR=0.7).

In **SFE Toolkit** C API use function [sfeFaceTemplateMatch]

Example

```
{ // STAGE 4: (optional) 1:1 matching with top candidate to showcase 1:1
  // matching
  utils::printFormatted("1:1 MATCHING WITH TOP CANDIDATE");

  if (identification_results.size() == 0) {
    std::cout << "No candidates found above identification threshold "
              << identification_threshold << std::endl;
    return 0;
  }

  auto best_candidate_template = gallery_face_templates[best_candidate_index];

  float match_confidence;
  error = sfeFaceTemplateMatch(&probe_face_template, &best_candidate_template,
                               &match_confidence);
  utils::checkError(error);

  std::cout << "Matching score of probe template with template index #"
            << best_candidate_index << ", score: " << match_confidence
            << std::endl;
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.template_match`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.templateMatch`
- In Swift use `SFEToolkitFace.templateMatch`
- In .NET use `Innovatrics.SFEToolkit.Face.TemplateMatch`

3.5 Face Template Identification

Identification is the process of finding the best matching candidates for the probe template in the template gallery (database). The maximum number of best candidates to be returned by the identification function can be configured. It depends on the use case. Identification returns only candidates whose matching score with the probe template is higher or equal to the matching score threshold. For the matching score threshold recommendation please refer to the section Face Template Verification

The identification is pure CPU-based and it can be configured how many CPU cores will be used for identification to optimize the performance within your application and also the CPU utilization.

In **SFE Toolkit** C API use function [sfeFaceTemplateIdentify]

Example

```
  size_t candidate_count = 1;
  int best_candidate_index = -1;
  std::vector<SFETemplateIdentificationCandidate> identification_results(
      image_paths.size());
  { // STAGE 3: Identification
    utils::printFormatted("1:N IDENTIFICATION");

    error = sfeFaceTemplateIdentify(
        &probe_face_template, gallery_face_templates.data(),
        gallery_face_templates.size(), identification_threshold,
        identification_results.data(), &candidate_count, 4);
    utils::checkError(error);

    // Resize the results vector to the actual number of candidates found, in
    // the case there is less than candidate_count
    identification_results.resize(candidate_count);

    std::cout « "Found " « identification_results.size()
             « " candidates above identification threshold "
             « identification_threshold « std::endl;
    for (auto &result : identification_results)
      std::cout « "Template index: #" « result.index
                « ", score: " « result.score « std::endl;

    // Since it's sorted vector by score, the best candidate is the first one
    best_candidate_index = identification_results[0].index;
  }
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.template_identify`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.templateIdentify`
- In Swift use `SFEToolkitFace.templateIdentify`
- In .NET use `Innovatrics.SFEToolkit.Face.templateIdentify`

SFE Toolkit also supports the identification of Entities. Entity ID is used to group multiple face templates of the same person. The result of identification is a list of Entity IDs ordered by best matching score of face templates associated with the same Entity ID.

In **SFE Toolkit** C API use function [sfeFaceEntityIdentify]

Example

```
size_t candidate_count = 1;
std::vector<SFEEntityIdentificationCandidate> results(candidate_count);
int best_candidate_index = -1;
{ // STAGE 3: Identification with entities.
  // Probe template is matched against every template in the gallery.
  // Function returns entity(UUID) which owns the best matching template.

  utils::printFormatted("1:N IDENTIFICATION WITH ENTITIES");
  SFEError error = sfeFaceEntityIdentify(
      &probe_face_template, gallery_face_templates.data(),
      enitity_pairs.data(), gallery_face_templates.size(),
      identification_threshold, results.data(), &candidate_count, 4);
  utils::checkError(error);

  if (candidate_count == 0) {
    std::cout « "No candidates found. Exiting.." « std::endl;
    return 0;
  }

  std::cout « std::endl;
  std::cout « "Found " « results.size() « " candidates " « std::endl;
  for (int i = 0; i < results.size(); i++) {
    std::cout « "Index: " « i « ", Score: " « results[i].score
              « std::endl;
    std::cout « "Entity UUID: ["
              « static_cast<int>(results[i].entity.uuid[0]) « ", "
              « static_cast<int>(results[i].entity.uuid[1]) « "..."
              « static_cast<int>(results[i].entity.uuid[15]) « "]"
              « std::endl;
  }
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.entity_identify`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.entityIdentify`
- In Swift use `SFEToolkitFace.entityIdentify`
- In .NET use `Innovatrics.SFEToolkit.Face.entityIdentify`

3.5.1 Face Identification Accuracy

Face identification accuracy varies depending on the template extraction solver used. The extraction solvers available per platform are listed in the Supported platforms and features table (typically fast, balanced, accurate).

Solver selection for identification

Choose the solver based on your security requirements and performance constraints:

- **Accurate**: Very low FAR, suitable for most security-sensitive applications
- **Balanced**: Low FAR, good for real-time identification with moderate security requirements
- **Fast**: Higher FAR, suitable for mobile/embedded devices where performance is prioritized over security

When matching against legacy templates (e.g. LegacyAccurateMask), use the same compatibility rules as in the Face template version compatibility table.

3.6 Face Liveness Detection

Face recognition systems are vulnerable to spoof attacks made by non-real faces. It is an easy way to spoof face recognition systems by facial pictures such as portrait photographs, masks and videos which are easily available from social media. A secure system needs a liveness evaluation strategy to guard against such spoofing.

Face liveness detection is a process of recognizing spoof attacks and non-real faces. It can recognize a real face against a photograph, masks and videos. The passive liveness score is defined over the interval $<0,1>$.

SmartFace Embedded currently supports two modes of passive liveness detection:

- DISTANT FAST - Passive liveness check for access control use-case (walkthrough), where the distance between the face and the camera is bigger and the size of the face (eye distance) is smaller.
- NEARBY FAST - Passive liveness check for digital onboarding use-case (selfie), where the distance between the face and the camera is smaller and the size of the face (eye distance) is bigger.

The specific threshold to evaluate the retrieved liveness score should be set according to your security needs. You should evaluate certain face attributes to achieve reliable accuracy.

3.6.1 Recommended threshold for passive liveness modes

Distant Fast	Threshold	FAR [%]	FRR [%]
1:100 FAR (APCER)	0.83	1	1.95
EER	0.81	1.36	1.36
1:100 FRR (BPCER)	0.80	1.60	1

Nearby Fast	Threshold	FAR [%]	FRR [%]
1:100 FAR (APCER)	0.87	1	3.17
EER	0.83	1.73	1.73
1:100 FRR (BPCER)	0.80	2.67	1

3.6.2 Recommended face attributes

Face Attribute	Distant Fast	Nearby Fast
Face confidence	[0.1; 1.0]	[0.1; 1.0]
Face size	[30; 60]	[60; inf]
Face relative area in the image	[0.9; inf]	-
Yaw angle	[-30; 30]	[-30; 30]
Pitch angle	[-30; 30]	[-30; 30]

In **SFE Toolkit** C API use function [sfeFaceLivenessPassive] To use the correct liveness detection mode, load the appropriate solver:

- face_liveness_passive_distant_fast
- face_liveness_passive_nearby_fast

Example

```
SFEFaceLiveness liveness = {};
{ // STAGE 5: Liveness check
  utils::printFormatted("LIVENESS CHECK");
  error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
                                 &liveness);
  utils::checkError(error);
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.liveness_passive`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.livenessPassive`
- In Swift use `SFEToolkitFace.livenessPassive`
- In .NET use `Innovatrics.SFEToolkit.Face.LivenessPassive`

3.7 Face Attributes

SmartFace Embedded supports various functions to evaluate face attributes.

3.7.1 Headpose angles

Face head pose attributes contain angle rotations of the head, specifically `yaw, pitch` and `roll`.

- The `roll` is a face attribute representing the angular rotation of the head towards the camera reference frame around the Z-axis.
- The `yaw` is a face attribute representing the angular rotation of the head towards the camera reference frame around the Y-axis.
- The `pitch` is a face attribute representing the angular rotation of the head towards the camera reference frame around the X-axis.

In **SFE Toolkit** C API use function [sfeFaceHeadpose]

Example

```
error = sfeFaceHeadPose(image, landmarks.data(), &head_pose);
utils::checkError(error);
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.head_pose`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.headPose`
- In Swift use `SFEToolkitFace.headPose`
- In .NET use `Innovatrics.SFEToolkit.Face.GetFaceHeadPose`

3.7.2 Face Area

The face area is closely related to the face size. It is an area around a face defined by a bounding rectangle with `width = 4 x face_size`, `height = width / 0.75` (according to ISO/IEC 19794-5 standard, section 9.2). The Point which is the geometric center between eyes is positioned in the face area in position X_Pos, Y_Pos, where `Y_Pos = 0.6f * width`, and X_Pos relies on the head yaw rotation. The main reason for this is to have the whole head in the face area no matter what the head rotation is.

- `face relative area` is the whole face area (including area exceeding image borders) relative to the image area.
- `face relative area in the image` is the face area within the image borders relative to the image area.

In **SFE Toolkit** C API use function [sfeFaceArea]

Example

```
error = sfeFaceArea(image, landmarks.data(), &face_area);
utils::checkError(error);
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.area`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.area`
- In Swift use `SFEToolkitFace.area`
- In .NET use `Innovatrics.SFEToolkit.Face.GetFaceArea`

3.7.3 Image quality attributes

The image face quality attributes that can be calculated are `sharpness`, `brightness`, `contrast` and `unique intensity levels`. All attributes are normalized and their value can be from range <0, 1>.

- The `sharpness` is a face attribute for evaluating whether an area of the face image is not blurred. The decision threshold for sharpness is 0.5, values near 0 indicate 'very blurred', and values near 1 indicate 'very sharp'.
- The `brightness` is a face attribute for evaluating whether an area of the face is correctly exposed. Values near 0 indicate 'too dark', values near 1 indicate 'too light', and values around 0.5 indicate OK. The decision thresholds are around 0.25 and 0.75.
- The `contrast` is a face attribute for evaluating whether an area of the face is contrast enough. Values near 0 indicate 'very low contrast', values near 1 indicate 'very high contrast', and values around 0.5 indicate OK. The decision thresholds are around 0.25 and 0.75.
- The `unique intensity levels` represent a face attribute for evaluating whether an area of the face has an appropriate number of unique intensity levels. Values near 0 indicate 'very few unique intensity levels', and values near 1 indicate 'enough unique intensity levels'. The decision threshold is around 0.5.

In **SFE Toolkit** C API use function [sfeFaceQualityAttributes]

Example

```
error =
    sfeFaceQualityAttributes(image, landmarks.data(), &quality_attributes);
utils::checkError(error);
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.quality_attributes`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.qualityAttributes`
- In Swift use `SFEToolkitFace.qualityAttributes`
- In .NET use `Innovatrics.SFEToolkit.Face.QualityAttributes`

3.7.4 Face Demographic Attributes

SmartFace Embedded can estimate face demographic attributes such as **age** and **gender** from a detected face.

- **Age**: Estimated age in years.
- **Gender**: Normalized gender score in range < -1, 1 >. Values near -1 indicate 'male', values near 1 indicate 'female', and values around 0 indicate uncertainty.

In **SFE Toolkit** C API use function [sfeFaceDemographicAttributes].

Example

```
SFEFaceDemographicAttributes attributes{};
{ // STAGE 5: Demographic attributes
  utils::printFormatted("FACE DEMOGRAPHIC ATTRIBUTES");
  error = sfeFaceDemographicAttributes(demographic_solver, image,
                                       landmarks.data(), &attributes);
  utils::checkError(error);
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_face.demographic_attributes`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.demographic↩ Attributes`
- In Swift use `SFEToolkitFace.demographicAttributes`
- In .NET use `Innovatrics.SFEToolkit.Face.DemographicAttributes`

3.8 Iris detection

Iris detection is a process of detecting the iris and pupil in the input eye image.

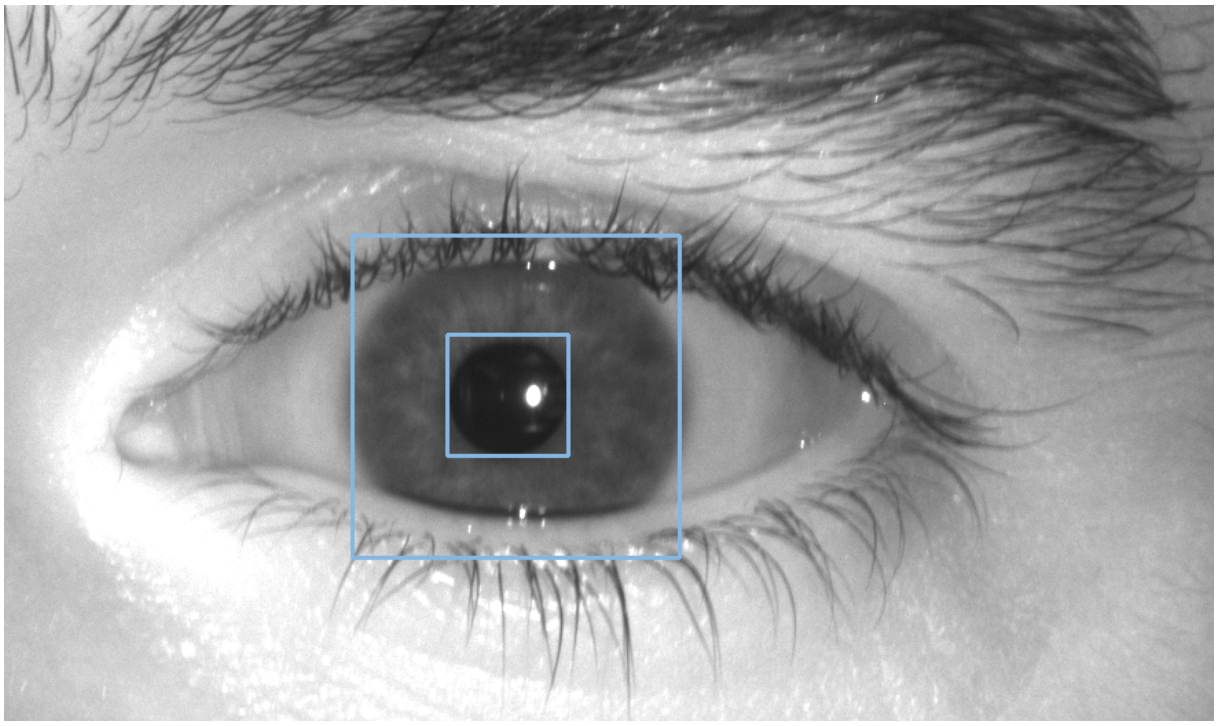

Figure 3.7 Iris detection results annotated in the input image.

Iris detection returns detected iris as a structure containing the iris bounding box, pupil bounding box, detection confidence and iris keypoints. Iris detection confidence is a value in the range <0.0, 1.0> and it refers to the confidence score of the iris related to iris detection. The iris detection confidence can be used to filter low-quality irises from further processing (template extraction and matching). The recommended threshold is 0.7.

In **SFE Toolkit** C API use function [sfeDetect] with an iris detection solver. The unified `sfeDetect()` function automatically determines the detection modality based on the solver type.

Example

```
SFEDetection detected_iris = {};
size_t iris_count = 1;
{ // STAGE 2: Detect iris in the image

  // Detect iris in the image
  error = sfeDetect(detector_solver, image, 0.1f, &detected_iris, &iris_count);
  utils::checkError(error);

  if (iris_count == 0 || detected_iris.confidence < 0.5) {
    throw std::runtime_error("No iris detected in the probe image.");
  }

  std::cout « "Found iris in the image. Extracting template..." « std::endl;
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.detect`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.detect`
- In Swift use `SFEToolkitIris.detect`
- In .NET iris functionality is not implemented yet

3.9 Iris Template Extraction

SmartFace Embedded can be also used for 1:1 (verification) and 1:N (identification) iris recognition.

To perform 1:1 or 1:N matching it is necessary to extract an iris template. The iris template is a feature vector describing the iris. The size of the iris template is only 522 B so the memory requirements for identification are low and the matching is very fast.

In **SFE Toolkit** C API use function [sfeIrisTemplateExtract]

Example

```
SFEIrisTemplate iris_template;
{ // STAGE 3 Extract probe iris template
  // Use template extraction solver to create new iris
  // template
  error = sfeIrisTemplateExtract(template_solver, image, &detected_iris,
                                 &iris_template);
  utils::checkError(error);
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.extract`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.extract`
- In Swift use `SFEToolkitIris.extract`
- In .NET iris functionality is not implemented yet

The iris template includes the template version and quality. Both can be obtained using C API.

- **Template version** is used during matching to make sure the versions of the template being matched are the same.
- **Template quality** is a value in the range $<0.0, 1.0>$ and it indicates the suitability of the template for matching. The higher template quality leads to better-matching results.

In **SFE Toolkit** C API use functions [sfeIrisTemplateVersion] and [sfeIrisTemplateQuality]

Example

```
SFEIrisTemplateVersion version = {};
error = sfeIrisTemplateVersion(&probe_iris_template, &version);
utils::checkError(error);

std::cout « "Version of the probe template: " « version.version_major
         « '.' « version.version_minor « std::endl;
float quality = {};
error = sfeIrisTemplateQuality(&probe_iris_template, &quality);
utils::checkError(error);

std::cout « "Quality of the probe template: " « quality « std::endl;
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.template_quality` and `sfe_iris.template_version`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.templateQuality` and `com.innovatrics.smartface.embedded.toolkit.iris.templateVersion`
- In Swift use `SFEToolkitIris.templateQuality` and `SFEToolkitIris.templateVersion`
- In .NET iris functionality is not implemented yet

3.9.1 Iris verification template compatibility

The iris template has its system of version numbers. Major number change defines radical changes in the internal structure of the template document. Minor number change defines a change in extraction algorithm or minor data change of template. Newer iris templates (extracted with a faster or more accurate algorithm) are not compatible with previous templates.

In **SFE Toolkit** C API use function [sfeIrisTemplateVersion]

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.template_version`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.templateVersion`
- In Swift use `SFEToolkitIris.templateVersion`
- In .NET iris functionality is not implemented yet

3.10 Iris Template Verification

Verification is a comparison of two extracted iris templates and it returns a matching score which refers to the probability that the two templates are extracted from the iris of the same eye. When the matching score is higher than a certain score threshold then the iris images belong to the same eye with high probability. The matching score range is <0.0, 1.0>. Its values can be interpreted as follows:

- Low values of the score, i.e. range <0, 0.6>, are normalized using FAR values and this formula: `score_L=-10*log(FAR)/100`. It means that score 0.3 is related to FAR=1:1000=10^-3, and score 0.5 is related to FAR=1:100000=10^-5 (evaluated on our large testing non-matching pairs dataset).
- High values of the score, i.e. range <0.8, 1.0>, are normalized using FRR values and this formula: `score_H=100/3*(FRR + 2)/100`. It means that a score of 0.8 is related to FRR=0.4, and a score of 0.9 is related to FRR=0.7 (evaluated on our large testing matching pairs dataset).
- Scores values in the range (0.6, 0.8) are weighted averages of *score_L* and *score_H*. This normalization helps the users to select the score threshold according to their needs. If it is too low e.g. threshold = 0.3, then the chance of falsely accepted non-matching irises is quite high (FAR=10^-3). When it is too high e.g. threshold = 0.9, then the chance of false rejected matching irises is quite high (FRR=0.7).

In **SFE Toolkit** C API use function [sfeIrisTemplateMatch]

Example

```
{ // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
  utils::printFormatted("1:1 MATCHING");

  if (identification_results.size() == 0) {
    std::cout « "No candidates found above identification threshold "
             « identification_threshold « std::endl;
    return 0;
  }

  // Since it's sorted vector by score, the best candidate is the first one
  auto match_iris_template =
      iris_template_gallery[identification_results[0].index];

  float match_confidence;
  error = sfeIrisTemplateMatch(&probe_iris_template, &match_iris_template,
                               &match_confidence);
  utils::checkError(error);

  std::cout
      « "Matching score of probe template with the best template, score: "
      « match_confidence « std::endl;
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.template_match`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.templateMatch`
- In Swift use `SFEToolkitIris.templateMatch`
- In .NET iris functionality is not implemented yet

3.11 Iris Template Identification

Identification is the process of finding the best matching candidates for the probe template in the template gallery (database). The maximum number of best candidates to be returned by the identification function can be configured. It depends on the use case. Identification returns only candidates whose matching score with the probe template is higher or equal to the matching score threshold. For the matching score threshold recommendation please refer to the section Iris Template Verification

The identification is pure CPU-based and it can be configured how many CPU cores will be used for identification to optimize the performance within your application and also the CPU utilization.

In **SFE Toolkit** C API use function [sfeIrisTemplateIdentify]

Example

```
size_t candidate_count = 1;
std::vector<SFETemplateIdentificationCandidate> identification_results(
    candidate_count);
{ // STAGE 3: Identify the probe template
  utils::printFormatted("1:N IDENTIFICATION");

  error = sfeIrisTemplateIdentify(
      &probe_iris_template, iris_template_gallery.data(),
      iris_template_gallery.size(), identification_threshold,
      identification_results.data(), &candidate_count, 4);
  utils::checkError(error);

  if (candidate_count == 0) {
    std::cout « "No candidates found above identification threshold "
             « identification_threshold « std::endl;
    return 0;
  }

  std::cout « "Found " « identification_results.size()
            « " candidates above identification threshold "
            « identification_threshold « std::endl;
  for (auto &result : identification_results)
    std::cout « "Template index: #" « result.index
             « ", score: " « result.score « std::endl;
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.template_identify`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.template↩ Identify`
- In Swift use `SFEToolkitIris.templateIdentify`
- In .NET iris functionality is not implemented yet

SFE Toolkit also supports the identification of Entities. Entity ID is used to group multiple iris templates of the same eye. The result of identification is a list of Entity IDs ordered by best matching score of iris templates associated with the same Entity ID.

In **SFE Toolkit** C API use function [sfeIrisEntityIdentify]

See the (Face Template Identification) section for entity example and documentation.

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_iris.entity_identify`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.iris.entityIdentify`
- In Swift use `SFEToolkitIris.entityIdentify`
- In .NET iris functionality is not implemented yet

3.12 Palm detection

Palm detection is a process of detecting the palm in the hand image. The whole hand should be present in the input image for the palm to be detected correctly.

Palm detection returns detected palm as a structure containing the palm bounding box, palm keypoints, hand side (left/right), and hand orientation (palmar/dorsal).

In **SFE Toolkit** C API use function [sfeDetect] with a palm detection solver. The unified `sfeDetect()` function automatically determines the detection modality based on the solver type.

Example

```
SFEDetection detected_palm = {};
{ // STAGE 2: Detect palm in the image

  // Detect palm in the image
  float detection_threshold = 0.1f;
  size_t detected_count = 1;
  error = sfeDetect(detector_solver, image, detection_threshold,
                    &detected_palm, &detected_count);
  utils::checkError(error);

  if (detected_count > 0 && detected_palm.confidence < 0.5) {
    throw std::runtime_error("No palm detected in the probe image.");
  }

  std::cout « "Found palm in the image. Extracting template..." « std::endl;
}
```

3.13 Palm Template Extraction

SmartFace Embedded can be also used for 1:1 (verification) and 1:N (identification) palm recognition.

To perform 1:1 or 1:N matching it is necessary to extract a palm template. The palm template is a binary representation of the palm. The size of the palm template is only 522B.

In **SFE Toolkit** C API use function [sfePalmTemplateExtract]

Example

```
SFEPalmTemplate palm_template;
{ // STAGE 4: Extract probe palm template
  // Use template extraction solver to create new palm template
  // Requires palm landmarks from previous step
  error = sfePalmTemplateExtract(extraction_solver, image, &landmarks,
                                 &palm_template);
  utils::checkError(error);
}
```

The palm template includes the template version that can be obtained using C API.

In **SFE Toolkit** C API use functions [sfePalmTemplateVersion]

Example

```
  SFEPalmTemplateVersion version = {};
  error = sfePalmTemplateVersion(&probe_palm_template, &version);
  utils::checkError(error);

  std::cout « "Version of the probe template: " « version.major « '.'
            « version.minor « '.' « version.patch « std::endl;
```

3.14 Palm Template Verification

Verification is a comparison of two extracted palm templates and it returns a matching score which refers to the probability that the two templates are extracted from the same palm or hand. When the matching score is higher than a certain score threshold then the palm images belong to the same hand with high probability. The matching score range is $<0.0, 1.0>$.

In **SFE Toolkit** C API use function [sfePalmTemplateMatch]

Example

```
{ // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
  utils::printFormatted("1:1 MATCHING");

  if (identification_results.size() == 0) {
    std::cout « "No candidates found above identification threshold "
             « identification_threshold « std::endl;
    return 0;
  }

  // Since it's sorted vector by score, the best candidate is the first one
  auto match_palm_template =
      palm_template_gallery[identification_results[0].index];

  float match_confidence;
  error = sfePalmTemplateMatch(&probe_palm_template, &match_palm_template,
                               &match_confidence);
  utils::checkError(error);

  std::cout
      « "Matching score of probe template with the best template, score: "
      « match_confidence « std::endl;
}
```

3.15 Palm Template Identification

Identification is the process of finding the best matching candidates for the probe template in the template gallery (database). The maximum number of best candidates to be returned by the identification function can be configured. It depends on the use case. Identification returns only candidates whose matching score with the probe template is higher or equal to the matching score threshold.

The identification is pure CPU-based and it can be configured how many CPU cores will be used for identification to optimize the performance within your application and also the CPU utilization.

In **SFE Toolkit** C API use function [sfePalmTemplateIdentify]

Example

```
size_t candidate_count = 1;
std::vector<SFETemplateIdentificationCandidate> identification_results(
    candidate_count);
{ // STAGE 3: Identify the probe template
  utils::printFormatted("1:N IDENTIFICATION");

  error = sfePalmTemplateIdentify(
      &probe_palm_template, palm_template_gallery.data(),
      palm_template_gallery.size(), identification_threshold,
      identification_results.data(), &candidate_count, 4);
  utils::checkError(error);

  if (candidate_count == 0) {
    std::cout « "No candidates found above identification threshold "
             « identification_threshold « std::endl;
    return 0;
  }

  std::cout « "Found " « identification_results.size()
           « " candidates above identification threshold "
           « identification_threshold « std::endl;
  for (auto &result : identification_results)
    std::cout « "Template index: #" « result.index
             « ", score: " « result.score « std::endl;
}
```

SFE Toolkit also supports the identification of Entities. Entity ID is used to group multiple palm templates of the hand or same person. The result of identification is a list of Entity IDs ordered by best matching score of palm templates associated with the same Entity ID.

In **SFE Toolkit** C API use function [sfePalmEntityIdentify]

See the (Face Template Identification) section for entity example and documentation.

3.16 Palm Liveness Detection

Palm recognition systems are vulnerable to spoof attacks made by non-real palms. It is an easy way to spoof palm recognition systems by palm images such as photographs, prints and videos which are easily available. A secure system needs a liveness evaluation strategy to guard against such spoofing.

Palm liveness detection is a process of recognizing spoof attacks and non-real palms. It can recognize a real palm against a photograph, prints and videos. The passive liveness score is defined over the interval $<0,1>$.

3.16.1 Recommended threshold for Palm passive liveness modes

Palms	Threshold	FAR [%]	FRR [%]
1:100 FAR (APCER)	0.8543	1	3.67
EER	0.8189	1.88	1.88
1:100 FRR (BPCER)	0.7952	3.93	1

In **SFE Toolkit** C API use function [sfePalmLivenessPassive] To use the correct liveness detection mode, load the appropriate solver:

- palm_liveness

Example

```
float liveness_score = 0.0f;
{ // STAGE 4: Liveness check
  utils::printFormatted("LIVENESS CHECK");
  error = sfePalmLivenessPassive(liveness_solver, image, &detected_palm,
                                 &liveness_score);
  utils::checkError(error);
}
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_palm.liveness_passive`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.palm.livenessPassive`
- In Swift use `SFEToolkitPalm.livenessPassive`
- In .NET use `Innovatrics.SFEToolkit.Palm.LivenessPassive`

3.17 Palm Attributes

Once a palm is detected, the **SFE Toolkit** can be used to calculate the following attributes:

- **Hand Position** Prediction of the hand position.

  Range <0, 1>. If it is nearer to 0 - it is left hand, if it is near to 1 it is right hand. You can set threshold to 0.5.

- **Hand Orientation** Prediction of the hand orientation.

  Range <0, 1>. If it is nearer to 0 - it is palmar side of the hand, if it is near to 1 it is dorsal side of the hand. You can set threshold to 0.5.

- **Quality** Indicates the quality of the palm image, used to decide whether it is suitable for further processing.

  Range <0, 1>. The recommended threshold for further palm processing is 0.6.

In **SFE Toolkit** C API use function [sfePalmAttributes]

Example

```
utils::printFormatted("ATTRIBUTES");
SFEPalmAttributes attributes = {};
error = sfePalmAttributes(attributes_solver, image, &landmarks,
                          &attributes);
utils::checkError(error);
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_palm.attributes`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.palm.attributes`
- In Swift use `SFEToolkitPalm.attributes`
- In .NET use `Innovatrics.SFEToolkit.Palm.Attributes`

3.18 Person Detection

SmartFace Embedded supports person detection with oriented bounding boxes for more accurate detection results.

3.18.1 Oriented Person Detection

Person detection with oriented bounding boxes provides more precise detection results compared to standard rectangular bounding boxes. This is particularly useful for detecting persons using fisheye cameras.

In **SFE Toolkit** C API use function [sfeDetect] with a person detection solver. The unified `sfeDetect()` function automatically determines the detection modality based on the solver type.

Example

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_core.detect` with a person detection solver
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.core.detect` with a person detection solver
- In Swift use `SFEToolkitCore.detect` with a person detection solver
- In .NET use `Innovatrics.SFEToolkit.Core.Detect` with a person detection solver

3.19 Person Attributes

SmartFace Embedded supports detection of 26 person attributes from a person image or crop. These attributes include clothing, accessories, and physical characteristics.

3.19.1 Supported Person Attributes

The system can detect the following 26 person attributes:

- **Clothing**: Hat, glasses, sunglasses, shirt, pants, shorts, skirt, dress, shoes, boots, sandals
- **Accessories**: Backpack, handbag, umbrella, scarf, tie
- **Physical**: Age group, gender, posture, body orientation
- **Activities**: Holding objects, walking, standing, sitting

3.19.2 Attribute Confidence and Thresholds

Each attribute returns a confidence score in the range <0,1>. Higher values indicate higher probability that the attribute is present in the image.

In **SFE Toolkit** C API use function [sfePersonAttributes]

Example

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_person.attributes`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.person.attributes`
- In Swift use `SFEToolkitPerson.attributes`
- In .NET use `Innovatrics.SFEToolkit.Person.Attributes`

3.20 Tattoo detection

Tattoo detection is a process of detecting the tattoo in the image.

Tattoo detection returns detected tattoos as a list of bounding box and confidence.

In **SFE Toolkit** C API use function [sfeDetect] with a tattoo detection solver. The unified `sfeDetect()` function automatically determines the detection modality based on the solver type.

Example

```
error = sfeSolverCreate(
    solver.value_or("").c_str(), SOLVER_PARAMETERS.data(), SOLVER_PARAMETERS.size(),
    &detection_solver);
utils::checkError(error);

// Run detection for up to 10 tattoos
size_t detection_count = detections.size();
float detection_threshold = 0.3;
error = sfeDetect(detection_solver, image, detection_threshold,
                        detections.data(), &detection_count);
```

3.21 Tattoo extraction

Tattoo extraction is a process of creating the tattoo biometric representation based on detection and the image of tattoo.

Tattoo extraction returns tattoo biometric representation.

In **SFE Toolkit** C API use function [sfeTattooTemplateExtract]

3.22 Supported platforms and features

3.22.1 Face modality

Solver	Face detect	Face landmarks	Face template extract	Face passive liveness	Demographic Attributes
ONNXRT	accurate	0.25, 0.50	fast, balanced, accurate	distant fast, nearby fast	fast, accurate
Rockchip RKNPU	accurate	0.25	accurate	distant fast	-
Rockchip RKNPU2	accurate	0.25	accurate	-	-
Ambarella	accurate	0.25	balanced, accurate	distant fast	-
Hailo	accurate	0.25	accurate	-	-
NXP iMX8	accurate	0.25 , 0.50	balanced, accurate	-	-

Other features such as face verification, face identification and face quality attributes or not implemented using a solver and so these are supported on all mentioned platforms.

3.22.2 Iris modality

Solver	Iris detect	Iris template extract
ONNXRT	yes	yes
Rockchip RKNPU2	yes	yes

Other features such as iris verification, iris identification, and iris template quality are not solver-specific and are supported on all mentioned platforms.

3.22.3 Palm modality

Solver	Palm detect	Palm template extract	Palm liveness
ONNXRT	yes	yes	yes
Ambarella	yes	yes	-

3.22.4 Tattoo modality

Solver	Tattoo detect	Tattoo extract
ONNXRT	yes	yes

3.22.5 Person modality

Solver	Person detect	Person attributes
ONNXRT	yes	yes

3.22.6 Face tracking

Solver	Face track
ONNXRT	yes
Rockchip RKNPU	yes
Rockchip RKNPU2	yes
Ambarella	yes

3.23 Input image data

SFE Toolkit can detect and recognize features of faces and irises in images. The is formatted as a raw color (3 channels) image. The image data structure is an array of bytes (unsigned char) where each pixel is formed by 3 bytes, which means 3 color channels ordered in BGR order. The origin of the image is in the upper-left corner. The size of the image data array can be calculated as `image height * image width * 3 bytes`.

SFE Toolkit provides functionality [sfeImageDecode] for loading of most common image formats (JPEG, PNG, BMP, ...) into an appropriate format suitable for SmartFace Embedded processing [SFEImage].

Example

```
// Load image data from file
auto image_data = utils::readFile(image_probe);

// Decode image from data
error = sfeImageDecode(image_data.data(), image_data.size(), &image);
utils::checkError(error);
```

Use the following function in SFE Toolkit bindings:

- In Python use `sfe_core.image_open`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.core.imageOpen`
- In Swift use `SFEToolkitCore.imageOpen`
- In .NET use `Innovatrics.SFEToolkit.Core.Image`

Chapter 4

Solvers

The SmartFace Embedded solvers are binaries processing the NN model inference and model-related pre-processing and post-processing operation. For each NN model, there is a specific solver.

Our NN models can be accelerated on a variety of HW, including CPUs, GPUs and NPUs. Depending on the target platform and inference engine we distinguish different solvers.

Currently, SFE Toolkit supports NN acceleration using the following inference engines.

4.1 ONNX Runtime

ONNX Runtime inference engine enables inference of NN models on a variety of HW using different so-called execution providers. ONNX Runtime solver comes with the suffix `onnxrt.solver` in its name.

ONNX Runtime solvers are currently supported for Windows x86, Linux x86, Jetson ARM64 and Android architectures.

SFE Toolkit currently uses the following ONNX Runtime versions:

- 1.13.1 - Linux, Windows and NVidia Jetson with Jetpack 5.0 and higher
- 1.12.0 - NVidia Jetson with Jetpack 4.6
- 1.7.2 - NVidia Jetson with Jetpack 4.4

The ONNX Runtime execution provider can be configured using a solver parameter ¨runtime_provider¨ or environment variable ONNXRUNTIME_SOLVER_RUNTIME_PROVIDER. Supported runtime/execution providers are:

- ¨cpu¨ - Linux, Windows, Jetson, Android
- ¨cuda¨ - Linux, Windows, Jetson
- ¨tensorrt¨ - Linux, Windows, Jetson *Note: Environment variable overrides solver parameter.*

ONNXRuntime solvers depend on the **onnxruntime** library. For Windows, Microsoft C and C++ (MSVC) runtime libraries are required to be installed see this page.

4.1.1 CPU

The default ONNX Runtime CPU Execution Provider is MLAS.

The performance and CPU utilization can be configured using the following solver parameters and environment variables:

- solver parameter ¨inter_threads¨ or env variable ONNXRUNTIME_SOLVER_INTER_THREADS
  - Sets the number of threads used to parallelize the execution of the graph (across nodes).
  - Default value is 0 which means the default number of threads will be used.
  - If ¨parallel¨ execution mode is turned on, this sets the maximum number of threads to use to run them in parallel.
  - If ¨sequential¨ execution mode is enabled this value is ignored, it acts as if it was set to 1.
- solver parameter ¨intra_threads¨ or env variable ONNXRUNTIME_SOLVER_INTRA_THREADS
  - Sets the number of threads used to parallelize the execution within nodes
  - Default value is 0 which means the default number of threads will be used.
- solver parameter ¨execution_mode¨ or env variable ONNXRUNTIME_SOLVER_EXECUTION_MODE
  - Controls whether the operators are executed in parallel or sequentially
  - ¨parallel¨ - execute operators in the graph in parallel.
  - ¨sequential¨ - execute operators in the graph sequentially.
  - default value is ¨parallel¨
- solver parameter ¨log_level¨ or env variable ONNXRUNTIME_SOLVER_LOG_LEVEL
  - Controls the logging level for ONNX solvers
  - Empty value means no logging
  - Supported values: ¨verbose¨, ¨info¨, ¨warning¨, ¨error¨, ¨fatal¨
  - Default value is empty (no logging)
- solver parameter ¨log_file¨ or env variable ONNXRUNTIME_SOLVER_LOG_FILE
  - Specifies the log file path for ONNX solver logging
  - Empty value means logging to stdout
  - Default value is empty (stdout)
- solver parameter ¨profile_prefix¨ or env variable ONNXRUNTIME_SOLVER_PROFILE_PREFIX
  - Enables dumping JSON statistics from inference
  - Specifies the prefix for the profile output files
  - Default value is empty (profiling disabled)
- solver parameter ¨session_config¨ or env variable ONNXRUNTIME_SOLVER_SESSION_CONFIG
  - Supports advanced session settings in ONNX solvers
  - Allows configuration of additional ONNX Runtime session parameters
  - Default value is empty (no additional configuration)

Note: Environment variable overrides solver parameter.

4.1.2 CUDA

The CUDA Execution Provider enables hardware-accelerated computation on Nvidia CUDA-enabled GPUs and NVidia Jetson platforms

The supported CUDA and cuDNN version requirements are documented in onnxruntime documentation.

You can specify the ID of a CUDA device where the NN model inference will be executed by setting a solver parameter ¨device_id¨ or env variable ONNXRUNTIME_SOLVER_DEVICE_ID:

- default value is 0 *Note: Environment variable overrides solver parameter.*

4.1.3 TensorRT

With the TensorRT execution provider, the ONNX Runtime delivers better inferencing performance on the same hardware compared to generic GPU acceleration.

The TensorRT execution provider in the ONNX Runtime makes use of NVIDIA's TensorRT Deep Learning inferencing engine to accelerate the ONNX model in their family of GPUs.

The supported TensorRT and CUDA versions are documented in onnxruntime documentation

You can configure TensorRT settings by environment variables. Find more details in onnxruntime documentation.

To decrease the ONNX model load time, you can enable the TensorRT engine caching with the following environment variables:

- ORT_TENSORRT_ENGINE_CACHE_ENABLE: Enable TensorRT engine caching. The default value is 0 (disabled), value 1 means caching is enabled.
- ORT_TENSORRT_CACHE_PATH: Specify the path for the TensorRT engine and profile files if ORT_↩ TENSORRT_ENGINE_CACHE_ENABLE is 1.

4.2 Rockchip NPU

Rockchip NPU is used for NN model acceleration on Rockchip SoC. SFE Toolkit supports both RKNPU and RKNPU2.

4.2.1 RKNPU

Supported chipsets are RV1109 and RV1126.

Rockchip solver comes with the suffix `rockchip.solver` in its name. The solvers are compatible with RKNPU 1.6.0.

4.3 RKNPU2

Supported chipsets are:

- RK3566/RK3568
- RK3588/RK3588S
- RV1103/RV1106
- RK3562 RV1109 and RV1126.

Rockchip solver comes with the suffix `rknn2.solver` in its name. The solvers are compatible with RKNPU2 1.5.2.

4.4 Ambarella CVFlow

Ambarella CVFlow is used for NN model acceleration on Ambarella CV28, CV25, CV22 and CV2 chips.

Ambarella solver comes with the suffix `ambarella.solver` in its name.

We support two different major versions of Ambarella SDK as of today:

- v3.0
- v2.5.8.

Solvers built against Ambarella SDK 3.0 use higher-level API (which Ambarella calls EazyAI).

Solvers built against Ambarella SDK v2.5.8 use lower-level API (called NNCTRL).

4.5 Tensorflow Lite

Tensorflow Lite inference engine is used for NN model acceleration on NXP's NPU hardware i.MX 8 series.

Tensorflow Lite solver comes with the suffix `tflite.solver` in its name.

Acceleration on the NPU is handled by the VX delegate plugin. VX Delegate enables accelerating the inference on on-chip hardware accelerator on i.MX 8 series. The VX Delegate directly uses the hardware accelerator driver (OpenVX with extension) to fully utilize the accelerator capabilities.

SFE Toolkit currently uses Tensorflow Lite 2.9.1.

Supported TFLite solver parameters

- **tflite_solver.delegate**
- **tflite_solver.num_threads**
- **tflite_solver.vx_delegate.device_id**

- **tflite_solver.vx_delegate.cache**
- **tflite_solver.vx_delegate.cache_file_path**

Supported TFLite solver environment variables

- **TFLITE_SOLVER_DELEGATE**
- **TFLITE_SOLVER_NUM_THREADS**
- **TFLITE_SOLVER_VX_DELEGATE_DEVICE_ID**
- **TFLITE_SOLVER_VX_DELEGATE_ENABLE_CACHING**
- **TFLITE_SOLVER_VX_DELEGATE_CACHE_FILE_PATH**

Note: Environment variable overrides solver parameter.

4.5.1 Parameters and their values

Delegate parameter

Enables users to specify a delegate that should be used for inference. For example, GPU delegate or VX delegate can be specified. Possible values of this parameter, whether it is set via generic solver API or using an environment variable, are:

- ¨cpu¨
- ¨gpu¨
- ¨nnapi¨
- ¨vx¨

Default value is `cpu`

Num Threads parameter

Allows users to specify the number of threads to be used for model inference. Currently, this parameter `only affects CPU delegate` and does nothing if specified along with a different delegate. Possible values are `non-float numbers` starting from -1 to infinity`. Special behavior is triggered when the following values are provided:

- `-1` - let the tflite engine choose the most suitable number of threads.
- `0` - Multithreading disabled.

Default value is `-1`*, which means that the Tensorflow-lite engine will decide how many threads it's going to use.*

Vx Delegate device id

In an environment with multiple VX NPUs, allows you to specify a device ID that a solver should use for inference.

Default value is `0`*.*

Vx Delegate caching enabled and cache file path

Enables you to turn on the VX model caching. This is useful when the first VX model inference takes too long. It is capable of caching its models into files and reusing them in another instance of the process. You can turn this VX feature on by setting `tflite_solver.vx_delegate.cache` to ¨true¨. By default, it will cache a model into a file with path: `/tmp/tflite_solver.vxcache`. *Default behavior: caching is disabled*

Chapter 5

Upgrade Guide: SFE Toolkit 3.x to 4.0

This guide covers the breaking changes when upgrading from SFE Toolkit 3.x to 4.0.

5.1 FFI Changes (C API)

5.1.1 Detection API

Before (3.x):

```
SFEFaceDetect faces[10];
size_t count = 10;
sfeFaceDetect(solver, image, 0.5f, faces, &count);
```

After (4.0):

```
SFEDetection detections[10];
size_t count = 10;
sfeDetect(solver, image, 0.5f, detections, &count);
// Check modality: detections[i].modality == SFE_MODALITY_FACE
```

5.1.2 Type Renames

3.x	4.0
SFEBbox	SFEBoundingBox
SFEOrientedBbox	SFEOrientedBoundingBox
SFELicenseSource	SFELicenseType
SFE_LICENSE_SOURCE_*	SFE_LICENSE_TYPE_*
SFEFaceDetect	SFEDetection
SFEIrisDetect	SFEDetection
SFEPalmDetect	SFEDetection
SFEObjectDetect	SFEDetection
SFETattooDetection	SFEDetection
SFEFaceDetectAccuracyType	SFEFaceDetectionAccuracyType

5.1.3 Function Renames

3.x	4.0
sfeImageLoad()	sfeImageDecode()
sfeImageSave()	sfeImageEncode()
sfeSolverCreateWithParameters()	sfeSolverCreate()
sfeFaceDetect()	sfeDetect()
sfeIrisDetect()	sfeDetect()
sfePalmDetect()	sfeDetect()
sfeObjectDetect()	sfeDetect()
sfeTattooDetect()	sfeDetect()

5.1.4 Signature Changes

License Validation

```
// 3.x
SFELicenseSource source;
sfeLicenseValidate(&source);

// 4.0
sfeLicenseValidate();
SFELicenseType type;
sfeLicenseType(&type);
```

Image Info

```
// 3.x
size_t width, height;
SFEImageFormat format;
sfeImageInfo(data, len, &width, &height, &format);

// 4.0
SFEImageInfo info;
sfeImageInfo(data, len, &info);
// Access: info.width, info.height, info.format
```

Image Color Transpose

```
// 3.x (in-place)
sfeImageColorTranspose(image);

// 4.0 (returns new image)
SFEImage transposed;
sfeImageColorTranspose(image, &transposed);
```

Face Landmarks

```
// 3.x
sfeFaceLandmarks(solver, image, bbox, landmarks);

// 4.0
sfeFaceLandmarks(solver, image, &detection, landmarks);
```

Face Template Extract

```
// 3.x
sfeFaceTemplateExtract(solver, image, landmarks, &tmpl);

// 4.0
sfeFaceTemplateExtract(solver, image, &detection, landmarks, &tmpl);
```

Face Liveness

```
// 3.x
float score;
sfeFaceLivenessPassive(solver, image, landmarks, &score);

// 4.0
SFEFaceLiveness liveness;
sfeFaceLivenessPassive(solver, image, landmarks, &liveness);
// Access: liveness.score
```

Face Crop

```
// 3.x
sfeFaceCrop(image, landmarks, (uint32_t)extension, max_size, &crop);

// 4.0
sfeFaceCrop(image, landmarks, (float)extension, max_size, &crop);
```

5.1.5 Palm API Changes

New intermediate step with `SFEPalmLandmarks`:

```
// 3.x
sfePalmTemplateExtract(solver, image, &palm_detect, &tmpl);
sfePalmAttributes(solver, image, &palm_detect, &attrs);

// 4.0
SFEPalmLandmarks landmarks;
sfePalmLandmarks(solver, image, &detection, &landmarks);
sfePalmTemplateExtract(solver, image, &landmarks, &tmpl);
sfePalmAttributes(solver, image, &landmarks, &attrs);
```

5.1.6 Tracker API

```
// 3.x
SFEFaceTracker tracker;
sfeFaceTrackerCreate(0.5f, 0.6f, 0.3f, 30, &tracker);
sfeFaceTrackerUpdate(tracker, faces, count, tracked, &tracked_count);
sfeFaceTrackerFree(tracker);

// 4.0
SFETracker tracker;
sfeDetectionTrackerCreate(0.5f, 0.6f, 0.4f, 0.3f, 30, &tracker);  // 5 params now
sfeDetectionTrackerUpdate(tracker, detections, count, tracked, &tracked_count);
sfeDetectionTrackerFree(tracker);
```

5.1.7 Removed Type Aliases

Replace deprecated aliases with unified types:

Deprecated	Use Instead
`SFEFaceEntity`, `SFEIrisEntity`, etc.	`SFEEntity`
`SFEIdentificationResult`	`SFETemplateIdentificationCandidate`
`SFEFaceTemplateIdentificationCandidate`	`SFETemplateIdentificationCandidate`
`SFEFaceEntityIdentificationCandidate`	`SFEEntityIdentificationCandidate`
`SFEPalmQualityAttributes`	`SFEPalmAttributes`

5.2 UniFFI Changes (Python, Kotlin, Swift, .NET)

5.2.1 Detection API

Python:

```python
# 3.x
from sfe_toolkit import sfe_face
detections = sfe_face.detect(solver, image, threshold)

# 4.0
from sfe_toolkit import sfe_core
detections = sfe_core.detect(solver, image, threshold)
# detections[i].modality indicates type
```

Kotlin:

```kotlin
// 3.x
val detections = Face.detect(solver, image, threshold)

// 4.0
val detections = Core.detect(solver, image, threshold)
```

Swift:

```swift
// 3.x
let detections = SFEToolkitFace.detect(solver: solver, image: image, threshold: threshold)

// 4.0
let detections = SFEToolkitCore.detect(solver: solver, image: image, threshold: threshold)
```

5.2.2 Type Renames

3.x	4.0
`Bbox`	`BoundingBox`
`OrientedBbox`	`OrientedBoundingBox`
`LicenseSource`	`LicenseType`
`FaceDetect`, `IrisDetect`, etc.	`Detection`

5.2.3 Function Renames

3.x	4.0
`image_load()`	`image_decode()`
`image_save()`	`image_encode()`
`face.detect()`	`core.detect()`
`iris.detect()`	`core.detect()`
`palm.detect()`	`core.detect()`

5.2.4 Palm API

```python
# 3.x
template = sfe_palm.extract(solver, image, detection)
attributes = sfe_palm.attributes(solver, image, detection)

# 4.0
landmarks = sfe_palm.landmarks(solver, image, detection)
template = sfe_palm.extract(solver, image, landmarks)
attributes = sfe_palm.attributes(solver, image, landmarks)
```

5.2.5 Tracker API

```python
# 3.x
tracker = sfe_face.tracker_create(0.5, 0.6, 0.3, 30)

# 4.0
tracker = sfe_core.tracker_create(0.5, 0.6, 0.4, 0.3, 30)  # new_track_thresh added
```

5.2.6 Face Liveness

```python
# 3.x
score = sfe_face.liveness_passive(solver, image, landmarks)

# 4.0
result = sfe_face.liveness_passive(solver, image, landmarks)
score = result.score
```

5.2.7 Entity Identification Result

`EntityIdentificationCandidate` now includes an `index` field:

```
# 4.0
for candidate in candidates:
    print(candidate.score, candidate.entity, candidate.index)
```

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Data Structure Documentation

8.1 SFEBoundingBox Struct Reference

Bounding box.
`#include <sfe_core_detection.h>`

Data Fields

- float x

  X coordinate of the top left corner of the bounding box relative to source image size - range <0,1>
- float y

  Y coordinate of the top left corner of the bounding box relative to source image size - range <0,1>
- float width

  Width of the bounding box relative to source image size - range <0,1>
- float height

  Height of the bounding box relative to source image size - range <0,1>

8.1.1 Detailed Description

Bounding box.
Definition at line 18 of file sfe_core_detection.h.

8.1.2 Field Documentation

8.1.2.1 height

`float SFEBoundingBox::height`
Height of the bounding box relative to source image size - range <0,1>

Examples

example_face_liveness.cpp, example_palm_attributes.cpp, and example_palm_liveness.cpp.

Definition at line 30 of file sfe_core_detection.h.

8.1.2.2 width

`float SFEBoundingBox::width`
Width of the bounding box relative to source image size - range <0,1>

Examples

example_face_liveness.cpp, example_palm_attributes.cpp, and example_palm_liveness.cpp.

Definition at line 27 of file sfe_core_detection.h.

8.1.2.3 x

`float SFEBoundingBox::x`
X coordinate of the top left corner of the bounding box relative to source image size - range <0,1>

Examples

example_face_liveness.cpp, example_palm_attributes.cpp, and example_palm_liveness.cpp.

Definition at line 21 of file sfe_core_detection.h.

8.1.2.4 y

`float SFEBoundingBox::y`
Y coordinate of the top left corner of the bounding box relative to source image size - range <0,1>

Examples

example_face_liveness.cpp, example_palm_attributes.cpp, and example_palm_liveness.cpp.

Definition at line 24 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.2 SFEDetection Struct Reference

Core detection - tagged union containing all detection types.
`#include <sfe_core_detection.h>`

Data Fields

- SFEBoundingBox bounding_box
  Bounding box (common for all detection types)
- SFEOrientedBoundingBox oriented_bounding_box
  Oriented bounding box (common for all detection types)
- float confidence
  Detection confidence - range <0,1> (common for all detection types)
- SFEModality modality
  Detection modality discriminator.
- SFEModalityData modality_data
  Union of all detection variant data structs (type-specific fields only)

8.2.1 Detailed Description

Core detection - tagged union containing all detection types.

Note

Contains a discriminator and a union of detection data types. Bounding box, oriented bounding box, and confidence are common for all detection types.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 323 of file sfe_core_detection.h.

8.2.2 Field Documentation

8.2.2.1 bounding_box

`SFEBoundingBox SFEDetection::bounding_box`
Bounding box (common for all detection types)

Examples

example_face_liveness.cpp, example_palm_attributes.cpp, and example_palm_liveness.cpp.

Definition at line 325 of file sfe_core_detection.h.

8.2.2.2 confidence

`float SFEDetection::confidence`
Detection confidence - range <0,1> (common for all detection types)

Examples

example_face_liveness.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 329 of file sfe_core_detection.h.

8.2.2.3 modality

`SFEModality SFEDetection::modality`
Detection modality discriminator.
Definition at line 331 of file sfe_core_detection.h.

8.2.2.4 modality_data

`SFEModalityData SFEDetection::modality_data`
Union of all detection variant data structs (type-specific fields only)
Definition at line 333 of file sfe_core_detection.h.

8.2.2.5 oriented_bounding_box

`SFEOrientedBoundingBox SFEDetection::oriented_bounding_box`
Oriented bounding box (common for all detection types)
Definition at line 327 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.3 SFEDetectionInputSize Struct Reference

Detection input size struct.
`#include <sfe_face.h>`

Data Fields

- uint32_t width

  Recommended image width.
- uint32_t height

  Recommended image height.

8.3.1 Detailed Description

Detection input size struct.

Examples

example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

Definition at line 59 of file sfe_face.h.

8.3.2 Field Documentation

8.3.2.1 height

`uint32_t SFEDetectionInputSize::height`
Recommended image height.
Definition at line 63 of file sfe_face.h.

8.3.2.2 width

`uint32_t SFEDetectionInputSize::width`
Recommended image width.
Definition at line 61 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.4 SFEEntity Struct Reference

Entity type, used to group templates for entity identification. This type is compatible with uuid_v4 that can be used to generate unique ids.
`#include <sfe_core.h>`

Data Fields

- uint8_t uuid [16]

8.4.1 Detailed Description

Entity type, used to group templates for entity identification. This type is compatible with uuid_v4 that can be used to generate unique ids.

Examples

example_face_track.cpp.

Definition at line 114 of file sfe_core.h.

8.4.2 Field Documentation

8.4.2.1 uuid

`uint8_t SFEEntity::uuid[16]`

Examples

example_face_track.cpp.

Definition at line 115 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.5 SFEEntityIdentificationCandidate Struct Reference

Candidate for identification by entity.
`#include <sfe_core.h>`

Data Fields

- float score

  matching score
- size_t index

  index of the entity from the input gallery
- SFEEntity entity

  candidate entity id

8.5.1 Detailed Description

Candidate for identification by entity.
Definition at line 127 of file sfe_core.h.

8.5.2 Field Documentation

8.5.2.1 entity

`SFEEntity SFEEntityIdentificationCandidate::entity`
candidate entity id
Definition at line 133 of file sfe_core.h.

8.5.2.2 index

`size_t SFEEntityIdentificationCandidate::index`
index of the entity from the input gallery
Definition at line 131 of file sfe_core.h.

8.5.2.3 score

`float SFEEntityIdentificationCandidate::score`
matching score
Definition at line 129 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.6 SFEFaceArea Struct Reference

Face area struct.
`#include <sfe_face.h>`

Data Fields

- float size

  absolute face size
- float area

  size of face area relative to the whole image; value in range <0,1>
- float area_in_image

  size of face area intersected with the whole image and relative to the whole image; value in range <0,1>

8.6.1 Detailed Description

Face area struct.

Examples

example_face_liveness.cpp.

Definition at line 237 of file sfe_face.h.

8.6.2 Field Documentation

8.6.2.1 area

`float SFEFaceArea::area`
size of face area relative to the whole image; value in range <0,1>

Examples

example_face_liveness.cpp.

Definition at line 241 of file sfe_face.h.

8.6.2.2 area_in_image

`float SFEFaceArea::area_in_image`
size of face area intersected with the whole image and relative to the whole image; value in range <0,1>

Examples

example_face_liveness.cpp.

Definition at line 244 of file sfe_face.h.

8.6.2.3 size

`float SFEFaceArea::size`
absolute face size

Examples

example_face_liveness.cpp.

Definition at line 239 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.7 SFEFaceCrop Struct Reference

Face crop struct.
`#include <sfe_face.h>`

Data Fields

- float face_size_extension

  Defines the size of the crop as an extension of the detection bounding box.
- SFEBoundingBox crop_box

  Bounding box of cropped face including the crop extension.
- SFEImage crop_image

  Image of face cropped according the crop_box.

8.7.1 Detailed Description

Face crop struct.

Examples

example_face_identify.cpp.

Definition at line 367 of file sfe_face.h.

8.7.2 Field Documentation

8.7.2.1 crop_box

`SFEBoundingBox SFEFaceCrop::crop_box`
Bounding box of cropped face including the crop extension.
Definition at line 372 of file sfe_face.h.

8.7.2.2 crop_image

`SFEImage SFEFaceCrop::crop_image`
Image of face cropped according the crop_box.

Examples

example_face_identify.cpp.

Definition at line 374 of file sfe_face.h.

8.7.2.3 face_size_extension

`float SFEFaceCrop::face_size_extension`
Defines the size of the crop as an extension of the detection bounding box.

Examples

example_face_identify.cpp.

Definition at line 370 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.8 SFEFaceDemographicAttributes Struct Reference

Demographic attributes of the detected face.
`#include <sfe_face.h>`

Data Fields

- float age
  Estimated age in years.
- float gender
  Normalized gender score in range [-1, 1]. Values near -1 indicate male, values near 1 indicate female.

8.8.1 Detailed Description

Demographic attributes of the detected face.

Examples

example_face_demographic_attributes.cpp.

Definition at line 309 of file sfe_face.h.

8.8.2 Field Documentation

8.8.2.1 age

`float SFEFaceDemographicAttributes::age`
Estimated age in years.
Definition at line 311 of file sfe_face.h.

8.8.2.2 gender

`float SFEFaceDemographicAttributes::gender`
Normalized gender score in range [-1, 1]. Values near -1 indicate male, values near 1 indicate female.
Definition at line 314 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.9 SFEFaceDetectionData Struct Reference

`#include <sfe_core_detection.h>`

Data Fields

- char _reserved

  Reserved field to ensure C/C++ compatibility.

8.9.1 Detailed Description

Definition at line 54 of file sfe_core_detection.h.

8.9.2 Field Documentation

8.9.2.1 _reserved

`char SFEFaceDetectionData::_reserved`
Reserved field to ensure C/C++ compatibility.
Definition at line 56 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.10 SFEFaceHeadPose Struct Reference

Face head pose struct containing angle rotations of head.
`#include <sfe_face.h>`

Data Fields

- float pitch

  Face attribute representing angle rotation of head towards camera reference frame around X-axis as per DIN9300.
- float yaw

  Face attribute representing angle rotation of head towards camera reference frame around Y-axis as per DIN9300.
- float roll

  Face attribute representing angle rotation of head towards camera reference frame around Z-axis as per DIN9300.

8.10.1 Detailed Description

Face head pose struct containing angle rotations of head.

Examples

example_face_liveness.cpp.

Definition at line 259 of file sfe_face.h.

8.10.2 Field Documentation

8.10.2.1 pitch

`float SFEFaceHeadPose::pitch`
Face attribute representing angle rotation of head towards camera reference frame around X-axis as per DIN9300.

Examples

example_face_liveness.cpp.

Definition at line 262 of file sfe_face.h.

8.10.2.2 roll

`float SFEFaceHeadPose::roll`
Face attribute representing angle rotation of head towards camera reference frame around Z-axis as per DIN9300.

Examples

example_face_liveness.cpp.

Definition at line 268 of file sfe_face.h.

8.10.2.3 yaw

`float SFEFaceHeadPose::yaw`
Face attribute representing angle rotation of head towards camera reference frame around Y-axis as per DIN9300.

Examples

example_face_liveness.cpp.

Definition at line 265 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.11 SFEFaceLandmarks Struct Reference

Face landmarks struct.
`#include <sfe_face.h>`

Data Fields

- SFEFaceLandmarkType landmark_type
  Type of the landmark detected on the face.
- float confidence
  Confidence of the landmark detected on the face - range <0,1>
- float x
  X coordinate of the landmark relative to source image - range <0,1>
- float y
  Y coordinate of the landmark relative to source image - range <0,1>

8.11.1 Detailed Description

Face landmarks struct.
Definition at line 47 of file sfe_face.h.

8.11.2 Field Documentation

8.11.2.1 confidence

`float SFEFaceLandmarks::confidence`
Confidence of the landmark detected on the face - range <0,1>
Definition at line 51 of file sfe_face.h.

8.11.2.2 landmark_type

`SFEFaceLandmarkType SFEFaceLandmarks::landmark_type`
Type of the landmark detected on the face.
Definition at line 49 of file sfe_face.h.

8.11.2.3 x

`float SFEFaceLandmarks::x`
X coordinate of the landmark relative to source image - range <0,1>
Definition at line 53 of file sfe_face.h.

8.11.2.4 y

`float SFEFaceLandmarks::y`
Y coordinate of the landmark relative to source image - range <0,1>
Definition at line 55 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.12 SFEFaceLiveness Struct Reference

Face liveness struct.
`#include <sfe_face.h>`

Data Fields

- float score

  Normalized passive liveness score - range <0,1>

8.12.1 Detailed Description

Face liveness struct.

Examples

example_face_liveness.cpp.

Definition at line 220 of file sfe_face.h.

8.12.2 Field Documentation

8.12.2.1 score

`float SFEFaceLiveness::score`
Normalized passive liveness score - range <0,1>

Examples

example_face_liveness.cpp.

Definition at line 222 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.13 SFEFaceQualityAttributes Struct Reference

Face quality attributes struct.
`#include <sfe_face.h>`

Data Fields

- float sharpness

  Normalized face attribute for evaluating whether an area of face image is not blurred. Sharpness values are from range <0,inf>. Values near 0 indicate 'very blurred', values near (or above) 1 indicate 'very sharp'. The decision threshold is in range <0.08, 0.7>.
- float brightness

  Normalized face attribute for evaluating whether an area of face is correctly exposed. Brightness values are from range <0,1>. Values near 0 indicate 'too dark', values near 1 indicate 'too light', values around 0 indicate OK. The decision thresholds are around 0.25 and 0.75.
- float contrast

  Normalized face attribute for evaluating whether an area of face is contrast enough. Contrast values are from range <0,1>. Values near 0 indicate 'very low contrast', values near 1 indicate 'very high contrast', values around 0 indicate OK. The decision thresholds are around 0.25 and 0.75.
- float unique_intensity_levels

  Normalized face attribute for evaluating whether an area of face has appropriate number of unique intensity levels. Unique intensity levels values are from range <0,1>. Values near 0 indicate 'very few unique intensity levels', values near 1 indicate 'enough unique intensity levels'. The decision threshold is around 0.5.

8.13.1 Detailed Description

Face quality attributes struct.

Examples

example_face_liveness.cpp.

Definition at line 283 of file sfe_face.h.

8.13.2 Field Documentation

8.13.2.1 brightness

`float SFEFaceQualityAttributes::brightness`
Normalized face attribute for evaluating whether an area of face is correctly exposed. Brightness values are from range <0,1>. Values near 0 indicate 'too dark', values near 1 indicate 'too light', values around 0 indicate OK. The decision thresholds are around 0.25 and 0.75.

Examples

example_face_liveness.cpp.

Definition at line 293 of file sfe_face.h.

8.13.2.2 contrast

`float SFEFaceQualityAttributes::contrast`
Normalized face attribute for evaluating whether an area of face is contrast enough. Contrast values are from range <0,1>. Values near 0 indicate 'very low contrast', values near 1 indicate 'very high contrast', values around 0 indicate OK. The decision thresholds are around 0.25 and 0.75.

Examples

example_face_liveness.cpp.

Definition at line 299 of file sfe_face.h.

8.13.2.3 sharpness

`float SFEFaceQualityAttributes::sharpness`
Normalized face attribute for evaluating whether an area of face image is not blurred. Sharpness values are from range <0,inf>. Values near 0 indicate 'very blurred', values near (or above) 1 indicate 'very sharp'. The decision threshold is in range <0.08, 0.7>.

Examples

example_face_liveness.cpp.

Definition at line 288 of file sfe_face.h.

8.13.2.4 unique_intensity_levels

`float SFEFaceQualityAttributes::unique_intensity_levels`
Normalized face attribute for evaluating whether an area of face has appropriate number of unique intensity levels. Unique intensity levels values are from range <0,1>. Values near 0 indicate 'very few unique intensity levels', values near 1 indicate 'enough unique intensity levels'. The decision threshold is around 0.5.

Examples

example_face_liveness.cpp.

Definition at line 305 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.14 SFEFaceTemplate Struct Reference

Face template struct.
`#include <sfe_face.h>`

Data Fields

- uint8_t data [SFE_FACE_TEMPLATE_SIZE]

8.14.1 Detailed Description

Face template struct.

Examples

example_face_identify.cpp, and example_face_identify_entity.cpp.

Definition at line 107 of file sfe_face.h.

8.14.2 Field Documentation

8.14.2.1 data

`uint8_t SFEFaceTemplate::data[SFE_FACE_TEMPLATE_SIZE]`
Definition at line 108 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.15 SFEFaceTemplateVersion Struct Reference

Face template version struct.
`#include <sfe_face.h>`

Data Fields

- uint8_t version_major
  major template version
- uint8_t version_minor
  minor template version

8.15.1 Detailed Description

Face template version struct.

Examples

example_face_identify.cpp.

Definition at line 112 of file sfe_face.h.

8.15.2 Field Documentation

8.15.2.1 version_major

`uint8_t SFEFaceTemplateVersion::version_major`
major template version

Examples

example_face_identify.cpp.

Definition at line 114 of file sfe_face.h.

8.15.2.2 version_minor

`uint8_t SFEFaceTemplateVersion::version_minor`
minor template version

Examples

example_face_identify.cpp.

Definition at line 116 of file sfe_face.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_face.h

8.16 SFEImage Struct Reference

Raw owned raster image representation, HWC|BGR order.
`#include <sfe_core.h>`

Data Fields

- size_t width

  Width of the image in pixels.
- size_t height

  Height of the image in pixels.
- unsigned char * data

  *Pointer to raw HWC|BGR data, size is width * height * 3.*

8.16.1 Detailed Description

Raw owned raster image representation, HWC|BGR order.

Note

For non-owning variant use `SFEImageView`.

Warning

This type retains ownership of the data, call `sfeImageFree` to free the allocated image data.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 70 of file sfe_core.h.

8.16.2 Field Documentation

8.16.2.1 data

`unsigned char* SFEImage::data`

Pointer to raw HWC|BGR data, size is width * height * 3.
Definition at line 76 of file sfe_core.h.

8.16.2.2 height

`size_t SFEImage::height`

Height of the image in pixels.

Examples

example_face_identify.cpp, and example_face_liveness.cpp.

Definition at line 74 of file sfe_core.h.

8.16.2.3 width

`size_t SFEImage::width`

Width of the image in pixels.

Examples

example_face_identify.cpp, and example_face_liveness.cpp.

Definition at line 72 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.17 SFEImageInfo Struct Reference

Image information structure, for describing image format and size.
`#include <sfe_core.h>`

Data Fields

- size_t width

  Width of the image in pixels.
- size_t height

  Height of the image in pixels.
- SFEImageFormat format

  Format of the image.

8.17.1 Detailed Description

Image information structure, for describing image format and size.
Definition at line 84 of file sfe_core.h.

8.17.2 Field Documentation

8.17.2.1 format

`SFEImageFormat SFEImageInfo::format`
Format of the image.
Definition at line 90 of file sfe_core.h.

8.17.2.2 height

`size_t SFEImageInfo::height`
Height of the image in pixels.
Definition at line 88 of file sfe_core.h.

8.17.2.3 width

`size_t SFEImageInfo::width`
Width of the image in pixels.
Definition at line 86 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.18 SFEIrisDetectionData Struct Reference

Iris detection data struct (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- SFEBoundingBox pupil_bounding_box

  Pupil bounding box.
- SFEIrisKeypoint keypoints [SFE_IRIS_KEYPOINT_COUNT]

  Keypoints detected on the eye.

8.18.1 Detailed Description

Iris detection data struct (type-specific fields only)
Definition at line 89 of file sfe_core_detection.h.

8.18.2 Field Documentation

8.18.2.1 keypoints

`SFEIrisKeypoint SFEIrisDetectionData::keypoints[SFE_IRIS_KEYPOINT_COUNT]`
Keypoints detected on the eye.
Definition at line 93 of file sfe_core_detection.h.

8.18.2.2 pupil_bounding_box

`SFEBoundingBox SFEIrisDetectionData::pupil_bounding_box`
Pupil bounding box.
Definition at line 91 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.19 SFEIrisKeypoint Struct Reference

Iris keypoint struct.
`#include <sfe_core_detection.h>`

Data Fields

- SFEIrisKeypointType keypoint_type
  Type of the keypoint.
- float confidence
  Confidence of the keypoint - range <0,1>
- float x
  X coordinate of the keypoint relative to source image - range <0,1>
- float y
  Y coordinate of the keypoint relative to source image - range <0,1>

8.19.1 Detailed Description

Iris keypoint struct.
Definition at line 74 of file sfe_core_detection.h.

8.19.2 Field Documentation

8.19.2.1 confidence

`float SFEIrisKeypoint::confidence`
Confidence of the keypoint - range <0,1>
Definition at line 78 of file sfe_core_detection.h.

8.19.2.2 keypoint_type

`SFEIrisKeypointType SFEIrisKeypoint::keypoint_type`
Type of the keypoint.
Definition at line 76 of file sfe_core_detection.h.

8.19.2.3 x

`float SFEIrisKeypoint::x`
X coordinate of the keypoint relative to source image - range <0,1>
Definition at line 80 of file sfe_core_detection.h.

8.19.2.4 y

`float SFEIrisKeypoint::y`
Y coordinate of the keypoint relative to source image - range <0,1>
Definition at line 82 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.20 SFEIrisTemplate Struct Reference

Iris template struct.
`#include <sfe_iris.h>`

Data Fields

- uint8_t data [SFE_IRIS_TEMPLATE_SIZE]

8.20.1 Detailed Description

Iris template struct.

Examples

example_iris_identify.cpp.

Definition at line 22 of file sfe_iris.h.

8.20.2 Field Documentation

8.20.2.1 data

`uint8_t SFEIrisTemplate::data[SFE_IRIS_TEMPLATE_SIZE]`
Definition at line 23 of file sfe_iris.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_iris.h

8.21 SFEIrisTemplateVersion Struct Reference

Iris template version struct.
`#include <sfe_iris.h>`

Data Fields

- uint8_t version_major
  Major template version.
- uint8_t version_minor
  Minor template version.

8.21.1 Detailed Description

Iris template version struct.

Examples

example_iris_identify.cpp.

Definition at line 27 of file sfe_iris.h.

8.21.2 Field Documentation

8.21.2.1 version_major

`uint8_t SFEIrisTemplateVersion::version_major`
Major template version.

Examples

example_iris_identify.cpp.

Definition at line 29 of file sfe_iris.h.

8.21.2.2 version_minor

`uint8_t SFEIrisTemplateVersion::version_minor`
Minor template version.

Examples

example_iris_identify.cpp.

Definition at line 31 of file sfe_iris.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_iris.h

8.22 SFEModalityData Union Reference

Union of detection data types (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- SFEFaceDetectionData face
  Face detection data.
- SFEIrisDetectionData iris
  Iris detection data.
- SFEPalmDetectionData palm
  Palm detection data.
- SFEObjectDetectionData object
  Object detection data.
- SFETattooDetectionData tattoo
  Tattoo detection data.
- SFEPersonDetectionData person
  Person detection data.
- SFEVisualCodeDetectionData visual_code
  Visual code detection data.

8.22.1 Detailed Description

Union of detection data types (type-specific fields only)
Definition at line 303 of file sfe_core_detection.h.

8.22.2 Field Documentation

8.22.2.1 face

`SFEFaceDetectionData SFEModalityData::face`
Face detection data.
Definition at line 305 of file sfe_core_detection.h.

8.22.2.2 iris

`SFEIrisDetectionData SFEModalityData::iris`
Iris detection data.
Definition at line 307 of file sfe_core_detection.h.

8.22.2.3 object

`SFEObjectDetectionData SFEModalityData::object`
Object detection data.
Definition at line 311 of file sfe_core_detection.h.

8.22.2.4 palm

`SFEPalmDetectionData SFEModalityData::palm`
Palm detection data.
Definition at line 309 of file sfe_core_detection.h.

8.22.2.5 person

`SFEPersonDetectionData SFEModalityData::person`
Person detection data.
Definition at line 315 of file sfe_core_detection.h.

8.22.2.6 tattoo

`SFETattooDetectionData SFEModalityData::tattoo`
Tattoo detection data.
Definition at line 313 of file sfe_core_detection.h.

8.22.2.7 visual_code

`SFEVisualCodeDetectionData SFEModalityData::visual_code`
Visual code detection data.
Definition at line 317 of file sfe_core_detection.h.
The documentation for this union was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.23 SFEObject Struct Reference

Object struct.
`#include <sfe_core_detection.h>`

Data Fields

- SFEObjectType object_type
  object types
- float confidence
  probability scores

8.23.1 Detailed Description

Object struct.
Definition at line 232 of file sfe_core_detection.h.

8.23.2 Field Documentation

8.23.2.1 confidence

`float SFEObject::confidence`
probability scores
Definition at line 236 of file sfe_core_detection.h.

8.23.2.2 object_type

`SFEObjectType SFEObject::object_type`
object types
Definition at line 234 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.24 SFEObjectDetectionData Struct Reference

Object detection data struct (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- SFEObject objects [SFE_OBJECT_DETECT]

  array of 80 Object structs

8.24.1 Detailed Description

Object detection data struct (type-specific fields only)
Definition at line 243 of file sfe_core_detection.h.

8.24.2 Field Documentation

8.24.2.1 objects

`SFEObject SFEObjectDetectionData::objects[SFE_OBJECT_DETECT]`
array of 80 Object structs
Definition at line 245 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.25 SFEOrientedBoundingBox Struct Reference

Oriented bounding box.
`#include <sfe_core_detection.h>`

Data Fields

- float center_x

  X coordinate of the center of the bounding box relative to source image size - range <0,1>
- float center_y

  Y coordinate of the center of the bounding box relative to source image size - range <0,1>
- float width

  Width of the bounding box relative to source image size - range <0,1>
- float height

  Height of the bounding box relative to source image size - range <0,1>

- float angle

  Angle of the bounding box in radians.

8.25.1 Detailed Description

Oriented bounding box.
Definition at line 34 of file sfe_core_detection.h.

8.25.2 Field Documentation

8.25.2.1 angle

`float SFEOrientedBoundingBox::angle`
Angle of the bounding box in radians.
Definition at line 49 of file sfe_core_detection.h.

8.25.2.2 center_x

`float SFEOrientedBoundingBox::center_x`
X coordinate of the center of the bounding box relative to source image size - range <0,1>
Definition at line 37 of file sfe_core_detection.h.

8.25.2.3 center_y

`float SFEOrientedBoundingBox::center_y`
Y coordinate of the center of the bounding box relative to source image size - range <0,1>
Definition at line 40 of file sfe_core_detection.h.

8.25.2.4 height

`float SFEOrientedBoundingBox::height`
Height of the bounding box relative to source image size - range <0,1>
Definition at line 46 of file sfe_core_detection.h.

8.25.2.5 width

`float SFEOrientedBoundingBox::width`
Width of the bounding box relative to source image size - range <0,1>
Definition at line 43 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.26 SFEPalmAttributes Struct Reference

`#include <sfe_palm.h>`

Data Fields

- float hand_position
- float hand_orientation
- float quality

  Range <0, 1>. The recommended threshold for further palm processing is 0.6.

8.26.1 Detailed Description

Examples

example_palm_attributes.cpp.

Definition at line 49 of file sfe_palm.h.

8.26.2 Field Documentation

8.26.2.1 hand_orientation

`float SFEPalmAttributes::hand_orientation`
Range <0, 1>. If it is nearer to 0 - it is palmar side of the hand, if it is near to 1 it is dorsal side of the hand. You can set threshold to 0.5.

Examples

example_palm_attributes.cpp.

Definition at line 56 of file sfe_palm.h.

8.26.2.2 hand_position

`float SFEPalmAttributes::hand_position`
Range <0, 1>. If it is nearer to 0 - it is left hand, if it is near to 1 it is right hand. You can set threshold to 0.5.

Examples

example_palm_attributes.cpp.

Definition at line 52 of file sfe_palm.h.

8.26.2.3 quality

`float SFEPalmAttributes::quality`
Range <0, 1>. The recommended threshold for further palm processing is 0.6.

Examples

example_palm_attributes.cpp.

Definition at line 58 of file sfe_palm.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_palm.h

8.27 SFEPalmDetectionData Struct Reference

Palm detection data struct (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- bool has_landmarks

  When true, landmarks is valid; when false, do not read landmarks.
- SFEPalmLandmarks landmarks

  Palm landmarks; only meaningful when has_landmarks is true.

8.27.1 Detailed Description

Palm detection data struct (type-specific fields only)
Definition at line 138 of file sfe_core_detection.h.

8.27.2 Field Documentation

8.27.2.1 has_landmarks

`bool SFEPalmDetectionData::has_landmarks`
When true, landmarks is valid; when false, do not read landmarks.
Definition at line 140 of file sfe_core_detection.h.

8.27.2.2 landmarks

`SFEPalmLandmarks SFEPalmDetectionData::landmarks`
Palm landmarks; only meaningful when has_landmarks is true.
Definition at line 142 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.28 SFEPalmKeypoint Struct Reference

Palm keypoint struct.
`#include <sfe_core_detection.h>`

Data Fields

- float x

  X coordinate of the keypoint relative to source image - range <0,1>
- float y

  Y coordinate of the keypoint relative to source image - range <0,1>
- float confidence

  Confidence of the keypoint - range <0,1>

8.28.1 Detailed Description

Palm keypoint struct.
Definition at line 99 of file sfe_core_detection.h.

8.28.2 Field Documentation

8.28.2.1 confidence

`float SFEPalmKeypoint::confidence`
Confidence of the keypoint - range <0,1>
Definition at line 105 of file sfe_core_detection.h.

8.28.2.2 x

`float SFEPalmKeypoint::x`
X coordinate of the keypoint relative to source image - range <0,1>
Definition at line 101 of file sfe_core_detection.h.

8.28.2.3 y

`float SFEPalmKeypoint::y`
Y coordinate of the keypoint relative to source image - range <0,1>
Definition at line 103 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.29 SFEPalmLandmarks Struct Reference

Palm landmarks (keypoints, hand position, orientation, confidence)
`#include <sfe_core_detection.h>`

Data Fields

- SFEPalmKeypoint keypoints [SFE_PALM_KEYPOINT_COUNT]

  Palm keypoints.
- SFEHandPosition hand_position

  Detected hand position.
- SFEHandOrientation hand_orientation

  Detected palm orientation.
- float confidence

  Confidence score - range <0,1>

8.29.1 Detailed Description

Palm landmarks (keypoints, hand position, orientation, confidence)

Examples

example_palm_attributes.cpp, and example_palm_identify.cpp.

Definition at line 126 of file sfe_core_detection.h.

8.29.2 Field Documentation

8.29.2.1 confidence

`float SFEPalmLandmarks::confidence`
Confidence score - range <0,1>
Definition at line 134 of file sfe_core_detection.h.

8.29.2.2 hand_orientation

`SFEHandOrientation SFEPalmLandmarks::hand_orientation`
Detected palm orientation.

Examples

example_palm_attributes.cpp.

Definition at line 132 of file sfe_core_detection.h.

8.29.2.3 hand_position

`SFEHandPosition SFEPalmLandmarks::hand_position`
Detected hand position.

Examples

example_palm_attributes.cpp.

Definition at line 130 of file sfe_core_detection.h.

8.29.2.4 keypoints

`SFEPalmKeypoint SFEPalmLandmarks::keypoints[SFE_PALM_KEYPOINT_COUNT]`
Palm keypoints.
Definition at line 128 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.30 SFEPalmTemplate Struct Reference

Palm template struct.
`#include <sfe_palm.h>`

Data Fields

- uint8_t data [SFE_PALM_TEMPLATE_SIZE]

8.30.1 Detailed Description

Palm template struct.

Examples

example_palm_identify.cpp.

Definition at line 22 of file sfe_palm.h.

8.30.2 Field Documentation

8.30.2.1 data

`uint8_t SFEPalmTemplate::data[SFE_PALM_TEMPLATE_SIZE]`
Definition at line 23 of file sfe_palm.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_palm.h

8.31 SFEPalmTemplateVersion Struct Reference

Palm template version struct.
`#include <sfe_palm.h>`

Data Fields

- uint8_t major
  Major template version.
- uint8_t minor
  Minor template version.
- uint8_t patch
  Patch template version.

8.31.1 Detailed Description

Palm template version struct.

Examples

example_palm_identify.cpp.

Definition at line 27 of file sfe_palm.h.

8.31.2 Field Documentation

8.31.2.1 major

`uint8_t SFEPalmTemplateVersion::major`
Major template version.

Examples

example_palm_identify.cpp.

Definition at line 29 of file sfe_palm.h.

8.31.2.2 minor

`uint8_t SFEPalmTemplateVersion::minor`
Minor template version.

Examples

example_palm_identify.cpp.

Definition at line 31 of file sfe_palm.h.

8.31.2.3 patch

`uint8_t SFEPalmTemplateVersion::patch`
Patch template version.

Examples

example_palm_identify.cpp.

Definition at line 33 of file sfe_palm.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_palm.h

8.32 SFEPersonDetectionData Struct Reference

Person detection data struct (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- char _reserved

  Reserved field to ensure C/C++ compatibility.

8.32.1 Detailed Description

Person detection data struct (type-specific fields only)
Definition at line 259 of file sfe_core_detection.h.

8.32.2 Field Documentation

8.32.2.1 _reserved

`char SFEPersonDetectionData::_reserved`
Reserved field to ensure C/C++ compatibility.
Definition at line 261 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.33 SFESolverParameter Struct Reference

Solver parameter used to configure solvers.
`#include <sfe_core.h>`

Data Fields

- const char * name

  parameter name
- const char * value

  parameter value

8.33.1 Detailed Description

Solver parameter used to configure solvers.
Definition at line 105 of file sfe_core.h.

8.33.2 Field Documentation

8.33.2.1 name

```
const char* SFESolverParameter::name
```

parameter name
Definition at line 107 of file sfe_core.h.

8.33.2.2 value

```
const char* SFESolverParameter::value
```

parameter value
Definition at line 109 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.34 SFETattooDetectionData Struct Reference

Tattoo detection data struct (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- char _reserved

  Reserved field to ensure C/C++ compatibility.

8.34.1 Detailed Description

Tattoo detection data struct (type-specific fields only)
Definition at line 251 of file sfe_core_detection.h.

8.34.2 Field Documentation

8.34.2.1 _reserved

```
char SFETattooDetectionData::_reserved
```

Reserved field to ensure C/C++ compatibility.
Definition at line 253 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.35 SFETattooTemplate Struct Reference

Tattoo template struct.
`#include <sfe_tattoo.h>`

Data Fields

- uint8_t data [SFE_TATTOO_TEMPLATE_SIZE]

8.35.1 Detailed Description

Tattoo template struct.
Definition at line 22 of file sfe_tattoo.h.

8.35.2 Field Documentation

8.35.2.1 data

`uint8_t SFETattooTemplate::data[SFE_TATTOO_TEMPLATE_SIZE]`
Definition at line 23 of file sfe_tattoo.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_tattoo.h

8.36 SFETattooTemplateVersion Struct Reference

Tattoo template version struct.
`#include <sfe_tattoo.h>`

Data Fields

- uint32_t version

  Version.

8.36.1 Detailed Description

Tattoo template version struct.
Definition at line 27 of file sfe_tattoo.h.

8.36.2 Field Documentation

8.36.2.1 version

`uint32_t SFETattooTemplateVersion::version`
Version.
Definition at line 29 of file sfe_tattoo.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_tattoo.h

8.37 SFETemplateIdentificationCandidate Struct Reference

Candidate for identification by template.
`#include <sfe_core.h>`

Data Fields

- float score

  matching score
- size_t index

  index of the template from the input gallery

8.37.1 Detailed Description

Candidate for identification by template.
Definition at line 119 of file sfe_core.h.

8.37.2 Field Documentation

8.37.2.1 index

`size_t SFETemplateIdentificationCandidate::index`
index of the template from the input gallery
Definition at line 123 of file sfe_core.h.

8.37.2.2 score

`float SFETemplateIdentificationCandidate::score`
matching score
Definition at line 121 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.38 SFETracked Struct Reference

Tracked entity struct.
`#include <sfe_core.h>`

Data Fields

- SFEDetection detection
  Tracked detection.
- uint64_t id
  Tracked ID.
- SFEEntity uuid
  UUID.
- enum SFETrackedState state
  Tracking state.

8.38.1 Detailed Description

Tracked entity struct.

Examples

example_face_track.cpp.

Definition at line 297 of file sfe_core.h.

8.38.2 Field Documentation

8.38.2.1 detection

`SFEDetection SFETracked::detection`
Tracked detection.
Definition at line 299 of file sfe_core.h.

8.38.2.2 id

`uint64_t SFETracked::id`
Tracked ID.

Examples

example_face_track.cpp.

Definition at line 301 of file sfe_core.h.

8.38.2.3 state

`enum SFETrackedState SFETracked::state`
Tracking state.

Examples

example_face_track.cpp.

Definition at line 305 of file sfe_core.h.

8.38.2.4 uuid

`SFEEntity SFETracked::uuid`
UUID.

Examples

example_face_track.cpp.

Definition at line 303 of file sfe_core.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core.h

8.39 SFEVisualCodeDetectionData Struct Reference

Visual code detection data struct (type-specific fields only)
`#include <sfe_core_detection.h>`

Data Fields

- SFEVisualCodeCategory category
  Visual code category.

8.39.1 Detailed Description

Visual code detection data struct (type-specific fields only)
Definition at line 284 of file sfe_core_detection.h.

8.39.2 Field Documentation

8.39.2.1 category

`SFEVisualCodeCategory SFEVisualCodeDetectionData::category`
Visual code category.
Definition at line 286 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

8.40 SFEVisualCodeKeypoint Struct Reference

Visual code keypoint struct.
`#include <sfe_core_detection.h>`

Data Fields

- float x
  X coordinate of the keypoint relative to source image - range <0,1>
- float y
  Y coordinate of the keypoint relative to source image - range <0,1>

8.40.1 Detailed Description

Visual code keypoint struct.
Definition at line 273 of file sfe_core_detection.h.

8.40.2 Field Documentation

8.40.2.1 x

`float SFEVisualCodeKeypoint::x`
X coordinate of the keypoint relative to source image - range <0,1>
Definition at line 275 of file sfe_core_detection.h.

8.40.2.2 y

`float SFEVisualCodeKeypoint::y`
Y coordinate of the keypoint relative to source image - range <0,1>
Definition at line 277 of file sfe_core_detection.h.
The documentation for this struct was generated from the following file:

- D:/glab/1512fd1724a/1/include/sfe_core_detection.h

Chapter 9

File Documentation

9.1 D:/glab/1512fd1724a/1/changelog.md File Reference

9.2 sfe_features.md File Reference

9.3 sfe_solvers.md File Reference

9.4 upgrade_3_to_4.md File Reference

9.5 D:/glab/1512fd1724a/1/example/example_face_demographic_↩ attributes.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_face.h¨
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <iomanip>
#include <iostream>
#include <sstream>
#include <unordered_map>
#include <vector>
```

Functions

- void printHelp ()
- int main (int argc, char ∗argv[ ])

Variables

- std::string image_probe = ¨assets/face/images/obiwan0.png¨
- std::string solver_face_detect = SOLVER_FACE_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
- std::string solver_face_demographic = SOLVER_FACE_DEMOGRAPHIC_ATTRIBUTES
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- size_t face_size_min = 28

  Required size of the face to be detected.
- size_t face_size_max = 170
- float detection_threshold = 0.1f

9.5.1 Function Documentation

9.5.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

[Image Load]
[Image Load]
[Face Detect]
[Face Detect]
[Face Landmarks]
[Face Landmarks]
[Face Demographic Attributes]
[Face Demographic Attributes]
Definition at line 46 of file example_face_demographic_attributes.cpp.

```
00046                                {
00047
00048   { // STAGE 0: Parse command line arguments
00049     std::unordered_map<std::string, std::string> args;
00050
00051     for (int i = 1; i < argc; ++i) {
00052       std::string arg = argv[i];
00053       if (arg[0] == '-') {
00054         if (arg == "-h") {
00055           printHelp();
00056           return 0;
00057         }
00058         if (i + 1 < argc && argv[i + 1][0] != '-') {
00059           args[arg] = argv[++i];
00060         } else {
00061           std::cerr << "Option " << arg << " requires a value." << std::endl;
00062           return 1;
00063         }
00064       } else {
00065         std::cerr << "Unknown option: " << arg << std::endl;
00066         return 1;
00067       }
00068     }
00069
00070     if (args.count("-p"))
00071       image_probe = args["-p"];
00072     if (args.count("-d"))
00073       solver_face_detect = args["-d"];
00074     if (args.count("-l"))
00075       solver_face_landmarks = args["-l"];
00076     if (args.count("-g"))
00077       solver_face_demographic = args["-g"];
00078     if (args.count("-t"))
00079       detection_threshold = std::stof(args["-t"]);
00080     if (args.count("-m"))
00081       face_size_min = std::stoi(args["-m"]);
00082     if (args.count("-x"))
00083       face_size_max = std::stoi(args["-x"]);
00084
00085     utils::printFormatted("PARAMETERS");
00086     std::cout << "Probe image: " << image_probe << std::endl;
00087     std::cout << "Face detection solver: " << solver_face_detect << std::endl;
00088     std::cout << "Face landmarks solver: " << solver_face_landmarks << std::endl;
00089     std::cout << "Face demographic attributes solver: " << solver_face_demographic
00090               << std::endl;
00091     std::cout << "Min face size [px]: " << face_size_min << std::endl;
00092     std::cout << "Max face size [px]: " << face_size_max << std::endl;
00093     std::cout << "Detection threshold: " << detection_threshold << std::endl;
00094   }
00095
00096   utils::printToolkitInfo();
00097
00098   SFEError error{};
00099
00100   SFESolver detector_solver{};
00101   DEFER(sfeSolverFree(detector_solver));
00102   SFESolver landmarks_solver{};
00103   DEFER(sfeSolverFree(landmarks_solver));
00104   SFESolver demographic_solver{};
00105   DEFER(sfeSolverFree(demographic_solver));
00106
00107   { // STAGE 1: loading solvers
00108     utils::printFormatted("LOADING solvers");
00109
00110     error = sfeSolverCreate(
```

```
00111         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00112         SOLVER_PARAMETERS.size(), &detector_solver);
00113     utils::checkError(error);
00114
00115     error = sfeSolverCreate(
00116         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00117         SOLVER_PARAMETERS.size(), &landmarks_solver);
00118     utils::checkError(error);
00119
00120     error = sfeSolverCreate(
00121         solver_face_demographic.c_str(), SOLVER_PARAMETERS.data(),
00122         SOLVER_PARAMETERS.size(), &demographic_solver);
00123     utils::checkError(error);
00124   }
00125
00126   SFEImage image{};
00127   DEFER(sfeImageFree(image));
00128
00129   { // STAGE 2: Load image
00130     utils::printFormatted("FACE DETECTION");
00132     // Load image data from file
00133     auto image_data = utils::readFile(image_probe);
00134
00135     // Decode image from data
00136     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00137     utils::checkError(error);
00139   }
00140
00142   SFEDetection detected_face = {};
00143   { // STAGE 3: Detect face in the image
00144     size_t detection_count = 1;
00145     error = sfeDetect(detector_solver, image, detection_threshold,
00146                       &detected_face, &detection_count);
00147     utils::checkError(error);
00148
00149     if (detection_count == 0) {
00150       std::cout « "No face detected in the probe image." « std::endl;
00151       return 0;
00152     } else {
00153       std::cout « "Found " « detection_count
00154                 « " face(s) in the probe image. Using the face with highest "
00155                    "confidence."
00156                 « std::endl;
00157     }
00158   }
00160
00162   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00163   { // STAGE 4: Get face landmarks
00164     error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00165                              landmarks.data());
00166     utils::checkError(error);
00167   }
00169
00171   SFEFaceDemographicAttributes attributes{};
00172   { // STAGE 5: Demographic attributes
00173     utils::printFormatted("FACE DEMOGRAPHIC ATTRIBUTES");
00174     error = sfeFaceDemographicAttributes(demographic_solver, image,
00175                                          landmarks.data(), &attributes);
00176     utils::checkError(error);
00177   }
00179
00180   { // STAGE 6: Print demographic attributes
00181     std::cout « "Demographic attributes:" « std::endl;
00182     std::cout « "  Age: " « std::fixed « std::setprecision(1) « attributes.age
00183               « " years" « std::endl;
00184     std::cout « "  Gender score: " « std::setprecision(2) « attributes.gender
00185               « " (" « (attributes.gender > 0.0f ? "female" : "male") « ")"
00186               « std::endl;
00187   }
00188
00189   utils::printFormatted("FINISHED");
00190 }
```

9.5.1.2 printHelp()

```
void printHelp ( )
```

Definition at line 33 of file example_face_demographic_attributes.cpp.

```
00033                 {
00034   std::cout « "Help: Usage of the program." « std::endl;
00035   std::cout « "Options:" « std::endl;
00036   std::cout « "-h: Display help." « std::endl;
00037   std::cout « "-p: Probe image file." « std::endl;
00038   std::cout « "-d: Path to detector solver." « std::endl;
00039   std::cout « "-l: Path to landmarks solver." « std::endl;
00040   std::cout « "-g: Path to demographic attributes solver." « std::endl;
```

```
00041   std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
00042   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00043   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00044 }
```

9.5.2 Variable Documentation

9.5.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 31 of file example_face_demographic_attributes.cpp.

9.5.2.2 face_size_max

```
size_t face_size_max = 170
```

Definition at line 29 of file example_face_demographic_attributes.cpp.

9.5.2.3 face_size_min

```
size_t face_size_min = 28
```

Required size of the face to be detected.
Definition at line 28 of file example_face_demographic_attributes.cpp.

9.5.2.4 image_probe

```
std::string image_probe = ¨assets/face/images/obiwan0.png¨
```

Definition at line 18 of file example_face_demographic_attributes.cpp.

9.5.2.5 solver_face_demographic

```
std::string solver_face_demographic = SOLVER_FACE_DEMOGRAPHIC_ATTRIBUTES
```

Examples

example_face_demographic_attributes.cpp.

Definition at line 23 of file example_face_demographic_attributes.cpp.

9.5.2.6 solver_face_detect

```
std::string solver_face_detect = SOLVER_FACE_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.
Definition at line 21 of file example_face_demographic_attributes.cpp.

9.5.2.7 solver_face_landmarks

```
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
```

Definition at line 22 of file example_face_demographic_attributes.cpp.

9.5.2.8 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Definition at line 25 of file example_face_demographic_attributes.cpp.

9.6 example_face_demographic_attributes.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_face.h"
00007
00008 #include "annotate.hpp"
00009 #include "solvers.h"
```

```cpp
00010 #include "utils.hpp"
00011
00012 #include <iomanip>
00013 #include <iostream>
00014 #include <sstream>
00015 #include <unordered_map>
00016 #include <vector>
00017
00018 std::string image_probe = "assets/face/images/obiwan0.png";
00019
00021 std::string solver_face_detect = SOLVER_FACE_DETECT;
00022 std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
00023 std::string solver_face_demographic = SOLVER_FACE_DEMOGRAPHIC_ATTRIBUTES;
00024
00025 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00026
00028 size_t face_size_min = 28;
00029 size_t face_size_max = 170;
00030
00031 float detection_threshold = 0.1f;
00032
00033 void printHelp() {
00034   std::cout « "Help: Usage of the program." « std::endl;
00035   std::cout « "Options:" « std::endl;
00036   std::cout « "-h: Display help." « std::endl;
00037   std::cout « "-p: Probe image file." « std::endl;
00038   std::cout « "-d: Path to detector solver." « std::endl;
00039   std::cout « "-l: Path to landmarks solver." « std::endl;
00040   std::cout « "-g: Path to demographic attributes solver." « std::endl;
00041   std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
00042   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00043   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00044 }
00045
00046 int main(int argc, char *argv[]) {
00047
00048   { // STAGE 0: Parse command line arguments
00049     std::unordered_map<std::string, std::string> args;
00050
00051     for (int i = 1; i < argc; ++i) {
00052       std::string arg = argv[i];
00053       if (arg[0] == '-') {
00054         if (arg == "-h") {
00055           printHelp();
00056           return 0;
00057         }
00058         if (i + 1 < argc && argv[i + 1][0] != '-') {
00059           args[arg] = argv[++i];
00060         } else {
00061           std::cerr « "Option " « arg « " requires a value." « std::endl;
00062           return 1;
00063         }
00064       } else {
00065         std::cerr « "Unknown option: " « arg « std::endl;
00066         return 1;
00067       }
00068     }
00069
00070     if (args.count("-p"))
00071       image_probe = args["-p"];
00072     if (args.count("-d"))
00073       solver_face_detect = args["-d"];
00074     if (args.count("-l"))
00075       solver_face_landmarks = args["-l"];
00076     if (args.count("-g"))
00077       solver_face_demographic = args["-g"];
00078     if (args.count("-t"))
00079       detection_threshold = std::stof(args["-t"]);
00080     if (args.count("-m"))
00081       face_size_min = std::stoi(args["-m"]);
00082     if (args.count("-x"))
00083       face_size_max = std::stoi(args["-x"]);
00084
00085     utils::printFormatted("PARAMETERS");
00086     std::cout « "Probe image: " « image_probe « std::endl;
00087     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00088     std::cout « "Face landmarks solver: " « solver_face_landmarks « std::endl;
00089     std::cout « "Face demographic attributes solver: " « solver_face_demographic
00090               « std::endl;
00091     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00092     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00093     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00094   }
00095
00096   utils::printToolkitInfo();
00097
00098   SFEError error{};
```

```
00099
00100   SFESolver detector_solver{};
00101   DEFER(sfeSolverFree(detector_solver));
00102   SFESolver landmarks_solver{};
00103   DEFER(sfeSolverFree(landmarks_solver));
00104   SFESolver demographic_solver{};
00105   DEFER(sfeSolverFree(demographic_solver));
00106
00107   { // STAGE 1: loading solvers
00108     utils::printFormatted("LOADING solvers");
00109
00110     error = sfeSolverCreate(
00111         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00112         SOLVER_PARAMETERS.size(), &detector_solver);
00113     utils::checkError(error);
00114
00115     error = sfeSolverCreate(
00116         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00117         SOLVER_PARAMETERS.size(), &landmarks_solver);
00118     utils::checkError(error);
00119
00120     error = sfeSolverCreate(
00121         solver_face_demographic.c_str(), SOLVER_PARAMETERS.data(),
00122         SOLVER_PARAMETERS.size(), &demographic_solver);
00123     utils::checkError(error);
00124   }
00125
00126   SFEImage image{};
00127   DEFER(sfeImageFree(image));
00128
00129   { // STAGE 2: Load image
00130     utils::printFormatted("FACE DETECTION");
00132     // Load image data from file
00133     auto image_data = utils::readFile(image_probe);
00134
00135     // Decode image from data
00136     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00137     utils::checkError(error);
00139   }
00140
00142   SFEDetection detected_face = {};
00143   { // STAGE 3: Detect face in the image
00144     size_t detection_count = 1;
00145     error = sfeDetect(detector_solver, image, detection_threshold,
00146                       &detected_face, &detection_count);
00147     utils::checkError(error);
00148
00149     if (detection_count == 0) {
00150       std::cout « "No face detected in the probe image." « std::endl;
00151       return 0;
00152     } else {
00153       std::cout « "Found " « detection_count
00154                 « " face(s) in the probe image. Using the face with highest "
00155                    "confidence."
00156                 « std::endl;
00157     }
00158   }
00160
00162   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00163   { // STAGE 4: Get face landmarks
00164     error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00165                              landmarks.data());
00166     utils::checkError(error);
00167   }
00169
00171   SFEFaceDemographicAttributes attributes{};
00172   { // STAGE 5: Demographic attributes
00173     utils::printFormatted("FACE DEMOGRAPHIC ATTRIBUTES");
00174     error = sfeFaceDemographicAttributes(demographic_solver, image,
00175                                          landmarks.data(), &attributes);
00176     utils::checkError(error);
00177   }
00179
00180   { // STAGE 6: Print demographic attributes
00181     std::cout « "Demographic attributes:" « std::endl;
00182     std::cout « "  Age: " « std::fixed « std::setprecision(1) « attributes.age
00183               « " years" « std::endl;
00184     std::cout « "  Gender score: " « std::setprecision(2) « attributes.gender
00185               « " (" « (attributes.gender > 0.0f ? "female" : "male") « ")"
00186               « std::endl;
00187   }
00188
00189   utils::printFormatted("FINISHED");
00190 }
00191
00192
```

9.7 D:/glab/1512fd1724a/1/example/example_face_identify.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_face.h¨
#include <regex>
#include <vector>
#include <iomanip>
#include <sstream>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- void getRecommendedImageSize (const std::string &solver_face_detect, SFEImage &image, const size_t face_size_min, const size_t face_size_max, size_t &recommended_width, size_t &recommended_height)

  Get recommended image size for face detection.
- void printHelp ()
- void detectFace (SFEImage &image, SFESolver &detector_solver, SFEDetection &detected_face)
- int main (int argc, char ∗argv[ ])

  Main fuction is used to parse command line arguments.

Variables

- std::string image_gallery = ¨./assets/face/entities/¨
- std::string image_probe = ¨assets/face/images/obiwan0.png¨
- std::string solver_face_detect = SOLVER_FACE_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
- std::string solver_face_template = SOLVER_FACE_EXTRACTION
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- size_t face_size_min = 28

  Required size of the face to be detected.
- size_t face_size_max = 170
- float detection_threshold = 0.1f
- float identification_threshold = 0.6f

9.7.1 Function Documentation

9.7.1.1 detectFace()

```
void detectFace (
            SFEImage & image,
            SFESolver & detector_solver,
            SFEDetection & detected_face )
```

Examples

example_face_identify.cpp.

Definition at line 93 of file example_face_identify.cpp.

```
00094                                                {
00095
00096   SFEImage resized_image{};
00097   DEFER(sfeImageFree(resized_image));
00098   SFEError error{};
00099   { // STAGE 1 Load image
```

```
00100     size_t recommended_width{};
00101     size_t recommended_height{};
00102
00103     // Calculate optimal input image size for face detection. Image will be
00104     // resized to recommended size for optimal performance.
00105     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00106                             face_size_max, recommended_width,
00107                             recommended_height);
00108
00109     // Resize image to recommended size
00110     // NOTE: This function will resize the image without preserving the aspect
00111     // ratio of the image.
00112
00113     error = sfeImageResize(image, recommended_width, recommended_height,
00114                            &resized_image);
00115     utils::checkError(error);
00116   }
00117
00118   size_t detection_count = 1;
00119   { // STAGE 2: Detect face in the image
00120
00121     // Detect face in the image
00122     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00123                               &detected_face, &detection_count);
00124     utils::checkError(error);
00125
00126     if (detection_count == 0) {
00127       std::ostringstream oss;
00128       oss « "Error: No face detected in the image ";
00129       throw std::runtime_error(oss.str());
00130
00131     } else if (detection_count > 1) {
00132       std::ostringstream oss;
00133       oss « "Error: Found " « detection_count
00134           « " faces. Please provide an image with only one face.";
00135
00136       throw std::runtime_error(oss.str());
00137     }
00138
00139     std::cout « "Detected face in the image." « std::endl;
00140   }
00141 }
```

9.7.1.2 getRecommendedImageSize()

```
void getRecommendedImageSize (
            const std::string & solver_face_detect,
            SFEImage & image,
            const size_t face_size_min,
            const size_t face_size_max,
            size_t & recommended_width,
            size_t & recommended_height )
```

Get recommended image size for face detection.

Parameters

solver_face_detect	name of the face detection solver
face_size_min	Desired minimal size of detected face
face_size_max	Desired maximal size of detected face
recommended_width	Recommended image width
recommended_height	Recommended image height

Examples

example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

Definition at line 42 of file example_face_identify.cpp.

```
00046                                                 {
00047
00048   // If the face detection solver requires static input (filename contains width
00049   // and height), we need to prepare the input image accordingly Typically the
00050   // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
00051   // This can be hardcoded into the application, or we can parse the width and
00052   // height from the solver filename
```

```
00053
00054   // Try to use regex capture to extract width and height from solver name
00055   std::smatch width_height;
00056   if (std::regex_match(solver_face_detect, width_height,
00057                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00058     recommended_width = std::stoi(width_height[1]);
00059     recommended_height = std::stoi(width_height[2]);
00060   } else {
00061     // Solvers without width and height in name can accept dynamic input
00062     // image size. For best results we recommend to scale the input image to
00063     // a resolution recommended by the sfeFaceDetectInputSize function
00064
00065     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00066     // for given min_face_size/max_face_size
00067     SFEDetectionInputSize input_size{};
00068     SFEError error = sfeFaceDetectInputSize(
00069         image,
00070         SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00071         face_size_min, face_size_max, &input_size);
00072     utils::checkError(error);
00073     recommended_width = input_size.width;
00074     recommended_height = input_size.height;
00075   };
00076 }
```

9.7.1.3 main()

```
int main (
            int argc,
            char * argv[] )
```

Main fuction is used to parse command line arguments.
[Face Template Extract]
[Face Template Extract]
[Face Template Version]
[Face Template Version]
[Face Template Export Import]
[Face Template Export Import]
[Face Template Identify]
[Face Template Identify]
[Face Template Match]
[Face Template Match]
[Face Crop]
[Face Crop]
Definition at line 144 of file example_face_identify.cpp.

```
00144                                   {
00145
00146   { // STAGE 0: Parse command line arguments
00147     // Map to store argument values
00148     std::unordered_map<std::string, std::string> args;
00149
00150     // Parse command line arguments
00151     for (int i = 1; i < argc; ++i) {
00152       std::string arg = argv[i];
00153       if (arg[0] == '-') {
00154         if (arg == "-h") {
00155           printHelp();
00156           return 0;
00157         }
00158         // Check if there's a next argument and it isn't another option
00159         if (i + 1 < argc && argv[i + 1][0] != '-') {
00160           args[arg] = argv[++i];
00161         } else {
00162           std::cerr << "Option " << arg << " requires a value." << std::endl;
00163           return 1;
00164         }
00165       } else {
00166         std::cerr << "Unknown option: " << arg << std::endl;
00167         return 1;
00168       }
00169     }
00170
00171     // Assign values to variables based on parsed arguments
00172     if (args.count("-p"))
00173       image_probe = args["-p"];
00174     if (args.count("-g"))
00175       image_gallery = args["-g"];
00176     if (args.count("-d"))
```

```
00177       solver_face_detect = args["-d"];
00178     if (args.count("-l"))
00179       solver_face_landmarks = args["-l"];
00180     if (args.count("-e"))
00181       solver_face_template = args["-e"];
00182     if (args.count("-t"))
00183       detection_threshold = std::stof(args["-t"]);
00184     if (args.count("-i"))
00185       identification_threshold = std::stof(args["-i"]);
00186     if (args.count("-m"))
00187       face_size_min = std::stoi(args["-m"]);
00188     if (args.count("-x"))
00189       face_size_max = std::stoi(args["-x"]);
00190
00191     utils::printFormatted("EXAMPLE PARAMETERS");
00192     // Print resource names
00193     std::cout « "Probe image: " « image_probe « std::endl;
00194     std::cout « "Gallery image: " « image_gallery « std::endl;
00195
00196     // Print solver names
00197     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00198     std::cout « "Face landmarks solver: " « solver_face_landmarks
00199               « std::endl;
00200     std::cout « "Face template solver: " « solver_face_template « std::endl;
00201
00202     // Print other options
00203     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00204     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00205     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00206     std::cout « "Identification threshold: " « identification_threshold
00207               « std::endl;
00208   }
00209
00210   utils::printToolkitInfo();
00211
00212   SFEError error{};
00213
00214   SFESolver detector_solver{};
00215   DEFER(sfeSolverFree(detector_solver));
00216   SFESolver landmarks_solver{};
00217   DEFER(sfeSolverFree(landmarks_solver));
00218   SFESolver template_solver{};
00219   DEFER(sfeSolverFree(template_solver));
00220   { // STAGE 1.1: loading solvers
00221     utils::printFormatted("LOADING SOLVERS");
00222     // Initialize face detection solver
00223     error = sfeSolverCreate(
00224         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00225         SOLVER_PARAMETERS.size(), &detector_solver);
00226     utils::checkError(error);
00227
00228     // Initialize landmarks detection solver
00229     error = sfeSolverCreate(
00230         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00231         SOLVER_PARAMETERS.size(), &landmarks_solver);
00232     utils::checkError(error);
00233
00234     // Initialize template extraction solver
00235     error = sfeSolverCreate(
00236         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00237         SOLVER_PARAMETERS.size(), &template_solver);
00238     utils::checkError(error);
00239   }
00240
00241   SFEFaceTemplate probe_face_template;
00242   { // STAGE 1: Extract probe face template
00243     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00244
00245     SFEImage image{};
00246     DEFER(sfeImageFree(image));
00247     { // STAGE 1.1: Load image
00248       // Load image data from file
00249       auto image_data = utils::readFile(image_probe);
00250
00251       // Decode image from data
00252       error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00253       utils::checkError(error);
00254     }
00255
00256     SFEDetection detected_face = {};
00257     { // STAGE 1.2: Detect face in the image
00258       // Detect a face in the image
00259       detectFace(image, detector_solver, detected_face);
00260     }
00261
00262     std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00263     { // STAGE 1.3: Get face landmarks
```

```
00264       // Use landmarks detection solver to get relevant landmarks for
00265       // the detected face
00266       error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00267                                landmarks.data());
00268       utils::checkError(error);
00269     }
00270
00272     { // STAGE 1.4: Extract probe face template
00273       // Use template extraction solver to create new face
00274       // template
00275       error = sfeFaceTemplateExtract(template_solver, image, &detected_face,
00276                                      landmarks.data(), &probe_face_template);
00277       utils::checkError(error);
00278     }
00280
00281     { // STAGE 1.5: (optional) Print template attributes
00282       utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00283
00285       SFEFaceTemplateVersion version = {};
00286       error = sfeFaceTemplateVersion(&probe_face_template, &version);
00287       utils::checkError(error);
00288
00289       auto version_minor = (int)version.version_minor - (int)'0';
00290       std::cout « "Version of the probe template: " « version.version_major
00291                 « '.' « version_minor « std::endl;
00293     }
00294
00296     { // STAGE 1.6: (optional) Export and re-import template
00297       std::vector<uint8_t> buf(1024);
00298       size_t size = buf.size();
00299       error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
00300       if (error && size > buf.size()) {
00301         buf.resize(size);
00302         size = buf.size();
00303         error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
00304       }
00305       utils::checkError(error);
00306
00307       SFEFaceTemplate imported_template = {};
00308       error = sfeFaceTemplateImport(buf.data(), size, &imported_template);
00309       utils::checkError(error);
00310
00311       float roundtrip_score = 0.f;
00312       error = sfeFaceTemplateMatch(&probe_face_template, &imported_template,
00313                                    &roundtrip_score);
00314       utils::checkError(error);
00315       std::cout « "Export/import round-trip match score: " « roundtrip_score
00316                 « std::endl;
00317     }
00319   }
00320
00321   std::vector<std::string> image_paths{};
00322   { // STAGE 2: Get gallery image paths
00323     // Get folder names from the face image gallery
00324     auto folder_names = utils::getFiles(image_gallery);
00325
00326     for (auto folder_name : folder_names) {
00327       auto gallery_folder = image_gallery + folder_name + "/";
00328
00329       // Get image names from folder
00330       auto image_names = utils::getFiles(gallery_folder);
00331
00332       // Extract template for each image in the folder
00333       for (auto image_name : image_names) {
00334
00335         auto image_path = gallery_folder + image_name;
00336
00337         image_paths.push_back(image_path);
00338       }
00339     }
00340   }
00341
00342   std::vector<SFEFaceTemplate> gallery_face_templates(image_paths.size());
00343   std::vector<SFEDetection> detected_faces(image_paths.size());
00344   std::vector<std::array<SFEFaceLandmarks, SFE_FACE_LANDMARK_COUNT» landmarks(
00345       image_paths.size());
00346   { // STAGE 2: Load gallery image and extract face templates for each image in
00347     // the gallery
00348     utils::printFormatted("GALLERY TEMPLATE EXTRACTION");
00349
00350     for (size_t i = 0; i < image_paths.size(); i++) {
00351       std::cout « "Processing image #" « i « ", path: " « image_paths[i]
00352                 « std::endl;
00353       SFEImage image{};
00354       DEFER(sfeImageFree(image));
00355       { // STAGE 2.1: Load image
00356         // Load image data from file
```

```
00357         auto image_data = utils::readFile(image_paths[i]);
00358
00359         // Decode image from data
00360         error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00361         utils::checkError(error);
00362       }
00363
00364       { // STAGE 2.2: Detect face in the image
00365         // Detect a face in the image
00366         detectFace(image, detector_solver, detected_faces[i]);
00367       }
00368
00369       { // STAGE 2.3: Get face landmarks
00370         // Use landmarks detection solver to get relevant landmarks for
00371         // the detected face
00372         error = sfeFaceLandmarks(landmarks_solver, image, &detected_faces[i],
00373                                  landmarks[i].data());
00374         utils::checkError(error);
00375       }
00376
00377       { // STAGE 2.3: Extract face template
00378         // Use template extraction solver to get face_template solver
00379         error = sfeFaceTemplateExtract(
00380             template_solver, image, &detected_faces[i], landmarks[i].data(),
00381             &gallery_face_templates[i]);
00382         utils::checkError(error);
00383         std::cout « "Extracted face template." « std::endl;
00384       }
00385       std::cout « std::endl;
00386     }
00387   }
00388
00390   size_t candidate_count = 1;
00391   int best_candidate_index = -1;
00392   std::vector<SFETemplateIdentificationCandidate> identification_results(
00393       image_paths.size());
00394   { // STAGE 3: Identification
00395     utils::printFormatted("1:N IDENTIFICATION");
00396
00397     error = sfeFaceTemplateIdentify(
00398         &probe_face_template, gallery_face_templates.data(),
00399         gallery_face_templates.size(), identification_threshold,
00400         identification_results.data(), &candidate_count, 4);
00401     utils::checkError(error);
00402
00403     // Resize the results vector to the actual number of candidates found, in
00404     // the case there is less than candidate_count
00405     identification_results.resize(candidate_count);
00406
00407     std::cout « "Found " « identification_results.size()
00408               « " candidates above identification threshold "
00409               « identification_threshold « std::endl;
00410     for (auto &result : identification_results)
00411       std::cout « "Template index: #" « result.index
00412                 « ", score: " « result.score « std::endl;
00413
00414     // Since it's sorted vector by score, the best candidate is the first one
00415     best_candidate_index = identification_results[0].index;
00416   }
00418
00420   { // STAGE 4: (optional) 1:1 matching with top candidate to showcase 1:1
00421     // matching
00422     utils::printFormatted("1:1 MATCHING WITH TOP CANDIDATE");
00423
00424     if (identification_results.size() == 0) {
00425       std::cout « "No candidates found above identification threshold "
00426                 « identification_threshold « std::endl;
00427       return 0;
00428     }
00429
00430     auto best_candidate_template = gallery_face_templates[best_candidate_index];
00431
00432     float match_confidence;
00433     error = sfeFaceTemplateMatch(&probe_face_template, &best_candidate_template,
00434                                  &match_confidence);
00435     utils::checkError(error);
00436
00437     std::cout « "Matching score of probe template with template index #"
00438               « best_candidate_index « ", score: " « match_confidence
00439               « std::endl;
00440   }
00442
00443   { // STAGE: 5 Optional, get face crop of best face and its bounding box and
00444     // save it to file
00445     utils::printFormatted("SAVE FACE CROP AND ANNOTATED IMAGE");
00446
00447     auto best_face_index = identification_results[0].index;
```

```
00448
00449     SFEImage image{};
00450     DEFER(sfeImageFree(image));
00451     { // STAGE 5.1: Load image
00452       // Load image data from file
00453       auto image_data = utils::readFile(image_paths[best_face_index]);
00454
00455       // Decode image from data
00456       error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00457       utils::checkError(error);
00458     }
00459
00461     SFEFaceCrop crop_data = {};
00462     DEFER(sfeImageFree(crop_data.crop_image));
00463     { // STAGE 5.2: Get face crop
00464       //  Defines the size of the crop as an extension of
00465       // the detection bounding box.
00466       float face_size_extension = 2.0f;
00467       SFEError error =
00468           sfeFaceCrop(image, landmarks[best_face_index].data(),
00469                       face_size_extension, (float)(face_size_max), &crop_data);
00470       utils::checkError(error);
00471
00472       std::cout « "Face crop extension: " « crop_data.face_size_extension
00473                 « ", width of cropped image: " « crop_data.crop_image.width
00474                 « "px, height of cropped image: "
00475                 « crop_data.crop_image.height « "px." « std::endl;
00476     }
00478
00479     { // STAGE 5.3: Save face crop to file
00480       auto png_file = std::vector<unsigned char>();
00481       size_t size =
00482           crop_data.crop_image.width * crop_data.crop_image.height * 3;
00483       png_file.resize(size);
00484       SFEError error2 = sfeImageEncode(crop_data.crop_image, SFE_IMAGE_FORMAT_PNG,
00485                                        png_file.data(), &size);
00486       utils::checkError(error2);
00487
00488       std::cout « "Face crop saved as crop_identified_person.png" « std::endl;
00489       utils::saveFile("crop_identified_person.png", png_file);
00490     }
00491
00492     { // STAGE 5.4: Annotate the image
00493       // Render bounding box with detection confidence and identification score
00494       std::stringstream label;
00495       label « " D:" « std::fixed « std::setprecision(2)
00496             « detected_faces[best_face_index].confidence
00497             « " M:" « identification_results[0].score;
00498
00499       auto color = annotate::RED;
00500
00501       annotate::labelBox(image, label.str(),
00502                          detected_faces[best_candidate_index].bounding_box,
00503                          annotate::WHITE, color);
00504
00505       // Render landmarks
00506       for (auto &landmark : landmarks[best_face_index]) {
00507         auto x = landmark.x * image.width;
00508         auto y = landmark.y * image.height;
00509         annotate::circle(image, x, y, 3, annotate::YELLOW);
00510       }
00511
00512       // Save annotated image
00513       size_t size = image.width * image.height * 3;
00514       auto png_file = std::vector<unsigned char>(size);
00515       error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00516       utils::checkError(error);
00517       png_file.resize(size);
00518       utils::saveFile("face_identify.png", png_file);
00519
00520       std::cout « std::endl;
00521       std::cout « "Annotated image saved as face_identify.png" « std::endl;
00522     }
00523   }
00524
00525   utils::printFormatted("FINISHED");
00526 }
```

9.7.1.4 printHelp()

```
void printHelp ( )
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp,

example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 78 of file example_face_identify.cpp.

```
00078                 {
00079   std::cout « "Help: Usage of the program." « std::endl;
00080   std::cout « "Options:" « std::endl;
00081   std::cout « "-h: Display help." « std::endl;
00082   std::cout « "-p: Probe image file." « std::endl;
00083   std::cout « "-g: Gallery folder." « std::endl;
00084   std::cout « "-d: Path to detector solver." « std::endl;
00085   std::cout « "-l: Path to landmarks solver." « std::endl;
00086   std::cout « "-e: Path to extraction solver." « std::endl;
00087   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00088   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00089   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00090   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00091 }
```

9.7.2 Variable Documentation

9.7.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 33 of file example_face_identify.cpp.

9.7.2.2 face_size_max

```
size_t face_size_max = 170
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

Definition at line 31 of file example_face_identify.cpp.

9.7.2.3 face_size_min

```
size_t face_size_min = 28
```

Required size of the face to be detected.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

Definition at line 30 of file example_face_identify.cpp.

9.7.2.4 identification_threshold

```
float identification_threshold = 0.6f
```

Examples

example_face_identify.cpp, example_face_identify_entity.cpp, example_iris_identify.cpp, and example_palm_identify.cpp.

Definition at line 34 of file example_face_identify.cpp.

9.7.2.5 image_gallery

```
std::string image_gallery = ¨./assets/face/entities/¨
```

Examples

example_face_identify.cpp.

Definition at line 18 of file example_face_identify.cpp.

9.7.2.6 image_probe

```
std::string image_probe = ¨assets/face/images/obiwan0.png¨
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 20 of file example_face_identify.cpp.

9.7.2.7 solver_face_detect

```
std::string solver_face_detect = SOLVER_FACE_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

Definition at line 23 of file example_face_identify.cpp.

9.7.2.8 solver_face_landmarks

```
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, and example_face_liveness.cpp.

Definition at line 24 of file example_face_identify.cpp.

9.7.2.9 solver_face_template

```
std::string solver_face_template = SOLVER_FACE_EXTRACTION
```

Examples

example_face_identify.cpp, and example_face_identify_entity.cpp.

Definition at line 25 of file example_face_identify.cpp.

9.7.2.10 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 27 of file example_face_identify.cpp.

9.8 example_face_identify.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_face.h"
00007 #include <regex>
00008 #include <vector>
00009
00010 #include <iomanip>
00011 #include <sstream>
00012
00013 #include "annotate.hpp"
00014 #include "solvers.h"
00015 #include "utils.hpp"
00016 #include <unordered_map>
00017
00018 std::string image_gallery = "./assets/face/entities/";
00019
00020 std::string image_probe = "assets/face/images/obiwan0.png";
00021
00023 std::string solver_face_detect = SOLVER_FACE_DETECT;
00024 std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
00025 std::string solver_face_template = SOLVER_FACE_EXTRACTION;
00026
00027 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00028
00030 size_t face_size_min = 28;
00031 size_t face_size_max = 170;
00032
00033 float detection_threshold = 0.1f;
00034 float identification_threshold = 0.6f;
00035
00042 void getRecommendedImageSize(const std::string &solver_face_detect,
00043                              SFEImage &image, const size_t face_size_min,
00044                              const size_t face_size_max,
00045                              size_t &recommended_width,
00046                              size_t &recommended_height) {
00047
00048   // If the face detection solver requires static input (filename contains width
00049   // and height), we need to prepare the input image accordingly Typically the
00050   // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
00051   // This can be hardcoded into the application, or we can parse the width and
00052   // height from the solver filename
00053
00054   // Try to use regex capture to extract width and height from solver name
00055   std::smatch width_height;
00056   if (std::regex_match(solver_face_detect, width_height,
00057                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00058     recommended_width = std::stoi(width_height[1]);
00059     recommended_height = std::stoi(width_height[2]);
00060   } else {
00061     // Solvers without width and height in name can accept dynamic input
00062     // image size. For best results we recommend to scale the input image to
00063     // a resolution recommended by the sfeFaceDetectInputSize function
00064
00065     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00066     // for given min_face_size/max_face_size
00067     SFEDetectionInputSize input_size{};
00068     SFEError error = sfeFaceDetectInputSize(
00069         image,
00070         SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00071         face_size_min, face_size_max, &input_size);
00072     utils::checkError(error);
00073     recommended_width = input_size.width;
00074     recommended_height = input_size.height;
00075   };
00076 }
00077
00078 void printHelp() {
00079   std::cout « "Help: Usage of the program." « std::endl;
00080   std::cout « "Options:" « std::endl;
00081   std::cout « "-h: Display help." « std::endl;
00082   std::cout « "-p: Probe image file." « std::endl;
00083   std::cout « "-g: Gallery folder." « std::endl;
00084   std::cout « "-d: Path to detector solver." « std::endl;
00085   std::cout « "-l: Path to landmarks solver." « std::endl;
00086   std::cout « "-e: Path to extraction solver." « std::endl;
00087   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00088   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00089   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00090   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00091 }
00092
00093 void detectFace(SFEImage &image, SFESolver &detector_solver,
00094                 SFEDetection &detected_face) {
```

```
00095
00096   SFEImage resized_image{};
00097   DEFER(sfeImageFree(resized_image));
00098   SFEError error{};
00099   { // STAGE 1 Load image
00100     size_t recommended_width{};
00101     size_t recommended_height{};
00102
00103     // Calculate optimal input image size for face detection. Image will be
00104     // resized to recommended size for optimal performance.
00105     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00106                             face_size_max, recommended_width,
00107                             recommended_height);
00108
00109     // Resize image to recommended size
00110     // NOTE: This function will resize the image without preserving the aspect
00111     // ratio of the image.
00112
00113     error = sfeImageResize(image, recommended_width, recommended_height,
00114                            &resized_image);
00115     utils::checkError(error);
00116   }
00117
00118   size_t detection_count = 1;
00119   { // STAGE 2: Detect face in the image
00120
00121     // Detect face in the image
00122     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00123                           &detected_face, &detection_count);
00124     utils::checkError(error);
00125
00126     if (detection_count == 0) {
00127       std::ostringstream oss;
00128       oss « "Error: No face detected in the image ";
00129       throw std::runtime_error(oss.str());
00130
00131     } else if (detection_count > 1) {
00132       std::ostringstream oss;
00133       oss « "Error: Found " « detection_count
00134           « " faces. Please provide an image with only one face.";
00135
00136       throw std::runtime_error(oss.str());
00137     }
00138
00139     std::cout « "Detected face in the image." « std::endl;
00140   }
00141 }
00142
00144 int main(int argc, char *argv[]) {
00145
00146   { // STAGE 0: Parse command line arguments
00147     // Map to store argument values
00148     std::unordered_map<std::string, std::string> args;
00149
00150     // Parse command line arguments
00151     for (int i = 1; i < argc; ++i) {
00152       std::string arg = argv[i];
00153       if (arg[0] == '-') {
00154         if (arg == "-h") {
00155           printHelp();
00156           return 0;
00157         }
00158         // Check if there's a next argument and it isn't another option
00159         if (i + 1 < argc && argv[i + 1][0] != '-') {
00160           args[arg] = argv[++i];
00161         } else {
00162           std::cerr « "Option " « arg « " requires a value." « std::endl;
00163           return 1;
00164         }
00165       } else {
00166         std::cerr « "Unknown option: " « arg « std::endl;
00167         return 1;
00168       }
00169     }
00170
00171     // Assign values to variables based on parsed arguments
00172     if (args.count("-p"))
00173       image_probe = args["-p"];
00174     if (args.count("-g"))
00175       image_gallery = args["-g"];
00176     if (args.count("-d"))
00177       solver_face_detect = args["-d"];
00178     if (args.count("-l"))
00179       solver_face_landmarks = args["-l"];
00180     if (args.count("-e"))
00181       solver_face_template = args["-e"];
00182     if (args.count("-t"))
```

```
00183       detection_threshold = std::stof(args["-t"]);
00184     if (args.count("-i"))
00185       identification_threshold = std::stof(args["-i"]);
00186     if (args.count("-m"))
00187       face_size_min = std::stoi(args["-m"]);
00188     if (args.count("-x"))
00189       face_size_max = std::stoi(args["-x"]);
00190
00191     utils::printFormatted("EXAMPLE PARAMETERS");
00192     // Print resource names
00193     std::cout « "Probe image: " « image_probe « std::endl;
00194     std::cout « "Gallery image: " « image_gallery « std::endl;
00195
00196     // Print solver names
00197     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00198     std::cout « "Face landmarks solver: " « solver_face_landmarks
00199               « std::endl;
00200     std::cout « "Face template solver: " « solver_face_template « std::endl;
00201
00202     // Print other options
00203     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00204     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00205     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00206     std::cout « "Identification threshold: " « identification_threshold
00207               « std::endl;
00208   }
00209
00210   utils::printToolkitInfo();
00211
00212   SFEError error{};
00213
00214   SFESolver detector_solver{};
00215   DEFER(sfeSolverFree(detector_solver));
00216   SFESolver landmarks_solver{};
00217   DEFER(sfeSolverFree(landmarks_solver));
00218   SFESolver template_solver{};
00219   DEFER(sfeSolverFree(template_solver));
00220   { // STAGE 1.1: loading solvers
00221     utils::printFormatted("LOADING SOLVERS");
00222     // Initialize face detection solver
00223     error = sfeSolverCreate(
00224         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00225         SOLVER_PARAMETERS.size(), &detector_solver);
00226     utils::checkError(error);
00227
00228     // Initialize landmarks detection solver
00229     error = sfeSolverCreate(
00230         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00231         SOLVER_PARAMETERS.size(), &landmarks_solver);
00232     utils::checkError(error);
00233
00234     // Initialize template extraction solver
00235     error = sfeSolverCreate(
00236         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00237         SOLVER_PARAMETERS.size(), &template_solver);
00238     utils::checkError(error);
00239   }
00240
00241   SFEFaceTemplate probe_face_template;
00242   { // STAGE 1: Extract probe face template
00243     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00244
00245     SFEImage image{};
00246     DEFER(sfeImageFree(image));
00247     { // STAGE 1.1: Load image
00248       // Load image data from file
00249       auto image_data = utils::readFile(image_probe);
00250
00251       // Decode image from data
00252       error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00253       utils::checkError(error);
00254     }
00255
00256     SFEDetection detected_face = {};
00257     { // STAGE 1.2: Detect face in the image
00258       // Detect a face in the image
00259       detectFace(image, detector_solver, detected_face);
00260     }
00261
00262     std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00263     { // STAGE 1.3: Get face landmarks
00264       // Use landmarks detection solver to get relevant landmarks for
00265       // the detected face
00266       error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00267                                landmarks.data());
00268       utils::checkError(error);
00269     }
```

```
00270
00272     { // STAGE 1.4: Extract probe face template
00273       // Use template extraction solver to create new face
00274       // template
00275       error = sfeFaceTemplateExtract(template_solver, image, &detected_face,
00276                                      landmarks.data(), &probe_face_template);
00277       utils::checkError(error);
00278     }
00280
00281     { // STAGE 1.5: (optional) Print template attributes
00282       utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00283
00285       SFEFaceTemplateVersion version = {};
00286       error = sfeFaceTemplateVersion(&probe_face_template, &version);
00287       utils::checkError(error);
00288
00289       auto version_minor = (int)version.version_minor - (int)'0';
00290       std::cout « "Version of the probe template: " « version.version_major
00291                 « '.' « version_minor « std::endl;
00293     }
00294
00296     { // STAGE 1.6: (optional) Export and re-import template
00297       std::vector<uint8_t> buf(1024);
00298       size_t size = buf.size();
00299       error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
00300       if (error && size > buf.size()) {
00301         buf.resize(size);
00302         size = buf.size();
00303         error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
00304       }
00305       utils::checkError(error);
00306
00307       SFEFaceTemplate imported_template = {};
00308       error = sfeFaceTemplateImport(buf.data(), size, &imported_template);
00309       utils::checkError(error);
00310
00311       float roundtrip_score = 0.f;
00312       error = sfeFaceTemplateMatch(&probe_face_template, &imported_template,
00313                                    &roundtrip_score);
00314       utils::checkError(error);
00315       std::cout « "Export/import round-trip match score: " « roundtrip_score
00316                 « std::endl;
00317     }
00319   }
00320
00321   std::vector<std::string> image_paths{};
00322   { // STAGE 2: Get gallery image paths
00323     // Get folder names from the face image gallery
00324     auto folder_names = utils::getFiles(image_gallery);
00325
00326     for (auto folder_name : folder_names) {
00327       auto gallery_folder = image_gallery + folder_name + "/";
00328
00329       // Get image names from folder
00330       auto image_names = utils::getFiles(gallery_folder);
00331
00332       // Extract template for each image in the folder
00333       for (auto image_name : image_names) {
00334
00335         auto image_path = gallery_folder + image_name;
00336
00337         image_paths.push_back(image_path);
00338       }
00339     }
00340   }
00341
00342   std::vector<SFEFaceTemplate> gallery_face_templates(image_paths.size());
00343   std::vector<SFEDetection> detected_faces(image_paths.size());
00344   std::vector<std::array<SFEFaceLandmarks, SFE_FACE_LANDMARK_COUNT» landmarks(
00345       image_paths.size());
00346   { // STAGE 2: Load gallery image and extract face templates for each image in
00347     // the gallery
00348     utils::printFormatted("GALLERY TEMPLATE EXTRACTION");
00349
00350     for (size_t i = 0; i < image_paths.size(); i++) {
00351       std::cout « "Processing image #" « i « ", path: " « image_paths[i]
00352                 « std::endl;
00353       SFEImage image{};
00354       DEFER(sfeImageFree(image));
00355       { // STAGE 2.1: Load image
00356         // Load image data from file
00357         auto image_data = utils::readFile(image_paths[i]);
00358
00359         // Decode image from data
00360         error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00361         utils::checkError(error);
00362       }
```

```
00363
00364       { // STAGE 2.2: Detect face in the image
00365         // Detect a face in the image
00366         detectFace(image, detector_solver, detected_faces[i]);
00367       }
00368
00369       { // STAGE 2.3: Get face landmarks
00370         // Use landmarks detection solver to get relevant landmarks for
00371         // the detected face
00372         error = sfeFaceLandmarks(landmarks_solver, image, &detected_faces[i],
00373                                  landmarks[i].data());
00374         utils::checkError(error);
00375       }
00376
00377       { // STAGE 2.3: Extract face template
00378         // Use template extraction solver to get face_template solver
00379         error = sfeFaceTemplateExtract(
00380             template_solver, image, &detected_faces[i], landmarks[i].data(),
00381             &gallery_face_templates[i]);
00382         utils::checkError(error);
00383         std::cout « "Extracted face template." « std::endl;
00384       }
00385       std::cout « std::endl;
00386     }
00387   }
00388
00390   size_t candidate_count = 1;
00391   int best_candidate_index = -1;
00392   std::vector<SFETemplateIdentificationCandidate> identification_results(
00393       image_paths.size());
00394   { // STAGE 3: Identification
00395     utils::printFormatted("1:N IDENTIFICATION");
00396
00397     error = sfeFaceTemplateIdentify(
00398         &probe_face_template, gallery_face_templates.data(),
00399         gallery_face_templates.size(), identification_threshold,
00400         identification_results.data(), &candidate_count, 4);
00401     utils::checkError(error);
00402
00403     // Resize the results vector to the actual number of candidates found, in
00404     // the case there is less than candidate_count
00405     identification_results.resize(candidate_count);
00406
00407     std::cout « "Found " « identification_results.size()
00408               « " candidates above identification threshold "
00409               « identification_threshold « std::endl;
00410     for (auto &result : identification_results)
00411       std::cout « "Template index: #" « result.index
00412                 « ", score: " « result.score « std::endl;
00413
00414     // Since it's sorted vector by score, the best candidate is the first one
00415     best_candidate_index = identification_results[0].index;
00416   }
00418
00420   { // STAGE 4: (optional) 1:1 matching with top candidate to showcase 1:1
00421     // matching
00422     utils::printFormatted("1:1 MATCHING WITH TOP CANDIDATE");
00423
00424     if (identification_results.size() == 0) {
00425       std::cout « "No candidates found above identification threshold "
00426                 « identification_threshold « std::endl;
00427       return 0;
00428     }
00429
00430     auto best_candidate_template = gallery_face_templates[best_candidate_index];
00431
00432     float match_confidence;
00433     error = sfeFaceTemplateMatch(&probe_face_template, &best_candidate_template,
00434                                  &match_confidence);
00435     utils::checkError(error);
00436
00437     std::cout « "Matching score of probe template with template index #"
00438               « best_candidate_index « ", score: " « match_confidence
00439               « std::endl;
00440   }
00442
00443   { // STAGE: 5 Optional, get face crop of best face and its bounding box and
00444     // save it to file
00445     utils::printFormatted("SAVE FACE CROP AND ANNOTATED IMAGE");
00446
00447     auto best_face_index = identification_results[0].index;
00448
00449     SFEImage image{};
00450     DEFER(sfeImageFree(image));
00451     { // STAGE 5.1: Load image
00452       // Load image data from file
00453       auto image_data = utils::readFile(image_paths[best_face_index]);
```

```
00454
00455       // Decode image from data
00456       error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00457       utils::checkError(error);
00458     }
00459
00461     SFEFaceCrop crop_data = {};
00462     DEFER(sfeImageFree(crop_data.crop_image));
00463     { // STAGE 5.2: Get face crop
00464       //  Defines the size of the crop as an extension of
00465       // the detection bounding box.
00466       float face_size_extension = 2.0f;
00467       SFEError error =
00468           sfeFaceCrop(image, landmarks[best_face_index].data(),
00469                       face_size_extension, (float)(face_size_max), &crop_data);
00470       utils::checkError(error);
00471
00472       std::cout « "Face crop extension: " « crop_data.face_size_extension
00473                 « ", width of cropped image: " « crop_data.crop_image.width
00474                 « "px, height of cropped image: "
00475                 « crop_data.crop_image.height « "px." « std::endl;
00476     }
00478
00479     { // STAGE 5.3: Save face crop to file
00480       auto png_file = std::vector<unsigned char>();
00481       size_t size =
00482           crop_data.crop_image.width * crop_data.crop_image.height * 3;
00483       png_file.resize(size);
00484       SFEError error2 = sfeImageEncode(crop_data.crop_image, SFE_IMAGE_FORMAT_PNG,
00485                                        png_file.data(), &size);
00486       utils::checkError(error2);
00487
00488       std::cout « "Face crop saved as crop_identified_person.png" « std::endl;
00489       utils::saveFile("crop_identified_person.png", png_file);
00490     }
00491
00492     { // STAGE 5.4: Annotate the image
00493       // Render bounding box with detection confidence and identification score
00494       std::stringstream label;
00495       label « " D:" « std::fixed « std::setprecision(2)
00496             « detected_faces[best_face_index].confidence
00497             « " M:" « identification_results[0].score;
00498
00499       auto color = annotate::RED;
00500
00501       annotate::labelBox(image, label.str(),
00502                          detected_faces[best_candidate_index].bounding_box,
00503                          annotate::WHITE, color);
00504
00505       // Render landmarks
00506       for (auto &landmark : landmarks[best_face_index]) {
00507         auto x = landmark.x * image.width;
00508         auto y = landmark.y * image.height;
00509         annotate::circle(image, x, y, 3, annotate::YELLOW);
00510       }
00511
00512       // Save annotated image
00513       size_t size = image.width * image.height * 3;
00514       auto png_file = std::vector<unsigned char>(size);
00515       error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00516       utils::checkError(error);
00517       png_file.resize(size);
00518       utils::saveFile("face_identify.png", png_file);
00519
00520       std::cout « std::endl;
00521       std::cout « "Annotated image saved as face_identify.png" « std::endl;
00522     }
00523   }
00524
00525   utils::printFormatted("FINISHED");
00526 }
```

9.9 D:/glab/1512fd1724a/1/example/example_face_identify_entity.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_face.h¨
#include <regex>
#include <vector>
#include ¨solvers.h¨
```

```
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- void getRecommendedImageSize (const std::string &solver_face_detect, SFEImage &image, const size_t face_size_min, const size_t face_size_max, size_t &recommended_width, size_t &recommended_height)

  Get recommended image size for face detection.
- SFEFaceTemplate extractTemplate (std::string image_path, SFESolver &detector_solver, SFESolver &landmarks_solver, SFESolver &template_solver)
- void printHelp ()
- int main (int argc, char ∗argv[ ])

Variables

- std::string gallery = ¨./assets/face/entities/¨
- std::string image_probe = ¨assets/face/images/obiwan0.png¨
- std::string solver_face_detect = SOLVER_FACE_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
- std::string solver_face_template = SOLVER_FACE_EXTRACTION
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- size_t face_size_min = 28

  Required size of the face to be detected.
- size_t face_size_max = 170
- float detection_threshold = 0.1f
- float identification_threshold = 0.6f

9.9.1 Function Documentation

9.9.1.1 extractTemplate()

```
SFEFaceTemplate extractTemplate (
            std::string image_path,
            SFESolver & detector_solver,
            SFESolver & landmarks_solver,
            SFESolver & template_solver )
```

Examples

example_face_identify_entity.cpp.

Definition at line 74 of file example_face_identify_entity.cpp.

```
00077                                                                    {
00078
00079   SFEImage image{};
00080   DEFER(sfeImageFree(image));
00081   SFEImage resized_image{};
00082   DEFER(sfeImageFree(resized_image));
00083   SFEError error{};
00084   { // STAGE 1 Load image
00085     // Load image data from file
00086     auto image_data = utils::readFile(image_path);
00087
00088     // Decode image from data
00089     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00090     utils::checkError(error);
00091
00092     size_t recommended_width{};
00093     size_t recommended_height{};
00094
00095     // Calculate optimal input image size for face detection. Image will be
00096     // resized to recommended size for optimal performance.
00097     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00098                             face_size_max, recommended_width,
```

```
00099                             recommended_height);
00100
00101     // Resize image to recommended size
00102     // NOTE: This function will resize the image without preserving the aspect
00103     // ratio of the image.
00104
00105     error = sfeImageResize(image, recommended_width, recommended_height,
00106                            &resized_image);
00107     utils::checkError(error);
00108   }
00109
00110   SFEDetection detected_face = {};
00111   size_t detection_count = 1;
00112   { // STAGE 3: Detect face in the image
00113
00114     // Detect face in the image
00115     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00116                           &detected_face, &detection_count);
00117     utils::checkError(error);
00118
00119     if (detection_count == 0) {
00120       throw std::runtime_error("No face detected in the image.");
00121     }
00122   }
00123
00124   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00125   { // STAGE 4: Get face landmarks
00126
00127     // Use landmarks detection solver to get relevant landmarks for
00128     // the detected face
00129     error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00130                              landmarks.data());
00131     utils::checkError(error);
00132   }
00133
00134   SFEFaceTemplate face_template = {};
00135   { // STAGE 5: Extract probe face template
00136     // Use template extraction solver to create new face
00137     // template
00138     error =
00139         sfeFaceTemplateExtract(template_solver, image, &detected_face,
00140                                landmarks.data(), &face_template);
00141     utils::checkError(error);
00142   }
00143   return face_template;
00144 }
```

9.9.1.2 getRecommendedImageSize()

```
void getRecommendedImageSize (
              const std::string & solver_face_detect,
              SFEImage & image,
              const size_t face_size_min,
              const size_t face_size_max,
              size_t & recommended_width,
              size_t & recommended_height )
```

Get recommended image size for face detection.

Parameters

solver_face_detect	name of the face detection solver
face_size_min	Desired minimal size of detected face
face_size_max	Desired maximal size of detected face
recommended_width	Recommended image width
recommended_height	Recommended image height

Definition at line 39 of file example_face_identify_entity.cpp.

```
00043                                                                    {
00044
00045   // If the face detection solver requires static input (filename contains width and height),
00046   // we need to prepare the input image accordingly Typically the format is:
00047   // face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver
00048   // NOTE: This can
00049   // be hardcoded into the application, or we can parse the width and height
00050   // from the solver filename
00051
```

```
00052   // Try to use regex capture to extract width and height from solver name
00053   std::smatch width_height;
00054   if (std::regex_match(solver_face_detect, width_height,
00055                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00056     recommended_width = std::stoi(width_height[1]);
00057     recommended_height = std::stoi(width_height[2]);
00058   } else {
00059     // Solvers without width and height in name can accept dynamic input
00060     // image size. For best results we recommend to scale the input image to
00061     // a resolution recommended by the sfeFaceDetectInputSize function
00062
00063     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00064     // for given min_face_size/max_face_size
00065     SFEDetectionInputSize input_size{};
00066     SFEError error = sfeFaceDetectInputSize(image,
      SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, face_size_min,
00067                                             face_size_max, &input_size);
00068     utils::checkError(error);
00069     recommended_width = input_size.width;
00070     recommended_height = input_size.height;
00071   };
00072 }
```

9.9.1.3 main()

```
int main (
            int argc,
            char * argv[] )
```

[Face Entity Identify]
[Face Entity Identify]
Definition at line 161 of file example_face_identify_entity.cpp.

```
00161                                {
00162
00163   { // STAGE 0: Parse command line arguments
00164
00165     // Map to store argument values
00166     std::unordered_map<std::string, std::string> args;
00167
00168     // Parse command line arguments
00169     for (int i = 1; i < argc; ++i) {
00170       std::string arg = argv[i];
00171       if (arg[0] == '-') {
00172         if (arg == "-h") {
00173           printHelp();
00174           return 0;
00175         }
00176         // Check if there's a next argument and it isn't another option
00177         if (i + 1 < argc && argv[i + 1][0] != '-') {
00178           args[arg] = argv[++i];
00179         } else {
00180           std::cerr << "Option " << arg << " requires a value." << std::endl;
00181           return 1;
00182         }
00183       } else {
00184         std::cerr << "Unknown option: " << arg << std::endl;
00185         return 1;
00186       }
00187     }
00188
00189     // Assign values to variables based on parsed arguments
00190     if (args.count("-p"))
00191       image_probe = args["-p"];
00192     if (args.count("-g"))
00193       gallery = args["-g"];
00194     if (args.count("-d"))
00195       solver_face_detect = args["-d"];
00196     if (args.count("-l"))
00197       solver_face_landmarks = args["-l"];
00198     if (args.count("-e"))
00199       solver_face_template = args["-e"];
00200     if (args.count("-t"))
00201       detection_threshold = std::stof(args["-t"]);
00202     if (args.count("-i"))
00203       identification_threshold = std::stof(args["-i"]);
00204     if (args.count("-m"))
00205       face_size_min = std::stoi(args["-m"]);
00206     if (args.count("-x"))
00207       face_size_max = std::stoi(args["-x"]);
00208
00209     utils::printFormatted("EXAMPLE PARAMETERS");
00210     // Print resource names
00211     std::cout << "Probe image: " << image_probe << std::endl;
00212     std::cout << "Entities folder: " << gallery << std::endl;
```

```
00213
00214     // Print solver names
00215     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00216     std::cout « "Face landmarks solver: " « solver_face_landmarks
00217               « std::endl;
00218     std::cout « "Face template solver: " « solver_face_template « std::endl;
00219
00220     // Print other options
00221     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00222     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00223     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00224     std::cout « "Identification threshold: " « identification_threshold
00225               « std::endl;
00226   }
00227
00228   utils::printToolkitInfo();
00229
00230   SFEError error{};
00231
00232   SFESolver detector_solver{};
00233   DEFER(sfeSolverFree(detector_solver));
00234   SFESolver landmarks_solver{};
00235   DEFER(sfeSolverFree(landmarks_solver));
00236   SFESolver template_solver{};
00237   DEFER(sfeSolverFree(template_solver));
00238   { // STAGE 0: loading solvers
00239     // Initialize face detection solver
00240     error = sfeSolverCreate(
00241         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00242         SOLVER_PARAMETERS.size(), &detector_solver);
00243     utils::checkError(error);
00244
00245     // Initialize landmarks detection solver
00246     error = sfeSolverCreate(
00247         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00248         SOLVER_PARAMETERS.size(), &landmarks_solver);
00249     utils::checkError(error);
00250
00251     // Initialize template extraction solver
00252     error = sfeSolverCreate(
00253         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00254         SOLVER_PARAMETERS.size(), &template_solver);
00255     utils::checkError(error);
00256   }
00257
00258   SFEFaceTemplate probe_face_template;
00259   { // STAGE 1: Extract probe face template
00260     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00261     probe_face_template = extractTemplate(image_probe, detector_solver,
00262                                           landmarks_solver, template_solver);
00263
00264     std::cout « "Probe face template extracted." « std::endl;
00265   }
00266
00267   std::vector<SFEFaceTemplate> gallery_face_templates = {};
00268   std::vector<SFEEntity> enitity_pairs = {};
00269   { // STAGE 2: Extract templates for each image in entity folder and associate them
00270     // with entity(UUID)
00271
00272     // Entities(UUID) tells the ownership of the templates in the gallery.
00273     // For example:
00274     // UUID1 -> template1
00275     // UUID1 -> template2
00276     // UUID2 -> template3
00277     // UUID2 -> template4
00278     // ...
00279     // UUUIDN -> templateN
00280
00281     utils::printFormatted("ENTITIES TEMPLATE EXTRACTION");
00282
00283     // Get folder names from entities_gallery
00284     auto folder_names = utils::getFiles(gallery);
00285
00286     for (auto folder_name : folder_names) {
00287       if (folder_name.find(".jpg") != std::string::npos ||
00288           folder_name.find(".png") != std::string::npos) {
00289         continue;
00290       }
00291
00292       // Genereate UUID for each entity.
00293       auto entity = utils::generateEntity();
00294
00295       auto entity_folder = gallery + folder_name + "/";
00296
00297       std::cout « "Folder: " « entity_folder « ", entity UUID: ["
00298                 « static_cast<int>(entity.uuid[0]) « ", "
00299                 « static_cast<int>(entity.uuid[1]) « "..."
```

```
00300                 « static_cast<int>(entity.uuid[15]) « "]" « std::endl;
00301
00302       // Get image names from folder
00303       auto image_names = utils::getFiles(entity_folder);
00304
00305       // Extract template for each image in the folder
00306       for (auto image_name : image_names) {
00307
00308         auto image_path = entity_folder + image_name;
00309
00310         auto face_template = extractTemplate(
00311             image_path, detector_solver, landmarks_solver, template_solver);
00312
00313         gallery_face_templates.push_back(face_template);
00314         enitity_pairs.push_back(entity);
00315       }
00316     }
00317   }
00318
00320   size_t candidate_count = 1;
00321   std::vector<SFEEntityIdentificationCandidate> results(candidate_count);
00322   int best_candidate_index = -1;
00323   { // STAGE 3: Identification with entities.
00324     // Probe template is matched against every template in the gallery.
00325     // Function returns entity(UUID) which owns the best matching template.
00326
00327     utils::printFormatted("1:N IDENTIFICATION WITH ENTITIES");
00328     SFEError error = sfeFaceEntityIdentify(
00329         &probe_face_template, gallery_face_templates.data(),
00330         enitity_pairs.data(), gallery_face_templates.size(),
00331         identification_threshold, results.data(), &candidate_count, 4);
00332     utils::checkError(error);
00333
00334     if (candidate_count == 0) {
00335       std::cout « "No candidates found. Exiting.." « std::endl;
00336       return 0;
00337     }
00338
00339     std::cout « std::endl;
00340     std::cout « "Found " « results.size() « " candidates " « std::endl;
00341     for (int i = 0; i < results.size(); i++) {
00342       std::cout « "Index: " « i « ", Score: " « results[i].score
00343                 « std::endl;
00344       std::cout « "Entity UUID: ["
00345                 « static_cast<int>(results[i].entity.uuid[0]) « ", "
00346                 « static_cast<int>(results[i].entity.uuid[1]) « "..."
00347                 « static_cast<int>(results[i].entity.uuid[15]) « "]"
00348                 « std::endl;
00349     }
00350   }
00352
00353   utils::printFormatted("FINISHED");
00354 }
```

9.9.1.4 printHelp()

```
void printHelp ( )
```

Definition at line 146 of file example_face_identify_entity.cpp.

```
00146                  {
00147   std::cout « "Help: Usage of the program." « std::endl;
00148   std::cout « "Options:" « std::endl;
00149   std::cout « "-h: Display help." « std::endl;
00150   std::cout « "-p: Probe image file." « std::endl;
00151   std::cout « "-g: Path to folder with entities." « std::endl;
00152   std::cout « "-d: Path to detector solver." « std::endl;
00153   std::cout « "-l: Path to landmarks solver." « std::endl;
00154   std::cout « "-e: Path to extraction solver." « std::endl;
00155   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00156   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00157   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00158   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00159 }
```

9.9.2 Variable Documentation

9.9.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 30 of file example_face_identify_entity.cpp.

9.9.2.2 face_size_max

```
size_t face_size_max = 170
```
Definition at line 28 of file example_face_identify_entity.cpp.

9.9.2.3 face_size_min

```
size_t face_size_min = 28
```
Required size of the face to be detected.
Definition at line 27 of file example_face_identify_entity.cpp.

9.9.2.4 gallery

```
std::string gallery = ¨./assets/face/entities/¨
```

Examples

example_face_identify_entity.cpp.

Definition at line 15 of file example_face_identify_entity.cpp.

9.9.2.5 identification_threshold

```
float identification_threshold = 0.6f
```
Definition at line 31 of file example_face_identify_entity.cpp.

9.9.2.6 image_probe

```
std::string image_probe = ¨assets/face/images/obiwan0.png¨
```
Definition at line 17 of file example_face_identify_entity.cpp.

9.9.2.7 solver_face_detect

```
std::string solver_face_detect = SOLVER_FACE_DETECT
```
Solvers to use in example, the defaults are filled in by CMake.
Definition at line 20 of file example_face_identify_entity.cpp.

9.9.2.8 solver_face_landmarks

```
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
```
Definition at line 21 of file example_face_identify_entity.cpp.

9.9.2.9 solver_face_template

```
std::string solver_face_template = SOLVER_FACE_EXTRACTION
```
Definition at line 22 of file example_face_identify_entity.cpp.

9.9.2.10 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```
Definition at line 24 of file example_face_identify_entity.cpp.

9.10 example_face_identify_entity.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_face.h"
00007
00008 #include <regex>
00009 #include <vector>
00010
00011 #include "solvers.h"
```

```
00012 #include "utils.hpp"
00013 #include <unordered_map>
00014
00015 std::string gallery = "./assets/face/entities/";
00016
00017 std::string image_probe = "assets/face/images/obiwan0.png";
00018
00020 std::string solver_face_detect = SOLVER_FACE_DETECT;
00021 std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
00022 std::string solver_face_template = SOLVER_FACE_EXTRACTION;
00023
00024 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00025
00027 size_t face_size_min = 28;
00028 size_t face_size_max = 170;
00029
00030 float detection_threshold = 0.1f;
00031 float identification_threshold = 0.6f;
00032
00039 void getRecommendedImageSize(const std::string &solver_face_detect,
00040                              SFEImage &image, const size_t face_size_min,
00041                              const size_t face_size_max,
00042                              size_t &recommended_width,
00043                              size_t &recommended_height) {
00044
00045   // If the face detection solver requires static input (filename contains width and height),
00046   // we need to prepare the input image accordingly Typically the format is:
00047   // face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver
00048   // NOTE: This can
00049   // be hardcoded into the application, or we can parse the width and height
00050   // from the solver filename
00051
00052   // Try to use regex capture to extract width and height from solver name
00053   std::smatch width_height;
00054   if (std::regex_match(solver_face_detect, width_height,
00055                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00056     recommended_width = std::stoi(width_height[1]);
00057     recommended_height = std::stoi(width_height[2]);
00058   } else {
00059     // Solvers without width and height in name can accept dynamic input
00060     // image size. For best results we recommend to scale the input image to
00061     // a resolution recommended by the sfeFaceDetectInputSize function
00062
00063     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00064     // for given min_face_size/max_face_size
00065     SFEDetectionInputSize input_size{};
00066     SFEError error = sfeFaceDetectInputSize(image,
      SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, face_size_min,
00067                                             face_size_max, &input_size);
00068     utils::checkError(error);
00069     recommended_width = input_size.width;
00070     recommended_height = input_size.height;
00071   };
00072 }
00073
00074 SFEFaceTemplate extractTemplate(std::string image_path,
00075                                 SFESolver &detector_solver,
00076                                 SFESolver &landmarks_solver,
00077                                 SFESolver &template_solver) {
00078
00079   SFEImage image{};
00080   DEFER(sfeImageFree(image));
00081   SFEImage resized_image{};
00082   DEFER(sfeImageFree(resized_image));
00083   SFEError error{};
00084   { // STAGE 1 Load image
00085     // Load image data from file
00086     auto image_data = utils::readFile(image_path);
00087
00088     // Decode image from data
00089     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00090     utils::checkError(error);
00091
00092     size_t recommended_width{};
00093     size_t recommended_height{};
00094
00095     // Calculate optimal input image size for face detection. Image will be
00096     // resized to recommended size for optimal performance.
00097     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00098                             face_size_max, recommended_width,
00099                             recommended_height);
00100
00101     // Resize image to recommended size
00102     // NOTE: This function will resize the image without preserving the aspect
00103     // ratio of the image.
00104
00105     error = sfeImageResize(image, recommended_width, recommended_height,
```

```
00106                             &resized_image);
00107     utils::checkError(error);
00108   }
00109
00110   SFEDetection detected_face = {};
00111   size_t detection_count = 1;
00112   { // STAGE 3: Detect face in the image
00113
00114     // Detect face in the image
00115     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00116                       &detected_face, &detection_count);
00117     utils::checkError(error);
00118
00119     if (detection_count == 0) {
00120       throw std::runtime_error("No face detected in the image.");
00121     }
00122   }
00123
00124   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00125   { // STAGE 4: Get face landmarks
00126
00127     // Use landmarks detection solver to get relevant landmarks for
00128     // the detected face
00129     error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00130                              landmarks.data());
00131     utils::checkError(error);
00132   }
00133
00134   SFEFaceTemplate face_template = {};
00135   { // STAGE 5: Extract probe face template
00136     // Use template extraction solver to create new face
00137     // template
00138     error =
00139         sfeFaceTemplateExtract(template_solver, image, &detected_face,
00140                                landmarks.data(), &face_template);
00141     utils::checkError(error);
00142   }
00143   return face_template;
00144 }
00145
00146 void printHelp() {
00147   std::cout « "Help: Usage of the program." « std::endl;
00148   std::cout « "Options:" « std::endl;
00149   std::cout « "-h: Display help." « std::endl;
00150   std::cout « "-p: Probe image file." « std::endl;
00151   std::cout « "-g: Path to folder with entities." « std::endl;
00152   std::cout « "-d: Path to detector solver." « std::endl;
00153   std::cout « "-l: Path to landmarks solver." « std::endl;
00154   std::cout « "-e: Path to extraction solver." « std::endl;
00155   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00156   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00157   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00158   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00159 }
00160
00161 int main(int argc, char *argv[]) {
00162
00163   { // STAGE 0: Parse command line arguments
00164
00165     // Map to store argument values
00166     std::unordered_map<std::string, std::string> args;
00167
00168     // Parse command line arguments
00169     for (int i = 1; i < argc; ++i) {
00170       std::string arg = argv[i];
00171       if (arg[0] == '-') {
00172         if (arg == "-h") {
00173           printHelp();
00174           return 0;
00175         }
00176         // Check if there's a next argument and it isn't another option
00177         if (i + 1 < argc && argv[i + 1][0] != '-') {
00178           args[arg] = argv[++i];
00179         } else {
00180           std::cerr « "Option " « arg « " requires a value." « std::endl;
00181           return 1;
00182         }
00183       } else {
00184         std::cerr « "Unknown option: " « arg « std::endl;
00185         return 1;
00186       }
00187     }
00188
00189     // Assign values to variables based on parsed arguments
00190     if (args.count("-p"))
00191       image_probe = args["-p"];
00192     if (args.count("-g"))
```

```
00193       gallery = args["-g"];
00194     if (args.count("-d"))
00195       solver_face_detect = args["-d"];
00196     if (args.count("-l"))
00197       solver_face_landmarks = args["-l"];
00198     if (args.count("-e"))
00199       solver_face_template = args["-e"];
00200     if (args.count("-t"))
00201       detection_threshold = std::stof(args["-t"]);
00202     if (args.count("-i"))
00203       identification_threshold = std::stof(args["-i"]);
00204     if (args.count("-m"))
00205       face_size_min = std::stoi(args["-m"]);
00206     if (args.count("-x"))
00207       face_size_max = std::stoi(args["-x"]);
00208
00209     utils::printFormatted("EXAMPLE PARAMETERS");
00210     // Print resource names
00211     std::cout << "Probe image: " << image_probe << std::endl;
00212     std::cout << "Entities folder: " << gallery << std::endl;
00213
00214     // Print solver names
00215     std::cout << "Face detection solver: " << solver_face_detect << std::endl;
00216     std::cout << "Face landmarks solver: " << solver_face_landmarks
00217               << std::endl;
00218     std::cout << "Face template solver: " << solver_face_template << std::endl;
00219
00220     // Print other options
00221     std::cout << "Min face size [px]: " << face_size_min << std::endl;
00222     std::cout << "Max face size [px]: " << face_size_max << std::endl;
00223     std::cout << "Detection threshold: " << detection_threshold << std::endl;
00224     std::cout << "Identification threshold: " << identification_threshold
00225               << std::endl;
00226   }
00227
00228   utils::printToolkitInfo();
00229
00230   SFEError error{};
00231
00232   SFESolver detector_solver{};
00233   DEFER(sfeSolverFree(detector_solver));
00234   SFESolver landmarks_solver{};
00235   DEFER(sfeSolverFree(landmarks_solver));
00236   SFESolver template_solver{};
00237   DEFER(sfeSolverFree(template_solver));
00238   { // STAGE 0: loading solvers
00239     // Initialize face detection solver
00240     error = sfeSolverCreate(
00241         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00242         SOLVER_PARAMETERS.size(), &detector_solver);
00243     utils::checkError(error);
00244
00245     // Initialize landmarks detection solver
00246     error = sfeSolverCreate(
00247         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00248         SOLVER_PARAMETERS.size(), &landmarks_solver);
00249     utils::checkError(error);
00250
00251     // Initialize template extraction solver
00252     error = sfeSolverCreate(
00253         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00254         SOLVER_PARAMETERS.size(), &template_solver);
00255     utils::checkError(error);
00256   }
00257
00258   SFEFaceTemplate probe_face_template;
00259   { // STAGE 1: Extract probe face template
00260     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00261     probe_face_template = extractTemplate(image_probe, detector_solver,
00262                                           landmarks_solver, template_solver);
00263
00264     std::cout << "Probe face template extracted." << std::endl;
00265   }
00266
00267   std::vector<SFEFaceTemplate> gallery_face_templates = {};
00268   std::vector<SFEEntity> enitity_pairs = {};
00269   { // STAGE 2: Extract templates for each image in entity folder and associate them
00270     // with entity(UUID)
00271
00272     // Entities(UUID) tells the ownership of the templates in the gallery.
00273     // For example:
00274     // UUID1 -> template1
00275     // UUID1 -> template2
00276     // UUID2 -> template3
00277     // UUID2 -> template4
00278     // ...
00279     // UUUIDN -> templateN
```

```
00280
00281     utils::printFormatted("ENTITIES TEMPLATE EXTRACTION");
00282
00283     // Get folder names from entities_gallery
00284     auto folder_names = utils::getFiles(gallery);
00285
00286     for (auto folder_name : folder_names) {
00287       if (folder_name.find(".jpg") != std::string::npos ||
00288           folder_name.find(".png") != std::string::npos) {
00289         continue;
00290       }
00291
00292       // Genereate UUID for each entity.
00293       auto entity = utils::generateEntity();
00294
00295       auto entity_folder = gallery + folder_name + "/";
00296
00297       std::cout « "Folder: " « entity_folder « ", entity UUID: ["
00298                 « static_cast<int>(entity.uuid[0]) « ", "
00299                 « static_cast<int>(entity.uuid[1]) « "..."
00300                 « static_cast<int>(entity.uuid[15]) « "]" « std::endl;
00301
00302       // Get image names from folder
00303       auto image_names = utils::getFiles(entity_folder);
00304
00305       // Extract template for each image in the folder
00306       for (auto image_name : image_names) {
00307
00308         auto image_path = entity_folder + image_name;
00309
00310         auto face_template = extractTemplate(
00311             image_path, detector_solver, landmarks_solver, template_solver);
00312
00313         gallery_face_templates.push_back(face_template);
00314         enitity_pairs.push_back(entity);
00315       }
00316     }
00317   }
00318
00320   size_t candidate_count = 1;
00321   std::vector<SFEEntityIdentificationCandidate> results(candidate_count);
00322   int best_candidate_index = -1;
00323   { // STAGE 3: Identification with entities.
00324     // Probe template is matched against every template in the gallery.
00325     // Function returns entity(UUID) which owns the best matching template.
00326
00327     utils::printFormatted("1:N IDENTIFICATION WITH ENTITIES");
00328     SFEError error = sfeFaceEntityIdentify(
00329         &probe_face_template, gallery_face_templates.data(),
00330         enitity_pairs.data(), gallery_face_templates.size(),
00331         identification_threshold, results.data(), &candidate_count, 4);
00332     utils::checkError(error);
00333
00334     if (candidate_count == 0) {
00335       std::cout « "No candidates found. Exiting.." « std::endl;
00336       return 0;
00337     }
00338
00339     std::cout « std::endl;
00340     std::cout « "Found " « results.size() « " candidates " « std::endl;
00341     for (int i = 0; i < results.size(); i++) {
00342       std::cout « "Index: " « i « ", Score: " « results[i].score
00343                 « std::endl;
00344       std::cout « "Entity UUID: ["
00345                 « static_cast<int>(results[i].entity.uuid[0]) « ", "
00346                 « static_cast<int>(results[i].entity.uuid[1]) « "..."
00347                 « static_cast<int>(results[i].entity.uuid[15]) « "]"
00348                 « std::endl;
00349     }
00350   }
00352
00353   utils::printFormatted("FINISHED");
00354 }
```

9.11 D:/glab/1512fd1724a/1/example/example_face_liveness.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_face.h¨
#include <iomanip>
#include <regex>
```

```
#include <sstream>
#include <vector>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- void getRecommendedImageSize (const std::string &solver_face_detect, SFEImage &image, const size_t face_size_min, const size_t face_size_max, size_t &recommended_width, size_t &recommended_height)

  Get recommended image size for face detection.
- void printFaceDetectionInfo (SFEDetection &detected_face)
- void prinFaceAreaInfo (SFEFaceArea &face_area)
- void printFaceHeadPose (SFEFaceHeadPose &head_pose)
- void printFaceSize (size_t face_size)
- void printMaskConfidence (float mask_confidence)
- void printFaceQualityAttributes (SFEFaceQualityAttributes &quality_attributes)
- void printHelp ()
- int main (int argc, char ∗argv[ ])

  Main fuction is used to parse command line arguments.

Variables

- std::string image_probe = ¨assets/face/images/obiwan0.png¨
- std::string solver_face_detect = SOLVER_FACE_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
- std::string solver_face_liveness = SOLVER_FACE_LIVENESS
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- size_t face_size_min = 28

  Required size of the face to be detected.
- size_t face_size_max = 170
- float detection_threshold = 0.1f

9.11.1 Function Documentation

9.11.1.1 getRecommendedImageSize()

```
void getRecommendedImageSize (
            const std::string & solver_face_detect,
            SFEImage & image,
            const size_t face_size_min,
            const size_t face_size_max,
            size_t & recommended_width,
            size_t & recommended_height )
```

Get recommended image size for face detection.

Parameters

solver_face_detect	name of the face detection solver
face_size_min	Desired minimal size of detected face
face_size_max	Desired maximal size of detected face
recommended_width	Recommended image width
recommended_height	Recommended image height

[Face Detect Input Size]
[Face Detect Input Size]
Definition at line 38 of file example_face_liveness.cpp.

```
00042                                                                {
00043
00044   // If the face detection solver requires static input (filename contains width
00045   // and height), we need to prepare the input image accordingly Typically the
00046   // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
00047   // This can be hardcoded into the application, or we can parse the width and
00048   // height from the solver filename
00049
00050   // Try to use regex capture to extract width and height from solver name
00051   std::smatch width_height;
00052   if (std::regex_match(solver_face_detect, width_height,
00053                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00054     recommended_width = std::stoi(width_height[1]);
00055     recommended_height = std::stoi(width_height[2]);
00056   } else {
00058     // Solvers without width and height in name can accept dynamic input
00059     // image size. For best results we recommend to scale the input image to
00060     // a resolution recommended by the sfeFaceDetectInputSize function
00061
00062     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00063     // for given min_face_size/max_face_size
00064     SFEDetectionInputSize input_size{};
00065     SFEError error = sfeFaceDetectInputSize(
00066         image,
00067         SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00068         face_size_min, face_size_max, &input_size);
00069     utils::checkError(error);
00070     recommended_width = input_size.width;
00071     recommended_height = input_size.height;
00073   };
00074 }
```

9.11.1.2 main()

```
int main (
              int argc,
              char * argv[] )
```

Main fuction is used to parse command line arguments.
[Image Load]
[Image Load]
[Image Resize]
[Image Resize]
[Face Detect]
[Face Detect]
[Face Landmarks]
[Face Landmarks]
[Face Liveness]
[Face Liveness]
[Face Size]
[Face Size]
[Face Mask Confidence]
[Face Mask Confidence]
[Face Area]
[Face Area]
[Face Head Pose]
[Face Head Pose]
[Face Quality]
[Face Quality]
Definition at line 143 of file example_face_liveness.cpp.

```
00143                                    {
00144
00145   { // STAGE: 0 Parse command line arguments
00146     // Map to store argument values
00147     std::unordered_map<std::string, std::string> args;
00148
00149     // Parse command line arguments
00150     for (int i = 1; i < argc; ++i) {
00151       std::string arg = argv[i];
00152       if (arg[0] == '-') {
```

```
00153         if (arg == "-h") {
00154           printHelp();
00155           return 0;
00156         }
00157         // Check if there's a next argument and it isn't another option
00158         if (i + 1 < argc && argv[i + 1][0] != '-') {
00159           args[arg] = argv[++i];
00160         } else {
00161           std::cerr « "Option " « arg « " requires a value." « std::endl;
00162           return 1;
00163         }
00164       } else {
00165         std::cerr « "Unknown option: " « arg « std::endl;
00166         return 1;
00167       }
00168     }
00169
00170     // Assign values to variables based on parsed arguments
00171     if (args.count("-p"))
00172       image_probe = args["-p"];
00173     if (args.count("-d"))
00174       solver_face_detect = args["-d"];
00175     if (args.count("-l"))
00176       solver_face_landmarks = args["-l"];
00177     if (args.count("-i"))
00178       solver_face_liveness = args["-i"];
00179     if (args.count("-t"))
00180       detection_threshold = std::stof(args["-t"]);
00181     if (args.count("-m"))
00182       face_size_min = std::stoi(args["-m"]);
00183     if (args.count("-x"))
00184       face_size_max = std::stoi(args["-x"]);
00185
00186     utils::printFormatted("PARAMETERS");
00187     // Print resource names
00188     std::cout « "Probe image: " « image_probe « std::endl;
00189
00190     // Print solver names
00191     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00192     std::cout « "Face landmarks solver: " « solver_face_landmarks
00193               « std::endl;
00194     std::cout « "Face liveness solver: " « solver_face_liveness « std::endl;
00195
00196     // Print other options
00197     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00198     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00199     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00200   }
00201
00202   utils::printToolkitInfo();
00203
00204   SFEError error{};
00205
00206   SFESolver detector_solver{};
00207   DEFER(sfeSolverFree(detector_solver));
00208   SFESolver landmarks_solver{};
00209   DEFER(sfeSolverFree(landmarks_solver));
00210   SFESolver liveness_solver{};
00211   DEFER(sfeSolverFree(liveness_solver));
00212   { // STAGE 1: loading solvers
00213     utils::printFormatted("LOADING solvers");
00214     // Initialize face detection solver
00215     error = sfeSolverCreate(
00216         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00217         SOLVER_PARAMETERS.size(), &detector_solver);
00218     utils::checkError(error);
00219
00220     // Initialize landmarks detection solver
00221     error = sfeSolverCreate(
00222         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00223         SOLVER_PARAMETERS.size(), &landmarks_solver);
00224     utils::checkError(error);
00225
00226     // Initialize face liveness solver
00227     error = sfeSolverCreate(
00228         solver_face_liveness.c_str(), SOLVER_PARAMETERS.data(),
00229         SOLVER_PARAMETERS.size(), &liveness_solver);
00230     utils::checkError(error);
00231   }
00232
00233   SFEImage image{};
00234   DEFER(sfeImageFree(image));
00235   SFEImage resized_image{};
00236   DEFER(sfeImageFree(resized_image));
00237   { // STAGE 2: Load image
00238
00239     utils::printFormatted("FACE DETECTION");
```

```
00241     // Load image data from file
00242     auto image_data = utils::readFile(image_probe);
00243
00244     // Decode image from data
00245     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00246     utils::checkError(error);
00248
00249     size_t recommended_width{};
00250     size_t recommended_height{};
00251
00252     // Calculate optimal input image size for face detection. Image will be
00253     // resized to recommended size for optimal performance.
00254     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00255                             face_size_max, recommended_width,
00256                             recommended_height);
00257
00259     // Resize image to recommended size
00260     // NOTE: This function will resize the image without preserving the aspect
00261     // ratio of the image.
00262     error = sfeImageResize(image, recommended_width, recommended_height,
00263                            &resized_image);
00264     utils::checkError(error);
00266   }
00268   SFEDetection detected_face = {};
00269   { // STAGE 3: Detect face in the image
00270     size_t detection_count = 1;
00271     // Detect face in the image
00272     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00273                       &detected_face, &detection_count);
00274     utils::checkError(error);
00275
00276     if (detection_count == 0) {
00277       std::cout « "No face detected in the probe image." « std::endl;
00278       return 0;
00279     } else {
00280       std::cout « "Found " « detection_count
00281                 « " face(s) in the probe image. Using the face with highest "
00282                    "confidence."
00283                 « std::endl;
00284     }
00285   }
00287
00289   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00290   { // STAGE 4: Get face landmarks
00291
00292     // Use landmarks detection solver to get relevant landmarks for
00293     // the detected face
00294     error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00295                              landmarks.data());
00296     utils::checkError(error);
00297   }
00299
00301   SFEFaceLiveness liveness = {};
00302   { // STAGE 5: Liveness check
00303     utils::printFormatted("LIVENESS CHECK");
00304     error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
00305                                    &liveness);
00306     utils::checkError(error);
00307   }
00309
00310   { // STAGE 6: Print the liveness score
00311     std::cout « "Liveness score is " « liveness.score;
00312     // Recommended threshold for liveness detection, see EER threshold for distant fast mode in the
      documentation
00313     static const float LIVENESS_THRESHOLD = 0.81f;
00314     if (liveness.score < LIVENESS_THRESHOLD) {
00315       std::cout « " which is below " « LIVENESS_THRESHOLD
00316                 « " threshold. Face is spoof." « std::endl;
00317     } else {
00318       std::cout « " which is above " « LIVENESS_THRESHOLD
00319                 « " threshold. Face is genuine." « std::endl;
00320     }
00321   }
00322
00323   // Optional face attributes to check for further analysis. See the
00324   // documentation for more information.
00325   float face_size = 0;
00326   float mask_confidence;
00327   SFEFaceArea face_area = {};
00328   SFEFaceHeadPose head_pose{};
00329   SFEFaceQualityAttributes quality_attributes{};
00330   { // STAGE 6: (optional) Face attributes
00331     utils::printFormatted("FACE ATTRIBUTES");
00332
00334     error = sfeFaceSize(image, landmarks.data(), &face_size);
00335     utils::checkError(error);
00337
```

```
00339     error = sfeFaceMaskConfidence(landmarks.data(), &mask_confidence);
00340     utils::checkError(error);
00342
00344     error = sfeFaceArea(image, landmarks.data(), &face_area);
00345     utils::checkError(error);
00347
00349     error = sfeFaceHeadPose(image, landmarks.data(), &head_pose);
00350     utils::checkError(error);
00352
00354     error =
00355         sfeFaceQualityAttributes(image, landmarks.data(), &quality_attributes);
00356     utils::checkError(error);
00358   }
00359
00360   { // STAGE 8: Print the face attributes
00361     printFaceDetectionInfo(detected_face);
00362     printFaceSize(face_size);
00363     printMaskConfidence(mask_confidence);
00364     prinFaceAreaInfo(face_area);
00365     printFaceHeadPose(head_pose);
00366     printFaceQualityAttributes(quality_attributes);
00367   }
00368
00369   { // STAGE 9: Annotate the image
00370     // Render bounding box
00371     std::stringstream label;
00372     label « "Face" « std::fixed « std::setprecision(2)
00373           « " d:" « detected_face.confidence « " m:" « mask_confidence
00374           « " l:" « liveness.score;
00375
00376     annotate::labelBox(image, label.str(), detected_face.bounding_box, annotate::WHITE,
00377                        annotate::GREEN);
00378
00379     // Render landmarks
00380     for (auto &landmark : landmarks) {
00381       auto x = landmark.x * image.width;
00382       auto y = landmark.y * image.height;
00383       annotate::circle(image, x, y, 3, annotate::YELLOW);
00384     }
00385
00386     // Save annotated image
00387     size_t size = image.width * image.height * 3;
00388     auto png_file = std::vector<unsigned char>(size);
00389     error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00390     utils::checkError(error);
00391     png_file.resize(size);
00392     utils::saveFile("face_liveness.png", png_file);
00393
00394     std::cout « std::endl;
00395     std::cout « "Annotated image saved to face_liveness.png" « std::endl;
00396   }
00397
00398   utils::printFormatted("FINISHED");
00399 }
```

9.11.1.3 prinFaceAreaInfo()

```
void prinFaceAreaInfo (
            SFEFaceArea & face_area )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 89 of file example_face_liveness.cpp.

```
00089                                                 {
00090   std::cout « "SFEFaceArea{ " « std::endl;
00091   std::cout « "  size: " « face_area.size « " [px]" « std::endl;
00092   std::cout « "  area: " « face_area.area « std::endl;
00093   std::cout « "  area_in_image: " « face_area.area_in_image « std::endl;
00094   std::cout « "}" « std::endl;
00095   std::cout « std::endl;
00096 }
```

9.11.1.4 printFaceDetectionInfo()

```
void printFaceDetectionInfo (
            SFEDetection & detected_face )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 76 of file example_face_liveness.cpp.

```
00076                                                                    {
00077   std::cout « "SFEDetection{ " « std::endl;
00078   std::cout « "  confidence: " « detected_face.confidence « std::endl;
00079   std::cout « "  SFEBoundingBox{ " « std::endl;
00080   std::cout « "    x: " « detected_face.bounding_box.x « std::endl;
00081   std::cout « "    y: " « detected_face.bounding_box.y « std::endl;
00082   std::cout « "    width: " « detected_face.bounding_box.width « std::endl;
00083   std::cout « "    height: " « detected_face.bounding_box.height « std::endl;
00084   std::cout « "    }" « std::endl;
00085   std::cout « "  } " « std::endl;
00086   std::cout « std::endl;
00087 }
```

9.11.1.5 printFaceHeadPose()

```
void printFaceHeadPose (
            SFEFaceHeadPose & head_pose )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 98 of file example_face_liveness.cpp.

```
00098                                                                {
00099   std::cout « "SFEFaceHeadPose{" « std::endl;
00100   std::cout « "  pitch: " « head_pose.pitch « std::endl;
00101   std::cout « "  yaw: " « head_pose.yaw « std::endl;
00102   std::cout « "  roll: " « head_pose.roll « std::endl;
00103   std::cout « "} " « std::endl;
00104   std::cout « std::endl;
00105 }
```

9.11.1.6 printFaceQualityAttributes()

```
void printFaceQualityAttributes (
            SFEFaceQualityAttributes & quality_attributes )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 118 of file example_face_liveness.cpp.

```
00118                                                                                {
00119   std::cout « "SFEFaceQualityAttributes{" « std::endl;
00120   std::cout « "  sharpness: " « quality_attributes.sharpness « std::endl;
00121   std::cout « "  brightness: " « quality_attributes.brightness « std::endl;
00122   std::cout « "  contrast: " « quality_attributes.contrast « std::endl;
00123   std::cout « "  unique_intensity_levels: "
00124             « quality_attributes.unique_intensity_levels « std::endl;
00125   std::cout « "} " « std::endl;
00126   std::cout « std::endl;
00127 }
```

9.11.1.7 printFaceSize()

```
void printFaceSize (
            size_t face_size )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 107 of file example_face_liveness.cpp.

```
00107                                                {
00108   std::cout « "Face size: " « face_size « " [px]" « std::endl;
00109   std::cout « std::endl;
00110 }
```

9.11.1.8 printHelp()

```
void printHelp ( )
```

Definition at line 129 of file example_face_liveness.cpp.

```
00129                {
```

```
00130   std::cout « "Help: Usage of the program." « std::endl;
00131   std::cout « "Options:" « std::endl;
00132   std::cout « "-h: Display help." « std::endl;
00133   std::cout « "-p: Probe image file." « std::endl;
00134   std::cout « "-d: Path to detector solver." « std::endl;
00135   std::cout « "-l: Path to landmarks solver." « std::endl;
00136   std::cout « "-i: Path to liveness solver." « std::endl;
00137   std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
00138   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00139   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00140 }
```

9.11.1.9 printMaskConfidence()

```
void printMaskConfidence (
            float mask_confidence )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 112 of file example_face_liveness.cpp.

```
00112                                                   {
00113   std::cout « "Mask confidence: " « mask_confidence « std::endl;
00114   std::cout « std::endl;
00115 }
```

9.11.2 Variable Documentation

9.11.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 30 of file example_face_liveness.cpp.

9.11.2.2 face_size_max

```
size_t face_size_max = 170
```

Definition at line 28 of file example_face_liveness.cpp.

9.11.2.3 face_size_min

```
size_t face_size_min = 28
```

Required size of the face to be detected.
Definition at line 27 of file example_face_liveness.cpp.

9.11.2.4 image_probe

```
std::string image_probe = ¨assets/face/images/obiwan0.png¨
```

Definition at line 17 of file example_face_liveness.cpp.

9.11.2.5 solver_face_detect

```
std::string solver_face_detect = SOLVER_FACE_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.
Definition at line 20 of file example_face_liveness.cpp.

9.11.2.6 solver_face_landmarks

```
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS
```

Definition at line 21 of file example_face_liveness.cpp.

9.11.2.7 solver_face_liveness

```
std::string solver_face_liveness = SOLVER_FACE_LIVENESS
```

Examples

example_face_liveness.cpp.

Definition at line 22 of file example_face_liveness.cpp.

9.11.2.8 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Definition at line 24 of file example_face_liveness.cpp.

9.12 example_face_liveness.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_face.h"
00007 #include <iomanip>
00008 #include <regex>
00009 #include <sstream>
00010 #include <vector>
00011
00012 #include "annotate.hpp"
00013 #include "solvers.h"
00014 #include "utils.hpp"
00015 #include <unordered_map>
00016
00017 std::string image_probe = "assets/face/images/obiwan0.png";
00018
00020 std::string solver_face_detect = SOLVER_FACE_DETECT;
00021 std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
00022 std::string solver_face_liveness = SOLVER_FACE_LIVENESS;
00023
00024 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00025
00027 size_t face_size_min = 28;
00028 size_t face_size_max = 170;
00029
00030 float detection_threshold = 0.1f;
00031
00038 void getRecommendedImageSize(const std::string &solver_face_detect,
00039                              SFEImage &image, const size_t face_size_min,
00040                              const size_t face_size_max,
00041                              size_t &recommended_width,
00042                              size_t &recommended_height) {
00043
00044   // If the face detection solver requires static input (filename contains width
00045   // and height), we need to prepare the input image accordingly Typically the
00046   // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
00047   // This can be hardcoded into the application, or we can parse the width and
00048   // height from the solver filename
00049
00050   // Try to use regex capture to extract width and height from solver name
00051   std::smatch width_height;
00052   if (std::regex_match(solver_face_detect, width_height,
00053                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00054     recommended_width = std::stoi(width_height[1]);
00055     recommended_height = std::stoi(width_height[2]);
00056   } else {
00058     // Solvers without width and height in name can accept dynamic input
00059     // image size. For best results we recommend to scale the input image to
00060     // a resolution recommended by the sfeFaceDetectInputSize function
00061
00062     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00063     // for given min_face_size/max_face_size
00064     SFEDetectionInputSize input_size{};
00065     SFEError error = sfeFaceDetectInputSize(
00066         image,
00067         SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00068         face_size_min, face_size_max, &input_size);
00069     utils::checkError(error);
00070     recommended_width = input_size.width;
00071     recommended_height = input_size.height;
00073   };
00074 }
00075
00076 inline void printFaceDetectionInfo(SFEDetection &detected_face) {
00077   std::cout « "SFEDetection{ " « std::endl;
00078   std::cout « "  confidence: " « detected_face.confidence « std::endl;
00079   std::cout « "  SFEBoundingBox{ " « std::endl;
00080   std::cout « "    x: " « detected_face.bounding_box.x « std::endl;
```

```
00081   std::cout « "    y: " « detected_face.bounding_box.y « std::endl;
00082   std::cout « "    width: " « detected_face.bounding_box.width « std::endl;
00083   std::cout « "    height: " « detected_face.bounding_box.height « std::endl;
00084   std::cout « "    }" « std::endl;
00085   std::cout « "  } " « std::endl;
00086   std::cout « std::endl;
00087 }
00088
00089 inline void prinFaceAreaInfo(SFEFaceArea &face_area) {
00090   std::cout « "SFEFaceArea{ " « std::endl;
00091   std::cout « "  size: " « face_area.size « " [px]" « std::endl;
00092   std::cout « "  area: " « face_area.area « std::endl;
00093   std::cout « "  area_in_image: " « face_area.area_in_image « std::endl;
00094   std::cout « "}" « std::endl;
00095   std::cout « std::endl;
00096 }
00097
00098 inline void printFaceHeadPose(SFEFaceHeadPose &head_pose) {
00099   std::cout « "SFEFaceHeadPose{" « std::endl;
00100   std::cout « "  pitch: " « head_pose.pitch « std::endl;
00101   std::cout « "  yaw: " « head_pose.yaw « std::endl;
00102   std::cout « "  roll: " « head_pose.roll « std::endl;
00103   std::cout « "} " « std::endl;
00104   std::cout « std::endl;
00105 }
00106
00107 inline void printFaceSize(size_t face_size) {
00108   std::cout « "Face size: " « face_size « " [px]" « std::endl;
00109   std::cout « std::endl;
00110 }
00111
00112 inline void printMaskConfidence(float mask_confidence) {
00113   std::cout « "Mask confidence: " « mask_confidence « std::endl;
00114   std::cout « std::endl;
00115 }
00116
00117 inline void
00118 printFaceQualityAttributes(SFEFaceQualityAttributes &quality_attributes) {
00119   std::cout « "SFEFaceQualityAttributes{" « std::endl;
00120   std::cout « "  sharpness: " « quality_attributes.sharpness « std::endl;
00121   std::cout « "  brightness: " « quality_attributes.brightness « std::endl;
00122   std::cout « "  contrast: " « quality_attributes.contrast « std::endl;
00123   std::cout « "  unique_intensity_levels: "
00124             « quality_attributes.unique_intensity_levels « std::endl;
00125   std::cout « "} " « std::endl;
00126   std::cout « std::endl;
00127 }
00128
00129 void printHelp() {
00130   std::cout « "Help: Usage of the program." « std::endl;
00131   std::cout « "Options:" « std::endl;
00132   std::cout « "-h: Display help." « std::endl;
00133   std::cout « "-p: Probe image file." « std::endl;
00134   std::cout « "-d: Path to detector solver." « std::endl;
00135   std::cout « "-l: Path to landmarks solver." « std::endl;
00136   std::cout « "-i: Path to liveness solver." « std::endl;
00137   std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
00138   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00139   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00140 }
00141
00143 int main(int argc, char *argv[]) {
00144
00145   { // STAGE: 0 Parse command line arguments
00146     // Map to store argument values
00147     std::unordered_map<std::string, std::string> args;
00148
00149     // Parse command line arguments
00150     for (int i = 1; i < argc; ++i) {
00151       std::string arg = argv[i];
00152       if (arg[0] == '-') {
00153         if (arg == "-h") {
00154           printHelp();
00155           return 0;
00156         }
00157         // Check if there's a next argument and it isn't another option
00158         if (i + 1 < argc && argv[i + 1][0] != '-') {
00159           args[arg] = argv[++i];
00160         } else {
00161           std::cerr « "Option " « arg « " requires a value." « std::endl;
00162           return 1;
00163         }
00164       } else {
00165         std::cerr « "Unknown option: " « arg « std::endl;
00166         return 1;
00167       }
00168     }
```

```
00169
00170     // Assign values to variables based on parsed arguments
00171     if (args.count("-p"))
00172       image_probe = args["-p"];
00173     if (args.count("-d"))
00174       solver_face_detect = args["-d"];
00175     if (args.count("-l"))
00176       solver_face_landmarks = args["-l"];
00177     if (args.count("-i"))
00178       solver_face_liveness = args["-i"];
00179     if (args.count("-t"))
00180       detection_threshold = std::stof(args["-t"]);
00181     if (args.count("-m"))
00182       face_size_min = std::stoi(args["-m"]);
00183     if (args.count("-x"))
00184       face_size_max = std::stoi(args["-x"]);
00185
00186     utils::printFormatted("PARAMETERS");
00187     // Print resource names
00188     std::cout « "Probe image: " « image_probe « std::endl;
00189
00190     // Print solver names
00191     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00192     std::cout « "Face landmarks solver: " « solver_face_landmarks
00193               « std::endl;
00194     std::cout « "Face liveness solver: " « solver_face_liveness « std::endl;
00195
00196     // Print other options
00197     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00198     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00199     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00200   }
00201
00202   utils::printToolkitInfo();
00203
00204   SFEError error{};
00205
00206   SFESolver detector_solver{};
00207   DEFER(sfeSolverFree(detector_solver));
00208   SFESolver landmarks_solver{};
00209   DEFER(sfeSolverFree(landmarks_solver));
00210   SFESolver liveness_solver{};
00211   DEFER(sfeSolverFree(liveness_solver));
00212   { // STAGE 1: loading solvers
00213     utils::printFormatted("LOADING solvers");
00214     // Initialize face detection solver
00215     error = sfeSolverCreate(
00216         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00217         SOLVER_PARAMETERS.size(), &detector_solver);
00218     utils::checkError(error);
00219
00220     // Initialize landmarks detection solver
00221     error = sfeSolverCreate(
00222         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00223         SOLVER_PARAMETERS.size(), &landmarks_solver);
00224     utils::checkError(error);
00225
00226     // Initialize face liveness solver
00227     error = sfeSolverCreate(
00228         solver_face_liveness.c_str(), SOLVER_PARAMETERS.data(),
00229         SOLVER_PARAMETERS.size(), &liveness_solver);
00230     utils::checkError(error);
00231   }
00232
00233   SFEImage image{};
00234   DEFER(sfeImageFree(image));
00235   SFEImage resized_image{};
00236   DEFER(sfeImageFree(resized_image));
00237   { // STAGE 2: Load image
00238
00239     utils::printFormatted("FACE DETECTION");
00241     // Load image data from file
00242     auto image_data = utils::readFile(image_probe);
00243
00244     // Decode image from data
00245     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00246     utils::checkError(error);
00248
00249     size_t recommended_width{};
00250     size_t recommended_height{};
00251
00252     // Calculate optimal input image size for face detection. Image will be
00253     // resized to recommended size for optimal performance.
00254     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00255                             face_size_max, recommended_width,
00256                             recommended_height);
00257
```

```
00259     // Resize image to recommended size
00260     // NOTE: This function will resize the image without preserving the aspect
00261     // ratio of the image.
00262     error = sfeImageResize(image, recommended_width, recommended_height,
00263                            &resized_image);
00264     utils::checkError(error);
00266   }
00268   SFEDetection detected_face = {};
00269   { // STAGE 3: Detect face in the image
00270     size_t detection_count = 1;
00271     // Detect face in the image
00272     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00273                       &detected_face, &detection_count);
00274     utils::checkError(error);
00275
00276     if (detection_count == 0) {
00277       std::cout « "No face detected in the probe image." « std::endl;
00278       return 0;
00279     } else {
00280       std::cout « "Found " « detection_count
00281                 « " face(s) in the probe image. Using the face with highest "
00282                    "confidence."
00283                 « std::endl;
00284     }
00285   }
00287
00289   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00290   { // STAGE 4: Get face landmarks
00291
00292     // Use landmarks detection solver to get relevant landmarks for
00293     // the detected face
00294     error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
00295                              landmarks.data());
00296     utils::checkError(error);
00297   }
00299
00301   SFEFaceLiveness liveness = {};
00302   { // STAGE 5: Liveness check
00303     utils::printFormatted("LIVENESS CHECK");
00304     error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
00305                                    &liveness);
00306     utils::checkError(error);
00307   }
00309
00310   { // STAGE 6: Print the liveness score
00311     std::cout « "Liveness score is " « liveness.score;
00312     // Recommended threshold for liveness detection, see EER threshold for distant fast mode in the
      documentation
00313     static const float LIVENESS_THRESHOLD = 0.81f;
00314     if (liveness.score < LIVENESS_THRESHOLD) {
00315       std::cout « " which is below " « LIVENESS_THRESHOLD
00316                 « " threshold. Face is spoof." « std::endl;
00317     } else {
00318       std::cout « " which is above " « LIVENESS_THRESHOLD
00319                 « " threshold. Face is genuine." « std::endl;
00320     }
00321   }
00322
00323   // Optional face attributes to check for further analysis. See the
00324   // documentation for more information.
00325   float face_size = 0;
00326   float mask_confidence;
00327   SFEFaceArea face_area = {};
00328   SFEFaceHeadPose head_pose{};
00329   SFEFaceQualityAttributes quality_attributes{};
00330   { // STAGE 6: (optional) Face attributes
00331     utils::printFormatted("FACE ATTRIBUTES");
00332
00334     error = sfeFaceSize(image, landmarks.data(), &face_size);
00335     utils::checkError(error);
00337
00339     error = sfeFaceMaskConfidence(landmarks.data(), &mask_confidence);
00340     utils::checkError(error);
00342
00344     error = sfeFaceArea(image, landmarks.data(), &face_area);
00345     utils::checkError(error);
00347
00349     error = sfeFaceHeadPose(image, landmarks.data(), &head_pose);
00350     utils::checkError(error);
00352
00354     error =
00355         sfeFaceQualityAttributes(image, landmarks.data(), &quality_attributes);
00356     utils::checkError(error);
00358   }
00359
00360   { // STAGE 8: Print the face attributes
00361     printFaceDetectionInfo(detected_face);
```

```
00362     printFaceSize(face_size);
00363     printMaskConfidence(mask_confidence);
00364     prinFaceAreaInfo(face_area);
00365     printFaceHeadPose(head_pose);
00366     printFaceQualityAttributes(quality_attributes);
00367   }
00368
00369   { // STAGE 9: Annotate the image
00370     // Render bounding box
00371     std::stringstream label;
00372     label « "Face" « std::fixed « std::setprecision(2)
00373           « " d:" « detected_face.confidence « " m:" « mask_confidence
00374           « " l:" « liveness.score;
00375
00376     annotate::labelBox(image, label.str(), detected_face.bounding_box, annotate::WHITE,
00377                        annotate::GREEN);
00378
00379     // Render landmarks
00380     for (auto &landmark : landmarks) {
00381       auto x = landmark.x * image.width;
00382       auto y = landmark.y * image.height;
00383       annotate::circle(image, x, y, 3, annotate::YELLOW);
00384     }
00385
00386     // Save annotated image
00387     size_t size = image.width * image.height * 3;
00388     auto png_file = std::vector<unsigned char>(size);
00389     error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00390     utils::checkError(error);
00391     png_file.resize(size);
00392     utils::saveFile("face_liveness.png", png_file);
00393
00394     std::cout « std::endl;
00395     std::cout « "Annotated image saved to face_liveness.png" « std::endl;
00396   }
00397
00398   utils::printFormatted("FINISHED");
00399 }
```

9.13 D:/glab/1512fd1724a/1/example/example_face_track.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_face.h¨
#include <ostream>
#include <regex>
#include <vector>
#include <iomanip>
#include <sstream>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- void getRecommendedImageSize (const std::string &solver_face_detect, SFEImage &image, const size_t face_size_min, const size_t face_size_max, size_t &recommended_width, size_t &recommended_height)

  Get recommended image size for face detection.
- void printHelp ()
- std::vector< SFEDetection > detectFaces (SFEImage &image, SFESolver &detector_solver)

  Detect faces in the image.
- std::ostream & operator<< (std::ostream &os, const SFEEntity &entity)

  Print UUID.
- std::ostream & operator<< (std::ostream &os, const SFETrackedState &state)

  Print tracked face state.
- std::ostream & operator<< (std::ostream &os, const SFETracked &tracked)

Print tracked face.

- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::vector< T > &vec)

  Print a container of printable objects.
- void frame (std::vector< SFEDetection > &detections, SFETracker tracker)

  Print out the results of the face tracking.
- int main (int argc, char *argv[ ])

  Main function is used to parse command line arguments.

Variables

- std::string image_probe = "assets/face/images/obiwan0.png"
- std::string solver_face_detect = SOLVER_FACE_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- size_t face_size_min = 28

  Required size of the face to be detected.
- size_t face_size_max = 170
- float detection_threshold = 0.1f

9.13.1 Function Documentation

9.13.1.1 detectFaces()

```
std::vector< SFEDetection > detectFaces (
            SFEImage & image,
            SFESolver & detector_solver )
```

Detect faces in the image.

Examples

example_face_track.cpp.

Definition at line 86 of file example_face_track.cpp.

```
00087                                                                    {
00088
00089   SFEImage resized_image{};
00090   DEFER(sfeImageFree(resized_image));
00091   SFEError error{};
00092   { // STAGE 1 Load image
00093     size_t recommended_width{};
00094     size_t recommended_height{};
00095
00096     // Calculate optimal input image size for face detection. Image will be
00097     // resized to recommended size for optimal performance.
00098     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00099                             face_size_max, recommended_width,
00100                             recommended_height);
00101
00102     // Resize image to recommended size
00103     // NOTE: This function will resize the image without preserving the aspect
00104     // ratio of the image.
00105
00106     error = sfeImageResize(image, recommended_width, recommended_height,
00107                            &resized_image);
00108     utils::checkError(error);
00109   }
00110
00111   size_t detection_count = 10;
00112   std::vector<SFEDetection> detected_faces(detection_count);
00113   { // STAGE 2: Detect face in the image
00114
00115     // Detect faces in the image
00116     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00117                           detected_faces.data(), &detection_count);
00118     utils::checkError(error);
00119
00120     if (detection_count == 0) {
00121       std::ostringstream oss;
00122       oss << "Error: No face detected in the image ";
```

```
00123       throw std::runtime_error(oss.str());
00124     }
00125     detected_faces.resize(detection_count);
00126
00127     std::cout « "Detected " « detection_count « " faces in the image."
00128               « std::endl;
00129   }
00130   return detected_faces;
00131 }
```

9.13.1.2 frame()

```
void frame (
            std::vector< SFEDetection > & detections,
            SFETracker tracker )
```

Print out the results of the face tracking.

Examples

example_face_track.cpp.

Definition at line 185 of file example_face_track.cpp.

```
00185                                                                    {
00186   size_t reserved_size = 10;
00187   size_t tracked_faces_count = reserved_size;
00188   size_t lost_faces_count = reserved_size;
00189   size_t removed_faces_count = reserved_size;
00190
00191   std::vector<SFETracked> tracked_faces(tracked_faces_count);
00192   std::vector<SFEEntity> lost_faces(lost_faces_count);
00193   std::vector<SFEEntity> removed_faces(removed_faces_count);
00194
00195   // Update tracker with detected faces
00196   SFEError error =
00197       sfeDetectionTrackerUpdate(tracker, detections.data(), detections.size(),
00198                                 tracked_faces.data(), &tracked_faces_count);
00199   utils::checkError(error);
00200   tracked_faces.resize(tracked_faces_count);
00201
00202   // Get lost faces
00203   error = sfeDetectionTrackerLost(tracker, lost_faces.data(), &lost_faces_count);
00204   utils::checkError(error);
00205   lost_faces.resize(lost_faces_count);
00206
00207   // Get removed faces
00208   error = sfeDetectionTrackerRemoved(tracker, removed_faces.data(),
00209                                      &removed_faces_count);
00210   utils::checkError(error);
00211   removed_faces.resize(removed_faces_count);
00212
00213   // Print out frame results
00214   std::cout « "Tracked" « std::endl;
00215   std::cout « tracked_faces;
00216   std::cout « "Lost" « std::endl;
00217   std::cout « lost_faces;
00218   std::cout « "Removed" « std::endl;
00219   std::cout « removed_faces;
00220 }
```

9.13.1.3 getRecommendedImageSize()

```
void getRecommendedImageSize (
            const std::string & solver_face_detect,
            SFEImage & image,
            const size_t face_size_min,
            const size_t face_size_max,
            size_t & recommended_width,
            size_t & recommended_height )
```

Get recommended image size for face detection.

Parameters

solver_face_detect	name of the face detection solver
face_size_min	Desired minimal size of detected face

Parameters

face_size_max	Desired maximal size of detected face
recommended_width	Recommended image width
recommended_height	Recommended image height

Definition at line 38 of file example_face_track.cpp.

```
00042                                                                    {
00043
00044   // If the face detection solver requires static input (filename contains width
00045   // and height), we need to prepare the input image accordingly Typically the
00046   // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
00047   // This can be hardcoded into the application, or we can parse the width and
00048   // height from the solver filename
00049
00050   // Try to use regex capture to extract width and height from solver name
00051   std::smatch width_height;
00052   if (std::regex_match(solver_face_detect, width_height,
00053                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00054     recommended_width = std::stoi(width_height[1]);
00055     recommended_height = std::stoi(width_height[2]);
00056   } else {
00057     // Solvers without width and height in name can accept dynamic input
00058     // image size. For best results we recommend to scale the input image to
00059     // a resolution recommended by the sfeFaceDetectInputSize function
00060
00061     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00062     // for given min_face_size/max_face_size
00063     SFEDetectionInputSize input_size{};
00064     SFEError error = sfeFaceDetectInputSize(
00065         image,
00066         SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00067         face_size_min, face_size_max, &input_size);
00068     utils::checkError(error);
00069     recommended_width = input_size.width;
00070     recommended_height = input_size.height;
00071   };
00072 }
```

9.13.1.4 main()

```
int main (
            int argc,
            char * argv[] )
```

Main function is used to parse command line arguments.

Definition at line 223 of file example_face_track.cpp.

```
00223                                    {
00224
00225   { // STAGE 0: Parse command line arguments
00226     // Map to store argument values
00227     std::unordered_map<std::string, std::string> args;
00228
00229     // Parse command line arguments
00230     for (int i = 1; i < argc; ++i) {
00231       std::string arg = argv[i];
00232       if (arg[0] == '-') {
00233         if (arg == "-h") {
00234           printHelp();
00235           return 0;
00236         }
00237         // Check if there's a next argument and it isn't another option
00238         if (i + 1 < argc && argv[i + 1][0] != '-') {
00239           args[arg] = argv[++i];
00240         } else {
00241           std::cerr << "Option " << arg << " requires a value." << std::endl;
00242           return 1;
00243         }
00244       } else {
00245         std::cerr << "Unknown option: " << arg << std::endl;
00246         return 1;
00247       }
00248     }
00249
00250     // Assign values to variables based on parsed arguments
00251     if (args.count("-p"))
00252       image_probe = args["-p"];
00253     if (args.count("-d"))
00254       solver_face_detect = args["-d"];
00255     if (args.count("-t"))
00256       detection_threshold = std::stof(args["-t"]);
```

```
00257     if (args.count("-m"))
00258       face_size_min = std::stoi(args["-m"]);
00259     if (args.count("-x"))
00260       face_size_max = std::stoi(args["-x"]);
00261
00262     utils::printFormatted("EXAMPLE PARAMETERS");
00263     // Print resource names
00264     std::cout « "Probe image: " « image_probe « std::endl;
00265
00266     // Print solver names
00267     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00268
00269     // Print other options
00270     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00271     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00272     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00273   }
00274
00275   utils::printToolkitInfo();
00276
00277   SFEError error{};
00278
00279   SFESolver detector_solver{};
00280   DEFER(sfeSolverFree(detector_solver));
00281   { // STAGE 1: loading solvers
00282     utils::printFormatted("LOADING SOLVERS");
00283     // Initialize face detection solver
00284     error = sfeSolverCreate(
00285         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00286         SOLVER_PARAMETERS.size(), &detector_solver);
00287     utils::checkError(error);
00288   }
00289
00290   std::vector<SFEDetection> detected_faces;
00291   { // STAGE 2: Detect faces
00292     utils::printFormatted("DETECT FACES");
00293
00294     SFEImage image{};
00295     DEFER(sfeImageFree(image));
00296     { // STAGE 2.1: Load image
00297       // Load image data from file
00298       auto image_data = utils::readFile(image_probe);
00299
00300       // Decode image from data
00301       error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00302       utils::checkError(error);
00303     }
00304
00305     SFEDetection detected_face = {};
00306     { // STAGE 2.2: Detect face in the image
00307       // Detect a face in the image
00308       detected_faces = detectFaces(image, detector_solver);
00309     }
00310   }
00311
00312   SFETracker tracker{};
00313   DEFER(sfeDetectionTrackerFree(tracker));
00314   { // STAGE 3: Track faces
00315     utils::printFormatted("TRACK FACES");
00316     error = sfeDetectionTrackerCreate(0.3f, 0.1f, 0.3f, 0.1f, 1, &tracker);
00317     utils::checkError(error);
00318
00319     // We will be using the current detected faces and simulate detection across
00320     // 4 frames In a real application, you would call detectFaces() for each
00321     // frame and pass the detected faces to the tracker
00322
00323     // First frame with detected faces, all should track as NEW
00324     utils::printFormatted("FRAME 1");
00325     frame(detected_faces, tracker);
00326
00327     // Second frame with the same UUIDs, all should track as TRACKED
00328     utils::printFormatted("FRAME 2");
00329     frame(detected_faces, tracker);
00330
00331     // Simulate a lost detection
00332     detected_faces.clear();
00333
00334     // We should see the lost UUIDs
00335     utils::printFormatted("FRAME 3");
00336     frame(detected_faces, tracker);
00337
00338     // We should see the removed UUIDs
00339     utils::printFormatted("FRAME 4");
00340     frame(detected_faces, tracker);
00341   }
00342   utils::printFormatted("FINISHED");
00343 }
```

9.13.1.5 operator<<() [1/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const SFEEntity & entity )
```

Print UUID.

Examples

example_face_track.cpp.

Definition at line 134 of file example_face_track.cpp.

```
00134                                                                    {
00135   for (size_t i = 0; i < 16; ++i) {
00136     os « std::hex « std::setw(2) « std::setfill('0')
00137        « static_cast<int>(entity.uuid[i]);
00138     if (i == 3 || i == 5 || i == 7 || i == 9) {
00139       os « '-';
00140     }
00141   }
00142   os « std::dec;
00143   return os;
00144 }
```

9.13.1.6 operator<<() [2/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const SFETracked & tracked )
```

Print tracked face.

Definition at line 169 of file example_face_track.cpp.

```
00169                                                                      {
00170   os « "Face ID: " « tracked.id « " UUID: " « tracked.uuid
00171      « " State: " « tracked.state;
00172   return os;
00173 }
```

9.13.1.7 operator<<() [3/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const SFETrackedState & state )
```

Print tracked face state.

Definition at line 147 of file example_face_track.cpp.

```
00147                                                                         {
00148   switch (state) {
00149   case SFE_TRACKED_STATE_NEW:
00150     os « "NEW";
00151     break;
00152   case SFE_TRACKED_STATE_TRACKED:
00153     os « "TRACKED";
00154     break;
00155   case SFE_TRACKED_STATE_LOST:
00156     os « "LOST";
00157     break;
00158   case SFE_TRACKED_STATE_REMOVED:
00159     os « "REMOVED";
00160     break;
00161   default:
00162     os « "UNKNOWN";
00163     break;
00164   }
00165   return os;
00166 }
```

9.13.1.8 operator<<() [4/4]

```
template<typename T >
std::ostream & operator<< (
            std::ostream & os,
            const std::vector< T > & vec )
```

Print a container of printable objects.

Definition at line 177 of file example_face_track.cpp.

```
00177                                                                    {
00178   for (auto &item : vec) {
00179     os « item « std::endl;
00180   }
00181   return os;
00182 }
```

9.13.1.9 printHelp()

```
void printHelp ( )
```

Definition at line 74 of file example_face_track.cpp.

```
00074                  {
00075   std::cout « "Help: Usage of the program." « std::endl;
00076   std::cout « "Options:" « std::endl;
00077   std::cout « "-h: Display help." « std::endl;
00078   std::cout « "-p: Probe image file." « std::endl;
00079   std::cout « "-d: Path to detector solver." « std::endl;
00080   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00081   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00082   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00083 }
```

9.13.2 Variable Documentation

9.13.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 30 of file example_face_track.cpp.

9.13.2.2 face_size_max

```
size_t face_size_max = 170
```

Definition at line 28 of file example_face_track.cpp.

9.13.2.3 face_size_min

```
size_t face_size_min = 28
```

Required size of the face to be detected.
Definition at line 27 of file example_face_track.cpp.

9.13.2.4 image_probe

```
std::string image_probe = ¨assets/face/images/obiwan0.png¨
```

Definition at line 19 of file example_face_track.cpp.

9.13.2.5 solver_face_detect

```
std::string solver_face_detect = SOLVER_FACE_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.
Definition at line 22 of file example_face_track.cpp.

9.13.2.6 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Definition at line 24 of file example_face_track.cpp.

9.14 example_face_track.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_face.h"
00007 #include <ostream>
00008 #include <regex>
00009 #include <vector>
```

```
00010
00011 #include <iomanip>
00012 #include <sstream>
00013
00014 #include "annotate.hpp"
00015 #include "solvers.h"
00016 #include "utils.hpp"
00017 #include <unordered_map>
00018
00019 std::string image_probe = "assets/face/images/obiwan0.png";
00020
00022 std::string solver_face_detect = SOLVER_FACE_DETECT;
00023
00024 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00025
00027 size_t face_size_min = 28;
00028 size_t face_size_max = 170;
00029
00030 float detection_threshold = 0.1f;
00031
00038 void getRecommendedImageSize(const std::string &solver_face_detect,
00039                              SFEImage &image, const size_t face_size_min,
00040                              const size_t face_size_max,
00041                              size_t &recommended_width,
00042                              size_t &recommended_height) {
00043
00044   // If the face detection solver requires static input (filename contains width
00045   // and height), we need to prepare the input image accordingly Typically the
00046   // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
00047   // This can be hardcoded into the application, or we can parse the width and
00048   // height from the solver filename
00049
00050   // Try to use regex capture to extract width and height from solver name
00051   std::smatch width_height;
00052   if (std::regex_match(solver_face_detect, width_height,
00053                        std::regex(".*w([0-9]+)h([0-9]+).*"))) {
00054     recommended_width = std::stoi(width_height[1]);
00055     recommended_height = std::stoi(width_height[2]);
00056   } else {
00057     // Solvers without width and height in name can accept dynamic input
00058     // image size. For best results we recommend to scale the input image to
00059     // a resolution recommended by the sfeFaceDetectInputSize function
00060
00061     // Use sfeFaceDetectInputSize to get recommended input image dimensions
00062     // for given min_face_size/max_face_size
00063     SFEDetectionInputSize input_size{};
00064     SFEError error = sfeFaceDetectInputSize(
00065         image,
00066         SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00067         face_size_min, face_size_max, &input_size);
00068     utils::checkError(error);
00069     recommended_width = input_size.width;
00070     recommended_height = input_size.height;
00071   };
00072 }
00073
00074 void printHelp() {
00075   std::cout « "Help: Usage of the program." « std::endl;
00076   std::cout « "Options:" « std::endl;
00077   std::cout « "-h: Display help." « std::endl;
00078   std::cout « "-p: Probe image file." « std::endl;
00079   std::cout « "-d: Path to detector solver." « std::endl;
00080   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00081   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00082   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00083 }
00084
00086 std::vector<SFEDetection> detectFaces(SFEImage &image,
00087                                       SFESolver &detector_solver) {
00088
00089   SFEImage resized_image{};
00090   DEFER(sfeImageFree(resized_image));
00091   SFEError error{};
00092   { // STAGE 1 Load image
00093     size_t recommended_width{};
00094     size_t recommended_height{};
00095
00096     // Calculate optimal input image size for face detection. Image will be
00097     // resized to recommended size for optimal performance.
00098     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00099                             face_size_max, recommended_width,
00100                             recommended_height);
00101
00102     // Resize image to recommended size
00103     // NOTE: This function will resize the image without preserving the aspect
00104     // ratio of the image.
00105
```

```
00106     error = sfeImageResize(image, recommended_width, recommended_height,
00107                            &resized_image);
00108     utils::checkError(error);
00109   }
00110
00111   size_t detection_count = 10;
00112   std::vector<SFEDetection> detected_faces(detection_count);
00113   { // STAGE 2: Detect face in the image
00114
00115     // Detect faces in the image
00116     error = sfeDetect(detector_solver, resized_image, detection_threshold,
00117                       detected_faces.data(), &detection_count);
00118     utils::checkError(error);
00119
00120     if (detection_count == 0) {
00121       std::ostringstream oss;
00122       oss « "Error: No face detected in the image ";
00123       throw std::runtime_error(oss.str());
00124     }
00125     detected_faces.resize(detection_count);
00126
00127     std::cout « "Detected " « detection_count « " faces in the image."
00128               « std::endl;
00129   }
00130   return detected_faces;
00131 }
00132
00134 std::ostream &operator«(std::ostream &os, const SFEEntity &entity) {
00135   for (size_t i = 0; i < 16; ++i) {
00136     os « std::hex « std::setw(2) « std::setfill('0')
00137        « static_cast<int>(entity.uuid[i]);
00138     if (i == 3 || i == 5 || i == 7 || i == 9) {
00139       os « '-';
00140     }
00141   }
00142   os « std::dec;
00143   return os;
00144 }
00145
00147 std::ostream &operator«(std::ostream &os, const SFETrackedState &state) {
00148   switch (state) {
00149   case SFE_TRACKED_STATE_NEW:
00150     os « "NEW";
00151     break;
00152   case SFE_TRACKED_STATE_TRACKED:
00153     os « "TRACKED";
00154     break;
00155   case SFE_TRACKED_STATE_LOST:
00156     os « "LOST";
00157     break;
00158   case SFE_TRACKED_STATE_REMOVED:
00159     os « "REMOVED";
00160     break;
00161   default:
00162     os « "UNKNOWN";
00163     break;
00164   }
00165   return os;
00166 }
00167
00169 std::ostream &operator«(std::ostream &os, const SFETracked &tracked) {
00170   os « "Face ID: " « tracked.id « " UUID: " « tracked.uuid
00171      « " State: " « tracked.state;
00172   return os;
00173 }
00174
00176 template <typename T>
00177 std::ostream &operator«(std::ostream &os, const std::vector<T> &vec) {
00178   for (auto &item : vec) {
00179     os « item « std::endl;
00180   }
00181   return os;
00182 }
00183
00185 void frame(std::vector<SFEDetection> &detections, SFETracker tracker) {
00186   size_t reserved_size = 10;
00187   size_t tracked_faces_count = reserved_size;
00188   size_t lost_faces_count = reserved_size;
00189   size_t removed_faces_count = reserved_size;
00190
00191   std::vector<SFETracked> tracked_faces(tracked_faces_count);
00192   std::vector<SFEEntity> lost_faces(lost_faces_count);
00193   std::vector<SFEEntity> removed_faces(removed_faces_count);
00194
00195   // Update tracker with detected faces
00196   SFEError error =
00197       sfeDetectionTrackerUpdate(tracker, detections.data(), detections.size(),
```

```
00198                           tracked_faces.data(), &tracked_faces_count);
00199   utils::checkError(error);
00200   tracked_faces.resize(tracked_faces_count);
00201
00202   // Get lost faces
00203   error = sfeDetectionTrackerLost(tracker, lost_faces.data(), &lost_faces_count);
00204   utils::checkError(error);
00205   lost_faces.resize(lost_faces_count);
00206
00207   // Get removed faces
00208   error = sfeDetectionTrackerRemoved(tracker, removed_faces.data(),
00209                                      &removed_faces_count);
00210   utils::checkError(error);
00211   removed_faces.resize(removed_faces_count);
00212
00213   // Print out frame results
00214   std::cout << "Tracked" << std::endl;
00215   std::cout << tracked_faces;
00216   std::cout << "Lost" << std::endl;
00217   std::cout << lost_faces;
00218   std::cout << "Removed" << std::endl;
00219   std::cout << removed_faces;
00220 }
00221
00223 int main(int argc, char *argv[]) {
00224
00225   { // STAGE 0: Parse command line arguments
00226     // Map to store argument values
00227     std::unordered_map<std::string, std::string> args;
00228
00229     // Parse command line arguments
00230     for (int i = 1; i < argc; ++i) {
00231       std::string arg = argv[i];
00232       if (arg[0] == '-') {
00233         if (arg == "-h") {
00234           printHelp();
00235           return 0;
00236         }
00237         // Check if there's a next argument and it isn't another option
00238         if (i + 1 < argc && argv[i + 1][0] != '-') {
00239           args[arg] = argv[++i];
00240         } else {
00241           std::cerr << "Option " << arg << " requires a value." << std::endl;
00242           return 1;
00243         }
00244       } else {
00245         std::cerr << "Unknown option: " << arg << std::endl;
00246         return 1;
00247       }
00248     }
00249
00250     // Assign values to variables based on parsed arguments
00251     if (args.count("-p"))
00252       image_probe = args["-p"];
00253     if (args.count("-d"))
00254       solver_face_detect = args["-d"];
00255     if (args.count("-t"))
00256       detection_threshold = std::stof(args["-t"]);
00257     if (args.count("-m"))
00258       face_size_min = std::stoi(args["-m"]);
00259     if (args.count("-x"))
00260       face_size_max = std::stoi(args["-x"]);
00261
00262     utils::printFormatted("EXAMPLE PARAMETERS");
00263     // Print resource names
00264     std::cout << "Probe image: " << image_probe << std::endl;
00265
00266     // Print solver names
00267     std::cout << "Face detection solver: " << solver_face_detect << std::endl;
00268
00269     // Print other options
00270     std::cout << "Min face size [px]: " << face_size_min << std::endl;
00271     std::cout << "Max face size [px]: " << face_size_max << std::endl;
00272     std::cout << "Detection threshold: " << detection_threshold << std::endl;
00273   }
00274
00275   utils::printToolkitInfo();
00276
00277   SFEError error{};
00278
00279   SFESolver detector_solver{};
00280   DEFER(sfeSolverFree(detector_solver));
00281   { // STAGE 1: loading solvers
00282     utils::printFormatted("LOADING SOLVERS");
00283     // Initialize face detection solver
00284     error = sfeSolverCreate(
00285         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
```

```
00286         SOLVER_PARAMETERS.size(), &detector_solver);
00287     utils::checkError(error);
00288   }
00289
00290   std::vector<SFEDetection> detected_faces;
00291   { // STAGE 2: Detect faces
00292     utils::printFormatted("DETECT FACES");
00293
00294     SFEImage image{};
00295     DEFER(sfeImageFree(image));
00296     { // STAGE 2.1: Load image
00297       // Load image data from file
00298       auto image_data = utils::readFile(image_probe);
00299
00300       // Decode image from data
00301       error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00302       utils::checkError(error);
00303     }
00304
00305     SFEDetection detected_face = {};
00306     { // STAGE 2.2: Detect face in the image
00307       // Detect a face in the image
00308       detected_faces = detectFaces(image, detector_solver);
00309     }
00310   }
00311
00312   SFETracker tracker{};
00313   DEFER(sfeDetectionTrackerFree(tracker));
00314   { // STAGE 3: Track faces
00315     utils::printFormatted("TRACK FACES");
00316     error = sfeDetectionTrackerCreate(0.3f, 0.1f, 0.3f, 0.1f, 1, &tracker);
00317     utils::checkError(error);
00318
00319     // We will be using the current detected faces and simulate detection across
00320     // 4 frames In a real application, you would call detectFaces() for each
00321     // frame and pass the detected faces to the tracker
00322
00323     // First frame with detected faces, all should track as NEW
00324     utils::printFormatted("FRAME 1");
00325     frame(detected_faces, tracker);
00326
00327     // Second frame with the same UUIDs, all should track as TRACKED
00328     utils::printFormatted("FRAME 2");
00329     frame(detected_faces, tracker);
00330
00331     // Simulate a lost detection
00332     detected_faces.clear();
00333
00334     // We should see the lost UUIDs
00335     utils::printFormatted("FRAME 3");
00336     frame(detected_faces, tracker);
00337
00338     // We should see the removed UUIDs
00339     utils::printFormatted("FRAME 4");
00340     frame(detected_faces, tracker);
00341   }
00342   utils::printFormatted("FINISHED");
00343 }
```

9.15 D:/glab/1512fd1724a/1/example/example_iris_identify.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_iris.h¨
#include <regex>
#include <vector>
#include <iomanip>
#include <sstream>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- SFEIrisTemplate extract_template (std::string image_path, SFESolver &detector_solver, SFESolver &template_solver)
- void printHelp ()
- int main (int argc, char *argv[])

  Main fuction is used to parse command line arguments.

Variables

- std::string image_match = "assets/iris/iris1_1.png"
- std::string image_probe = "assets/iris/iris1_0.png"
- std::string image_non_match = "assets/iris/iris0.png"
- std::string solver_iris_detect = SOLVER_IRIS_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::string solver_iris_template = SOLVER_IRIS_TEMPLATE
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- float identification_threshold = 0.6f

9.15.1 Function Documentation

9.15.1.1 extract_template()

```
SFEIrisTemplate extract_template (
            std::string image_path,
            SFESolver & detector_solver,
            SFESolver & template_solver )
```

[Iris Detect]
[Iris Detect]
[Iris Template Extract]
[Iris Template Extract]

Examples

example_iris_identify.cpp, and example_palm_identify.cpp.

Definition at line 30 of file example_iris_identify.cpp.

```
00032                                                                {
00033
00034   SFEImage image{};
00035   DEFER(sfeImageFree(image));
00036   SFEError error{};
00037   { // STAGE 1: Load image
00038     // Load image data from file
00039     auto image_data = utils::readFile(image_path);
00040
00041     // Decode image from data
00042     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00043     utils::checkError(error);
00044   }
00045
00047   SFEDetection detected_iris = {};
00048   size_t iris_count = 1;
00049   { // STAGE 2: Detect iris in the image
00050
00051     // Detect iris in the image
00052     error = sfeDetect(detector_solver, image, 0.1f, &detected_iris, &iris_count);
00053     utils::checkError(error);
00054
00055     if (iris_count == 0 || detected_iris.confidence < 0.5) {
00056       throw std::runtime_error("No iris detected in the probe image.");
00057     }
00058
00059     std::cout « "Found iris in the image. Extracting template..." « std::endl;
00060   }
00062
00064   SFEIrisTemplate iris_template;
00065   { // STAGE 3 Extract probe iris template
00066     // Use template extraction solver to create new iris
00067     // template
00068     error = sfeIrisTemplateExtract(template_solver, image, &detected_iris,
```

```
00069                                   &iris_template);
00070     utils::checkError(error);
00071   }
00073   return iris_template;
00074 }
```

9.15.1.2 main()

```
int main (
              int argc,
              char * argv[] )
```

Main fuction is used to parse command line arguments.
[Iris Template Version]
[Iris Template Version]
[Iris Template Quality]
[Iris Template Quality]
[Iris Template Identify]
[Iris Template Identify]
[Iris Template Match]
[Iris Template Match]
Definition at line 89 of file example_iris_identify.cpp.

```
00089                                   {
00090
00091   { // STAGE 0: Parse command line arguments
00092     // Map to store argument values
00093     std::unordered_map<std::string, std::string> args;
00094
00095     // Parse command line arguments
00096     for (int i = 1; i < argc; ++i) {
00097       std::string arg = argv[i];
00098       if (arg[0] == '-') {
00099         if (arg == "-h") {
00100           printHelp();
00101           return 0;
00102         }
00103         // Check if there's a next argument and it isn't another option
00104         if (i + 1 < argc && argv[i + 1][0] != '-') {
00105           args[arg] = argv[++i];
00106         } else {
00107           std::cerr « "Option " « arg « " requires a value." « std::endl;
00108           return 1;
00109         }
00110       } else {
00111         std::cerr « "Unknown option: " « arg « std::endl;
00112         return 1;
00113       }
00114     }
00115
00116     // Assign values to variables based on parsed arguments
00117     if (args.count("-p"))
00118       image_probe = args["-p"];
00119     if (args.count("-g"))
00120       image_match = args["-g"];
00121     if (args.count("-d"))
00122       solver_iris_detect = args["-d"];
00123     if (args.count("-e"))
00124       solver_iris_template = args["-e"];
00125     if (args.count("-i"))
00126       identification_threshold = std::stof(args["-i"]);
00127
00128     utils::printFormatted("EXAMPLE PARAMETERS");
00129     // Print resource names
00130     std::cout « "Probe image: " « image_probe « std::endl;
00131     std::cout « "Matching image: " « image_match « std::endl;
00132
00133     // Print solver names
00134     std::cout « "Iris detection solver: " « solver_iris_detect « std::endl;
00135     std::cout « "Iris template solver: " « solver_iris_template « std::endl;
00136
00137     // Print other options
00138     std::cout « "Identification threshold: " « identification_threshold
00139               « std::endl;
00140   }
00141
00142   utils::printToolkitInfo();
00143
00144   SFEError error{};
00145
00146   SFESolver detector_solver{};
```

```
00147   DEFER(sfeSolverFree(detector_solver));
00148   SFESolver landmarks_solver{};
00149   DEFER(sfeSolverFree(landmarks_solver));
00150   SFESolver template_solver{};
00151   DEFER(sfeSolverFree(template_solver));
00152   { // STAGE 1.1: loading solvers
00153     utils::printFormatted("LOADING SOLVERS");
00154     // Initialize Iris detection solver
00155     error = sfeSolverCreate(
00156         solver_iris_detect.c_str(), SOLVER_PARAMETERS.data(),
00157         SOLVER_PARAMETERS.size(), &detector_solver);
00158     utils::checkError(error);
00159
00160     // Initialize template extraction solver
00161     error = sfeSolverCreate(
00162         solver_iris_template.c_str(), SOLVER_PARAMETERS.data(),
00163         SOLVER_PARAMETERS.size(), &template_solver);
00164     utils::checkError(error);
00165   }
00166
00167   SFEIrisTemplate probe_iris_template;
00168   { // STAGE 1: Extract probe iris template
00169     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00170
00171     probe_iris_template =
00172         extract_template(image_probe, detector_solver, template_solver);
00173   }
00174
00175   { // STAGE 1.1: (optional) Print template attributes
00176     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00177
00179     SFEIrisTemplateVersion version = {};
00180     error = sfeIrisTemplateVersion(&probe_iris_template, &version);
00181     utils::checkError(error);
00182
00183     std::cout « "Version of the probe template: " « version.version_major
00184               « '.' « version.version_minor « std::endl;
00186
00188     float quality = {};
00189     error = sfeIrisTemplateQuality(&probe_iris_template, &quality);
00190     utils::checkError(error);
00191
00192     std::cout « "Quality of the probe template: " « quality « std::endl;
00194   }
00195
00196   size_t gallery_size = 10;
00197   std::vector<SFEIrisTemplate> iris_template_gallery(gallery_size);
00198   { // STAGE 2: Create iris templates gallery
00199     utils::printFormatted("IRIS GALLERY EXTRACTION");
00200
00201     auto matching_iris_template =
00202         extract_template(image_match, detector_solver, template_solver);
00203
00204     auto non_matching_iris_template =
00205         extract_template(image_non_match, detector_solver, template_solver);
00206
00207     // Fill the gallery with non-matching templates for demonstration purposes
00208     for (size_t i = 0; i < gallery_size; i++) {
00209       iris_template_gallery[i] = non_matching_iris_template;
00210     }
00211
00212     // Add matching template to the gallery at index 5
00213     iris_template_gallery[5] = matching_iris_template;
00214   }
00215
00217   size_t candidate_count = 1;
00218   std::vector<SFETemplateIdentificationCandidate> identification_results(
00219       candidate_count);
00220   { // STAGE 3: Identify the probe template
00221     utils::printFormatted("1:N IDENTIFICATION");
00222
00223     error = sfeIrisTemplateIdentify(
00224         &probe_iris_template, iris_template_gallery.data(),
00225         iris_template_gallery.size(), identification_threshold,
00226         identification_results.data(), &candidate_count, 4);
00227     utils::checkError(error);
00228
00229     if (candidate_count == 0) {
00230       std::cout « "No candidates found above identification threshold "
00231                 « identification_threshold « std::endl;
00232       return 0;
00233     }
00234
00235     std::cout « "Found " « identification_results.size()
00236               « " candidates above identification threshold "
00237               « identification_threshold « std::endl;
00238     for (auto &result : identification_results)
```

```
00239       std::cout « "Template index: #" « result.index
00240                 « ", score: " « result.score « std::endl;
00241   }
00243
00245   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00246     utils::printFormatted("1:1 MATCHING");
00247
00248     if (identification_results.size() == 0) {
00249       std::cout « "No candidates found above identification threshold "
00250                 « identification_threshold « std::endl;
00251       return 0;
00252     }
00253
00254     // Since it's sorted vector by score, the best candidate is the first one
00255     auto match_iris_template =
00256         iris_template_gallery[identification_results[0].index];
00257
00258     float match_confidence;
00259     error = sfeIrisTemplateMatch(&probe_iris_template, &match_iris_template,
00260                                  &match_confidence);
00261     utils::checkError(error);
00262
00263     std::cout
00264         « "Matching score of probe template with the best template, score: "
00265         « match_confidence « std::endl;
00266   }
00268
00269   utils::printFormatted("FINISHED");
00270 }
```

9.15.1.3 printHelp()

```
void printHelp ( )
```

Definition at line 76 of file example_iris_identify.cpp.

```
00076                {
00077   std::cout « "Help: Usage of the program." « std::endl;
00078   std::cout « "Options:" « std::endl;
00079   std::cout « "-h: Display help." « std::endl;
00080   std::cout « "-p: Probe image file." « std::endl;
00081   std::cout « "-g: Matching image file. Our \"iris database\"." « std::endl;
00082   std::cout « "-d: Path to detector solver." « std::endl;
00083   std::cout « "-e: Path to extraction solver." « std::endl;
00084   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00085   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00086 }
```

9.15.2 Variable Documentation

9.15.2.1 identification_threshold

```
float identification_threshold = 0.6f
```

Definition at line 28 of file example_iris_identify.cpp.

9.15.2.2 image_match

```
std::string image_match = ¨assets/iris/iris1_1.png¨
```

Examples

example_iris_identify.cpp, and example_palm_identify.cpp.

Definition at line 18 of file example_iris_identify.cpp.

9.15.2.3 image_non_match

```
std::string image_non_match = ¨assets/iris/iris0.png¨
```

Examples

example_iris_identify.cpp, and example_palm_identify.cpp.

Definition at line 20 of file example_iris_identify.cpp.

9.15.2.4 image_probe

```
std::string image_probe = ¨assets/iris/iris1_0.png¨
```
Definition at line 19 of file example_iris_identify.cpp.

9.15.2.5 solver_iris_detect

```
std::string solver_iris_detect = SOLVER_IRIS_DETECT
```
Solvers to use in example, the defaults are filled in by CMake.

Examples

example_iris_identify.cpp.

Definition at line 23 of file example_iris_identify.cpp.

9.15.2.6 solver_iris_template

```
std::string solver_iris_template = SOLVER_IRIS_TEMPLATE
```

Examples

example_iris_identify.cpp.

Definition at line 24 of file example_iris_identify.cpp.

9.15.2.7 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```
Definition at line 26 of file example_iris_identify.cpp.

9.16 example_iris_identify.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_iris.h"
00007 #include <regex>
00008 #include <vector>
00009
00010 #include <iomanip>
00011 #include <sstream>
00012
00013 #include "annotate.hpp"
00014 #include "solvers.h"
00015 #include "utils.hpp"
00016 #include <unordered_map>
00017
00018 std::string image_match = "assets/iris/iris1_1.png";
00019 std::string image_probe = "assets/iris/iris1_0.png";
00020 std::string image_non_match = "assets/iris/iris0.png";
00021
00023 std::string solver_iris_detect = SOLVER_IRIS_DETECT;
00024 std::string solver_iris_template = SOLVER_IRIS_TEMPLATE;
00025
00026 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00027
00028 float identification_threshold = 0.6f;
00029
00030 SFEIrisTemplate extract_template(std::string image_path,
00031                                  SFESolver &detector_solver,
00032                                  SFESolver &template_solver) {
00033
00034   SFEImage image{};
00035   DEFER(sfeImageFree(image));
00036   SFEError error{};
00037   { // STAGE 1: Load image
00038     // Load image data from file
00039     auto image_data = utils::readFile(image_path);
00040
00041     // Decode image from data
00042     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00043     utils::checkError(error);
00044   }
```

```
00045
00047   SFEDetection detected_iris = {};
00048   size_t iris_count = 1;
00049   { // STAGE 2: Detect iris in the image
00050
00051     // Detect iris in the image
00052     error = sfeDetect(detector_solver, image, 0.1f, &detected_iris, &iris_count);
00053     utils::checkError(error);
00054
00055     if (iris_count == 0 || detected_iris.confidence < 0.5) {
00056       throw std::runtime_error("No iris detected in the probe image.");
00057     }
00058
00059     std::cout « "Found iris in the image. Extracting template..." « std::endl;
00060   }
00062
00064   SFEIrisTemplate iris_template;
00065   { // STAGE 3 Extract probe iris template
00066     // Use template extraction solver to create new iris
00067     // template
00068     error = sfeIrisTemplateExtract(template_solver, image, &detected_iris,
00069                                    &iris_template);
00070     utils::checkError(error);
00071   }
00073   return iris_template;
00074 }
00075
00076 void printHelp() {
00077   std::cout « "Help: Usage of the program." « std::endl;
00078   std::cout « "Options:" « std::endl;
00079   std::cout « "-h: Display help." « std::endl;
00080   std::cout « "-p: Probe image file." « std::endl;
00081   std::cout « "-g: Matching image file. Our \"iris database\"." « std::endl;
00082   std::cout « "-d: Path to detector solver." « std::endl;
00083   std::cout « "-e: Path to extraction solver." « std::endl;
00084   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00085   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00086 }
00087
00089 int main(int argc, char *argv[]) {
00090
00091   { // STAGE 0: Parse command line arguments
00092     // Map to store argument values
00093     std::unordered_map<std::string, std::string> args;
00094
00095     // Parse command line arguments
00096     for (int i = 1; i < argc; ++i) {
00097       std::string arg = argv[i];
00098       if (arg[0] == '-') {
00099         if (arg == "-h") {
00100           printHelp();
00101           return 0;
00102         }
00103         // Check if there's a next argument and it isn't another option
00104         if (i + 1 < argc && argv[i + 1][0] != '-') {
00105           args[arg] = argv[++i];
00106         } else {
00107           std::cerr « "Option " « arg « " requires a value." « std::endl;
00108           return 1;
00109         }
00110       } else {
00111         std::cerr « "Unknown option: " « arg « std::endl;
00112         return 1;
00113       }
00114     }
00115
00116     // Assign values to variables based on parsed arguments
00117     if (args.count("-p"))
00118       image_probe = args["-p"];
00119     if (args.count("-g"))
00120       image_match = args["-g"];
00121     if (args.count("-d"))
00122       solver_iris_detect = args["-d"];
00123     if (args.count("-e"))
00124       solver_iris_template = args["-e"];
00125     if (args.count("-i"))
00126       identification_threshold = std::stof(args["-i"]);
00127
00128     utils::printFormatted("EXAMPLE PARAMETERS");
00129     // Print resource names
00130     std::cout « "Probe image: " « image_probe « std::endl;
00131     std::cout « "Matching image: " « image_match « std::endl;
00132
00133     // Print solver names
00134     std::cout « "Iris detection solver: " « solver_iris_detect « std::endl;
00135     std::cout « "Iris template solver: " « solver_iris_template « std::endl;
00136
```

```
00137     // Print other options
00138     std::cout « "Identification threshold: " « identification_threshold
00139               « std::endl;
00140   }
00141
00142   utils::printToolkitInfo();
00143
00144   SFEError error{};
00145
00146   SFESolver detector_solver{};
00147   DEFER(sfeSolverFree(detector_solver));
00148   SFESolver landmarks_solver{};
00149   DEFER(sfeSolverFree(landmarks_solver));
00150   SFESolver template_solver{};
00151   DEFER(sfeSolverFree(template_solver));
00152   { // STAGE 1.1: loading solvers
00153     utils::printFormatted("LOADING SOLVERS");
00154     // Initialize Iris detection solver
00155     error = sfeSolverCreate(
00156         solver_iris_detect.c_str(), SOLVER_PARAMETERS.data(),
00157         SOLVER_PARAMETERS.size(), &detector_solver);
00158     utils::checkError(error);
00159
00160     // Initialize template extraction solver
00161     error = sfeSolverCreate(
00162         solver_iris_template.c_str(), SOLVER_PARAMETERS.data(),
00163         SOLVER_PARAMETERS.size(), &template_solver);
00164     utils::checkError(error);
00165   }
00166
00167   SFEIrisTemplate probe_iris_template;
00168   { // STAGE 1: Extract probe iris template
00169     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00170
00171     probe_iris_template =
00172         extract_template(image_probe, detector_solver, template_solver);
00173   }
00174
00175   { // STAGE 1.1: (optional) Print template attributes
00176     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00177
00179     SFEIrisTemplateVersion version = {};
00180     error = sfeIrisTemplateVersion(&probe_iris_template, &version);
00181     utils::checkError(error);
00182
00183     std::cout « "Version of the probe template: " « version.version_major
00184               « '.' « version.version_minor « std::endl;
00186
00188     float quality = {};
00189     error = sfeIrisTemplateQuality(&probe_iris_template, &quality);
00190     utils::checkError(error);
00191
00192     std::cout « "Quality of the probe template: " « quality « std::endl;
00194   }
00195
00196   size_t gallery_size = 10;
00197   std::vector<SFEIrisTemplate> iris_template_gallery(gallery_size);
00198   { // STAGE 2: Create iris templates gallery
00199     utils::printFormatted("IRIS GALLERY EXTRACTION");
00200
00201     auto matching_iris_template =
00202         extract_template(image_match, detector_solver, template_solver);
00203
00204     auto non_matching_iris_template =
00205         extract_template(image_non_match, detector_solver, template_solver);
00206
00207     // Fill the gallery with non-matching templates for demonstration purposes
00208     for (size_t i = 0; i < gallery_size; i++) {
00209       iris_template_gallery[i] = non_matching_iris_template;
00210     }
00211
00212     // Add matching template to the gallery at index 5
00213     iris_template_gallery[5] = matching_iris_template;
00214   }
00215
00217   size_t candidate_count = 1;
00218   std::vector<SFETemplateIdentificationCandidate> identification_results(
00219       candidate_count);
00220   { // STAGE 3: Identify the probe template
00221     utils::printFormatted("1:N IDENTIFICATION");
00222
00223     error = sfeIrisTemplateIdentify(
00224         &probe_iris_template, iris_template_gallery.data(),
00225         iris_template_gallery.size(), identification_threshold,
00226         identification_results.data(), &candidate_count, 4);
00227     utils::checkError(error);
00228
```

```
00229     if (candidate_count == 0) {
00230       std::cout « "No candidates found above identification threshold "
00231                « identification_threshold « std::endl;
00232       return 0;
00233     }
00234
00235     std::cout « "Found " « identification_results.size()
00236               « " candidates above identification threshold "
00237               « identification_threshold « std::endl;
00238     for (auto &result : identification_results)
00239       std::cout « "Template index: #" « result.index
00240                 « ", score: " « result.score « std::endl;
00241   }
00243
00245   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00246     utils::printFormatted("1:1 MATCHING");
00247
00248     if (identification_results.size() == 0) {
00249       std::cout « "No candidates found above identification threshold "
00250                 « identification_threshold « std::endl;
00251       return 0;
00252     }
00253
00254     // Since it's sorted vector by score, the best candidate is the first one
00255     auto match_iris_template =
00256         iris_template_gallery[identification_results[0].index];
00257
00258     float match_confidence;
00259     error = sfeIrisTemplateMatch(&probe_iris_template, &match_iris_template,
00260                                  &match_confidence);
00261     utils::checkError(error);
00262
00263     std::cout
00264         « "Matching score of probe template with the best template, score: "
00265         « match_confidence « std::endl;
00266   }
00268
00269   utils::printFormatted("FINISHED");
00270 }
```

9.17 D:/glab/1512fd1724a/1/example/example_palm_attributes.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_palm.h¨
#include <iomanip>
#include <optional>
#include <regex>
#include <sstream>
#include <vector>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- void printPalmDetectionInfo (SFEDetection &detected_palm, SFEPalmLandmarks &attributes)
- void printHelp ()
- int main (int argc, char ∗argv[ ])

  Main fuction is used to parse command line arguments.

Variables

- std::string image_probe = ¨assets/palm/palm_id1_r1.jpg¨
- std::string solver_palm_detect = SOLVER_PALM_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::optional< std::string > solver_palm_attributes = std::nullopt

- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- float detection_threshold = 0.1f

9.17.1 Function Documentation

9.17.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main fuction is used to parse command line arguments.
[Image Load]
[Image Load]
[Palm Detect]
[Palm Detect]
[Palm Landmarks]
[Palm Landmarks]
[Palm Attributes]
[Palm Attributes]
Definition at line 67 of file example_palm_attributes.cpp.

```
00067                                {
00068
00069   { // STAGE: 0 Parse command line arguments
00070     // Map to store argument values
00071     std::unordered_map<std::string, std::string> args;
00072
00073     // Parse command line arguments
00074     for (int i = 1; i < argc; ++i) {
00075       std::string arg = argv[i];
00076       if (arg[0] == '-') {
00077         if (arg == "-h") {
00078           printHelp();
00079           return 0;
00080         }
00081         // Check if there's a next argument and it isn't another option
00082         if (i + 1 < argc && argv[i + 1][0] != '-') {
00083           args[arg] = argv[++i];
00084         } else {
00085           std::cerr << "Option " << arg << " requires a value." << std::endl;
00086           return 1;
00087         }
00088       } else {
00089         std::cerr << "Unknown option: " << arg << std::endl;
00090         return 1;
00091       }
00092     }
00093
00094     // Assign values to variables based on parsed arguments
00095     if (args.count("-p"))
00096       image_probe = args["-p"];
00097     if (args.count("-d"))
00098       solver_palm_detect = args["-d"];
00099     if (args.count("-i"))
00100       solver_palm_attributes = args["-i"];
00101     if (args.count("-t"))
00102       detection_threshold = std::stof(args["-t"]);
00103
00104     utils::printFormatted("PARAMETERS");
00105     // Print resource names
00106     std::cout << "Probe image: " << image_probe << std::endl;
00107
00108     // Print solver names
00109     std::cout << "Palm detection solver: " << solver_palm_detect << std::endl;
00110     std::cout << "Palm attributes solver: "
00111               << (solver_palm_attributes.has_value() ? solver_palm_attributes.value() : "not
      specified")
00112               << std::endl;
00113
00114     // Print other options
00115     std::cout << "Detection threshold: " << detection_threshold << std::endl;
00116   }
00117
00118   utils::printToolkitInfo();
00119
00120   SFEError error{};
00121
00122   SFESolver detector_solver{};
00123   DEFER(sfeSolverFree(detector_solver));
```

```
00124   { // STAGE 1: loading solvers
00125     utils::printFormatted("LOADING solvers");
00126     // Initialize palm detection solver
00127     error = sfeSolverCreate(
00128         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00129         SOLVER_PARAMETERS.size(), &detector_solver);
00130     utils::checkError(error);
00131   }
00132
00133   SFEImage image{};
00134   DEFER(sfeImageFree(image));
00135   { // STAGE 2: Load image
00136
00137     utils::printFormatted("PALM DETECTION");
00139     // Load image data from file
00140     auto image_data = utils::readFile(image_probe);
00141
00142     // Decode image from data
00143     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00144     utils::checkError(error);
00146   }
00148   SFEDetection detected_palm = {};
00149   { // STAGE 3: Detect palm in the image
00150     size_t detection_count = 1;
00151     // Detect palm in the image
00152     error = sfeDetect(detector_solver, image, detection_threshold,
00153                       &detected_palm, &detection_count);
00154     utils::checkError(error);
00155
00156     if (detection_count == 0) {
00157       std::cout « "No palm detected in the probe image." « std::endl;
00158       return 0;
00159     } else {
00160       std::cout « "Found " « detection_count
00161                 « " palm(s) in the probe image. Using the palm with highest "
00162                    "confidence."
00163                 « std::endl;
00164     }
00165   }
00167
00169   SFEPalmLandmarks landmarks = {};
00170   { // STAGE 4: Extract palm landmarks
00171     utils::printFormatted("LANDMARKS");
00172     // Extract palm landmarks (hand position, orientation, keypoints)
00173     error = sfePalmLandmarks(detector_solver, image, &detected_palm, &landmarks);
00174     utils::checkError(error);
00175   }
00177
00178   if (solver_palm_attributes.has_value()) { // STAGE 5: Attributes (if solver available)
00179       SFESolver attributes_solver{};
00180       DEFER(sfeSolverFree(attributes_solver));
00181       // Initialize palm attributes solver
00182       error = sfeSolverCreate(
00183           solver_palm_attributes.value().c_str(), SOLVER_PARAMETERS.data(),
00184           SOLVER_PARAMETERS.size(), &attributes_solver);
00185       utils::checkError(error);
00186
00188       utils::printFormatted("ATTRIBUTES");
00189       SFEPalmAttributes attributes = {};
00190       error = sfePalmAttributes(attributes_solver, image, &landmarks,
00191                                 &attributes);
00192       utils::checkError(error);
00194
00195       printPalmDetectionInfo(detected_palm, landmarks);
00196       std::cout « "Hand orientation " « attributes.hand_orientation « std::endl;
00197       std::cout « "Hand position " « attributes.hand_position « std::endl;
00198       std::cout « "Quality " « attributes.quality « std::endl;
00199   } else {
00200     printPalmDetectionInfo(detected_palm, landmarks);
00201     std::cout « "Palm attributes solver not specified, skipping attributes analysis." « std::endl;
00202   }
00203
00204   utils::printFormatted("FINISHED");
00205 }
```

9.17.1.2 printHelp()

```
void printHelp ( )
```

Definition at line 56 of file example_palm_attributes.cpp.

```
00056                 {
00057   std::cout « "Help: Usage of the program." « std::endl;
00058   std::cout « "Options:" « std::endl;
00059   std::cout « "-h: Display help." « std::endl;
00060   std::cout « "-p: Probe image file." « std::endl;
00061   std::cout « "-d: Path to detector solver." « std::endl;
```

```
00062   std::cout « "-i: Path to attributes solver." « std::endl;
00063   std::cout « "-t: Palm detection threshold. <0,1>" « std::endl;
00064 }
```

9.17.1.3 printPalmDetectionInfo()

```
void printPalmDetectionInfo (
            SFEDetection & detected_palm,
            SFEPalmLandmarks & attributes )  [inline]
```

Definition at line 33 of file example_palm_attributes.cpp.

```
00033                                                                                                                {
00034   std::cout « "SFEDetection{ " « std::endl;
00035   std::cout « "  confidence: " « detected_palm.confidence « std::endl;
00036   std::cout « "  hand_position: " « attributes.hand_position « std::endl;
00037   std::cout « "  hand_orientation: " « attributes.hand_orientation « std::endl;
00038   std::cout « "  SFEBoundingBox{ " « std::endl;
00039   std::cout « "    x: " « detected_palm.bounding_box.x « std::endl;
00040   std::cout « "    y: " « detected_palm.bounding_box.y « std::endl;
00041   std::cout « "    width: " « detected_palm.bounding_box.width « std::endl;
00042   std::cout « "    height: " « detected_palm.bounding_box.height « std::endl;
00043   std::cout « "    }" « std::endl;
00044   std::cout « "  Keypoints{ " « std::endl;
00045   for (auto &keypoint : attributes.keypoints) {
00046     std::cout « "      SFEPalmKeypoint{ " « std::endl;
00047     std::cout « "        x: " « keypoint.x « std::endl;
00048     std::cout « "        y: " « keypoint.y « std::endl;
00049     std::cout « "      }" « std::endl;
00050   }
00051   std::cout « "    }" « std::endl;
00052   std::cout « "  }" « std::endl;
00053   std::cout « std::endl;
00054 }
```

9.17.2 Variable Documentation

9.17.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 31 of file example_palm_attributes.cpp.

9.17.2.2 image_probe

```
std::string image_probe = ¨assets/palm/palm_id1_r1.jpg¨
```

Definition at line 18 of file example_palm_attributes.cpp.

9.17.2.3 solver_palm_attributes

```
std::optional<std::string> solver_palm_attributes = std::nullopt
```

Examples

example_palm_attributes.cpp.

Definition at line 26 of file example_palm_attributes.cpp.

9.17.2.4 solver_palm_detect

```
std::string solver_palm_detect = SOLVER_PALM_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.
Definition at line 21 of file example_palm_attributes.cpp.

9.17.2.5 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Definition at line 29 of file example_palm_attributes.cpp.

9.18 example_palm_attributes.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_palm.h"
00007 #include <iomanip>
00008 #include <optional>
00009 #include <regex>
00010 #include <sstream>
00011 #include <vector>
00012
00013 #include "annotate.hpp"
00014 #include "solvers.h"
00015 #include "utils.hpp"
00016 #include <unordered_map>
00017
00018 std::string image_probe = "assets/palm/palm_id1_r1.jpg";
00019
00021 std::string solver_palm_detect = SOLVER_PALM_DETECT;
00022
00023 #ifdef SOLVER_PALM_ATTRIBUTES
00024 std::optional<std::string> solver_palm_attributes = SOLVER_PALM_ATTRIBUTES;
00025 #else
00026 std::optional<std::string> solver_palm_attributes = std::nullopt;
00027 #endif // SOLVER_PALM_ATTRIBUTES
00028
00029 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00030
00031 float detection_threshold = 0.1f;
00032
00033 inline void printPalmDetectionInfo(SFEDetection &detected_palm, SFEPalmLandmarks &attributes) {
00034   std::cout « "SFEDetection{ " « std::endl;
00035   std::cout « "  confidence: " « detected_palm.confidence « std::endl;
00036   std::cout « "  hand_position: " « attributes.hand_position « std::endl;
00037   std::cout « "  hand_orientation: " « attributes.hand_orientation « std::endl;
00038   std::cout « "  SFEBoundingBox{ " « std::endl;
00039   std::cout « "    x: " « detected_palm.bounding_box.x « std::endl;
00040   std::cout « "    y: " « detected_palm.bounding_box.y « std::endl;
00041   std::cout « "    width: " « detected_palm.bounding_box.width « std::endl;
00042   std::cout « "    height: " « detected_palm.bounding_box.height « std::endl;
00043   std::cout « "    }" « std::endl;
00044   std::cout « "  Keypoints{ " « std::endl;
00045   for (auto &keypoint : attributes.keypoints) {
00046     std::cout « "      SFEPalmKeypoint{ " « std::endl;
00047     std::cout « "        x: " « keypoint.x « std::endl;
00048     std::cout « "        y: " « keypoint.y « std::endl;
00049     std::cout « "      }" « std::endl;
00050   }
00051   std::cout « "    }" « std::endl;
00052   std::cout « "  }" « std::endl;
00053   std::cout « std::endl;
00054 }
00055
00056 void printHelp() {
00057   std::cout « "Help: Usage of the program." « std::endl;
00058   std::cout « "Options:" « std::endl;
00059   std::cout « "-h: Display help." « std::endl;
00060   std::cout « "-p: Probe image file." « std::endl;
00061   std::cout « "-d: Path to detector solver." « std::endl;
00062   std::cout « "-i: Path to attributes solver." « std::endl;
00063   std::cout « "-t: Palm detection threshold. <0,1>" « std::endl;
00064 }
00065
00067 int main(int argc, char *argv[]) {
00068
00069   { // STAGE: 0 Parse command line arguments
00070     // Map to store argument values
00071     std::unordered_map<std::string, std::string> args;
00072
00073     // Parse command line arguments
00074     for (int i = 1; i < argc; ++i) {
00075       std::string arg = argv[i];
00076       if (arg[0] == '-') {
00077         if (arg == "-h") {
00078           printHelp();
00079           return 0;
00080         }
00081         // Check if there's a next argument and it isn't another option
00082         if (i + 1 < argc && argv[i + 1][0] != '-') {
00083           args[arg] = argv[++i];
00084         } else {
00085           std::cerr « "Option " « arg « " requires a value." « std::endl;
00086           return 1;
00087         }
00088       } else {
```

```
00089         std::cerr « "Unknown option: " « arg « std::endl;
00090         return 1;
00091       }
00092     }
00093
00094     // Assign values to variables based on parsed arguments
00095     if (args.count("-p"))
00096       image_probe = args["-p"];
00097     if (args.count("-d"))
00098       solver_palm_detect = args["-d"];
00099     if (args.count("-i"))
00100       solver_palm_attributes = args["-i"];
00101     if (args.count("-t"))
00102       detection_threshold = std::stof(args["-t"]);
00103
00104     utils::printFormatted("PARAMETERS");
00105     // Print resource names
00106     std::cout « "Probe image: " « image_probe « std::endl;
00107
00108     // Print solver names
00109     std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
00110     std::cout « "Palm attributes solver: "
00111               « (solver_palm_attributes.has_value() ? solver_palm_attributes.value() : "not
    specified")
00112               « std::endl;
00113
00114     // Print other options
00115     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00116   }
00117
00118   utils::printToolkitInfo();
00119
00120   SFEError error{};
00121
00122   SFESolver detector_solver{};
00123   DEFER(sfeSolverFree(detector_solver));
00124   { // STAGE 1: loading solvers
00125     utils::printFormatted("LOADING solvers");
00126     // Initialize palm detection solver
00127     error = sfeSolverCreate(
00128         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00129         SOLVER_PARAMETERS.size(), &detector_solver);
00130     utils::checkError(error);
00131   }
00132
00133   SFEImage image{};
00134   DEFER(sfeImageFree(image));
00135   { // STAGE 2: Load image
00136
00137     utils::printFormatted("PALM DETECTION");
00139     // Load image data from file
00140     auto image_data = utils::readFile(image_probe);
00141
00142     // Decode image from data
00143     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00144     utils::checkError(error);
00146   }
00148   SFEDetection detected_palm = {};
00149   { // STAGE 3: Detect palm in the image
00150     size_t detection_count = 1;
00151     // Detect palm in the image
00152     error = sfeDetect(detector_solver, image, detection_threshold,
00153                       &detected_palm, &detection_count);
00154     utils::checkError(error);
00155
00156     if (detection_count == 0) {
00157       std::cout « "No palm detected in the probe image." « std::endl;
00158       return 0;
00159     } else {
00160       std::cout « "Found " « detection_count
00161                 « " palm(s) in the probe image. Using the palm with highest "
00162                   "confidence."
00163                 « std::endl;
00164     }
00165   }
00167
00169   SFEPalmLandmarks landmarks = {};
00170   { // STAGE 4: Extract palm landmarks
00171     utils::printFormatted("LANDMARKS");
00172     // Extract palm landmarks (hand position, orientation, keypoints)
00173     error = sfePalmLandmarks(detector_solver, image, &detected_palm, &landmarks);
00174     utils::checkError(error);
00175   }
00177
00178   if (solver_palm_attributes.has_value()) { // STAGE 5: Attributes (if solver available)
00179       SFESolver attributes_solver{};
00180       DEFER(sfeSolverFree(attributes_solver));
```

```
00181       // Initialize palm attributes solver
00182       error = sfeSolverCreate(
00183           solver_palm_attributes.value().c_str(), SOLVER_PARAMETERS.data(),
00184           SOLVER_PARAMETERS.size(), &attributes_solver);
00185       utils::checkError(error);
00186
00188       utils::printFormatted("ATTRIBUTES");
00189       SFEPalmAttributes attributes = {};
00190       error = sfePalmAttributes(attributes_solver, image, &landmarks,
00191                                 &attributes);
00192       utils::checkError(error);
00194
00195       printPalmDetectionInfo(detected_palm, landmarks);
00196       std::cout « "Hand orientation " « attributes.hand_orientation « std::endl;
00197       std::cout « "Hand position " « attributes.hand_position « std::endl;
00198       std::cout « "Quality " « attributes.quality « std::endl;
00199   } else {
00200     printPalmDetectionInfo(detected_palm, landmarks);
00201     std::cout « "Palm attributes solver not specified, skipping attributes analysis." « std::endl;
00202   }
00203
00204   utils::printFormatted("FINISHED");
00205 }
```

9.19 D:/glab/1512fd1724a/1/example/example_palm_identify.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_palm.h¨
#include <regex>
#include <vector>
#include <iomanip>
#include <sstream>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- SFEPalmTemplate extract_template (std::string image_path, SFESolver &detector_solver, SFESolver &landmarks_solver, SFESolver &extraction_solver)
- void printHelp ()
- int main (int argc, char ∗argv[ ])

  Main fuction is used to parse command line arguments.

Variables

- std::string image_match = ¨assets/palm/palm_id1_r1.jpg¨
- std::string image_probe = ¨assets/palm/palm_id1_r2.jpg¨
- std::string image_non_match = ¨assets/palm/palm_id2_l1.jpg¨
- std::string solver_palm_detect = SOLVER_PALM_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- std::string solver_palm_extraction = SOLVER_PALM_EXTRACTION
- std::string solver_palm_landmarks = SOLVER_PALM_DETECT
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- float identification_threshold = 0.6f

9.19.1 Function Documentation

9.19.1.1 extract_template()

```
SFEPalmTemplate extract_template (
            std::string image_path,
```

```
            SFESolver & detector_solver,
            SFESolver & landmarks_solver,
            SFESolver & extraction_solver )
```

[Palm Detect]
[Palm Detect]
[Palm Landmarks]
[Palm Landmarks]
[Palm Template Extract]
[Palm Template Extract]
Definition at line 33 of file example_palm_identify.cpp.

```
00034
     {
00035
00036   SFEImage image{};
00037   DEFER(sfeImageFree(image));
00038   SFEError error{};
00039   { // STAGE 1: Load image
00040     // Load image data from file
00041     auto image_data = utils::readFile(image_path);
00042
00043     // Decode image from data
00044     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00045     utils::checkError(error);
00046   }
00047
00049   SFEDetection detected_palm = {};
00050   { // STAGE 2: Detect palm in the image
00051
00052     // Detect palm in the image
00053     float detection_threshold = 0.1f;
00054     size_t detected_count = 1;
00055     error = sfeDetect(detector_solver, image, detection_threshold,
00056                       &detected_palm, &detected_count);
00057     utils::checkError(error);
00058
00059     if (detected_count > 0 && detected_palm.confidence < 0.5) {
00060       throw std::runtime_error("No palm detected in the probe image.");
00061     }
00062
00063     std::cout « "Found palm in the image. Extracting template..." « std::endl;
00064   }
00066
00068   SFEPalmLandmarks landmarks = {};
00069   { // STAGE 3: Extract palm landmarks
00070     // Extract palm landmarks (hand position, orientation, keypoints)
00071     // These are required for template extraction
00072     error = sfePalmLandmarks(landmarks_solver, image, &detected_palm, &landmarks);
00073     utils::checkError(error);
00074   }
00076
00078   SFEPalmTemplate palm_template;
00079   { // STAGE 4: Extract probe palm template
00080     // Use template extraction solver to create new palm template
00081     // Requires palm landmarks from previous step
00082     error = sfePalmTemplateExtract(extraction_solver, image, &landmarks,
00083                                    &palm_template);
00084     utils::checkError(error);
00085   }
00087   return palm_template;
00088 }
```

9.19.1.2 main()

```
int main (
            int argc,
            char * argv[] )
```

Main fuction is used to parse command line arguments.
[Palm Template Version]
[Palm Template Version]
[Palm Template Identify]
[Palm Template Identify]
[Palm Template Match]
[Palm Template Match]
Definition at line 104 of file example_palm_identify.cpp.

```
00104                                {
00105
```

```
00106   { // STAGE 0: Parse command line arguments
00107     // Map to store argument values
00108     std::unordered_map<std::string, std::string> args;
00109
00110     // Parse command line arguments
00111     for (int i = 1; i < argc; ++i) {
00112       std::string arg = argv[i];
00113       if (arg[0] == '-') {
00114         if (arg == "-h") {
00115           printHelp();
00116           return 0;
00117         }
00118         // Check if there's a next argument and it isn't another option
00119         if (i + 1 < argc && argv[i + 1][0] != '-') {
00120           args[arg] = argv[++i];
00121         } else {
00122           std::cerr « "Option " « arg « " requires a value." « std::endl;
00123           return 1;
00124         }
00125       } else {
00126         std::cerr « "Unknown option: " « arg « std::endl;
00127         return 1;
00128       }
00129     }
00130
00131     // Assign values to variables based on parsed arguments
00132     if (args.count("-p"))
00133       image_probe = args["-p"];
00134     if (args.count("-g"))
00135       image_match = args["-g"];
00136     if (args.count("-d"))
00137       solver_palm_detect = args["-d"];
00138     if (args.count("-l"))
00139       solver_palm_landmarks = args["-l"];
00140     if (args.count("-e"))
00141       solver_palm_extraction = args["-e"];
00142     if (args.count("-i"))
00143       identification_threshold = std::stof(args["-i"]);
00144
00145     utils::printFormatted("EXAMPLE PARAMETERS");
00146     // Print resource names
00147     std::cout « "Probe image: " « image_probe « std::endl;
00148     std::cout « "Matching image: " « image_match « std::endl;
00149
00150     // Print solver names
00151     std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
00152     std::cout « "Palm landmarks solver: " « solver_palm_landmarks « std::endl;
00153     std::cout « "Palm extraction solver: " « solver_palm_extraction « std::endl;
00154
00155     // Print other options
00156     std::cout « "Identification threshold: " « identification_threshold
00157               « std::endl;
00158   }
00159
00160   utils::printToolkitInfo();
00161
00162   SFEError error{};
00163   SFESolver detector_solver{};
00164   DEFER(sfeSolverFree(detector_solver));
00165   SFESolver landmarks_solver{};
00166   DEFER(sfeSolverFree(landmarks_solver));
00167   SFESolver extraction_solver{};
00168   DEFER(sfeSolverFree(extraction_solver));
00169   { // STAGE 1.1: loading solvers
00170     utils::printFormatted("LOADING SOLVERS");
00171     // Initialize Palm detection solver
00172     error = sfeSolverCreate(
00173         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00174         SOLVER_PARAMETERS.size(), &detector_solver);
00175     utils::checkError(error);
00176     // Initialize Palm landmarks solver (using same solver as detection)
00177     error = sfeSolverCreate(
00178         solver_palm_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00179         SOLVER_PARAMETERS.size(), &landmarks_solver);
00180     utils::checkError(error);
00181     // Initialize Palm extraction solver
00182     error = sfeSolverCreate(
00183         solver_palm_extraction.c_str(), SOLVER_PARAMETERS.data(),
00184         SOLVER_PARAMETERS.size(), &extraction_solver);
00185     utils::checkError(error);
00186   }
00187
00188   SFEPalmTemplate probe_palm_template;
00189   { // STAGE 1: Extract probe palm template
00190     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00191
00192     probe_palm_template = extract_template(image_probe, detector_solver, landmarks_solver,
```

```
      extraction_solver);
00193   }
00194
00195   { // STAGE 1.1: (optional) Print template attributes
00196     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00197
00199     SFEPalmTemplateVersion version = {};
00200     error = sfePalmTemplateVersion(&probe_palm_template, &version);
00201     utils::checkError(error);
00202
00203     std::cout « "Version of the probe template: " « version.major « '.'
00204               « version.minor « '.' « version.patch « std::endl;
00206   }
00207
00208   size_t gallery_size = 10;
00209   std::vector<SFEPalmTemplate> palm_template_gallery(gallery_size);
00210   { // STAGE 2: Create palm templates gallery
00211     utils::printFormatted("PALM GALLERY EXTRACTION");
00212
00213     auto matching_palm_template =
00214         extract_template(image_match, detector_solver, landmarks_solver, extraction_solver);
00215
00216     auto non_matching_palm_template =
00217         extract_template(image_non_match, detector_solver, landmarks_solver, extraction_solver);
00218
00219     // Fill the gallery with non-matching templates for demonstration purposes
00220     for (size_t i = 0; i < gallery_size; i++) {
00221       palm_template_gallery[i] = non_matching_palm_template;
00222     }
00223
00224     // Add matching template to the gallery at index 5
00225     palm_template_gallery[5] = matching_palm_template;
00226   }
00227
00229   size_t candidate_count = 1;
00230   std::vector<SFETemplateIdentificationCandidate> identification_results(
00231       candidate_count);
00232   { // STAGE 3: Identify the probe template
00233     utils::printFormatted("1:N IDENTIFICATION");
00234
00235     error = sfePalmTemplateIdentify(
00236         &probe_palm_template, palm_template_gallery.data(),
00237         palm_template_gallery.size(), identification_threshold,
00238         identification_results.data(), &candidate_count, 4);
00239     utils::checkError(error);
00240
00241     if (candidate_count == 0) {
00242       std::cout « "No candidates found above identification threshold "
00243                 « identification_threshold « std::endl;
00244       return 0;
00245     }
00246
00247     std::cout « "Found " « identification_results.size()
00248               « " candidates above identification threshold "
00249               « identification_threshold « std::endl;
00250     for (auto &result : identification_results)
00251       std::cout « "Template index: #" « result.index
00252                 « ", score: " « result.score « std::endl;
00253   }
00255
00257   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00258     utils::printFormatted("1:1 MATCHING");
00259
00260     if (identification_results.size() == 0) {
00261       std::cout « "No candidates found above identification threshold "
00262                 « identification_threshold « std::endl;
00263       return 0;
00264     }
00265
00266     // Since it's sorted vector by score, the best candidate is the first one
00267     auto match_palm_template =
00268         palm_template_gallery[identification_results[0].index];
00269
00270     float match_confidence;
00271     error = sfePalmTemplateMatch(&probe_palm_template, &match_palm_template,
00272                                  &match_confidence);
00273     utils::checkError(error);
00274
00275     std::cout
00276         « "Matching score of probe template with the best template, score: "
00277         « match_confidence « std::endl;
00278   }
00280
00281   utils::printFormatted("FINISHED");
00282 }
```

9.19.1.3 printHelp()

```
void printHelp ( )
```

Definition at line 90 of file example_palm_identify.cpp.

```
00090                 {
00091   std::cout « "Help: Usage of the program." « std::endl;
00092   std::cout « "Options:" « std::endl;
00093   std::cout « "-h: Display help." « std::endl;
00094   std::cout « "-p: Probe image file." « std::endl;
00095   std::cout « "-g: Matching image file. Our \"palm database\"." « std::endl;
00096     std::cout « "-d: Path to detector solver." « std::endl;
00097     std::cout « "-l: Path to landmarks solver." « std::endl;
00098     std::cout « "-e: Path to template extraction solver." « std::endl;
00099     std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00100     std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00101 }
```

9.19.2 Variable Documentation

9.19.2.1 identification_threshold

```
float identification_threshold = 0.6f
```

Definition at line 31 of file example_palm_identify.cpp.

9.19.2.2 image_match

```
std::string image_match = ¨assets/palm/palm_id1_r1.jpg¨
```

Definition at line 18 of file example_palm_identify.cpp.

9.19.2.3 image_non_match

```
std::string image_non_match = ¨assets/palm/palm_id2_l1.jpg¨
```

Definition at line 20 of file example_palm_identify.cpp.

9.19.2.4 image_probe

```
std::string image_probe = ¨assets/palm/palm_id1_r2.jpg¨
```

Definition at line 19 of file example_palm_identify.cpp.

9.19.2.5 solver_palm_detect

```
std::string solver_palm_detect = SOLVER_PALM_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.

Examples

example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 23 of file example_palm_identify.cpp.

9.19.2.6 solver_palm_extraction

```
std::string solver_palm_extraction = SOLVER_PALM_EXTRACTION
```

Examples

example_palm_identify.cpp.

Definition at line 24 of file example_palm_identify.cpp.

9.19.2.7 solver_palm_landmarks

```
std::string solver_palm_landmarks = SOLVER_PALM_DETECT
```

Examples

example_palm_identify.cpp.

Definition at line 27 of file example_palm_identify.cpp.

9.19.2.8 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Definition at line 29 of file example_palm_identify.cpp.

9.20 example_palm_identify.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_palm.h"
00007 #include <regex>
00008 #include <vector>
00009
00010 #include <iomanip>
00011 #include <sstream>
00012
00013 #include "annotate.hpp"
00014 #include "solvers.h"
00015 #include "utils.hpp"
00016 #include <unordered_map>
00017
00018 std::string image_match = "assets/palm/palm_id1_r1.jpg";
00019 std::string image_probe = "assets/palm/palm_id1_r2.jpg";
00020 std::string image_non_match = "assets/palm/palm_id2_l1.jpg";
00021
00023 std::string solver_palm_detect = SOLVER_PALM_DETECT;
00024 std::string solver_palm_extraction = SOLVER_PALM_EXTRACTION;
00025
00026 // NOTE: Using the same solver for landmarks as for detection, this is intentional.
00027 std::string solver_palm_landmarks = SOLVER_PALM_DETECT;
00028
00029 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00030
00031 float identification_threshold = 0.6f;
00032
00033 SFEPalmTemplate extract_template(std::string image_path,
00034                                  SFESolver &detector_solver, SFESolver &landmarks_solver, SFESolver
      &extraction_solver) {
00035
00036   SFEImage image{};
00037   DEFER(sfeImageFree(image));
00038   SFEError error{};
00039   { // STAGE 1: Load image
00040     // Load image data from file
00041     auto image_data = utils::readFile(image_path);
00042
00043     // Decode image from data
00044     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00045     utils::checkError(error);
00046   }
00047
00049   SFEDetection detected_palm = {};
00050   { // STAGE 2: Detect palm in the image
00051
00052     // Detect palm in the image
00053     float detection_threshold = 0.1f;
00054     size_t detected_count = 1;
00055     error = sfeDetect(detector_solver, image, detection_threshold,
00056                       &detected_palm, &detected_count);
00057     utils::checkError(error);
00058
00059     if (detected_count > 0 && detected_palm.confidence < 0.5) {
00060       throw std::runtime_error("No palm detected in the probe image.");
00061     }
00062
00063     std::cout « "Found palm in the image. Extracting template..." « std::endl;
00064   }
00066
00068   SFEPalmLandmarks landmarks = {};
00069   { // STAGE 3: Extract palm landmarks
00070     // Extract palm landmarks (hand position, orientation, keypoints)
00071     // These are required for template extraction
00072     error = sfePalmLandmarks(landmarks_solver, image, &detected_palm, &landmarks);
00073     utils::checkError(error);
00074   }
00076
00078   SFEPalmTemplate palm_template;
00079   { // STAGE 4: Extract probe palm template
00080     // Use template extraction solver to create new palm template
00081     // Requires palm landmarks from previous step
00082     error = sfePalmTemplateExtract(extraction_solver, image, &landmarks,
00083                                    &palm_template);
```

```
00084     utils::checkError(error);
00085   }
00087   return palm_template;
00088 }
00089
00090 void printHelp() {
00091   std::cout « "Help: Usage of the program." « std::endl;
00092   std::cout « "Options:" « std::endl;
00093   std::cout « "-h: Display help." « std::endl;
00094   std::cout « "-p: Probe image file." « std::endl;
00095   std::cout « "-g: Matching image file. Our \"palm database\"." « std::endl;
00096     std::cout « "-d: Path to detector solver." « std::endl;
00097     std::cout « "-l: Path to landmarks solver." « std::endl;
00098     std::cout « "-e: Path to template extraction solver." « std::endl;
00099     std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00100     std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00101 }
00102
00104 int main(int argc, char *argv[]) {
00105
00106   { // STAGE 0: Parse command line arguments
00107     // Map to store argument values
00108     std::unordered_map<std::string, std::string> args;
00109
00110     // Parse command line arguments
00111     for (int i = 1; i < argc; ++i) {
00112       std::string arg = argv[i];
00113       if (arg[0] == '-') {
00114         if (arg == "-h") {
00115           printHelp();
00116           return 0;
00117         }
00118         // Check if there's a next argument and it isn't another option
00119         if (i + 1 < argc && argv[i + 1][0] != '-') {
00120           args[arg] = argv[++i];
00121         } else {
00122           std::cerr « "Option " « arg « " requires a value." « std::endl;
00123           return 1;
00124         }
00125       } else {
00126         std::cerr « "Unknown option: " « arg « std::endl;
00127         return 1;
00128       }
00129     }
00130
00131     // Assign values to variables based on parsed arguments
00132     if (args.count("-p"))
00133       image_probe = args["-p"];
00134     if (args.count("-g"))
00135       image_match = args["-g"];
00136     if (args.count("-d"))
00137       solver_palm_detect = args["-d"];
00138     if (args.count("-l"))
00139       solver_palm_landmarks = args["-l"];
00140     if (args.count("-e"))
00141       solver_palm_extraction = args["-e"];
00142     if (args.count("-i"))
00143       identification_threshold = std::stof(args["-i"]);
00144
00145     utils::printFormatted("EXAMPLE PARAMETERS");
00146     // Print resource names
00147     std::cout « "Probe image: " « image_probe « std::endl;
00148     std::cout « "Matching image: " « image_match « std::endl;
00149
00150     // Print solver names
00151     std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
00152     std::cout « "Palm landmarks solver: " « solver_palm_landmarks « std::endl;
00153     std::cout « "Palm extraction solver: " « solver_palm_extraction « std::endl;
00154
00155     // Print other options
00156     std::cout « "Identification threshold: " « identification_threshold
00157               « std::endl;
00158   }
00159
00160   utils::printToolkitInfo();
00161
00162   SFEError error{};
00163   SFESolver detector_solver{};
00164   DEFER(sfeSolverFree(detector_solver));
00165   SFESolver landmarks_solver{};
00166   DEFER(sfeSolverFree(landmarks_solver));
00167   SFESolver extraction_solver{};
00168   DEFER(sfeSolverFree(extraction_solver));
00169   { // STAGE 1.1: loading solvers
00170     utils::printFormatted("LOADING SOLVERS");
00171     // Initialize Palm detection solver
00172     error = sfeSolverCreate(
```

```
00173         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00174         SOLVER_PARAMETERS.size(), &detector_solver);
00175     utils::checkError(error);
00176     // Initialize Palm landmarks solver (using same solver as detection)
00177     error = sfeSolverCreate(
00178         solver_palm_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00179         SOLVER_PARAMETERS.size(), &landmarks_solver);
00180     utils::checkError(error);
00181     // Initialize Palm extraction solver
00182     error = sfeSolverCreate(
00183         solver_palm_extraction.c_str(), SOLVER_PARAMETERS.data(),
00184         SOLVER_PARAMETERS.size(), &extraction_solver);
00185     utils::checkError(error);
00186   }
00187
00188   SFEPalmTemplate probe_palm_template;
00189   { // STAGE 1: Extract probe palm template
00190     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00191
00192     probe_palm_template = extract_template(image_probe, detector_solver, landmarks_solver,
      extraction_solver);
00193   }
00194
00195   { // STAGE 1.1: (optional) Print template attributes
00196     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00197
00199     SFEPalmTemplateVersion version = {};
00200     error = sfePalmTemplateVersion(&probe_palm_template, &version);
00201     utils::checkError(error);
00202
00203     std::cout « "Version of the probe template: " « version.major « '.'
00204               « version.minor « '.' « version.patch « std::endl;
00206   }
00207
00208   size_t gallery_size = 10;
00209   std::vector<SFEPalmTemplate> palm_template_gallery(gallery_size);
00210   { // STAGE 2: Create palm templates gallery
00211     utils::printFormatted("PALM GALLERY EXTRACTION");
00212
00213     auto matching_palm_template =
00214         extract_template(image_match, detector_solver, landmarks_solver, extraction_solver);
00215
00216     auto non_matching_palm_template =
00217         extract_template(image_non_match, detector_solver, landmarks_solver, extraction_solver);
00218
00219     // Fill the gallery with non-matching templates for demonstration purposes
00220     for (size_t i = 0; i < gallery_size; i++) {
00221       palm_template_gallery[i] = non_matching_palm_template;
00222     }
00223
00224     // Add matching template to the gallery at index 5
00225     palm_template_gallery[5] = matching_palm_template;
00226   }
00227
00229   size_t candidate_count = 1;
00230   std::vector<SFETemplateIdentificationCandidate> identification_results(
00231       candidate_count);
00232   { // STAGE 3: Identify the probe template
00233     utils::printFormatted("1:N IDENTIFICATION");
00234
00235     error = sfePalmTemplateIdentify(
00236         &probe_palm_template, palm_template_gallery.data(),
00237         palm_template_gallery.size(), identification_threshold,
00238         identification_results.data(), &candidate_count, 4);
00239     utils::checkError(error);
00240
00241     if (candidate_count == 0) {
00242       std::cout « "No candidates found above identification threshold "
00243                 « identification_threshold « std::endl;
00244       return 0;
00245     }
00246
00247     std::cout « "Found " « identification_results.size()
00248               « " candidates above identification threshold "
00249               « identification_threshold « std::endl;
00250     for (auto &result : identification_results)
00251       std::cout « "Template index: #" « result.index
00252                 « ", score: " « result.score « std::endl;
00253   }
00255
00257   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00258     utils::printFormatted("1:1 MATCHING");
00259
00260     if (identification_results.size() == 0) {
00261       std::cout « "No candidates found above identification threshold "
00262                 « identification_threshold « std::endl;
00263       return 0;
```

```
00264     }
00265
00266     // Since it's sorted vector by score, the best candidate is the first one
00267     auto match_palm_template =
00268         palm_template_gallery[identification_results[0].index];
00269
00270     float match_confidence;
00271     error = sfePalmTemplateMatch(&probe_palm_template, &match_palm_template,
00272                                  &match_confidence);
00273     utils::checkError(error);
00274
00275     std::cout
00276         « "Matching score of probe template with the best template, score: "
00277         « match_confidence « std::endl;
00278   }
00280
00281   utils::printFormatted("FINISHED");
00282 }
```

9.21 D:/glab/1512fd1724a/1/example/example_palm_liveness.cpp File Reference

```
#include ¨sfe_toolkit/sfe_core.h¨
#include ¨sfe_toolkit/sfe_palm.h¨
#include <iomanip>
#include <regex>
#include <sstream>
#include <vector>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include <unordered_map>
```

Functions

- void printPalmDetectionInfo (SFEDetection &detected_palm)
- void printHelp ()
- int main (int argc, char ∗argv[ ])

  Main fuction is used to parse command line arguments.

Variables

- std::string image_probe = ¨assets/palm/palm_id1_r1.jpg¨
- std::string solver_palm_detect = SOLVER_PALM_DETECT

  Solvers to use in example, the defaults are filled in by CMake.
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- float detection_threshold = 0.1f

9.21.1 Function Documentation

9.21.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main fuction is used to parse command line arguments.
[Image Load]
[Image Load]
[Palm Detect]
[Palm Detect]
[Palm Liveness]
[Palm Liveness]

Definition at line 53 of file example_palm_liveness.cpp.

```
00053                                   {
00054
00055   { // STAGE: 0 Parse command line arguments
00056     // Map to store argument values
00057     std::unordered_map<std::string, std::string> args;
00058
00059     // Parse command line arguments
00060     for (int i = 1; i < argc; ++i) {
00061       std::string arg = argv[i];
00062       if (arg[0] == '-') {
00063         if (arg == "-h") {
00064           printHelp();
00065           return 0;
00066         }
00067         // Check if there's a next argument and it isn't another option
00068         if (i + 1 < argc && argv[i + 1][0] != '-') {
00069           args[arg] = argv[++i];
00070         } else {
00071           std::cerr << "Option " << arg << " requires a value." << std::endl;
00072           return 1;
00073         }
00074       } else {
00075         std::cerr << "Unknown option: " << arg << std::endl;
00076         return 1;
00077       }
00078     }
00079
00080     // Assign values to variables based on parsed arguments
00081     if (args.count("-p"))
00082       image_probe = args["-p"];
00083     if (args.count("-d"))
00084       solver_palm_detect = args["-d"];
00085     if (args.count("-i"))
00086       solver_palm_liveness = args["-i"];
00087     if (args.count("-t"))
00088       detection_threshold = std::stof(args["-t"]);
00089
00090     utils::printFormatted("PARAMETERS");
00091     // Print resource names
00092     std::cout << "Probe image: " << image_probe << std::endl;
00093
00094     // Print solver names
00095     std::cout << "Palm detection solver: " << solver_palm_detect << std::endl;
00096     std::cout << "Palm liveness solver: " << solver_palm_liveness << std::endl;
00097
00098     // Print other options
00099     std::cout << "Detection threshold: " << detection_threshold << std::endl;
00100   }
00101
00102   utils::printToolkitInfo();
00103
00104   SFEError error{};
00105
00106   SFESolver detector_solver{};
00107   DEFER(sfeSolverFree(detector_solver));
00108   SFESolver liveness_solver{};
00109   DEFER(sfeSolverFree(liveness_solver));
00110   { // STAGE 1: loading solvers
00111     utils::printFormatted("LOADING solvers");
00112     // Initialize palm detection solver
00113     error = sfeSolverCreate(
00114         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00115         SOLVER_PARAMETERS.size(), &detector_solver);
00116     utils::checkError(error);
00117
00118     // Initialize palm liveness solver
00119     error = sfeSolverCreate(
00120         solver_palm_liveness.c_str(), SOLVER_PARAMETERS.data(),
00121         SOLVER_PARAMETERS.size(), &liveness_solver);
00122     utils::checkError(error);
00123   }
00124
00125   SFEImage image{};
00126   DEFER(sfeImageFree(image));
00127   { // STAGE 2: Load image
00128
00129     utils::printFormatted("PALM DETECTION");
00131     // Load image data from file
00132     auto image_data = utils::readFile(image_probe);
00133
00134     // Decode image from data
00135     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00136     utils::checkError(error);
00138   }
00140   SFEDetection detected_palm = {};
00141   { // STAGE 3: Detect palm in the image
```

```
00142     size_t detection_count = 1;
00143     // Detect palm in the image
00144     error = sfeDetect(detector_solver, image, detection_threshold,
00145                       &detected_palm, &detection_count);
00146     utils::checkError(error);
00147
00148     if (detection_count == 0) {
00149       std::cout « "No palm detected in the probe image." « std::endl;
00150       return 0;
00151     } else {
00152       std::cout « "Found " « detection_count
00153                 « " palm(s) in the probe image. Using the palm with highest "
00154                    "confidence."
00155                 « std::endl;
00156     }
00157   }
00159
00160   printPalmDetectionInfo(detected_palm);
00161
00163   float liveness_score = 0.0f;
00164   { // STAGE 4: Liveness check
00165     utils::printFormatted("LIVENESS CHECK");
00166     error = sfePalmLivenessPassive(liveness_solver, image, &detected_palm,
00167                                    &liveness_score);
00168     utils::checkError(error);
00169   }
00171   { // STAGE 5: Print the liveness score
00172     std::cout « "Liveness score is " « liveness_score;
00173     // Recommended threshold for liveness score, see EER threshold for palm liveness in the
      documentation
00174     static const float LIVENESS_THRESHOLD = 0.8189f;
00175     if (liveness_score < LIVENESS_THRESHOLD) {
00176       std::cout « " which is below " « LIVENESS_THRESHOLD
00177                 « " threshold. Palm is spoof." « std::endl;
00178     } else {
00179       std::cout « " which is above " « LIVENESS_THRESHOLD
00180                 « " threshold. Palm is genuine." « std::endl;
00181     }
00182   }
00183
00184   { // STAGE 6: Annotate the image
00185     // Render bounding box
00186     std::stringstream label;
00187     label « "Palm" « std::fixed « std::setprecision(2)
00188           « " d:" « detected_palm.confidence « " l:" « liveness_score;
00189
00190     annotate::labelBox(image, label.str(), detected_palm.bounding_box,
00191                        annotate::WHITE, annotate::GREEN);
00192
00193     // Save annotated image
00194     size_t size = image.width * image.height * 3;
00195     auto png_file = std::vector<unsigned char>(size);
00196     error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00197     utils::checkError(error);
00198     png_file.resize(size);
00199     utils::saveFile("palm_liveness.png", png_file);
00200
00201     std::cout « std::endl;
00202     std::cout « "Annotated image saved to palm_liveness.png" « std::endl;
00203   }
00204   utils::printFormatted("FINISHED");
00205 }
```

9.21.1.2 printHelp()

```
void printHelp ( )
```

Definition at line 42 of file example_palm_liveness.cpp.

```
00042                {
00043   std::cout « "Help: Usage of the program." « std::endl;
00044   std::cout « "Options:" « std::endl;
00045   std::cout « "-h: Display help." « std::endl;
00046   std::cout « "-p: Probe image file." « std::endl;
00047   std::cout « "-d: Path to detector solver." « std::endl;
00048   std::cout « "-i: Path to liveness solver." « std::endl;
00049   std::cout « "-t: Palm detection threshold. <0,1>" « std::endl;
00050 }
```

9.21.1.3 printPalmDetectionInfo()

```
void printPalmDetectionInfo (
            SFEDetection & detected_palm )  [inline]
```

Examples

example_palm_attributes.cpp, and example_palm_liveness.cpp.

Definition at line 29 of file example_palm_liveness.cpp.

```
00029                                                                 {
00030   std::cout « "SFEDetection{ " « std::endl;
00031   std::cout « "  confidence: " « detected_palm.confidence « std::endl;
00032   std::cout « "  SFEBoundingBox{ " « std::endl;
00033   std::cout « "    x: " « detected_palm.bounding_box.x « std::endl;
00034   std::cout « "    y: " « detected_palm.bounding_box.y « std::endl;
00035   std::cout « "    width: " « detected_palm.bounding_box.width « std::endl;
00036   std::cout « "    height: " « detected_palm.bounding_box.height « std::endl;
00037   std::cout « "    }" « std::endl;
00038   std::cout « "  }" « std::endl;
00039   std::cout « std::endl;
00040 }
```

9.21.2 Variable Documentation

9.21.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 27 of file example_palm_liveness.cpp.

9.21.2.2 image_probe

```
std::string image_probe = ¨assets/palm/palm_id1_r1.jpg¨
```

Definition at line 17 of file example_palm_liveness.cpp.

9.21.2.3 solver_palm_detect

```
std::string solver_palm_detect = SOLVER_PALM_DETECT
```

Solvers to use in example, the defaults are filled in by CMake.

Definition at line 20 of file example_palm_liveness.cpp.

9.21.2.4 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
```

Definition at line 25 of file example_palm_liveness.cpp.

9.22 example_palm_liveness.cpp

Go to the documentation of this file.

```
00001
00005 #include "sfe_toolkit/sfe_core.h"
00006 #include "sfe_toolkit/sfe_palm.h"
00007 #include <iomanip>
00008 #include <regex>
00009 #include <sstream>
00010 #include <vector>
00011
00012 #include "annotate.hpp"
00013 #include "solvers.h"
00014 #include "utils.hpp"
00015 #include <unordered_map>
00016
00017 std::string image_probe = "assets/palm/palm_id1_r1.jpg";
00018
00020 std::string solver_palm_detect = SOLVER_PALM_DETECT;
00021 #ifdef SOLVER_PALM_LIVENESS
00022 std::string solver_palm_liveness = SOLVER_PALM_LIVENESS;
00023 #endif // SOLVER_PALM_LIVENESS
00024
00025 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00026
00027 float detection_threshold = 0.1f;
00028
00029 inline void printPalmDetectionInfo(SFEDetection &detected_palm) {
00030   std::cout « "SFEDetection{ " « std::endl;
00031   std::cout « "  confidence: " « detected_palm.confidence « std::endl;
00032   std::cout « "  SFEBoundingBox{ " « std::endl;
00033   std::cout « "    x: " « detected_palm.bounding_box.x « std::endl;
```

```
00034   std::cout « "    y: " « detected_palm.bounding_box.y « std::endl;
00035   std::cout « "    width: " « detected_palm.bounding_box.width « std::endl;
00036   std::cout « "    height: " « detected_palm.bounding_box.height « std::endl;
00037   std::cout « "    }" « std::endl;
00038   std::cout « "  }" « std::endl;
00039   std::cout « std::endl;
00040 }
00041
00042 void printHelp() {
00043   std::cout « "Help: Usage of the program." « std::endl;
00044   std::cout « "Options:" « std::endl;
00045   std::cout « "-h: Display help." « std::endl;
00046   std::cout « "-p: Probe image file." « std::endl;
00047   std::cout « "-d: Path to detector solver." « std::endl;
00048   std::cout « "-i: Path to liveness solver." « std::endl;
00049   std::cout « "-t: Palm detection threshold. <0,1>" « std::endl;
00050 }
00051
00053 int main(int argc, char *argv[]) {
00054
00055   { // STAGE: 0 Parse command line arguments
00056     // Map to store argument values
00057     std::unordered_map<std::string, std::string> args;
00058
00059     // Parse command line arguments
00060     for (int i = 1; i < argc; ++i) {
00061       std::string arg = argv[i];
00062       if (arg[0] == '-') {
00063         if (arg == "-h") {
00064           printHelp();
00065           return 0;
00066         }
00067         // Check if there's a next argument and it isn't another option
00068         if (i + 1 < argc && argv[i + 1][0] != '-') {
00069           args[arg] = argv[++i];
00070         } else {
00071           std::cerr « "Option " « arg « " requires a value." « std::endl;
00072           return 1;
00073         }
00074       } else {
00075         std::cerr « "Unknown option: " « arg « std::endl;
00076         return 1;
00077       }
00078     }
00079
00080     // Assign values to variables based on parsed arguments
00081     if (args.count("-p"))
00082       image_probe = args["-p"];
00083     if (args.count("-d"))
00084       solver_palm_detect = args["-d"];
00085     if (args.count("-i"))
00086       solver_palm_liveness = args["-i"];
00087     if (args.count("-t"))
00088       detection_threshold = std::stof(args["-t"]);
00089
00090     utils::printFormatted("PARAMETERS");
00091     // Print resource names
00092     std::cout « "Probe image: " « image_probe « std::endl;
00093
00094     // Print solver names
00095     std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
00096     std::cout « "Palm liveness solver: " « solver_palm_liveness « std::endl;
00097
00098     // Print other options
00099     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00100   }
00101
00102   utils::printToolkitInfo();
00103
00104   SFEError error{};
00105
00106   SFESolver detector_solver{};
00107   DEFER(sfeSolverFree(detector_solver));
00108   SFESolver liveness_solver{};
00109   DEFER(sfeSolverFree(liveness_solver));
00110   { // STAGE 1: loading solvers
00111     utils::printFormatted("LOADING solvers");
00112     // Initialize palm detection solver
00113     error = sfeSolverCreate(
00114         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00115         SOLVER_PARAMETERS.size(), &detector_solver);
00116     utils::checkError(error);
00117
00118     // Initialize palm liveness solver
00119     error = sfeSolverCreate(
00120         solver_palm_liveness.c_str(), SOLVER_PARAMETERS.data(),
00121         SOLVER_PARAMETERS.size(), &liveness_solver);
```

```
00122     utils::checkError(error);
00123   }
00124
00125   SFEImage image{};
00126   DEFER(sfeImageFree(image));
00127   { // STAGE 2: Load image
00128
00129     utils::printFormatted("PALM DETECTION");
00131     // Load image data from file
00132     auto image_data = utils::readFile(image_probe);
00133
00134     // Decode image from data
00135     error = sfeImageDecode(image_data.data(), image_data.size(), &image);
00136     utils::checkError(error);
00138   }
00140   SFEDetection detected_palm = {};
00141   { // STAGE 3: Detect palm in the image
00142     size_t detection_count = 1;
00143     // Detect palm in the image
00144     error = sfeDetect(detector_solver, image, detection_threshold,
00145                       &detected_palm, &detection_count);
00146     utils::checkError(error);
00147
00148     if (detection_count == 0) {
00149       std::cout « "No palm detected in the probe image." « std::endl;
00150       return 0;
00151     } else {
00152       std::cout « "Found " « detection_count
00153                 « " palm(s) in the probe image. Using the palm with highest "
00154                    "confidence."
00155                 « std::endl;
00156     }
00157   }
00159
00160   printPalmDetectionInfo(detected_palm);
00161
00163   float liveness_score = 0.0f;
00164   { // STAGE 4: Liveness check
00165     utils::printFormatted("LIVENESS CHECK");
00166     error = sfePalmLivenessPassive(liveness_solver, image, &detected_palm,
00167                                    &liveness_score);
00168     utils::checkError(error);
00169   }
00171   { // STAGE 5: Print the liveness score
00172     std::cout « "Liveness score is " « liveness_score;
00173     // Recommended threshold for liveness score, see EER threshold for palm liveness in the
      documentation
00174     static const float LIVENESS_THRESHOLD = 0.8189f;
00175     if (liveness_score < LIVENESS_THRESHOLD) {
00176       std::cout « " which is below " « LIVENESS_THRESHOLD
00177                 « " threshold. Palm is spoof." « std::endl;
00178     } else {
00179       std::cout « " which is above " « LIVENESS_THRESHOLD
00180                 « " threshold. Palm is genuine." « std::endl;
00181     }
00182   }
00183
00184   { // STAGE 6: Annotate the image
00185     // Render bounding box
00186     std::stringstream label;
00187     label « "Palm" « std::fixed « std::setprecision(2)
00188           « " d:" « detected_palm.confidence « " l:" « liveness_score;
00189
00190     annotate::labelBox(image, label.str(), detected_palm.bounding_box,
00191                        annotate::WHITE, annotate::GREEN);
00192
00193     // Save annotated image
00194     size_t size = image.width * image.height * 3;
00195     auto png_file = std::vector<unsigned char>(size);
00196     error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00197     utils::checkError(error);
00198     png_file.resize(size);
00199     utils::saveFile("palm_liveness.png", png_file);
00200
00201     std::cout « std::endl;
00202     std::cout « "Annotated image saved to palm_liveness.png" « std::endl;
00203   }
00204   utils::printFormatted("FINISHED");
00205 }
```

9.23 D:/glab/1512fd1724a/1/example/example_person_attributes.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <sfe_toolkit/sfe_core.h>
#include <sfe_toolkit/sfe_object.h>
#include <sfe_toolkit/sfe_person.h>
#include ¨utils.hpp¨
#include ¨solvers.h¨
#include ¨annotate.hpp¨
#include ¨strings.hpp¨
```

Functions

- int main ()

Variables

- const auto SOLVER_PARAMETERS
- std::string image_path = ¨assets/person/person_attributes.jpg¨
- std::string solver_object_detect = SOLVER_OBJECT_DETECT
- std::string solver_person_attributes = SOLVER_PERSON_ATTRIBUTES
- std::string solver_person_pose = SOLVER_PERSON_POSE

9.23.1 Function Documentation

9.23.1.1 main()

```
int main ( )
```

Definition at line 25 of file example_person_attributes.cpp.

```
00025           {
00026     SFEError error;
00027
00028     SFEImage image{};
00029     DEFER(sfeImageFree(image));
00030     { // STAGE 1: Load Image
00031         auto encoded_image = utils::readFile(image_path);
00032         error = sfeImageDecode(encoded_image.data(), encoded_image.size(), &image);
00033         utils::checkError(error);
00034
00035         std::cout « "Image size: " « image.width « "x" « image.height « std::endl;
00036     }
00037
00038     std::vector<SFEDetection> detections(10);
00039     { // STAGE 2: Run object detection
00040         // Initialize detection solver
00041         SFESolver detection_solver{};
00042         DEFER(sfeSolverFree(detection_solver));
00043         error = sfeSolverCreate(solver_object_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detection_solver);
00044         utils::checkError(error);
00045
00046         // Run detection for up to 10 objects
00047         size_t detection_count = detections.size();
00048         float detection_threshold = 0.3;
00049         error = sfeDetect(detection_solver,
00050             image,
00051             detection_threshold,
00052             detections.data(), &detection_count);
00053         utils::checkError(error);
00054         detections.resize(detection_count);
00055
00056         std::cout « "Detected " « detection_count « " objects" « std::endl;
00057     }
00058
00059     std::vector<size_t> detection_classes;
00060     { // STAGE 3: Check for best detection classes
```

```
00061         for (size_t i = 0; i < detections.size(); i++) {
00062             auto& detection = detections[i];
00063             size_t best_class_index = 0;
00064             for (size_t c = 0; c < SFE_OBJECT_DETECT; c++) {
00065                 auto object = detection.modality_data.object.objects[c];
00066                 if (object.confidence < 0.01) continue;
00067                 if (object.confidence >
      detection.modality_data.object.objects[best_class_index].confidence) {
00068                     best_class_index = c;
00069                 }
00070
00071                 std::cout « "Detection " « i « " : " « OBJECT_CLASS_NAMES[c] « " confidence: " «
      std::fixed « std::setprecision(2) « detection.modality_data.object.objects[c].confidence « std::endl;
00072             }
00073             detection_classes.push_back(best_class_index);
00074         }
00075     }
00076
00077     size_t person_index;
00078     { // STAGE 4: Find first person in detections
00079         auto person_iter = std::find(detection_classes.begin(), detection_classes.end(),
      SFE_OBJECT_TYPE_PERSON);
00080         if (person_iter == detection_classes.end()) {
00081             std::cout « "No person detected" « std::endl;
00082             return 0;
00083         }
00084         person_index = std::distance(detection_classes.begin(), person_iter);
00085
00086         std::cout « "Person detected at index " « person_index « std::endl;
00087     }
00088
00089     std::array<SFEPersonAttribute, SFE_PERSON_ATTRIBUTE_COUNT> person_attributes;
00090     { // STAGE 5: Extract person attributes
00091         SFESolver attribute_solver{};
00092         DEFER(sfeSolverFree(attribute_solver));
00093         error = sfeSolverCreate(solver_person_attributes.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &attribute_solver);
00094         utils::checkError(error);
00095
00096         // Extract attributes for the detected person
00097         error = sfePersonAttributes(attribute_solver, image, &detections[person_index],
      person_attributes.data());
00098         utils::checkError(error);
00099
00100         for (auto& attribute : person_attributes) {
00101             if (attribute.confidence < 0.1) continue;
00102             std::cout « "Attribute " « PERSON_ATTRIBUTE_TYPE[attribute.attribute_type] « " confidence:
      " « std::fixed « std::setprecision(2) « attribute.confidence « std::endl;
00103         }
00104     }
00105
00106     std::array<SFEPersonPoseKeypoint, SFE_PERSON_POSE_KEYPOINT_COUNT> person_pose;
00107     { // STAGE 6: Person pose estimation
00108         SFESolver pose_solver;
00109         DEFER(sfeSolverFree(pose_solver));
00110         error = sfeSolverCreate(solver_person_pose.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &pose_solver);
00111         utils::checkError(error);
00112
00113         // Estimate pose for the detected person
00114         error = sfePersonPose(pose_solver, image, &detections[person_index], person_pose.data());
00115         utils::checkError(error);
00116
00117         for (auto& keypoint : person_pose) {
00118             if (keypoint.confidence < 0.1) continue;
00119             std::cout « "Keypoint " « keypoint.keypoint_type « " confidence: " « std::fixed «
      std::setprecision(2) « keypoint.confidence « std::endl;
00120         }
00121     }
00122
00123     { // END STAGE: Display results
00124         for (size_t i = 0; i < detections.size(); i++) {
00125             auto& detection = detections[i];
00126             auto& class_index = detection_classes[i];
00127             auto color = i == person_index ? annotate::GREEN : annotate::RED;
00128             // Render bounding box
00129             std::stringstream label;
00130             label « i « ": " « OBJECT_CLASS_NAMES[class_index] « std::fixed « std::setprecision(2)
00131                 « " c=" « detection.modality_data.object.objects[class_index].confidence
00132                 « " d=" « detection.confidence;
00133
00134             annotate::labelBox(image, label.str(), detection.bounding_box, annotate::WHITE, color);
00135         }
00136
00137         // Render pose points
00138         for (auto& keypoint : person_pose) {
00139             annotate::circle(image, keypoint.x * image.width, keypoint.y * image.height, 3,
```

```
      annotate::YELLOW);
00140         }
00141
00142         // Save annotated image
00143         size_t size = image.width * image.height * 3;
00144         auto png_file = std::vector<unsigned char>(size);
00145         error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00146         utils::checkError(error);
00147         png_file.resize(size);
00148         utils::saveFile("person_attributes.png", png_file);
00149     }
00150 }
```

9.23.2 Variable Documentation

9.23.2.1 image_path

```
std::string image_path = ¨assets/person/person_attributes.jpg¨
```

Definition at line 20 of file example_person_attributes.cpp.

9.23.2.2 solver_object_detect

```
std::string solver_object_detect = SOLVER_OBJECT_DETECT
```

Definition at line 21 of file example_person_attributes.cpp.

9.23.2.3 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS
```

Initial value:

```
= std::vector<SFESolverParameter>{
    SFESolverParameter { "intra_threads", "4" },
}
```

Definition at line 16 of file example_person_attributes.cpp.

9.23.2.4 solver_person_attributes

```
std::string solver_person_attributes = SOLVER_PERSON_ATTRIBUTES
```

Definition at line 22 of file example_person_attributes.cpp.

9.23.2.5 solver_person_pose

```
std::string solver_person_pose = SOLVER_PERSON_POSE
```

Definition at line 23 of file example_person_attributes.cpp.

9.24 example_person_attributes.cpp

Go to the documentation of this file.

```
00001 #include <sstream>
00002 #include <iomanip>
00003 #include <vector>
00004 #include <algorithm>
00005
00006 #include <sfe_toolkit/sfe_core.h>
00007 #include <sfe_toolkit/sfe_object.h>
00008 #include <sfe_toolkit/sfe_person.h>
00009
00010 #include "utils.hpp"
00011 #include "solvers.h"
00012 #include "annotate.hpp"
00013 #include "strings.hpp"
00014
00015 // Solver parameters, we are using ONNX Runtime CPU backend
00016 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{
00017     SFESolverParameter { "intra_threads", "4" },
00018 };
00019
00020 std::string image_path = "assets/person/person_attributes.jpg";
00021 std::string solver_object_detect = SOLVER_OBJECT_DETECT;
00022 std::string solver_person_attributes = SOLVER_PERSON_ATTRIBUTES;
00023 std::string solver_person_pose = SOLVER_PERSON_POSE;
00024
00025 int main() {
```

```
00026     SFEError error;
00027
00028     SFEImage image{};
00029     DEFER(sfeImageFree(image));
00030     { // STAGE 1: Load Image
00031         auto encoded_image = utils::readFile(image_path);
00032         error = sfeImageDecode(encoded_image.data(), encoded_image.size(), &image);
00033         utils::checkError(error);
00034
00035         std::cout « "Image size: " « image.width « "x" « image.height « std::endl;
00036     }
00037
00038     std::vector<SFEDetection> detections(10);
00039     { // STAGE 2: Run object detection
00040         // Initialize detection solver
00041         SFESolver detection_solver{};
00042         DEFER(sfeSolverFree(detection_solver));
00043         error = sfeSolverCreate(solver_object_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detection_solver);
00044         utils::checkError(error);
00045
00046         // Run detection for up to 10 objects
00047         size_t detection_count = detections.size();
00048         float detection_threshold = 0.3;
00049         error = sfeDetect(detection_solver,
00050             image,
00051             detection_threshold,
00052             detections.data(), &detection_count);
00053         utils::checkError(error);
00054         detections.resize(detection_count);
00055
00056         std::cout « "Detected " « detection_count « " objects" « std::endl;
00057     }
00058
00059     std::vector<size_t> detection_classes;
00060     { // STAGE 3: Check for best detection classes
00061         for (size_t i = 0; i < detections.size(); i++) {
00062             auto& detection = detections[i];
00063             size_t best_class_index = 0;
00064             for (size_t c = 0; c < SFE_OBJECT_DETECT; c++) {
00065                 auto object = detection.modality_data.object.objects[c];
00066                 if (object.confidence < 0.01) continue;
00067                 if (object.confidence >
      detection.modality_data.object.objects[best_class_index].confidence) {
00068                     best_class_index = c;
00069                 }
00070
00071                 std::cout « "Detection " « i « " : " « OBJECT_CLASS_NAMES[c] « " confidence: " «
      std::fixed « std::setprecision(2) « detection.modality_data.object.objects[c].confidence « std::endl;
00072             }
00073             detection_classes.push_back(best_class_index);
00074         }
00075     }
00076
00077     size_t person_index;
00078     { // STAGE 4: Find first person in detections
00079         auto person_iter = std::find(detection_classes.begin(), detection_classes.end(),
      SFE_OBJECT_TYPE_PERSON);
00080         if (person_iter == detection_classes.end()) {
00081             std::cout « "No person detected" « std::endl;
00082             return 0;
00083         }
00084         person_index = std::distance(detection_classes.begin(), person_iter);
00085
00086         std::cout « "Person detected at index " « person_index « std::endl;
00087     }
00088
00089     std::array<SFEPersonAttribute, SFE_PERSON_ATTRIBUTE_COUNT> person_attributes;
00090     { // STAGE 5: Extract person attributes
00091         SFESolver attribute_solver{};
00092         DEFER(sfeSolverFree(attribute_solver));
00093         error = sfeSolverCreate(solver_person_attributes.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &attribute_solver);
00094         utils::checkError(error);
00095
00096         // Extract attributes for the detected person
00097         error = sfePersonAttributes(attribute_solver, image, &detections[person_index],
      person_attributes.data());
00098         utils::checkError(error);
00099
00100         for (auto& attribute : person_attributes) {
00101             if (attribute.confidence < 0.1) continue;
00102             std::cout « "Attribute " « PERSON_ATTRIBUTE_TYPE[attribute.attribute_type] « " confidence:
      " « std::fixed « std::setprecision(2) « attribute.confidence « std::endl;
00103         }
00104     }
00105
```

```
00106     std::array<SFEPersonPoseKeypoint, SFE_PERSON_POSE_KEYPOINT_COUNT> person_pose;
00107     { // STAGE 6: Person pose estimation
00108         SFESolver pose_solver;
00109         DEFER(sfeSolverFree(pose_solver));
00110         error = sfeSolverCreate(solver_person_pose.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &pose_solver);
00111         utils::checkError(error);
00112 
00113         // Estimate pose for the detected person
00114         error = sfePersonPose(pose_solver, image, &detections[person_index], person_pose.data());
00115         utils::checkError(error);
00116 
00117         for (auto& keypoint : person_pose) {
00118             if (keypoint.confidence < 0.1) continue;
00119             std::cout << "Keypoint " << keypoint.keypoint_type << " confidence: " << std::fixed <<
      std::setprecision(2) << keypoint.confidence << std::endl;
00120         }
00121     }
00122 
00123     { // END STAGE: Display results
00124         for (size_t i = 0; i < detections.size(); i++) {
00125             auto& detection = detections[i];
00126             auto& class_index = detection_classes[i];
00127             auto color = i == person_index ? annotate::GREEN : annotate::RED;
00128             // Render bounding box
00129             std::stringstream label;
00130             label << i << ": " << OBJECT_CLASS_NAMES[class_index] << std::fixed << std::setprecision(2)
00131                 << " c=" << detection.modality_data.object.objects[class_index].confidence
00132                 << " d=" << detection.confidence;
00133 
00134             annotate::labelBox(image, label.str(), detection.bounding_box, annotate::WHITE, color);
00135         }
00136 
00137         // Render pose points
00138         for (auto& keypoint : person_pose) {
00139             annotate::circle(image, keypoint.x * image.width, keypoint.y * image.height, 3,
      annotate::YELLOW);
00140         }
00141 
00142         // Save annotated image
00143         size_t size = image.width * image.height * 3;
00144         auto png_file = std::vector<unsigned char>(size);
00145         error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00146         utils::checkError(error);
00147         png_file.resize(size);
00148         utils::saveFile("person_attributes.png", png_file);
00149     }
00150 }
```

9.25 D:/glab/1512fd1724a/1/example/example_person_detect.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <sfe_toolkit/sfe_core.h>
#include <sfe_toolkit/sfe_person.h>
#include ¨utils.hpp¨
#include ¨solvers.h¨
#include ¨annotate.hpp¨
#include ¨strings.hpp¨
```

Functions

- int main ()

Variables

- const auto SOLVER_PARAMETERS
- std::string image_path = ¨assets/person/person_detect_fisheye.jpg¨
- std::string solver_detect = SOLVER_PERSON_DETECT

9.25.1 Function Documentation

9.25.1.1 main()

```
int main ( )
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

Definition at line 22 of file example_person_detect.cpp.

```
00022           {
00023     SFEError error;
00024
00025     SFEImage image{};
00026     DEFER(sfeImageFree(image));
00027     { // STAGE 1: Load Image
00028         auto encoded_image = utils::readFile(image_path);
00029         error = sfeImageDecode(encoded_image.data(), encoded_image.size(), &image);
00030         utils::checkError(error);
00031
00032         std::cout « "Image size: " « image.width « "x" « image.height « std::endl;
00033     }
00034
00035     std::vector<SFEDetection> detections(10);
00036     { // STAGE 2: Run person detection
00037         // Initialize detection solver
00038         SFESolver detection_solver{};
00039         DEFER(sfeSolverFree(detection_solver));
00040         error = sfeSolverCreate(solver_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detection_solver);
00041         utils::checkError(error);
00042
00043         // Run detection for up to 10 persons
00044         size_t detection_count = detections.size();
00045         float detection_threshold = 0.3;
00046         error = sfeDetect(detection_solver,
00047             image,
00048             detection_threshold,
00049             detections.data(), &detection_count);
00050         utils::checkError(error);
00051         detections.resize(detection_count);
00052
00053         std::cout « "Detected " « detection_count « " persons" « std::endl;
00054     }
00055 }
```

9.25.2 Variable Documentation

9.25.2.1 image_path

```
std::string image_path = ¨assets/person/person_detect_fisheye.jpg¨
```

Examples

example_face_identify.cpp, example_face_identify_entity.cpp, example_iris_identify.cpp, and example_palm_identify.cpp.

Definition at line 19 of file example_person_detect.cpp.

9.25.2.2 solver_detect

```
std::string solver_detect = SOLVER_PERSON_DETECT
```

Definition at line 20 of file example_person_detect.cpp.

9.25.2.3 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS
```

Initial value:

```
= std::vector<SFESolverParameter>{
    SFESolverParameter { "intra_threads", "4" },
}
```

Definition at line 15 of file example_person_detect.cpp.

9.26 example_person_detect.cpp

Go to the documentation of this file.

```
00001 #include <sstream>
00002 #include <iomanip>
00003 #include <vector>
00004 #include <algorithm>
00005
00006 #include <sfe_toolkit/sfe_core.h>
00007 #include <sfe_toolkit/sfe_person.h>
00008
00009 #include "utils.hpp"
00010 #include "solvers.h"
00011 #include "annotate.hpp"
00012 #include "strings.hpp"
00013
00014 // Solver parameters, we are using ONNX Runtime CPU backend
00015 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{
00016     SFESolverParameter { "intra_threads", "4" },
00017 };
00018
00019 std::string image_path = "assets/person/person_detect_fisheye.jpg";
00020 std::string solver_detect = SOLVER_PERSON_DETECT;
00021
00022 int main() {
00023     SFEError error;
00024
00025     SFEImage image{};
00026     DEFER(sfeImageFree(image));
00027     { // STAGE 1: Load Image
00028         auto encoded_image = utils::readFile(image_path);
00029         error = sfeImageDecode(encoded_image.data(), encoded_image.size(), &image);
00030         utils::checkError(error);
00031
00032         std::cout « "Image size: " « image.width « "x" « image.height « std::endl;
00033     }
00034
00035     std::vector<SFEDetection> detections(10);
00036     { // STAGE 2: Run person detection
00037         // Initialize detection solver
00038         SFESolver detection_solver{};
00039         DEFER(sfeSolverFree(detection_solver));
00040         error = sfeSolverCreate(solver_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detection_solver);
00041         utils::checkError(error);
00042
00043         // Run detection for up to 10 persons
00044         size_t detection_count = detections.size();
00045         float detection_threshold = 0.3;
00046         error = sfeDetect(detection_solver,
00047             image,
00048             detection_threshold,
00049             detections.data(), &detection_count);
00050         utils::checkError(error);
00051         detections.resize(detection_count);
00052
00053         std::cout « "Detected " « detection_count « " persons" « std::endl;
00054     }
00055 }
```

9.27 D:/glab/1512fd1724a/1/example/example_tattoo_detect.cpp File Reference

```
#include <algorithm>
#include <iomanip>
#include <sstream>
#include <vector>
#include <sfe_toolkit/sfe_core.h>
#include <sfe_toolkit/sfe_tattoo.h>
#include ¨annotate.hpp¨
#include ¨solvers.h¨
#include ¨utils.hpp¨
#include ¨optional¨
```

Functions

- int main ()

Variables

- const auto SOLVER_PARAMETERS
- std::string image_path = ¨assets/tattoo/tattoo_0.png¨
- std::optional< std::string > solver = std::nullopt

9.27.1 Function Documentation

9.27.1.1 main()

```
int main ( )
```

[Tattoo Detect]
[Tattoo Detect]
Definition at line 26 of file example_tattoo_detect.cpp.

```
00026           {
00027   SFEError error;
00028
00029   SFEImage image{};
00030   DEFER(sfeImageFree(image));
00031   { // STAGE 1: Load Image
00032     auto encoded_image = utils::readFile(image_path);
00033     error = sfeImageDecode(encoded_image.data(), encoded_image.size(), &image);
00034     utils::checkError(error);
00035
00036     std::cout « "Image size: " « image.width « "x" « image.height
00037               « std::endl;
00038   }
00039
00040   std::vector<SFEDetection> detections(10);
00041   { // STAGE 2: Run person detection
00042     // Initialize detection solver
00043     SFESolver detection_solver{};
00044     DEFER(sfeSolverFree(detection_solver));
00046     error = sfeSolverCreate(
00047         solver.value_or("").c_str(), SOLVER_PARAMETERS.data(), SOLVER_PARAMETERS.size(),
00048         &detection_solver);
00049     utils::checkError(error);
00050
00051     // Run detection for up to 10 tattoos
00052     size_t detection_count = detections.size();
00053     float detection_threshold = 0.3;
00054     error = sfeDetect(detection_solver, image, detection_threshold,
00055                       detections.data(), &detection_count);
00057     utils::checkError(error);
00058     detections.resize(detection_count);
00059
00060     std::cout « "Detected " « detection_count « " tattoos" « std::endl;
00061   }
00062 }
```

9.27.2 Variable Documentation

9.27.2.1 image_path

```
std::string image_path = ¨assets/tattoo/tattoo_0.png¨
```

Definition at line 19 of file example_tattoo_detect.cpp.

9.27.2.2 solver

```
std::optional<std::string> solver = std::nullopt
```

Definition at line 23 of file example_tattoo_detect.cpp.

9.27.2.3 SOLVER_PARAMETERS

```
const auto SOLVER_PARAMETERS
```

Initial value:

```
= std::vector<SFESolverParameter>{
    SFESolverParameter{"intra_threads", "4"},
```

}
Definition at line 15 of file example_tattoo_detect.cpp.

9.28 example_tattoo_detect.cpp

Go to the documentation of this file.

```
00001 #include <algorithm>
00002 #include <iomanip>
00003 #include <sstream>
00004 #include <vector>
00005
00006 #include <sfe_toolkit/sfe_core.h>
00007 #include <sfe_toolkit/sfe_tattoo.h>
00008
00009 #include "annotate.hpp"
00010 #include "solvers.h"
00011 #include "utils.hpp"
00012 #include "optional"
00013
00014 // Solver parameters, we are using ONNX Runtime CPU backend
00015 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{
00016     SFESolverParameter{"intra_threads", "4"},
00017 };
00018
00019 std::string image_path = "assets/tattoo/tattoo_0.png";
00020 #ifdef SOLVER_TATTOO_DETECT
00021 std::optional<std::string> solver = SOLVER_TATTOO_DETECT;
00022 #else
00023 std::optional<std::string> solver = std::nullopt;
00024 #endif // SOLVER_TATTOO_DETECT
00025
00026 int main() {
00027   SFEError error;
00028
00029   SFEImage image{};
00030   DEFER(sfeImageFree(image));
00031   { // STAGE 1: Load Image
00032     auto encoded_image = utils::readFile(image_path);
00033     error = sfeImageDecode(encoded_image.data(), encoded_image.size(), &image);
00034     utils::checkError(error);
00035
00036     std::cout « "Image size: " « image.width « "x" « image.height
00037               « std::endl;
00038   }
00039
00040   std::vector<SFEDetection> detections(10);
00041   { // STAGE 2: Run person detection
00042     // Initialize detection solver
00043     SFESolver detection_solver{};
00044     DEFER(sfeSolverFree(detection_solver));
00046     error = sfeSolverCreate(
00047         solver.value_or("").c_str(), SOLVER_PARAMETERS.data(), SOLVER_PARAMETERS.size(),
00048         &detection_solver);
00049     utils::checkError(error);
00050
00051     // Run detection for up to 10 tattoos
00052     size_t detection_count = detections.size();
00053     float detection_threshold = 0.3;
00054     error = sfeDetect(detection_solver, image, detection_threshold,
00055                       detections.data(), &detection_count);
00057     utils::checkError(error);
00058     detections.resize(detection_count);
00059
00060     std::cout « "Detected " « detection_count « " tattoos" « std::endl;
00061   }
00062 }
```

9.29 D:/glab/1512fd1724a/1/include/sfe_core.h File Reference

File containing API of sfe_core library as part of SFE Toolkit.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include ¨sfe_core_detection.h¨
```

Data Structures

- struct SFEImage

  Raw owned raster image representation, HWC|BGR order.
- struct SFEImageInfo

  Image information structure, for describing image format and size.
- struct SFESolverParameter

  Solver parameter used to configure solvers.
- struct SFEEntity

  Entity type, used to group templates for entity identification. This type is compatible with uuid_v4 that can be used to generate unique ids.
- struct SFETemplateIdentificationCandidate

  Candidate for identification by template.
- struct SFEEntityIdentificationCandidate

  Candidate for identification by entity.
- struct SFETracked

  Tracked entity struct.

Typedefs

- typedef enum SFEHwidMethod SFEHwidMethod

  Supported methods for Hardware ID (HWID) computation.
- typedef enum SFELicenseType SFELicenseType

  License types.
- typedef enum SFEImageFormat SFEImageFormat

  Supported image formats.
- typedef struct SFEImage SFEImage

  Raw owned raster image representation, HWC|BGR order.
- typedef SFEImage SFEImageView

  Raw raster image view, HWC|BGR order.
- typedef struct SFEImageInfo SFEImageInfo

  Image information structure, for describing image format and size.
- typedef void * SFEError

  Error type used to hold optional error message.
- typedef void * SFESolver

  Solver provides an abstract interface over inference models and engines.
- typedef struct SFESolverParameter SFESolverParameter

  Solver parameter used to configure solvers.
- typedef struct SFEEntity SFEEntity

  Entity type, used to group templates for entity identification. This type is compatible with uuid_v4 that can be used to generate unique ids.
- typedef struct SFETemplateIdentificationCandidate SFETemplateIdentificationCandidate

  Candidate for identification by template.
- typedef struct SFEEntityIdentificationCandidate SFEEntityIdentificationCandidate

  Candidate for identification by entity.
- typedef void * SFETracker

  Tracker is a ByteTrack implementation for multi-modal tracking across multiple frames.
- typedef struct SFETracked SFETracked

  Tracked entity struct.

Enumerations

- enum SFEHwidMethod {
  SFE_HWID_METHOD_AUTO = 0 , SFE_HWID_METHOD_DISK_ID = 1 , SFE_HWID_METHOD_MAC = 2 ,
  SFE_HWID_METHOD_SERIAL_NUMBER = 3 ,
  SFE_HWID_METHOD_IMEI = 4 , SFE_HWID_METHOD_SM_BIOS = 5 , SFE_HWID_METHOD_AMAZON = 6 , SFE_HWID_METHOD_APP_ID = 7 ,
  SFE_HWID_METHOD_PHY = 8 , SFE_HWID_METHOD_MAC_ADDRESS = 9 , SFE_HWID_METHOD_SYS = 10 , SFE_HWID_METHOD_ANDROID_ID = 11 }

  Supported methods for Hardware ID (HWID) computation.
- enum SFELicenseType {
  SFE_LICENSE_TYPE_NONE = 0 , SFE_LICENSE_TYPE_FILE = 1 , SFE_LICENSE_TYPE_MEMORY = 2 , SFE_LICENSE_TYPE_TOKEN = 3 ,
  SFE_LICENSE_TYPE_ENV = 4 }

  License types.
- enum SFEImageFormat {
  SFE_IMAGE_FORMAT_PNG = 0 , SFE_IMAGE_FORMAT_JPEG = 1 , SFE_IMAGE_FORMAT_GIF = 2 ,
  SFE_IMAGE_FORMAT_TGA = 3 ,
  SFE_IMAGE_FORMAT_BMP = 4 , SFE_IMAGE_FORMAT_QOI = 5 }

  Supported image formats.
- enum SFETrackedState { SFE_TRACKED_STATE_NEW = 0 , SFE_TRACKED_STATE_TRACKED = 1 , SFE_TRACKED_STATE_LOST = 2 , SFE_TRACKED_STATE_REMOVED = 3 }

  Tracked state.

Functions

- const char *const sfeVersion ()

  Get version of the toolkit library.
- void sfeErrorFree (SFEError error)

  Free memory associated with `SFEError`.
- const char * sfeErrorMessage (SFEError error)

  Get error message.
- SFEError sfeHwidGet (char *out_hwid, size_t *in_out_hwid_len, SFEHwidMethod method)

  Get Hardware ID (HWID) of the current machine.
- SFEError sfeLicenseValidate (void)

  Validate a license for this product. This process conducts checks to confirm the license's validity for this product and the current machine. Upon successful validation, the product is ready for use.
- SFEError sfeLicenseType (SFELicenseType *out_license_type)

  Get license type. This function returns the type of license that is currently active without performing full license validation.
- void sfeLicenseSet (const unsigned char *data, const size_t data_len)

  Set the license data via environment variable. The environment variable that will be set: ILICENSE_DATA. The provided data will be converted into a base64 string internally since the ILICENSE_DATA environment variable expects data in this format.
- SFEError sfeSolverCreate (const char *solver_file, const SFESolverParameter *solver_parameters, size_t solver_parameters_count, SFESolver *out_solver)

  Create new solver from solver file.
- void sfeSolverFree (SFESolver solver)

  Free memory associated with `SFESolver`.
- SFEError sfeImageDecode (const unsigned char *data, size_t data_len, SFEImage *out_image)

  Decode `SFEImage` from raw image data of various formats. Eg. PNG, JPEG ..
- SFEError sfeImageEncode (SFEImageView image, SFEImageFormat image_format, unsigned char *out_↩ data, size_t *in_out_data_len)

  Encode `SFEImage` into a buffer with specified image_format.
- SFEError sfeImageInfo (const unsigned char *data, size_t data_len, SFEImageInfo *out_image_info)

Try to get information from image data without fully decoding it.

- void sfeImageFree (SFEImage image)

  Free memory associated with `SFEImage`.

- SFEError sfeImageColorTranspose (SFEImageView image, SFEImage *out_image)

  Transpose RGB<->BGR color channel order.

- SFEError sfeImageResize (SFEImageView image, size_t width, size_t height, SFEImage *out_image)

  Resize image.

- SFEError sfeImageResizeWithAspectRatio (SFEImageView image, size_t width, size_t height, SFEImage *out_image)

  Resize image with maintaining aspect ratio. Whole resized image will be fitted and centered inside output image with black edges.

- SFEError sfeDetect (SFESolver solver, SFEImageView image, float threshold, SFEDetection *out_↩ detections, size_t *in_out_detection_count)

  Detect objects in the source image using unified detection API.

- SFEError sfeDetectionTrackerCreate (float new_track_threshold, float track_high_threshold, float track_low↩ _threshold, float match_threshold, uint64_t max_time_lost, SFETracker *out_tracker)

  Create a new tracker.

- void sfeDetectionTrackerFree (SFETracker tracker)

  Free the tracker.

- SFEError sfeDetectionTrackerUpdate (SFETracker tracker, const SFEDetection *detections, size_↩ t detections_count, SFETracked *out_tracked, size_t *in_out_tracked_count)

  Update the tracker.

- SFEError sfeDetectionTrackerLost (SFETracker tracker, SFEEntity *out_entities, size_t *in_out_entities_↩ count)

  Get lost entities after update.

- SFEError sfeDetectionTrackerRemoved (SFETracker tracker, SFEEntity *out_entities, size_t *in_out_↩ entities_count)

  Get removed entities after update.

9.29.1 Detailed Description

File containing API of sfe_core library as part of SFE Toolkit.

Author

Innovatrics - SmartFace Embedded

Date

16.5.2023

Definition in file sfe_core.h.

9.29.2 Typedef Documentation

9.29.2.1 SFEEntity

```
typedef struct SFEEntity SFEEntity
```

Entity type, used to group templates for entity identification. This type is compatible with uuid_v4 that can be used to generate unique ids.

9.29.2.2 SFEEntityIdentificationCandidate

```
typedef struct SFEEntityIdentificationCandidate SFEEntityIdentificationCandidate
```

Candidate for identification by entity.

9.29.2.3 SFEError

`typedef void* SFEError`
Error type used to hold optional error message.

Note

In case of no error NULL is returned.

Warning

Use `sfeErrorFree` to release memory associated with returned errors.

Definition at line 97 of file sfe_core.h.

9.29.2.4 SFEHwidMethod

`typedef enum SFEHwidMethod SFEHwidMethod`
Supported methods for Hardware ID (HWID) computation.

9.29.2.5 SFEImage

`typedef struct SFEImage SFEImage`
Raw owned raster image representation, HWC|BGR order.

Note

For non-owning variant use `SFEImageView`.

Warning

This type retains ownership of the data, call `sfeImageFree` to free the allocated image data.

9.29.2.6 SFEImageFormat

`typedef enum SFEImageFormat SFEImageFormat`
Supported image formats.

9.29.2.7 SFEImageInfo

`typedef struct SFEImageInfo SFEImageInfo`
Image information structure, for describing image format and size.

9.29.2.8 SFEImageView

`typedef SFEImage SFEImageView`
Raw raster image view, HWC|BGR order.

Note

For owning variant use `SFEImage`.

Definition at line 81 of file sfe_core.h.

9.29.2.9 SFELicenseType

`typedef enum SFELicenseType SFELicenseType`
License types.

9.29.2.10 SFESolver

`typedef void* SFESolver`
Solver provides an abstract interface over inference models and engines.

Warning

Use `sfeSolverFree` to release memory associated with the solver.

Definition at line 102 of file sfe_core.h.

9.29.2.11 SFESolverParameter

`typedef struct SFESolverParameter SFESolverParameter`
Solver parameter used to configure solvers.

9.29.2.12 SFETemplateIdentificationCandidate

`typedef struct SFETemplateIdentificationCandidate SFETemplateIdentificationCandidate`
Candidate for identification by template.

9.29.2.13 SFETracked

`typedef struct SFETracked SFETracked`
Tracked entity struct.

9.29.2.14 SFETracker

`typedef void* SFETracker`
Tracker is a ByteTrack implementation for multi-modal tracking across multiple frames.

Warning

Use `sfeDetectionTrackerFree` to release memory associated with the tracker.

Definition at line 286 of file sfe_core.h.

9.29.3 Enumeration Type Documentation

9.29.3.1 SFEHwidMethod

`enum SFEHwidMethod`
Supported methods for Hardware ID (HWID) computation.

Enumerator

SFE_HWID_METHOD_AUTO	
SFE_HWID_METHOD_DISK_ID	
SFE_HWID_METHOD_MAC	
SFE_HWID_METHOD_SERIAL_NUMBER	
SFE_HWID_METHOD_IMEI	
SFE_HWID_METHOD_SM_BIOS	
SFE_HWID_METHOD_AMAZON	
SFE_HWID_METHOD_APP_ID	
SFE_HWID_METHOD_PHY	
SFE_HWID_METHOD_MAC_ADDRESS	
SFE_HWID_METHOD_SYS	
SFE_HWID_METHOD_ANDROID_ID	

Definition at line 21 of file sfe_core.h.

```
00021                           {
00022   SFE_HWID_METHOD_AUTO = 0,
00023   SFE_HWID_METHOD_DISK_ID = 1,
00024   SFE_HWID_METHOD_MAC = 2,
00025   SFE_HWID_METHOD_SERIAL_NUMBER = 3,
00026   SFE_HWID_METHOD_IMEI = 4,
00027   SFE_HWID_METHOD_SM_BIOS = 5,
00028   SFE_HWID_METHOD_AMAZON = 6,
00029   SFE_HWID_METHOD_APP_ID = 7,
00030   SFE_HWID_METHOD_PHY = 8,
00031   SFE_HWID_METHOD_MAC_ADDRESS = 9,
00032   SFE_HWID_METHOD_SYS = 10,
00033   SFE_HWID_METHOD_ANDROID_ID = 11,
00034 } SFEHwidMethod;
```

9.29.3.2 SFEImageFormat

`enum SFEImageFormat`

Supported image formats.

Enumerator

SFE_IMAGE_FORMAT_PNG	Portable Network Graphics format.
SFE_IMAGE_FORMAT_JPEG	Joint Photographic Experts Group format.
SFE_IMAGE_FORMAT_GIF	Graphics Interchange Format.
SFE_IMAGE_FORMAT_TGA	Truevision TGA format.
SFE_IMAGE_FORMAT_BMP	Windows Bitmap format.
SFE_IMAGE_FORMAT_QOI	Quite OK Image format.

Definition at line 51 of file sfe_core.h.

```
00051                           {
00053   SFE_IMAGE_FORMAT_PNG = 0,
00055   SFE_IMAGE_FORMAT_JPEG = 1,
00057   SFE_IMAGE_FORMAT_GIF = 2,
00059   SFE_IMAGE_FORMAT_TGA = 3,
00061   SFE_IMAGE_FORMAT_BMP = 4,
00063   SFE_IMAGE_FORMAT_QOI = 5,
00064 } SFEImageFormat;
```

9.29.3.3 SFELicenseType

`enum SFELicenseType`

License types.

Enumerator

SFE_LICENSE_TYPE_NONE	License was not read.
SFE_LICENSE_TYPE_FILE	License was read from a file.
SFE_LICENSE_TYPE_MEMORY	License was read from memory.
SFE_LICENSE_TYPE_TOKEN	License was read from a USB token.
SFE_LICENSE_TYPE_ENV	License was read from an environment variable.

Definition at line 37 of file sfe_core.h.

```
00037                           {
00039   SFE_LICENSE_TYPE_NONE = 0,
00041   SFE_LICENSE_TYPE_FILE = 1,
00043   SFE_LICENSE_TYPE_MEMORY = 2,
00045   SFE_LICENSE_TYPE_TOKEN = 3,
00047   SFE_LICENSE_TYPE_ENV = 4,
00048 } SFELicenseType;
```

9.29.3.4 SFETrackedState

`enum SFETrackedState`

Tracked state.

Enumerator

SFE_TRACKED_STATE_NEW	
SFE_TRACKED_STATE_TRACKED	
SFE_TRACKED_STATE_LOST	
SFE_TRACKED_STATE_REMOVED	

Definition at line 289 of file sfe_core.h.

```
00289                          {
00290   SFE_TRACKED_STATE_NEW = 0,
00291   SFE_TRACKED_STATE_TRACKED = 1,
00292   SFE_TRACKED_STATE_LOST = 2,
00293   SFE_TRACKED_STATE_REMOVED = 3,
00294 };
```

9.29.4 Function Documentation

9.29.4.1 sfeDetect()

```
SFEError sfeDetect (
            SFESolver solver,
            SFEImageView image,
            float threshold,
            SFEDetection * out_detections,
            size_t * in_out_detection_count )
```

Detect objects in the source image using unified detection API.

Parameters

in	*solver*	Detection solver (can be face, iris, palm, object, tattoo, or person solver)
in	*image*	Source image
in	*threshold*	Detection confidence threshold - range <0,1>
out	*out_detections*	Pointer to an array of detected objects
in,out	*in_out_detection_count*	IN: maximum number of detections to return OUT: real number of detected objects

Returns

Error in case the detection fails.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

9.29.4.2 sfeDetectionTrackerCreate()

```
SFEError sfeDetectionTrackerCreate (
            float new_track_threshold,
            float track_high_threshold,
            float track_low_threshold,
            float match_threshold,
            uint64_t max_time_lost,
            SFETracker * out_tracker )
```

Create a new tracker.

Parameters

in	*new_track_threshold*	Minimum confidence score required to initialize a new track
in	*track_high_threshold*	High threshold for tracking
in	*track_low_threshold*	Low threshold for tracking
in	*match_threshold*	Match threshold
in	*max_time_lost*	Maximum time lost
out	*out_tracker*	OUT: pointer to the created tracker

Examples

example_face_track.cpp.

9.29.4.3 sfeDetectionTrackerFree()

```
void sfeDetectionTrackerFree (
            SFETracker tracker )
```

Free the tracker.

Parameters

in	*tracker*	Tracker to free

Examples

example_face_track.cpp.

9.29.4.4 sfeDetectionTrackerLost()

```
SFEError sfeDetectionTrackerLost (
            SFETracker tracker,
            SFEEntity * out_entities,
            size_t * in_out_entities_count )
```

Get lost entities after update.

Parameters

in	*tracker*	Tracker to get lost entities from
out	*out_entities*	OUT: pointer to the array of lost entities
in,out	*in_out_entities_count*	IN: Available space in out_entities OUT: number of lost entities

Returns

Error in case of failed lost entities retrieval

Examples

example_face_track.cpp.

9.29.4.5 sfeDetectionTrackerRemoved()

```
SFEError sfeDetectionTrackerRemoved (
            SFETracker tracker,
            SFEEntity * out_entities,
            size_t * in_out_entities_count )
```

Get removed entities after update.

Parameters

in	*tracker*	Tracker to get removed entities from
out	*out_entities*	OUT: pointer to the array of removed entities
in,out	*in_out_entities_count*	IN: Available space in out_entities OUT: number of removed entities

Returns

Error in case of failed removed entities retrieval

Examples

example_face_track.cpp.

9.29.4.6 sfeDetectionTrackerUpdate()

```
SFEError sfeDetectionTrackerUpdate (
            SFETracker tracker,
            const SFEDetection * detections,
            size_t detections_count,
            SFETracked * out_tracked,
            size_t * in_out_tracked_count )
```

Update the tracker.

Parameters

in	*tracker*	Tracker to update
in	*detections*	Detections to update the tracker with (can be NULL if detections_count is 0)
in	*detections_count*	Number of detections
out	*out_tracked*	OUT: pointer to the array of tracked entities (can be NULL if only count is needed)
in,out	*in_out_tracked_count*	IN: Available space in out_tracked (if out_tracked is not NULL) OUT: Number of tracked entities. Can be NULL if out_tracked is NULL.

Returns

Error in case of failed update

Examples

example_face_track.cpp.

9.29.4.7 sfeErrorFree()

```
void sfeErrorFree (
            SFEError error )
```

Free memory associated with SFEError.

Parameters

in	*error*	Error to free.

9.29.4.8 sfeErrorMessage()

```
const char * sfeErrorMessage (
```

```
            SFEError error )
```

Get error message.

Parameters

in	*error*	Error to get message from.

Returns

NULL terminated C string containing the error message.

Warning

The returned C string is only valid as long as `sfeErrorFree` was not called for the given error.

9.29.4.9 sfeHwidGet()

```
SFEError sfeHwidGet (
            char * out_hwid,
            size_t * in_out_hwid_len,
            SFEHwidMethod method )
```

Get Hardware ID (HWID) of the current machine.

Note

Some methods for HWID computation are not supported on some platforms. If unsure, use `SFE_HWID_METHOD_AUTO` that will choose a default method for the current platform.

Parameters

out	*out_hwid*	Pointer to a buffer where the HWID will be stored. Providing a null pointer will determine the necessary buffer size.
in,out	*in_out_hwid_len*	IN: Total size of the out_hwid buffer. OUT: Actual size of the data written to the out_hwid buffer.
in	*method*	A method to use for HWID computation.

Returns

Error in case the HWID computation fails or an invalid parameter is provided (e.g. buffer length is smaller than the size of the HWID).

9.29.4.10 sfeImageColorTranspose()

```
SFEError sfeImageColorTranspose (
            SFEImageView image,
            SFEImage * out_image )
```

Transpose RGB<->BGR color channel order.

Parameters

in	*image*	Image to color transpose.
out	*out_image*	Output image with transposed color channels.

Returns

Error

9.29.4.11 sfeImageDecode()

```
SFEError sfeImageDecode (
            const unsigned char * data,
            size_t data_len,
            SFEImage * out_image )
```

Decode SFEImage from raw image data of various formats. Eg. PNG, JPEG ..

Parameters

in	*data*	Image data to decode image from.
in	*data_len*	Size of the image data in bytes.
out	*out_image*	Raw raster image.

Returns

Error in case of unknown image type.

Warning

The returned SFEImage retains ownership of the image data, call sfeImageFree to release the associated memory.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

9.29.4.12 sfeImageEncode()

```
SFEError sfeImageEncode (
            SFEImageView image,
            SFEImageFormat image_format,
            unsigned char * out_data,
            size_t * in_out_data_len )
```

Encode SFEImage into a buffer with specified image_format.

Parameters

in	*image*	Image to encode.
in	*image_format*	Image format to encode to.
out	*out_data*	Image data to encode image into.
in,out	*in_out_data_len*	IN: Total size of the out_data buffer. OUT: Actual size of the data written to the buffer.

Returns

Error in image format conversion or in case the buffer size is too low.

Examples

example_face_identify.cpp, example_face_liveness.cpp, and example_palm_liveness.cpp.

9.29.4.13 sfeImageFree()

```
void sfeImageFree (
            SFEImage image )
```

Free memory associated with SFEImage.

Parameters

in	*image*	Image to free.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

9.29.4.14 sfeImageInfo()

```
SFEError sfeImageInfo (
            const unsigned char * data,
            size_t data_len,
            SFEImageInfo * out_image_info )
```

Try to get information from image data without fully decoding it.

Parameters

in	*data*	Image data to get information from.
in	*data_len*	Size of the image data in bytes.
out	*out_image_info*	Image information structure.

Returns

Error in case of unknown image type.

9.29.4.15 sfeImageResize()

```
SFEError sfeImageResize (
            SFEImageView image,
            size_t width,
            size_t height,
            SFEImage * out_image )
```

Resize image.

Parameters

in	*image*	Source image view to resize.
in	*width*	Target resize width.
in	*height*	Target resize height.
out	*out_image*	Resized image.

Returns

Error

Examples

example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

9.29.4.16 sfeImageResizeWithAspectRatio()

```
SFEError sfeImageResizeWithAspectRatio (
            SFEImageView image,
            size_t width,
            size_t height,
            SFEImage * out_image )
```

Resize image with maintaining aspect ratio. Whole resized image will be fitted and centered inside output image with black edges.

Parameters

in	*image*	Source image view to resize.
in	*width*	Target resize width.
in	*height*	Target resize height.
out	*out_image*	Resized image.

Returns

Error

9.29.4.17 sfeLicenseSet()

```
void sfeLicenseSet (
            const unsigned char * data,
            const size_t data_len )
```

Set the license data via environment variable. The environment variable that will be set: ILICENSE_DATA. The provided data will be converted into a base64 string internally since the ILICENSE_DATA environment variable expects data in this format.

Parameters

in	*data*	License data.
in	*data_len*	Length of the license data.

9.29.4.18 sfeLicenseType()

```
SFEError sfeLicenseType (
            SFELicenseType * out_license_type )
```

Get license type. This function returns the type of license that is currently active without performing full license validation.

Parameters

out	*out_license_type*	Pointer to a `SFELicenseType` to receive the license type.

Returns

Error in case the license type is not available.

9.29.4.19 sfeLicenseValidate()

```
SFEError sfeLicenseValidate (
            void  )
```

Validate a license for this product. This process conducts checks to confirm the license's validity for this product and the current machine. Upon successful validation, the product is ready for use.

Returns

Upon failure, the returned error will contain a descriptive message for the reason behind the failure.

Note

Use `sfeLicenseType()` to get the license type after validation.

9.29.4.20 sfeSolverCreate()

```
SFEError sfeSolverCreate (
            const char * solver_file,
            const SFESolverParameter * solver_parameters,
            size_t solver_parameters_count,
            SFESolver * out_solver )
```

Create new solver from solver file.

Parameters

in	*solver_file*	Solver file to create the solver from.
in	*solver_parameters*	Pointer to collection of solver parameters.
in	*solver_parameters_count*	Number of solver parameters.
out	*out_solver*	New solver.

Returns

Error in case the solver fails to load.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

9.29.4.21 sfeSolverFree()

```
void sfeSolverFree (
            SFESolver solver )
```

Free memory associated with `SFESolver`.

Parameters

in	*solver*	Solver to free.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, example_palm_attributes.cpp, example_palm_identify.cpp, and example_palm_liveness.cpp.

9.29.4.22 sfeVersion()

```
const char *const sfeVersion ( )
```

Get version of the toolkit library.

Note

The returned pointer is statically allocated, don't free it.

Returns

NULL terminated C string containing the version.

9.30 sfe_core.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include <stdbool.h>
00011 #include <stddef.h>
00012 #include <stdint.h>
00013
00014 #include "sfe_core_detection.h"
00015
00016 #ifdef __cplusplus
00017 extern "C" {
00018 #endif
00019
00021 typedef enum SFEHwidMethod {
00022   SFE_HWID_METHOD_AUTO = 0,
00023   SFE_HWID_METHOD_DISK_ID = 1,
00024   SFE_HWID_METHOD_MAC = 2,
00025   SFE_HWID_METHOD_SERIAL_NUMBER = 3,
00026   SFE_HWID_METHOD_IMEI = 4,
00027   SFE_HWID_METHOD_SM_BIOS = 5,
00028   SFE_HWID_METHOD_AMAZON = 6,
00029   SFE_HWID_METHOD_APP_ID = 7,
00030   SFE_HWID_METHOD_PHY = 8,
00031   SFE_HWID_METHOD_MAC_ADDRESS = 9,
00032   SFE_HWID_METHOD_SYS = 10,
00033   SFE_HWID_METHOD_ANDROID_ID = 11,
00034 } SFEHwidMethod;
00035
00037 typedef enum SFELicenseType {
00039   SFE_LICENSE_TYPE_NONE = 0,
00041   SFE_LICENSE_TYPE_FILE = 1,
00043   SFE_LICENSE_TYPE_MEMORY = 2,
00045   SFE_LICENSE_TYPE_TOKEN = 3,
00047   SFE_LICENSE_TYPE_ENV = 4,
00048 } SFELicenseType;
00049
00051 typedef enum SFEImageFormat {
00053   SFE_IMAGE_FORMAT_PNG = 0,
00055   SFE_IMAGE_FORMAT_JPEG = 1,
00057   SFE_IMAGE_FORMAT_GIF = 2,
00059   SFE_IMAGE_FORMAT_TGA = 3,
00061   SFE_IMAGE_FORMAT_BMP = 4,
00063   SFE_IMAGE_FORMAT_QOI = 5,
00064 } SFEImageFormat;
00065
00070 typedef struct SFEImage {
00072   size_t width;
00074   size_t height;
00076   unsigned char *data;
00077 } SFEImage;
00078
00081 typedef SFEImage SFEImageView;
00082
00084 typedef struct SFEImageInfo {
00086   size_t width;
00088   size_t height;
00090   SFEImageFormat format;
00091 } SFEImageInfo;
00092
00097 typedef void *SFEError;
00098
00102 typedef void *SFESolver;
00103
00105 typedef struct SFESolverParameter {
00107   const char *name;
00109   const char *value;
00110 } SFESolverParameter;
00111
00114 typedef struct SFEEntity {
00115   uint8_t uuid[16];
00116 } SFEEntity;
00117
00119 typedef struct SFETemplateIdentificationCandidate {
00121   float score;
00123   size_t index;
00124 } SFETemplateIdentificationCandidate;
00125
```

```
00127 typedef struct SFEEntityIdentificationCandidate {
00129   float score;
00131   size_t index;
00133   SFEEntity entity;
00134 } SFEEntityIdentificationCandidate;
00135
00139 const char *const sfeVersion();
00140
00143 void sfeErrorFree(SFEError error);
00144
00150 const char *sfeErrorMessage(SFEError error);
00151
00163 SFEError sfeHwidGet(char *out_hwid, size_t *in_out_hwid_len,
00164                     SFEHwidMethod method);
00165
00173 SFEError sfeLicenseValidate(void);
00174
00181 SFEError sfeLicenseType(SFELicenseType *out_license_type);
00182
00189 void sfeLicenseSet(const unsigned char *data, const size_t data_len);
00190
00197 SFEError sfeSolverCreate(const char *solver_file,
00198                          const SFESolverParameter *solver_parameters,
00199                          size_t solver_parameters_count, SFESolver *out_solver);
00200
00203 void sfeSolverFree(SFESolver solver);
00204
00214 SFEError sfeImageDecode(const unsigned char *data, size_t data_len,
00215                         SFEImage *out_image);
00216
00225 SFEError sfeImageEncode(SFEImageView image, SFEImageFormat image_format,
00226                         unsigned char *out_data, size_t *in_out_data_len);
00227
00233 SFEError sfeImageInfo(const unsigned char *data, size_t data_len,
00234                       SFEImageInfo *out_image_info);
00235
00238 void sfeImageFree(SFEImage image);
00239
00244 SFEError sfeImageColorTranspose(SFEImageView image, SFEImage *out_image);
00245
00252 SFEError sfeImageResize(SFEImageView image, size_t width, size_t height,
00253                         SFEImage *out_image);
00254
00262 SFEError sfeImageResizeWithAspectRatio(SFEImageView image, size_t width,
00263                                        size_t height, SFEImage *out_image);
00264
00265 // CORE DETECTION
00266
00276 SFEError sfeDetect(SFESolver solver, SFEImageView image, float threshold,
00277                    SFEDetection *out_detections,
00278                    size_t *in_out_detection_count);
00279
00280 // CORE TRACKING
00281
00286 typedef void *SFETracker;
00287
00289 enum SFETrackedState {
00290   SFE_TRACKED_STATE_NEW = 0,
00291   SFE_TRACKED_STATE_TRACKED = 1,
00292   SFE_TRACKED_STATE_LOST = 2,
00293   SFE_TRACKED_STATE_REMOVED = 3,
00294 };
00295
00297 typedef struct SFETracked {
00299   SFEDetection detection;
00301   uint64_t id;
00303   SFEEntity uuid;
00305   enum SFETrackedState state;
00306 } SFETracked;
00307
00316 SFEError
00317 sfeDetectionTrackerCreate(float new_track_threshold, float track_high_threshold,
00318                           float track_low_threshold, float match_threshold,
00319                           uint64_t max_time_lost, SFETracker *out_tracker);
00320
00323 void sfeDetectionTrackerFree(SFETracker tracker);
00324
00336 SFEError sfeDetectionTrackerUpdate(SFETracker tracker,
00337                                    const SFEDetection *detections,
00338                                    size_t detections_count,
00339                                    SFETracked *out_tracked,
00340                                    size_t *in_out_tracked_count);
00341
00348 SFEError sfeDetectionTrackerLost(SFETracker tracker, SFEEntity *out_entities,
00349                                  size_t *in_out_entities_count);
00350
00357 SFEError sfeDetectionTrackerRemoved(SFETracker tracker, SFEEntity *out_entities,
```

```
00358                                     size_t *in_out_entities_count);
00359
00360 #ifdef __cplusplus
00361 } // extern "C"
00362 #endif
```

9.31 D:/glab/1512fd1724a/1/include/sfe_core_detection.h File Reference

File containing core detection type definitions.

```
#include <stdbool.h>
#include <stddef.h>
```

Data Structures

- struct SFEBoundingBox

  Bounding box.
- struct SFEOrientedBoundingBox

  Oriented bounding box.
- struct SFEFaceDetectionData
- struct SFEIrisKeypoint

  Iris keypoint struct.
- struct SFEIrisDetectionData

  Iris detection data struct (type-specific fields only)
- struct SFEPalmKeypoint

  Palm keypoint struct.
- struct SFEPalmLandmarks

  Palm landmarks (keypoints, hand position, orientation, confidence)
- struct SFEPalmDetectionData

  Palm detection data struct (type-specific fields only)
- struct SFEObject

  Object struct.
- struct SFEObjectDetectionData

  Object detection data struct (type-specific fields only)
- struct SFETattooDetectionData

  Tattoo detection data struct (type-specific fields only)
- struct SFEPersonDetectionData

  Person detection data struct (type-specific fields only)
- struct SFEVisualCodeKeypoint

  Visual code keypoint struct.
- struct SFEVisualCodeDetectionData

  Visual code detection data struct (type-specific fields only)
- union SFEModalityData

  Union of detection data types (type-specific fields only)
- struct SFEDetection

  Core detection - tagged union containing all detection types.

Macros

- #define SFE_IRIS_KEYPOINT_COUNT 8

  Iris keypoints count.
- #define SFE_PALM_KEYPOINT_COUNT 8

  Palm keypoints count.
- #define SFE_OBJECT_DETECT 80

Object detection count.

- #define SFE_VISUAL_CODE_KEYPOINT_COUNT 4

  Visual code keypoints count.

Typedefs

- typedef struct SFEBoundingBox SFEBoundingBox

  Bounding box.
- typedef struct SFEOrientedBoundingBox SFEOrientedBoundingBox

  Oriented bounding box.
- typedef struct SFEFaceDetectionData SFEFaceDetectionData
- typedef enum SFEIrisKeypointType SFEIrisKeypointType

  Iris keypoint types.
- typedef struct SFEIrisKeypoint SFEIrisKeypoint

  Iris keypoint struct.
- typedef struct SFEIrisDetectionData SFEIrisDetectionData

  Iris detection data struct (type-specific fields only)
- typedef struct SFEPalmKeypoint SFEPalmKeypoint

  Palm keypoint struct.
- typedef enum SFEHandPosition SFEHandPosition

  Palm hand enum.
- typedef enum SFEHandOrientation SFEHandOrientation

  Palm orientation enum.
- typedef struct SFEPalmLandmarks SFEPalmLandmarks

  Palm landmarks (keypoints, hand position, orientation, confidence)
- typedef struct SFEPalmDetectionData SFEPalmDetectionData

  Palm detection data struct (type-specific fields only)
- typedef enum SFEObjectType SFEObjectType

  Object type enumeration.
- typedef struct SFEObject SFEObject

  Object struct.
- typedef struct SFEObjectDetectionData SFEObjectDetectionData

  Object detection data struct (type-specific fields only)
- typedef struct SFETattooDetectionData SFETattooDetectionData

  Tattoo detection data struct (type-specific fields only)
- typedef struct SFEPersonDetectionData SFEPersonDetectionData

  Person detection data struct (type-specific fields only)
- typedef enum SFEVisualCodeCategory SFEVisualCodeCategory

  Visual code category enumeration.
- typedef struct SFEVisualCodeKeypoint SFEVisualCodeKeypoint

  Visual code keypoint struct.
- typedef struct SFEVisualCodeDetectionData SFEVisualCodeDetectionData

  Visual code detection data struct (type-specific fields only)
- typedef enum SFEModality SFEModality

  Detection modality enumeration.
- typedef union SFEModalityData SFEModalityData

  Union of detection data types (type-specific fields only)
- typedef struct SFEDetection SFEDetection

  Core detection - tagged union containing all detection types.

Enumerations

- enum SFEIrisKeypointType {
  SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT = 0 , SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT = 1 ,
  SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT = 2 , SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT
  = 3 ,
  SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT = 4 , SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT
  = 5 , SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT = 6 , SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT = 7
  }

  Iris keypoint types.
- enum SFEHandPosition { SFE_HAND_UNKNOWN = 0 , SFE_HAND_LEFT = 1 , SFE_HAND_RIGHT = 2 }

  Palm hand enum.
- enum SFEHandOrientation { SFE_ORIENTATION_UNKNOWN = 0 , SFE_ORIENTATION_PALMAR = 1 ,
  SFE_ORIENTATION_DORSAL = 2 }

  Palm orientation enum.
- enum SFEObjectType {
  SFE_OBJECT_TYPE_PERSON = 0 , SFE_OBJECT_TYPE_BICYCLE = 1 , SFE_OBJECT_TYPE_CAR = 2
  , SFE_OBJECT_TYPE_MOTORCYCLE = 3 ,
  SFE_OBJECT_TYPE_AIRPLANE = 4 , SFE_OBJECT_TYPE_BUS = 5 , SFE_OBJECT_TYPE_TRAIN = 6 ,
  SFE_OBJECT_TYPE_TRUCK = 7 ,
  SFE_OBJECT_TYPE_BOAT = 8 , SFE_OBJECT_TYPE_TRAFFICLIGHT = 9 , SFE_OBJECT_TYPE_FIREHYDRANT
  = 10 , SFE_OBJECT_TYPE_STOPSIGN = 11 ,
  SFE_OBJECT_TYPE_PARKINGMETER = 12 , SFE_OBJECT_TYPE_BENCH = 13 , SFE_OBJECT_TYPE_BIRD
  = 14 , SFE_OBJECT_TYPE_CAT = 15 ,
  SFE_OBJECT_TYPE_DOG = 16 , SFE_OBJECT_TYPE_HORSE = 17 , SFE_OBJECT_TYPE_SHEEP =
  18 , SFE_OBJECT_TYPE_COW = 19 ,
  SFE_OBJECT_TYPE_ELEPHANT = 20 , SFE_OBJECT_TYPE_BEAR = 21 , SFE_OBJECT_TYPE_ZEBRA
  = 22 , SFE_OBJECT_TYPE_GIRAFFE = 23 ,
  SFE_OBJECT_TYPE_BACKPACK = 24 , SFE_OBJECT_TYPE_UMBRELLA = 25 , SFE_OBJECT_TYPE_HANDBAG
  = 26 , SFE_OBJECT_TYPE_TIE = 27 ,
  SFE_OBJECT_TYPE_SUITCASE = 28 , SFE_OBJECT_TYPE_FRISBEE = 29 , SFE_OBJECT_TYPE_SKIS
  = 30 , SFE_OBJECT_TYPE_SNOWBOARD = 31 ,
  SFE_OBJECT_TYPE_SPORTSBALL = 32 , SFE_OBJECT_TYPE_KITE = 33 , SFE_OBJECT_TYPE_BASEBALLBAT
  = 34 , SFE_OBJECT_TYPE_BASEBALLGLOVE = 35 ,
  SFE_OBJECT_TYPE_SKATEBOARD = 36 , SFE_OBJECT_TYPE_SURFBOARD = 37 , SFE_OBJECT_TYPE_TENNISRACKE
  = 38 , SFE_OBJECT_TYPE_BOTTLE = 39 ,
  SFE_OBJECT_TYPE_WINEGLASS = 40 , SFE_OBJECT_TYPE_CUP = 41 , SFE_OBJECT_TYPE_FORK
  = 42 , SFE_OBJECT_TYPE_KNIFE = 43 ,
  SFE_OBJECT_TYPE_SPOON = 44 , SFE_OBJECT_TYPE_BOWL = 45 , SFE_OBJECT_TYPE_BANANA
  = 46 , SFE_OBJECT_TYPE_APPLE = 47 ,
  SFE_OBJECT_TYPE_SANDWICH = 48 , SFE_OBJECT_TYPE_ORANGE = 49 , SFE_OBJECT_TYPE_BROCCOLI
  = 50 , SFE_OBJECT_TYPE_CARROT = 51 ,
  SFE_OBJECT_TYPE_HOTDOG = 52 , SFE_OBJECT_TYPE_PIZZA = 53 , SFE_OBJECT_TYPE_DONUT
  = 54 , SFE_OBJECT_TYPE_CAKE = 55 ,
  SFE_OBJECT_TYPE_CHAIR = 56 , SFE_OBJECT_TYPE_COUCH = 57 , SFE_OBJECT_TYPE_POTTEDPLANT
  = 58 , SFE_OBJECT_TYPE_BED = 59 ,
  SFE_OBJECT_TYPE_DININGTABLE = 60 , SFE_OBJECT_TYPE_TOILET = 61 , SFE_OBJECT_TYPE_TV
  = 62 , SFE_OBJECT_TYPE_LAPTOP = 63 ,
  SFE_OBJECT_TYPE_MOUSE = 64 , SFE_OBJECT_TYPE_REMOTE = 65 , SFE_OBJECT_TYPE_KEYBOARD
  = 66 , SFE_OBJECT_TYPE_CELLPHONE = 67 ,
  SFE_OBJECT_TYPE_MICROWAVE = 68 , SFE_OBJECT_TYPE_OVEN = 69 , SFE_OBJECT_TYPE_TOASTER
  = 70 , SFE_OBJECT_TYPE_SINK = 71 ,
  SFE_OBJECT_TYPE_REFRIGERATOR = 72 , SFE_OBJECT_TYPE_BOOK = 73 , SFE_OBJECT_TYPE_CLOCK
  = 74 , SFE_OBJECT_TYPE_VASE = 75 ,
  SFE_OBJECT_TYPE_SCISSORS = 76 , SFE_OBJECT_TYPE_TEDDYBEAR = 77 , SFE_OBJECT_TYPE_HAIRDRIER
  = 78 , SFE_OBJECT_TYPE_TOOTHBRUSH = 79 }

  Object type enumeration.

- enum SFEVisualCodeCategory { SFE_VISUAL_CODE_CATEGORY_QR_CODE = 0 , SFE_VISUAL_CODE_CATEGORY_PAL = 1 }

  Visual code category enumeration.
- enum SFEModality { SFE_MODALITY_FACE = 0 , SFE_MODALITY_IRIS = 1 , SFE_MODALITY_PALM = 2 , SFE_MODALITY_TATTOO = 3 , SFE_MODALITY_VISUAL_CODE = 4 , SFE_MODALITY_OBJECT = 5 , SFE_MODALITY_PERSON = 6 }

  Detection modality enumeration.

9.31.1 Detailed Description

File containing core detection type definitions.

Author

Innovatrics - SmartFace Embedded

Date

2025-12-02

Definition in file sfe_core_detection.h.

9.31.2 Macro Definition Documentation

9.31.2.1 SFE_IRIS_KEYPOINT_COUNT

```
#define SFE_IRIS_KEYPOINT_COUNT 8
```

Iris keypoints count.
Definition at line 86 of file sfe_core_detection.h.

9.31.2.2 SFE_OBJECT_DETECT

```
#define SFE_OBJECT_DETECT 80
```

Object detection count.
Definition at line 240 of file sfe_core_detection.h.

9.31.2.3 SFE_PALM_KEYPOINT_COUNT

```
#define SFE_PALM_KEYPOINT_COUNT 8
```

Palm keypoints count.
Definition at line 109 of file sfe_core_detection.h.

9.31.2.4 SFE_VISUAL_CODE_KEYPOINT_COUNT

```
#define SFE_VISUAL_CODE_KEYPOINT_COUNT 4
```

Visual code keypoints count.
Definition at line 281 of file sfe_core_detection.h.

9.31.3 Typedef Documentation

9.31.3.1 SFEBoundingBox

```
typedef struct SFEBoundingBox SFEBoundingBox
```

Bounding box.

9.31.3.2 SFEDetection

`typedef struct SFEDetection SFEDetection`
Core detection - tagged union containing all detection types.

Note

Contains a discriminator and a union of detection data types. Bounding box, oriented bounding box, and confidence are common for all detection types.

9.31.3.3 SFEFaceDetectionData

`typedef struct SFEFaceDetectionData SFEFaceDetectionData`

9.31.3.4 SFEHandOrientation

`typedef enum SFEHandOrientation SFEHandOrientation`
Palm orientation enum.

9.31.3.5 SFEHandPosition

`typedef enum SFEHandPosition SFEHandPosition`
Palm hand enum.

9.31.3.6 SFEIrisDetectionData

`typedef struct SFEIrisDetectionData SFEIrisDetectionData`
Iris detection data struct (type-specific fields only)

9.31.3.7 SFEIrisKeypoint

`typedef struct SFEIrisKeypoint SFEIrisKeypoint`
Iris keypoint struct.

9.31.3.8 SFEIrisKeypointType

`typedef enum SFEIrisKeypointType SFEIrisKeypointType`
Iris keypoint types.

9.31.3.9 SFEModality

`typedef enum SFEModality SFEModality`
Detection modality enumeration.

9.31.3.10 SFEModalityData

`typedef union SFEModalityData SFEModalityData`
Union of detection data types (type-specific fields only)

9.31.3.11 SFEObject

`typedef struct SFEObject SFEObject`
Object struct.

9.31.3.12 SFEObjectDetectionData

`typedef struct SFEObjectDetectionData SFEObjectDetectionData`
Object detection data struct (type-specific fields only)

9.31.3.13 SFEObjectType

`typedef enum SFEObjectType SFEObjectType`
Object type enumeration.

9.31.3.14 SFEOrientedBoundingBox

`typedef struct SFEOrientedBoundingBox SFEOrientedBoundingBox`
Oriented bounding box.

9.31.3.15 SFEPalmDetectionData

`typedef struct SFEPalmDetectionData SFEPalmDetectionData`
Palm detection data struct (type-specific fields only)

9.31.3.16 SFEPalmKeypoint

`typedef struct SFEPalmKeypoint SFEPalmKeypoint`
Palm keypoint struct.

9.31.3.17 SFEPalmLandmarks

`typedef struct SFEPalmLandmarks SFEPalmLandmarks`
Palm landmarks (keypoints, hand position, orientation, confidence)

9.31.3.18 SFEPersonDetectionData

`typedef struct SFEPersonDetectionData SFEPersonDetectionData`
Person detection data struct (type-specific fields only)

9.31.3.19 SFETattooDetectionData

`typedef struct SFETattooDetectionData SFETattooDetectionData`
Tattoo detection data struct (type-specific fields only)

9.31.3.20 SFEVisualCodeCategory

`typedef enum SFEVisualCodeCategory SFEVisualCodeCategory`
Visual code category enumeration.

9.31.3.21 SFEVisualCodeDetectionData

`typedef struct SFEVisualCodeDetectionData SFEVisualCodeDetectionData`
Visual code detection data struct (type-specific fields only)

9.31.3.22 SFEVisualCodeKeypoint

`typedef struct SFEVisualCodeKeypoint SFEVisualCodeKeypoint`
Visual code keypoint struct.

9.31.4 Enumeration Type Documentation

9.31.4.1 SFEHandOrientation

`enum SFEHandOrientation`
Palm orientation enum.

Enumerator

SFE_ORIENTATION_UNKNOWN	
SFE_ORIENTATION_PALMAR	

Enumerator

SFE_ORIENTATION_DORSAL	

Definition at line 119 of file sfe_core_detection.h.

```
00119                                  {
00120    SFE_ORIENTATION_UNKNOWN = 0,
00121    SFE_ORIENTATION_PALMAR = 1,
00122    SFE_ORIENTATION_DORSAL = 2,
00123 } SFEHandOrientation;
```

9.31.4.2 SFEHandPosition

`enum SFEHandPosition`

Palm hand enum.

Enumerator

SFE_HAND_UNKNOWN	
SFE_HAND_LEFT	
SFE_HAND_RIGHT	

Definition at line 112 of file sfe_core_detection.h.

```
00112                                  {
00113    SFE_HAND_UNKNOWN = 0,
00114    SFE_HAND_LEFT = 1,
00115    SFE_HAND_RIGHT = 2,
00116 } SFEHandPosition;
```

9.31.4.3 SFEIrisKeypointType

`enum SFEIrisKeypointType`

Iris keypoint types.

Enumerator

SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT	
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT	
SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT	
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT	
SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT	
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT	
SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT	
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT	

Definition at line 62 of file sfe_core_detection.h.

```
00062                                  {
00063    SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT = 0,
00064    SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT = 1,
00065    SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT = 2,
00066    SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT = 3,
00067    SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT = 4,
00068    SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT = 5,
00069    SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT = 6,
00070    SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT = 7,
00071 } SFEIrisKeypointType;
```

9.31.4.4 SFEModality

`enum SFEModality`

Detection modality enumeration.

Enumerator

SFE_MODALITY_FACE	
SFE_MODALITY_IRIS	
SFE_MODALITY_PALM	
SFE_MODALITY_TATTOO	
SFE_MODALITY_VISUAL_CODE	
SFE_MODALITY_OBJECT	
SFE_MODALITY_PERSON	

Definition at line 292 of file sfe_core_detection.h.

```
00292                       {
00293   SFE_MODALITY_FACE = 0,
00294   SFE_MODALITY_IRIS = 1,
00295   SFE_MODALITY_PALM = 2,
00296   SFE_MODALITY_TATTOO = 3,
00297   SFE_MODALITY_VISUAL_CODE = 4,
00298   SFE_MODALITY_OBJECT = 5,
00299   SFE_MODALITY_PERSON = 6,
00300 } SFEModality;
```

9.31.4.5 SFEObjectType

```
enum SFEObjectType
```

Object type enumeration.

Enumerator

SFE_OBJECT_TYPE_PERSON	
SFE_OBJECT_TYPE_BICYCLE	
SFE_OBJECT_TYPE_CAR	
SFE_OBJECT_TYPE_MOTORCYCLE	
SFE_OBJECT_TYPE_AIRPLANE	
SFE_OBJECT_TYPE_BUS	
SFE_OBJECT_TYPE_TRAIN	
SFE_OBJECT_TYPE_TRUCK	
SFE_OBJECT_TYPE_BOAT	
SFE_OBJECT_TYPE_TRAFFICLIGHT	
SFE_OBJECT_TYPE_FIREHYDRANT	
SFE_OBJECT_TYPE_STOPSIGN	
SFE_OBJECT_TYPE_PARKINGMETER	
SFE_OBJECT_TYPE_BENCH	
SFE_OBJECT_TYPE_BIRD	
SFE_OBJECT_TYPE_CAT	
SFE_OBJECT_TYPE_DOG	
SFE_OBJECT_TYPE_HORSE	
SFE_OBJECT_TYPE_SHEEP	
SFE_OBJECT_TYPE_COW	
SFE_OBJECT_TYPE_ELEPHANT	
SFE_OBJECT_TYPE_BEAR	
SFE_OBJECT_TYPE_ZEBRA	
SFE_OBJECT_TYPE_GIRAFFE	
SFE_OBJECT_TYPE_BACKPACK	
SFE_OBJECT_TYPE_UMBRELLA	
SFE_OBJECT_TYPE_HANDBAG	
SFE_OBJECT_TYPE_TIE	
SFE_OBJECT_TYPE_SUITCASE	

Enumerator

SFE_OBJECT_TYPE_FRISBEE	
SFE_OBJECT_TYPE_SKIS	
SFE_OBJECT_TYPE_SNOWBOARD	
SFE_OBJECT_TYPE_SPORTSBALL	
SFE_OBJECT_TYPE_KITE	
SFE_OBJECT_TYPE_BASEBALLBAT	
SFE_OBJECT_TYPE_BASEBALLGLOVE	
SFE_OBJECT_TYPE_SKATEBOARD	
SFE_OBJECT_TYPE_SURFBOARD	
SFE_OBJECT_TYPE_TENNISRACKET	
SFE_OBJECT_TYPE_BOTTLE	
SFE_OBJECT_TYPE_WINEGLASS	
SFE_OBJECT_TYPE_CUP	
SFE_OBJECT_TYPE_FORK	
SFE_OBJECT_TYPE_KNIFE	
SFE_OBJECT_TYPE_SPOON	
SFE_OBJECT_TYPE_BOWL	
SFE_OBJECT_TYPE_BANANA	
SFE_OBJECT_TYPE_APPLE	
SFE_OBJECT_TYPE_SANDWICH	
SFE_OBJECT_TYPE_ORANGE	
SFE_OBJECT_TYPE_BROCCOLI	
SFE_OBJECT_TYPE_CARROT	
SFE_OBJECT_TYPE_HOTDOG	
SFE_OBJECT_TYPE_PIZZA	
SFE_OBJECT_TYPE_DONUT	
SFE_OBJECT_TYPE_CAKE	
SFE_OBJECT_TYPE_CHAIR	
SFE_OBJECT_TYPE_COUCH	
SFE_OBJECT_TYPE_POTTEDPLANT	
SFE_OBJECT_TYPE_BED	
SFE_OBJECT_TYPE_DININGTABLE	
SFE_OBJECT_TYPE_TOILET	
SFE_OBJECT_TYPE_TV	
SFE_OBJECT_TYPE_LAPTOP	
SFE_OBJECT_TYPE_MOUSE	
SFE_OBJECT_TYPE_REMOTE	
SFE_OBJECT_TYPE_KEYBOARD	
SFE_OBJECT_TYPE_CELLPHONE	
SFE_OBJECT_TYPE_MICROWAVE	
SFE_OBJECT_TYPE_OVEN	
SFE_OBJECT_TYPE_TOASTER	
SFE_OBJECT_TYPE_SINK	
SFE_OBJECT_TYPE_REFRIGERATOR	
SFE_OBJECT_TYPE_BOOK	
SFE_OBJECT_TYPE_CLOCK	
SFE_OBJECT_TYPE_VASE	
SFE_OBJECT_TYPE_SCISSORS	
SFE_OBJECT_TYPE_TEDDYBEAR	

Enumerator

SFE_OBJECT_TYPE_HAIRDRIER	
SFE_OBJECT_TYPE_TOOTHBRUSH	

Definition at line 148 of file sfe_core_detection.h.

```
00148                                {
00149     SFE_OBJECT_TYPE_PERSON = 0,
00150     SFE_OBJECT_TYPE_BICYCLE = 1,
00151     SFE_OBJECT_TYPE_CAR = 2,
00152     SFE_OBJECT_TYPE_MOTORCYCLE = 3,
00153     SFE_OBJECT_TYPE_AIRPLANE = 4,
00154     SFE_OBJECT_TYPE_BUS = 5,
00155     SFE_OBJECT_TYPE_TRAIN = 6,
00156     SFE_OBJECT_TYPE_TRUCK = 7,
00157     SFE_OBJECT_TYPE_BOAT = 8,
00158     SFE_OBJECT_TYPE_TRAFFICLIGHT = 9,
00159     SFE_OBJECT_TYPE_FIREHYDRANT = 10,
00160     SFE_OBJECT_TYPE_STOPSIGN = 11,
00161     SFE_OBJECT_TYPE_PARKINGMETER = 12,
00162     SFE_OBJECT_TYPE_BENCH = 13,
00163     SFE_OBJECT_TYPE_BIRD = 14,
00164     SFE_OBJECT_TYPE_CAT = 15,
00165     SFE_OBJECT_TYPE_DOG = 16,
00166     SFE_OBJECT_TYPE_HORSE = 17,
00167     SFE_OBJECT_TYPE_SHEEP = 18,
00168     SFE_OBJECT_TYPE_COW = 19,
00169     SFE_OBJECT_TYPE_ELEPHANT = 20,
00170     SFE_OBJECT_TYPE_BEAR = 21,
00171     SFE_OBJECT_TYPE_ZEBRA = 22,
00172     SFE_OBJECT_TYPE_GIRAFFE = 23,
00173     SFE_OBJECT_TYPE_BACKPACK = 24,
00174     SFE_OBJECT_TYPE_UMBRELLA = 25,
00175     SFE_OBJECT_TYPE_HANDBAG = 26,
00176     SFE_OBJECT_TYPE_TIE = 27,
00177     SFE_OBJECT_TYPE_SUITCASE = 28,
00178     SFE_OBJECT_TYPE_FRISBEE = 29,
00179     SFE_OBJECT_TYPE_SKIS = 30,
00180     SFE_OBJECT_TYPE_SNOWBOARD = 31,
00181     SFE_OBJECT_TYPE_SPORTSBALL = 32,
00182     SFE_OBJECT_TYPE_KITE = 33,
00183     SFE_OBJECT_TYPE_BASEBALLBAT = 34,
00184     SFE_OBJECT_TYPE_BASEBALLGLOVE = 35,
00185     SFE_OBJECT_TYPE_SKATEBOARD = 36,
00186     SFE_OBJECT_TYPE_SURFBOARD = 37,
00187     SFE_OBJECT_TYPE_TENNISRACKET = 38,
00188     SFE_OBJECT_TYPE_BOTTLE = 39,
00189     SFE_OBJECT_TYPE_WINEGLASS = 40,
00190     SFE_OBJECT_TYPE_CUP = 41,
00191     SFE_OBJECT_TYPE_FORK = 42,
00192     SFE_OBJECT_TYPE_KNIFE = 43,
00193     SFE_OBJECT_TYPE_SPOON = 44,
00194     SFE_OBJECT_TYPE_BOWL = 45,
00195     SFE_OBJECT_TYPE_BANANA = 46,
00196     SFE_OBJECT_TYPE_APPLE = 47,
00197     SFE_OBJECT_TYPE_SANDWICH = 48,
00198     SFE_OBJECT_TYPE_ORANGE = 49,
00199     SFE_OBJECT_TYPE_BROCCOLI = 50,
00200     SFE_OBJECT_TYPE_CARROT = 51,
00201     SFE_OBJECT_TYPE_HOTDOG = 52,
00202     SFE_OBJECT_TYPE_PIZZA = 53,
00203     SFE_OBJECT_TYPE_DONUT = 54,
00204     SFE_OBJECT_TYPE_CAKE = 55,
00205     SFE_OBJECT_TYPE_CHAIR = 56,
00206     SFE_OBJECT_TYPE_COUCH = 57,
00207     SFE_OBJECT_TYPE_POTTEDPLANT = 58,
00208     SFE_OBJECT_TYPE_BED = 59,
00209     SFE_OBJECT_TYPE_DININGTABLE = 60,
00210     SFE_OBJECT_TYPE_TOILET = 61,
00211     SFE_OBJECT_TYPE_TV = 62,
00212     SFE_OBJECT_TYPE_LAPTOP = 63,
00213     SFE_OBJECT_TYPE_MOUSE = 64,
00214     SFE_OBJECT_TYPE_REMOTE = 65,
00215     SFE_OBJECT_TYPE_KEYBOARD = 66,
00216     SFE_OBJECT_TYPE_CELLPHONE = 67,
00217     SFE_OBJECT_TYPE_MICROWAVE = 68,
00218     SFE_OBJECT_TYPE_OVEN = 69,
00219     SFE_OBJECT_TYPE_TOASTER = 70,
00220     SFE_OBJECT_TYPE_SINK = 71,
00221     SFE_OBJECT_TYPE_REFRIGERATOR = 72,
00222     SFE_OBJECT_TYPE_BOOK = 73,
00223     SFE_OBJECT_TYPE_CLOCK = 74,
00224     SFE_OBJECT_TYPE_VASE = 75,
00225     SFE_OBJECT_TYPE_SCISSORS = 76,
```

```
00226   SFE_OBJECT_TYPE_TEDDYBEAR = 77,
00227   SFE_OBJECT_TYPE_HAIRDRIER = 78,
00228   SFE_OBJECT_TYPE_TOOTHBRUSH = 79,
00229 } SFEObjectType;
```

9.31.4.6 SFEVisualCodeCategory

enum SFEVisualCodeCategory

Visual code category enumeration.

Enumerator

SFE_VISUAL_CODE_CATEGORY_QR_CODE	
SFE_VISUAL_CODE_CATEGORY_PALM	

Definition at line 267 of file sfe_core_detection.h.

```
00267                                   {
00268   SFE_VISUAL_CODE_CATEGORY_QR_CODE = 0,
00269   SFE_VISUAL_CODE_CATEGORY_PALM = 1,
00270 } SFEVisualCodeCategory;
```

9.32 sfe_core_detection.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include <stdbool.h>
00011 #include <stddef.h>
00012
00013 #ifdef __cplusplus
00014 extern "C" {
00015 #endif
00016
00018 typedef struct SFEBoundingBox {
00021   float x;
00024   float y;
00027   float width;
00030   float height;
00031 } SFEBoundingBox;
00032
00034 typedef struct SFEOrientedBoundingBox {
00037   float center_x;
00040   float center_y;
00043   float width;
00046   float height;
00049   float angle;
00050 } SFEOrientedBoundingBox;
00051
00052 // FACE DETECTION TYPES
00053
00054 typedef struct SFEFaceDetectionData {
00056   char _reserved;
00057 } SFEFaceDetectionData;
00058
00059 // IRIS DETECTION TYPES
00060
00062 typedef enum SFEIrisKeypointType {
00063   SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT = 0,
00064   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT = 1,
00065   SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT = 2,
00066   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT = 3,
00067   SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT = 4,
00068   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT = 5,
00069   SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT = 6,
00070   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT = 7,
00071 } SFEIrisKeypointType;
00072
00074 typedef struct SFEIrisKeypoint {
00076   SFEIrisKeypointType keypoint_type;
00078   float confidence;
00080   float x;
00082   float y;
00083 } SFEIrisKeypoint;
00084
00086 #define SFE_IRIS_KEYPOINT_COUNT 8
00087
00089 typedef struct SFEIrisDetectionData {
```

```
00091   SFEBoundingBox pupil_bounding_box;
00093   SFEIrisKeypoint keypoints[SFE_IRIS_KEYPOINT_COUNT];
00094 } SFEIrisDetectionData;
00095
00096 // PALM DETECTION TYPES
00097
00099 typedef struct SFEPalmKeypoint {
00101   float x;
00103   float y;
00105   float confidence;
00106 } SFEPalmKeypoint;
00107
00109 #define SFE_PALM_KEYPOINT_COUNT 8
00110
00112 typedef enum SFEHandPosition {
00113   SFE_HAND_UNKNOWN = 0,
00114   SFE_HAND_LEFT = 1,
00115   SFE_HAND_RIGHT = 2,
00116 } SFEHandPosition;
00117
00119 typedef enum SFEHandOrientation {
00120   SFE_ORIENTATION_UNKNOWN = 0,
00121   SFE_ORIENTATION_PALMAR = 1,
00122   SFE_ORIENTATION_DORSAL = 2,
00123 } SFEHandOrientation;
00124
00126 typedef struct SFEPalmLandmarks {
00128   SFEPalmKeypoint keypoints[SFE_PALM_KEYPOINT_COUNT];
00130   SFEHandPosition hand_position;
00132   SFEHandOrientation hand_orientation;
00134   float confidence;
00135 } SFEPalmLandmarks;
00136
00138 typedef struct SFEPalmDetectionData {
00140   bool has_landmarks;
00142   SFEPalmLandmarks landmarks;
00143 } SFEPalmDetectionData;
00144
00145 // OBJECT DETECTION TYPES
00146
00148 typedef enum SFEObjectType {
00149   SFE_OBJECT_TYPE_PERSON = 0,
00150   SFE_OBJECT_TYPE_BICYCLE = 1,
00151   SFE_OBJECT_TYPE_CAR = 2,
00152   SFE_OBJECT_TYPE_MOTORCYCLE = 3,
00153   SFE_OBJECT_TYPE_AIRPLANE = 4,
00154   SFE_OBJECT_TYPE_BUS = 5,
00155   SFE_OBJECT_TYPE_TRAIN = 6,
00156   SFE_OBJECT_TYPE_TRUCK = 7,
00157   SFE_OBJECT_TYPE_BOAT = 8,
00158   SFE_OBJECT_TYPE_TRAFFICLIGHT = 9,
00159   SFE_OBJECT_TYPE_FIREHYDRANT = 10,
00160   SFE_OBJECT_TYPE_STOPSIGN = 11,
00161   SFE_OBJECT_TYPE_PARKINGMETER = 12,
00162   SFE_OBJECT_TYPE_BENCH = 13,
00163   SFE_OBJECT_TYPE_BIRD = 14,
00164   SFE_OBJECT_TYPE_CAT = 15,
00165   SFE_OBJECT_TYPE_DOG = 16,
00166   SFE_OBJECT_TYPE_HORSE = 17,
00167   SFE_OBJECT_TYPE_SHEEP = 18,
00168   SFE_OBJECT_TYPE_COW = 19,
00169   SFE_OBJECT_TYPE_ELEPHANT = 20,
00170   SFE_OBJECT_TYPE_BEAR = 21,
00171   SFE_OBJECT_TYPE_ZEBRA = 22,
00172   SFE_OBJECT_TYPE_GIRAFFE = 23,
00173   SFE_OBJECT_TYPE_BACKPACK = 24,
00174   SFE_OBJECT_TYPE_UMBRELLA = 25,
00175   SFE_OBJECT_TYPE_HANDBAG = 26,
00176   SFE_OBJECT_TYPE_TIE = 27,
00177   SFE_OBJECT_TYPE_SUITCASE = 28,
00178   SFE_OBJECT_TYPE_FRISBEE = 29,
00179   SFE_OBJECT_TYPE_SKIS = 30,
00180   SFE_OBJECT_TYPE_SNOWBOARD = 31,
00181   SFE_OBJECT_TYPE_SPORTSBALL = 32,
00182   SFE_OBJECT_TYPE_KITE = 33,
00183   SFE_OBJECT_TYPE_BASEBALLBAT = 34,
00184   SFE_OBJECT_TYPE_BASEBALLGLOVE = 35,
00185   SFE_OBJECT_TYPE_SKATEBOARD = 36,
00186   SFE_OBJECT_TYPE_SURFBOARD = 37,
00187   SFE_OBJECT_TYPE_TENNISRACKET = 38,
00188   SFE_OBJECT_TYPE_BOTTLE = 39,
00189   SFE_OBJECT_TYPE_WINEGLASS = 40,
00190   SFE_OBJECT_TYPE_CUP = 41,
00191   SFE_OBJECT_TYPE_FORK = 42,
00192   SFE_OBJECT_TYPE_KNIFE = 43,
00193   SFE_OBJECT_TYPE_SPOON = 44,
00194   SFE_OBJECT_TYPE_BOWL = 45,
```

```
00195   SFE_OBJECT_TYPE_BANANA = 46,
00196   SFE_OBJECT_TYPE_APPLE = 47,
00197   SFE_OBJECT_TYPE_SANDWICH = 48,
00198   SFE_OBJECT_TYPE_ORANGE = 49,
00199   SFE_OBJECT_TYPE_BROCCOLI = 50,
00200   SFE_OBJECT_TYPE_CARROT = 51,
00201   SFE_OBJECT_TYPE_HOTDOG = 52,
00202   SFE_OBJECT_TYPE_PIZZA = 53,
00203   SFE_OBJECT_TYPE_DONUT = 54,
00204   SFE_OBJECT_TYPE_CAKE = 55,
00205   SFE_OBJECT_TYPE_CHAIR = 56,
00206   SFE_OBJECT_TYPE_COUCH = 57,
00207   SFE_OBJECT_TYPE_POTTEDPLANT = 58,
00208   SFE_OBJECT_TYPE_BED = 59,
00209   SFE_OBJECT_TYPE_DININGTABLE = 60,
00210   SFE_OBJECT_TYPE_TOILET = 61,
00211   SFE_OBJECT_TYPE_TV = 62,
00212   SFE_OBJECT_TYPE_LAPTOP = 63,
00213   SFE_OBJECT_TYPE_MOUSE = 64,
00214   SFE_OBJECT_TYPE_REMOTE = 65,
00215   SFE_OBJECT_TYPE_KEYBOARD = 66,
00216   SFE_OBJECT_TYPE_CELLPHONE = 67,
00217   SFE_OBJECT_TYPE_MICROWAVE = 68,
00218   SFE_OBJECT_TYPE_OVEN = 69,
00219   SFE_OBJECT_TYPE_TOASTER = 70,
00220   SFE_OBJECT_TYPE_SINK = 71,
00221   SFE_OBJECT_TYPE_REFRIGERATOR = 72,
00222   SFE_OBJECT_TYPE_BOOK = 73,
00223   SFE_OBJECT_TYPE_CLOCK = 74,
00224   SFE_OBJECT_TYPE_VASE = 75,
00225   SFE_OBJECT_TYPE_SCISSORS = 76,
00226   SFE_OBJECT_TYPE_TEDDYBEAR = 77,
00227   SFE_OBJECT_TYPE_HAIRDRIER = 78,
00228   SFE_OBJECT_TYPE_TOOTHBRUSH = 79,
00229 } SFEObjectType;
00230
00232 typedef struct SFEObject {
00234   SFEObjectType object_type;
00236   float confidence;
00237 } SFEObject;
00238
00240 #define SFE_OBJECT_DETECT 80
00241
00243 typedef struct SFEObjectDetectionData {
00245   SFEObject objects[SFE_OBJECT_DETECT];
00246 } SFEObjectDetectionData;
00247
00248 // TATTOO DETECTION TYPES
00249
00251 typedef struct SFETattooDetectionData {
00253   char _reserved;
00254 } SFETattooDetectionData;
00255
00256 // PERSON DETECTION TYPES
00257
00259 typedef struct SFEPersonDetectionData {
00261   char _reserved;
00262 } SFEPersonDetectionData;
00263
00264 // VISUAL CODE DETECTION TYPES
00265
00267 typedef enum SFEVisualCodeCategory {
00268   SFE_VISUAL_CODE_CATEGORY_QR_CODE = 0,
00269   SFE_VISUAL_CODE_CATEGORY_PALM = 1,
00270 } SFEVisualCodeCategory;
00271
00273 typedef struct SFEVisualCodeKeypoint {
00275   float x;
00277   float y;
00278 } SFEVisualCodeKeypoint;
00279
00281 #define SFE_VISUAL_CODE_KEYPOINT_COUNT 4
00282
00284 typedef struct SFEVisualCodeDetectionData {
00286   SFEVisualCodeCategory category;
00287 } SFEVisualCodeDetectionData;
00288
00289 // CORE DETECTION TYPES
00290
00292 typedef enum SFEModality {
00293   SFE_MODALITY_FACE = 0,
00294   SFE_MODALITY_IRIS = 1,
00295   SFE_MODALITY_PALM = 2,
00296   SFE_MODALITY_TATTOO = 3,
00297   SFE_MODALITY_VISUAL_CODE = 4,
00298   SFE_MODALITY_OBJECT = 5,
00299   SFE_MODALITY_PERSON = 6,
```

```
00300 } SFEModality;
00301
00303 typedef union SFEModalityData {
00305   SFEFaceDetectionData face;
00307   SFEIrisDetectionData iris;
00309   SFEPalmDetectionData palm;
00311   SFEObjectDetectionData object;
00313   SFETattooDetectionData tattoo;
00315   SFEPersonDetectionData person;
00317   SFEVisualCodeDetectionData visual_code;
00318 } SFEModalityData;
00319
00323 typedef struct SFEDetection {
00325   SFEBoundingBox bounding_box;
00327   SFEOrientedBoundingBox oriented_bounding_box;
00329   float confidence;
00331   SFEModality modality;
00333   SFEModalityData modality_data;
00334 } SFEDetection;
00335
00336 #ifdef __cplusplus
00337 } // extern "C"
00338 #endif
```

9.33 D:/glab/1512fd1724a/1/include/sfe_face.h File Reference

File containing API of sfe_face library as part of SFE Toolkit.
`#include ¨sfe_core.h¨`

Data Structures

- struct SFEFaceLandmarks

  Face landmarks struct.
- struct SFEDetectionInputSize

  Detection input size struct.
- struct SFEFaceTemplate

  Face template struct.
- struct SFEFaceTemplateVersion

  Face template version struct.
- struct SFEFaceLiveness

  Face liveness struct.
- struct SFEFaceArea

  Face area struct.
- struct SFEFaceHeadPose

  Face head pose struct containing angle rotations of head.
- struct SFEFaceQualityAttributes

  Face quality attributes struct.
- struct SFEFaceDemographicAttributes

  Demographic attributes of the detected face.
- struct SFEFaceCrop

  Face crop struct.

Macros

- #define SFE_FACE_LANDMARK_COUNT 23
- #define SFE_FACE_TEMPLATE_SIZE 522

Typedefs

- typedef enum SFEFaceLandmarkType SFEFaceLandmarkType

  Face landmark types.
- typedef struct SFEFaceLandmarks SFEFaceLandmarks

  Face landmarks struct.
- typedef struct SFEDetectionInputSize SFEDetectionInputSize

  Detection input size struct.
- typedef struct SFEFaceTemplate SFEFaceTemplate

  Face template struct.
- typedef struct SFEFaceTemplateVersion SFEFaceTemplateVersion

  Face template version struct.
- typedef struct SFEFaceLiveness SFEFaceLiveness

  Face liveness struct.
- typedef struct SFEFaceArea SFEFaceArea

  Face area struct.
- typedef struct SFEFaceHeadPose SFEFaceHeadPose

  Face head pose struct containing angle rotations of head.
- typedef struct SFEFaceQualityAttributes SFEFaceQualityAttributes

  Face quality attributes struct.
- typedef struct SFEFaceDemographicAttributes SFEFaceDemographicAttributes

  Demographic attributes of the detected face.
- typedef enum SFEFaceDetectionAccuracyType SFEFaceDetectionAccuracyType

  Supported 2 detection accuracy types.
- typedef struct SFEFaceCrop SFEFaceCrop

  Face crop struct.

Enumerations

- enum SFEFaceLandmarkType {
  SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER = 0 , SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENT
  = 1 , SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER = 2 , SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INN
  = 3 ,
  SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER = 4 , SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER
  = 5 , SFE_FACE_LANDMARK_TYPE_NOSE_ROOT = 6 , SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM
  = 7 ,
  SFE_FACE_LANDMARK_TYPE_NOSE_TIP = 8 , SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM
  = 9 , SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM = 10 , SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER
  = 11 ,
  SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER = 12 , SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER
  = 13 , SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE = 14 , SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_ED
  = 15 ,
  SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END = 16 , SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBRO
  = 17 , SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END = 18 , SFE_FACE_LANDMARK_TYPE_LEFT_EYEBR
  = 19 ,
  SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE = 20 , SFE_FACE_LANDMARK_TYPE_CHIN_TIP = 21 ,
  SFE_FACE_LANDMARK_TYPE_LEFT_EDGE = 22 }

  Face landmark types.
- enum SFEFaceDetectionAccuracyType { SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE = 0 ,
  SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED = 1 }

  Supported 2 detection accuracy types.

Functions

- SFEError sfeFaceLandmarks (SFESolver solver, SFEImageView image, const SFEDetection *detection, SFEFaceLandmarks out_face_landmarks[SFE_FACE_LANDMARK_COUNT])

  Detect 23 landmarks of the face detected in the area of source image marked with detection.
- SFEError sfeFaceMaskConfidence (const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], float *out_mask_confidence)

  Get confidence from given landmarks if the face is wearing a face mask.
- SFEError sfeFaceSize (SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], float *face_size)

  Get the face size in pixels from the face landmarks and the source image. Face size is defined as a maximum of values of inter eyes centers distance and distance between center of mouth and center point between eyes (nose root): face_size = max(distance(left_eye_center, right_eye_center), distance(mouth_center, eyes_center))
- SFEError sfeFaceTemplateExtract (SFESolver solver, SFEImageView image, const SFEDetection *detection, const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], SFEFaceTemplate *out_face_template)

  Extract template from source image and given face landmarks.
- SFEError sfeFaceTemplateMatch (SFEFaceTemplate *template1, SFEFaceTemplate *template2, float *out_matching_score)

  Match two face templates.
- SFEError sfeFaceTemplateQuality (SFEFaceTemplate *face_template, float *out_face_template_quality)

  Get template quality.
- SFEError sfeFaceTemplateVersion (SFEFaceTemplate *face_template, SFEFaceTemplateVersion *out_face_template_version)

  Get face template version.
- SFEError sfeFaceTemplateExport (const SFEFaceTemplate *face_template, uint8_t *bytes, size_t *size)

  Export face template to bytes that can be shared between Innovatrics components.
- SFEError sfeFaceTemplateImport (const uint8_t *bytes, size_t size, SFEFaceTemplate *out_face_template)

  Import face template from bytes; format is auto-detected (iface-template protobuf or raw ICF 522-byte).
- SFEError sfeFaceTemplateIdentify (SFEFaceTemplate *probe_face_template, SFEFaceTemplate *templates_gallery, size_t gallery_size, float matching_score_threshold, SFETemplateIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

  Get an ordered array of SFETemplateIdentificationCandidate from tested template best matches of probe template with the templates in templates_gallery.
- SFEError sfeFaceEntityIdentify (SFEFaceTemplate *probe_face_template, SFEFaceTemplate *templates_gallery, SFEEntity *entities_gallery, size_t gallery_size, float matching_score_threshold, SFEEntityIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

  Get an ordered array of SFEEntityIdentificationCandidate from tested template best matches of probe template with the templates in templates_gallery Entity is then used to group the results by entity using the best score of any template associated with the entity.
- SFEError sfeFaceLivenessPassive (SFESolver solver, SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], SFEFaceLiveness *out_liveness)

  Passive liveness score calculation.
- SFEError sfeFaceArea (SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], SFEFaceArea *out_area)

  Get face area.
- SFEError sfeFaceHeadPose (SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], SFEFaceHeadPose *out_head_pose)

  Calculate angle rotations of head towards camera reference frame from given landmarks.
- SFEError sfeFaceDemographicAttributes (SFESolver solver, SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], SFEFaceDemographicAttributes *out_demographic_attributes)

  Estimate demographic attributes (age, gender) from a face image.
- SFEError sfeFaceQualityAttributes (SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_CO SFEFaceQualityAttributes *out_quality_attributes)

Calculate face image quality attributes from given source image and landmarks data.

- SFEError sfeFaceDetectInputSize (SFEImageView image, SFEFaceDetectionAccuracyType detection_↩ mode, size_t min_face_size, size_t max_face_size, SFEDetectionInputSize *out_input_size)

  Calculation of recommended input image width and height according to desired minimal and maximal size of detected faces. The input combination of required maximal and minimal face sizes is validated against the model constraints. If valid, the recommended width and height of the image are calculated. If invalid, the error is returned. It is recommended to set the max_face_size to `max_face_size < 30 * min_face_size`. Performance could decrease if you do not comply with this requirement.

- SFEError sfeFaceCrop (SFEImageView image, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], float face_size_extension, float max_face_size, SFEFaceCrop *out_crop)

  Crop the face according the given landmarks, the face size extension and the maximal face size. The face size extension defines the size of the crop as an extension of the detection bounding box. The crop will be centered on the detection bounding box and the size will be multiplied by this value. Valid values are 0, 1, 2, 3, 4 and 5.

9.33.1 Detailed Description

File containing API of sfe_face library as part of SFE Toolkit.

Author

Innovatrics - SmartFace Embedded

Date

16.5.2023

Definition in file sfe_face.h.

9.33.2 Macro Definition Documentation

9.33.2.1 SFE_FACE_LANDMARK_COUNT

```
#define SFE_FACE_LANDMARK_COUNT 23
```

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, and example_face_liveness.cpp.

Definition at line 17 of file sfe_face.h.

9.33.2.2 SFE_FACE_TEMPLATE_SIZE

```
#define SFE_FACE_TEMPLATE_SIZE 522
```

Definition at line 104 of file sfe_face.h.

9.33.3 Typedef Documentation

9.33.3.1 SFEDetectionInputSize

```
typedef struct SFEDetectionInputSize SFEDetectionInputSize
```

Detection input size struct.

9.33.3.2 SFEFaceArea

```
typedef struct SFEFaceArea SFEFaceArea
```

Face area struct.

9.33.3.3 SFEFaceCrop

```
typedef struct SFEFaceCrop SFEFaceCrop
```

Face crop struct.

9.33.3.4 SFEFaceDemographicAttributes

`typedef struct SFEFaceDemographicAttributes SFEFaceDemographicAttributes`
Demographic attributes of the detected face.

9.33.3.5 SFEFaceDetectionAccuracyType

`typedef enum SFEFaceDetectionAccuracyType SFEFaceDetectionAccuracyType`
Supported 2 detection accuracy types.

9.33.3.6 SFEFaceHeadPose

`typedef struct SFEFaceHeadPose SFEFaceHeadPose`
Face head pose struct containing angle rotations of head.

9.33.3.7 SFEFaceLandmarks

`typedef struct SFEFaceLandmarks SFEFaceLandmarks`
Face landmarks struct.

9.33.3.8 SFEFaceLandmarkType

`typedef enum SFEFaceLandmarkType SFEFaceLandmarkType`
Face landmark types.

9.33.3.9 SFEFaceLiveness

`typedef struct SFEFaceLiveness SFEFaceLiveness`
Face liveness struct.

9.33.3.10 SFEFaceQualityAttributes

`typedef struct SFEFaceQualityAttributes SFEFaceQualityAttributes`
Face quality attributes struct.

9.33.3.11 SFEFaceTemplate

`typedef struct SFEFaceTemplate SFEFaceTemplate`
Face template struct.

9.33.3.12 SFEFaceTemplateVersion

`typedef struct SFEFaceTemplateVersion SFEFaceTemplateVersion`
Face template version struct.

9.33.4 Enumeration Type Documentation

9.33.4.1 SFEFaceDetectionAccuracyType

`enum SFEFaceDetectionAccuracyType`
Supported 2 detection accuracy types.

Enumerator

SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE	
SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED	

Definition at line 342 of file sfe_face.h.

```
00342                                   {
00343   SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE = 0,
00344   SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED = 1,
```

```
00345 } SFEFaceDetectionAccuracyType;
```

9.33.4.2 SFEFaceLandmarkType

```
enum SFEFaceLandmarkType
```

Face landmark types.

Enumerator

SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER	
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER	
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER	
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER	
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER	
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER	
SFE_FACE_LANDMARK_TYPE_NOSE_ROOT	
SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM	
SFE_FACE_LANDMARK_TYPE_NOSE_TIP	
SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM	
SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM	
SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER	
SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER	
SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER	
SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE	
SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE	
SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END	
SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END	
SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END	
SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END	
SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE	
SFE_FACE_LANDMARK_TYPE_CHIN_TIP	
SFE_FACE_LANDMARK_TYPE_LEFT_EDGE	

Definition at line 20 of file sfe_face.h.

```
00020                                   {
00021   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER = 0,
00022   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER = 1,
00023   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER = 2,
00024   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER = 3,
00025   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER = 4,
00026   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER = 5,
00027   SFE_FACE_LANDMARK_TYPE_NOSE_ROOT = 6,
00028   SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM = 7,
00029   SFE_FACE_LANDMARK_TYPE_NOSE_TIP = 8,
00030   SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM = 9,
00031   SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM = 10,
00032   SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER = 11,
00033   SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER = 12,
00034   SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER = 13,
00035   SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE = 14,
00036   SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE = 15,
00037   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END = 16,
00038   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END = 17,
00039   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END = 18,
00040   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END = 19,
00041   SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE = 20,
00042   SFE_FACE_LANDMARK_TYPE_CHIN_TIP = 21,
00043   SFE_FACE_LANDMARK_TYPE_LEFT_EDGE = 22,
00044 } SFEFaceLandmarkType;
```

9.33.5 Function Documentation

9.33.5.1 sfeFaceArea()

```
SFEError sfeFaceArea (
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            SFEFaceArea * out_area )
```

Get face area.

Parameters

in	*image*	source image
in	*face_landmarks*	face landmarks detected in source image
out	*out_area*	OUT: structure containing the face size, the face area relative to the source image, the face area intersected with the source image and relative to the source image.

Returns

Error in case of failed area struct calculations

Examples

example_face_liveness.cpp.

9.33.5.2 sfeFaceCrop()

```
SFEError sfeFaceCrop (
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            float face_size_extension,
            float max_face_size,
            SFEFaceCrop * out_crop )
```

Crop the face according the given landmarks, the face size extension and the maximal face size. The face size extension defines the size of the crop as an extension of the detection bounding box. The crop will be centered on the detection bounding box and the size will be multiplied by this value. Valid values are 0, 1, 2, 3, 4 and 5.

Parameters

in	*image*	Source image
in	*face_landmarks*	Array of face landmarks
in	*face_size_extension*	Defines the size of the crop as an extension of the detection bounding box.
in	*max_face_size*	Defines the maximum size of the face in the crop area. If the actual face size is larger, the crop image will be downscaled.
out	*out_crop*	OUT: Structure containing used crop extension, calculated crop's bounding box and the cropped image of the face

Examples

example_face_identify.cpp.

9.33.5.3 sfeFaceDemographicAttributes()

```
SFEError sfeFaceDemographicAttributes (
            SFESolver solver,
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            SFEFaceDemographicAttributes * out_demographic_attributes )
```

Estimate demographic attributes (age, gender) from a face image.

Parameters

in	*solver*	demographic attributes solver
in	*image*	source image
in	*face_landmarks*	face landmarks detected in source image
out	*out_demographic_attributes*	OUT: estimated demographic attributes

Returns

Error in case of failed demographic attributes estimation

Examples

example_face_demographic_attributes.cpp.

9.33.5.4 sfeFaceDetectInputSize()

```
SFEError sfeFaceDetectInputSize (
            SFEImageView image,
            SFEFaceDetectionAccuracyType detection_mode,
            size_t min_face_size,
            size_t max_face_size,
            SFEDetectionInputSize * out_input_size )
```

Calculation of recommended input image width and height according to desired minimal and maximal size of detected faces. The input combination of required maximal and minimal face sizes is validated against the model constraints. If valid, the recommended width and height of the image are calculated. If invalid, the error is returned. It is recommended to set the max_face_size to `max_face_size < 30 * min_face_size`. Performance could decrease if you do not comply with this requirement.

Parameters

in	*image*	Source image
in	*detection_mode*	Accuracy type, it can be accurate or balanced
in	*min_face_size*	Desired minimal size of detected face
in	*max_face_size*	Desired maximal size of detected face
out	*out_input_size*	Recommended image dimensions (width and height)

Returns

Error in case of non valid input combination

Examples

example_face_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, and example_face_track.cpp.

9.33.5.5 sfeFaceEntityIdentify()

```
SFEError sfeFaceEntityIdentify (
            SFEFaceTemplate * probe_face_template,
            SFEFaceTemplate * templates_gallery,
            SFEEntity * entities_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFEEntityIdentificationCandidate * out_candidates,
```

```
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFEEntityIdentificationCandidate from tested template best matches of probe template with the templates in templates_gallery Entity is then used to group the results by entity using the best score of any template associated with the entity.

Parameters

in	*probe_face_template*	probe template
in	*templates_gallery*	pointer to an array of SFEFaceTemplate
in	*entities_gallery*	pointer to an array of SFEEntity that corresponds with templates_gallery
in	*gallery_size*	number of templates in gallery
in	*matching_score_threshold*	threshold for matching - range <0,1> Function will return only candidates with score above the threshold
out	*out_candidates*	Pointer to an ordered array of candidates
in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found entities with best score above the threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification

Examples

example_face_identify_entity.cpp.

9.33.5.6 sfeFaceHeadPose()

```
SFEError sfeFaceHeadPose (
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            SFEFaceHeadPose * out_head_pose )
```

Calculate angle rotations of head towards camera reference frame from given landmarks.

Parameters

in	*face_landmarks*	face landmarks detected in source image
out	*out_head_pose*	OUT: structure containing the angle rotations - roll, yaw and pitch

Returns

Error in case of failed head pose calculationse

Examples

example_face_liveness.cpp.

9.33.5.7 sfeFaceLandmarks()

```
SFEError sfeFaceLandmarks (
            SFESolver solver,
            SFEImageView image,
            const SFEDetection * detection,
            SFEFaceLandmarks out_face_landmarks[SFE_FACE_LANDMARK_COUNT] )
```

Detect 23 landmarks of the face detected in the area of source image marked with detection.

Parameters

in	*solver*	face landmarks solver
in	*image*	source image
in	*detection*	core detection containing the detected face
out	*out_face_landmarks*	OUT: pointer to an array of detected landmarks, the size of the array should be 23 which is the number of landmarks detected by the solver

Returns

Error in case of failed landmarks detection.

Examples

example_face_demographic_attributes.cpp, example_face_identify.cpp, example_face_identify_entity.cpp, and example_face_liveness.cpp.

9.33.5.8 sfeFaceLivenessPassive()

```
SFEError sfeFaceLivenessPassive (
            SFESolver solver,
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            SFEFaceLiveness * out_liveness )
```

Passive liveness score calculation.

Parameters

in	*solver*	passive liveness solver
in	*image*	source image
in	*face_landmarks*	face landmarks detected in source image
out	*out_liveness*	OUT: structure containing normalized score of passive liveness

Returns

Error in case of failed passive liveness score calculation

Examples

example_face_liveness.cpp.

9.33.5.9 sfeFaceMaskConfidence()

```
SFEError sfeFaceMaskConfidence (
            const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            float * out_mask_confidence )
```

Get confidence from given landmarks if the face is wearing a face mask.

Parameters

in	*face_landmarks*	array of detected landmarks, the size of the array should be 23 which is the number of landmarks detected by the solver
out	*out_mask_confidence*	OUT: confidence of wearing a face mask - range <0-1>

Examples

example_face_liveness.cpp.

9.33.5.10 sfeFaceQualityAttributes()

```
SFEError sfeFaceQualityAttributes (
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            SFEFaceQualityAttributes * out_quality_attributes )
```

Calculate face image quality attributes from given source image and landmarks data.

Parameters

in	*image*	source image
in	*face_landmarks*	face landmarks detected in source image
out	*out_quality_attributes*	structure containing normalized `sharpness`, `brightness`, `contrast` and `unique_intensity_levels`

Returns

Error in case of failed quality attributes calculation

Examples

example_face_liveness.cpp.

9.33.5.11 sfeFaceSize()

```
SFEError sfeFaceSize (
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            float * face_size )
```

Get the face size in pixels from the face landmarks and the source image. Face size is defined as a maximum of values of inter eyes centers distance and distance between center of mouth and center point between eyes (nose root): face_size = max(distance(left_eye_center, right_eye_center), distance(mouth_center, eyes_center))

Parameters

in	*image*	source image
in	*face_landmarks*	array of face landmarks
out	*face_size*	OUT: size of the face in pixels

Returns

Error

Examples

example_face_liveness.cpp.

9.33.5.12 sfeFaceTemplateExport()

```
SFEError sfeFaceTemplateExport (
            const SFEFaceTemplate * face_template,
            uint8_t * bytes,
            size_t * size )
```

Export face template to bytes that can be shared between Innovatrics components.

Parameters

in	*face_template*	face template to export

Parameters

in,out	*bytes*	buffer to export the face template to
in,out	*size*	IN: capacity of bytes buffer, OUT: number of bytes written

Returns

Error in case of failed export

Examples

example_face_identify.cpp.

9.33.5.13 sfeFaceTemplateExtract()

```
SFEError sfeFaceTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            const SFEDetection * detection,
            const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            SFEFaceTemplate * out_face_template )
```

Extract template from source image and given face landmarks.

Parameters

in	*solver*	template extraction solver
in	*image*	source image
in	*detection*	core detection containing the detected face
in	*face_landmarks*	face landmarks detected in source image
out	*out_face_template*	extracted face template

Returns

Error in case of failed extraction.

Examples

example_face_identify.cpp, and example_face_identify_entity.cpp.

9.33.5.14 sfeFaceTemplateIdentify()

```
SFEError sfeFaceTemplateIdentify (
            SFEFaceTemplate * probe_face_template,
            SFEFaceTemplate * templates_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFETemplateIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFETemplateIdentificationCandidate from tested template best matches of probe template with the templates in templates_gallery.

Parameters

in	*probe_face_template*	probe template
in	*gallery_size*	number of templates in gallery
in	*templates_gallery*	pointer to an array of SFEFaceTemplate

Parameters

in	*matching_score_threshold*	threshold for matching - range <0,1> Function will return only candidates with score above the threshold
out	*out_candidates*	Pointer to an ordered array of candidates
in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification

Examples

example_face_identify.cpp.

9.33.5.15 sfeFaceTemplateImport()

```
SFEError sfeFaceTemplateImport (
            const uint8_t * bytes,
            size_t size,
            SFEFaceTemplate * out_face_template )
```

Import face template from bytes; format is auto-detected (iface-template protobuf or raw ICF 522-byte).

Parameters

in	*bytes*	bytes to import the face template from
in	*size*	size of the bytes
out	*out_face_template*	imported face template

Returns

Error in case of failed import

Examples

example_face_identify.cpp.

9.33.5.16 sfeFaceTemplateMatch()

```
SFEError sfeFaceTemplateMatch (
            SFEFaceTemplate * template1,
            SFEFaceTemplate * template2,
            float * out_matching_score )
```

Match two face templates.

Parameters

in	*template1*	first template to match
in	*template2*	second template to match
out	*out_matching_score*	OUT: matching score - range <0,1>

Returns

Error in case of failed matching

Examples

example_face_identify.cpp.

9.33.5.17 sfeFaceTemplateQuality()

```
SFEError sfeFaceTemplateQuality (
            SFEFaceTemplate * face_template,
            float * out_face_template_quality )
```

Get template quality.

Parameters

in	*face_template*	source template
out	*out_face_template_quality*	OUT: template quality - range <0,1>

Returns

Error in case of template checksum mismatch

9.33.5.18 sfeFaceTemplateVersion()

```
SFEError sfeFaceTemplateVersion (
            SFEFaceTemplate * face_template,
            SFEFaceTemplateVersion * out_face_template_version )
```

Get face template version.

Parameters

in	*face_template*	source template
out	*out_face_template_version*	OUT: face template version struct

Returns

Error in case of template checksum mismatch

Examples

example_face_identify.cpp.

9.34 sfe_face.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include "sfe_core.h"
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
00015
00016 // FACE LANDMARKS
00017 #define SFE_FACE_LANDMARK_COUNT 23
00018
00020 typedef enum SFEFaceLandmarkType {
00021   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER = 0,
00022   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER = 1,
00023   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER = 2,
```

```
00024   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER = 3,
00025   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER = 4,
00026   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER = 5,
00027   SFE_FACE_LANDMARK_TYPE_NOSE_ROOT = 6,
00028   SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM = 7,
00029   SFE_FACE_LANDMARK_TYPE_NOSE_TIP = 8,
00030   SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM = 9,
00031   SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM = 10,
00032   SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER = 11,
00033   SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER = 12,
00034   SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER = 13,
00035   SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE = 14,
00036   SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE = 15,
00037   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END = 16,
00038   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END = 17,
00039   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END = 18,
00040   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END = 19,
00041   SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE = 20,
00042   SFE_FACE_LANDMARK_TYPE_CHIN_TIP = 21,
00043   SFE_FACE_LANDMARK_TYPE_LEFT_EDGE = 22,
00044 } SFEFaceLandmarkType;
00045
00047 typedef struct SFEFaceLandmarks {
00049   SFEFaceLandmarkType landmark_type;
00051   float confidence;
00053   float x;
00055   float y;
00056 } SFEFaceLandmarks;
00057
00059 typedef struct SFEDetectionInputSize {
00061   uint32_t width;
00063   uint32_t height;
00064 } SFEDetectionInputSize;
00065
00075 SFEError
00076 sfeFaceLandmarks(SFESolver solver, SFEImageView image,
00077                  const SFEDetection *detection,
00078                  SFEFaceLandmarks out_face_landmarks[SFE_FACE_LANDMARK_COUNT]);
00079
00086 SFEError sfeFaceMaskConfidence(
00087     const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00088     float *out_mask_confidence);
00089
00099 SFEError sfeFaceSize(SFEImageView image,
00100                      SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00101                      float *face_size);
00102
00103 // FACE TEMPLATE
00104 #define SFE_FACE_TEMPLATE_SIZE 522
00105
00107 typedef struct SFEFaceTemplate {
00108   uint8_t data[SFE_FACE_TEMPLATE_SIZE];
00109 } SFEFaceTemplate;
00110
00112 typedef struct SFEFaceTemplateVersion {
00114   uint8_t version_major;
00116   uint8_t version_minor;
00117 } SFEFaceTemplateVersion;
00118
00126 SFEError
00127 sfeFaceTemplateExtract(SFESolver solver, SFEImageView image,
00128                        const SFEDetection *detection,
00129                        const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00130                        SFEFaceTemplate *out_face_template);
00131
00137 SFEError sfeFaceTemplateMatch(SFEFaceTemplate *template1,
00138                               SFEFaceTemplate *template2,
00139                               float *out_matching_score);
00140
00145 SFEError sfeFaceTemplateQuality(SFEFaceTemplate *face_template,
00146                                 float *out_face_template_quality);
00147
00152 SFEError
00153 sfeFaceTemplateVersion(SFEFaceTemplate *face_template,
00154                        SFEFaceTemplateVersion *out_face_template_version);
00155
00161 SFEError sfeFaceTemplateExport(const SFEFaceTemplate *face_template,
00162                                uint8_t *bytes,
00163                                size_t *size);
00164
00170 SFEError sfeFaceTemplateImport(const uint8_t *bytes,
00171                                size_t size,
00172                                SFEFaceTemplate *out_face_template);
00173
00187 SFEError
00188 sfeFaceTemplateIdentify(SFEFaceTemplate *probe_face_template,
00189                         SFEFaceTemplate *templates_gallery, size_t gallery_size,
```

```
00190                         float matching_score_threshold,
00191                         SFETemplateIdentificationCandidate *out_candidates,
00192                         size_t *in_out_candidates_count, size_t thread_count);
00193
00211 SFEError sfeFaceEntityIdentify(SFEFaceTemplate *probe_face_template,
00212                                 SFEFaceTemplate *templates_gallery,
00213                                 SFEEntity *entities_gallery, size_t gallery_size,
00214                                 float matching_score_threshold,
00215                                 SFEEntityIdentificationCandidate *out_candidates,
00216                                 size_t *in_out_candidates_count,
00217                                 size_t thread_count);
00218
00220 typedef struct SFEFaceLiveness {
00222   float score;
00223 } SFEFaceLiveness;
00224
00231 SFEError
00232 sfeFaceLivenessPassive(SFESolver solver, SFEImageView image,
00233                        SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00234                        SFEFaceLiveness *out_liveness);
00235
00237 typedef struct SFEFaceArea {
00239   float size;
00241   float area;
00244   float area_in_image;
00245 } SFEFaceArea;
00246
00254 SFEError sfeFaceArea(SFEImageView image,
00255                      SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00256                      SFEFaceArea *out_area);
00257
00259 typedef struct SFEFaceHeadPose {
00262   float pitch;
00265   float yaw;
00268   float roll;
00269 } SFEFaceHeadPose;
00270
00277 SFEError
00278 sfeFaceHeadPose(SFEImageView image,
00279                 SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00280                 SFEFaceHeadPose *out_head_pose);
00281
00283 typedef struct SFEFaceQualityAttributes {
00288   float sharpness;
00293   float brightness;
00299   float contrast;
00305   float unique_intensity_levels;
00306 } SFEFaceQualityAttributes;
00307
00309 typedef struct SFEFaceDemographicAttributes {
00311   float age;
00314   float gender;
00315 } SFEFaceDemographicAttributes;
00316
00323 SFEError sfeFaceDemographicAttributes(
00324     SFESolver solver,
00325     SFEImageView image,
00326     SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00327     SFEFaceDemographicAttributes *out_demographic_attributes);
00328
00336 SFEError sfeFaceQualityAttributes(
00337     SFEImageView image,
00338     SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00339     SFEFaceQualityAttributes *out_quality_attributes);
00340
00342 typedef enum SFEFaceDetectionAccuracyType {
00343   SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE = 0,
00344   SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED = 1,
00345 } SFEFaceDetectionAccuracyType;
00346
00361 SFEError sfeFaceDetectInputSize(SFEImageView image,
00362                                 SFEFaceDetectionAccuracyType detection_mode,
00363                                 size_t min_face_size, size_t max_face_size,
00364                                 SFEDetectionInputSize *out_input_size);
00365
00367 typedef struct SFEFaceCrop {
00370   float face_size_extension;
00372   SFEBoundingBox crop_box;
00374   SFEImage crop_image;
00375 } SFEFaceCrop;
00376
00390 SFEError sfeFaceCrop(SFEImageView image,
00391                      SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00392                      float face_size_extension, float max_face_size,
00393                      SFEFaceCrop *out_crop);
00394
00395 #ifdef __cplusplus
```

```
00396 } // extern "C"
00397 #endif
```

9.35 D:/glab/1512fd1724a/1/include/sfe_iris.h File Reference

File containing API of sfe_iris library as part of SFE Toolkit.
`#include ¨sfe_core.h¨`

Data Structures

- struct SFEIrisTemplate

  Iris template struct.
- struct SFEIrisTemplateVersion

  Iris template version struct.

Macros

- #define SFE_IRIS_TEMPLATE_SIZE 522

  Iris template size in bytes.

Typedefs

- typedef struct SFEIrisTemplate SFEIrisTemplate

  Iris template struct.
- typedef struct SFEIrisTemplateVersion SFEIrisTemplateVersion

  Iris template version struct.

Functions

- SFEError sfeIrisTemplateExtract (SFESolver solver, SFEImageView image, const SFEDetection *detection, SFEIrisTemplate *out_iris_template)

  Extract template from source image and given iris detection.
- SFEError sfeIrisTemplateMatch (const SFEIrisTemplate *template1, const SFEIrisTemplate *template2, float *out_matching_score)

  Match two iris templates.
- SFEError sfeIrisTemplateVersion (const SFEIrisTemplate *iris_template, SFEIrisTemplateVersion *out_iris↩_template_version)

  Get iris template version.
- SFEError sfeIrisTemplateQuality (const SFEIrisTemplate *iris_template, float *out_iris_template_quality)

  Get iris template quality.
- SFEError sfeIrisTemplateIdentify (const SFEIrisTemplate *probe_iris_template, const SFEIrisTemplate *templates_gallery, size_t gallery_size, float matching_score_threshold, SFETemplateIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

  Get an ordered array of SFETemplateIdentificationCandidate from probe's template best matches with templates in the gallery.
- SFEError sfeIrisEntityIdentify (const SFEIrisTemplate *probe_iris_template, const SFEIrisTemplate *templates_gallery, const SFEEntity *entities_gallery, size_t gallery_size, float matching_score_threshold, SFEEntityIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

  Get an ordered array of SFEEntityIdentificationCandidate from probe's template best matches with templates in the gallery. Entity is then used to group the results by entity using the best score of any template associated with the entity.

9.35.1 Detailed Description

File containing API of sfe_iris library as part of SFE Toolkit.

Author

Innovatrics - SmartFace Embedded

Date

2.1.2024

Definition in file sfe_iris.h.

9.35.2 Macro Definition Documentation

9.35.2.1 SFE_IRIS_TEMPLATE_SIZE

```
#define SFE_IRIS_TEMPLATE_SIZE 522
```

Iris template size in bytes.
Definition at line 19 of file sfe_iris.h.

9.35.3 Typedef Documentation

9.35.3.1 SFEIrisTemplate

```
typedef struct SFEIrisTemplate SFEIrisTemplate
```

Iris template struct.

9.35.3.2 SFEIrisTemplateVersion

```
typedef struct SFEIrisTemplateVersion SFEIrisTemplateVersion
```

Iris template version struct.

9.35.4 Function Documentation

9.35.4.1 sfeIrisEntityIdentify()

```
SFEError sfeIrisEntityIdentify (
            const SFEIrisTemplate * probe_iris_template,
            const SFEIrisTemplate * templates_gallery,
            const SFEEntity * entities_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFEEntityIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFEEntityIdentificationCandidate from probe's template best matches with templates in the gallery. Entity is then used to group the results by entity using the best score of any template associated with the entity.

Parameters

in	*probe_iris_template*	probe template
in	*templates_gallery*	pointer to an array of SFEIrisTemplate
in	*entities_gallery*	pointer to an array of SFEEntity that corresponds with templates_gallery
in	*gallery_size*	number of templates in gallery
in	*matching_score_threshold*	threshold for matching - range <0,1>
out	*out_candidates*	Pointer to an ordered array of candidates

Parameters

in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the matching score threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification.

9.35.4.2 sfeIrisTemplateExtract()

```
SFEError sfeIrisTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            const SFEDetection * detection,
            SFEIrisTemplate * out_iris_template )
```

Extract template from source image and given iris detection.

Parameters

in	*solver*	iris template extraction solver
in	*image*	source image
in	*detection*	core detection containing the detected iris
out	*out_iris_template*	extracted iris template

Returns

Error in case of failed extraction.

Examples

example_iris_identify.cpp.

9.35.4.3 sfeIrisTemplateIdentify()

```
SFEError sfeIrisTemplateIdentify (
            const SFEIrisTemplate * probe_iris_template,
            const SFEIrisTemplate * templates_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFETemplateIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFETemplateIdentificationCandidate from probe's template best matches with templates in the gallery.

Parameters

in	*probe_iris_template*	probe template
in	*templates_gallery*	pointer to an array of SFIrisTemplate
in	*gallery_size*	number of templates in the gallery
in	*matching_score_threshold*	threshold for matching - range <0,1>
out	*out_candidates*	Pointer to an ordered array of candidates

Parameters

in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the matching score threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification.

Examples

example_iris_identify.cpp.

9.35.4.4 sfeIrisTemplateMatch()

```
SFEError sfeIrisTemplateMatch (
            const SFEIrisTemplate * template1,
            const SFEIrisTemplate * template2,
            float * out_matching_score )
```

Match two iris templates.

Parameters

in	*template1*	first template to match
in	*template2*	second template to match
out	*out_matching_score*	matching score - range <0,1>

Returns

Error in case of failed matching.

Examples

example_iris_identify.cpp.

9.35.4.5 sfeIrisTemplateQuality()

```
SFEError sfeIrisTemplateQuality (
            const SFEIrisTemplate * iris_template,
            float * out_iris_template_quality )
```

Get iris template quality.

Parameters

in	*iris_template*	source iris template
out	*out_iris_template_quality*	template quality - range <0,1>

Returns

Error in case of template checksum mismatch.

Examples

example_iris_identify.cpp.

9.35.4.6 sfeIrisTemplateVersion()

```
SFEError sfeIrisTemplateVersion (
            const SFEIrisTemplate * iris_template,
            SFEIrisTemplateVersion * out_iris_template_version )
```

Get iris template version.

Parameters

in	*iris_template*	source iris template
out	*out_iris_template_version*	iris template version struct

Returns

Error in case of template checksum mismatch.

Examples

example_iris_identify.cpp.

9.36 sfe_iris.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include "sfe_core.h"
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
00015
00016 // IRIS TEMPLATE EXTRACTION AND MATCHING
00017
00019 #define SFE_IRIS_TEMPLATE_SIZE 522
00020
00022 typedef struct SFEIrisTemplate {
00023   uint8_t data[SFE_IRIS_TEMPLATE_SIZE];
00024 } SFEIrisTemplate;
00025
00027 typedef struct SFEIrisTemplateVersion {
00029   uint8_t version_major;
00031   uint8_t version_minor;
00032 } SFEIrisTemplateVersion;
00033
00040 SFEError sfeIrisTemplateExtract(SFESolver solver, SFEImageView image,
00041                                 const SFEDetection *detection,
00042                                 SFEIrisTemplate *out_iris_template);
00043
00049 SFEError sfeIrisTemplateMatch(const SFEIrisTemplate *template1,
00050                               const SFEIrisTemplate *template2,
00051                               float *out_matching_score);
00052
00057 SFEError
00058 sfeIrisTemplateVersion(const SFEIrisTemplate *iris_template,
00059                        SFEIrisTemplateVersion *out_iris_template_version);
00060
00065 SFEError sfeIrisTemplateQuality(const SFEIrisTemplate *iris_template,
00066                                 float *out_iris_template_quality);
00067
00080 SFEError
00081 sfeIrisTemplateIdentify(const SFEIrisTemplate *probe_iris_template,
00082                         const SFEIrisTemplate *templates_gallery,
00083                         size_t gallery_size, float matching_score_threshold,
00084                         SFETemplateIdentificationCandidate *out_candidates,
00085                         size_t *in_out_candidates_count, size_t thread_count);
00086
00103 SFEError sfeIrisEntityIdentify(const SFEIrisTemplate *probe_iris_template,
00104                                const SFEIrisTemplate *templates_gallery,
00105                                const SFEEntity *entities_gallery,
00106                                size_t gallery_size,
00107                                float matching_score_threshold,
00108                                SFEEntityIdentificationCandidate *out_candidates,
00109                                size_t *in_out_candidates_count,
00110                                size_t thread_count);
00111
```

```
00112 #ifdef __cplusplus
00113 } // extern "C"
00114 #endif
```

9.37 D:/glab/1512fd1724a/1/include/sfe_palm.h File Reference

File containing API of sfe_palm library as part of SFE Toolkit.

```
#include ¨sfe_core.h¨
```

Data Structures

- struct SFEPalmTemplate

  Palm template struct.
- struct SFEPalmTemplateVersion

  Palm template version struct.
- struct SFEPalmAttributes

Macros

- #define SFE_PALM_TEMPLATE_SIZE 522

  Palm template size in bytes.

Typedefs

- typedef struct SFEPalmTemplate SFEPalmTemplate

  Palm template struct.
- typedef struct SFEPalmTemplateVersion SFEPalmTemplateVersion

  Palm template version struct.
- typedef struct SFEPalmAttributes SFEPalmAttributes

Functions

- SFEError sfePalmLandmarks (SFESolver solver, SFEImageView image, const SFEDetection *detection, SFEPalmLandmarks *out_landmarks)

  Calculate palm landmarks from given source image and palm detection.
- SFEError sfePalmAttributes (SFESolver solver, SFEImageView image, const SFEPalmLandmarks *landmarks, SFEPalmAttributes *out_attributes)

  Calculate palm image quality attributes from given source image and palm detection.
- SFEError sfePalmTemplateExtract (SFESolver solver, SFEImageView image, const SFEPalmLandmarks *landmarks, SFEPalmTemplate *out_palm_template)

  Extract template from source image and given palm attributes.
- SFEError sfePalmTemplateMatch (const SFEPalmTemplate *template1, const SFEPalmTemplate *template2, float *out_matching_score)

  Match two palm templates.
- SFEError sfePalmTemplateVersion (const SFEPalmTemplate *palm_template, SFEPalmTemplateVersion *out_palm_template_version)

  Get palm template version.
- SFEError sfePalmTemplateIdentify (const SFEPalmTemplate *probe_palm_template, const SFEPalmTemplate *templates_gallery, size_t gallery_size, float matching_score_threshold, SFETemplateIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

  Get an ordered array of SFETemplateIdentificationCandidate from probe's template best matches with templates in the gallery.
- SFEError sfePalmEntityIdentify (const SFEPalmTemplate *probe_palm_template, const SFEPalmTemplate *templates_gallery, const SFEEntity *entities_gallery, size_t gallery_size, float matching_score_threshold, SFEEntityIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

Get an ordered array of SFEEntityIdentificationCandidate from probe's template best matches with templates in the gallery. Entity is then used to group the results by entity using the best score of any template associated with the entity.

- SFEError sfePalmLivenessPassive (SFESolver solver, SFEImageView image, const SFEDetection *detection, float *out_liveness_score)

  Passive liveness score calculation.

9.37.1 Detailed Description

File containing API of sfe_palm library as part of SFE Toolkit.

Author

Innovatrics - SmartFace Embedded

Date

28.10.2024

Definition in file sfe_palm.h.

9.37.2 Macro Definition Documentation

9.37.2.1 SFE_PALM_TEMPLATE_SIZE

```
#define SFE_PALM_TEMPLATE_SIZE 522
```

Palm template size in bytes.
Definition at line 19 of file sfe_palm.h.

9.37.3 Typedef Documentation

9.37.3.1 SFEPalmAttributes

```
typedef struct SFEPalmAttributes SFEPalmAttributes
```

9.37.3.2 SFEPalmTemplate

```
typedef struct SFEPalmTemplate SFEPalmTemplate
```

Palm template struct.

9.37.3.3 SFEPalmTemplateVersion

```
typedef struct SFEPalmTemplateVersion SFEPalmTemplateVersion
```

Palm template version struct.

9.37.4 Function Documentation

9.37.4.1 sfePalmAttributes()

```
SFEError sfePalmAttributes (
            SFESolver solver,
            SFEImageView image,
            const SFEPalmLandmarks * landmarks,
            SFEPalmAttributes * out_attributes )
```

Calculate palm image quality attributes from given source image and palm detection.

Parameters

in	*solver*	attributes solver
in	*image*	source image
in	*landmarks*	palm landmarks
out	*out_attributes*	structure containing hand_orientation, hand_position, quality

Returns

Error in case of failed attributes calculation

Examples

example_palm_attributes.cpp.

9.37.4.2 sfePalmEntityIdentify()

```
SFEError sfePalmEntityIdentify (
            const SFEPalmTemplate * probe_palm_template,
            const SFEPalmTemplate * templates_gallery,
            const SFEEntity * entities_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFEEntityIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFEEntityIdentificationCandidate from probe's template best matches with templates in the gallery. Entity is then used to group the results by entity using the best score of any template associated with the entity.

Parameters

in	*probe_palm_template*	probe template
in	*templates_gallery*	pointer to an array of SFEPalmTemplate
in	*entities_gallery*	pointer to an array of SFEEntity that corresponds with templates_gallery
in	*gallery_size*	number of templates in gallery
in	*matching_score_threshold*	threshold for matching - range <0,1>
out	*out_candidates*	Pointer to an ordered array of candidates
in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the matching score threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification.

9.37.4.3 sfePalmLandmarks()

```
SFEError sfePalmLandmarks (
            SFESolver solver,
            SFEImageView image,
            const SFEDetection * detection,
            SFEPalmLandmarks * out_landmarks )
```

Calculate palm landmarks from given source image and palm detection.

Parameters

in	*solver*	landmarks solver
in	*image*	source image
in	*detection*	core detection containing the detected palm
out	*out_landmarks*	structure containing hand_position, hand_orientation, confidence, and keypoints

Returns

Error in case of failed landmarks calculation

Examples

example_palm_attributes.cpp, and example_palm_identify.cpp.

9.37.4.4 sfePalmLivenessPassive()

```
SFEError sfePalmLivenessPassive (
            SFESolver solver,
            SFEImageView image,
            const SFEDetection * detection,
            float * out_liveness_score )
```

Passive liveness score calculation.

Parameters

in	*solver*	passive liveness solver
in	*image*	source image
in	*detection*	core detection containing the detected palm
out	*out_liveness_score*	OUT: normalized score of passive liveness

Returns

Error in case of failed passive liveness score calculation

Examples

example_palm_liveness.cpp.

9.37.4.5 sfePalmTemplateExtract()

```
SFEError sfePalmTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            const SFEPalmLandmarks * landmarks,
            SFEPalmTemplate * out_palm_template )
```

Extract template from source image and given palm attributes.

Parameters

in	*solver*	palm extraction solver
in	*image*	source image
in	*landmarks*	palm landmarks
out	*out_palm_template*	extracted palm template

Returns

Error in case of failed extraction.

Examples

example_palm_identify.cpp.

9.37.4.6 sfePalmTemplateIdentify()

```
SFEError sfePalmTemplateIdentify (
```

```
            const SFEPalmTemplate * probe_palm_template,
            const SFEPalmTemplate * templates_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFETemplateIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFETemplateIdentificationCandidate from probe's template best matches with templates in the gallery.

Parameters

in	*probe_palm_template*	probe template
in	*templates_gallery*	pointer to an array of SFPalmTemplate
in	*gallery_size*	number of templates in the gallery
in	*matching_score_threshold*	threshold for matching - range <0,1>
out	*out_candidates*	Pointer to an ordered array of candidates
in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the matching score threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification.

Examples

example_palm_identify.cpp.

9.37.4.7 sfePalmTemplateMatch()

```
SFEError sfePalmTemplateMatch (
            const SFEPalmTemplate * template1,
            const SFEPalmTemplate * template2,
            float * out_matching_score )
```

Match two palm templates.

Parameters

in	*template1*	first template to match
in	*template2*	second template to match
out	*out_matching_score*	matching score - range <0,1>

Returns

Error in case of failed matching.

Examples

example_palm_identify.cpp.

9.37.4.8 sfePalmTemplateVersion()

```
SFEError sfePalmTemplateVersion (
            const SFEPalmTemplate * palm_template,
            SFEPalmTemplateVersion * out_palm_template_version )
```

Get palm template version.

Parameters

in	*palm_template*	source palm template
out	*out_palm_template_version*	palm template version struct

Returns

Error in case of template checksum mismatch.

Examples

example_palm_identify.cpp.

9.38 sfe_palm.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include "sfe_core.h"
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
00015
00016 // PALM TEMPLATE EXTRACTION AND MATCHING
00017
00019 #define SFE_PALM_TEMPLATE_SIZE 522
00020
00022 typedef struct SFEPalmTemplate {
00023   uint8_t data[SFE_PALM_TEMPLATE_SIZE];
00024 } SFEPalmTemplate;
00025
00027 typedef struct SFEPalmTemplateVersion {
00029   uint8_t major;
00031   uint8_t minor;
00033   uint8_t patch;
00034 } SFEPalmTemplateVersion;
00035
00036 /* SFEPalmLandmarks is defined in sfe_core_detection.h (included via sfe_core.h). */
00037
00045 SFEError sfePalmLandmarks(SFESolver solver, SFEImageView image,
00046                           const SFEDetection *detection,
00047                           SFEPalmLandmarks *out_landmarks);
00048
00049 typedef struct SFEPalmAttributes {
00052   float hand_position;
00056   float hand_orientation;
00058   float quality;
00059 } SFEPalmAttributes;
00060
00069 SFEError sfePalmAttributes(SFESolver solver, SFEImageView image,
00070                            const SFEPalmLandmarks *landmarks,
00071                            SFEPalmAttributes *out_attributes);
00072
00073
00074
00081 SFEError sfePalmTemplateExtract(SFESolver solver, SFEImageView image,
00082                                 const SFEPalmLandmarks *landmarks,
00083                                 SFEPalmTemplate *out_palm_template);
00084
00090 SFEError sfePalmTemplateMatch(const SFEPalmTemplate *template1,
00091                               const SFEPalmTemplate *template2,
00092                               float *out_matching_score);
00093
00098 SFEError
00099 sfePalmTemplateVersion(const SFEPalmTemplate *palm_template,
00100                        SFEPalmTemplateVersion *out_palm_template_version);
00101
00115 SFEError
00116 sfePalmTemplateIdentify(const SFEPalmTemplate *probe_palm_template,
00117                         const SFEPalmTemplate *templates_gallery,
00118                         size_t gallery_size, float matching_score_threshold,
00119                         SFETemplateIdentificationCandidate *out_candidates,
00120                         size_t *in_out_candidates_count, size_t thread_count);
00121
00139 SFEError sfePalmEntityIdentify(const SFEPalmTemplate *probe_palm_template,
00140                                const SFEPalmTemplate *templates_gallery,
00141                                const SFEEntity *entities_gallery,
```

```
00142                                size_t gallery_size,
00143                                float matching_score_threshold,
00144                                SFEEntityIdentificationCandidate *out_candidates,
00145                                size_t *in_out_candidates_count,
00146                                size_t thread_count);
00147
00154 SFEError sfePalmLivenessPassive(SFESolver solver, SFEImageView image,
00155                                 const SFEDetection *detection,
00156                                 float *out_liveness_score);
00157
00158 #ifdef __cplusplus
00159 } // extern "C"
00160 #endif
```

9.39 D:/glab/1512fd1724a/1/include/sfe_tattoo.h File Reference

File containing API of sfe_tattoo library as part of SFE Toolkit.

`#include ¨sfe_core.h¨`

Data Structures

- struct SFETattooTemplate

  Tattoo template struct.
- struct SFETattooTemplateVersion

  Tattoo template version struct.

Macros

- #define SFE_TATTOO_TEMPLATE_SIZE 522

  Tattoo template size in bytes.

Typedefs

- typedef struct SFETattooTemplate SFETattooTemplate

  Tattoo template struct.
- typedef struct SFETattooTemplateVersion SFETattooTemplateVersion

  Tattoo template version struct.

Functions

- SFEError sfeTattooTemplateExtract (SFESolver solver, SFEImageView image, const SFEDetection *detection, SFETattooTemplate *out_tattoo_template)

  Extract template from source image and given tattoo detection.
- SFEError sfeTattooTemplateMatch (const SFETattooTemplate *template1, const SFETattooTemplate *template2, float *out_matching_score)

  Match two tattoo templates.
- SFEError sfeTattooTemplateVersion (const SFETattooTemplate *tattoo_template, SFETattooTemplateVersion *out_tattoo_template_version)

  Get tattoo template version.
- SFEError sfeTattooTemplateIdentify (const SFETattooTemplate *probe_tattoo_template, const SFETattooTemplate *templates_gallery, size_t gallery_size, float matching_score_threshold, SFETemplateIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

  Get an ordered array of SFETemplateIdentificationCandidate from probe's template best matches with templates in the gallery.
- SFEError sfeTattooEntityIdentify (const SFETattooTemplate *probe_tattoo_template, const SFETattooTemplate *templates_gallery, const SFEEntity *entities_gallery, size_t gallery_size, float matching_score_threshold, SFEEntityIdentificationCandidate *out_candidates, size_t *in_out_candidates_count, size_t thread_count)

Get an ordered array of SFEEntityIdentificationCandidate from probe's template best matches with templates in the gallery. Entity is then used to group the results by entity using the best score of any template associated with the entity.

- SFEError sfeTattooTemplateExport (const SFETattooTemplate *tattoo_template, uint8_t *bytes, size_t *size)

  Export tattoo template to bytes that can be shared between Innovatrics components. This is version v1 of the template format.

- SFEError sfeTattooTemplateImport (const uint8_t *bytes, size_t size, SFETattooTemplate *out_tattoo_↩ template)

  Import tattoo template from bytes that can be shared between Innovatrics components.

9.39.1 Detailed Description

File containing API of sfe_tattoo library as part of SFE Toolkit.

Author

Innovatrics - SmartFace Embedded

Date

28.10.2024

Definition in file sfe_tattoo.h.

9.39.2 Macro Definition Documentation

9.39.2.1 SFE_TATTOO_TEMPLATE_SIZE

```
#define SFE_TATTOO_TEMPLATE_SIZE 522
```

Tattoo template size in bytes.
Definition at line 19 of file sfe_tattoo.h.

9.39.3 Typedef Documentation

9.39.3.1 SFETattooTemplate

```
typedef struct SFETattooTemplate SFETattooTemplate
```

Tattoo template struct.

9.39.3.2 SFETattooTemplateVersion

```
typedef struct SFETattooTemplateVersion SFETattooTemplateVersion
```

Tattoo template version struct.

9.39.4 Function Documentation

9.39.4.1 sfeTattooEntityIdentify()

```
SFEError sfeTattooEntityIdentify (
            const SFETattooTemplate * probe_tattoo_template,
            const SFETattooTemplate * templates_gallery,
            const SFEEntity * entities_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFEEntityIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFEEntityIdentificationCandidate from probe's template best matches with templates in the gallery. Entity is then used to group the results by entity using the best score of any template associated with the entity.

Parameters

in	*probe_tattoo_template*	probe template
in	*templates_gallery*	pointer to an array of SFETattooTemplate
in	*entities_gallery*	pointer to an array of SFEEntity that corresponds with templates_gallery
in	*gallery_size*	number of templates in gallery
in	*matching_score_threshold*	threshold for matching - range <0,1>
out	*out_candidates*	Pointer to an ordered array of candidates
in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the matching score threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification.

9.39.4.2 sfeTattooTemplateExport()

```
SFEError sfeTattooTemplateExport (
            const SFETattooTemplate * tattoo_template,
            uint8_t * bytes,
            size_t * size )
```

Export tattoo template to bytes that can be shared between Innovatrics components. This is version v1 of the template format.

Parameters

in	*tattoo_template*	tattoo template to export
in,out	*bytes*	bytes to export the tattoo template to
in,out	*size*	size of the exported bytes

Returns

Error in case of failed export.

9.39.4.3 sfeTattooTemplateExtract()

```
SFEError sfeTattooTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            const SFEDetection * detection,
            SFETattooTemplate * out_tattoo_template )
```

Extract template from source image and given tattoo detection.

Parameters

in	*solver*	tattoo extraction solver
in	*image*	source image
in	*detection*	core detection containing the detected tattoo
out	*out_tattoo_template*	extracted tattoo template

Returns

Error in case of failed extraction.

9.39.4.4 sfeTattooTemplateIdentify()

```
SFEError sfeTattooTemplateIdentify (
            const SFETattooTemplate * probe_tattoo_template,
            const SFETattooTemplate * templates_gallery,
            size_t gallery_size,
            float matching_score_threshold,
            SFETemplateIdentificationCandidate * out_candidates,
            size_t * in_out_candidates_count,
            size_t thread_count )
```

Get an ordered array of SFETemplateIdentificationCandidate from probe's template best matches with templates in the gallery.

Parameters

in	*probe_tattoo_template*	probe template
in	*templates_gallery*	pointer to an array of SFETattooTemplate
in	*gallery_size*	number of templates in the gallery
in	*matching_score_threshold*	threshold for matching - range <0,1>
out	*out_candidates*	Pointer to an ordered array of candidates
in,out	*in_out_candidates_count*	IN: Expected number of candidates OUT: Number of found candidates with matching score above the matching score threshold
in	*thread_count*	Number of threads to use in the identification - 0: use all available cores, 1: use single thread, N: use N number of threads

Returns

Error in case of failed identification.

9.39.4.5 sfeTattooTemplateImport()

```
SFEError sfeTattooTemplateImport (
            const uint8_t * bytes,
            size_t size,
            SFETattooTemplate * out_tattoo_template )
```

Import tattoo template from bytes that can be shared between Innovatrics components.

Parameters

in	*bytes*	bytes to import the tattoo template from
in	*size*	size of the imported bytes
out	*out_tattoo_template*	imported tattoo template

Returns

Error in case of failed import.

9.39.4.6 sfeTattooTemplateMatch()

```
SFEError sfeTattooTemplateMatch (
            const SFETattooTemplate * template1,
```

```
            const SFETattooTemplate * template2,
            float * out_matching_score )
```

Match two tattoo templates.

Parameters

in	*template1*	first template to match
in	*template2*	second template to match
out	*out_matching_score*	matching score - range <0,1>

Returns

Error in case of failed matching.

9.39.4.7 sfeTattooTemplateVersion()

```
SFEError sfeTattooTemplateVersion (
            const SFETattooTemplate * tattoo_template,
            SFETattooTemplateVersion * out_tattoo_template_version )
```

Get tattoo template version.

Parameters

in	*tattoo_template*	source tattoo template
out	*out_tattoo_template_version*	tattoo template version struct

Returns

Error in case of template checksum mismatch.

9.40 sfe_tattoo.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include "sfe_core.h"
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
00015
00016 // TATTOO TEMPLATE EXTRACTION AND MATCHING
00017
00019 #define SFE_TATTOO_TEMPLATE_SIZE 522
00020
00022 typedef struct SFETattooTemplate {
00023   uint8_t data[SFE_TATTOO_TEMPLATE_SIZE];
00024 } SFETattooTemplate;
00025
00027 typedef struct SFETattooTemplateVersion {
00029   uint32_t version;
00030 } SFETattooTemplateVersion;
00031
00038 SFEError sfeTattooTemplateExtract(SFESolver solver, SFEImageView image,
00039                                   const SFEDetection *detection,
00040                                   SFETattooTemplate *out_tattoo_template);
00041
00047 SFEError sfeTattooTemplateMatch(const SFETattooTemplate *template1,
00048                                 const SFETattooTemplate *template2,
00049                                 float *out_matching_score);
00050
00055 SFEError
00056 sfeTattooTemplateVersion(const SFETattooTemplate *tattoo_template,
00057                          SFETattooTemplateVersion *out_tattoo_template_version);
00058
00072 SFEError
```

```
00073 sfeTattooTemplateIdentify(const SFETattooTemplate *probe_tattoo_template,
00074                           const SFETattooTemplate *templates_gallery,
00075                           size_t gallery_size, float matching_score_threshold,
00076                           SFETemplateIdentificationCandidate *out_candidates,
00077                           size_t *in_out_candidates_count, size_t thread_count);
00078
00096 SFEError sfeTattooEntityIdentify(const SFETattooTemplate *probe_tattoo_template,
00097                                  const SFETattooTemplate *templates_gallery,
00098                                  const SFEEntity *entities_gallery,
00099                                  size_t gallery_size,
00100                                  float matching_score_threshold,
00101                                  SFEEntityIdentificationCandidate *out_candidates,
00102                                  size_t *in_out_candidates_count,
00103                                  size_t thread_count);
00104
00111 SFEError sfeTattooTemplateExport(const SFETattooTemplate *tattoo_template,
00112                                  uint8_t *bytes,
00113                                  size_t *size);
00114
00120 SFEError sfeTattooTemplateImport(const uint8_t *bytes,
00121                                  size_t size,
00122                                  SFETattooTemplate *out_tattoo_template);
00123
00124 #ifdef __cplusplus
00125 } // extern "C"
00126 #endif
```

9.41 D:/glab/1512fd1724a/1/readme.md File Reference

Chapter 10

Examples

10.1 example_face_identify.cpp

Example of use of sfe_face library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_face.h"
#include <regex>
#include <vector>

#include <iomanip>
#include <sstream>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_gallery = "./assets/face/entities/";

std::string image_probe = "assets/face/images/obiwan0.png";

std::string solver_face_detect = SOLVER_FACE_DETECT;
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
std::string solver_face_template = SOLVER_FACE_EXTRACTION;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

size_t face_size_min = 28;
size_t face_size_max = 170;

float detection_threshold = 0.1f;
float identification_threshold = 0.6f;

void getRecommendedImageSize(const std::string &solver_face_detect,
                             SFEImage &image, const size_t face_size_min,
                             const size_t face_size_max,
                             size_t &recommended_width,
                             size_t &recommended_height) {

  // If the face detection solver requires static input (filename contains width
  // and height), we need to prepare the input image accordingly Typically the
  // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
  // This can be hardcoded into the application, or we can parse the width and
  // height from the solver filename

  // Try to use regex capture to extract width and height from solver name
  std::smatch width_height;
  if (std::regex_match(solver_face_detect, width_height,
                       std::regex(".*w([0-9]+)h([0-9]+).*"))) {
    recommended_width = std::stoi(width_height[1]);
    recommended_height = std::stoi(width_height[2]);
  } else {
    // Solvers without width and height in name can accept dynamic input
    // image size. For best results we recommend to scale the input image to
    // a resolution recommended by the sfeFaceDetectInputSize function

    // Use sfeFaceDetectInputSize to get recommended input image dimensions
    // for given min_face_size/max_face_size
    SFEDetectionInputSize input_size{};
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &input_size);
```

```
    utils::checkError(error);
    recommended_width = input_size.width;
    recommended_height = input_size.height;
  };
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-g: Gallery folder." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-l: Path to landmarks solver." « std::endl;
  std::cout « "-e: Path to extraction solver." « std::endl;
  std::cout « "-t: Detection threshold. <0,1>" « std::endl;
  std::cout « "-i: Identification threshold. <0,1>" « std::endl;
  std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
  std::cout « "-x: Max face size in pixels to detect." « std::endl;
}

void detectFace(SFEImage &image, SFESolver &detector_solver,
                SFEDetection &detected_face) {

  SFEImage resized_image{};
  DEFER(sfeImageFree(resized_image));
  SFEError error{};
  { // STAGE 1 Load image
    size_t recommended_width{};
    size_t recommended_height{};

    // Calculate optimal input image size for face detection. Image will be
    // resized to recommended size for optimal performance.
    getRecommendedImageSize(solver_face_detect, image, face_size_min,
                            face_size_max, recommended_width,
                            recommended_height);

    // Resize image to recommended size
    // NOTE: This function will resize the image without preserving the aspect
    // ratio of the image.

    error = sfeImageResize(image, recommended_width, recommended_height,
                           &resized_image);
    utils::checkError(error);
  }

  size_t detection_count = 1;
  { // STAGE 2: Detect face in the image

    // Detect face in the image
    error = sfeDetect(detector_solver, resized_image, detection_threshold,
                          &detected_face, &detection_count);
    utils::checkError(error);

    if (detection_count == 0) {
      std::ostringstream oss;
      oss « "Error: No face detected in the image ";
      throw std::runtime_error(oss.str());

    } else if (detection_count > 1) {
      std::ostringstream oss;
      oss « "Error: Found " « detection_count
          « " faces. Please provide an image with only one face.";

      throw std::runtime_error(oss.str());
    }

    std::cout « "Detected face in the image." « std::endl;
  }
}

int main(int argc, char *argv[]) {

  { // STAGE 0: Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
```

```
        args[arg] = argv[++i];
      } else {
        std::cerr « "Option " « arg « " requires a value." « std::endl;
        return 1;
      }
    } else {
      std::cerr « "Unknown option: " « arg « std::endl;
      return 1;
    }
  }

  // Assign values to variables based on parsed arguments
  if (args.count("-p"))
    image_probe = args["-p"];
  if (args.count("-g"))
    image_gallery = args["-g"];
  if (args.count("-d"))
    solver_face_detect = args["-d"];
  if (args.count("-l"))
    solver_face_landmarks = args["-l"];
  if (args.count("-e"))
    solver_face_template = args["-e"];
  if (args.count("-t"))
    detection_threshold = std::stof(args["-t"]);
  if (args.count("-i"))
    identification_threshold = std::stof(args["-i"]);
  if (args.count("-m"))
    face_size_min = std::stoi(args["-m"]);
  if (args.count("-x"))
    face_size_max = std::stoi(args["-x"]);

  utils::printFormatted("EXAMPLE PARAMETERS");
  // Print resource names
  std::cout « "Probe image: " « image_probe « std::endl;
  std::cout « "Gallery image: " « image_gallery « std::endl;

  // Print solver names
  std::cout « "Face detection solver: " « solver_face_detect « std::endl;
  std::cout « "Face landmarks solver: " « solver_face_landmarks
            « std::endl;
  std::cout « "Face template solver: " « solver_face_template « std::endl;

  // Print other options
  std::cout « "Min face size [px]: " « face_size_min « std::endl;
  std::cout « "Max face size [px]: " « face_size_max « std::endl;
  std::cout « "Detection threshold: " « detection_threshold « std::endl;
  std::cout « "Identification threshold: " « identification_threshold
            « std::endl;
}

utils::printToolkitInfo();

SFEError error{};

SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
SFESolver landmarks_solver{};
DEFER(sfeSolverFree(landmarks_solver));
SFESolver template_solver{};
DEFER(sfeSolverFree(template_solver));
{ // STAGE 1.1: loading solvers
  utils::printFormatted("LOADING SOLVERS");
  // Initialize face detection solver
  error = sfeSolverCreate(
      solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  // Initialize landmarks detection solver
  error = sfeSolverCreate(
      solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);

  // Initialize template extraction solver
  error = sfeSolverCreate(
      solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &template_solver);
  utils::checkError(error);
}

SFEFaceTemplate probe_face_template;
{ // STAGE 1: Extract probe face template
  utils::printFormatted("PROBE TEMPLATE EXTRACTION");

  SFEImage image{};
  DEFER(sfeImageFree(image));
```

```
  { // STAGE 1.1: Load image
    // Load image data from file
    auto image_data = utils::readFile(image_probe);

    // Decode image from data
    error = sfeImageDecode(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
  }

  SFEDetection detected_face = {};
  { // STAGE 1.2: Detect face in the image
    // Detect a face in the image
    detectFace(image, detector_solver, detected_face);
  }

  std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
  { // STAGE 1.3: Get face landmarks
    // Use landmarks detection solver to get relevant landmarks for
    // the detected face
    error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
                             landmarks.data());
    utils::checkError(error);
  }

  { // STAGE 1.4: Extract probe face template
    // Use template extraction solver to create new face
    // template
    error = sfeFaceTemplateExtract(template_solver, image, &detected_face,
                                   landmarks.data(), &probe_face_template);
    utils::checkError(error);
  }

  { // STAGE 1.5: (optional) Print template attributes
    utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");

    SFEFaceTemplateVersion version = {};
    error = sfeFaceTemplateVersion(&probe_face_template, &version);
    utils::checkError(error);

    auto version_minor = (int)version.version_minor - (int)'0';
    std::cout << "Version of the probe template: " << version.version_major
              << '.' << version_minor << std::endl;
  }

  { // STAGE 1.6: (optional) Export and re-import template
    std::vector<uint8_t> buf(1024);
    size_t size = buf.size();
    error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
    if (error && size > buf.size()) {
      buf.resize(size);
      size = buf.size();
      error = sfeFaceTemplateExport(&probe_face_template, buf.data(), &size);
    }
    utils::checkError(error);

    SFEFaceTemplate imported_template = {};
    error = sfeFaceTemplateImport(buf.data(), size, &imported_template);
    utils::checkError(error);

    float roundtrip_score = 0.f;
    error = sfeFaceTemplateMatch(&probe_face_template, &imported_template,
                                 &roundtrip_score);
    utils::checkError(error);
    std::cout << "Export/import round-trip match score: " << roundtrip_score
              << std::endl;
  }
}

std::vector<std::string> image_paths{};
{ // STAGE 2: Get gallery image paths
  // Get folder names from the face image gallery
  auto folder_names = utils::getFiles(image_gallery);

  for (auto folder_name : folder_names) {
    auto gallery_folder = image_gallery + folder_name + "/";

    // Get image names from folder
    auto image_names = utils::getFiles(gallery_folder);

    // Extract template for each image in the folder
    for (auto image_name : image_names) {

      auto image_path = gallery_folder + image_name;

      image_paths.push_back(image_path);
    }
  }
```

```
}

std::vector<SFEFaceTemplate> gallery_face_templates(image_paths.size());
std::vector<SFEDetection> detected_faces(image_paths.size());
std::vector<std::array<SFEFaceLandmarks, SFE_FACE_LANDMARK_COUNT» landmarks(
    image_paths.size());
{ // STAGE 2: Load gallery image and extract face templates for each image in
  // the gallery
  utils::printFormatted("GALLERY TEMPLATE EXTRACTION");

  for (size_t i = 0; i < image_paths.size(); i++) {
    std::cout « "Processing image #" « i « ", path: " « image_paths[i]
             « std::endl;
    SFEImage image{};
    DEFER(sfeImageFree(image));
    { // STAGE 2.1: Load image
      // Load image data from file
      auto image_data = utils::readFile(image_paths[i]);

      // Decode image from data
      error = sfeImageDecode(image_data.data(), image_data.size(), &image);
      utils::checkError(error);
    }

    { // STAGE 2.2: Detect face in the image
      // Detect a face in the image
      detectFace(image, detector_solver, detected_faces[i]);
    }

    { // STAGE 2.3: Get face landmarks
      // Use landmarks detection solver to get relevant landmarks for
      // the detected face
      error = sfeFaceLandmarks(landmarks_solver, image, &detected_faces[i],
                               landmarks[i].data());
      utils::checkError(error);
    }

    { // STAGE 2.3: Extract face template
      // Use template extraction solver to get face_template solver
      error = sfeFaceTemplateExtract(
          template_solver, image, &detected_faces[i], landmarks[i].data(),
          &gallery_face_templates[i]);
      utils::checkError(error);
      std::cout « "Extracted face template." « std::endl;
    }
    std::cout « std::endl;
  }
}

size_t candidate_count = 1;
int best_candidate_index = -1;
std::vector<SFETemplateIdentificationCandidate> identification_results(
    image_paths.size());
{ // STAGE 3: Identification
  utils::printFormatted("1:N IDENTIFICATION");

  error = sfeFaceTemplateIdentify(
      &probe_face_template, gallery_face_templates.data(),
      gallery_face_templates.size(), identification_threshold,
      identification_results.data(), &candidate_count, 4);
  utils::checkError(error);

  // Resize the results vector to the actual number of candidates found, in
  // the case there is less than candidate_count
  identification_results.resize(candidate_count);

  std::cout « "Found " « identification_results.size()
           « " candidates above identification threshold "
           « identification_threshold « std::endl;
  for (auto &result : identification_results)
    std::cout « "Template index: #" « result.index
             « ", score: " « result.score « std::endl;

  // Since it's sorted vector by score, the best candidate is the first one
  best_candidate_index = identification_results[0].index;
}

{ // STAGE 4: (optional) 1:1 matching with top candidate to showcase 1:1
  // matching
  utils::printFormatted("1:1 MATCHING WITH TOP CANDIDATE");

  if (identification_results.size() == 0) {
    std::cout « "No candidates found above identification threshold "
             « identification_threshold « std::endl;
    return 0;
```

```
    }

    auto best_candidate_template = gallery_face_templates[best_candidate_index];

    float match_confidence;
    error = sfeFaceTemplateMatch(&probe_face_template, &best_candidate_template,
                                 &match_confidence);
    utils::checkError(error);

    std::cout « "Matching score of probe template with template index #"
              « best_candidate_index « ", score: " « match_confidence
              « std::endl;
  }

  { // STAGE: 5 Optional, get face crop of best face and its bounding box and
    // save it to file
    utils::printFormatted("SAVE FACE CROP AND ANNOTATED IMAGE");

    auto best_face_index = identification_results[0].index;

    SFEImage image{};
    DEFER(sfeImageFree(image));
    { // STAGE 5.1: Load image
      // Load image data from file
      auto image_data = utils::readFile(image_paths[best_face_index]);

      // Decode image from data
      error = sfeImageDecode(image_data.data(), image_data.size(), &image);
      utils::checkError(error);
    }

    SFEFaceCrop crop_data = {};
    DEFER(sfeImageFree(crop_data.crop_image));
    { // STAGE 5.2: Get face crop
      //  Defines the size of the crop as an extension of
      // the detection bounding box.
      float face_size_extension = 2.0f;
      SFEError error =
          sfeFaceCrop(image, landmarks[best_face_index].data(),
                      face_size_extension, (float)(face_size_max), &crop_data);
      utils::checkError(error);

      std::cout « "Face crop extension: " « crop_data.face_size_extension
                « ", width of cropped image: " « crop_data.crop_image.width
                « "px, height of cropped image: "
                « crop_data.crop_image.height « "px." « std::endl;
    }

    { // STAGE 5.3: Save face crop to file
      auto png_file = std::vector<unsigned char>();
      size_t size =
          crop_data.crop_image.width * crop_data.crop_image.height * 3;
      png_file.resize(size);
      SFEError error2 = sfeImageEncode(crop_data.crop_image, SFE_IMAGE_FORMAT_PNG,
                                       png_file.data(), &size);
      utils::checkError(error2);

      std::cout « "Face crop saved as crop_identified_person.png" « std::endl;
      utils::saveFile("crop_identified_person.png", png_file);
    }

    { // STAGE 5.4: Annotate the image
      // Render bounding box with detection confidence and identification score
      std::stringstream label;
      label « " D:" « std::fixed « std::setprecision(2)
            « detected_faces[best_face_index].confidence
            « " M:" « identification_results[0].score;

      auto color = annotate::RED;

      annotate::labelBox(image, label.str(),
                         detected_faces[best_candidate_index].bounding_box,
                         annotate::WHITE, color);

      // Render landmarks
      for (auto &landmark : landmarks[best_face_index]) {
        auto x = landmark.x * image.width;
        auto y = landmark.y * image.height;
        annotate::circle(image, x, y, 3, annotate::YELLOW);
      }

      // Save annotated image
      size_t size = image.width * image.height * 3;
      auto png_file = std::vector<unsigned char>(size);
      error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
      utils::checkError(error);
      png_file.resize(size);
```

```
      utils::saveFile("face_identify.png", png_file);

      std::cout « std::endl;
      std::cout « "Annotated image saved as face_identify.png" « std::endl;
    }
  }

  utils::printFormatted("FINISHED");
}
```

10.2 example_face_identify_entity.cpp

Example of use of sfe_face library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_face.h"

#include <regex>
#include <vector>

#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string gallery = "./assets/face/entities/";

std::string image_probe = "assets/face/images/obiwan0.png";

std::string solver_face_detect = SOLVER_FACE_DETECT;
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
std::string solver_face_template = SOLVER_FACE_EXTRACTION;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

size_t face_size_min = 28;
size_t face_size_max = 170;

float detection_threshold = 0.1f;
float identification_threshold = 0.6f;

void getRecommendedImageSize(const std::string &solver_face_detect,
                             SFEImage &image, const size_t face_size_min,
                             const size_t face_size_max,
                             size_t &recommended_width,
                             size_t &recommended_height) {

  // If the face detection solver requires static input (filename contains width and height),
  // we need to prepare the input image accordingly Typically the format is:
  // face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver
  // NOTE: This can
  // be hardcoded into the application, or we can parse the width and height
  // from the solver filename

  // Try to use regex capture to extract width and height from solver name
  std::smatch width_height;
  if (std::regex_match(solver_face_detect, width_height,
                       std::regex(".*w([0-9]+)h([0-9]+).*"))) {
    recommended_width = std::stoi(width_height[1]);
    recommended_height = std::stoi(width_height[2]);
  } else {
    // Solvers without width and height in name can accept dynamic input
    // image size. For best results we recommend to scale the input image to
    // a resolution recommended by the sfeFaceDetectInputSize function

    // Use sfeFaceDetectInputSize to get recommended input image dimensions
    // for given min_face_size/max_face_size
    SFEDetectionInputSize input_size{};
    SFEError error = sfeFaceDetectInputSize(image,
      SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, face_size_min,
                                            face_size_max, &input_size);
    utils::checkError(error);
    recommended_width = input_size.width;
    recommended_height = input_size.height;
  };
}

SFEFaceTemplate extractTemplate(std::string image_path,
                                SFESolver &detector_solver,
                                SFESolver &landmarks_solver,
                                SFESolver &template_solver) {

  SFEImage image{};
  DEFER(sfeImageFree(image));
  SFEImage resized_image{};
```

```
  DEFER(sfeImageFree(resized_image));
  SFEError error{};
  { // STAGE 1 Load image
    // Load image data from file
    auto image_data = utils::readFile(image_path);

    // Decode image from data
    error = sfeImageDecode(image_data.data(), image_data.size(), &image);
    utils::checkError(error);

    size_t recommended_width{};
    size_t recommended_height{};

    // Calculate optimal input image size for face detection. Image will be
    // resized to recommended size for optimal performance.
    getRecommendedImageSize(solver_face_detect, image, face_size_min,
                            face_size_max, recommended_width,
                            recommended_height);

    // Resize image to recommended size
    // NOTE: This function will resize the image without preserving the aspect
    // ratio of the image.

    error = sfeImageResize(image, recommended_width, recommended_height,
                           &resized_image);
    utils::checkError(error);
  }

  SFEDetection detected_face = {};
  size_t detection_count = 1;
  { // STAGE 3: Detect face in the image

    // Detect face in the image
    error = sfeDetect(detector_solver, resized_image, detection_threshold,
                      &detected_face, &detection_count);
    utils::checkError(error);

    if (detection_count == 0) {
      throw std::runtime_error("No face detected in the image.");
    }
  }

  std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
  { // STAGE 4: Get face landmarks

    // Use landmarks detection solver to get relevant landmarks for
    // the detected face
    error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
                             landmarks.data());
    utils::checkError(error);
  }

  SFEFaceTemplate face_template = {};
  { // STAGE 5: Extract probe face template
    // Use template extraction solver to create new face
    // template
    error =
        sfeFaceTemplateExtract(template_solver, image, &detected_face,
                               landmarks.data(), &face_template);
    utils::checkError(error);
  }
  return face_template;
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-g: Path to folder with entities." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-l: Path to landmarks solver." « std::endl;
  std::cout « "-e: Path to extraction solver." « std::endl;
  std::cout « "-t: Detection threshold. <0,1>" « std::endl;
  std::cout « "-i: Identification threshold. <0,1>" « std::endl;
  std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
  std::cout « "-x: Max face size in pixels to detect." « std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE 0: Parse command line arguments

    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
```

```
  for (int i = 1; i < argc; ++i) {
    std::string arg = argv[i];
    if (arg[0] == '-') {
      if (arg == "-h") {
        printHelp();
        return 0;
      }
      // Check if there's a next argument and it isn't another option
      if (i + 1 < argc && argv[i + 1][0] != '-') {
        args[arg] = argv[++i];
      } else {
        std::cerr << "Option " << arg << " requires a value." << std::endl;
        return 1;
      }
    } else {
      std::cerr << "Unknown option: " << arg << std::endl;
      return 1;
    }
  }

  // Assign values to variables based on parsed arguments
  if (args.count("-p"))
    image_probe = args["-p"];
  if (args.count("-g"))
    gallery = args["-g"];
  if (args.count("-d"))
    solver_face_detect = args["-d"];
  if (args.count("-l"))
    solver_face_landmarks = args["-l"];
  if (args.count("-e"))
    solver_face_template = args["-e"];
  if (args.count("-t"))
    detection_threshold = std::stof(args["-t"]);
  if (args.count("-i"))
    identification_threshold = std::stof(args["-i"]);
  if (args.count("-m"))
    face_size_min = std::stoi(args["-m"]);
  if (args.count("-x"))
    face_size_max = std::stoi(args["-x"]);

  utils::printFormatted("EXAMPLE PARAMETERS");
  // Print resource names
  std::cout << "Probe image: " << image_probe << std::endl;
  std::cout << "Entities folder: " << gallery << std::endl;

  // Print solver names
  std::cout << "Face detection solver: " << solver_face_detect << std::endl;
  std::cout << "Face landmarks solver: " << solver_face_landmarks
            << std::endl;
  std::cout << "Face template solver: " << solver_face_template << std::endl;

  // Print other options
  std::cout << "Min face size [px]: " << face_size_min << std::endl;
  std::cout << "Max face size [px]: " << face_size_max << std::endl;
  std::cout << "Detection threshold: " << detection_threshold << std::endl;
  std::cout << "Identification threshold: " << identification_threshold
            << std::endl;
}

utils::printToolkitInfo();

SFEError error{};

SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
SFESolver landmarks_solver{};
DEFER(sfeSolverFree(landmarks_solver));
SFESolver template_solver{};
DEFER(sfeSolverFree(template_solver));
{ // STAGE 0: loading solvers
  // Initialize face detection solver
  error = sfeSolverCreate(
      solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  // Initialize landmarks detection solver
  error = sfeSolverCreate(
      solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);

  // Initialize template extraction solver
  error = sfeSolverCreate(
      solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &template_solver);
  utils::checkError(error);
```

```
}

SFEFaceTemplate probe_face_template;
{ // STAGE 1: Extract probe face template
  utils::printFormatted("PROBE TEMPLATE EXTRACTION");
  probe_face_template = extractTemplate(image_probe, detector_solver,
                                        landmarks_solver, template_solver);

  std::cout « "Probe face template extracted." « std::endl;
}

std::vector<SFEFaceTemplate> gallery_face_templates = {};
std::vector<SFEEntity> enitity_pairs = {};
{ // STAGE 2: Extract templates for each image in entity folder and associate them
  // with entity(UUID)

  // Entities(UUID) tells the ownership of the templates in the gallery.
  // For example:
  // UUID1 -> template1
  // UUID1 -> template2
  // UUID2 -> template3
  // UUID2 -> template4
  // ...
  // UUUIDN -> templateN

  utils::printFormatted("ENTITIES TEMPLATE EXTRACTION");

  // Get folder names from entities_gallery
  auto folder_names = utils::getFiles(gallery);

  for (auto folder_name : folder_names) {
    if (folder_name.find(".jpg") != std::string::npos ||
        folder_name.find(".png") != std::string::npos) {
      continue;
    }

    // Genereate UUID for each entity.
    auto entity = utils::generateEntity();

    auto entity_folder = gallery + folder_name + "/";

    std::cout « "Folder: " « entity_folder « ", entity UUID: ["
              « static_cast<int>(entity.uuid[0]) « ", "
              « static_cast<int>(entity.uuid[1]) « "..."
              « static_cast<int>(entity.uuid[15]) « "]" « std::endl;

    // Get image names from folder
    auto image_names = utils::getFiles(entity_folder);

    // Extract template for each image in the folder
    for (auto image_name : image_names) {

      auto image_path = entity_folder + image_name;

      auto face_template = extractTemplate(
          image_path, detector_solver, landmarks_solver, template_solver);

      gallery_face_templates.push_back(face_template);
      enitity_pairs.push_back(entity);
    }
  }
}

size_t candidate_count = 1;
std::vector<SFEEntityIdentificationCandidate> results(candidate_count);
int best_candidate_index = -1;
{ // STAGE 3: Identification with entities.
  // Probe template is matched against every template in the gallery.
  // Function returns entity(UUID) which owns the best matching template.

  utils::printFormatted("1:N IDENTIFICATION WITH ENTITIES");
  SFEError error = sfeFaceEntityIdentify(
      &probe_face_template, gallery_face_templates.data(),
      enitity_pairs.data(), gallery_face_templates.size(),
      identification_threshold, results.data(), &candidate_count, 4);
  utils::checkError(error);

  if (candidate_count == 0) {
    std::cout « "No candidates found. Exiting.." « std::endl;
    return 0;
  }

  std::cout « std::endl;
  std::cout « "Found " « results.size() « " candidates " « std::endl;
  for (int i = 0; i < results.size(); i++) {
    std::cout « "Index: " « i « ", Score: " « results[i].score
              « std::endl;
```

```
      std::cout « "Entity UUID: ["
                « static_cast<int>(results[i].entity.uuid[0]) « ", "
                « static_cast<int>(results[i].entity.uuid[1]) « "..."
                « static_cast<int>(results[i].entity.uuid[15]) « "]"
                « std::endl;
    }
  }

  utils::printFormatted("FINISHED");
}
```

10.3 example_face_track.cpp

Example of use of sfe_face library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_face.h"
#include <ostream>
#include <regex>
#include <vector>

#include <iomanip>
#include <sstream>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_probe = "assets/face/images/obiwan0.png";

std::string solver_face_detect = SOLVER_FACE_DETECT;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

size_t face_size_min = 28;
size_t face_size_max = 170;

float detection_threshold = 0.1f;

void getRecommendedImageSize(const std::string &solver_face_detect,
                             SFEImage &image, const size_t face_size_min,
                             const size_t face_size_max,
                             size_t &recommended_width,
                             size_t &recommended_height) {

  // If the face detection solver requires static input (filename contains width
  // and height), we need to prepare the input image accordingly Typically the
  // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
  // This can be hardcoded into the application, or we can parse the width and
  // height from the solver filename

  // Try to use regex capture to extract width and height from solver name
  std::smatch width_height;
  if (std::regex_match(solver_face_detect, width_height,
                       std::regex(".*w([0-9]+)h([0-9]+).*"))) {
    recommended_width = std::stoi(width_height[1]);
    recommended_height = std::stoi(width_height[2]);
  } else {
    // Solvers without width and height in name can accept dynamic input
    // image size. For best results we recommend to scale the input image to
    // a resolution recommended by the sfeFaceDetectInputSize function

    // Use sfeFaceDetectInputSize to get recommended input image dimensions
    // for given min_face_size/max_face_size
    SFEDetectionInputSize input_size{};
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &input_size);
    utils::checkError(error);
    recommended_width = input_size.width;
    recommended_height = input_size.height;
  };
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-t: Detection threshold. <0,1>" « std::endl;
  std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
```

```
  std::cout « "-x: Max face size in pixels to detect." « std::endl;
}

std::vector<SFEDetection> detectFaces(SFEImage &image,
                                      SFESolver &detector_solver) {

  SFEImage resized_image{};
  DEFER(sfeImageFree(resized_image));
  SFEError error{};
  { // STAGE 1 Load image
    size_t recommended_width{};
    size_t recommended_height{};

    // Calculate optimal input image size for face detection. Image will be
    // resized to recommended size for optimal performance.
    getRecommendedImageSize(solver_face_detect, image, face_size_min,
                            face_size_max, recommended_width,
                            recommended_height);

    // Resize image to recommended size
    // NOTE: This function will resize the image without preserving the aspect
    // ratio of the image.

    error = sfeImageResize(image, recommended_width, recommended_height,
                           &resized_image);
    utils::checkError(error);
  }

  size_t detection_count = 10;
  std::vector<SFEDetection> detected_faces(detection_count);
  { // STAGE 2: Detect face in the image

    // Detect faces in the image
    error = sfeDetect(detector_solver, resized_image, detection_threshold,
                          detected_faces.data(), &detection_count);
    utils::checkError(error);

    if (detection_count == 0) {
      std::ostringstream oss;
      oss « "Error: No face detected in the image ";
      throw std::runtime_error(oss.str());
    }
    detected_faces.resize(detection_count);

    std::cout « "Detected " « detection_count « " faces in the image."
              « std::endl;
  }
  return detected_faces;
}

std::ostream &operator«(std::ostream &os, const SFEEntity &entity) {
  for (size_t i = 0; i < 16; ++i) {
    os « std::hex « std::setw(2) « std::setfill('0')
       « static_cast<int>(entity.uuid[i]);
    if (i == 3 || i == 5 || i == 7 || i == 9) {
      os « '-';
    }
  }
  os « std::dec;
  return os;
}

std::ostream &operator«(std::ostream &os, const SFETrackedState &state) {
  switch (state) {
  case SFE_TRACKED_STATE_NEW:
    os « "NEW";
    break;
  case SFE_TRACKED_STATE_TRACKED:
    os « "TRACKED";
    break;
  case SFE_TRACKED_STATE_LOST:
    os « "LOST";
    break;
  case SFE_TRACKED_STATE_REMOVED:
    os « "REMOVED";
    break;
  default:
    os « "UNKNOWN";
    break;
  }
  return os;
}

std::ostream &operator«(std::ostream &os, const SFETracked &tracked) {
  os « "Face ID: " « tracked.id « " UUID: " « tracked.uuid
     « " State: " « tracked.state;
  return os;
```

```
}

template <typename T>
std::ostream &operator«(std::ostream &os, const std::vector<T> &vec) {
  for (auto &item : vec) {
    os « item « std::endl;
  }
  return os;
}

void frame(std::vector<SFEDetection> &detections, SFETracker tracker) {
  size_t reserved_size = 10;
  size_t tracked_faces_count = reserved_size;
  size_t lost_faces_count = reserved_size;
  size_t removed_faces_count = reserved_size;

  std::vector<SFETracked> tracked_faces(tracked_faces_count);
  std::vector<SFEEntity> lost_faces(lost_faces_count);
  std::vector<SFEEntity> removed_faces(removed_faces_count);

  // Update tracker with detected faces
  SFEError error =
      sfeDetectionTrackerUpdate(tracker, detections.data(), detections.size(),
                                tracked_faces.data(), &tracked_faces_count);
  utils::checkError(error);
  tracked_faces.resize(tracked_faces_count);

  // Get lost faces
  error = sfeDetectionTrackerLost(tracker, lost_faces.data(), &lost_faces_count);
  utils::checkError(error);
  lost_faces.resize(lost_faces_count);

  // Get removed faces
  error = sfeDetectionTrackerRemoved(tracker, removed_faces.data(),
                                     &removed_faces_count);
  utils::checkError(error);
  removed_faces.resize(removed_faces_count);

  // Print out frame results
  std::cout « "Tracked" « std::endl;
  std::cout « tracked_faces;
  std::cout « "Lost" « std::endl;
  std::cout « lost_faces;
  std::cout « "Removed" « std::endl;
  std::cout « removed_faces;
}

int main(int argc, char *argv[]) {

  { // STAGE 0: Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
        } else {
          std::cerr « "Option " « arg « " requires a value." « std::endl;
          return 1;
        }
      } else {
        std::cerr « "Unknown option: " « arg « std::endl;
        return 1;
      }
    }

    // Assign values to variables based on parsed arguments
    if (args.count("-p"))
      image_probe = args["-p"];
    if (args.count("-d"))
      solver_face_detect = args["-d"];
    if (args.count("-t"))
      detection_threshold = std::stof(args["-t"]);
    if (args.count("-m"))
      face_size_min = std::stoi(args["-m"]);
    if (args.count("-x"))
      face_size_max = std::stoi(args["-x"]);

    utils::printFormatted("EXAMPLE PARAMETERS");
```

```
    // Print resource names
    std::cout « "Probe image: " « image_probe « std::endl;

    // Print solver names
    std::cout « "Face detection solver: " « solver_face_detect « std::endl;

    // Print other options
    std::cout « "Min face size [px]: " « face_size_min « std::endl;
    std::cout « "Max face size [px]: " « face_size_max « std::endl;
    std::cout « "Detection threshold: " « detection_threshold « std::endl;
  }

  utils::printToolkitInfo();

  SFEError error{};

  SFESolver detector_solver{};
  DEFER(sfeSolverFree(detector_solver));
  { // STAGE 1: loading solvers
    utils::printFormatted("LOADING SOLVERS");
    // Initialize face detection solver
    error = sfeSolverCreate(
        solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &detector_solver);
    utils::checkError(error);
  }

  std::vector<SFEDetection> detected_faces;
  { // STAGE 2: Detect faces
    utils::printFormatted("DETECT FACES");

    SFEImage image{};
    DEFER(sfeImageFree(image));
    { // STAGE 2.1: Load image
      // Load image data from file
      auto image_data = utils::readFile(image_probe);

      // Decode image from data
      error = sfeImageDecode(image_data.data(), image_data.size(), &image);
      utils::checkError(error);
    }

    SFEDetection detected_face = {};
    { // STAGE 2.2: Detect face in the image
      // Detect a face in the image
      detected_faces = detectFaces(image, detector_solver);
    }
  }

  SFETracker tracker{};
  DEFER(sfeDetectionTrackerFree(tracker));
  { // STAGE 3: Track faces
    utils::printFormatted("TRACK FACES");
    error = sfeDetectionTrackerCreate(0.3f, 0.1f, 0.3f, 0.1f, 1, &tracker);
    utils::checkError(error);

    // We will be using the current detected faces and simulate detection across
    // 4 frames In a real application, you would call detectFaces() for each
    // frame and pass the detected faces to the tracker

    // First frame with detected faces, all should track as NEW
    utils::printFormatted("FRAME 1");
    frame(detected_faces, tracker);

    // Second frame with the same UUIDs, all should track as TRACKED
    utils::printFormatted("FRAME 2");
    frame(detected_faces, tracker);

    // Simulate a lost detection
    detected_faces.clear();

    // We should see the lost UUIDs
    utils::printFormatted("FRAME 3");
    frame(detected_faces, tracker);

    // We should see the removed UUIDs
    utils::printFormatted("FRAME 4");
    frame(detected_faces, tracker);
  }
  utils::printFormatted("FINISHED");
}
```

10.4 example_face_liveness.cpp

Example of use of sfe_face library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_face.h"
#include <iomanip>
#include <regex>
#include <sstream>
#include <vector>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_probe = "assets/face/images/obiwan0.png";

std::string solver_face_detect = SOLVER_FACE_DETECT;
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
std::string solver_face_liveness = SOLVER_FACE_LIVENESS;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

size_t face_size_min = 28;
size_t face_size_max = 170;

float detection_threshold = 0.1f;

void getRecommendedImageSize(const std::string &solver_face_detect,
                             SFEImage &image, const size_t face_size_min,
                             const size_t face_size_max,
                             size_t &recommended_width,
                             size_t &recommended_height) {

  // If the face detection solver requires static input (filename contains width
  // and height), we need to prepare the input image accordingly Typically the
  // format is: face_detect_accurate_mask_w1920h1080_op11.onnxrt.solver NOTE:
  // This can be hardcoded into the application, or we can parse the width and
  // height from the solver filename

  // Try to use regex capture to extract width and height from solver name
  std::smatch width_height;
  if (std::regex_match(solver_face_detect, width_height,
                       std::regex(".*w([0-9]+)h([0-9]+).*"))) {
    recommended_width = std::stoi(width_height[1]);
    recommended_height = std::stoi(width_height[2]);
  } else {
    // Solvers without width and height in name can accept dynamic input
    // image size. For best results we recommend to scale the input image to
    // a resolution recommended by the sfeFaceDetectInputSize function

    // Use sfeFaceDetectInputSize to get recommended input image dimensions
    // for given min_face_size/max_face_size
    SFEDetectionInputSize input_size{};
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectionAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &input_size);
    utils::checkError(error);
    recommended_width = input_size.width;
    recommended_height = input_size.height;
  };
}

inline void printFaceDetectionInfo(SFEDetection &detected_face) {
  std::cout << "SFEDetection{ " << std::endl;
  std::cout << "  confidence: " << detected_face.confidence << std::endl;
  std::cout << "  SFEBoundingBox{ " << std::endl;
  std::cout << "    x: " << detected_face.bounding_box.x << std::endl;
  std::cout << "    y: " << detected_face.bounding_box.y << std::endl;
  std::cout << "    width: " << detected_face.bounding_box.width << std::endl;
  std::cout << "    height: " << detected_face.bounding_box.height << std::endl;
  std::cout << "    }" << std::endl;
  std::cout << "  } " << std::endl;
  std::cout << std::endl;
}

inline void prinFaceAreaInfo(SFEFaceArea &face_area) {
  std::cout << "SFEFaceArea{ " << std::endl;
  std::cout << "  size: " << face_area.size << " [px]" << std::endl;
  std::cout << "  area: " << face_area.area << std::endl;
  std::cout << "  area_in_image: " << face_area.area_in_image << std::endl;
  std::cout << "}" << std::endl;
  std::cout << std::endl;
```

```
}

inline void printFaceHeadPose(SFEFaceHeadPose &head_pose) {
  std::cout « "SFEFaceHeadPose{" « std::endl;
  std::cout « "  pitch: " « head_pose.pitch « std::endl;
  std::cout « "  yaw: " « head_pose.yaw « std::endl;
  std::cout « "  roll: " « head_pose.roll « std::endl;
  std::cout « "} " « std::endl;
  std::cout « std::endl;
}

inline void printFaceSize(size_t face_size) {
  std::cout « "Face size: " « face_size « " [px]" « std::endl;
  std::cout « std::endl;
}

inline void printMaskConfidence(float mask_confidence) {
  std::cout « "Mask confidence: " « mask_confidence « std::endl;
  std::cout « std::endl;
}

inline void
printFaceQualityAttributes(SFEFaceQualityAttributes &quality_attributes) {
  std::cout « "SFEFaceQualityAttributes{" « std::endl;
  std::cout « "  sharpness: " « quality_attributes.sharpness « std::endl;
  std::cout « "  brightness: " « quality_attributes.brightness « std::endl;
  std::cout « "  contrast: " « quality_attributes.contrast « std::endl;
  std::cout « "  unique_intensity_levels: "
            « quality_attributes.unique_intensity_levels « std::endl;
  std::cout « "} " « std::endl;
  std::cout « std::endl;
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-l: Path to landmarks solver." « std::endl;
  std::cout « "-i: Path to liveness solver." « std::endl;
  std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
  std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
  std::cout « "-x: Max face size in pixels to detect." « std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE: 0 Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
        } else {
          std::cerr « "Option " « arg « " requires a value." « std::endl;
          return 1;
        }
      } else {
        std::cerr « "Unknown option: " « arg « std::endl;
        return 1;
      }
    }

    // Assign values to variables based on parsed arguments
    if (args.count("-p"))
      image_probe = args["-p"];
    if (args.count("-d"))
      solver_face_detect = args["-d"];
    if (args.count("-l"))
      solver_face_landmarks = args["-l"];
    if (args.count("-i"))
      solver_face_liveness = args["-i"];
    if (args.count("-t"))
      detection_threshold = std::stof(args["-t"]);
    if (args.count("-m"))
      face_size_min = std::stoi(args["-m"]);
    if (args.count("-x"))
```

```
    face_size_max = std::stoi(args["-x"]);

  utils::printFormatted("PARAMETERS");
  // Print resource names
  std::cout << "Probe image: " << image_probe << std::endl;

  // Print solver names
  std::cout << "Face detection solver: " << solver_face_detect << std::endl;
  std::cout << "Face landmarks solver: " << solver_face_landmarks
            << std::endl;
  std::cout << "Face liveness solver: " << solver_face_liveness << std::endl;

  // Print other options
  std::cout << "Min face size [px]: " << face_size_min << std::endl;
  std::cout << "Max face size [px]: " << face_size_max << std::endl;
  std::cout << "Detection threshold: " << detection_threshold << std::endl;
}

utils::printToolkitInfo();

SFEError error{};

SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
SFESolver landmarks_solver{};
DEFER(sfeSolverFree(landmarks_solver));
SFESolver liveness_solver{};
DEFER(sfeSolverFree(liveness_solver));
{ // STAGE 1: loading solvers
  utils::printFormatted("LOADING solvers");
  // Initialize face detection solver
  error = sfeSolverCreate(
      solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  // Initialize landmarks detection solver
  error = sfeSolverCreate(
      solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);

  // Initialize face liveness solver
  error = sfeSolverCreate(
      solver_face_liveness.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &liveness_solver);
  utils::checkError(error);
}

SFEImage image{};
DEFER(sfeImageFree(image));
SFEImage resized_image{};
DEFER(sfeImageFree(resized_image));
{ // STAGE 2: Load image

  utils::printFormatted("FACE DETECTION");
  // Load image data from file
  auto image_data = utils::readFile(image_probe);

  // Decode image from data
  error = sfeImageDecode(image_data.data(), image_data.size(), &image);
  utils::checkError(error);

  size_t recommended_width{};
  size_t recommended_height{};

  // Calculate optimal input image size for face detection. Image will be
  // resized to recommended size for optimal performance.
  getRecommendedImageSize(solver_face_detect, image, face_size_min,
                          face_size_max, recommended_width,
                          recommended_height);

  // Resize image to recommended size
  // NOTE: This function will resize the image without preserving the aspect
  // ratio of the image.
  error = sfeImageResize(image, recommended_width, recommended_height,
                         &resized_image);
  utils::checkError(error);
}
SFEDetection detected_face = {};
{ // STAGE 3: Detect face in the image
  size_t detection_count = 1;
  // Detect face in the image
  error = sfeDetect(detector_solver, resized_image, detection_threshold,
                        &detected_face, &detection_count);
```

```
  utils::checkError(error);

  if (detection_count == 0) {
    std::cout « "No face detected in the probe image." « std::endl;
    return 0;
  } else {
    std::cout « "Found " « detection_count
              « " face(s) in the probe image. Using the face with highest "
                 "confidence."
              « std::endl;
  }
}

std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
{ // STAGE 4: Get face landmarks

  // Use landmarks detection solver to get relevant landmarks for
  // the detected face
  error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
                           landmarks.data());
  utils::checkError(error);
}

SFEFaceLiveness liveness = {};
{ // STAGE 5: Liveness check
  utils::printFormatted("LIVENESS CHECK");
  error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
                                 &liveness);
  utils::checkError(error);
}

{ // STAGE 6: Print the liveness score
  std::cout « "Liveness score is " « liveness.score;
  // Recommended threshold for liveness detection, see EER threshold for distant fast mode in the
    documentation
  static const float LIVENESS_THRESHOLD = 0.81f;
  if (liveness.score < LIVENESS_THRESHOLD) {
    std::cout « " which is below " « LIVENESS_THRESHOLD
              « " threshold. Face is spoof." « std::endl;
  } else {
    std::cout « " which is above " « LIVENESS_THRESHOLD
              « " threshold. Face is genuine." « std::endl;
  }
}

// Optional face attributes to check for further analysis. See the
// documentation for more information.
float face_size = 0;
float mask_confidence;
SFEFaceArea face_area = {};
SFEFaceHeadPose head_pose{};
SFEFaceQualityAttributes quality_attributes{};
{ // STAGE 6: (optional) Face attributes
  utils::printFormatted("FACE ATTRIBUTES");

  error = sfeFaceSize(image, landmarks.data(), &face_size);
  utils::checkError(error);

  error = sfeFaceMaskConfidence(landmarks.data(), &mask_confidence);
  utils::checkError(error);

  error = sfeFaceArea(image, landmarks.data(), &face_area);
  utils::checkError(error);

  error = sfeFaceHeadPose(image, landmarks.data(), &head_pose);
  utils::checkError(error);

  error =
      sfeFaceQualityAttributes(image, landmarks.data(), &quality_attributes);
  utils::checkError(error);
}

{ // STAGE 8: Print the face attributes
  printFaceDetectionInfo(detected_face);
  printFaceSize(face_size);
  printMaskConfidence(mask_confidence);
  prinFaceAreaInfo(face_area);
  printFaceHeadPose(head_pose);
  printFaceQualityAttributes(quality_attributes);
}

{ // STAGE 9: Annotate the image
  // Render bounding box
  std::stringstream label;
  label « "Face" « std::fixed « std::setprecision(2)
        « " d:" « detected_face.confidence « " m:" « mask_confidence
```

```
          « " l:" « liveness.score;

    annotate::labelBox(image, label.str(), detected_face.bounding_box, annotate::WHITE,
                       annotate::GREEN);

    // Render landmarks
    for (auto &landmark : landmarks) {
      auto x = landmark.x * image.width;
      auto y = landmark.y * image.height;
      annotate::circle(image, x, y, 3, annotate::YELLOW);
    }

    // Save annotated image
    size_t size = image.width * image.height * 3;
    auto png_file = std::vector<unsigned char>(size);
    error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
    utils::checkError(error);
    png_file.resize(size);
    utils::saveFile("face_liveness.png", png_file);

    std::cout « std::endl;
    std::cout « "Annotated image saved to face_liveness.png" « std::endl;
  }

  utils::printFormatted("FINISHED");
}
```

10.5 example_face_demographic_attributes.cpp

Example of use of sfe_face library - demographic attributes (age, gender)

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_face.h"

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"

#include <iomanip>
#include <iostream>
#include <sstream>
#include <unordered_map>
#include <vector>

std::string image_probe = "assets/face/images/obiwan0.png";

std::string solver_face_detect = SOLVER_FACE_DETECT;
std::string solver_face_landmarks = SOLVER_FACE_LANDMARKS;
std::string solver_face_demographic = SOLVER_FACE_DEMOGRAPHIC_ATTRIBUTES;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

size_t face_size_min = 28;
size_t face_size_max = 170;

float detection_threshold = 0.1f;

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-l: Path to landmarks solver." « std::endl;
  std::cout « "-g: Path to demographic attributes solver." « std::endl;
  std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
  std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
  std::cout « "-x: Max face size in pixels to detect." « std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE 0: Parse command line arguments
    std::unordered_map<std::string, std::string> args;

    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
```

```
      } else {
        std::cerr « "Option " « arg « " requires a value." « std::endl;
        return 1;
      }
    } else {
      std::cerr « "Unknown option: " « arg « std::endl;
      return 1;
    }
  }

  if (args.count("-p"))
    image_probe = args["-p"];
  if (args.count("-d"))
    solver_face_detect = args["-d"];
  if (args.count("-l"))
    solver_face_landmarks = args["-l"];
  if (args.count("-g"))
    solver_face_demographic = args["-g"];
  if (args.count("-t"))
    detection_threshold = std::stof(args["-t"]);
  if (args.count("-m"))
    face_size_min = std::stoi(args["-m"]);
  if (args.count("-x"))
    face_size_max = std::stoi(args["-x"]);

  utils::printFormatted("PARAMETERS");
  std::cout « "Probe image: " « image_probe « std::endl;
  std::cout « "Face detection solver: " « solver_face_detect « std::endl;
  std::cout « "Face landmarks solver: " « solver_face_landmarks « std::endl;
  std::cout « "Face demographic attributes solver: " « solver_face_demographic
            « std::endl;
  std::cout « "Min face size [px]: " « face_size_min « std::endl;
  std::cout « "Max face size [px]: " « face_size_max « std::endl;
  std::cout « "Detection threshold: " « detection_threshold « std::endl;
}

utils::printToolkitInfo();

SFEError error{};

SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
SFESolver landmarks_solver{};
DEFER(sfeSolverFree(landmarks_solver));
SFESolver demographic_solver{};
DEFER(sfeSolverFree(demographic_solver));

{ // STAGE 1: loading solvers
  utils::printFormatted("LOADING solvers");

  error = sfeSolverCreate(
      solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  error = sfeSolverCreate(
      solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);

  error = sfeSolverCreate(
      solver_face_demographic.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &demographic_solver);
  utils::checkError(error);
}

SFEImage image{};
DEFER(sfeImageFree(image));

{ // STAGE 2: Load image
  utils::printFormatted("FACE DETECTION");
  // Load image data from file
  auto image_data = utils::readFile(image_probe);

  // Decode image from data
  error = sfeImageDecode(image_data.data(), image_data.size(), &image);
  utils::checkError(error);
}

SFEDetection detected_face = {};
{ // STAGE 3: Detect face in the image
  size_t detection_count = 1;
  error = sfeDetect(detector_solver, image, detection_threshold,
                    &detected_face, &detection_count);
  utils::checkError(error);

  if (detection_count == 0) {
```

```
      std::cout « "No face detected in the probe image." « std::endl;
      return 0;
    } else {
      std::cout « "Found " « detection_count
                « " face(s) in the probe image. Using the face with highest "
                   "confidence."
                « std::endl;
    }
  }

  std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
  { // STAGE 4: Get face landmarks
    error = sfeFaceLandmarks(landmarks_solver, image, &detected_face,
                             landmarks.data());
    utils::checkError(error);
  }

  SFEFaceDemographicAttributes attributes{};
  { // STAGE 5: Demographic attributes
    utils::printFormatted("FACE DEMOGRAPHIC ATTRIBUTES");
    error = sfeFaceDemographicAttributes(demographic_solver, image,
                                         landmarks.data(), &attributes);
    utils::checkError(error);
  }

  { // STAGE 6: Print demographic attributes
    std::cout « "Demographic attributes:" « std::endl;
    std::cout « "  Age: " « std::fixed « std::setprecision(1) « attributes.age
              « " years" « std::endl;
    std::cout « "  Gender score: " « std::setprecision(2) « attributes.gender
              « " (" « (attributes.gender > 0.0f ? "female" : "male") « ")"
              « std::endl;
  }

  utils::printFormatted("FINISHED");
}
```

10.6 example_iris_identify.cpp

Example of use of sfe_iris library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_iris.h"
#include <regex>
#include <vector>

#include <iomanip>
#include <sstream>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_match = "assets/iris/iris1_1.png";
std::string image_probe = "assets/iris/iris1_0.png";
std::string image_non_match = "assets/iris/iris0.png";

std::string solver_iris_detect = SOLVER_IRIS_DETECT;
std::string solver_iris_template = SOLVER_IRIS_TEMPLATE;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

float identification_threshold = 0.6f;

SFEIrisTemplate extract_template(std::string image_path,
                                 SFESolver &detector_solver,
                                 SFESolver &template_solver) {

  SFEImage image{};
  DEFER(sfeImageFree(image));
  SFEError error{};
  { // STAGE 1: Load image
    // Load image data from file
    auto image_data = utils::readFile(image_path);

    // Decode image from data
    error = sfeImageDecode(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
  }

  SFEDetection detected_iris = {};
```

```
  size_t iris_count = 1;
  { // STAGE 2: Detect iris in the image

    // Detect iris in the image
    error = sfeDetect(detector_solver, image, 0.1f, &detected_iris, &iris_count);
    utils::checkError(error);

    if (iris_count == 0 || detected_iris.confidence < 0.5) {
      throw std::runtime_error("No iris detected in the probe image.");
    }

    std::cout « "Found iris in the image. Extracting template..." « std::endl;
  }

  SFEIrisTemplate iris_template;
  { // STAGE 3 Extract probe iris template
    // Use template extraction solver to create new iris
    // template
    error = sfeIrisTemplateExtract(template_solver, image, &detected_iris,
                                   &iris_template);
    utils::checkError(error);
  }
  return iris_template;
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-g: Matching image file. Our \"iris database\"." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-e: Path to extraction solver." « std::endl;
  std::cout « "-t: Detection threshold. <0,1>" « std::endl;
  std::cout « "-i: Identification threshold. <0,1>" « std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE 0: Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
        } else {
          std::cerr « "Option " « arg « " requires a value." « std::endl;
          return 1;
        }
      } else {
        std::cerr « "Unknown option: " « arg « std::endl;
        return 1;
      }
    }

    // Assign values to variables based on parsed arguments
    if (args.count("-p"))
      image_probe = args["-p"];
    if (args.count("-g"))
      image_match = args["-g"];
    if (args.count("-d"))
      solver_iris_detect = args["-d"];
    if (args.count("-e"))
      solver_iris_template = args["-e"];
    if (args.count("-i"))
      identification_threshold = std::stof(args["-i"]);

    utils::printFormatted("EXAMPLE PARAMETERS");
    // Print resource names
    std::cout « "Probe image: " « image_probe « std::endl;
    std::cout « "Matching image: " « image_match « std::endl;

    // Print solver names
    std::cout « "Iris detection solver: " « solver_iris_detect « std::endl;
    std::cout « "Iris template solver: " « solver_iris_template « std::endl;

    // Print other options
    std::cout « "Identification threshold: " « identification_threshold
```

```
            « std::endl;
}

utils::printToolkitInfo();

SFEError error{};

SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
SFESolver landmarks_solver{};
DEFER(sfeSolverFree(landmarks_solver));
SFESolver template_solver{};
DEFER(sfeSolverFree(template_solver));
{ // STAGE 1.1: loading solvers
  utils::printFormatted("LOADING SOLVERS");
  // Initialize Iris detection solver
  error = sfeSolverCreate(
      solver_iris_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  // Initialize template extraction solver
  error = sfeSolverCreate(
      solver_iris_template.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &template_solver);
  utils::checkError(error);
}

SFEIrisTemplate probe_iris_template;
{ // STAGE 1: Extract probe iris template
  utils::printFormatted("PROBE TEMPLATE EXTRACTION");

  probe_iris_template =
      extract_template(image_probe, detector_solver, template_solver);
}

{ // STAGE 1.1: (optional) Print template attributes
  utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");

  SFEIrisTemplateVersion version = {};
  error = sfeIrisTemplateVersion(&probe_iris_template, &version);
  utils::checkError(error);

  std::cout « "Version of the probe template: " « version.version_major
            « '.' « version.version_minor « std::endl;

  float quality = {};
  error = sfeIrisTemplateQuality(&probe_iris_template, &quality);
  utils::checkError(error);

  std::cout « "Quality of the probe template: " « quality « std::endl;
}

size_t gallery_size = 10;
std::vector<SFEIrisTemplate> iris_template_gallery(gallery_size);
{ // STAGE 2: Create iris templates gallery
  utils::printFormatted("IRIS GALLERY EXTRACTION");

  auto matching_iris_template =
      extract_template(image_match, detector_solver, template_solver);

  auto non_matching_iris_template =
      extract_template(image_non_match, detector_solver, template_solver);

  // Fill the gallery with non-matching templates for demonstration purposes
  for (size_t i = 0; i < gallery_size; i++) {
    iris_template_gallery[i] = non_matching_iris_template;
  }

  // Add matching template to the gallery at index 5
  iris_template_gallery[5] = matching_iris_template;
}

size_t candidate_count = 1;
std::vector<SFETemplateIdentificationCandidate> identification_results(
    candidate_count);
{ // STAGE 3: Identify the probe template
  utils::printFormatted("1:N IDENTIFICATION");

  error = sfeIrisTemplateIdentify(
      &probe_iris_template, iris_template_gallery.data(),
      iris_template_gallery.size(), identification_threshold,
      identification_results.data(), &candidate_count, 4);
  utils::checkError(error);

  if (candidate_count == 0) {
    std::cout « "No candidates found above identification threshold "
```

```
              « identification_threshold « std::endl;
      return 0;
    }

    std::cout « "Found " « identification_results.size()
              « " candidates above identification threshold "
              « identification_threshold « std::endl;
    for (auto &result : identification_results)
      std::cout « "Template index: #" « result.index
                « ", score: " « result.score « std::endl;
  }

  { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
    utils::printFormatted("1:1 MATCHING");

    if (identification_results.size() == 0) {
      std::cout « "No candidates found above identification threshold "
                « identification_threshold « std::endl;
      return 0;
    }

    // Since it's sorted vector by score, the best candidate is the first one
    auto match_iris_template =
        iris_template_gallery[identification_results[0].index];

    float match_confidence;
    error = sfeIrisTemplateMatch(&probe_iris_template, &match_iris_template,
                                 &match_confidence);
    utils::checkError(error);

    std::cout
        « "Matching score of probe template with the best template, score: "
        « match_confidence « std::endl;
  }

  utils::printFormatted("FINISHED");
}
```

10.7 example_palm_identify.cpp

Example of use of sfe_palm library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_palm.h"
#include <regex>
#include <vector>

#include <iomanip>
#include <sstream>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_match = "assets/palm/palm_id1_r1.jpg";
std::string image_probe = "assets/palm/palm_id1_r2.jpg";
std::string image_non_match = "assets/palm/palm_id2_l1.jpg";

std::string solver_palm_detect = SOLVER_PALM_DETECT;
std::string solver_palm_extraction = SOLVER_PALM_EXTRACTION;

// NOTE: Using the same solver for landmarks as for detection, this is intentional.
std::string solver_palm_landmarks = SOLVER_PALM_DETECT;

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

float identification_threshold = 0.6f;

SFEPalmTemplate extract_template(std::string image_path,
                                 SFESolver &detector_solver, SFESolver &landmarks_solver, SFESolver
      &extraction_solver) {

  SFEImage image{};
  DEFER(sfeImageFree(image));
  SFEError error{};
  { // STAGE 1: Load image
    // Load image data from file
    auto image_data = utils::readFile(image_path);

    // Decode image from data
    error = sfeImageDecode(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
```

```
  }

  SFEDetection detected_palm = {};
  { // STAGE 2: Detect palm in the image

    // Detect palm in the image
    float detection_threshold = 0.1f;
    size_t detected_count = 1;
    error = sfeDetect(detector_solver, image, detection_threshold,
                      &detected_palm, &detected_count);
    utils::checkError(error);

    if (detected_count > 0 && detected_palm.confidence < 0.5) {
      throw std::runtime_error("No palm detected in the probe image.");
    }

    std::cout « "Found palm in the image. Extracting template..." « std::endl;
  }

  SFEPalmLandmarks landmarks = {};
  { // STAGE 3: Extract palm landmarks
    // Extract palm landmarks (hand position, orientation, keypoints)
    // These are required for template extraction
    error = sfePalmLandmarks(landmarks_solver, image, &detected_palm, &landmarks);
    utils::checkError(error);
  }

  SFEPalmTemplate palm_template;
  { // STAGE 4: Extract probe palm template
    // Use template extraction solver to create new palm template
    // Requires palm landmarks from previous step
    error = sfePalmTemplateExtract(extraction_solver, image, &landmarks,
                                   &palm_template);
    utils::checkError(error);
  }
  return palm_template;
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-g: Matching image file. Our \"palm database\"." « std::endl;
    std::cout « "-d: Path to detector solver." « std::endl;
    std::cout « "-l: Path to landmarks solver." « std::endl;
    std::cout « "-e: Path to template extraction solver." « std::endl;
    std::cout « "-t: Detection threshold. <0,1>" « std::endl;
    std::cout « "-i: Identification threshold. <0,1>" « std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE 0: Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
        } else {
          std::cerr « "Option " « arg « " requires a value." « std::endl;
          return 1;
        }
      } else {
        std::cerr « "Unknown option: " « arg « std::endl;
        return 1;
      }
    }

    // Assign values to variables based on parsed arguments
    if (args.count("-p"))
      image_probe = args["-p"];
    if (args.count("-g"))
      image_match = args["-g"];
    if (args.count("-d"))
      solver_palm_detect = args["-d"];
    if (args.count("-l"))
      solver_palm_landmarks = args["-l"];
```

```
  if (args.count("-e"))
    solver_palm_extraction = args["-e"];
  if (args.count("-i"))
    identification_threshold = std::stof(args["-i"]);

  utils::printFormatted("EXAMPLE PARAMETERS");
  // Print resource names
  std::cout « "Probe image: " « image_probe « std::endl;
  std::cout « "Matching image: " « image_match « std::endl;

  // Print solver names
  std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
  std::cout « "Palm landmarks solver: " « solver_palm_landmarks « std::endl;
  std::cout « "Palm extraction solver: " « solver_palm_extraction « std::endl;

  // Print other options
  std::cout « "Identification threshold: " « identification_threshold
            « std::endl;
}

utils::printToolkitInfo();

SFEError error{};
SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
SFESolver landmarks_solver{};
DEFER(sfeSolverFree(landmarks_solver));
SFESolver extraction_solver{};
DEFER(sfeSolverFree(extraction_solver));
{ // STAGE 1.1: loading solvers
  utils::printFormatted("LOADING SOLVERS");
  // Initialize Palm detection solver
  error = sfeSolverCreate(
      solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);
  // Initialize Palm landmarks solver (using same solver as detection)
  error = sfeSolverCreate(
      solver_palm_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);
  // Initialize Palm extraction solver
  error = sfeSolverCreate(
      solver_palm_extraction.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &extraction_solver);
  utils::checkError(error);
}

SFEPalmTemplate probe_palm_template;
{ // STAGE 1: Extract probe palm template
  utils::printFormatted("PROBE TEMPLATE EXTRACTION");

  probe_palm_template = extract_template(image_probe, detector_solver, landmarks_solver,
    extraction_solver);
}

{ // STAGE 1.1: (optional) Print template attributes
  utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");

  SFEPalmTemplateVersion version = {};
  error = sfePalmTemplateVersion(&probe_palm_template, &version);
  utils::checkError(error);

  std::cout « "Version of the probe template: " « version.major « '.'
            « version.minor « '.' « version.patch « std::endl;
}

size_t gallery_size = 10;
std::vector<SFEPalmTemplate> palm_template_gallery(gallery_size);
{ // STAGE 2: Create palm templates gallery
  utils::printFormatted("PALM GALLERY EXTRACTION");

  auto matching_palm_template =
      extract_template(image_match, detector_solver, landmarks_solver, extraction_solver);

  auto non_matching_palm_template =
      extract_template(image_non_match, detector_solver, landmarks_solver, extraction_solver);

  // Fill the gallery with non-matching templates for demonstration purposes
  for (size_t i = 0; i < gallery_size; i++) {
    palm_template_gallery[i] = non_matching_palm_template;
  }

  // Add matching template to the gallery at index 5
  palm_template_gallery[5] = matching_palm_template;
```

```
  }

  size_t candidate_count = 1;
  std::vector<SFETemplateIdentificationCandidate> identification_results(
      candidate_count);
  { // STAGE 3: Identify the probe template
    utils::printFormatted("1:N IDENTIFICATION");

    error = sfePalmTemplateIdentify(
        &probe_palm_template, palm_template_gallery.data(),
        palm_template_gallery.size(), identification_threshold,
        identification_results.data(), &candidate_count, 4);
    utils::checkError(error);

    if (candidate_count == 0) {
      std::cout « "No candidates found above identification threshold "
                « identification_threshold « std::endl;
      return 0;
    }

    std::cout « "Found " « identification_results.size()
              « " candidates above identification threshold "
              « identification_threshold « std::endl;
    for (auto &result : identification_results)
      std::cout « "Template index: #" « result.index
                « ", score: " « result.score « std::endl;
  }

  { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
    utils::printFormatted("1:1 MATCHING");

    if (identification_results.size() == 0) {
      std::cout « "No candidates found above identification threshold "
                « identification_threshold « std::endl;
      return 0;
    }

    // Since it's sorted vector by score, the best candidate is the first one
    auto match_palm_template =
        palm_template_gallery[identification_results[0].index];

    float match_confidence;
    error = sfePalmTemplateMatch(&probe_palm_template, &match_palm_template,
                                 &match_confidence);
    utils::checkError(error);

    std::cout
        « "Matching score of probe template with the best template, score: "
        « match_confidence « std::endl;
  }

  utils::printFormatted("FINISHED");
}
```

10.8 example_palm_liveness.cpp

Example of use of sfe_palm library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_palm.h"
#include <iomanip>
#include <regex>
#include <sstream>
#include <vector>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_probe = "assets/palm/palm_id1_r1.jpg";

std::string solver_palm_detect = SOLVER_PALM_DETECT;
#ifdef SOLVER_PALM_LIVENESS
std::string solver_palm_liveness = SOLVER_PALM_LIVENESS;
#endif // SOLVER_PALM_LIVENESS

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

float detection_threshold = 0.1f;

inline void printPalmDetectionInfo(SFEDetection &detected_palm) {
  std::cout « "SFEDetection{ " « std::endl;
  std::cout « "  confidence: " « detected_palm.confidence « std::endl;
```

```
  std::cout << "  SFEBoundingBox{ " << std::endl;
  std::cout << "    x: " << detected_palm.bounding_box.x << std::endl;
  std::cout << "    y: " << detected_palm.bounding_box.y << std::endl;
  std::cout << "    width: " << detected_palm.bounding_box.width << std::endl;
  std::cout << "    height: " << detected_palm.bounding_box.height << std::endl;
  std::cout << "    }" << std::endl;
  std::cout << "  }" << std::endl;
  std::cout << std::endl;
}

void printHelp() {
  std::cout << "Help: Usage of the program." << std::endl;
  std::cout << "Options:" << std::endl;
  std::cout << "-h: Display help." << std::endl;
  std::cout << "-p: Probe image file." << std::endl;
  std::cout << "-d: Path to detector solver." << std::endl;
  std::cout << "-i: Path to liveness solver." << std::endl;
  std::cout << "-t: Palm detection threshold. <0,1>" << std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE: 0 Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
        } else {
          std::cerr << "Option " << arg << " requires a value." << std::endl;
          return 1;
        }
      } else {
        std::cerr << "Unknown option: " << arg << std::endl;
        return 1;
      }
    }

    // Assign values to variables based on parsed arguments
    if (args.count("-p"))
      image_probe = args["-p"];
    if (args.count("-d"))
      solver_palm_detect = args["-d"];
    if (args.count("-i"))
      solver_palm_liveness = args["-i"];
    if (args.count("-t"))
      detection_threshold = std::stof(args["-t"]);

    utils::printFormatted("PARAMETERS");
    // Print resource names
    std::cout << "Probe image: " << image_probe << std::endl;

    // Print solver names
    std::cout << "Palm detection solver: " << solver_palm_detect << std::endl;
    std::cout << "Palm liveness solver: " << solver_palm_liveness << std::endl;

    // Print other options
    std::cout << "Detection threshold: " << detection_threshold << std::endl;
  }

  utils::printToolkitInfo();

  SFEError error{};

  SFESolver detector_solver{};
  DEFER(sfeSolverFree(detector_solver));
  SFESolver liveness_solver{};
  DEFER(sfeSolverFree(liveness_solver));
  { // STAGE 1: loading solvers
    utils::printFormatted("LOADING solvers");
    // Initialize palm detection solver
    error = sfeSolverCreate(
        solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &detector_solver);
    utils::checkError(error);

    // Initialize palm liveness solver
    error = sfeSolverCreate(
```

```
        solver_palm_liveness.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &liveness_solver);
    utils::checkError(error);
  }

  SFEImage image{};
  DEFER(sfeImageFree(image));
  { // STAGE 2: Load image

    utils::printFormatted("PALM DETECTION");
    // Load image data from file
    auto image_data = utils::readFile(image_probe);

    // Decode image from data
    error = sfeImageDecode(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
  }
  SFEDetection detected_palm = {};
  { // STAGE 3: Detect palm in the image
    size_t detection_count = 1;
    // Detect palm in the image
    error = sfeDetect(detector_solver, image, detection_threshold,
                          &detected_palm, &detection_count);
    utils::checkError(error);

    if (detection_count == 0) {
      std::cout « "No palm detected in the probe image." « std::endl;
      return 0;
    } else {
      std::cout « "Found " « detection_count
                « " palm(s) in the probe image. Using the palm with highest "
                   "confidence."
                « std::endl;
    }
  }

  printPalmDetectionInfo(detected_palm);

  float liveness_score = 0.0f;
  { // STAGE 4: Liveness check
    utils::printFormatted("LIVENESS CHECK");
    error = sfePalmLivenessPassive(liveness_solver, image, &detected_palm,
                                   &liveness_score);
    utils::checkError(error);
  }
  { // STAGE 5: Print the liveness score
    std::cout « "Liveness score is " « liveness_score;
    // Recommended threshold for liveness score, see EER threshold for palm liveness in the documentation
    static const float LIVENESS_THRESHOLD = 0.8189f;
    if (liveness_score < LIVENESS_THRESHOLD) {
      std::cout « " which is below " « LIVENESS_THRESHOLD
                « " threshold. Palm is spoof." « std::endl;
    } else {
      std::cout « " which is above " « LIVENESS_THRESHOLD
                « " threshold. Palm is genuine." « std::endl;
    }
  }

  { // STAGE 6: Annotate the image
    // Render bounding box
    std::stringstream label;
    label « "Palm" « std::fixed « std::setprecision(2)
          « " d:" « detected_palm.confidence « " l:" « liveness_score;

    annotate::labelBox(image, label.str(), detected_palm.bounding_box,
                       annotate::WHITE, annotate::GREEN);

    // Save annotated image
    size_t size = image.width * image.height * 3;
    auto png_file = std::vector<unsigned char>(size);
    error = sfeImageEncode(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
    utils::checkError(error);
    png_file.resize(size);
    utils::saveFile("palm_liveness.png", png_file);

    std::cout « std::endl;
    std::cout « "Annotated image saved to palm_liveness.png" « std::endl;
  }
  utils::printFormatted("FINISHED");
}
```

10.9 example_palm_attributes.cpp

Example of use of sfe_palm library

```
#include "sfe_toolkit/sfe_core.h"
#include "sfe_toolkit/sfe_palm.h"
#include <iomanip>
#include <optional>
#include <regex>
#include <sstream>
#include <vector>

#include "annotate.hpp"
#include "solvers.h"
#include "utils.hpp"
#include <unordered_map>

std::string image_probe = "assets/palm/palm_id1_r1.jpg";

std::string solver_palm_detect = SOLVER_PALM_DETECT;

#ifdef SOLVER_PALM_ATTRIBUTES
std::optional<std::string> solver_palm_attributes = SOLVER_PALM_ATTRIBUTES;
#else
std::optional<std::string> solver_palm_attributes = std::nullopt;
#endif // SOLVER_PALM_ATTRIBUTES

const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};

float detection_threshold = 0.1f;

inline void printPalmDetectionInfo(SFEDetection &detected_palm, SFEPalmLandmarks &attributes) {
  std::cout « "SFEDetection{ " « std::endl;
  std::cout « "  confidence: " « detected_palm.confidence « std::endl;
  std::cout « "  hand_position: " « attributes.hand_position « std::endl;
  std::cout « "  hand_orientation: " « attributes.hand_orientation « std::endl;
  std::cout « "  SFEBoundingBox{ " « std::endl;
  std::cout « "    x: " « detected_palm.bounding_box.x « std::endl;
  std::cout « "    y: " « detected_palm.bounding_box.y « std::endl;
  std::cout « "    width: " « detected_palm.bounding_box.width « std::endl;
  std::cout « "    height: " « detected_palm.bounding_box.height « std::endl;
  std::cout « "    }" « std::endl;
  std::cout « "  Keypoints{ " « std::endl;
  for (auto &keypoint : attributes.keypoints) {
    std::cout « "      SFEPalmKeypoint{ " « std::endl;
    std::cout « "        x: " « keypoint.x « std::endl;
    std::cout « "        y: " « keypoint.y « std::endl;
    std::cout « "      }" « std::endl;
  }
  std::cout « "    }" « std::endl;
  std::cout « "  }" « std::endl;
  std::cout « std::endl;
}

void printHelp() {
  std::cout « "Help: Usage of the program." « std::endl;
  std::cout « "Options:" « std::endl;
  std::cout « "-h: Display help." « std::endl;
  std::cout « "-p: Probe image file." « std::endl;
  std::cout « "-d: Path to detector solver." « std::endl;
  std::cout « "-i: Path to attributes solver." « std::endl;
  std::cout « "-t: Palm detection threshold. <0,1>" « std::endl;
}

int main(int argc, char *argv[]) {

  { // STAGE: 0 Parse command line arguments
    // Map to store argument values
    std::unordered_map<std::string, std::string> args;

    // Parse command line arguments
    for (int i = 1; i < argc; ++i) {
      std::string arg = argv[i];
      if (arg[0] == '-') {
        if (arg == "-h") {
          printHelp();
          return 0;
        }
        // Check if there's a next argument and it isn't another option
        if (i + 1 < argc && argv[i + 1][0] != '-') {
          args[arg] = argv[++i];
        } else {
          std::cerr « "Option " « arg « " requires a value." « std::endl;
          return 1;
        }
      } else {
```

```
      std::cerr « "Unknown option: " « arg « std::endl;
      return 1;
    }
  }

  // Assign values to variables based on parsed arguments
  if (args.count("-p"))
    image_probe = args["-p"];
  if (args.count("-d"))
    solver_palm_detect = args["-d"];
  if (args.count("-i"))
    solver_palm_attributes = args["-i"];
  if (args.count("-t"))
    detection_threshold = std::stof(args["-t"]);

  utils::printFormatted("PARAMETERS");
  // Print resource names
  std::cout « "Probe image: " « image_probe « std::endl;

  // Print solver names
  std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
  std::cout « "Palm attributes solver: "
            « (solver_palm_attributes.has_value() ? solver_palm_attributes.value() : "not specified")
            « std::endl;

  // Print other options
  std::cout « "Detection threshold: " « detection_threshold « std::endl;
}

utils::printToolkitInfo();

SFEError error{};

SFESolver detector_solver{};
DEFER(sfeSolverFree(detector_solver));
{ // STAGE 1: loading solvers
  utils::printFormatted("LOADING solvers");
  // Initialize palm detection solver
  error = sfeSolverCreate(
      solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);
}

SFEImage image{};
DEFER(sfeImageFree(image));
{ // STAGE 2: Load image

  utils::printFormatted("PALM DETECTION");
  // Load image data from file
  auto image_data = utils::readFile(image_probe);

  // Decode image from data
  error = sfeImageDecode(image_data.data(), image_data.size(), &image);
  utils::checkError(error);
}
SFEDetection detected_palm = {};
{ // STAGE 3: Detect palm in the image
  size_t detection_count = 1;
  // Detect palm in the image
  error = sfeDetect(detector_solver, image, detection_threshold,
                    &detected_palm, &detection_count);
  utils::checkError(error);

  if (detection_count == 0) {
    std::cout « "No palm detected in the probe image." « std::endl;
    return 0;
  } else {
    std::cout « "Found " « detection_count
              « " palm(s) in the probe image. Using the palm with highest "
                "confidence."
              « std::endl;
  }
}

SFEPalmLandmarks landmarks = {};
{ // STAGE 4: Extract palm landmarks
  utils::printFormatted("LANDMARKS");
  // Extract palm landmarks (hand position, orientation, keypoints)
  error = sfePalmLandmarks(detector_solver, image, &detected_palm, &landmarks);
  utils::checkError(error);
}

if (solver_palm_attributes.has_value()) { // STAGE 5: Attributes (if solver available)
    SFESolver attributes_solver{};
    DEFER(sfeSolverFree(attributes_solver));
    // Initialize palm attributes solver
```

```
      error = sfeSolverCreate(
          solver_palm_attributes.value().c_str(), SOLVER_PARAMETERS.data(),
          SOLVER_PARAMETERS.size(), &attributes_solver);
      utils::checkError(error);

      utils::printFormatted("ATTRIBUTES");
      SFEPalmAttributes attributes = {};
      error = sfePalmAttributes(attributes_solver, image, &landmarks,
                                &attributes);
      utils::checkError(error);

      printPalmDetectionInfo(detected_palm, landmarks);
      std::cout « "Hand orientation " « attributes.hand_orientation « std::endl;
      std::cout « "Hand position " « attributes.hand_position « std::endl;
      std::cout « "Quality " « attributes.quality « std::endl;
  } else {
    printPalmDetectionInfo(detected_palm, landmarks);
    std::cout « "Palm attributes solver not specified, skipping attributes analysis." « std::endl;
  }

  utils::printFormatted("FINISHED");
}
```

Index