SmartFace Embedded Toolkit

3.2.2

Generated by Doxygen 1.9.7

1 Overview 1

1.1 Key Features . . . 1
1.1.1 Face Detection . . . 1
1.1.2 Face Quality Evaluation . . . 1
1.1.3 Face Recognition . . . 1
1.1.4 Iris Recognition . . . 2
1.1.5 Palm Recognition . . . 2
1.2 Platform support . . . 2
1.2.1 HW acceleration . . . 2
1.3 Package . . . 3
1.4 Libraries . . . 3
1.4.1 API . . . 4
1.5 Licensing . . . 4
1.5.1 Hardware ID . . . 4
1.5.2 License generating . . . 4
1.5.3 License deployment . . . 5
1.6 Example application . . . 5
1.6.1 Python example . . . 5
1.7 Benchmarks application . . . 6
1.8 Support . . . 6

2 Changelog 7

2.1 [3.2.2] - 2025-11-05 . . . 7
2.1.1 Fixed . . . 7
2.2 [3.2.1] - 2025-10-29 . . . 7
2.2.1 Changed . . . 7
2.3 [3.2.0] - 2025-10-14 . . . 7
2.3.1 Breaking Change . . . 7
2.4 [3.1.5] - 2025-10-10 . . . 7
2.4.1 Added . . . 7
2.5 [3.1.4] - 2025-10-06 . . . 8
2.5.1 Added . . . 8
2.5.2 Changed . . . 8
2.5.3 Fixed . . . 8
2.6 [3.1.3] - 2025-09-29 . . . 8
2.6.1 Fixed . . . 8
2.7 [3.1.2] - 2024-09-25 . . . 8
2.7.1 Changed . . . 8
2.8 [3.1.1] - 2025-09-18 . . . 8
2.8.1 Added . . . 8
2.8.2 Changed . . . 9
2.8.3 Fixed . . . 9

2.9 [3.1.0] - 2025-09-12 . . . 9
2.9.1 Added . . . 9
2.9.2 Changed . . . 9
2.10 [3.0.2] - 2025-08-21 . . . 9
2.10.1 Breaking Change . . . 9
2.10.2 Added . . . 9
2.11 [3.0.1] - 2025-08-18 . . . 9
2.11.1 Added . . . 9
2.11.2 Changed . . . 10
2.11.3 Fixed . . . 10
2.12 [3.0.0] - 2025-08-12 . . . 10
2.12.1 Breaking Change . . . 10
2.12.2 Changed . . . 10
2.13 [2.0.2] - 2025-07-18 . . . 10
2.13.1 Changed . . . 10
2.14 [2.0.1] - 2025-07-08 . . . 10
2.14.1 Changed . . . 10
2.15 [2.0.0] - 2025-07-04 . . . 11
2.15.1 Breaking Change . . . 11
2.15.2 Deprecated . . . 11
2.15.3 Fixed . . . 11
2.15.4 Added . . . 11
2.16 [1.15.7] - 2025-06-25 . . . 11
2.16.1 Added . . . 11
2.16.2 Changed . . . 12
2.16.3 Fixed . . . 12
2.17 [1.15.6] - 2025-06-18 . . . 12
2.17.1 Fixed . . . 12
2.18 [1.15.5] - 2025-06-17 . . . 12
2.18.1 Added . . . 12
2.18.2 Fixed . . . 12
2.19 [1.15.4] - 2025-06-13 . . . 12
2.19.1 autotoc_md53 . . . 12
2.20 [1.15.3] - 2025-06-06 . . . 12
2.20.1 Added . . . 12
2.20.2 Fixed . . . 13
2.21 [1.15.2] - 2025-06-04 . . . 13
2.21.1 Fixed . . . 13
2.22 [1.15.1] - 2025-06-03 . . . 13
2.22.1 Added . . . 13
2.22.2 Fixed . . . 13
2.23 [1.15.0] - 2025-05-15 . . . 13

2.23.1 Added . . . 13
2.23.2 Changed . . . 13
2.23.3 Fixed . . . 14
2.23.4 Breaking Changes . . . 14
2.24 [1.14.6] - 2025-04-25 . . . 14
2.24.1 Added . . . 14
2.24.2 Changed . . . 14
2.25 [1.14.5] - 2025-04-11 . . . 14
2.25.1 Fixed . . . 14
2.26 [1.14.4] - 2025-04-10 . . . 14
2.26.1 Added . . . 14
2.26.2 Changed . . . 14
2.27 [1.14.3] - 2025-03-28 . . . 15
2.27.1 Fixed . . . 15
2.28 [1.14.2] - 2025-03-13 . . . 15
2.28.1 Changed . . . 15
2.29 [1.14.1] - 2025-03-05 . . . 15
2.29.1 Added . . . 15
2.29.2 Changed . . . 15
2.29.3 Fixed . . . 15
2.30 [1.13.5] - 2025-02-18 . . . 16
2.30.1 Changed . . . 16
2.31 [1.13.4] - 2025-01-21 . . . 16
2.31.1 Added . . . 16
2.31.2 Changed . . . 16
2.32 [1.13.3] - 2024-12-04 . . . 16
2.32.1 Added . . . 16
2.32.2 Fixed . . . 16
2.33 [1.13.2] - 2024-11-28 . . . 16
2.33.1 Changed . . . 16
2.34 [1.13.1] - 2024-11-19 . . . 17
2.34.1 Added . . . 17
2.34.2 Changed . . . 17
2.34.3 Fixed . . . 17
2.34.4 Removed . . . 17
2.35 [1.13.0] - 2024-11-07 . . . 17
2.35.1 Added . . . 17
2.35.2 Changed . . . 17
2.35.3 Fixed . . . 17
2.35.4 Removed . . . 17
2.36 [1.12.1] - 2024-11-06 . . . 17
2.36.1 Added . . . 17

2.36.2 Changed . . . 18
2.36.3 Fixed . . . 18
2.36.4 Removed . . . 18
2.37 [1.12.0] - 2024-10-25 . . . 18
2.37.1 Added . . . 18
2.37.2 Changed . . . 18
2.37.3 Fixed . . . 18
2.37.4 Removed . . . 18
2.38 [1.11.2] - 2024-10-04 . . . 18
2.38.1 Added . . . 18
2.38.2 Changed . . . 18
2.38.3 Fixed . . . 19
2.38.4 Removed . . . 19
2.39 [1.11.1] - 2024-08-05 . . . 19
2.39.1 Added . . . 19
2.39.2 Changed . . . 19
2.39.3 Fixed . . . 19
2.39.4 Removed . . . 19
2.40 [1.11.0] - 2024-07-26 . . . 19
2.40.1 Added . . . 19
2.40.2 Changed . . . 19
2.40.3 Fixed . . . 19
2.40.4 Removed . . . 19
2.41 [1.10.3] - 2024-06-12 . . . 19
2.41.1 Added . . . 19
2.41.2 Changed . . . 20
2.41.3 Fixed . . . 20
2.41.4 Removed . . . 20
2.42 [1.10.2] - 2024-05-28 . . . 20
2.42.1 Added . . . 20
2.42.2 Changed . . . 20
2.42.3 Fixed . . . 20
2.42.4 Removed . . . 20
2.43 [1.10.1] - 2024-05-24 . . . 20
2.43.1 Added . . . 20
2.43.2 Changed . . . 20
2.43.3 Fixed . . . 20
2.43.4 Removed . . . 20
2.44 [1.10.0] - 2024-05-20 . . . 20
2.44.1 Added . . . 20
2.44.2 Changed . . . 21
2.44.3 Fixed . . . 21

2.44.4 Removed . . . 21
2.45 [1.9.1] - 2024-03-05 . . . 21
2.45.1 Added . . . 21
2.45.2 Changed . . . 21
2.45.3 Fixed . . . 21
2.45.4 Removed . . . 21
2.46 [1.9.0] - 2024-02-29 . . . 21
2.46.1 Added . . . 21
2.46.2 Changed . . . 21
2.46.3 Fixed . . . 21
2.46.4 Removed . . . 21
2.47 [1.8.3] - 2024-02-23 . . . 21
2.47.1 Added . . . 21
2.47.2 Changed . . . 21
2.47.3 Fixed . . . 21
2.47.4 Removed . . . 22
2.48 [1.8.0] - 2024-01-31 . . . 22
2.48.1 Added . . . 22
2.48.2 Changed . . . 22
2.48.3 Fixed . . . 22
2.48.4 Removed . . . 22
2.49 [1.7.2] - 2023-11-29 . . . 22
2.49.1 Added . . . 22
2.49.2 Changed . . . 22
2.49.3 Fixed . . . 22
2.49.4 Removed . . . 23
2.50 [1.7.1] - 2023-11-27 . . . 23
2.50.1 Added . . . 23
2.50.2 Changed . . . 23
2.50.3 Fixed . . . 23
2.50.4 Removed . . . 23
2.51 [1.7.0] - 2023-11-17 . . . 23
2.51.1 Added . . . 23
2.51.2 Changed . . . 23
2.51.3 Fixed . . . 23
2.51.4 Removed . . . 23
2.52 [1.6.0] - 2023-11-07 . . . 23
2.52.1 Added . . . 23
2.52.2 Changed . . . 24
2.52.3 Fixed . . . 24
2.52.4 Removed . . . 24
2.53 [1.5.3] - 2023-10-27 . . . 24

2.53.1 Added . . . 24
2.53.2 Changed . . . 24
2.53.3 Fixed . . . 24
2.53.4 Removed . . . 24
2.54 [1.5.2] - 2023-09-14 . . . 24
2.54.1 Added . . . 24
2.54.2 Changed . . . 24
2.54.3 Fixed . . . 24
2.54.4 Removed . . . 24
2.55 [1.5.1] - 2023-09-07 . . . 24
2.55.1 Added . . . 24
2.55.2 Changed . . . 25
2.55.3 Fixed . . . 25
2.55.4 Removed . . . 25
2.56 [1.5.0] - 2023-08-25 . . . 25
2.56.1 Added . . . 25
2.56.2 Changed . . . 25
2.56.3 Fixed . . . 25
2.56.4 Removed . . . 25
2.57 [1.4.2] - 2023-08-14 . . . 25
2.57.1 Added . . . 25
2.57.2 Changed . . . 25
2.57.3 Fixed . . . 25
2.57.4 Removed . . . 25
2.58 [1.4.1] - 2023-07-10 . . . 25
2.58.1 Added . . . 25
2.58.2 Changed . . . 25
2.58.3 Fixed . . . 26
2.58.4 Removed . . . 26
2.59 [1.4.0] - 2023-06-23 . . . 26
2.59.1 Added . . . 26
2.59.2 Changed . . . 26
2.59.3 Fixed . . . 26
2.59.4 Removed . . . 26
2.60 [1.3.4] - 2023-06-06 . . . 26
2.60.1 Added . . . 26
2.60.2 Changed . . . 26
2.60.3 Fixed . . . 27
2.60.4 Removed . . . 27
2.61 [1.3.3] - 2023-06-02 . . . 27
2.61.1 Added . . . 27
2.61.2 Changed . . . 27

2.61.3 Fixed . . . 27
2.61.4 Removed . . . 27
2.62 [1.3.2] - 2023-05-18 . . . 27
2.62.1 Added . . . 27
2.62.2 Fixed . . . 27
2.63 [1.3.1] - 2023-05-09 . . . 27
2.63.1 Added . . . 27
2.63.2 Fixed . . . 28
2.64 [1.3.0] - 2023-04-23 . . . 28
2.64.1 Added . . . 28
2.65 [1.2.4] - 2023-04-18 . . . 28
2.65.1 Added . . . 28
2.66 [1.2.3] - 2023-04-14 . . . 28
2.66.1 Added . . . 28
2.67 [1.2.2] - 2023-04-03 . . . 28
2.67.1 Fixed . . . 28
2.68 [1.2.1] - 2023-03-21 . . . 28
2.68.1 Added . . . 28
2.69 [1.2.0] - 2023-03-09 . . . 29
2.69.1 Added . . . 29
2.69.2 Changed . . . 29
2.70 [1.1.0] - 2023-02-20 . . . 29
2.70.1 Added . . . 29
2.70.2 Changed . . . 29
2.71 [1.0.3] - 2023-02-13 . . . 29
2.71.1 Added . . . 29
2.71.2 Fixed . . . 29
2.72 [1.0.2] - 2023-01-30 . . . 30
2.72.1 Added . . . 30
2.72.2 Changed . . . 30
2.73 [1.0.1] - 2023-01-23 . . . 30
2.73.1 Added . . . 30
2.74 [1.0.0] - 2023-01-09 . . . 30

3 Features **31**

3.1 Face Detection . . . 31
3.1.1 Recommended input image dimensions . . . 35
3.1.2 Dynamic model input shape . . . 35
3.1.3 Static model input shape . . . 36
3.2 Face Landmarks Extraction . . . 36
3.2.1 Face Crop . . . 37
3.2.2 Face Mask Detection . . . 38

3.3 Face Template Extraction . . . 38
3.3.1 Face verification template compatibility . . . 39
3.4 Face Template Verification . . . 40
3.5 Face Template Identification . . . 41
3.5.1 Face Identification Accuracy . . . 42
3.6 Face Liveness Detection . . . 42
3.6.1 Recommended threshold for passive liveness modes . . . 43
3.6.2 Recommended face attributes . . . 43
3.7 Face Attributes . . . 44
3.7.1 Headpose angles . . . 44
3.7.2 Face Area . . . 44
3.7.3 Image quality attributes . . . 45
3.8 Iris detection . . . 45
3.9 Iris Template Extraction . . . 47
3.9.1 Iris verification template compatibility . . . 48
3.10 Iris Template Verification . . . 48
3.11 Iris Template Identification . . . 49
3.12 Palm detection . . . 50
3.13 Palm Template Extraction . . . 50
3.14 Palm Template Verification . . . 51
3.15 Palm Template Identification . . . 51
3.16 Palm Liveness Detection . . . 52
3.16.1 Recommended threshold for Palm passive liveness modes . . . 52
3.17 Palm Attributes . . . 53
3.18 Person Detection . . . 53
3.18.1 Oriented Person Detection . . . 53
3.19 Person Attributes . . . 54
3.19.1 Supported Person Attributes . . . 54
3.19.2 Attribute Confidence and Thresholds . . . 54
3.20 Supported platforms and features . . . 54
3.20.1 Face modality . . . 54
3.20.2 Iris modality . . . 55
3.20.3 Palm modality . . . 55
3.20.4 Person modality . . . 55
3.20.5 Face tracking . . . 55
3.21 Input image data . . . 55

4 Solvers **57**

4.1 ONNX Runtime . . . 57
4.1.1 CPU . . . 58
4.1.2 CUDA . . . 59
4.1.3 TensorRT . . . 59

4.2 Rockchip NPU . . . 59
4.2.1 RKNPU . . . 59
4.3 RKNPU2 . . . 60
4.4 Ambarella CVFlow . . . 60
4.5 Tensorflow Lite . . . 60
4.5.1 Parameters and their values . . . 61

5 Deprecated List **63**

6 Data Structure Index **65**
6.1 Data Structures . . . 65

7 File Index **67**
7.1 File List . . . 67

8 Data Structure Documentation **69**
8.1 SFEBbox Struct Reference . . . 69
8.1.1 Detailed Description . . . 69
8.1.2 Field Documentation . . . 69
8.1.2.1 height . . . 69
8.1.2.2 width . . . 70
8.1.2.3 x . . . 70
8.1.2.4 y . . . 70
8.2 SFEEntity Struct Reference . . . 70
8.2.1 Detailed Description . . . 71
8.2.2 Field Documentation . . . 71
8.2.2.1 uuid . . . 71
8.3 SFEEntityIdentificationCandidate Struct Reference . . . 71
8.3.1 Detailed Description . . . 72
8.3.2 Field Documentation . . . 72
8.3.2.1 entity . . . 72
8.3.2.2 score . . . 72
8.4 SFEFaceArea Struct Reference . . . 72
8.4.1 Detailed Description . . . 73
8.4.2 Field Documentation . . . 73
8.4.2.1 area . . . 73
8.4.2.2 area_in_image . . . 73
8.4.2.3 size . . . 73
8.5 SFEFaceCrop Struct Reference . . . 74
8.5.1 Detailed Description . . . 74
8.5.2 Field Documentation . . . 74
8.5.2.1 crop_box . . . 74
8.5.2.2 crop_image . . . 74
8.5.2.3 face_size_extension . . . 75

8.6 SFEFaceDetect Struct Reference . . . 75
8.6.1 Detailed Description . . . 75
8.6.2 Field Documentation . . . 76
8.6.2.1 bbox . . . 76
8.6.2.2 confidence . . . 76
8.6.2.3 raw_confidence . . . 76
8.7 SFEFaceHeadPose Struct Reference . . . 76
8.7.1 Detailed Description . . . 77
8.7.2 Field Documentation . . . 77
8.7.2.1 pitch . . . 77
8.7.2.2 roll . . . 77
8.7.2.3 yaw . . . 78
8.8 SFEFaceLandmarks Struct Reference . . . 78
8.8.1 Detailed Description . . . 78
8.8.2 Field Documentation . . . 78
8.8.2.1 confidence . . . 78
8.8.2.2 landmark_type . . . 79
8.8.2.3 x . . . 79
8.8.2.4 y . . . 79
8.9 SFEFaceQualityAttributes Struct Reference . . . 79
8.9.1 Detailed Description . . . 80
8.9.2 Field Documentation . . . 80
8.9.2.1 brightness . . . 80
8.9.2.2 contrast . . . 80
8.9.2.3 sharpness . . . 80
8.9.2.4 unique_intensity_levels . . . 81
8.10 SFEFaceTemplate Struct Reference . . . 81
8.10.1 Detailed Description . . . 81
8.10.2 Field Documentation . . . 81
8.10.2.1 data . . . 81
8.11 SFEFaceTemplateVersion Struct Reference . . . 82
8.11.1 Detailed Description . . . 82
8.11.2 Field Documentation . . . 82
8.11.2.1 version_major . . . 82
8.11.2.2 version_minor . . . 82
8.12 SFEFaceTracked Struct Reference . . . 83
8.12.1 Detailed Description . . . 83
8.12.2 Field Documentation . . . 83
8.12.2.1 id . . . 83
8.12.2.2 state . . . 83
8.12.2.3 tracked . . . 84
8.12.2.4 uuid . . . 84

8.13 SFEImage Struct Reference . . . 84
8.13.1 Detailed Description . . . 84
8.13.2 Field Documentation . . . 85
8.13.2.1 data . . . 85
8.13.2.2 height . . . 85
8.13.2.3 width . . . 85
8.14 SFEIrisDetect Struct Reference . . . 85
8.14.1 Detailed Description . . . 86
8.14.2 Field Documentation . . . 86
8.14.2.1 confidence . . . 86
8.14.2.2 iris_bounding_box . . . 86
8.14.2.3 keypoints . . . 86
8.14.2.4 pupil_bounding_box . . . 87
8.15 SFEIrisKeypoint Struct Reference . . . 87
8.15.1 Detailed Description . . . 87
8.15.2 Field Documentation . . . 87
8.15.2.1 confidence . . . 87
8.15.2.2 keypoint_type . . . 88
8.15.2.3 x . . . 88
8.15.2.4 y . . . 88
8.16 SFEIrisTemplate Struct Reference . . . 88
8.16.1 Detailed Description . . . 88
8.16.2 Field Documentation . . . 89
8.16.2.1 data . . . 89
8.17 SFEIrisTemplateVersion Struct Reference . . . 89
8.17.1 Detailed Description . . . 89
8.17.2 Field Documentation . . . 89
8.17.2.1 version_major . . . 89
8.17.2.2 version_minor . . . 90
8.18 SFEIrisTracked Struct Reference . . . 90
8.18.1 Detailed Description . . . 90
8.18.2 Field Documentation . . . 90
8.18.2.1 id . . . 90
8.18.2.2 state . . . 91
8.18.2.3 tracked . . . 91
8.18.2.4 uuid . . . 91
8.19 SFEOrientedBbox Struct Reference . . . 91
8.19.1 Detailed Description . . . 92
8.19.2 Field Documentation . . . 92
8.19.2.1 angle . . . 92
8.19.2.2 center_x . . . 92
8.19.2.3 center_y . . . 92

8.19.2.4 height . . . 92
8.19.2.5 width . . . 92
8.20 SFEPalmAttributes Struct Reference . . . 93
8.20.1 Detailed Description . . . 93
8.20.2 Field Documentation . . . 93
8.20.2.1 hand_orientation . . . 93
8.20.2.2 hand_position . . . 93
8.20.2.3 quality . . . 93
8.21 SFEPalmDetect Struct Reference . . . 94
8.21.1 Detailed Description . . . 94
8.21.2 Field Documentation . . . 94
8.21.2.1 bounding_box . . . 94
8.21.2.2 confidence . . . 94
8.21.2.3 hand_orientation . . . 95
8.21.2.4 hand_position . . . 95
8.21.2.5 keypoints . . . 95
8.22 SFEPalmKeypoint Struct Reference . . . 95
8.22.1 Detailed Description . . . 95
8.22.2 Field Documentation . . . 96
8.22.2.1 confidence . . . 96
8.22.2.2 x . . . 96
8.22.2.3 y . . . 96
8.23 SFEPalmQualityAttributes Struct Reference . . . 96
8.23.1 Detailed Description . . . 97
8.23.2 Field Documentation . . . 97
8.23.2.1 brightness . . . 97
8.23.2.2 hotspots_score . . . 97
8.23.2.3 sharpness . . . 97
8.24 SFEPalmTemplate Struct Reference . . . 97
8.24.1 Detailed Description . . . 98
8.24.2 Field Documentation . . . 98
8.24.2.1 data . . . 98
8.25 SFEPalmTemplateVersion Struct Reference . . . 98
8.25.1 Detailed Description . . . 98
8.25.2 Field Documentation . . . 99
8.25.2.1 major . . . 99
8.25.2.2 minor . . . 99
8.25.2.3 patch . . . 99
8.26 SFEPalmTracked Struct Reference . . . 99
8.26.1 Detailed Description . . . 100
8.26.2 Field Documentation . . . 100
8.26.2.1 id . . . 100

8.26.2.2 state . . . 100
8.26.2.3 tracked . . . 100
8.26.2.4 uuid . . . 101
8.27 SFESolverParameter Struct Reference . . . 101
8.27.1 Detailed Description . . . 101
8.27.2 Field Documentation . . . 101
8.27.2.1 name . . . 101
8.27.2.2 value . . . 101
8.28 SFETattooDetect Struct Reference . . . 102
8.28.1 Detailed Description . . . 102
8.28.2 Field Documentation . . . 102
8.28.2.1 bounding_box . . . 102
8.28.2.2 confidence . . . 102
8.29 SFETattooTemplate Struct Reference . . . 102
8.29.1 Detailed Description . . . 103
8.29.2 Field Documentation . . . 103
8.29.2.1 data . . . 103
8.30 SFETattooTemplateVersion Struct Reference . . . 103
8.30.1 Detailed Description . . . 103
8.30.2 Field Documentation . . . 104
8.30.2.1 version . . . 104
8.31 SFETemplateIdentificationCandidate Struct Reference . . . 104
8.31.1 Detailed Description . . . 104
8.31.2 Field Documentation . . . 104
8.31.2.1 index . . . 104
8.31.2.2 score . . . 104

9 File Documentation **105**

9.1 D:/glab/da0a89/1/changelog.md File Reference . . . 105
9.2 sfe_features.md File Reference . . . 105
9.3 sfe_solvers.md File Reference . . . 105
9.4 D:/glab/da0a89/1/example/example_face_identify.cpp File Reference . . . 105
9.4.1 Function Documentation . . . 106
9.4.1.1 detectFace() . . . 106
9.4.1.2 getRecommendedImageSize() . . . 107
9.4.1.3 main() . . . 108
9.4.1.4 printHelp() . . . 112
9.4.2 Variable Documentation . . . 113
9.4.2.1 detection_threshold . . . 113
9.4.2.2 face_size_max . . . 113
9.4.2.3 face_size_min . . . 113
9.4.2.4 identification_threshold . . . 113

9.4.2.5 image_gallery . . . 114
9.4.2.6 image_probe . . . 114
9.4.2.7 solver_face_detect . . . 114
9.4.2.8 solver_face_landmarks . . . 114
9.4.2.9 solver_face_template . . . 115
9.4.2.10 SOLVER_PARAMETERS . . . 115
9.5 example_face_identify.cpp . . . 115
9.6 D:/glab/da0a89/1/example/example_face_identify_entity.cpp File Reference . . . 121
9.6.1 Function Documentation . . . 121
9.6.1.1 extractTemplate() . . . 121
9.6.1.2 getRecommendedImageSize() . . . 122
9.6.1.3 main() . . . 124
9.6.1.4 printHelp() . . . 127
9.6.2 Variable Documentation . . . 127
9.6.2.1 detection_threshold . . . 127
9.6.2.2 face_size_max . . . 127
9.6.2.3 face_size_min . . . 127
9.6.2.4 gallery . . . 127
9.6.2.5 identification_threshold . . . 128
9.6.2.6 image_probe . . . 128
9.6.2.7 solver_face_detect . . . 128
9.6.2.8 solver_face_landmarks . . . 128
9.6.2.9 solver_face_template . . . 128
9.6.2.10 SOLVER_PARAMETERS . . . 128
9.7 example_face_identify_entity.cpp . . . 129
9.8 D:/glab/da0a89/1/example/example_face_liveness.cpp File Reference . . . 133
9.8.1 Function Documentation . . . 133
9.8.1.1 getRecommendedImageSize() . . . 133
9.8.1.2 main() . . . 134
9.8.1.3 prinFaceAreaInfo() . . . 138
9.8.1.4 printFaceDetectionInfo() . . . 138
9.8.1.5 printFaceHeadPose() . . . 139
9.8.1.6 printFaceQualityAttributes() . . . 139
9.8.1.7 printFaceSize() . . . 139
9.8.1.8 printHelp() . . . 140
9.8.1.9 printMaskConfidence() . . . 140
9.8.2 Variable Documentation . . . 140
9.8.2.1 detection_threshold . . . 140
9.8.2.2 face_size_max . . . 140
9.8.2.3 face_size_min . . . 140
9.8.2.4 image_probe . . . 141
9.8.2.5 solver_face_detect . . . 141

9.8.2.6 solver_face_landmarks . . . 141
9.8.2.7 solver_face_liveness . . . 141
9.8.2.8 SOLVER_PARAMETERS . . . 141
9.9 example_face_liveness.cpp . . . 142
9.10 D:/glab/da0a89/1/example/example_face_track.cpp File Reference . . . 146
9.10.1 Function Documentation . . . 147
9.10.1.1 detectFaces() . . . 147
9.10.1.2 frame() . . . 148
9.10.1.3 getRecommendedImageSize() . . . 148
9.10.1.4 main() . . . 149
9.10.1.5 operator<<() [1/4] . . . 150
9.10.1.6 operator<<() [2/4] . . . 151
9.10.1.7 operator<<() [3/4] . . . 151
9.10.1.8 operator<<() [4/4] . . . 152
9.10.1.9 printHelp() . . . 152
9.10.2 Variable Documentation . . . 152
9.10.2.1 detection_threshold . . . 152
9.10.2.2 face_size_max . . . 152
9.10.2.3 face_size_min . . . 152
9.10.2.4 image_probe . . . 153
9.10.2.5 solver_face_detect . . . 153
9.10.2.6 SOLVER_PARAMETERS . . . 153
9.11 example_face_track.cpp . . . 153
9.12 D:/glab/da0a89/1/example/example_iris_identify.cpp File Reference . . . 157
9.12.1 Function Documentation . . . 158
9.12.1.1 extract_template() . . . 158
9.12.1.2 main() . . . 159
9.12.1.3 printHelp() . . . 161
9.12.2 Variable Documentation . . . 161
9.12.2.1 identification_threshold . . . 161
9.12.2.2 image_match . . . 161
9.12.2.3 image_non_match . . . 162
9.12.2.4 image_probe . . . 162
9.12.2.5 solver_iris_detect . . . 162
9.12.2.6 solver_iris_template . . . 162
9.12.2.7 SOLVER_PARAMETERS . . . 162
9.13 example_iris_identify.cpp . . . 163
9.14 D:/glab/da0a89/1/example/example_palm_identify.cpp File Reference . . . 166
9.14.1 Function Documentation . . . 166
9.14.1.1 extract_template() . . . 166
9.14.1.2 main() . . . 167
9.14.1.3 printHelp() . . . 169

9.14.2 Variable Documentation . . . 170
9.14.2.1 identification_threshold . . . 170
9.14.2.2 image_match . . . 170
9.14.2.3 image_non_match . . . 170
9.14.2.4 image_probe . . . 170
9.14.2.5 solver_palm_detect . . . 170
9.14.2.6 solver_palm_extraction . . . 170
9.14.2.7 SOLVER_PARAMETERS . . . 171
9.15 example_palm_identify.cpp . . . 171
9.16 D:/glab/da0a89/1/include/sfe_core.h File Reference . . . 174
9.16.1 Detailed Description . . . 176
9.16.2 Typedef Documentation . . . 176
9.16.2.1 SFEBbox . . . 176
9.16.2.2 SFEEntity . . . 176
9.16.2.3 SFEEntityIdentificationCandidate . . . 176
9.16.2.4 SFEError . . . 177
9.16.2.5 SFEHwidMethod . . . 177
9.16.2.6 SFEImage . . . 177
9.16.2.7 SFEImageFormat . . . 177
9.16.2.8 SFEImageView . . . 178
9.16.2.9 SFELicenseSource . . . 178
9.16.2.10 SFEOrientedBbox . . . 178
9.16.2.11 SFESolver . . . 178
9.16.2.12 SFESolverParameter . . . 178
9.16.2.13 SFETemplateIdentificationCandidate . . . 178
9.16.3 Enumeration Type Documentation . . . 178
9.16.3.1 SFEHwidMethod . . . 178
9.16.3.2 SFEImageFormat . . . 179
9.16.3.3 SFELicenseSource . . . 180
9.16.4 Function Documentation . . . 180
9.16.4.1 sfeErrorFree() . . . 180
9.16.4.2 sfeErrorMessage() . . . 180
9.16.4.3 sfeHwidGet() . . . 181
9.16.4.4 sfeImageColorTranspose() . . . 181
9.16.4.5 sfeImageFree() . . . 182
9.16.4.6 sfeImageInfo() . . . 182
9.16.4.7 sfeImageLoad() . . . 183
9.16.4.8 sfeImageResize() . . . 183
9.16.4.9 sfeImageResizeWithAspectRatio() . . . 184
9.16.4.10 sfeImageSave() . . . 184
9.16.4.11 sfeLicenseSet() . . . 185
9.16.4.12 sfeLicenseValidate() . . . 185

9.16.4.13 sfeSolverCreate() . . . 185
9.16.4.14 sfeSolverCreateWithParameters() . . . 186
9.16.4.15 sfeSolverFree() . . . 186
9.16.4.16 sfeVersion() . . . 187
9.17 sfe_core.h . . . 187
9.18 D:/glab/da0a89/1/include/sfe_face.h File Reference . . . 189
9.18.1 Detailed Description . . . 192
9.18.2 Macro Definition Documentation . . . 193
9.18.2.1 SFE_FACE_LANDMARK_COUNT . . . 193
9.18.2.2 SFE_FACE_TEMPLATE_SIZE . . . 193
9.18.3 Typedef Documentation . . . 193
9.18.3.1 SFEFaceArea . . . 193
9.18.3.2 SFEFaceCrop . . . 193
9.18.3.3 SFEFaceDetect . . . 193
9.18.3.4 SFEFaceDetectAccuracyType . . . 193
9.18.3.5 SFEFaceEntity . . . 194
9.18.3.6 SFEFaceEntityIdentificationCandidate . . . 194
9.18.3.7 SFEFaceHeadPose . . . 194
9.18.3.8 SFEFaceLandmarks . . . 194
9.18.3.9 SFEFaceLandmarkType . . . 194
9.18.3.10 SFEFaceQualityAttributes . . . 194
9.18.3.11 SFEFaceTemplate . . . 195
9.18.3.12 SFEFaceTemplateIdentificationCandidate . . . 195
9.18.3.13 SFEFaceTemplateVersion . . . 195
9.18.3.14 SFEFaceTracked . . . 195
9.18.3.15 SFEFaceTracker . . . 195
9.18.3.16 SFEIdentificationResult . . . 195
9.18.4 Enumeration Type Documentation . . . 195
9.18.4.1 SFEFaceDetectAccuracyType . . . 195
9.18.4.2 SFEFaceLandmarkType . . . 196
9.18.4.3 SFEFaceTrackedState . . . 197
9.18.5 Function Documentation . . . 197
9.18.5.1 sfeFaceArea() . . . 197
9.18.5.2 sfeFaceCrop() . . . 198
9.18.5.3 sfeFaceDetect() . . . 198
9.18.5.4 sfeFaceDetectInputSize() . . . 199
9.18.5.5 sfeFaceEntityIdentify() . . . 199
9.18.5.6 sfeFaceHeadPose() . . . 201
9.18.5.7 sfeFaceLandmarks() . . . 201
9.18.5.8 sfeFaceLivenessPassive() . . . 202
9.18.5.9 sfeFaceMaskConfidence() . . . 202
9.18.5.10 sfeFaceQualityAttributes() . . . 203

9.18.5.11 sfeFaceSize() . . . 203
9.18.5.12 sfeFaceTemplateExtract() . . . 204
9.18.5.13 sfeFaceTemplateIdentify() . . . 204
9.18.5.14 sfeFaceTemplateMatch() . . . 205
9.18.5.15 sfeFaceTemplateQuality() . . . 206
9.18.5.16 sfeFaceTemplateVersion() . . . 206
9.18.5.17 sfeFaceTrackerCreate() . . . 206
9.18.5.18 sfeFaceTrackerFree() . . . 207
9.18.5.19 sfeFaceTrackerLost() . . . 207
9.18.5.20 sfeFaceTrackerRemoved() . . . 208
9.18.5.21 sfeFaceTrackerUpdate() . . . 208
9.19 sfe_face.h . . . 209
9.20 D:/glab/da0a89/1/include/sfe_iris.h File Reference . . . 211
9.20.1 Detailed Description . . . 213
9.20.2 Macro Definition Documentation . . . 213
9.20.2.1 SFE_IRIS_KEYPOINT_COUNT . . . 213
9.20.2.2 SFE_IRIS_TEMPLATE_SIZE . . . 214
9.20.3 Typedef Documentation . . . 214
9.20.3.1 SFEIrisDetect . . . 214
9.20.3.2 SFEIrisEntity . . . 214
9.20.3.3 SFEIrisEntityIdentificationCandidate . . . 214
9.20.3.4 SFEIrisKeypoint . . . 214
9.20.3.5 SFEIrisKeypointType . . . 214
9.20.3.6 SFEIrisTemplate . . . 215
9.20.3.7 SFEIrisTemplateIdentificationCandidate . . . 215
9.20.3.8 SFEIrisTemplateVersion . . . 215
9.20.3.9 SFEIrisTracked . . . 215
9.20.3.10 SFEIrisTracker . . . 215
9.20.4 Enumeration Type Documentation . . . 215
9.20.4.1 SFEIrisKeypointType . . . 215
9.20.4.2 SFEIrisTrackedState . . . 216
9.20.5 Function Documentation . . . 216
9.20.5.1 sfeIrisDetect() . . . 216
9.20.5.2 sfeIrisEntityIdentify() . . . 217
9.20.5.3 sfeIrisTemplateExtract() . . . 217
9.20.5.4 sfeIrisTemplateIdentify() . . . 218
9.20.5.5 sfeIrisTemplateMatch() . . . 219
9.20.5.6 sfeIrisTemplateQuality() . . . 219
9.20.5.7 sfeIrisTemplateVersion() . . . 220
9.20.5.8 sfeIrisTrackerCreate() . . . 220
9.20.5.9 sfeIrisTrackerFree() . . . 220
9.20.5.10 sfeIrisTrackerLost() . . . 221

9.20.5.11 sfeIrisTrackerRemoved() . . . 221
9.20.5.12 sfeIrisTrackerUpdate() . . . 221
9.21 sfe_iris.h . . . 222
9.22 D:/glab/da0a89/1/include/sfe_palm.h File Reference . . . 223
9.22.1 Detailed Description . . . 226
9.22.2 Macro Definition Documentation . . . 226
9.22.2.1 SFE_PALM_KEYPOINT_COUNT . . . 226
9.22.2.2 SFE_PALM_TEMPLATE_SIZE . . . 226
9.22.3 Typedef Documentation . . . 226
9.22.3.1 SFEHandOrientation . . . 226
9.22.3.2 SFEHandPosition . . . 227
9.22.3.3 SFEPalmAttributes . . . 227
9.22.3.4 SFEPalmDetect . . . 227
9.22.3.5 SFEPalmEntity . . . 227
9.22.3.6 SFEPalmEntityIdentificationCandidate . . . 227
9.22.3.7 SFEPalmKeypoint . . . 227
9.22.3.8 SFEPalmQualityAttributes . . . 227
9.22.3.9 SFEPalmTemplate . . . 228
9.22.3.10 SFEPalmTemplateIdentificationCandidate . . . 228
9.22.3.11 SFEPalmTemplateVersion . . . 228
9.22.3.12 SFEPalmTracked . . . 228
9.22.3.13 SFEPalmTracker . . . 228
9.22.4 Enumeration Type Documentation . . . 228
9.22.4.1 SFEHandOrientation . . . 228
9.22.4.2 SFEHandPosition . . . 229
9.22.4.3 SFEPalmTrackedState . . . 229
9.22.5 Function Documentation . . . 230
9.22.5.1 sfePalmAttributes() . . . 230
9.22.5.2 sfePalmDetect() . . . 230
9.22.5.3 sfePalmEntityIdentify() . . . 231
9.22.5.4 sfePalmLivenessPassive() . . . 231
9.22.5.5 sfePalmQualityAttributes() . . . 232
9.22.5.6 sfePalmTemplateExtract() . . . 232
9.22.5.7 sfePalmTemplateIdentify() . . . 233
9.22.5.8 sfePalmTemplateMatch() . . . 233
9.22.5.9 sfePalmTemplateVersion() . . . 234
9.22.5.10 sfePalmTrackerCreate() . . . 234
9.22.5.11 sfePalmTrackerFree() . . . 235
9.22.5.12 sfePalmTrackerLost() . . . 235
9.22.5.13 sfePalmTrackerRemoved() . . . 235
9.22.5.14 sfePalmTrackerUpdate() . . . 236
9.23 sfe_palm.h . . . 236

9.24 D:/glab/da0a89/1/include/sfe_tattoo.h File Reference . . . 238
9.24.1 Detailed Description . . . 239
9.24.2 Macro Definition Documentation . . . 239
9.24.2.1 SFE_TATTOO_TEMPLATE_SIZE . . . 239
9.24.3 Typedef Documentation . . . 240
9.24.3.1 SFETattooDetect . . . 240
9.24.3.2 SFETattooEntity . . . 240
9.24.3.3 SFETattooTemplate . . . 240
9.24.3.4 SFETattooTemplateVersion . . . 240
9.24.4 Function Documentation . . . 240
9.24.4.1 sfeTattooDetect() . . . 240
9.24.4.2 sfeTattooEntityIdentify() . . . 241
9.24.4.3 sfeTattooTemplateExport() . . . 241
9.24.4.4 sfeTattooTemplateExtract() . . . 242
9.24.4.5 sfeTattooTemplateIdentify() . . . 242
9.24.4.6 sfeTattooTemplateImport() . . . 243
9.24.4.7 sfeTattooTemplateMatch() . . . 243
9.24.4.8 sfeTattooTemplateVersion() . . . 243
9.25 sfe_tattoo.h . . . 244
9.26 D:/glab/da0a89/1/readme.md File Reference . . . 245

10 Examples **247**

10.1 example_face_identify.cpp . . . 247
10.2 example_face_identify_entity.cpp . . . 253
10.3 example_face_track.cpp . . . 257
10.4 example_face_liveness.cpp . . . 260
10.5 example_iris_identify.cpp . . . 265
10.6 example_palm_identify.cpp . . . 268

Index **273**

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 [3.2.2] - 2025-11-05

2.1.1 Fixed

- Minor documentation fixes.

2.2 [3.2.1] - 2025-10-29

2.2.1 Changed

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

2.3 [3.2.0] - 2025-10-14

2.3.1 Breaking Change

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

2.4 [3.1.5] - 2025-10-10

2.4.1 Added

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

2.5 [3.1.4] - 2025-10-06

2.5.1 Added

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

2.5.2 Changed

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

2.5.3 Fixed

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

2.6 [3.1.3] - 2025-09-29

2.6.1 Fixed

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

2.7 [3.1.2] - 2024-09-25

2.7.1 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]

2.8 [3.1.1] - 2025-09-18

2.8.1 Added

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

2.8.2 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]

2.8.3 Fixed

- Missing sfeImageFree in example_face_identify.cpp

2.9 [3.1.0] - 2025-09-12

2.9.1 Added

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

2.9.2 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]

2.10 [3.0.2] - 2025-08-21

2.10.1 Breaking Change

- Removed deprecated `palm::metrix` functionality

2.10.2 Added

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

2.11 [3.0.1] - 2025-08-18

2.11.1 Added

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

2.11.2 Changed

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

2.11.3 Fixed

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

2.12 [3.0.0] - 2025-08-12

2.12.1 Breaking Change

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

2.12.2 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

2.13 [2.0.2] - 2025-07-18

2.13.1 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

2.14 [2.0.1] - 2025-07-08

2.14.1 Changed

- The palm accurate detector performance for rockchip RKNNv2 is improved

2.15 [2.0.0] - 2025-07-04

2.15.1 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`

2.15.2 Deprecated

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

2.15.3 Fixed

- Added missing palm solvers in Rockchip RKNN2 packages

2.15.4 Added

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

2.16 [1.15.7] - 2025-06-25

2.16.1 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

2.16.2 Changed

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

2.16.3 Fixed

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

2.17 [1.15.6] - 2025-06-18

2.17.1 Fixed

- Upload conan packages into artifactory (BSDK-174)

2.18 [1.15.5] - 2025-06-17

2.18.1 Added

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

2.18.2 Fixed

- Python binding performance regression

2.19 [1.15.4] - 2025-06-13

2.19.1 autotoc_md53

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

2.20 [1.15.3] - 2025-06-06

2.20.1 Added

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

2.20.2 Fixed

- Missing generated Python bindings

2.21 [1.15.2] - 2025-06-04

2.21.1 Fixed

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

2.22 [1.15.1] - 2025-06-03

2.22.1 Added

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

2.22.2 Fixed

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

2.23 [1.15.0] - 2025-05-15

2.23.1 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

2.23.2 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

2.23.3 Fixed

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

2.23.4 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

2.24 [1.14.6] - 2025-04-25

2.24.1 Added

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

2.24.2 Changed

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

2.25 [1.14.5] - 2025-04-11

2.25.1 Fixed

- A bug in reported version string

2.26 [1.14.4] - 2025-04-10

2.26.1 Added

- Added support for Palm liveness with examples and benchmark

2.26.2 Changed

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

2.27 [1.14.3] - 2025-03-28

2.27.1 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

2.28 [1.14.2] - 2025-03-13

2.28.1 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

2.29 [1.14.1] - 2025-03-05

2.29.1 Added

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

2.29.2 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]

2.29.3 Fixed

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

2.30 [1.13.5] - 2025-02-18

2.30.1 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]

2.31 [1.13.4] - 2025-01-21

2.31.1 Added

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

2.31.2 Changed

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

2.32 [1.13.3] - 2024-12-04

2.32.1 Added

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

2.32.2 Fixed

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

2.33 [1.13.2] - 2024-11-28

2.33.1 Changed

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

2.34 [1.13.1] - 2024-11-19

2.34.1 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

2.34.2 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

2.34.3 Fixed

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

2.34.4 Removed

2.35 [1.13.0] - 2024-11-07

2.35.1 Added

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

2.35.2 Changed

2.35.3 Fixed

2.35.4 Removed

2.36 [1.12.1] - 2024-11-06

2.36.1 Added

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

2.36.2 Changed

- Separated examples from benchmarks

2.36.3 Fixed

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

2.36.4 Removed

2.37 [1.12.0] - 2024-10-25

2.37.1 Added

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

2.37.2 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

2.37.3 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

2.37.4 Removed

- Removed old passive liveness DISTANT and NEARBY model

2.38 [1.11.2] - 2024-10-04

2.38.1 Added

2.38.2 Changed

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

2.38.3 Fixed

- Fixed sfeFaceSize mismatched output parameter type from size_t to float

2.38.4 Removed

2.39 [1.11.1] - 2024-08-05

2.39.1 Added

2.39.2 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

2.39.3 Fixed

2.39.4 Removed

2.40 [1.11.0] - 2024-07-26

2.40.1 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

2.40.2 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

2.40.3 Fixed

2.40.4 Removed

2.41 [1.10.3] - 2024-06-12

2.41.1 Added

- Added reference to bindings API on documentation

2.41.2 Changed

- Fixed missing tensorflow_lite.so in Android AAR package

2.41.3 Fixed

2.41.4 Removed

2.42 [1.10.2] - 2024-05-28

2.42.1 Added

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

2.42.2 Changed

2.42.3 Fixed

2.42.4 Removed

2.43 [1.10.1] - 2024-05-24

2.43.1 Added

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

2.43.2 Changed

2.43.3 Fixed

2.43.4 Removed

2.44 [1.10.0] - 2024-05-20

2.44.1 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

2.44.2 Changed

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

2.44.3 Fixed

2.44.4 Removed

2.45 [1.9.1] - 2024-03-05

2.45.1 Added

- sfePersonTemplateType()

2.45.2 Changed

2.45.3 Fixed

2.45.4 Removed

2.46 [1.9.0] - 2024-02-29

2.46.1 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

2.46.2 Changed

2.46.3 Fixed

2.46.4 Removed

2.47 [1.8.3] - 2024-02-23

2.47.1 Added

2.47.2 Changed

2.47.3 Fixed

- More robust solver dll loading on Windows

2.47.4 Removed

2.48 [1.8.0] - 2024-01-31

2.48.1 Added

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

2.48.2 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

2.48.3 Fixed

2.48.4 Removed

2.49 [1.7.2] - 2023-11-29

2.49.1 Added

2.49.2 Changed

2.49.3 Fixed

- Fixed padding in sfeLicenseSet

2.49.4 Removed

2.50 [1.7.1] - 2023-11-27

2.50.1 Added

2.50.2 Changed

2.50.3 Fixed

- Fix in identification API

2.50.4 Removed

2.51 [1.7.0] - 2023-11-17

2.51.1 Added

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

2.51.2 Changed

2.51.3 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

2.51.4 Removed

2.52 [1.6.0] - 2023-11-07

2.52.1 Added

- sfeFaceEntityIdentify function - Identify entities with multiple templates.

2.52.2 Changed

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

2.52.3 Fixed

2.52.4 Removed

2.53 [1.5.3] - 2023-10-27

2.53.1 Added

2.53.2 Changed

2.53.3 Fixed

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

2.53.4 Removed

2.54 [1.5.2] - 2023-09-14

2.54.1 Added

2.54.2 Changed

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

2.54.3 Fixed

2.54.4 Removed

2.55 [1.5.1] - 2023-09-07

2.55.1 Added

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

2.55.2 Changed

2.55.3 Fixed

2.55.4 Removed

2.56 [1.5.0] - 2023-08-25

2.56.1 Added

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

2.56.2 Changed

2.56.3 Fixed

2.56.4 Removed

2.57 [1.4.2] - 2023-08-14

2.57.1 Added

2.57.2 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

2.57.3 Fixed

2.57.4 Removed

2.58 [1.4.1] - 2023-07-10

2.58.1 Added

2.58.2 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.

2.58.3 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

2.58.4 Removed

2.59 [1.4.0] - 2023-06-23

2.59.1 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

2.59.2 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.

2.59.3 Fixed

2.59.4 Removed

- Removed onnxrt_cuda and onnxrt_tensorrt solvers.

2.60 [1.3.4] - 2023-06-06

2.60.1 Added

2.60.2 Changed

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

2.60.3 Fixed

2.60.4 Removed

2.61 [1.3.3] - 2023-06-02

2.61.1 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

2.61.2 Changed

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

2.61.3 Fixed

2.61.4 Removed

2.62 [1.3.2] - 2023-05-18

2.62.1 Added

- New function sfeVersion() added into the sfe_core library

2.62.2 Fixed

- Fixed swapped roll and pitch calculation

2.63 [1.3.1] - 2023-05-09

2.63.1 Added

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

2.63.2 Fixed

- Fixed missing liveness solvers in Android package

2.64 [1.3.0] - 2023-04-23

2.64.1 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

2.65 [1.2.4] - 2023-04-18

2.65.1 Added

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

2.66 [1.2.3] - 2023-04-14

2.66.1 Added

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

2.67 [1.2.2] - 2023-04-03

2.67.1 Fixed

- Debugging symbols are now stripped

2.68 [1.2.1] - 2023-03-21

2.68.1 Added

- Face size api request to sfe_face lib

2.69 [1.2.0] - 2023-03-09

2.69.1 Added

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

2.69.2 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

2.70 [1.1.0] - 2023-02-20

2.70.1 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

2.70.2 Changed

- SFEBBox parameter added into sfeGetPersonAttributes function

2.71 [1.0.3] - 2023-02-13

2.71.1 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

2.71.2 Fixed

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

2.72 [1.0.2] - 2023-01-30

2.72.1 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

2.72.2 Changed

- SFEBBox structure moved to libsfe_core

2.73 [1.0.1] - 2023-01-23

2.73.1 Added

- Added face template identification
- Added SFEFaceLandmarkType enumeration

2.74 [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.

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))
```

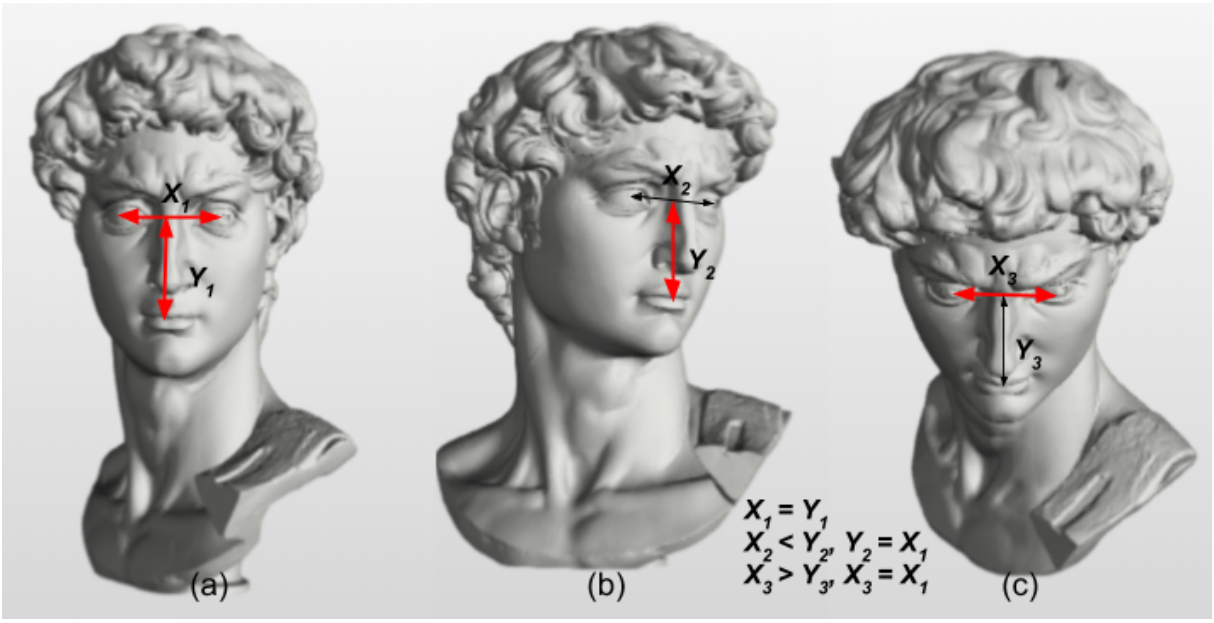

Figure 3.1 Face size: visualizations of various distances of inter eyes centers distances (X=distance(left_eye_center,right_eye_center)) and distances between the center of mouth and center point between eyes (Y=distance(mouth_center, eyes_center)). Face size is the maximum of these two distances. It 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.

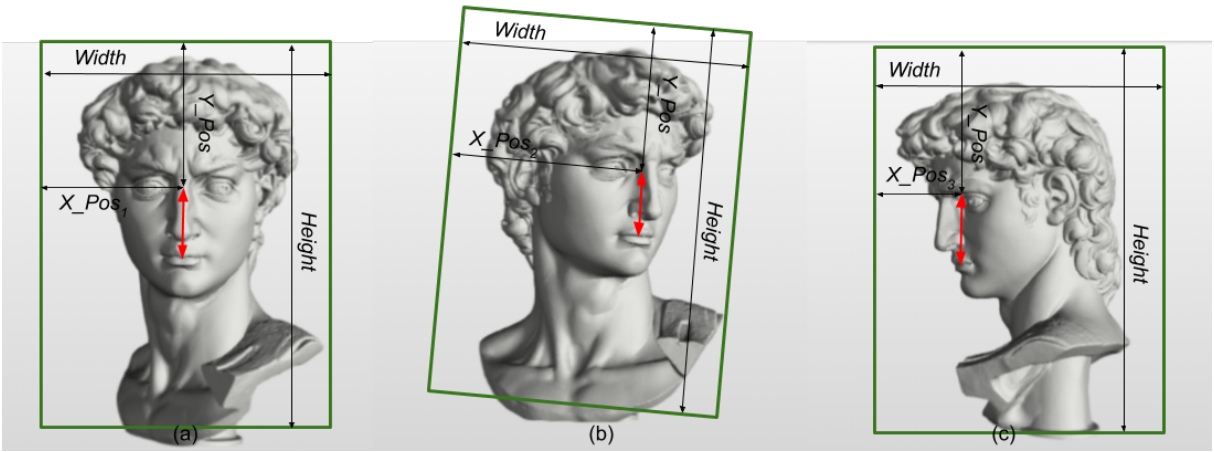

Figure 3.2 Face area: visualizations of face areas (green boxes) for the face in various positions. The face size is defined by a distance shown as the red arrow. There are different positions of the face area according to the face yaw rotation shown in (a),(b) and (c). Positions of the eyes' centers 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.

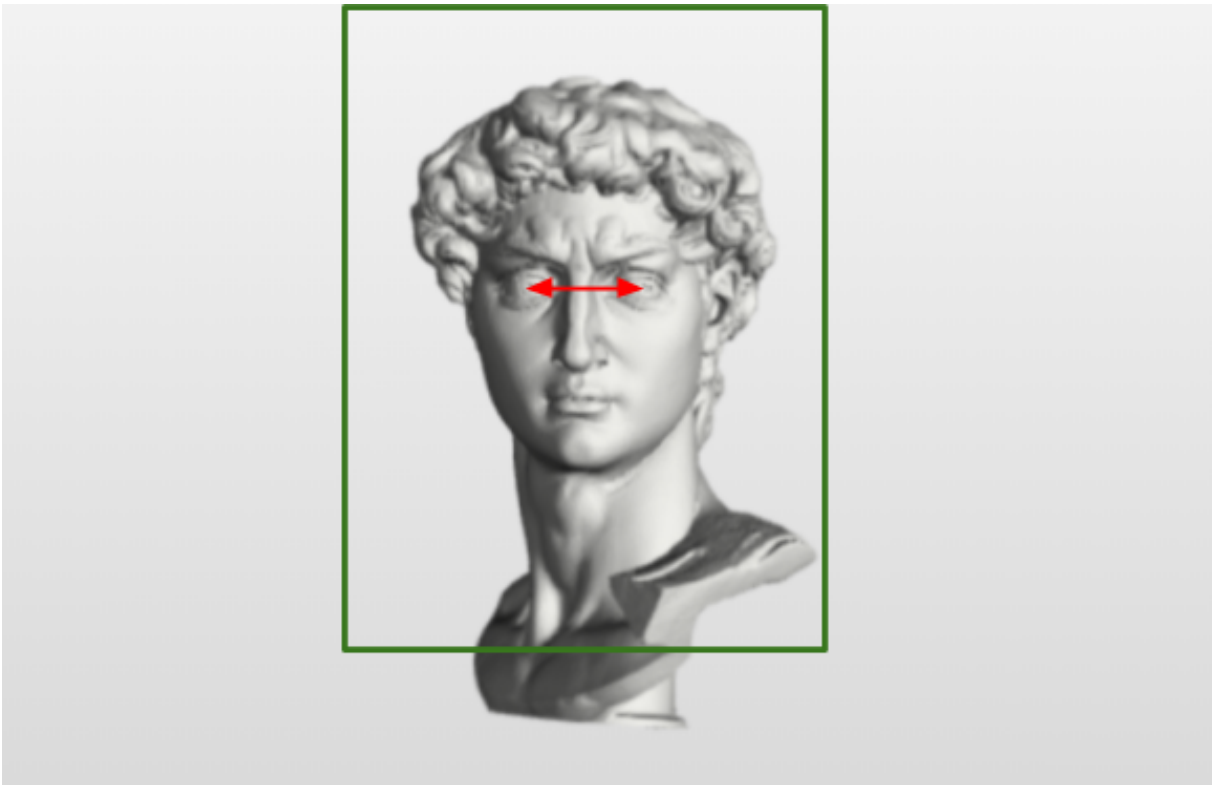

Figure 3.3 If image above has resolution of 730x470 pixels (image_area_size = 730 ∗ 470 = 343100) and face_size is 70 pixels (face_area_width = 70 ∗ 4 = 280, face_area_height = face_area_width / 0.75 = 373, face_area_size = 104440), then relative image size is relative_image_size = 104440 / 343100 = 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.

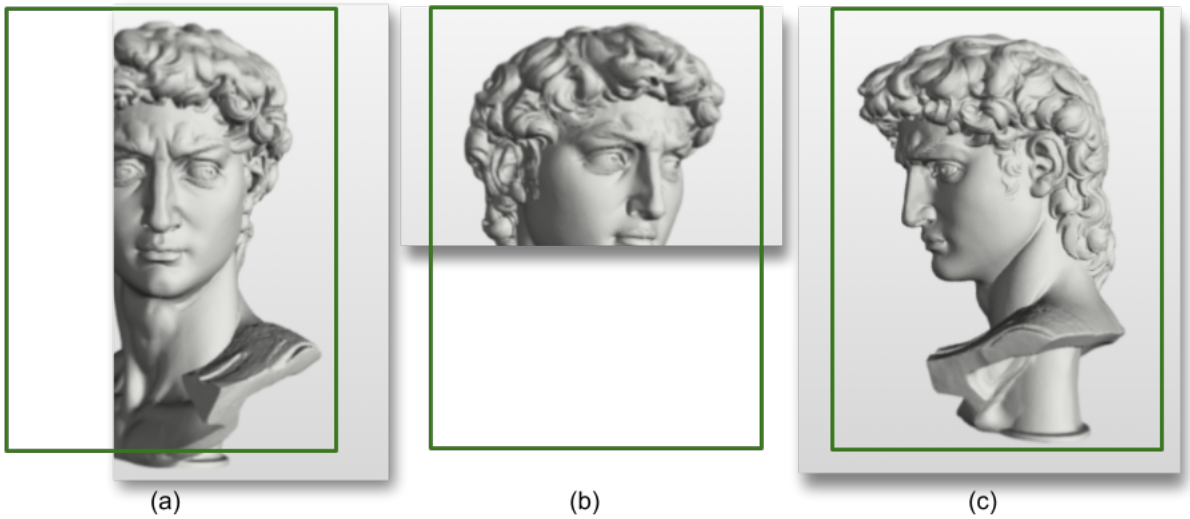

Figure 3.4 Example of face area visible in the image. Relative values: (a) 0.30, (b) 0.5, (c) 1.

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.

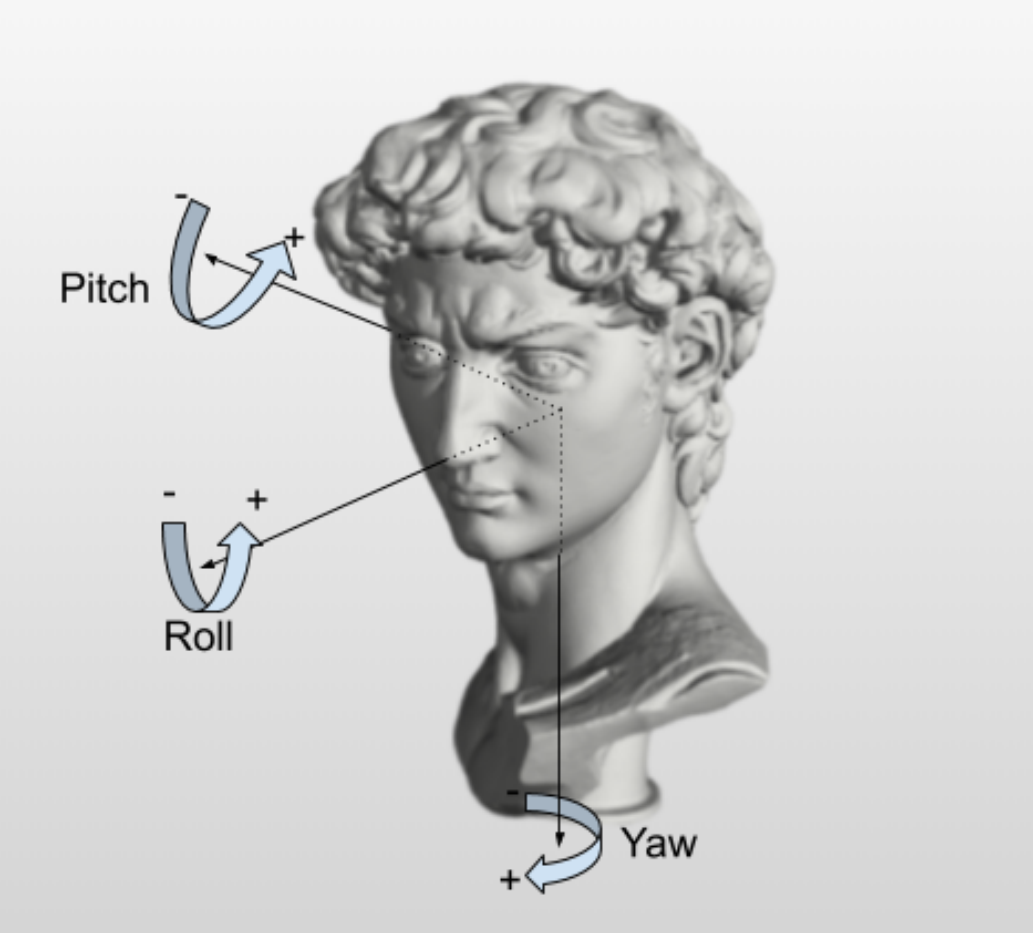

Figure 3.5 Possible face rotations (defined as in DIN9300).

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 [sfeFaceDetect]

Example

```
SFEFaceDetect detected_face = {};
{ // STAGE 3: Detect face in the image
  size_t detection_count = 1;
  // Detect face in the image
  error = sfeFaceDetect(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_face.detect`
- In Kotlin use `com.innovatrics.smartface.embedded.toolkit.face.detect`
- In Swift use `SFEToolkitFace.detect`
- In .NET use `Innovatrics.SFEToolkit.Face.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
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &recommended_width, &recommended_height);
    utils::checkError(error);
```

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.

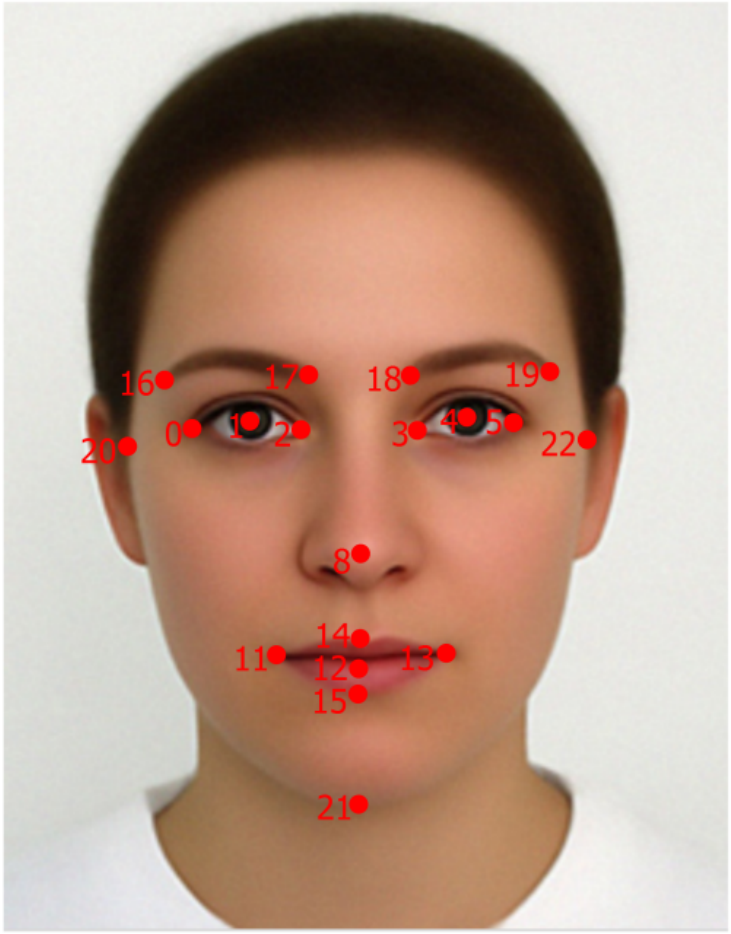

Figure 3.6 Facial landmarks and their IDs detected by SmartFace Embedded.

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.bbox,
                           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. Valid 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.
  uint32_t face_size_extension = 2;
  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 not compatible and therefore they cannot be combined in the verification and identification process. 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_mask* and *fast* modes. The *balanced* model is also capable of template extraction from faces with the face mask present.
- *accurate_mask* - Face templates suitable for verification/identification of very high accuracy are created when the *accurate_mask* 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_template_extract_fast
- face_template_extract_balanced
- face_template_extract_accurate_mask

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_template_extract_fast
- face_template_extract_balanced
- face_template_extract_accurate_mask

Example

```
{ // STAGE 1.4: Extract probe face template
  // Use template extraction solver to create new face
  // template
  error = sfeFaceTemplateExtract(template_solver, image, landmarks.data(),
                                 detected_face.raw_confidence,
                                 &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 to make sure the versions of the template being matched are the same, *balanced* and *accurate_mask* models produce templates of different versions that cannot be matched. When using multiple SmartFace Embedded Toolkits on multiple platforms or in conjunction with the Smart↩ Face platform and template compatibility is required, you need to make sure the same template extraction model is used on both sides.

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. Newer face templates (extracted with a faster or more accurate algorithm) are not compatible with previous templates.

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`

3.4 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<SFEIdentificationResult> 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.template↩ Identify`
- 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<SFEFaceEntityIdentificationCandidate> 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.

Solver selection for identification

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

- **Accurate Mask**: Lowest FAR, best for high-security applications where false positives are critical
- **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

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

```
float liveness_score = 0.0f;
{ // STAGE 5: Liveness check
  utils::printFormatted("LIVENESS CHECK");
  error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
                                 &liveness_score);
  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`

In **SFE Stream Processor** enable passive liveness detection by setting the `face_liveness_passive.enable=true` in `settings.yaml`. To use the correct liveness detection mode, set the path to the appropriate solver in `solvers.face_liveness_passive` in `settings.yaml`:

- face_liveness_passive_distant_fast
- face_liveness_passive_nearby

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`

3.8 Iris detection

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

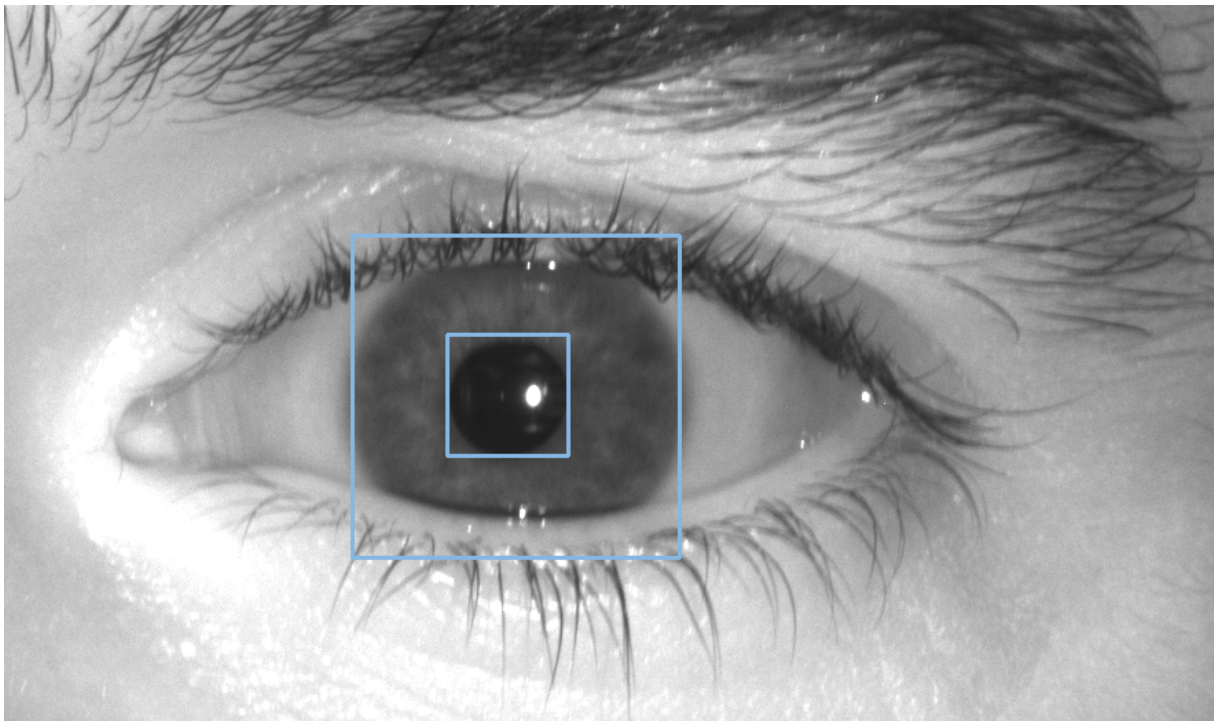

Figure 3.7 Iris detection results annotated in the input image.

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 [sfeIrisDetect]

Example

```
SFEIrisDetect detected_iris = {};
{ // STAGE 2: Detect iris in the image

  // Detect iris in the image
  error = sfeIrisDetect(detector_solver, image, &detected_iris);
  utils::checkError(error);

  if (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 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. Newer face 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<SFEIrisTemplateIdentificationCandidate> 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 [sfePalmDetect]

Example

```
SFEPalmDetect 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 = sfePalmDetect(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 3 Extract probe palm template
  // Use template extraction solver to create new palm
  // template
  error = sfePalmTemplateExtract(extraction_solver, image, &detected_palm,
                                 &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<SFEPalmTemplateIdentificationCandidate> 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, &detected_palm,
                          &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 [sfePersonDetectOriented]

Example

Use the following function in SFE Toolkit bindings:

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

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 Supported platforms and features

3.20.1 Face modality

Solver	Face detect	Face landmarks	Face template extract	Face passive liveness
ONNXRT	accurate_mask	0.25, 0.50	fast, balanced, accurate_mask	distant fast, nearby fast
Rockchip RKNPU	accurate_mask	0.25	accurate_mask	distant fast
Rockchip RKNPU2	accurate_mask	0.25	accurate_mask	-
Ambarella	accurate_mask	0.25	balanced, accurate_↩ mask	distant fast
Hailo	accurate_mask	0.25	accurate_mask	-
NXP iMX8	accurate_mask	0.25 , 0.50	balanced, accurate_↩ mask	-

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.

3.20.2 Iris modality

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

Other features such as iris verification and iris identification and face quality attributes

3.20.3 Palm modality

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

3.20.4 Person modality

Solver	Person detect	Person attributes
ONNXRT	yes	yes

3.20.5 Face tracking

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

Palm template extraction, palm matching and and palm identification are not implemented using a solver and so these are supported on all mentioned platforms.

3.21 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 [sfeImageLoad] 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 = sfeImageLoad(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

Deprecated List

Global SFEFaceEntity
Use SFEEntity instead, this type will be removed in future

Global SFEFaceEntityIdentificationCandidate
Replace with SFEEntityIdentificationCandidate

Global SFEFaceTemplateIdentificationCandidate
Replace with SFETemplateIdentificationCandidate

Global SFEIdentificationResult
Replace with SFETemplateIdentificationCandidate

Global SFEIrisEntity
Use SFEEntity instead, this type will be removed in future

Global SFEIrisEntityIdentificationCandidate
Replace with SFEEntityIdentificationCandidate

Global SFEIrisTemplateIdentificationCandidate
Replace with SFETemplateIdentificationCandidate

Global SFEPalmEntity
Use SFEEntity instead, this type will be removed in future

Global SFEPalmEntityIdentificationCandidate
Replace with SFEEntityIdentificationCandidate

Global sfePalmQualityAttributes (SFEImageView image, const SFEPalmDetect *palm_detect, SFEPalmQualityAttributes *out_quality_attributes)
Use `sfePalmAttributes` instead. Since version 2.0.0.

Global SFEPalmTemplateIdentificationCandidate
Replace with SFETemplateIdentificationCandidate

Global sfeSolverCreate (const char *solver_file, SFESolver *out_solver)
Use `sfeSolverCreateWithParameters` instead.

Global SFETattooEntity
Use SFEEntity instead, this type will be removed in future

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

SFEBbox
Bounding box . . . 69
SFEEntity
Face entity type, used to group templates for entity identification. This type is compatible with uuid_v4 that can be used to generate unique ids . . . 70
SFEEntityIdentificationCandidate
Candidate for identification by entity . . . 71
SFEFaceArea
Face area struct . . . 72
SFEFaceCrop
Face crop struct . . . 74
SFEFaceDetect
Face detect struct . . . 75
SFEFaceHeadPose
Face head pose struct containing angle rotations of head . . . 76
SFEFaceLandmarks
Face landmarks struct . . . 78
SFEFaceQualityAttributes
Face quality attributes struct . . . 79
SFEFaceTemplate
Face template struct . . . 81
SFEFaceTemplateVersion
Face template version struct . . . 82
SFEFaceTracked
Tracked entity struct . . . 83
SFEImage
Raw owned raster image representation, HWC|BGR order . . . 84
SFEIrisDetect
Iris detection struct . . . 85
SFEIrisKeypoint
Iris keypoint struct . . . 87
SFEIrisTemplate
Iris template struct . . . 88
SFEIrisTemplateVersion
Iris template version struct . . . 89

SFEIrisTracked
Tracked entity struct . . . 90
SFEOrientedBbox
Oriented bounding box . . . 91
SFEPalmAttributes . . . 93
SFEPalmDetect
Palm detection struct . . . 94
SFEPalmKeypoint
Palm keypoint struct . . . 95
SFEPalmQualityAttributes
Palm quality attributes struct . . . 96
SFEPalmTemplate
Palm template struct . . . 97
SFEPalmTemplateVersion
Palm template version struct . . . 98
SFEPalmTracked
Tracked entity struct . . . 99
SFESolverParameter
Solver parameter used to configure solvers . . . 101
SFETattooDetect
Tattoo detection struct . . . 102
SFETattooTemplate
Tattoo template struct . . . 102
SFETattooTemplateVersion
Tattoo template version struct . . . 103
SFETemplateIdentificationCandidate
Candidate for identification by template . . . 104

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

D:/glab/da0a89/1/example/example_face_identify.cpp . . . 105
D:/glab/da0a89/1/example/example_face_identify_entity.cpp . . . 121
D:/glab/da0a89/1/example/example_face_liveness.cpp . . . 133
D:/glab/da0a89/1/example/example_face_track.cpp . . . 146
D:/glab/da0a89/1/example/example_iris_identify.cpp . . . 157
D:/glab/da0a89/1/example/example_palm_identify.cpp . . . 166
D:/glab/da0a89/1/include/sfe_core.h
File containing API of sfe_core library as part of SFE Toolkit . . . 174
D:/glab/da0a89/1/include/sfe_face.h
File containing API of sfe_face library as part of SFE Toolkit . . . 189
D:/glab/da0a89/1/include/sfe_iris.h
File containing API of sfe_iris library as part of SFE Toolkit . . . 211
D:/glab/da0a89/1/include/sfe_palm.h
File containing API of sfe_palm library as part of SFE Toolkit . . . 223
D:/glab/da0a89/1/include/sfe_tattoo.h
File containing API of sfe_tattoo library as part of SFE Toolkit . . . 238

Chapter 8

Data Structure Documentation

8.1 SFEBbox Struct Reference

Bounding box.

`#include <sfe_core.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 83 of file sfe_core.h.

8.1.2 Field Documentation

8.1.2.1 height

`float SFEBbox::height`

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

Examples

example_face_liveness.cpp.

Definition at line 95 of file sfe_core.h.

8.1.2.2 width

```
float SFEBbox::width
```

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

Examples

example_face_liveness.cpp.

Definition at line 92 of file sfe_core.h.

8.1.2.3 x

```
float SFEBbox::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.

Definition at line 86 of file sfe_core.h.

8.1.2.4 y

```
float SFEBbox::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.

Definition at line 89 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

8.2 SFEEntity Struct Reference

Face 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]

8.2.1 Detailed Description

Face 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 138 of file sfe_core.h.

8.2.2 Field Documentation

8.2.2.1 uuid

```
uint8_t SFEEntity::uuid[16]
```

Examples

example_face_track.cpp.

Definition at line 139 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

8.3 SFEEntityIdentificationCandidate Struct Reference

Candidate for identification by entity.

```
#include <sfe_core.h>
```

Data Fields

- float score
  matching score
- SFEEntity entity
  candidate entity id

8.3.1 Detailed Description

Candidate for identification by entity.

Definition at line 151 of file sfe_core.h.

8.3.2 Field Documentation

8.3.2.1 entity

```
SFEEntity SFEEntityIdentificationCandidate::entity
```

candidate entity id

Definition at line 155 of file sfe_core.h.

8.3.2.2 score

```
float SFEEntityIdentificationCandidate::score
```

matching score

Definition at line 153 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

8.4 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>

8.4.1 Detailed Description

Face area struct.

Examples

example_face_liveness.cpp.

Definition at line 245 of file sfe_face.h.

8.4.2 Field Documentation

8.4.2.1 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 249 of file sfe_face.h.

8.4.2.2 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 252 of file sfe_face.h.

8.4.2.3 size

```
float SFEFaceArea::size
```

absolute face size

Examples

example_face_liveness.cpp.

Definition at line 247 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.5 SFEFaceCrop Struct Reference

Face crop struct.

`#include <sfe_face.h>`

Data Fields

- uint32_t face_size_extension

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

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

  Image of face cropped according the crop_box.

8.5.1 Detailed Description

Face crop struct.

Examples

example_face_identify.cpp.

Definition at line 355 of file sfe_face.h.

8.5.2 Field Documentation

8.5.2.1 crop_box

`SFEBbox SFEFaceCrop::crop_box`

Bounding box of cropped face including the crop extension.

Definition at line 360 of file sfe_face.h.

8.5.2.2 crop_image

`SFEImage SFEFaceCrop::crop_image`

Image of face cropped according the crop_box.

Examples

example_face_identify.cpp.

Definition at line 362 of file sfe_face.h.

8.5.2.3 face_size_extension

```
uint32_t 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 358 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.6 SFEFaceDetect Struct Reference

Face detect struct.

```
#include <sfe_face.h>
```

Data Fields

- SFEBbox bbox
  bounding box
- float confidence
  normalised value of detection confidence of detected face - range <0,1>
- float raw_confidence
  raw value of detection confidence of detected face - range <0,1>

8.6.1 Detailed Description

Face detect struct.

Examples

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

Definition at line 34 of file sfe_face.h.

8.6.2 Field Documentation

8.6.2.1 bbox

```
SFEBbox SFEFaceDetect::bbox
```

bounding box

Examples

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

Definition at line 36 of file sfe_face.h.

8.6.2.2 confidence

```
float SFEFaceDetect::confidence
```

normalised value of detection confidence of detected face - range <0,1>

Examples

example_face_liveness.cpp.

Definition at line 38 of file sfe_face.h.

8.6.2.3 raw_confidence

```
float SFEFaceDetect::raw_confidence
```

raw value of detection confidence of detected face - range <0,1>

Examples

example_face_identify.cpp, and example_face_identify_entity.cpp.

Definition at line 40 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.7 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.

8.7.1 Detailed Description

Face head pose struct containing angle rotations of head.

Examples

example_face_liveness.cpp.

Definition at line 267 of file sfe_face.h.

8.7.2 Field Documentation

8.7.2.1 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 270 of file sfe_face.h.

8.7.2.2 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 276 of file sfe_face.h.

8.7.2.3 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 273 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.8 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>

8.8.1 Detailed Description

Face landmarks struct.

Definition at line 87 of file sfe_face.h.

8.8.2 Field Documentation

8.8.2.1 confidence

```
float SFEFaceLandmarks::confidence
```

Confidence of the landmark detected on the face - range <0,1>

Definition at line 91 of file sfe_face.h.

8.8.2.2 landmark_type

```
SFEFaceLandmarkType SFEFaceLandmarks::landmark_type
```

Type of the landmark detected on the face.

Definition at line 89 of file sfe_face.h.

8.8.2.3 x

```
float SFEFaceLandmarks::x
```

X coordinate of the landmark relative to source image - range <0,1>

Definition at line 93 of file sfe_face.h.

8.8.2.4 y

```
float SFEFaceLandmarks::y
```

Y coordinate of the landmark relative to source image - range <0,1>

Definition at line 95 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.9 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.

8.9.1 Detailed Description

Face quality attributes struct.

Examples

example_face_liveness.cpp.

Definition at line 291 of file sfe_face.h.

8.9.2 Field Documentation

8.9.2.1 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 301 of file sfe_face.h.

8.9.2.2 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 307 of file sfe_face.h.

8.9.2.3 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 296 of file sfe_face.h.

8.9.2.4 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 313 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.10 SFEFaceTemplate Struct Reference

Face template struct.

```
#include <sfe_face.h>
```

Data Fields

- uint8_t data [SFE_FACE_TEMPLATE_SIZE]

8.10.1 Detailed Description

Face template struct.

Examples

example_face_identify.cpp, and example_face_identify_entity.cpp.

Definition at line 138 of file sfe_face.h.

8.10.2 Field Documentation

8.10.2.1 data

```
uint8_t SFEFaceTemplate::data[SFE_FACE_TEMPLATE_SIZE]
```

Definition at line 139 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.11 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

8.11.1 Detailed Description

Face template version struct.

Examples

example_face_identify.cpp.

Definition at line 143 of file sfe_face.h.

8.11.2 Field Documentation

8.11.2.1 version_major

`uint8_t SFEFaceTemplateVersion::version_major`

major template version

Examples

example_face_identify.cpp.

Definition at line 145 of file sfe_face.h.

8.11.2.2 version_minor

`uint8_t SFEFaceTemplateVersion::version_minor`

minor template version

Examples

example_face_identify.cpp.

Definition at line 147 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.12 SFEFaceTracked Struct Reference

Tracked entity struct.

`#include <sfe_face.h>`

Data Fields

- SFEFaceDetect tracked
  Tracked detection.
- uint64_t id
  Tracked ID.
- SFEEntity uuid
  UUID.
- SFEFaceTrackedState state
  Tracking state.

8.12.1 Detailed Description

Tracked entity struct.

Examples

example_face_track.cpp.

Definition at line 397 of file sfe_face.h.

8.12.2 Field Documentation

8.12.2.1 id

`uint64_t SFEFaceTracked::id`

Tracked ID.

Examples

example_face_track.cpp.

Definition at line 401 of file sfe_face.h.

8.12.2.2 state

`SFEFaceTrackedState SFEFaceTracked::state`

Tracking state.

Examples

example_face_track.cpp.

Definition at line 405 of file sfe_face.h.

8.12.2.3 tracked

`SFEFaceDetect SFEFaceTracked::tracked`

Tracked detection.

Definition at line 399 of file sfe_face.h.

8.12.2.4 uuid

`SFEEntity SFEFaceTracked::uuid`

UUID.

Examples

example_face_track.cpp.

Definition at line 403 of file sfe_face.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_face.h

8.13 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.*

8.13.1 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_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, and example_palm_identify.cpp.

Definition at line 69 of file sfe_core.h.

8.13.2 Field Documentation

8.13.2.1 data

```
unsigned char* SFEImage::data
```

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

Definition at line 75 of file sfe_core.h.

8.13.2.2 height

```
size_t SFEImage::height
```

Height of the image in pixels.

Examples

example_face_identify.cpp, and example_face_liveness.cpp.

Definition at line 73 of file sfe_core.h.

8.13.2.3 width

```
size_t SFEImage::width
```

Width of the image in pixels.

Examples

example_face_identify.cpp, and example_face_liveness.cpp.

Definition at line 71 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

8.14 SFEIrisDetect Struct Reference

Iris detection struct.

```
#include <sfe_iris.h>
```

Data Fields

- SFEBbox iris_bounding_box
  Iris bounding box.
- SFEBbox pupil_bounding_box
  Pupil bounding box.
- float confidence
  Iris detection confidence - range <0,1>
- SFEIrisKeypoint keypoints [SFE_IRIS_KEYPOINT_COUNT]
  Keypoints detected on the eye.

8.14.1 Detailed Description

Iris detection struct.

Examples

example_iris_identify.cpp.

Definition at line 55 of file sfe_iris.h.

8.14.2 Field Documentation

8.14.2.1 confidence

```
float SFEIrisDetect::confidence
```

Iris detection confidence - range <0,1>

Examples

example_iris_identify.cpp.

Definition at line 61 of file sfe_iris.h.

8.14.2.2 iris_bounding_box

```
SFEBbox SFEIrisDetect::iris_bounding_box
```

Iris bounding box.

Definition at line 57 of file sfe_iris.h.

8.14.2.3 keypoints

```
SFEIrisKeypoint SFEIrisDetect::keypoints[SFE_IRIS_KEYPOINT_COUNT]
```

Keypoints detected on the eye.

Definition at line 63 of file sfe_iris.h.

8.14.2.4 pupil_bounding_box

```
SFEBbox SFEIrisDetect::pupil_bounding_box
```

Pupil bounding box.

Definition at line 59 of file sfe_iris.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_iris.h

8.15 SFEIrisKeypoint Struct Reference

Iris keypoint struct.

```
#include <sfe_iris.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>

8.15.1 Detailed Description

Iris keypoint struct.

Definition at line 40 of file sfe_iris.h.

8.15.2 Field Documentation

8.15.2.1 confidence

```
float SFEIrisKeypoint::confidence
```

Confidence of the keypoint - range <0,1>

Definition at line 44 of file sfe_iris.h.

8.15.2.2 keypoint_type

```
SFEIrisKeypointType SFEIrisKeypoint::keypoint_type
```

Type of the keypoint.

Definition at line 42 of file sfe_iris.h.

8.15.2.3 x

```
float SFEIrisKeypoint::x
```

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

Definition at line 46 of file sfe_iris.h.

8.15.2.4 y

```
float SFEIrisKeypoint::y
```

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

Definition at line 48 of file sfe_iris.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_iris.h

8.16 SFEIrisTemplate Struct Reference

Iris template struct.

```
#include <sfe_iris.h>
```

Data Fields

- uint8_t data [SFE_IRIS_TEMPLATE_SIZE]

8.16.1 Detailed Description

Iris template struct.

Examples

example_iris_identify.cpp.

Definition at line 81 of file sfe_iris.h.

8.16.2 Field Documentation

8.16.2.1 data

```
uint8_t SFEIrisTemplate::data[SFE_IRIS_TEMPLATE_SIZE]
```

Definition at line 82 of file sfe_iris.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_iris.h

8.17 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.

8.17.1 Detailed Description

Iris template version struct.

Examples

example_iris_identify.cpp.

Definition at line 86 of file sfe_iris.h.

8.17.2 Field Documentation

8.17.2.1 version_major

```
uint8_t SFEIrisTemplateVersion::version_major
```

Major template version.

Examples

example_iris_identify.cpp.

Definition at line 88 of file sfe_iris.h.

8.17.2.2 version_minor

```
uint8_t SFEIrisTemplateVersion::version_minor
```

Minor template version.

Examples

example_iris_identify.cpp.

Definition at line 90 of file sfe_iris.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_iris.h

8.18 SFEIrisTracked Struct Reference

Tracked entity struct.

```
#include <sfe_iris.h>
```

Data Fields

- SFEIrisDetect tracked
  Tracked detection.
- uint64_t id
  Entity.
- SFEEntity uuid
  UUID.
- SFEIrisTrackedState state
  Tracking state.

8.18.1 Detailed Description

Tracked entity struct.

Definition at line 186 of file sfe_iris.h.

8.18.2 Field Documentation

8.18.2.1 id

```
uint64_t SFEIrisTracked::id
```

Entity.

Definition at line 190 of file sfe_iris.h.

8.18.2.2 state

```
SFEIrisTrackedState SFEIrisTracked::state
```

Tracking state.

Definition at line 194 of file sfe_iris.h.

8.18.2.3 tracked

```
SFEIrisDetect SFEIrisTracked::tracked
```

Tracked detection.

Definition at line 188 of file sfe_iris.h.

8.18.2.4 uuid

```
SFEEntity SFEIrisTracked::uuid
```

UUID.

Definition at line 192 of file sfe_iris.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_iris.h

8.19 SFEOrientedBbox Struct Reference

Oriented bounding box.

```
#include <sfe_core.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.

8.19.1 Detailed Description

Oriented bounding box.

Definition at line 99 of file sfe_core.h.

8.19.2 Field Documentation

8.19.2.1 angle

`float SFEOrientedBbox::angle`

Angle of the bounding box in radians.

Definition at line 114 of file sfe_core.h.

8.19.2.2 center_x

`float SFEOrientedBbox::center_x`

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

Definition at line 102 of file sfe_core.h.

8.19.2.3 center_y

`float SFEOrientedBbox::center_y`

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

Definition at line 105 of file sfe_core.h.

8.19.2.4 height

`float SFEOrientedBbox::height`

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

Definition at line 111 of file sfe_core.h.

8.19.2.5 width

`float SFEOrientedBbox::width`

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

Definition at line 108 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

8.20 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.

8.20.1 Detailed Description

Definition at line 274 of file sfe_palm.h.

8.20.2 Field Documentation

8.20.2.1 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.

Definition at line 281 of file sfe_palm.h.

8.20.2.2 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.

Definition at line 277 of file sfe_palm.h.

8.20.2.3 quality

`float SFEPalmAttributes::quality`

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

Definition at line 283 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.21 SFEPalmDetect Struct Reference

Palm detection struct.

`#include <sfe_palm.h>`

Data Fields

- SFEBbox bounding_box
  Palm bounding box.
- float confidence
  Palm detection confidence - range <0,1>
- SFEHandPosition hand_position
  Palm hand.
- SFEHandOrientation hand_orientation
  Palm orientation.
- SFEPalmKeypoint keypoints [SFE_PALM_KEYPOINT_COUNT]
  Keypoints detected on the palm.

8.21.1 Detailed Description

Palm detection struct.

Examples

example_palm_identify.cpp.

Definition at line 56 of file sfe_palm.h.

8.21.2 Field Documentation

8.21.2.1 bounding_box

```
SFEBbox SFEPalmDetect::bounding_box
```

Palm bounding box.

Definition at line 58 of file sfe_palm.h.

8.21.2.2 confidence

```
float SFEPalmDetect::confidence
```

Palm detection confidence - range <0,1>

Examples

example_palm_identify.cpp.

Definition at line 60 of file sfe_palm.h.

8.21.2.3 hand_orientation

```
SFEHandOrientation SFEPalmDetect::hand_orientation
```

Palm orientation.

Definition at line 64 of file sfe_palm.h.

8.21.2.4 hand_position

```
SFEHandPosition SFEPalmDetect::hand_position
```

Palm hand.

Definition at line 62 of file sfe_palm.h.

8.21.2.5 keypoints

```
SFEPalmKeypoint SFEPalmDetect::keypoints[SFE_PALM_KEYPOINT_COUNT]
```

Keypoints detected on the palm.

Definition at line 66 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.22 SFEPalmKeypoint Struct Reference

Palm keypoint struct.

```
#include <sfe_palm.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>

8.22.1 Detailed Description

Palm keypoint struct.

Definition at line 29 of file sfe_palm.h.

8.22.2 Field Documentation

8.22.2.1 confidence

```
float SFEPalmKeypoint::confidence
```

Confidence of the keypoint - range <0,1>

Definition at line 35 of file sfe_palm.h.

8.22.2.2 x

```
float SFEPalmKeypoint::x
```

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

Definition at line 31 of file sfe_palm.h.

8.22.2.3 y

```
float SFEPalmKeypoint::y
```

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

Definition at line 33 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.23 SFEPalmQualityAttributes Struct Reference

Palm quality attributes struct.

```
#include <sfe_palm.h>
```

Data Fields

- float sharpness
  Sharpness of the palm image.
- float brightness
  Brightness of the palm image.
- float hotspots_score
  Hotspots score of the palm image.

8.23.1 Detailed Description

Palm quality attributes struct.

Definition at line 253 of file sfe_palm.h.

8.23.2 Field Documentation

8.23.2.1 brightness

```
float SFEPalmQualityAttributes::brightness
```

Brightness of the palm image.

Definition at line 257 of file sfe_palm.h.

8.23.2.2 hotspots_score

```
float SFEPalmQualityAttributes::hotspots_score
```

Hotspots score of the palm image.

Definition at line 259 of file sfe_palm.h.

8.23.2.3 sharpness

```
float SFEPalmQualityAttributes::sharpness
```

Sharpness of the palm image.

Definition at line 255 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.24 SFEPalmTemplate Struct Reference

Palm template struct.

```
#include <sfe_palm.h>
```

Data Fields

- uint8_t data [SFE_PALM_TEMPLATE_SIZE]

8.24.1 Detailed Description

Palm template struct.

Examples

example_palm_identify.cpp.

Definition at line 88 of file sfe_palm.h.

8.24.2 Field Documentation

8.24.2.1 data

```
uint8_t SFEPalmTemplate::data[SFE_PALM_TEMPLATE_SIZE]
```

Definition at line 89 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.25 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.

8.25.1 Detailed Description

Palm template version struct.

Examples

example_palm_identify.cpp.

Definition at line 93 of file sfe_palm.h.

8.25.2 Field Documentation

8.25.2.1 major

```
uint8_t SFEPalmTemplateVersion::major
```

Major template version.

Examples

example_palm_identify.cpp.

Definition at line 95 of file sfe_palm.h.

8.25.2.2 minor

```
uint8_t SFEPalmTemplateVersion::minor
```

Minor template version.

Examples

example_palm_identify.cpp.

Definition at line 97 of file sfe_palm.h.

8.25.2.3 patch

```
uint8_t SFEPalmTemplateVersion::patch
```

Patch template version.

Examples

example_palm_identify.cpp.

Definition at line 99 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.26 SFEPalmTracked Struct Reference

Tracked entity struct.

```
#include <sfe_palm.h>
```

Data Fields

- SFEPalmDetect tracked
  Tracked detection.
- uint64_t id
  Tracked ID.
- SFEEntity uuid
  UUID.
- SFEPalmTrackedState state
  Tracking state.

8.26.1 Detailed Description

Tracked entity struct.

Definition at line 200 of file sfe_palm.h.

8.26.2 Field Documentation

8.26.2.1 id

```
uint64_t SFEPalmTracked::id
```

Tracked ID.

Definition at line 204 of file sfe_palm.h.

8.26.2.2 state

```
SFEPalmTrackedState SFEPalmTracked::state
```

Tracking state.

Definition at line 208 of file sfe_palm.h.

8.26.2.3 tracked

```
SFEPalmDetect SFEPalmTracked::tracked
```

Tracked detection.

Definition at line 202 of file sfe_palm.h.

8.26.2.4 uuid

```
SFEEntity SFEPalmTracked::uuid
```

UUID.

Definition at line 206 of file sfe_palm.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_palm.h

8.27 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

8.27.1 Detailed Description

Solver parameter used to configure solvers.

Definition at line 129 of file sfe_core.h.

8.27.2 Field Documentation

8.27.2.1 name

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

parameter name

Definition at line 131 of file sfe_core.h.

8.27.2.2 value

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

parameter value

Definition at line 133 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

8.28 SFETattooDetect Struct Reference

Tattoo detection struct.

`#include <sfe_tattoo.h>`

Data Fields

- SFEBbox bounding_box
  Tattoo bounding box.
- float confidence
  Tattoo detection confidence - range <0,1>

8.28.1 Detailed Description

Tattoo detection struct.

Definition at line 22 of file sfe_tattoo.h.

8.28.2 Field Documentation

8.28.2.1 bounding_box

`SFEBbox SFETattooDetect::bounding_box`

Tattoo bounding box.

Definition at line 24 of file sfe_tattoo.h.

8.28.2.2 confidence

`float SFETattooDetect::confidence`

Tattoo detection confidence - range <0,1>

Definition at line 26 of file sfe_tattoo.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_tattoo.h

8.29 SFETattooTemplate Struct Reference

Tattoo template struct.

`#include <sfe_tattoo.h>`

Data Fields

- uint8_t data [SFE_TATTOO_TEMPLATE_SIZE]

8.29.1 Detailed Description

Tattoo template struct.

Definition at line 48 of file sfe_tattoo.h.

8.29.2 Field Documentation

8.29.2.1 data

```
uint8_t SFETattooTemplate::data[SFE_TATTOO_TEMPLATE_SIZE]
```

Definition at line 49 of file sfe_tattoo.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_tattoo.h

8.30 SFETattooTemplateVersion Struct Reference

Tattoo template version struct.

```
#include <sfe_tattoo.h>
```

Data Fields

- uint32_t version
  Version.

8.30.1 Detailed Description

Tattoo template version struct.

Definition at line 53 of file sfe_tattoo.h.

8.30.2 Field Documentation

8.30.2.1 version

`uint32_t SFETattooTemplateVersion::version`

Version.

Definition at line 55 of file sfe_tattoo.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_tattoo.h

8.31 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

8.31.1 Detailed Description

Candidate for identification by template.

Definition at line 143 of file sfe_core.h.

8.31.2 Field Documentation

8.31.2.1 index

`size_t SFETemplateIdentificationCandidate::index`

index of the template from the input gallery

Definition at line 147 of file sfe_core.h.

8.31.2.2 score

`float SFETemplateIdentificationCandidate::score`

matching score

Definition at line 145 of file sfe_core.h.

The documentation for this struct was generated from the following file:

- D:/glab/da0a89/1/include/sfe_core.h

Chapter 9

File Documentation

9.1 D:/glab/da0a89/1/changelog.md File Reference

9.2 sfe_features.md File Reference

9.3 sfe_solvers.md File Reference

9.4 D:/glab/da0a89/1/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, SFEFaceDetect &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_TEMPLATE
- 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

9.4.1 Function Documentation

9.4.1.1 detectFace()

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

Examples

example_face_identify.cpp.

Definition at line 90 of file example_face_identify.cpp.

```
00091                                                {
00092
00093   SFEImage resized_image{};
00094   DEFER(sfeImageFree(resized_image));
00095   SFEError error{};
00096   { // STAGE 1 Load image
00097     size_t recommended_width{};
00098     size_t recommended_height{};
00099
00100     // Calculate optimal input image size for face detection. Image will be
00101     // resized to recommended size for optimal performance.
00102     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00103                             face_size_max, recommended_width,
00104                             recommended_height);
00105
00106     // Resize image to recommended size
00107     // NOTE: This function will resize the image without preserving the aspect
00108     // ratio of the image.
00109
00110     error = sfeImageResize(image, recommended_width, recommended_height,
00111                            &resized_image);
00112     utils::checkError(error);
00113   }
00114
00115   size_t detection_count = 1;
00116   { // STAGE 2: Detect face in the image
00117
00118     // Detect face in the image
00119     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00120                           &detected_face, &detection_count);
00121     utils::checkError(error);
00122
00123     if (detection_count == 0) {
00124       std::ostringstream oss;
00125       oss << "Error: No face detected in the image ";
00126       throw std::runtime_error(oss.str());
00127
00128     } else if (detection_count > 1) {
```

```
00129       std::ostringstream oss;
00130       oss « "Error: Found " « detection_count
00131           « " faces. Please provide an image with only one face.";
00132
00133       throw std::runtime_error(oss.str());
00134     }
00135
00136     std::cout « "Detected face in the image." « std::endl;
00137   }
00138 }
```

9.4.1.2 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     SFEError error = sfeFaceDetectInputSize(
00068         image,
00069         SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00070         face_size_min, face_size_max, &recommended_width, &recommended_height);
00071     utils::checkError(error);
00072   };
00073 }
```

9.4.1.3 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 Identify]

[Face Template Identify]

[Face Template Match]

[Face Template Match]

[Face Crop]

[Face Crop]

Examples

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

Definition at line 141 of file example_face_identify.cpp.

```
00141                                   {
00142
00143   { // STAGE 0: Parse command line arguments
00144     // Map to store argument values
00145     std::unordered_map<std::string, std::string> args;
00146
00147     // Parse command line arguments
00148     for (int i = 1; i < argc; ++i) {
00149       std::string arg = argv[i];
00150       if (arg[0] == '-') {
00151         if (arg == "-h") {
00152           printHelp();
00153           return 0;
00154         }
00155         // Check if there's a next argument and it isn't another option
00156         if (i + 1 < argc && argv[i + 1][0] != '-') {
00157           args[arg] = argv[++i];
00158         } else {
00159           std::cerr << "Option " << arg << " requires a value." << std::endl;
00160           return 1;
00161         }
00162       } else {
00163         std::cerr << "Unknown option: " << arg << std::endl;
00164         return 1;
00165       }
00166     }
00167
00168     // Assign values to variables based on parsed arguments
00169     if (args.count("-p"))
00170       image_probe = args["-p"];
00171     if (args.count("-g"))
00172       image_gallery = args["-g"];
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("-e"))
00178       solver_face_template = args["-e"];
00179     if (args.count("-t"))
00180       detection_threshold = std::stof(args["-t"]);
00181     if (args.count("-i"))
00182       identification_threshold = std::stof(args["-i"]);
00183     if (args.count("-m"))
00184       face_size_min = std::stoi(args["-m"]);
00185     if (args.count("-x"))
00186       face_size_max = std::stoi(args["-x"]);
00187 
00188     utils::printFormatted("EXAMPLE PARAMETERS");
00189     // Print resource names
00190     std::cout « "Probe image: " « image_probe « std::endl;
00191     std::cout « "Gallery image: " « image_gallery « std::endl;
00192 
00193     // Print solver names
00194     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00195     std::cout « "Face landmarks solver: " « solver_face_landmarks
00196               « std::endl;
00197     std::cout « "Face template solver: " « solver_face_template « std::endl;
00198 
00199     // Print other options
00200     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00201     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00202     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00203     std::cout « "Identification threshold: " « identification_threshold
00204               « std::endl;
00205   }
00206 
00207   utils::printToolkitInfo();
00208 
00209   SFEError error{};
00210 
00211   SFESolver detector_solver{};
00212   DEFER(sfeSolverFree(detector_solver));
00213   SFESolver landmarks_solver{};
00214   DEFER(sfeSolverFree(landmarks_solver));
00215   SFESolver template_solver{};
00216   DEFER(sfeSolverFree(template_solver));
00217   { // STAGE 1.1: loading solvers
00218     utils::printFormatted("LOADING SOLVERS");
00219     // Initialize face detection solver
00220     error = sfeSolverCreateWithParameters(
00221         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00222         SOLVER_PARAMETERS.size(), &detector_solver);
00223     utils::checkError(error);
00224 
00225     // Initialize landmarks detection solver
00226     error = sfeSolverCreateWithParameters(
00227         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00228         SOLVER_PARAMETERS.size(), &landmarks_solver);
00229     utils::checkError(error);
00230 
00231     // Initialize template extraction solver
00232     error = sfeSolverCreateWithParameters(
00233         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00234         SOLVER_PARAMETERS.size(), &template_solver);
00235     utils::checkError(error);
00236   }
00237 
00238   SFEFaceTemplate probe_face_template;
00239   { // STAGE 1: Extract probe face template
00240     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00241 
00242     SFEImage image{};
00243     DEFER(sfeImageFree(image));
00244     { // STAGE 1.1: Load image
00245       // Load image data from file
00246       auto image_data = utils::readFile(image_probe);
00247 
00248       // Decode image from data
00249       error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00250       utils::checkError(error);
00251     }
00252 
00253     SFEFaceDetect detected_face = {};
00254     { // STAGE 1.2: Detect face in the image
00255       // Detect a face in the image
00256       detectFace(image, detector_solver, detected_face);
00257     }
00258 
00259     std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00260     { // STAGE 1.3: Get face landmarks
00261       // Use landmarks detection solver to get relevant landmarks for
00262       // the detected face
00263       error = sfeFaceLandmarks(landmarks_solver, image, detected_face.bbox,
```

```
00264                                landmarks.data());
00265       utils::checkError(error);
00266     }
00267
00269     { // STAGE 1.4: Extract probe face template
00270       // Use template extraction solver to create new face
00271       // template
00272       error = sfeFaceTemplateExtract(template_solver, image, landmarks.data(),
00273                                      detected_face.raw_confidence,
00274                                      &probe_face_template);
00275       utils::checkError(error);
00276     }
00278
00279     { // STAGE 1.5: (optional) Print template attributes
00280       utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00281
00283       SFEFaceTemplateVersion version = {};
00284       error = sfeFaceTemplateVersion(&probe_face_template, &version);
00285       utils::checkError(error);
00286
00287       auto version_minor = (int)version.version_minor - (int)'0';
00288       std::cout « "Version of the probe template: " « version.version_major
00289                 « '.' « version_minor « std::endl;
00291     }
00292   }
00293
00294   std::vector<std::string> image_paths{};
00295   { // STAGE 2: Get gallery image paths
00296     // Get folder names from the face image gallery
00297     auto folder_names = utils::getFiles(image_gallery);
00298
00299     for (auto folder_name : folder_names) {
00300       auto gallery_folder = image_gallery + folder_name + "/";
00301
00302       // Get image names from folder
00303       auto image_names = utils::getFiles(gallery_folder);
00304
00305       // Extract template for each image in the folder
00306       for (auto image_name : image_names) {
00307
00308         auto image_path = gallery_folder + image_name;
00309
00310         image_paths.push_back(image_path);
00311       }
00312     }
00313   }
00314
00315   std::vector<SFEFaceTemplate> gallery_face_templates(image_paths.size());
00316   std::vector<SFEFaceDetect> detected_faces(image_paths.size());
00317   std::vector<std::array<SFEFaceLandmarks, SFE_FACE_LANDMARK_COUNT» landmarks(
00318       image_paths.size());
00319   { // STAGE 2: Load gallery image and extract face templates for each image in
00320     // the gallery
00321     utils::printFormatted("GALLERY TEMPLATE EXTRACTION");
00322
00323     for (size_t i = 0; i < image_paths.size(); i++) {
00324       std::cout « "Processing image #" « i « ", path: " « image_paths[i]
00325                 « std::endl;
00326       SFEImage image{};
00327       DEFER(sfeImageFree(image));
00328       { // STAGE 2.1: Load image
00329         // Load image data from file
00330         auto image_data = utils::readFile(image_paths[i]);
00331
00332         // Decode image from data
00333         error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00334         utils::checkError(error);
00335       }
00336
00337       { // STAGE 2.2: Detect face in the image
00338         // Detect a face in the image
00339         detectFace(image, detector_solver, detected_faces[i]);
00340       }
00341
00342       { // STAGE 2.3: Get face landmarks
00343         // Use landmarks detection solver to get relevant landmarks for
00344         // the detected face
00345         error = sfeFaceLandmarks(landmarks_solver, image,
00346                                  detected_faces[i].bbox, landmarks[i].data());
00347         utils::checkError(error);
00348       }
00349
00350       { // STAGE 2.3: Extract face template
00351         // Use template extraction solver to get face_template solver
00352         error = sfeFaceTemplateExtract(
00353             template_solver, image, landmarks[i].data(),
00354             detected_faces[i].raw_confidence, &gallery_face_templates[i]);
```

```
00355         utils::checkError(error);
00356         std::cout « "Extracted face template." « std::endl;
00357       }
00358       std::cout « std::endl;
00359     }
00360   }
00361
00363   size_t candidate_count = 1;
00364   int best_candidate_index = -1;
00365   std::vector<SFEIdentificationResult> identification_results(
00366       image_paths.size());
00367   { // STAGE 3: Identification
00368     utils::printFormatted("1:N IDENTIFICATION");
00369
00370     error = sfeFaceTemplateIdentify(
00371         &probe_face_template, gallery_face_templates.data(),
00372         gallery_face_templates.size(), identification_threshold,
00373         identification_results.data(), &candidate_count, 4);
00374     utils::checkError(error);
00375
00376     // Resize the results vector to the actual number of candidates found, in
00377     // the case there is less than candidate_count
00378     identification_results.resize(candidate_count);
00379
00380     std::cout « "Found " « identification_results.size()
00381               « " candidates above identification threshold "
00382               « identification_threshold « std::endl;
00383     for (auto &result : identification_results)
00384       std::cout « "Template index: #" « result.index
00385                 « ", score: " « result.score « std::endl;
00386
00387     // Since it's sorted vector by score, the best candidate is the first one
00388     best_candidate_index = identification_results[0].index;
00389   }
00391
00393   { // STAGE 4: (optional) 1:1 matching with top candidate to showcase 1:1
00394     // matching
00395     utils::printFormatted("1:1 MATCHING WITH TOP CANDIDATE");
00396
00397     if (identification_results.size() == 0) {
00398       std::cout « "No candidates found above identification threshold "
00399                 « identification_threshold « std::endl;
00400       return 0;
00401     }
00402
00403     auto best_candidate_template = gallery_face_templates[best_candidate_index];
00404
00405     float match_confidence;
00406     error = sfeFaceTemplateMatch(&probe_face_template, &best_candidate_template,
00407                                  &match_confidence);
00408     utils::checkError(error);
00409
00410     std::cout « "Matching score of probe template with template index #"
00411               « best_candidate_index « ", score: " « match_confidence
00412               « std::endl;
00413   }
00415
00416   { // STAGE: 5 Optional, get face crop of best face and its bounding box and
00417     // save it to file
00418     utils::printFormatted("SAVE FACE CROP AND ANNOTATED IMAGE");
00419
00420     auto best_face_index = identification_results[0].index;
00421
00422     SFEImage image{};
00423     DEFER(sfeImageFree(image));
00424     { // STAGE 5.1: Load image
00425       // Load image data from file
00426       auto image_data = utils::readFile(image_paths[best_face_index]);
00427
00428       // Decode image from data
00429       error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00430       utils::checkError(error);
00431     }
00432
00434     SFEFaceCrop crop_data = {};
00435     DEFER(sfeImageFree(crop_data.crop_image));
00436     { // STAGE 5.2: Get face crop
00437       //  Defines the size of the crop as an extension of
00438       // the detection bounding box.
00439       uint32_t face_size_extension = 2;
00440       SFEError error =
00441           sfeFaceCrop(image, landmarks[best_face_index].data(),
00442                       face_size_extension, (float)(face_size_max), &crop_data);
00443       utils::checkError(error);
00444
00445       std::cout « "Face crop extension: " « crop_data.face_size_extension
00446                 « ", width of cropped image: " « crop_data.crop_image.width
```

```
00447                 « "px, height of cropped image: "
00448                 « crop_data.crop_image.height « "px." « std::endl;
00449     }
00451
00452     { // STAGE 5.3: Save face crop to file
00453       auto png_file = std::vector<unsigned char>();
00454       size_t size =
00455           crop_data.crop_image.width * crop_data.crop_image.height * 3;
00456       png_file.resize(size);
00457       SFEError error2 = sfeImageSave(crop_data.crop_image, SFE_IMAGE_FORMAT_PNG,
00458                                      png_file.data(), &size);
00459       utils::checkError(error2);
00460
00461       std::cout « "Face crop saved as crop_identified_person.png" « std::endl;
00462       utils::saveFile("crop_identified_person.png", png_file);
00463     }
00464
00465     { // STAGE 5.4: Annotate the image
00466       // Render bounding box with detection confidence and identification score
00467       std::stringstream label;
00468       label « " D:" « std::fixed « std::setprecision(2)
00469             « detected_faces[best_face_index].confidence
00470             « " M:" « identification_results[0].score;
00471
00472       auto color = annotate::RED;
00473
00474       annotate::labelBox(image, label.str(),
00475                          detected_faces[best_candidate_index].bbox,
00476                          annotate::WHITE, color);
00477
00478       // Render landmarks
00479       for (auto &landmark : landmarks[best_face_index]) {
00480         auto x = landmark.x * image.width;
00481         auto y = landmark.y * image.height;
00482         annotate::circle(image, x, y, 3, annotate::YELLOW);
00483       }
00484
00485       // Save annotated image
00486       size_t size = image.width * image.height * 3;
00487       auto png_file = std::vector<unsigned char>(size);
00488       error = sfeImageSave(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00489       utils::checkError(error);
00490       png_file.resize(size);
00491       utils::saveFile("face_identify.png", png_file);
00492
00493       std::cout « std::endl;
00494       std::cout « "Annotated image saved as face_identify.png" « std::endl;
00495     }
00496   }
00497
00498   utils::printFormatted("FINISHED");
00499 }
```

9.4.1.4 printHelp()

```
void printHelp ( )
```

Examples

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

Definition at line 75 of file example_face_identify.cpp.

```
00075                 {
00076   std::cout « "Help: Usage of the program." « std::endl;
00077   std::cout « "Options:" « std::endl;
00078   std::cout « "-h: Display help." « std::endl;
00079   std::cout « "-p: Probe image file." « std::endl;
00080   std::cout « "-g: Gallery folder." « std::endl;
00081   std::cout « "-d: Path to detector solver." « std::endl;
00082   std::cout « "-l: Path to landmarks 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   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00087   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00088 }
```

9.4.2 Variable Documentation

9.4.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Examples

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

Definition at line 33 of file example_face_identify.cpp.

9.4.2.2 face_size_max

```
size_t face_size_max = 170
```

Examples

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.

9.4.2.3 face_size_min

```
size_t face_size_min = 28
```

Required size of the face to be detected.

Examples

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.

9.4.2.4 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.

9.4.2.5 image_gallery

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

Examples

example_face_identify.cpp.

Definition at line 18 of file example_face_identify.cpp.

9.4.2.6 image_probe

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

Examples

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

Definition at line 20 of file example_face_identify.cpp.

9.4.2.7 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_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.

9.4.2.8 solver_face_landmarks

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

Examples

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

Definition at line 24 of file example_face_identify.cpp.

9.4.2.9 solver_face_template

```
std::string solver_face_template = SOLVER_FACE_TEMPLATE
```

Examples

example_face_identify.cpp, and example_face_identify_entity.cpp.

Definition at line 25 of file example_face_identify.cpp.

9.4.2.10 SOLVER_PARAMETERS

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

Examples

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

Definition at line 27 of file example_face_identify.cpp.

9.5 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_TEMPLATE;
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     SFEError error = sfeFaceDetectInputSize(
00068         image,
00069         SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00070         face_size_min, face_size_max, &recommended_width, &recommended_height);
00071     utils::checkError(error);
00072   };
00073 }
00074
00075 void printHelp() {
00076   std::cout « "Help: Usage of the program." « std::endl;
00077   std::cout « "Options:" « std::endl;
00078   std::cout « "-h: Display help." « std::endl;
00079   std::cout « "-p: Probe image file." « std::endl;
00080   std::cout « "-g: Gallery folder." « std::endl;
00081   std::cout « "-d: Path to detector solver." « std::endl;
00082   std::cout « "-l: Path to landmarks 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   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00087   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00088 }
00089
00090 void detectFace(SFEImage &image, SFESolver &detector_solver,
00091                 SFEFaceDetect &detected_face) {
00092
00093   SFEImage resized_image{};
00094   DEFER(sfeImageFree(resized_image));
00095   SFEError error{};
00096   { // STAGE 1 Load image
00097     size_t recommended_width{};
00098     size_t recommended_height{};
00099
00100     // Calculate optimal input image size for face detection. Image will be
00101     // resized to recommended size for optimal performance.
00102     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00103                             face_size_max, recommended_width,
00104                             recommended_height);
00105
00106     // Resize image to recommended size
00107     // NOTE: This function will resize the image without preserving the aspect
00108     // ratio of the image.
00109
00110     error = sfeImageResize(image, recommended_width, recommended_height,
00111                            &resized_image);
00112     utils::checkError(error);
00113   }
00114
00115   size_t detection_count = 1;
00116   { // STAGE 2: Detect face in the image
00117
00118     // Detect face in the image
00119     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00120                           &detected_face, &detection_count);
00121     utils::checkError(error);
00122
00123     if (detection_count == 0) {
00124       std::ostringstream oss;
00125       oss « "Error: No face detected in the image ";
00126       throw std::runtime_error(oss.str());
00127
00128     } else if (detection_count > 1) {
00129       std::ostringstream oss;
00130       oss « "Error: Found " « detection_count
00131           « " faces. Please provide an image with only one face.";
00132
00133       throw std::runtime_error(oss.str());
00134     }
00135
00136     std::cout « "Detected face in the image." « std::endl;
00137   }
00138 }
00139
00141 int main(int argc, char *argv[]) {
```

```
00142
00143   { // STAGE 0: Parse command line arguments
00144     // Map to store argument values
00145     std::unordered_map<std::string, std::string> args;
00146
00147     // Parse command line arguments
00148     for (int i = 1; i < argc; ++i) {
00149       std::string arg = argv[i];
00150       if (arg[0] == '-') {
00151         if (arg == "-h") {
00152           printHelp();
00153           return 0;
00154         }
00155         // Check if there's a next argument and it isn't another option
00156         if (i + 1 < argc && argv[i + 1][0] != '-') {
00157           args[arg] = argv[++i];
00158         } else {
00159           std::cerr << "Option " << arg << " requires a value." << std::endl;
00160           return 1;
00161         }
00162       } else {
00163         std::cerr << "Unknown option: " << arg << std::endl;
00164         return 1;
00165       }
00166     }
00167
00168     // Assign values to variables based on parsed arguments
00169     if (args.count("-p"))
00170       image_probe = args["-p"];
00171     if (args.count("-g"))
00172       image_gallery = args["-g"];
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("-e"))
00178       solver_face_template = args["-e"];
00179     if (args.count("-t"))
00180       detection_threshold = std::stof(args["-t"]);
00181     if (args.count("-i"))
00182       identification_threshold = std::stof(args["-i"]);
00183     if (args.count("-m"))
00184       face_size_min = std::stoi(args["-m"]);
00185     if (args.count("-x"))
00186       face_size_max = std::stoi(args["-x"]);
00187
00188     utils::printFormatted("EXAMPLE PARAMETERS");
00189     // Print resource names
00190     std::cout << "Probe image: " << image_probe << std::endl;
00191     std::cout << "Gallery image: " << image_gallery << std::endl;
00192
00193     // Print solver names
00194     std::cout << "Face detection solver: " << solver_face_detect << std::endl;
00195     std::cout << "Face landmarks solver: " << solver_face_landmarks
00196               << std::endl;
00197     std::cout << "Face template solver: " << solver_face_template << std::endl;
00198
00199     // Print other options
00200     std::cout << "Min face size [px]: " << face_size_min << std::endl;
00201     std::cout << "Max face size [px]: " << face_size_max << std::endl;
00202     std::cout << "Detection threshold: " << detection_threshold << std::endl;
00203     std::cout << "Identification threshold: " << identification_threshold
00204               << std::endl;
00205   }
00206
00207   utils::printToolkitInfo();
00208
00209   SFEError error{};
00210
00211   SFESolver detector_solver{};
00212   DEFER(sfeSolverFree(detector_solver));
00213   SFESolver landmarks_solver{};
00214   DEFER(sfeSolverFree(landmarks_solver));
00215   SFESolver template_solver{};
00216   DEFER(sfeSolverFree(template_solver));
00217   { // STAGE 1.1: loading solvers
00218     utils::printFormatted("LOADING SOLVERS");
00219     // Initialize face detection solver
00220     error = sfeSolverCreateWithParameters(
00221         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00222         SOLVER_PARAMETERS.size(), &detector_solver);
00223     utils::checkError(error);
00224
00225     // Initialize landmarks detection solver
00226     error = sfeSolverCreateWithParameters(
00227         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00228         SOLVER_PARAMETERS.size(), &landmarks_solver);
```

```
00229     utils::checkError(error);
00230
00231     // Initialize template extraction solver
00232     error = sfeSolverCreateWithParameters(
00233         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00234         SOLVER_PARAMETERS.size(), &template_solver);
00235     utils::checkError(error);
00236   }
00237
00238   SFEFaceTemplate probe_face_template;
00239   { // STAGE 1: Extract probe face template
00240     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00241
00242     SFEImage image{};
00243     DEFER(sfeImageFree(image));
00244     { // STAGE 1.1: Load image
00245       // Load image data from file
00246       auto image_data = utils::readFile(image_probe);
00247
00248       // Decode image from data
00249       error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00250       utils::checkError(error);
00251     }
00252
00253     SFEFaceDetect detected_face = {};
00254     { // STAGE 1.2: Detect face in the image
00255       // Detect a face in the image
00256       detectFace(image, detector_solver, detected_face);
00257     }
00258
00259     std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00260     { // STAGE 1.3: Get face landmarks
00261       // Use landmarks detection solver to get relevant landmarks for
00262       // the detected face
00263       error = sfeFaceLandmarks(landmarks_solver, image, detected_face.bbox,
00264                                landmarks.data());
00265       utils::checkError(error);
00266     }
00267
00269     { // STAGE 1.4: Extract probe face template
00270       // Use template extraction solver to create new face
00271       // template
00272       error = sfeFaceTemplateExtract(template_solver, image, landmarks.data(),
00273                                      detected_face.raw_confidence,
00274                                      &probe_face_template);
00275       utils::checkError(error);
00276     }
00278
00279     { // STAGE 1.5: (optional) Print template attributes
00280       utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00281
00283       SFEFaceTemplateVersion version = {};
00284       error = sfeFaceTemplateVersion(&probe_face_template, &version);
00285       utils::checkError(error);
00286
00287       auto version_minor = (int)version.version_minor - (int)'0';
00288       std::cout « "Version of the probe template: " « version.version_major
00289                 « '.' « version_minor « std::endl;
00291     }
00292   }
00293
00294   std::vector<std::string> image_paths{};
00295   { // STAGE 2: Get gallery image paths
00296     // Get folder names from the face image gallery
00297     auto folder_names = utils::getFiles(image_gallery);
00298
00299     for (auto folder_name : folder_names) {
00300       auto gallery_folder = image_gallery + folder_name + "/";
00301
00302       // Get image names from folder
00303       auto image_names = utils::getFiles(gallery_folder);
00304
00305       // Extract template for each image in the folder
00306       for (auto image_name : image_names) {
00307
00308         auto image_path = gallery_folder + image_name;
00309
00310         image_paths.push_back(image_path);
00311       }
00312     }
00313   }
00314
00315   std::vector<SFEFaceTemplate> gallery_face_templates(image_paths.size());
00316   std::vector<SFEFaceDetect> detected_faces(image_paths.size());
00317   std::vector<std::array<SFEFaceLandmarks, SFE_FACE_LANDMARK_COUNT» landmarks(
00318       image_paths.size());
00319   { // STAGE 2: Load gallery image and extract face templates for each image in
```

```
00320     // the gallery
00321     utils::printFormatted("GALLERY TEMPLATE EXTRACTION");
00322
00323     for (size_t i = 0; i < image_paths.size(); i++) {
00324       std::cout « "Processing image #" « i « ", path: " « image_paths[i]
00325                « std::endl;
00326       SFEImage image{};
00327       DEFER(sfeImageFree(image));
00328       { // STAGE 2.1: Load image
00329         // Load image data from file
00330         auto image_data = utils::readFile(image_paths[i]);
00331
00332         // Decode image from data
00333         error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00334         utils::checkError(error);
00335       }
00336
00337       { // STAGE 2.2: Detect face in the image
00338         // Detect a face in the image
00339         detectFace(image, detector_solver, detected_faces[i]);
00340       }
00341
00342       { // STAGE 2.3: Get face landmarks
00343         // Use landmarks detection solver to get relevant landmarks for
00344         // the detected face
00345         error = sfeFaceLandmarks(landmarks_solver, image,
00346                                  detected_faces[i].bbox, landmarks[i].data());
00347         utils::checkError(error);
00348       }
00349
00350       { // STAGE 2.3: Extract face template
00351         // Use template extraction solver to get face_template solver
00352         error = sfeFaceTemplateExtract(
00353             template_solver, image, landmarks[i].data(),
00354             detected_faces[i].raw_confidence, &gallery_face_templates[i]);
00355         utils::checkError(error);
00356         std::cout « "Extracted face template." « std::endl;
00357       }
00358       std::cout « std::endl;
00359     }
00360   }
00361
00363   size_t candidate_count = 1;
00364   int best_candidate_index = -1;
00365   std::vector<SFEIdentificationResult> identification_results(
00366       image_paths.size());
00367   { // STAGE 3: Identification
00368     utils::printFormatted("1:N IDENTIFICATION");
00369
00370     error = sfeFaceTemplateIdentify(
00371         &probe_face_template, gallery_face_templates.data(),
00372         gallery_face_templates.size(), identification_threshold,
00373         identification_results.data(), &candidate_count, 4);
00374     utils::checkError(error);
00375
00376     // Resize the results vector to the actual number of candidates found, in
00377     // the case there is less than candidate_count
00378     identification_results.resize(candidate_count);
00379
00380     std::cout « "Found " « identification_results.size()
00381               « " candidates above identification threshold "
00382               « identification_threshold « std::endl;
00383     for (auto &result : identification_results)
00384       std::cout « "Template index: #" « result.index
00385                 « ", score: " « result.score « std::endl;
00386
00387     // Since it's sorted vector by score, the best candidate is the first one
00388     best_candidate_index = identification_results[0].index;
00389   }
00391
00393   { // STAGE 4: (optional) 1:1 matching with top candidate to showcase 1:1
00394     // matching
00395     utils::printFormatted("1:1 MATCHING WITH TOP CANDIDATE");
00396
00397     if (identification_results.size() == 0) {
00398       std::cout « "No candidates found above identification threshold "
00399                 « identification_threshold « std::endl;
00400       return 0;
00401     }
00402
00403     auto best_candidate_template = gallery_face_templates[best_candidate_index];
00404
00405     float match_confidence;
00406     error = sfeFaceTemplateMatch(&probe_face_template, &best_candidate_template,
00407                                  &match_confidence);
00408     utils::checkError(error);
00409
```

```
00410     std::cout « "Matching score of probe template with template index #"
00411               « best_candidate_index « ", score: " « match_confidence
00412               « std::endl;
00413   }
00415
00416   { // STAGE: 5 Optional, get face crop of best face and its bounding box and
00417     // save it to file
00418     utils::printFormatted("SAVE FACE CROP AND ANNOTATED IMAGE");
00419
00420     auto best_face_index = identification_results[0].index;
00421
00422     SFEImage image{};
00423     DEFER(sfeImageFree(image));
00424     { // STAGE 5.1: Load image
00425       // Load image data from file
00426       auto image_data = utils::readFile(image_paths[best_face_index]);
00427
00428       // Decode image from data
00429       error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00430       utils::checkError(error);
00431     }
00432
00434     SFEFaceCrop crop_data = {};
00435     DEFER(sfeImageFree(crop_data.crop_image));
00436     { // STAGE 5.2: Get face crop
00437       //  Defines the size of the crop as an extension of
00438       // the detection bounding box.
00439       uint32_t face_size_extension = 2;
00440       SFEError error =
00441           sfeFaceCrop(image, landmarks[best_face_index].data(),
00442                       face_size_extension, (float)(face_size_max), &crop_data);
00443       utils::checkError(error);
00444
00445       std::cout « "Face crop extension: " « crop_data.face_size_extension
00446                 « ", width of cropped image: " « crop_data.crop_image.width
00447                 « "px, height of cropped image: "
00448                 « crop_data.crop_image.height « "px." « std::endl;
00449     }
00451
00452     { // STAGE 5.3: Save face crop to file
00453       auto png_file = std::vector<unsigned char>();
00454       size_t size =
00455           crop_data.crop_image.width * crop_data.crop_image.height * 3;
00456       png_file.resize(size);
00457       SFEError error2 = sfeImageSave(crop_data.crop_image, SFE_IMAGE_FORMAT_PNG,
00458                                      png_file.data(), &size);
00459       utils::checkError(error2);
00460
00461       std::cout « "Face crop saved as crop_identified_person.png" « std::endl;
00462       utils::saveFile("crop_identified_person.png", png_file);
00463     }
00464
00465     { // STAGE 5.4: Annotate the image
00466       // Render bounding box with detection confidence and identification score
00467       std::stringstream label;
00468       label « " D:" « std::fixed « std::setprecision(2)
00469             « detected_faces[best_face_index].confidence
00470             « " M:" « identification_results[0].score;
00471
00472       auto color = annotate::RED;
00473
00474       annotate::labelBox(image, label.str(),
00475                          detected_faces[best_candidate_index].bbox,
00476                          annotate::WHITE, color);
00477
00478       // Render landmarks
00479       for (auto &landmark : landmarks[best_face_index]) {
00480         auto x = landmark.x * image.width;
00481         auto y = landmark.y * image.height;
00482         annotate::circle(image, x, y, 3, annotate::YELLOW);
00483       }
00484
00485       // Save annotated image
00486       size_t size = image.width * image.height * 3;
00487       auto png_file = std::vector<unsigned char>(size);
00488       error = sfeImageSave(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00489       utils::checkError(error);
00490       png_file.resize(size);
00491       utils::saveFile("face_identify.png", png_file);
00492
00493       std::cout « std::endl;
00494       std::cout « "Annotated image saved as face_identify.png" « std::endl;
00495     }
00496   }
00497
00498   utils::printFormatted("FINISHED");
00499 }
```

9.6 D:/glab/da0a89/1/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_TEMPLATE
- 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

9.6.1 Function Documentation

9.6.1.1 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 72 of file example_face_identify_entity.cpp.

```
00075                                                                    {
00076
00077   SFEImage image{};
00078   DEFER(sfeImageFree(image));
00079   SFEImage resized_image{};
00080   DEFER(sfeImageFree(resized_image));
00081   SFEError error{};
00082   { // STAGE 1 Load image
00083     // Load image data from file
00084     auto image_data = utils::readFile(image_path);
00085
00086     // Decode image from data
00087     error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00088     utils::checkError(error);
00089
00090     size_t recommended_width{};
00091     size_t recommended_height{};
00092
00093     // Calculate optimal input image size for face detection. Image will be
00094     // resized to recommended size for optimal performance.
00095     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00096                             face_size_max, recommended_width,
00097                             recommended_height);
00098
00099     // Resize image to recommended size
00100     // NOTE: This function will resize the image without preserving the aspect
00101     // ratio of the image.
00102
00103     error = sfeImageResize(image, recommended_width, recommended_height,
00104                            &resized_image);
00105     utils::checkError(error);
00106   }
00107
00108   SFEFaceDetect detected_face = {};
00109   size_t detection_count = 1;
00110   { // STAGE 3: Detect face in the image
00111
00112     // Detect face in the image
00113     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00114                           &detected_face, &detection_count);
00115     utils::checkError(error);
00116
00117     if (detection_count == 0) {
00118       throw std::runtime_error("No face detected in the image.");
00119     }
00120   }
00121
00122   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00123   { // STAGE 4: Get face landmarks
00124
00125     // Use landmarks detection solver to get relevant landmarks for
00126     // the detected face
00127     error = sfeFaceLandmarks(landmarks_solver, image, detected_face.bbox,
00128                              landmarks.data());
00129     utils::checkError(error);
00130   }
00131
00132   SFEFaceTemplate face_template = {};
00133   { // STAGE 5: Extract probe face template
00134     // Use template extraction solver to create new face
00135     // template
00136     error =
00137         sfeFaceTemplateExtract(template_solver, image, landmarks.data(),
00138                                detected_face.raw_confidence, &face_template);
00139     utils::checkError(error);
00140   }
00141   return face_template;
00142 }
```

9.6.1.2 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     SFEError error = sfeFaceDetectInputSize(image,
      SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, face_size_min,
00066                                             face_size_max, &recommended_width,
00067                                             &recommended_height);
00068     utils::checkError(error);
00069   };
00070 }
```

9.6.1.3 main()

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

[Face Entity Identify]

[Face Entity Identify]

Definition at line 159 of file example_face_identify_entity.cpp.

```
00159                                    {
00160
00161   { // STAGE 0: Parse command line arguments
00162
00163     // Map to store argument values
00164     std::unordered_map<std::string, std::string> args;
00165
00166     // Parse command line arguments
00167     for (int i = 1; i < argc; ++i) {
00168       std::string arg = argv[i];
00169       if (arg[0] == '-') {
00170         if (arg == "-h") {
00171           printHelp();
00172           return 0;
00173         }
00174         // Check if there's a next argument and it isn't another option
00175         if (i + 1 < argc && argv[i + 1][0] != '-') {
00176           args[arg] = argv[++i];
00177         } else {
00178           std::cerr « "Option " « arg « " requires a value." « std::endl;
00179           return 1;
00180         }
```

```
00181       } else {
00182         std::cerr « "Unknown option: " « arg « std::endl;
00183         return 1;
00184       }
00185     }
00186
00187     // Assign values to variables based on parsed arguments
00188     if (args.count("-p"))
00189       image_probe = args["-p"];
00190     if (args.count("-g"))
00191       gallery = args["-g"];
00192     if (args.count("-d"))
00193       solver_face_detect = args["-d"];
00194     if (args.count("-l"))
00195       solver_face_landmarks = args["-l"];
00196     if (args.count("-e"))
00197       solver_face_template = args["-e"];
00198     if (args.count("-t"))
00199       detection_threshold = std::stof(args["-t"]);
00200     if (args.count("-i"))
00201       identification_threshold = std::stof(args["-i"]);
00202     if (args.count("-m"))
00203       face_size_min = std::stoi(args["-m"]);
00204     if (args.count("-x"))
00205       face_size_max = std::stoi(args["-x"]);
00206
00207     utils::printFormatted("EXAMPLE PARAMETERS");
00208     // Print resource names
00209     std::cout « "Probe image: " « image_probe « std::endl;
00210     std::cout « "Entities folder: " « gallery « std::endl;
00211
00212     // Print solver names
00213     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00214     std::cout « "Face landmarks solver: " « solver_face_landmarks
00215               « std::endl;
00216     std::cout « "Face template solver: " « solver_face_template « std::endl;
00217
00218     // Print other options
00219     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00220     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00221     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00222     std::cout « "Identification threshold: " « identification_threshold
00223               « std::endl;
00224   }
00225
00226   utils::printToolkitInfo();
00227
00228   SFEError error{};
00229
00230   SFESolver detector_solver{};
00231   DEFER(sfeSolverFree(detector_solver));
00232   SFESolver landmarks_solver{};
00233   DEFER(sfeSolverFree(landmarks_solver));
00234   SFESolver template_solver{};
00235   DEFER(sfeSolverFree(template_solver));
00236   { // STAGE 0: loading solvers
00237     // Initialize face detection solver
00238     error = sfeSolverCreateWithParameters(
00239         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00240         SOLVER_PARAMETERS.size(), &detector_solver);
00241     utils::checkError(error);
00242
00243     // Initialize landmarks detection solver
00244     error = sfeSolverCreateWithParameters(
00245         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00246         SOLVER_PARAMETERS.size(), &landmarks_solver);
00247     utils::checkError(error);
00248
00249     // Initialize template extraction solver
00250     error = sfeSolverCreateWithParameters(
00251         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00252         SOLVER_PARAMETERS.size(), &template_solver);
00253     utils::checkError(error);
00254   }
00255
00256   SFEFaceTemplate probe_face_template;
00257   { // STAGE 1: Extract probe face template
00258     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00259     probe_face_template = extractTemplate(image_probe, detector_solver,
00260                                           landmarks_solver, template_solver);
00261
00262     std::cout « "Probe face template extracted." « std::endl;
00263   }
00264
00265   std::vector<SFEFaceTemplate> gallery_face_templates = {};
00266   std::vector<SFEFaceEntity> enitity_pairs = {};
00267   { // STAGE 2: Extract templates for each image in entity folder and associate them
```

```
00268     // with entity(UUID)
00269
00270     // Entities(UUID) tells the ownership of the templates in the gallery.
00271     // For example:
00272     // UUID1 -> template1
00273     // UUID1 -> template2
00274     // UUID2 -> template3
00275     // UUID2 -> template4
00276     // ...
00277     // UUUIDN -> templateN
00278
00279     utils::printFormatted("ENTITIES TEMPLATE EXTRACTION");
00280
00281     // Get folder names from entities_gallery
00282     auto folder_names = utils::getFiles(gallery);
00283
00284     for (auto folder_name : folder_names) {
00285       if (folder_name.find(".jpg") != std::string::npos ||
00286           folder_name.find(".png") != std::string::npos) {
00287         continue;
00288       }
00289
00290       // Genereate UUID for each entity.
00291       auto entity = utils::generateEntity();
00292
00293       auto entity_folder = gallery + folder_name + "/";
00294
00295       std::cout « "Folder: " « entity_folder « ", entity UUID: ["
00296                 « static_cast<int>(entity.uuid[0]) « ", "
00297                 « static_cast<int>(entity.uuid[1]) « "..."
00298                 « static_cast<int>(entity.uuid[15]) « "]" « std::endl;
00299
00300       // Get image names from folder
00301       auto image_names = utils::getFiles(entity_folder);
00302
00303       // Extract template for each image in the folder
00304       for (auto image_name : image_names) {
00305
00306         auto image_path = entity_folder + image_name;
00307
00308         auto face_template = extractTemplate(
00309             image_path, detector_solver, landmarks_solver, template_solver);
00310
00311         gallery_face_templates.push_back(face_template);
00312         enitity_pairs.push_back(entity);
00313       }
00314     }
00315   }
00316
00318   size_t candidate_count = 1;
00319   std::vector<SFEFaceEntityIdentificationCandidate> results(candidate_count);
00320   int best_candidate_index = -1;
00321   { // STAGE 3: Identification with entities.
00322     // Probe template is matched against every template in the gallery.
00323     // Function returns entity(UUID) which owns the best matching template.
00324
00325     utils::printFormatted("1:N IDENTIFICATION WITH ENTITIES");
00326     SFEError error = sfeFaceEntityIdentify(
00327         &probe_face_template, gallery_face_templates.data(),
00328         enitity_pairs.data(), gallery_face_templates.size(),
00329         identification_threshold, results.data(), &candidate_count, 4);
00330     utils::checkError(error);
00331
00332     if (candidate_count == 0) {
00333       std::cout « "No candidates found. Exiting.." « std::endl;
00334       return 0;
00335     }
00336
00337     std::cout « std::endl;
00338     std::cout « "Found " « results.size() « " candidates " « std::endl;
00339     for (int i = 0; i < results.size(); i++) {
00340       std::cout « "Index: " « i « ", Score: " « results[i].score
00341                 « std::endl;
00342       std::cout « "Entity UUID: ["
00343                 « static_cast<int>(results[i].entity.uuid[0]) « ", "
00344                 « static_cast<int>(results[i].entity.uuid[1]) « "..."
00345                 « static_cast<int>(results[i].entity.uuid[15]) « "]"
00346                 « std::endl;
00347     }
00348   }
00350
00351   utils::printFormatted("FINISHED");
00352 }
```

9.6.1.4 printHelp()

```
void printHelp ( )
```

Definition at line 144 of file example_face_identify_entity.cpp.

```
00144                 {
00145   std::cout « "Help: Usage of the program." « std::endl;
00146   std::cout « "Options:" « std::endl;
00147   std::cout « "-h: Display help." « std::endl;
00148   std::cout « "-p: Probe image file." « std::endl;
00149   std::cout « "-g: Path to folder with entities." « std::endl;
00150   std::cout « "-d: Path to detector solver." « std::endl;
00151   std::cout « "-l: Path to landmarks solver." « std::endl;
00152   std::cout « "-e: Path to extraction solver." « std::endl;
00153   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00154   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00155   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00156   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00157 }
```

9.6.2 Variable Documentation

9.6.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 30 of file example_face_identify_entity.cpp.

9.6.2.2 face_size_max

```
size_t face_size_max = 170
```

Definition at line 28 of file example_face_identify_entity.cpp.

9.6.2.3 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.

9.6.2.4 gallery

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

Examples

example_face_identify_entity.cpp.

Definition at line 15 of file example_face_identify_entity.cpp.

9.6.2.5 identification_threshold

```
float identification_threshold = 0.6f
```

Definition at line 31 of file example_face_identify_entity.cpp.

9.6.2.6 image_probe

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

Definition at line 17 of file example_face_identify_entity.cpp.

9.6.2.7 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.

9.6.2.8 solver_face_landmarks

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

Definition at line 21 of file example_face_identify_entity.cpp.

9.6.2.9 solver_face_template

```
std::string solver_face_template = SOLVER_FACE_TEMPLATE
```

Definition at line 22 of file example_face_identify_entity.cpp.

9.6.2.10 SOLVER_PARAMETERS

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

Definition at line 24 of file example_face_identify_entity.cpp.

9.7 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_TEMPLATE;
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     SFEError error = sfeFaceDetectInputSize(image,
      SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, face_size_min,
00066                                             face_size_max, &recommended_width,
00067                                             &recommended_height);
00068     utils::checkError(error);
00069   };
00070 }
00071
00072 SFEFaceTemplate extractTemplate(std::string image_path,
00073                                 SFESolver &detector_solver,
00074                                 SFESolver &landmarks_solver,
00075                                 SFESolver &template_solver) {
00076
00077   SFEImage image{};
00078   DEFER(sfeImageFree(image));
00079   SFEImage resized_image{};
00080   DEFER(sfeImageFree(resized_image));
00081   SFEError error{};
00082   { // STAGE 1 Load image
00083     // Load image data from file
00084     auto image_data = utils::readFile(image_path);
00085
00086     // Decode image from data
00087     error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00088     utils::checkError(error);
00089
00090     size_t recommended_width{};
00091     size_t recommended_height{};
00092
```

```
00093     // Calculate optimal input image size for face detection. Image will be
00094     // resized to recommended size for optimal performance.
00095     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00096                             face_size_max, recommended_width,
00097                             recommended_height);
00098
00099     // Resize image to recommended size
00100     // NOTE: This function will resize the image without preserving the aspect
00101     // ratio of the image.
00102
00103     error = sfeImageResize(image, recommended_width, recommended_height,
00104                            &resized_image);
00105     utils::checkError(error);
00106   }
00107
00108   SFEFaceDetect detected_face = {};
00109   size_t detection_count = 1;
00110   { // STAGE 3: Detect face in the image
00111
00112     // Detect face in the image
00113     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00114                           &detected_face, &detection_count);
00115     utils::checkError(error);
00116
00117     if (detection_count == 0) {
00118       throw std::runtime_error("No face detected in the image.");
00119     }
00120   }
00121
00122   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00123   { // STAGE 4: Get face landmarks
00124
00125     // Use landmarks detection solver to get relevant landmarks for
00126     // the detected face
00127     error = sfeFaceLandmarks(landmarks_solver, image, detected_face.bbox,
00128                              landmarks.data());
00129     utils::checkError(error);
00130   }
00131
00132   SFEFaceTemplate face_template = {};
00133   { // STAGE 5: Extract probe face template
00134     // Use template extraction solver to create new face
00135     // template
00136     error =
00137         sfeFaceTemplateExtract(template_solver, image, landmarks.data(),
00138                                detected_face.raw_confidence, &face_template);
00139     utils::checkError(error);
00140   }
00141   return face_template;
00142 }
00143
00144 void printHelp() {
00145   std::cout « "Help: Usage of the program." « std::endl;
00146   std::cout « "Options:" « std::endl;
00147   std::cout « "-h: Display help." « std::endl;
00148   std::cout « "-p: Probe image file." « std::endl;
00149   std::cout « "-g: Path to folder with entities." « std::endl;
00150   std::cout « "-d: Path to detector solver." « std::endl;
00151   std::cout « "-l: Path to landmarks solver." « std::endl;
00152   std::cout « "-e: Path to extraction solver." « std::endl;
00153   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00154   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00155   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00156   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00157 }
00158
00159 int main(int argc, char *argv[]) {
00160
00161   { // STAGE 0: Parse command line arguments
00162
00163     // Map to store argument values
00164     std::unordered_map<std::string, std::string> args;
00165
00166     // Parse command line arguments
00167     for (int i = 1; i < argc; ++i) {
00168       std::string arg = argv[i];
00169       if (arg[0] == '-') {
00170         if (arg == "-h") {
00171           printHelp();
00172           return 0;
00173         }
00174         // Check if there's a next argument and it isn't another option
00175         if (i + 1 < argc && argv[i + 1][0] != '-') {
00176           args[arg] = argv[++i];
00177         } else {
00178           std::cerr « "Option " « arg « " requires a value." « std::endl;
00179           return 1;
```

```
00180         }
00181       } else {
00182         std::cerr « "Unknown option: " « arg « std::endl;
00183         return 1;
00184       }
00185     }
00186
00187     // Assign values to variables based on parsed arguments
00188     if (args.count("-p"))
00189       image_probe = args["-p"];
00190     if (args.count("-g"))
00191       gallery = args["-g"];
00192     if (args.count("-d"))
00193       solver_face_detect = args["-d"];
00194     if (args.count("-l"))
00195       solver_face_landmarks = args["-l"];
00196     if (args.count("-e"))
00197       solver_face_template = args["-e"];
00198     if (args.count("-t"))
00199       detection_threshold = std::stof(args["-t"]);
00200     if (args.count("-i"))
00201       identification_threshold = std::stof(args["-i"]);
00202     if (args.count("-m"))
00203       face_size_min = std::stoi(args["-m"]);
00204     if (args.count("-x"))
00205       face_size_max = std::stoi(args["-x"]);
00206
00207     utils::printFormatted("EXAMPLE PARAMETERS");
00208     // Print resource names
00209     std::cout « "Probe image: " « image_probe « std::endl;
00210     std::cout « "Entities folder: " « gallery « std::endl;
00211
00212     // Print solver names
00213     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00214     std::cout « "Face landmarks solver: " « solver_face_landmarks
00215               « std::endl;
00216     std::cout « "Face template solver: " « solver_face_template « std::endl;
00217
00218     // Print other options
00219     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00220     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00221     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00222     std::cout « "Identification threshold: " « identification_threshold
00223               « std::endl;
00224   }
00225
00226   utils::printToolkitInfo();
00227
00228   SFEError error{};
00229
00230   SFESolver detector_solver{};
00231   DEFER(sfeSolverFree(detector_solver));
00232   SFESolver landmarks_solver{};
00233   DEFER(sfeSolverFree(landmarks_solver));
00234   SFESolver template_solver{};
00235   DEFER(sfeSolverFree(template_solver));
00236   { // STAGE 0: loading solvers
00237     // Initialize face detection solver
00238     error = sfeSolverCreateWithParameters(
00239         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00240         SOLVER_PARAMETERS.size(), &detector_solver);
00241     utils::checkError(error);
00242
00243     // Initialize landmarks detection solver
00244     error = sfeSolverCreateWithParameters(
00245         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00246         SOLVER_PARAMETERS.size(), &landmarks_solver);
00247     utils::checkError(error);
00248
00249     // Initialize template extraction solver
00250     error = sfeSolverCreateWithParameters(
00251         solver_face_template.c_str(), SOLVER_PARAMETERS.data(),
00252         SOLVER_PARAMETERS.size(), &template_solver);
00253     utils::checkError(error);
00254   }
00255
00256   SFEFaceTemplate probe_face_template;
00257   { // STAGE 1: Extract probe face template
00258     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00259     probe_face_template = extractTemplate(image_probe, detector_solver,
00260                                           landmarks_solver, template_solver);
00261
00262     std::cout « "Probe face template extracted." « std::endl;
00263   }
00264
00265   std::vector<SFEFaceTemplate> gallery_face_templates = {};
00266   std::vector<SFEFaceEntity> enitity_pairs = {};
```

```
00267   { // STAGE 2: Extract templates for each image in entity folder and associate them
00268     // with entity(UUID)
00269
00270     // Entities(UUID) tells the ownership of the templates in the gallery.
00271     // For example:
00272     // UUID1 -> template1
00273     // UUID1 -> template2
00274     // UUID2 -> template3
00275     // UUID2 -> template4
00276     // ...
00277     // UUUIDN -> templateN
00278
00279     utils::printFormatted("ENTITIES TEMPLATE EXTRACTION");
00280
00281     // Get folder names from entities_gallery
00282     auto folder_names = utils::getFiles(gallery);
00283
00284     for (auto folder_name : folder_names) {
00285       if (folder_name.find(".jpg") != std::string::npos ||
00286           folder_name.find(".png") != std::string::npos) {
00287         continue;
00288       }
00289
00290       // Genereate UUID for each entity.
00291       auto entity = utils::generateEntity();
00292
00293       auto entity_folder = gallery + folder_name + "/";
00294
00295       std::cout « "Folder: " « entity_folder « ", entity UUID: ["
00296                 « static_cast<int>(entity.uuid[0]) « ", "
00297                 « static_cast<int>(entity.uuid[1]) « "..."
00298                 « static_cast<int>(entity.uuid[15]) « "]" « std::endl;
00299
00300       // Get image names from folder
00301       auto image_names = utils::getFiles(entity_folder);
00302
00303       // Extract template for each image in the folder
00304       for (auto image_name : image_names) {
00305
00306         auto image_path = entity_folder + image_name;
00307
00308         auto face_template = extractTemplate(
00309             image_path, detector_solver, landmarks_solver, template_solver);
00310
00311         gallery_face_templates.push_back(face_template);
00312         enitity_pairs.push_back(entity);
00313       }
00314     }
00315   }
00316
00318   size_t candidate_count = 1;
00319   std::vector<SFEFaceEntityIdentificationCandidate> results(candidate_count);
00320   int best_candidate_index = -1;
00321   { // STAGE 3: Identification with entities.
00322     // Probe template is matched against every template in the gallery.
00323     // Function returns entity(UUID) which owns the best matching template.
00324
00325     utils::printFormatted("1:N IDENTIFICATION WITH ENTITIES");
00326     SFEError error = sfeFaceEntityIdentify(
00327         &probe_face_template, gallery_face_templates.data(),
00328         enitity_pairs.data(), gallery_face_templates.size(),
00329         identification_threshold, results.data(), &candidate_count, 4);
00330     utils::checkError(error);
00331
00332     if (candidate_count == 0) {
00333       std::cout « "No candidates found. Exiting.." « std::endl;
00334       return 0;
00335     }
00336
00337     std::cout « std::endl;
00338     std::cout « "Found " « results.size() « " candidates " « std::endl;
00339     for (int i = 0; i < results.size(); i++) {
00340       std::cout « "Index: " « i « ", Score: " « results[i].score
00341                 « std::endl;
00342       std::cout « "Entity UUID: ["
00343                 « static_cast<int>(results[i].entity.uuid[0]) « ", "
00344                 « static_cast<int>(results[i].entity.uuid[1]) « "..."
00345                 « static_cast<int>(results[i].entity.uuid[15]) « "]"
00346                 « std::endl;
00347     }
00348   }
00350
00351   utils::printFormatted("FINISHED");
00352 }
```

9.8 D:/glab/da0a89/1/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 (SFEFaceDetect &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

9.8.1 Function Documentation

9.8.1.1 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     SFEError error = sfeFaceDetectInputSize(
00065         image,
00066         SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00067         face_size_min, face_size_max, &recommended_width, &recommended_height);
00068     utils::checkError(error);
00070   };
00071 }
```

9.8.1.2 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]f

[Face Mask Confidence]

[Face Area]

[Face Area]

[Face Head Pose]

[Face Head Pose]

[Face Quality]

[Face Quality]

Definition at line 140 of file example_face_liveness.cpp.

```
00140                                   {
00141
00142   { // STAGE: 0 Parse command line arguments
00143     // Map to store argument values
00144     std::unordered_map<std::string, std::string> args;
00145
00146     // Parse command line arguments
00147     for (int i = 1; i < argc; ++i) {
00148       std::string arg = argv[i];
00149       if (arg[0] == '-') {
00150         if (arg == "-h") {
00151           printHelp();
00152           return 0;
00153         }
00154         // Check if there's a next argument and it isn't another option
00155         if (i + 1 < argc && argv[i + 1][0] != '-') {
00156           args[arg] = argv[++i];
00157         } else {
00158           std::cerr << "Option " << arg << " requires a value." << std::endl;
00159           return 1;
00160         }
00161       } else {
00162         std::cerr << "Unknown option: " << arg << std::endl;
00163         return 1;
00164       }
00165     }
00166
00167     // Assign values to variables based on parsed arguments
00168     if (args.count("-p"))
00169       image_probe = args["-p"];
00170     if (args.count("-d"))
00171       solver_face_detect = args["-d"];
00172     if (args.count("-l"))
00173       solver_face_landmarks = args["-l"];
00174     if (args.count("-i"))
00175       solver_face_liveness = args["-i"];
00176     if (args.count("-t"))
00177       detection_threshold = std::stof(args["-t"]);
00178     if (args.count("-m"))
00179       face_size_min = std::stoi(args["-m"]);
00180     if (args.count("-x"))
00181       face_size_max = std::stoi(args["-x"]);
00182
00183     utils::printFormatted("PARAMETERS");
00184     // Print resource names
00185     std::cout << "Probe image: " << image_probe << std::endl;
00186
00187     // Print solver names
```

```
00188     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00189     std::cout « "Face landmarks solver: " « solver_face_landmarks
00190               « std::endl;
00191     std::cout « "Face liveness solver: " « solver_face_liveness « std::endl;
00192
00193     // Print other options
00194     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00195     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00196     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00197   }
00198
00199   utils::printToolkitInfo();
00200
00201   SFEError error{};
00202
00203   SFESolver detector_solver{};
00204   DEFER(sfeSolverFree(detector_solver));
00205   SFESolver landmarks_solver{};
00206   DEFER(sfeSolverFree(landmarks_solver));
00207   SFESolver liveness_solver{};
00208   DEFER(sfeSolverFree(liveness_solver));
00209   { // STAGE 1: loading solvers
00210     utils::printFormatted("LOADING solvers");
00211     // Initialize face detection solver
00212     error = sfeSolverCreateWithParameters(
00213         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00214         SOLVER_PARAMETERS.size(), &detector_solver);
00215     utils::checkError(error);
00216
00217     // Initialize landmarks detection solver
00218     error = sfeSolverCreateWithParameters(
00219         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00220         SOLVER_PARAMETERS.size(), &landmarks_solver);
00221     utils::checkError(error);
00222
00223     // Initialize face liveness solver
00224     error = sfeSolverCreateWithParameters(
00225         solver_face_liveness.c_str(), SOLVER_PARAMETERS.data(),
00226         SOLVER_PARAMETERS.size(), &liveness_solver);
00227     utils::checkError(error);
00228   }
00229
00230   SFEImage image{};
00231   DEFER(sfeImageFree(image));
00232   SFEImage resized_image{};
00233   DEFER(sfeImageFree(resized_image));
00234   { // STAGE 2: Load image
00235
00236     utils::printFormatted("FACE DETECTION");
00238     // Load image data from file
00239     auto image_data = utils::readFile(image_probe);
00240
00241     // Decode image from data
00242     error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00243     utils::checkError(error);
00245
00246     size_t recommended_width{};
00247     size_t recommended_height{};
00248
00249     // Calculate optimal input image size for face detection. Image will be
00250     // resized to recommended size for optimal performance.
00251     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00252                             face_size_max, recommended_width,
00253                             recommended_height);
00254
00256     // Resize image to recommended size
00257     // NOTE: This function will resize the image without preserving the aspect
00258     // ratio of the image.
00259     error = sfeImageResize(image, recommended_width, recommended_height,
00260                            &resized_image);
00261     utils::checkError(error);
00263   }
00265   SFEFaceDetect detected_face = {};
00266   { // STAGE 3: Detect face in the image
00267     size_t detection_count = 1;
00268     // Detect face in the image
00269     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00270                           &detected_face, &detection_count);
00271     utils::checkError(error);
00272
00273     if (detection_count == 0) {
00274       std::cout « "No face detected in the probe image." « std::endl;
00275       return 0;
00276     } else {
00277       std::cout « "Found " « detection_count
00278                 « " face(s) in the probe image. Using the face with highest "
00279                    "confidence."
```

```
00280                 « std::endl;
00281     }
00282   }
00284
00286   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00287   { // STAGE 4: Get face landmarks
00288
00289     // Use landmarks detection solver to get relevant landmarks for
00290     // the detected face
00291     error = sfeFaceLandmarks(landmarks_solver, image, detected_face.bbox,
00292                              landmarks.data());
00293     utils::checkError(error);
00294   }
00296
00298   float liveness_score = 0.0f;
00299   { // STAGE 5: Liveness check
00300     utils::printFormatted("LIVENESS CHECK");
00301     error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
00302                                    &liveness_score);
00303     utils::checkError(error);
00304   }
00306
00307   { // STAGE 6: Print the liveness score
00308     std::cout « "Liveness score is " « liveness_score;
00309     // Recommended threshold for liveness detection, see EER threshold for distant fast mode in the
    documentation
00310     static const float LIVENESS_THRESHOLD = 0.81f;
00311     if (liveness_score < LIVENESS_THRESHOLD) {
00312       std::cout « " which is below " « LIVENESS_THRESHOLD
00313                 « " threshold. Face is spoof." « std::endl;
00314     } else {
00315       std::cout « " which is above " « LIVENESS_THRESHOLD
00316                 « " threshold. Face is genuine." « std::endl;
00317     }
00318   }
00319
00320   // Optional face attributes to check for further analysis. See the
00321   // documentation for more information.
00322   float face_size = 0;
00323   float mask_confidence;
00324   SFEFaceArea face_area = {};
00325   SFEFaceHeadPose head_pose{};
00326   SFEFaceQualityAttributes quality_attributes{};
00327   { // STAGE 6: (optional) Face attributes
00328     utils::printFormatted("FACE ATTRIBUTES");
00329
00331     error = sfeFaceSize(image, landmarks.data(), &face_size);
00332     utils::checkError(error);
00334
00336     error = sfeFaceMaskConfidence(landmarks.data(), &mask_confidence);
00337     utils::checkError(error);
00339
00341     error = sfeFaceArea(image, landmarks.data(), &face_area);
00342     utils::checkError(error);
00344
00346     error = sfeFaceHeadPose(image, landmarks.data(), &head_pose);
00347     utils::checkError(error);
00349
00351     error =
00352         sfeFaceQualityAttributes(image, landmarks.data(), &quality_attributes);
00353     utils::checkError(error);
00355   }
00356
00357   { // STAGE 8: Print the face attributes
00358     printFaceDetectionInfo(detected_face);
00359     printFaceSize(face_size);
00360     printMaskConfidence(mask_confidence);
00361     prinFaceAreaInfo(face_area);
00362     printFaceHeadPose(head_pose);
00363     printFaceQualityAttributes(quality_attributes);
00364   }
00365
00366   { // STAGE 9: Annotate the image
00367     // Render bounding box
00368     std::stringstream label;
00369     label « "Face" « std::fixed « std::setprecision(2)
00370           « " d:" « detected_face.confidence « " m:" « mask_confidence
00371           « " l:" « liveness_score;
00372
00373     annotate::labelBox(image, label.str(), detected_face.bbox, annotate::WHITE,
00374                        annotate::GREEN);
00375
00376     // Render landmarks
00377     for (auto &landmark : landmarks) {
00378       auto x = landmark.x * image.width;
00379       auto y = landmark.y * image.height;
00380       annotate::circle(image, x, y, 3, annotate::YELLOW);
```

```
00381     }
00382
00383     // Save annotated image
00384     size_t size = image.width * image.height * 3;
00385     auto png_file = std::vector<unsigned char>(size);
00386     error = sfeImageSave(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00387     utils::checkError(error);
00388     png_file.resize(size);
00389     utils::saveFile("face_liveness.png", png_file);
00390
00391     std::cout << std::endl;
00392     std::cout << "Annotated image saved to face_liveness.png" << std::endl;
00393   }
00394
00395   utils::printFormatted("FINISHED");
00396 }
```

9.8.1.3 prinFaceAreaInfo()

```
void prinFaceAreaInfo (
            SFEFaceArea & face_area )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 86 of file example_face_liveness.cpp.

```
00086                                                    {
00087   std::cout << "SFEFaceArea{ " << std::endl;
00088   std::cout << "  size: " << face_area.size << " [px]" << std::endl;
00089   std::cout << "  area: " << face_area.area << std::endl;
00090   std::cout << "  area_in_image: " << face_area.area_in_image << std::endl;
00091   std::cout << "}" << std::endl;
00092   std::cout << std::endl;
00093 }
```

9.8.1.4 printFaceDetectionInfo()

```
void printFaceDetectionInfo (
            SFEFaceDetect & detected_face )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 73 of file example_face_liveness.cpp.

```
00073                                                          {
00074   std::cout << "SFEFaceDetect{ " << std::endl;
00075   std::cout << "  confidence: " << detected_face.confidence << std::endl;
00076   std::cout << "  SFEBbox{ " << std::endl;
00077   std::cout << "    x: " << detected_face.bbox.x << std::endl;
00078   std::cout << "    y: " << detected_face.bbox.y << std::endl;
00079   std::cout << "    width: " << detected_face.bbox.width << std::endl;
00080   std::cout << "    height: " << detected_face.bbox.height << std::endl;
00081   std::cout << "    }" << std::endl;
00082   std::cout << "  } " << std::endl;
00083   std::cout << std::endl;
00084 }
```

9.8.1.5 printFaceHeadPose()

```
void printFaceHeadPose (
            SFEFaceHeadPose & head_pose )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 95 of file example_face_liveness.cpp.

```
00095                                                          {
00096   std::cout « "SFEFaceHeadPose{" « std::endl;
00097   std::cout « "  pitch: " « head_pose.pitch « std::endl;
00098   std::cout « "  yaw: " « head_pose.yaw « std::endl;
00099   std::cout « "  roll: " « head_pose.roll « std::endl;
00100   std::cout « "} " « std::endl;
00101   std::cout « std::endl;
00102 }
```

9.8.1.6 printFaceQualityAttributes()

```
void printFaceQualityAttributes (
            SFEFaceQualityAttributes & quality_attributes )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 115 of file example_face_liveness.cpp.

```
00115                                                                  {
00116   std::cout « "SFEFaceQualityAttributes{" « std::endl;
00117   std::cout « "  sharpness: " « quality_attributes.sharpness « std::endl;
00118   std::cout « "  brightness: " « quality_attributes.brightness « std::endl;
00119   std::cout « "  contrast: " « quality_attributes.contrast « std::endl;
00120   std::cout « "  unique_intensity_levels: "
00121             « quality_attributes.unique_intensity_levels « std::endl;
00122   std::cout « "} " « std::endl;
00123   std::cout « std::endl;
00124 }
```

9.8.1.7 printFaceSize()

```
void printFaceSize (
            size_t face_size )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 104 of file example_face_liveness.cpp.

```
00104                                          {
00105   std::cout « "Face size: " « face_size « " [px]" « std::endl;
00106   std::cout « std::endl;
00107 }
```

9.8.1.8 printHelp()

```
void printHelp ( )
```

Definition at line 126 of file example_face_liveness.cpp.

```
00126                  {
00127   std::cout « "Help: Usage of the program." « std::endl;
00128   std::cout « "Options:" « std::endl;
00129   std::cout « "-h: Display help." « std::endl;
00130   std::cout « "-p: Probe image file." « std::endl;
00131   std::cout « "-d: Path to detector solver." « std::endl;
00132   std::cout « "-l: Path to landmarks solver." « std::endl;
00133   std::cout « "-i: Path to liveness solver." « std::endl;
00134   std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
00135   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00136   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00137 }
```

9.8.1.9 printMaskConfidence()

```
void printMaskConfidence (
            float mask_confidence )  [inline]
```

Examples

example_face_liveness.cpp.

Definition at line 109 of file example_face_liveness.cpp.

```
00109                                                    {
00110   std::cout « "Mask confidence: " « mask_confidence « std::endl;
00111   std::cout « std::endl;
00112 }
```

9.8.2 Variable Documentation

9.8.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 30 of file example_face_liveness.cpp.

9.8.2.2 face_size_max

```
size_t face_size_max = 170
```

Definition at line 28 of file example_face_liveness.cpp.

9.8.2.3 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.

9.8.2.4 image_probe

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

Definition at line 17 of file example_face_liveness.cpp.

9.8.2.5 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.

9.8.2.6 solver_face_landmarks

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

Definition at line 21 of file example_face_liveness.cpp.

9.8.2.7 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.

9.8.2.8 SOLVER_PARAMETERS

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

Definition at line 24 of file example_face_liveness.cpp.

9.9 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     SFEError error = sfeFaceDetectInputSize(
00065         image,
00066         SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00067         face_size_min, face_size_max, &recommended_width, &recommended_height);
00068     utils::checkError(error);
00070   };
00071 }
00072
00073 inline void printFaceDetectionInfo(SFEFaceDetect &detected_face) {
00074   std::cout « "SFEFaceDetect{ " « std::endl;
00075   std::cout « "  confidence: " « detected_face.confidence « std::endl;
00076   std::cout « "  SFEBbox{ " « std::endl;
00077   std::cout « "    x: " « detected_face.bbox.x « std::endl;
00078   std::cout « "    y: " « detected_face.bbox.y « std::endl;
00079   std::cout « "    width: " « detected_face.bbox.width « std::endl;
00080   std::cout « "    height: " « detected_face.bbox.height « std::endl;
00081   std::cout « "    }" « std::endl;
00082   std::cout « "  } " « std::endl;
00083   std::cout « std::endl;
00084 }
00085
00086 inline void prinFaceAreaInfo(SFEFaceArea &face_area) {
00087   std::cout « "SFEFaceArea{ " « std::endl;
00088   std::cout « "  size: " « face_area.size « " [px]" « std::endl;
00089   std::cout « "  area: " « face_area.area « std::endl;
00090   std::cout « "  area_in_image: " « face_area.area_in_image « std::endl;
00091   std::cout « "}" « std::endl;
00092   std::cout « std::endl;
00093 }
00094
00095 inline void printFaceHeadPose(SFEFaceHeadPose &head_pose) {
```

```
00096   std::cout « "SFEFaceHeadPose{" « std::endl;
00097   std::cout « "  pitch: " « head_pose.pitch « std::endl;
00098   std::cout « "  yaw: " « head_pose.yaw « std::endl;
00099   std::cout « "  roll: " « head_pose.roll « std::endl;
00100   std::cout « "} " « std::endl;
00101   std::cout « std::endl;
00102 }
00103
00104 inline void printFaceSize(size_t face_size) {
00105   std::cout « "Face size: " « face_size « " [px]" « std::endl;
00106   std::cout « std::endl;
00107 }
00108
00109 inline void printMaskConfidence(float mask_confidence) {
00110   std::cout « "Mask confidence: " « mask_confidence « std::endl;
00111   std::cout « std::endl;
00112 }
00113
00114 inline void
00115 printFaceQualityAttributes(SFEFaceQualityAttributes &quality_attributes) {
00116   std::cout « "SFEFaceQualityAttributes{" « std::endl;
00117   std::cout « "  sharpness: " « quality_attributes.sharpness « std::endl;
00118   std::cout « "  brightness: " « quality_attributes.brightness « std::endl;
00119   std::cout « "  contrast: " « quality_attributes.contrast « std::endl;
00120   std::cout « "  unique_intensity_levels: "
00121             « quality_attributes.unique_intensity_levels « std::endl;
00122   std::cout « "} " « std::endl;
00123   std::cout « std::endl;
00124 }
00125
00126 void printHelp() {
00127   std::cout « "Help: Usage of the program." « std::endl;
00128   std::cout « "Options:" « std::endl;
00129   std::cout « "-h: Display help." « std::endl;
00130   std::cout « "-p: Probe image file." « std::endl;
00131   std::cout « "-d: Path to detector solver." « std::endl;
00132   std::cout « "-l: Path to landmarks solver." « std::endl;
00133   std::cout « "-i: Path to liveness solver." « std::endl;
00134   std::cout « "-t: Face detection threshold. <0,1>" « std::endl;
00135   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00136   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00137 }
00138
00140 int main(int argc, char *argv[]) {
00141
00142   { // STAGE: 0 Parse command line arguments
00143     // Map to store argument values
00144     std::unordered_map<std::string, std::string> args;
00145
00146     // Parse command line arguments
00147     for (int i = 1; i < argc; ++i) {
00148       std::string arg = argv[i];
00149       if (arg[0] == '-') {
00150         if (arg == "-h") {
00151           printHelp();
00152           return 0;
00153         }
00154         // Check if there's a next argument and it isn't another option
00155         if (i + 1 < argc && argv[i + 1][0] != '-') {
00156           args[arg] = argv[++i];
00157         } else {
00158           std::cerr « "Option " « arg « " requires a value." « std::endl;
00159           return 1;
00160         }
00161       } else {
00162         std::cerr « "Unknown option: " « arg « std::endl;
00163         return 1;
00164       }
00165     }
00166
00167     // Assign values to variables based on parsed arguments
00168     if (args.count("-p"))
00169       image_probe = args["-p"];
00170     if (args.count("-d"))
00171       solver_face_detect = args["-d"];
00172     if (args.count("-l"))
00173       solver_face_landmarks = args["-l"];
00174     if (args.count("-i"))
00175       solver_face_liveness = args["-i"];
00176     if (args.count("-t"))
00177       detection_threshold = std::stof(args["-t"]);
00178     if (args.count("-m"))
00179       face_size_min = std::stoi(args["-m"]);
00180     if (args.count("-x"))
00181       face_size_max = std::stoi(args["-x"]);
00182
00183     utils::printFormatted("PARAMETERS");
```

```
00184     // Print resource names
00185     std::cout « "Probe image: " « image_probe « std::endl;
00186
00187     // Print solver names
00188     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00189     std::cout « "Face landmarks solver: " « solver_face_landmarks
00190               « std::endl;
00191     std::cout « "Face liveness solver: " « solver_face_liveness « std::endl;
00192
00193     // Print other options
00194     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00195     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00196     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00197   }
00198
00199   utils::printToolkitInfo();
00200
00201   SFEError error{};
00202
00203   SFESolver detector_solver{};
00204   DEFER(sfeSolverFree(detector_solver));
00205   SFESolver landmarks_solver{};
00206   DEFER(sfeSolverFree(landmarks_solver));
00207   SFESolver liveness_solver{};
00208   DEFER(sfeSolverFree(liveness_solver));
00209   { // STAGE 1: loading solvers
00210     utils::printFormatted("LOADING solvers");
00211     // Initialize face detection solver
00212     error = sfeSolverCreateWithParameters(
00213         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00214         SOLVER_PARAMETERS.size(), &detector_solver);
00215     utils::checkError(error);
00216
00217     // Initialize landmarks detection solver
00218     error = sfeSolverCreateWithParameters(
00219         solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
00220         SOLVER_PARAMETERS.size(), &landmarks_solver);
00221     utils::checkError(error);
00222
00223     // Initialize face liveness solver
00224     error = sfeSolverCreateWithParameters(
00225         solver_face_liveness.c_str(), SOLVER_PARAMETERS.data(),
00226         SOLVER_PARAMETERS.size(), &liveness_solver);
00227     utils::checkError(error);
00228   }
00229
00230   SFEImage image{};
00231   DEFER(sfeImageFree(image));
00232   SFEImage resized_image{};
00233   DEFER(sfeImageFree(resized_image));
00234   { // STAGE 2: Load image
00235
00236     utils::printFormatted("FACE DETECTION");
00238     // Load image data from file
00239     auto image_data = utils::readFile(image_probe);
00240
00241     // Decode image from data
00242     error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00243     utils::checkError(error);
00245
00246     size_t recommended_width{};
00247     size_t recommended_height{};
00248
00249     // Calculate optimal input image size for face detection. Image will be
00250     // resized to recommended size for optimal performance.
00251     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00252                             face_size_max, recommended_width,
00253                             recommended_height);
00254
00256     // Resize image to recommended size
00257     // NOTE: This function will resize the image without preserving the aspect
00258     // ratio of the image.
00259     error = sfeImageResize(image, recommended_width, recommended_height,
00260                            &resized_image);
00261     utils::checkError(error);
00263   }
00265   SFEFaceDetect detected_face = {};
00266   { // STAGE 3: Detect face in the image
00267     size_t detection_count = 1;
00268     // Detect face in the image
00269     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00270                           &detected_face, &detection_count);
00271     utils::checkError(error);
00272
00273     if (detection_count == 0) {
00274       std::cout « "No face detected in the probe image." « std::endl;
00275       return 0;
```

```
00276     } else {
00277       std::cout « "Found " « detection_count
00278                « " face(s) in the probe image. Using the face with highest "
00279                   "confidence."
00280                « std::endl;
00281     }
00282   }
00284
00286   std::vector<SFEFaceLandmarks> landmarks(SFE_FACE_LANDMARK_COUNT);
00287   { // STAGE 4: Get face landmarks
00288
00289     // Use landmarks detection solver to get relevant landmarks for
00290     // the detected face
00291     error = sfeFaceLandmarks(landmarks_solver, image, detected_face.bbox,
00292                              landmarks.data());
00293     utils::checkError(error);
00294   }
00296
00298   float liveness_score = 0.0f;
00299   { // STAGE 5: Liveness check
00300     utils::printFormatted("LIVENESS CHECK");
00301     error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
00302                                    &liveness_score);
00303     utils::checkError(error);
00304   }
00306
00307   { // STAGE 6: Print the liveness score
00308     std::cout « "Liveness score is " « liveness_score;
00309     // Recommended threshold for liveness detection, see EER threshold for distant fast mode in the
      documentation
00310     static const float LIVENESS_THRESHOLD = 0.81f;
00311     if (liveness_score < LIVENESS_THRESHOLD) {
00312       std::cout « " which is below " « LIVENESS_THRESHOLD
00313                « " threshold. Face is spoof." « std::endl;
00314     } else {
00315       std::cout « " which is above " « LIVENESS_THRESHOLD
00316                « " threshold. Face is genuine." « std::endl;
00317     }
00318   }
00319
00320   // Optional face attributes to check for further analysis. See the
00321   // documentation for more information.
00322   float face_size = 0;
00323   float mask_confidence;
00324   SFEFaceArea face_area = {};
00325   SFEFaceHeadPose head_pose{};
00326   SFEFaceQualityAttributes quality_attributes{};
00327   { // STAGE 6: (optional) Face attributes
00328     utils::printFormatted("FACE ATTRIBUTES");
00329
00331     error = sfeFaceSize(image, landmarks.data(), &face_size);
00332     utils::checkError(error);
00334
00336     error = sfeFaceMaskConfidence(landmarks.data(), &mask_confidence);
00337     utils::checkError(error);
00339
00341     error = sfeFaceArea(image, landmarks.data(), &face_area);
00342     utils::checkError(error);
00344
00346     error = sfeFaceHeadPose(image, landmarks.data(), &head_pose);
00347     utils::checkError(error);
00349
00351     error =
00352         sfeFaceQualityAttributes(image, landmarks.data(), &quality_attributes);
00353     utils::checkError(error);
00355   }
00356
00357   { // STAGE 8: Print the face attributes
00358     printFaceDetectionInfo(detected_face);
00359     printFaceSize(face_size);
00360     printMaskConfidence(mask_confidence);
00361     prinFaceAreaInfo(face_area);
00362     printFaceHeadPose(head_pose);
00363     printFaceQualityAttributes(quality_attributes);
00364   }
00365
00366   { // STAGE 9: Annotate the image
00367     // Render bounding box
00368     std::stringstream label;
00369     label « "Face" « std::fixed « std::setprecision(2)
00370           « " d:" « detected_face.confidence « " m:" « mask_confidence
00371           « " l:" « liveness_score;
00372
00373     annotate::labelBox(image, label.str(), detected_face.bbox, annotate::WHITE,
00374                        annotate::GREEN);
00375
00376     // Render landmarks
```

```
00377     for (auto &landmark : landmarks) {
00378       auto x = landmark.x * image.width;
00379       auto y = landmark.y * image.height;
00380       annotate::circle(image, x, y, 3, annotate::YELLOW);
00381     }
00382
00383     // Save annotated image
00384     size_t size = image.width * image.height * 3;
00385     auto png_file = std::vector<unsigned char>(size);
00386     error = sfeImageSave(image, SFE_IMAGE_FORMAT_PNG, png_file.data(), &size);
00387     utils::checkError(error);
00388     png_file.resize(size);
00389     utils::saveFile("face_liveness.png", png_file);
00390
00391     std::cout « std::endl;
00392     std::cout « "Annotated image saved to face_liveness.png" « std::endl;
00393   }
00394
00395   utils::printFormatted("FINISHED");
00396 }
```

9.10 D:/glab/da0a89/1/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< SFEFaceDetect > 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 SFEFaceTrackedState &state)

  Print tracked face state.
- std::ostream & operator<< (std::ostream &os, const SFEFaceTracked &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< SFEFaceDetect > &detections, SFEFaceTracker 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

9.10.1 Function Documentation

9.10.1.1 detectFaces()

```
std::vector< SFEFaceDetect > detectFaces (
            SFEImage & image,
            SFESolver & detector_solver )
```

Detect faces in the image.

Examples

example_face_track.cpp.

Definition at line 83 of file example_face_track.cpp.

```
00084                                                                    {
00085
00086   SFEImage resized_image{};
00087   DEFER(sfeImageFree(resized_image));
00088   SFEError error{};
00089   { // STAGE 1 Load image
00090     size_t recommended_width{};
00091     size_t recommended_height{};
00092
00093     // Calculate optimal input image size for face detection. Image will be
00094     // resized to recommended size for optimal performance.
00095     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00096                             face_size_max, recommended_width,
00097                             recommended_height);
00098
00099     // Resize image to recommended size
00100     // NOTE: This function will resize the image without preserving the aspect
00101     // ratio of the image.
00102
00103     error = sfeImageResize(image, recommended_width, recommended_height,
00104                            &resized_image);
00105     utils::checkError(error);
00106   }
00107
00108   size_t detection_count = 10;
00109   std::vector<SFEFaceDetect> detected_faces(detection_count);
00110   { // STAGE 2: Detect face in the image
00111
00112     // Detect faces in the image
00113     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00114                           detected_faces.data(), &detection_count);
00115     utils::checkError(error);
00116
00117     if (detection_count == 0) {
00118       std::ostringstream oss;
00119       oss « "Error: No face detected in the image ";
00120       throw std::runtime_error(oss.str());
00121     }
00122     detected_faces.resize(detection_count);
00123
00124     std::cout « "Detected " « detection_count « " faces in the image."
00125               « std::endl;
00126   }
00127   return detected_faces;
00128 }
```

9.10.1.2 frame()

```
void frame (
            std::vector< SFEFaceDetect > & detections,
            SFEFaceTracker tracker )
```

Print out the results of the face tracking.

Examples

example_face_track.cpp.

Definition at line 182 of file example_face_track.cpp.

```
00182                                                                    {
00183   size_t reserved_size = 10;
00184   size_t tracked_faces_count = reserved_size;
00185   size_t lost_faces_count = reserved_size;
00186   size_t removed_faces_count = reserved_size;
00187
00188   std::vector<SFEFaceTracked> tracked_faces(tracked_faces_count);
00189   std::vector<SFEEntity> lost_faces(lost_faces_count);
00190   std::vector<SFEEntity> removed_faces(removed_faces_count);
00191
00192   // Update tracker with detected faces
00193   SFEError error =
00194       sfeFaceTrackerUpdate(tracker, detections.data(), detections.size(),
00195                            tracked_faces.data(), &tracked_faces_count);
00196   utils::checkError(error);
00197   tracked_faces.resize(tracked_faces_count);
00198
00199   // Get lost faces
00200   error = sfeFaceTrackerLost(tracker, lost_faces.data(), &lost_faces_count);
00201   utils::checkError(error);
00202   lost_faces.resize(lost_faces_count);
00203
00204   // Get removed faces
00205   error = sfeFaceTrackerRemoved(tracker, removed_faces.data(),
00206                                 &removed_faces_count);
00207   utils::checkError(error);
00208   removed_faces.resize(removed_faces_count);
00209
00210   // Print out frame results
00211   std::cout « "Tracked" « std::endl;
00212   std::cout « tracked_faces;
00213   std::cout « "Lost" « std::endl;
00214   std::cout « lost_faces;
00215   std::cout « "Removed" « std::endl;
00216   std::cout « removed_faces;
00217 }
```

9.10.1.3 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
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     SFEError error = sfeFaceDetectInputSize(
00064         image,
00065         SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00066         face_size_min, face_size_max, &recommended_width, &recommended_height);
00067     utils::checkError(error);
00068   };
00069 }
```

9.10.1.4 main()

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

Main function is used to parse command line arguments.

Definition at line 220 of file example_face_track.cpp.

```
00220                                    {
00221
00222   { // STAGE 0: Parse command line arguments
00223     // Map to store argument values
00224     std::unordered_map<std::string, std::string> args;
00225
00226     // Parse command line arguments
00227     for (int i = 1; i < argc; ++i) {
00228       std::string arg = argv[i];
00229       if (arg[0] == '-') {
00230         if (arg == "-h") {
00231           printHelp();
00232           return 0;
00233         }
00234         // Check if there's a next argument and it isn't another option
00235         if (i + 1 < argc && argv[i + 1][0] != '-') {
00236           args[arg] = argv[++i];
00237         } else {
00238           std::cerr « "Option " « arg « " requires a value." « std::endl;
00239           return 1;
00240         }
00241       } else {
00242         std::cerr « "Unknown option: " « arg « std::endl;
00243         return 1;
00244       }
00245     }
00246
00247     // Assign values to variables based on parsed arguments
00248     if (args.count("-p"))
00249       image_probe = args["-p"];
00250     if (args.count("-d"))
00251       solver_face_detect = args["-d"];
00252     if (args.count("-t"))
00253       detection_threshold = std::stof(args["-t"]);
00254     if (args.count("-m"))
00255       face_size_min = std::stoi(args["-m"]);
00256     if (args.count("-x"))
00257       face_size_max = std::stoi(args["-x"]);
00258
00259     utils::printFormatted("EXAMPLE PARAMETERS");
00260     // Print resource names
```

```
00261     std::cout « "Probe image: " « image_probe « std::endl;
00262
00263     // Print solver names
00264     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00265
00266     // Print other options
00267     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00268     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00269     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00270   }
00271
00272   utils::printToolkitInfo();
00273
00274   SFEError error{};
00275
00276   SFESolver detector_solver{};
00277   DEFER(sfeSolverFree(detector_solver));
00278   { // STAGE 1: loading solvers
00279     utils::printFormatted("LOADING SOLVERS");
00280     // Initialize face detection solver
00281     error = sfeSolverCreateWithParameters(
00282         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00283         SOLVER_PARAMETERS.size(), &detector_solver);
00284     utils::checkError(error);
00285   }
00286
00287   std::vector<SFEFaceDetect> detected_faces;
00288   { // STAGE 2: Detect faces
00289     utils::printFormatted("DETECT FACES");
00290
00291     SFEImage image{};
00292     DEFER(sfeImageFree(image));
00293     { // STAGE 2.1: Load image
00294       // Load image data from file
00295       auto image_data = utils::readFile(image_probe);
00296
00297       // Decode image from data
00298       error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00299       utils::checkError(error);
00300     }
00301
00302     SFEFaceDetect detected_face = {};
00303     { // STAGE 2.2: Detect face in the image
00304       // Detect a face in the image
00305       detected_faces = detectFaces(image, detector_solver);
00306     }
00307   }
00308
00309   SFEFaceTracker tracker{};
00310   DEFER(sfeFaceTrackerFree(tracker));
00311   { // STAGE 3: Track faces
00312     utils::printFormatted("TRACK FACES");
00313     error = sfeFaceTrackerCreate(0.1f, 0.3f, 0.1f, 1, &tracker);
00314     utils::checkError(error);
00315
00316     // We will be using the current detected faces and simulate detection across
00317     // 4 frames In a real application, you would call detectFaces() for each
00318     // frame and pass the detected faces to the tracker
00319
00320     // First frame with detected faces, all should track as NEW
00321     utils::printFormatted("FRAME 1");
00322     frame(detected_faces, tracker);
00323
00324     // Second frame with the same UUIDs, all should track as TRACKED
00325     utils::printFormatted("FRAME 2");
00326     frame(detected_faces, tracker);
00327
00328     // Simulate a lost detection
00329     detected_faces.clear();
00330
00331     // We should see the lost UUIDs
00332     utils::printFormatted("FRAME 3");
00333     frame(detected_faces, tracker);
00334
00335     // We should see the removed UUIDs
00336     utils::printFormatted("FRAME 4");
00337     frame(detected_faces, tracker);
00338   }
00339   utils::printFormatted("FINISHED");
00340 }
```

9.10.1.5 operator<<() [1/4]

```
std::ostream & operator<< (
```

```
            std::ostream & os,
            const SFEEntity & entity )
```

Print UUID.

Examples

example_face_track.cpp.

Definition at line 131 of file example_face_track.cpp.

```
00131                                                                    {
00132   for (size_t i = 0; i < 16; ++i) {
00133     os « std::hex « std::setw(2) « std::setfill('0')
00134        « static_cast<int>(entity.uuid[i]);
00135     if (i == 3 || i == 5 || i == 7 || i == 9) {
00136       os « '-';
00137     }
00138   }
00139   os « std::dec;
00140   return os;
00141 }
```

9.10.1.6 operator<<() [2/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const SFEFaceTracked & tracked )
```

Print tracked face.

Definition at line 166 of file example_face_track.cpp.

```
00166                                                                        {
00167   os « "Face ID: " « tracked.id « " UUID: " « tracked.uuid
00168      « " State: " « tracked.state;
00169   return os;
00170 }
```

9.10.1.7 operator<<() [3/4]

```
std::ostream & operator<< (
            std::ostream & os,
            const SFEFaceTrackedState & state )
```

Print tracked face state.

Definition at line 144 of file example_face_track.cpp.

```
00144                                                                      {
00145   switch (state) {
00146   case SFE_FACE_TRACKED_STATE_NEW:
00147     os « "NEW";
00148     break;
00149   case SFE_FACE_TRACKED_STATE_TRACKED:
00150     os « "TRACKED";
00151     break;
00152   case SFE_FACE_TRACKED_STATE_LOST:
00153     os « "LOST";
00154     break;
00155   case SFE_FACE_TRACKED_STATE_REMOVED:
00156     os « "REMOVED";
00157     break;
00158   default:
00159     os « "UNKNOWN";
00160     break;
00161   }
00162   return os;
00163 }
```

9.10.1.8 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 174 of file example_face_track.cpp.

```
00174                                                                    {
00175   for (auto &item : vec) {
00176     os « item « std::endl;
00177   }
00178   return os;
00179 }
```

9.10.1.9 printHelp()

```
void printHelp ( )
```

Definition at line 71 of file example_face_track.cpp.

```
00071                 {
00072   std::cout « "Help: Usage of the program." « std::endl;
00073   std::cout « "Options:" « std::endl;
00074   std::cout « "-h: Display help." « std::endl;
00075   std::cout « "-p: Probe image file." « std::endl;
00076   std::cout « "-d: Path to detector solver." « std::endl;
00077   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00078   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00079   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00080 }
```

9.10.2 Variable Documentation

9.10.2.1 detection_threshold

```
float detection_threshold = 0.1f
```

Definition at line 30 of file example_face_track.cpp.

9.10.2.2 face_size_max

```
size_t face_size_max = 170
```

Definition at line 28 of file example_face_track.cpp.

9.10.2.3 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.

9.10.2.4 image_probe

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

Definition at line 19 of file example_face_track.cpp.

9.10.2.5 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.

9.10.2.6 SOLVER_PARAMETERS

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

Definition at line 24 of file example_face_track.cpp.

9.11 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     SFEError error = sfeFaceDetectInputSize(
00064         image,
00065         SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
00066         face_size_min, face_size_max, &recommended_width, &recommended_height);
00067     utils::checkError(error);
00068   };
00069 }
00070
00071 void printHelp() {
00072   std::cout « "Help: Usage of the program." « std::endl;
00073   std::cout « "Options:" « std::endl;
00074   std::cout « "-h: Display help." « std::endl;
00075   std::cout « "-p: Probe image file." « std::endl;
00076   std::cout « "-d: Path to detector solver." « std::endl;
00077   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00078   std::cout « "-m: Minimal face size in pixels to detect." « std::endl;
00079   std::cout « "-x: Max face size in pixels to detect." « std::endl;
00080 }
00081
00083 std::vector<SFEFaceDetect> detectFaces(SFEImage &image,
00084                                        SFESolver &detector_solver) {
00085
00086   SFEImage resized_image{};
00087   DEFER(sfeImageFree(resized_image));
00088   SFEError error{};
00089   { // STAGE 1 Load image
00090     size_t recommended_width{};
00091     size_t recommended_height{};
00092
00093     // Calculate optimal input image size for face detection. Image will be
00094     // resized to recommended size for optimal performance.
00095     getRecommendedImageSize(solver_face_detect, image, face_size_min,
00096                             face_size_max, recommended_width,
00097                             recommended_height);
00098
00099     // Resize image to recommended size
00100     // NOTE: This function will resize the image without preserving the aspect
00101     // ratio of the image.
00102
00103     error = sfeImageResize(image, recommended_width, recommended_height,
00104                            &resized_image);
00105     utils::checkError(error);
00106   }
00107
00108   size_t detection_count = 10;
00109   std::vector<SFEFaceDetect> detected_faces(detection_count);
00110   { // STAGE 2: Detect face in the image
00111
00112     // Detect faces in the image
00113     error = sfeFaceDetect(detector_solver, resized_image, detection_threshold,
00114                           detected_faces.data(), &detection_count);
00115     utils::checkError(error);
00116
00117     if (detection_count == 0) {
00118       std::ostringstream oss;
00119       oss « "Error: No face detected in the image ";
00120       throw std::runtime_error(oss.str());
00121     }
00122     detected_faces.resize(detection_count);
00123
00124     std::cout « "Detected " « detection_count « " faces in the image."
00125               « std::endl;
00126   }
00127   return detected_faces;
00128 }
00129
00131 std::ostream &operator«(std::ostream &os, const SFEEntity &entity) {
00132   for (size_t i = 0; i < 16; ++i) {
00133     os « std::hex « std::setw(2) « std::setfill('0')
00134        « static_cast<int>(entity.uuid[i]);
00135     if (i == 3 || i == 5 || i == 7 || i == 9) {
00136       os « '-';
00137     }
00138   }
00139   os « std::dec;
00140   return os;
00141 }
00142
00144 std::ostream &operator«(std::ostream &os, const SFEFaceTrackedState &state) {
```

```
00145   switch (state) {
00146   case SFE_FACE_TRACKED_STATE_NEW:
00147     os « "NEW";
00148     break;
00149   case SFE_FACE_TRACKED_STATE_TRACKED:
00150     os « "TRACKED";
00151     break;
00152   case SFE_FACE_TRACKED_STATE_LOST:
00153     os « "LOST";
00154     break;
00155   case SFE_FACE_TRACKED_STATE_REMOVED:
00156     os « "REMOVED";
00157     break;
00158   default:
00159     os « "UNKNOWN";
00160     break;
00161   }
00162   return os;
00163 }
00164
00166 std::ostream &operator«(std::ostream &os, const SFEFaceTracked &tracked) {
00167   os « "Face ID: " « tracked.id « " UUID: " « tracked.uuid
00168      « " State: " « tracked.state;
00169   return os;
00170 }
00171
00173 template <typename T>
00174 std::ostream &operator«(std::ostream &os, const std::vector<T> &vec) {
00175   for (auto &item : vec) {
00176     os « item « std::endl;
00177   }
00178   return os;
00179 }
00180
00182 void frame(std::vector<SFEFaceDetect> &detections, SFEFaceTracker tracker) {
00183   size_t reserved_size = 10;
00184   size_t tracked_faces_count = reserved_size;
00185   size_t lost_faces_count = reserved_size;
00186   size_t removed_faces_count = reserved_size;
00187
00188   std::vector<SFEFaceTracked> tracked_faces(tracked_faces_count);
00189   std::vector<SFEEntity> lost_faces(lost_faces_count);
00190   std::vector<SFEEntity> removed_faces(removed_faces_count);
00191
00192   // Update tracker with detected faces
00193   SFEError error =
00194       sfeFaceTrackerUpdate(tracker, detections.data(), detections.size(),
00195                            tracked_faces.data(), &tracked_faces_count);
00196   utils::checkError(error);
00197   tracked_faces.resize(tracked_faces_count);
00198
00199   // Get lost faces
00200   error = sfeFaceTrackerLost(tracker, lost_faces.data(), &lost_faces_count);
00201   utils::checkError(error);
00202   lost_faces.resize(lost_faces_count);
00203
00204   // Get removed faces
00205   error = sfeFaceTrackerRemoved(tracker, removed_faces.data(),
00206                                 &removed_faces_count);
00207   utils::checkError(error);
00208   removed_faces.resize(removed_faces_count);
00209
00210   // Print out frame results
00211   std::cout « "Tracked" « std::endl;
00212   std::cout « tracked_faces;
00213   std::cout « "Lost" « std::endl;
00214   std::cout « lost_faces;
00215   std::cout « "Removed" « std::endl;
00216   std::cout « removed_faces;
00217 }
00218
00220 int main(int argc, char *argv[]) {
00221
00222   { // STAGE 0: Parse command line arguments
00223     // Map to store argument values
00224     std::unordered_map<std::string, std::string> args;
00225
00226     // Parse command line arguments
00227     for (int i = 1; i < argc; ++i) {
00228       std::string arg = argv[i];
00229       if (arg[0] == '-') {
00230         if (arg == "-h") {
00231           printHelp();
00232           return 0;
00233         }
00234         // Check if there's a next argument and it isn't another option
00235         if (i + 1 < argc && argv[i + 1][0] != '-') {
```

```
00236           args[arg] = argv[++i];
00237         } else {
00238           std::cerr « "Option " « arg « " requires a value." « std::endl;
00239           return 1;
00240         }
00241       } else {
00242         std::cerr « "Unknown option: " « arg « std::endl;
00243         return 1;
00244       }
00245     }
00246
00247     // Assign values to variables based on parsed arguments
00248     if (args.count("-p"))
00249       image_probe = args["-p"];
00250     if (args.count("-d"))
00251       solver_face_detect = args["-d"];
00252     if (args.count("-t"))
00253       detection_threshold = std::stof(args["-t"]);
00254     if (args.count("-m"))
00255       face_size_min = std::stoi(args["-m"]);
00256     if (args.count("-x"))
00257       face_size_max = std::stoi(args["-x"]);
00258
00259     utils::printFormatted("EXAMPLE PARAMETERS");
00260     // Print resource names
00261     std::cout « "Probe image: " « image_probe « std::endl;
00262
00263     // Print solver names
00264     std::cout « "Face detection solver: " « solver_face_detect « std::endl;
00265
00266     // Print other options
00267     std::cout « "Min face size [px]: " « face_size_min « std::endl;
00268     std::cout « "Max face size [px]: " « face_size_max « std::endl;
00269     std::cout « "Detection threshold: " « detection_threshold « std::endl;
00270   }
00271
00272   utils::printToolkitInfo();
00273
00274   SFEError error{};
00275
00276   SFESolver detector_solver{};
00277   DEFER(sfeSolverFree(detector_solver));
00278   { // STAGE 1: loading solvers
00279     utils::printFormatted("LOADING SOLVERS");
00280     // Initialize face detection solver
00281     error = sfeSolverCreateWithParameters(
00282         solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
00283         SOLVER_PARAMETERS.size(), &detector_solver);
00284     utils::checkError(error);
00285   }
00286
00287   std::vector<SFEFaceDetect> detected_faces;
00288   { // STAGE 2: Detect faces
00289     utils::printFormatted("DETECT FACES");
00290
00291     SFEImage image{};
00292     DEFER(sfeImageFree(image));
00293     { // STAGE 2.1: Load image
00294       // Load image data from file
00295       auto image_data = utils::readFile(image_probe);
00296
00297       // Decode image from data
00298       error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00299       utils::checkError(error);
00300     }
00301
00302     SFEFaceDetect detected_face = {};
00303     { // STAGE 2.2: Detect face in the image
00304       // Detect a face in the image
00305       detected_faces = detectFaces(image, detector_solver);
00306     }
00307   }
00308
00309   SFEFaceTracker tracker{};
00310   DEFER(sfeFaceTrackerFree(tracker));
00311   { // STAGE 3: Track faces
00312     utils::printFormatted("TRACK FACES");
00313     error = sfeFaceTrackerCreate(0.1f, 0.3f, 0.1f, 1, &tracker);
00314     utils::checkError(error);
00315
00316     // We will be using the current detected faces and simulate detection across
00317     // 4 frames In a real application, you would call detectFaces() for each
00318     // frame and pass the detected faces to the tracker
00319
00320     // First frame with detected faces, all should track as NEW
00321     utils::printFormatted("FRAME 1");
00322     frame(detected_faces, tracker);
```

```
00323
00324     // Second frame with the same UUIDs, all should track as TRACKED
00325     utils::printFormatted("FRAME 2");
00326     frame(detected_faces, tracker);
00327
00328     // Simulate a lost detection
00329     detected_faces.clear();
00330
00331     // We should see the lost UUIDs
00332     utils::printFormatted("FRAME 3");
00333     frame(detected_faces, tracker);
00334
00335     // We should see the removed UUIDs
00336     utils::printFormatted("FRAME 4");
00337     frame(detected_faces, tracker);
00338   }
00339   utils::printFormatted("FINISHED");
00340 }
```

9.12 D:/glab/da0a89/1/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

9.12.1 Function Documentation

9.12.1.1 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 = sfeImageLoad(image_data.data(), image_data.size(), &image);
00043     utils::checkError(error);
00044   }
00045
00047   SFEIrisDetect detected_iris = {};
00048   { // STAGE 2: Detect iris in the image
00049
00050     // Detect iris in the image
00051     error = sfeIrisDetect(detector_solver, image, &detected_iris);
00052     utils::checkError(error);
00053
00054     if (detected_iris.confidence < 0.5) {
00055       throw std::runtime_error("No iris detected in the probe image.");
00056     }
00057
00058     std::cout « "Found iris in the image. Extracting template..." « std::endl;
00059   }
00061
00063   SFEIrisTemplate iris_template;
00064   { // STAGE 3 Extract probe iris template
00065     // Use template extraction solver to create new iris
00066     // template
00067     error = sfeIrisTemplateExtract(template_solver, image, &detected_iris,
00068                                    &iris_template);
00069     utils::checkError(error);
00070   }
00072   return iris_template;
00073 }
```

9.12.1.2 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 88 of file example_iris_identify.cpp.

```
00088                                   {
00089
00090   { // STAGE 0: Parse command line arguments
00091     // Map to store argument values
00092     std::unordered_map<std::string, std::string> args;
00093
00094     // Parse command line arguments
00095     for (int i = 1; i < argc; ++i) {
00096       std::string arg = argv[i];
00097       if (arg[0] == '-') {
00098         if (arg == "-h") {
00099           printHelp();
00100           return 0;
00101         }
00102         // Check if there's a next argument and it isn't another option
00103         if (i + 1 < argc && argv[i + 1][0] != '-') {
00104           args[arg] = argv[++i];
00105         } else {
00106           std::cerr << "Option " << arg << " requires a value." << std::endl;
00107           return 1;
00108         }
00109       } else {
00110         std::cerr << "Unknown option: " << arg << std::endl;
00111         return 1;
00112       }
00113     }
00114
00115     // Assign values to variables based on parsed arguments
00116     if (args.count("-p"))
00117       image_probe = args["-p"];
00118     if (args.count("-g"))
00119       image_match = args["-g"];
00120     if (args.count("-d"))
00121       solver_iris_detect = args["-d"];
00122     if (args.count("-e"))
00123       solver_iris_template = args["-e"];
00124     if (args.count("-i"))
00125       identification_threshold = std::stof(args["-i"]);
00126
00127     utils::printFormatted("EXAMPLE PARAMETERS");
00128     // Print resource names
00129     std::cout << "Probe image: " << image_probe << std::endl;
00130     std::cout << "Matching image: " << image_match << std::endl;
00131
00132     // Print solver names
00133     std::cout << "Iris detection solver: " << solver_iris_detect << std::endl;
00134     std::cout << "Iris template solver: " << solver_iris_template << std::endl;
00135
00136     // Print other options
00137     std::cout << "Identification threshold: " << identification_threshold
```

```
00138               « std::endl;
00139   }
00140
00141   utils::printToolkitInfo();
00142
00143   SFEError error{};
00144
00145   SFESolver detector_solver{};
00146   DEFER(sfeSolverFree(detector_solver));
00147   SFESolver landmarks_solver{};
00148   DEFER(sfeSolverFree(landmarks_solver));
00149   SFESolver template_solver{};
00150   DEFER(sfeSolverFree(template_solver));
00151   { // STAGE 1.1: loading solvers
00152     utils::printFormatted("LOADING SOLVERS");
00153     // Initialize Iris detection solver
00154     error = sfeSolverCreateWithParameters(
00155         solver_iris_detect.c_str(), SOLVER_PARAMETERS.data(),
00156         SOLVER_PARAMETERS.size(), &detector_solver);
00157     utils::checkError(error);
00158
00159     // Initialize template extraction solver
00160     error = sfeSolverCreateWithParameters(
00161         solver_iris_template.c_str(), SOLVER_PARAMETERS.data(),
00162         SOLVER_PARAMETERS.size(), &template_solver);
00163     utils::checkError(error);
00164   }
00165
00166   SFEIrisTemplate probe_iris_template;
00167   { // STAGE 1: Extract probe iris template
00168     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00169
00170     probe_iris_template =
00171         extract_template(image_probe, detector_solver, template_solver);
00172   }
00173
00174   { // STAGE 1.1: (optional) Print template attributes
00175     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00176
00178     SFEIrisTemplateVersion version = {};
00179     error = sfeIrisTemplateVersion(&probe_iris_template, &version);
00180     utils::checkError(error);
00181
00182     std::cout « "Version of the probe template: " « version.version_major
00183               « '.' « version.version_minor « std::endl;
00185
00187     float quality = {};
00188     error = sfeIrisTemplateQuality(&probe_iris_template, &quality);
00189     utils::checkError(error);
00190
00191     std::cout « "Quality of the probe template: " « quality « std::endl;
00193   }
00194
00195   size_t gallery_size = 10;
00196   std::vector<SFEIrisTemplate> iris_template_gallery(gallery_size);
00197   { // STAGE 2: Create iris templates gallery
00198     utils::printFormatted("IRIS GALLERY EXTRACTION");
00199
00200     auto matching_iris_template =
00201         extract_template(image_match, detector_solver, template_solver);
00202
00203     auto non_matching_iris_template =
00204         extract_template(image_non_match, detector_solver, template_solver);
00205
00206     // Fill the gallery with non-matching templates for demonstration purposes
00207     for (size_t i = 0; i < gallery_size; i++) {
00208       iris_template_gallery[i] = non_matching_iris_template;
00209     }
00210
00211     // Add matching template to the gallery at index 5
00212     iris_template_gallery[5] = matching_iris_template;
00213   }
00214
00216   size_t candidate_count = 1;
00217   std::vector<SFEIrisTemplateIdentificationCandidate> identification_results(
00218       candidate_count);
00219   { // STAGE 3: Identify the probe template
00220     utils::printFormatted("1:N IDENTIFICATION");
00221
00222     error = sfeIrisTemplateIdentify(
00223         &probe_iris_template, iris_template_gallery.data(),
00224         iris_template_gallery.size(), identification_threshold,
00225         identification_results.data(), &candidate_count, 4);
00226     utils::checkError(error);
00227
00228     if (candidate_count == 0) {
00229       std::cout « "No candidates found above identification threshold "
```

```
00230                 « identification_threshold « std::endl;
00231       return 0;
00232     }
00233
00234     std::cout « "Found " « identification_results.size()
00235               « " candidates above identification threshold "
00236               « identification_threshold « std::endl;
00237     for (auto &result : identification_results)
00238       std::cout « "Template index: #" « result.index
00239                 « ", score: " « result.score « std::endl;
00240   }
00242
00244   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00245     utils::printFormatted("1:1 MATCHING");
00246
00247     if (identification_results.size() == 0) {
00248       std::cout « "No candidates found above identification threshold "
00249                 « identification_threshold « std::endl;
00250       return 0;
00251     }
00252
00253     // Since it's sorted vector by score, the best candidate is the first one
00254     auto match_iris_template =
00255         iris_template_gallery[identification_results[0].index];
00256
00257     float match_confidence;
00258     error = sfeIrisTemplateMatch(&probe_iris_template, &match_iris_template,
00259                                  &match_confidence);
00260     utils::checkError(error);
00261
00262     std::cout
00263         « "Matching score of probe template with the best template, score: "
00264         « match_confidence « std::endl;
00265   }
00267
00268   utils::printFormatted("FINISHED");
00269 }
```

9.12.1.3 printHelp()

```
void printHelp ( )
```

Definition at line 75 of file example_iris_identify.cpp.

```
00075                  {
00076   std::cout « "Help: Usage of the program." « std::endl;
00077   std::cout « "Options:" « std::endl;
00078   std::cout « "-h: Display help." « std::endl;
00079   std::cout « "-p: Probe image file." « std::endl;
00080   std::cout « "-g: Matching image file. Our \"iris database\"." « std::endl;
00081   std::cout « "-d: Path to detector solver." « std::endl;
00082   std::cout « "-e: Path to extraction solver." « std::endl;
00083   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00084   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00085 }
```

9.12.2 Variable Documentation

9.12.2.1 identification_threshold

```
float identification_threshold = 0.6f
```

Definition at line 28 of file example_iris_identify.cpp.

9.12.2.2 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.

9.12.2.3 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.

9.12.2.4 image_probe

```
std::string image_probe = ¨assets/iris/iris1_0.png¨
```

Definition at line 19 of file example_iris_identify.cpp.

9.12.2.5 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.

9.12.2.6 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.

9.12.2.7 SOLVER_PARAMETERS

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

Definition at line 26 of file example_iris_identify.cpp.

9.13 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 = sfeImageLoad(image_data.data(), image_data.size(), &image);
00043     utils::checkError(error);
00044   }
00045
00047   SFEIrisDetect detected_iris = {};
00048   { // STAGE 2: Detect iris in the image
00049
00050     // Detect iris in the image
00051     error = sfeIrisDetect(detector_solver, image, &detected_iris);
00052     utils::checkError(error);
00053
00054     if (detected_iris.confidence < 0.5) {
00055       throw std::runtime_error("No iris detected in the probe image.");
00056     }
00057
00058     std::cout « "Found iris in the image. Extracting template..." « std::endl;
00059   }
00061
00063   SFEIrisTemplate iris_template;
00064   { // STAGE 3 Extract probe iris template
00065     // Use template extraction solver to create new iris
00066     // template
00067     error = sfeIrisTemplateExtract(template_solver, image, &detected_iris,
00068                                    &iris_template);
00069     utils::checkError(error);
00070   }
00072   return iris_template;
00073 }
00074
00075 void printHelp() {
00076   std::cout « "Help: Usage of the program." « std::endl;
00077   std::cout « "Options:" « std::endl;
00078   std::cout « "-h: Display help." « std::endl;
00079   std::cout « "-p: Probe image file." « std::endl;
00080   std::cout « "-g: Matching image file. Our \"iris database\"." « std::endl;
00081   std::cout « "-d: Path to detector solver." « std::endl;
00082   std::cout « "-e: Path to extraction solver." « std::endl;
00083   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00084   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00085 }
00086
00088 int main(int argc, char *argv[]) {
00089
00090   { // STAGE 0: Parse command line arguments
00091     // Map to store argument values
```

```
00092     std::unordered_map<std::string, std::string> args;
00093
00094     // Parse command line arguments
00095     for (int i = 1; i < argc; ++i) {
00096       std::string arg = argv[i];
00097       if (arg[0] == '-') {
00098         if (arg == "-h") {
00099           printHelp();
00100           return 0;
00101         }
00102         // Check if there's a next argument and it isn't another option
00103         if (i + 1 < argc && argv[i + 1][0] != '-') {
00104           args[arg] = argv[++i];
00105         } else {
00106           std::cerr « "Option " « arg « " requires a value." « std::endl;
00107           return 1;
00108         }
00109       } else {
00110         std::cerr « "Unknown option: " « arg « std::endl;
00111         return 1;
00112       }
00113     }
00114
00115     // Assign values to variables based on parsed arguments
00116     if (args.count("-p"))
00117       image_probe = args["-p"];
00118     if (args.count("-g"))
00119       image_match = args["-g"];
00120     if (args.count("-d"))
00121       solver_iris_detect = args["-d"];
00122     if (args.count("-e"))
00123       solver_iris_template = args["-e"];
00124     if (args.count("-i"))
00125       identification_threshold = std::stof(args["-i"]);
00126
00127     utils::printFormatted("EXAMPLE PARAMETERS");
00128     // Print resource names
00129     std::cout « "Probe image: " « image_probe « std::endl;
00130     std::cout « "Matching image: " « image_match « std::endl;
00131
00132     // Print solver names
00133     std::cout « "Iris detection solver: " « solver_iris_detect « std::endl;
00134     std::cout « "Iris template solver: " « solver_iris_template « std::endl;
00135
00136     // Print other options
00137     std::cout « "Identification threshold: " « identification_threshold
00138               « std::endl;
00139   }
00140
00141   utils::printToolkitInfo();
00142
00143   SFEError error{};
00144
00145   SFESolver detector_solver{};
00146   DEFER(sfeSolverFree(detector_solver));
00147   SFESolver landmarks_solver{};
00148   DEFER(sfeSolverFree(landmarks_solver));
00149   SFESolver template_solver{};
00150   DEFER(sfeSolverFree(template_solver));
00151   { // STAGE 1.1: loading solvers
00152     utils::printFormatted("LOADING SOLVERS");
00153     // Initialize Iris detection solver
00154     error = sfeSolverCreateWithParameters(
00155         solver_iris_detect.c_str(), SOLVER_PARAMETERS.data(),
00156         SOLVER_PARAMETERS.size(), &detector_solver);
00157     utils::checkError(error);
00158
00159     // Initialize template extraction solver
00160     error = sfeSolverCreateWithParameters(
00161         solver_iris_template.c_str(), SOLVER_PARAMETERS.data(),
00162         SOLVER_PARAMETERS.size(), &template_solver);
00163     utils::checkError(error);
00164   }
00165
00166   SFEIrisTemplate probe_iris_template;
00167   { // STAGE 1: Extract probe iris template
00168     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00169
00170     probe_iris_template =
00171         extract_template(image_probe, detector_solver, template_solver);
00172   }
00173
00174   { // STAGE 1.1: (optional) Print template attributes
00175     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00176
00178     SFEIrisTemplateVersion version = {};
00179     error = sfeIrisTemplateVersion(&probe_iris_template, &version);
```

```
00180     utils::checkError(error);
00181
00182     std::cout « "Version of the probe template: " « version.version_major
00183               « '.' « version.version_minor « std::endl;
00185
00187     float quality = {};
00188     error = sfeIrisTemplateQuality(&probe_iris_template, &quality);
00189     utils::checkError(error);
00190
00191     std::cout « "Quality of the probe template: " « quality « std::endl;
00193   }
00194
00195   size_t gallery_size = 10;
00196   std::vector<SFEIrisTemplate> iris_template_gallery(gallery_size);
00197   { // STAGE 2: Create iris templates gallery
00198     utils::printFormatted("IRIS GALLERY EXTRACTION");
00199
00200     auto matching_iris_template =
00201         extract_template(image_match, detector_solver, template_solver);
00202
00203     auto non_matching_iris_template =
00204         extract_template(image_non_match, detector_solver, template_solver);
00205
00206     // Fill the gallery with non-matching templates for demonstration purposes
00207     for (size_t i = 0; i < gallery_size; i++) {
00208       iris_template_gallery[i] = non_matching_iris_template;
00209     }
00210
00211     // Add matching template to the gallery at index 5
00212     iris_template_gallery[5] = matching_iris_template;
00213   }
00214
00216   size_t candidate_count = 1;
00217   std::vector<SFEIrisTemplateIdentificationCandidate> identification_results(
00218       candidate_count);
00219   { // STAGE 3: Identify the probe template
00220     utils::printFormatted("1:N IDENTIFICATION");
00221
00222     error = sfeIrisTemplateIdentify(
00223         &probe_iris_template, iris_template_gallery.data(),
00224         iris_template_gallery.size(), identification_threshold,
00225         identification_results.data(), &candidate_count, 4);
00226     utils::checkError(error);
00227
00228     if (candidate_count == 0) {
00229       std::cout « "No candidates found above identification threshold "
00230                 « identification_threshold « std::endl;
00231       return 0;
00232     }
00233
00234     std::cout « "Found " « identification_results.size()
00235               « " candidates above identification threshold "
00236               « identification_threshold « std::endl;
00237     for (auto &result : identification_results)
00238       std::cout « "Template index: #" « result.index
00239                 « ", score: " « result.score « std::endl;
00240   }
00242
00244   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00245     utils::printFormatted("1:1 MATCHING");
00246
00247     if (identification_results.size() == 0) {
00248       std::cout « "No candidates found above identification threshold "
00249                 « identification_threshold « std::endl;
00250       return 0;
00251     }
00252
00253     // Since it's sorted vector by score, the best candidate is the first one
00254     auto match_iris_template =
00255         iris_template_gallery[identification_results[0].index];
00256
00257     float match_confidence;
00258     error = sfeIrisTemplateMatch(&probe_iris_template, &match_iris_template,
00259                                  &match_confidence);
00260     utils::checkError(error);
00261
00262     std::cout
00263         « "Matching score of probe template with the best template, score: "
00264         « match_confidence « std::endl;
00265   }
00267
00268   utils::printFormatted("FINISHED");
00269 }
```

9.14 D:/glab/da0a89/1/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 &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
- const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{}
- float identification_threshold = 0.6f

9.14.1 Function Documentation

9.14.1.1 extract_template()

```
SFEPalmTemplate extract_template (
            std::string image_path,
            SFESolver & detector_solver,
            SFESolver & extraction_solver )
```

[Palm Detect]

[Palm Detect]

[Palm Template Extract]

[Palm Template Extract]

Definition at line 30 of file example_palm_identify.cpp.

```
00031                                                                                                {
00032
00033   SFEImage image{};
00034   DEFER(sfeImageFree(image));
00035   SFEError error{};
00036   { // STAGE 1: Load image
00037     // Load image data from file
00038     auto image_data = utils::readFile(image_path);
00039
00040     // Decode image from data
00041     error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00042     utils::checkError(error);
00043   }
00044
00046   SFEPalmDetect detected_palm = {};
00047   { // STAGE 2: Detect palm in the image
00048
00049     // Detect palm in the image
00050     float detection_threshold = 0.1f;
00051     size_t detected_count = 1;
00052     error = sfePalmDetect(detector_solver, image, detection_threshold,
00053                           &detected_palm, &detected_count);
00054     utils::checkError(error);
00055
00056     if (detected_count > 0 && detected_palm.confidence < 0.5) {
00057       throw std::runtime_error("No palm detected in the probe image.");
00058     }
00059
00060     std::cout « "Found palm in the image. Extracting template..." « std::endl;
00061   }
00063
00065   SFEPalmTemplate palm_template;
00066   { // STAGE 3 Extract probe palm template
00067     // Use template extraction solver to create new palm
00068     // template
00069     error = sfePalmTemplateExtract(extraction_solver, image, &detected_palm,
00070                                    &palm_template);
00071     utils::checkError(error);
00072   }
00074   return palm_template;
00075 }
```

9.14.1.2 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 90 of file example_palm_identify.cpp.

```
00090                                    {
00091
00092   { // STAGE 0: Parse command line arguments
00093     // Map to store argument values
00094     std::unordered_map<std::string, std::string> args;
00095
00096     // Parse command line arguments
00097     for (int i = 1; i < argc; ++i) {
00098       std::string arg = argv[i];
00099       if (arg[0] == '-') {
```

```
00100         if (arg == "-h") {
00101           printHelp();
00102           return 0;
00103         }
00104         // Check if there's a next argument and it isn't another option
00105         if (i + 1 < argc && argv[i + 1][0] != '-') {
00106           args[arg] = argv[++i];
00107         } else {
00108           std::cerr << "Option " << arg << " requires a value." << std::endl;
00109           return 1;
00110         }
00111       } else {
00112         std::cerr << "Unknown option: " << arg << std::endl;
00113         return 1;
00114       }
00115     }
00116
00117     // Assign values to variables based on parsed arguments
00118     if (args.count("-p"))
00119       image_probe = args["-p"];
00120     if (args.count("-g"))
00121       image_match = args["-g"];
00122     if (args.count("-d"))
00123       solver_palm_detect = args["-d"];
00124     if (args.count("-e"))
00125       solver_palm_extraction = args["-e"];
00126     if (args.count("-i"))
00127       identification_threshold = std::stof(args["-i"]);
00128
00129     utils::printFormatted("EXAMPLE PARAMETERS");
00130     // Print resource names
00131     std::cout << "Probe image: " << image_probe << std::endl;
00132     std::cout << "Matching image: " << image_match << std::endl;
00133
00134     // Print solver names
00135     std::cout << "Palm detection solver: " << solver_palm_detect << std::endl;
00136     std::cout << "Palm extraction solver: " << solver_palm_extraction << std::endl;
00137
00138     // Print other options
00139     std::cout << "Identification threshold: " << identification_threshold
00140               << std::endl;
00141   }
00142
00143   utils::printToolkitInfo();
00144
00145   SFEError error{};
00146   SFESolver detector_solver{};
00147   DEFER(sfeSolverFree(detector_solver));
00148   SFESolver extraction_solver{};
00149   DEFER(sfeSolverFree(extraction_solver));
00150   { // STAGE 1.1: loading solvers
00151     utils::printFormatted("LOADING SOLVERS");
00152     // Initialize Palm detection solver
00153     error = sfeSolverCreateWithParameters(
00154         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00155         SOLVER_PARAMETERS.size(), &detector_solver);
00156     utils::checkError(error);
00157     error = sfeSolverCreateWithParameters(
00158         solver_palm_extraction.c_str(), SOLVER_PARAMETERS.data(),
00159         SOLVER_PARAMETERS.size(), &extraction_solver);
00160     utils::checkError(error);
00161   }
00162
00163   SFEPalmTemplate probe_palm_template;
00164   { // STAGE 1: Extract probe palm template
00165     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
00166
00167     probe_palm_template = extract_template(image_probe, detector_solver, extraction_solver);
00168   }
00169
00170   { // STAGE 1.1: (optional) Print template attributes
00171     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00172
00174     SFEPalmTemplateVersion version = {};
00175     error = sfePalmTemplateVersion(&probe_palm_template, &version);
00176     utils::checkError(error);
00177
00178     std::cout << "Version of the probe template: " << version.major << '.'
00179               << version.minor << '.' << version.patch << std::endl;
00181   }
00182
00183   size_t gallery_size = 10;
00184   std::vector<SFEPalmTemplate> palm_template_gallery(gallery_size);
00185   { // STAGE 2: Create palm templates gallery
00186     utils::printFormatted("PALM GALLERY EXTRACTION");
00187
00188     auto matching_palm_template =
```

```
00189         extract_template(image_match, detector_solver, extraction_solver);
00190
00191     auto non_matching_palm_template =
00192         extract_template(image_non_match, detector_solver, extraction_solver);
00193
00194     // Fill the gallery with non-matching templates for demonstration purposes
00195     for (size_t i = 0; i < gallery_size; i++) {
00196       palm_template_gallery[i] = non_matching_palm_template;
00197     }
00198
00199     // Add matching template to the gallery at index 5
00200     palm_template_gallery[5] = matching_palm_template;
00201   }
00202
00204   size_t candidate_count = 1;
00205   std::vector<SFEPalmTemplateIdentificationCandidate> identification_results(
00206       candidate_count);
00207   { // STAGE 3: Identify the probe template
00208     utils::printFormatted("1:N IDENTIFICATION");
00209
00210     error = sfePalmTemplateIdentify(
00211         &probe_palm_template, palm_template_gallery.data(),
00212         palm_template_gallery.size(), identification_threshold,
00213         identification_results.data(), &candidate_count, 4);
00214     utils::checkError(error);
00215
00216     if (candidate_count == 0) {
00217       std::cout « "No candidates found above identification threshold "
00218                 « identification_threshold « std::endl;
00219       return 0;
00220     }
00221
00222     std::cout « "Found " « identification_results.size()
00223               « " candidates above identification threshold "
00224               « identification_threshold « std::endl;
00225     for (auto &result : identification_results)
00226       std::cout « "Template index: #" « result.index
00227                 « ", score: " « result.score « std::endl;
00228   }
00230
00232   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00233     utils::printFormatted("1:1 MATCHING");
00234
00235     if (identification_results.size() == 0) {
00236       std::cout « "No candidates found above identification threshold "
00237                 « identification_threshold « std::endl;
00238       return 0;
00239     }
00240
00241     // Since it's sorted vector by score, the best candidate is the first one
00242     auto match_palm_template =
00243         palm_template_gallery[identification_results[0].index];
00244
00245     float match_confidence;
00246     error = sfePalmTemplateMatch(&probe_palm_template, &match_palm_template,
00247                                  &match_confidence);
00248     utils::checkError(error);
00249
00250     std::cout
00251         « "Matching score of probe template with the best template, score: "
00252         « match_confidence « std::endl;
00253   }
00255
00256   utils::printFormatted("FINISHED");
00257 }
```

9.14.1.3 printHelp()

```
void printHelp ( )
```

Definition at line 77 of file example_palm_identify.cpp.

```
00077                 {
00078   std::cout « "Help: Usage of the program." « std::endl;
00079   std::cout « "Options:" « std::endl;
00080   std::cout « "-h: Display help." « std::endl;
00081   std::cout « "-p: Probe image file." « std::endl;
00082   std::cout « "-g: Matching image file. Our \"palm database\"." « std::endl;
00083   std::cout « "-d: Path to detector solver." « std::endl;
00084   std::cout « "-e: Path to template extraction solver." « std::endl;
00085   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00086   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00087 }
```

9.14.2 Variable Documentation

9.14.2.1 identification_threshold

```
float identification_threshold = 0.6f
```

Definition at line 28 of file example_palm_identify.cpp.

9.14.2.2 image_match

```
std::string image_match = ¨assets/palm/palm_id1_r1.jpg¨
```

Definition at line 18 of file example_palm_identify.cpp.

9.14.2.3 image_non_match

```
std::string image_non_match = ¨assets/palm/palm_id2_l1.jpg¨
```

Definition at line 20 of file example_palm_identify.cpp.

9.14.2.4 image_probe

```
std::string image_probe = ¨assets/palm/palm_id1_r2.jpg¨
```

Definition at line 19 of file example_palm_identify.cpp.

9.14.2.5 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_identify.cpp.

Definition at line 23 of file example_palm_identify.cpp.

9.14.2.6 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.

9.14.2.7 SOLVER_PARAMETERS

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

Definition at line 26 of file example_palm_identify.cpp.

9.15 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 const auto SOLVER_PARAMETERS = std::vector<SFESolverParameter>{};
00027
00028 float identification_threshold = 0.6f;
00029
00030 SFEPalmTemplate extract_template(std::string image_path,
00031                                  SFESolver &detector_solver, SFESolver &extraction_solver) {
00032
00033   SFEImage image{};
00034   DEFER(sfeImageFree(image));
00035   SFEError error{};
00036   { // STAGE 1: Load image
00037     // Load image data from file
00038     auto image_data = utils::readFile(image_path);
00039
00040     // Decode image from data
00041     error = sfeImageLoad(image_data.data(), image_data.size(), &image);
00042     utils::checkError(error);
00043   }
00044
00046   SFEPalmDetect detected_palm = {};
00047   { // STAGE 2: Detect palm in the image
00048
00049     // Detect palm in the image
00050     float detection_threshold = 0.1f;
00051     size_t detected_count = 1;
00052     error = sfePalmDetect(detector_solver, image, detection_threshold,
00053                           &detected_palm, &detected_count);
00054     utils::checkError(error);
00055
00056     if (detected_count > 0 && detected_palm.confidence < 0.5) {
00057       throw std::runtime_error("No palm detected in the probe image.");
00058     }
00059
00060     std::cout « "Found palm in the image. Extracting template..." « std::endl;
00061   }
00063
00065   SFEPalmTemplate palm_template;
00066   { // STAGE 3 Extract probe palm template
00067     // Use template extraction solver to create new palm
00068     // template
00069     error = sfePalmTemplateExtract(extraction_solver, image, &detected_palm,
00070                                    &palm_template);
00071     utils::checkError(error);
00072   }
00074   return palm_template;
00075 }
00076
00077 void printHelp() {
```

```
00078   std::cout « "Help: Usage of the program." « std::endl;
00079   std::cout « "Options:" « std::endl;
00080   std::cout « "-h: Display help." « std::endl;
00081   std::cout « "-p: Probe image file." « std::endl;
00082   std::cout « "-g: Matching image file. Our \"palm database\"." « std::endl;
00083   std::cout « "-d: Path to detector solver." « std::endl;
00084   std::cout « "-e: Path to template extraction solver." « std::endl;
00085   std::cout « "-t: Detection threshold. <0,1>" « std::endl;
00086   std::cout « "-i: Identification threshold. <0,1>" « std::endl;
00087 }
00088
00090 int main(int argc, char *argv[]) {
00091
00092   { // STAGE 0: Parse command line arguments
00093     // Map to store argument values
00094     std::unordered_map<std::string, std::string> args;
00095
00096     // Parse command line arguments
00097     for (int i = 1; i < argc; ++i) {
00098       std::string arg = argv[i];
00099       if (arg[0] == '-') {
00100         if (arg == "-h") {
00101           printHelp();
00102           return 0;
00103         }
00104         // Check if there's a next argument and it isn't another option
00105         if (i + 1 < argc && argv[i + 1][0] != '-') {
00106           args[arg] = argv[++i];
00107         } else {
00108           std::cerr « "Option " « arg « " requires a value." « std::endl;
00109           return 1;
00110         }
00111       } else {
00112         std::cerr « "Unknown option: " « arg « std::endl;
00113         return 1;
00114       }
00115     }
00116
00117     // Assign values to variables based on parsed arguments
00118     if (args.count("-p"))
00119       image_probe = args["-p"];
00120     if (args.count("-g"))
00121       image_match = args["-g"];
00122     if (args.count("-d"))
00123       solver_palm_detect = args["-d"];
00124     if (args.count("-e"))
00125       solver_palm_extraction = args["-e"];
00126     if (args.count("-i"))
00127       identification_threshold = std::stof(args["-i"]);
00128
00129     utils::printFormatted("EXAMPLE PARAMETERS");
00130     // Print resource names
00131     std::cout « "Probe image: " « image_probe « std::endl;
00132     std::cout « "Matching image: " « image_match « std::endl;
00133
00134     // Print solver names
00135     std::cout « "Palm detection solver: " « solver_palm_detect « std::endl;
00136     std::cout « "Palm extraction solver: " « solver_palm_extraction « std::endl;
00137
00138     // Print other options
00139     std::cout « "Identification threshold: " « identification_threshold
00140               « std::endl;
00141   }
00142
00143   utils::printToolkitInfo();
00144
00145   SFEError error{};
00146   SFESolver detector_solver{};
00147   DEFER(sfeSolverFree(detector_solver));
00148   SFESolver extraction_solver{};
00149   DEFER(sfeSolverFree(extraction_solver));
00150   { // STAGE 1.1: loading solvers
00151     utils::printFormatted("LOADING SOLVERS");
00152     // Initialize Palm detection solver
00153     error = sfeSolverCreateWithParameters(
00154         solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
00155         SOLVER_PARAMETERS.size(), &detector_solver);
00156     utils::checkError(error);
00157     error = sfeSolverCreateWithParameters(
00158         solver_palm_extraction.c_str(), SOLVER_PARAMETERS.data(),
00159         SOLVER_PARAMETERS.size(), &extraction_solver);
00160     utils::checkError(error);
00161   }
00162
00163   SFEPalmTemplate probe_palm_template;
00164   { // STAGE 1: Extract probe palm template
00165     utils::printFormatted("PROBE TEMPLATE EXTRACTION");
```

```
00166
00167     probe_palm_template = extract_template(image_probe, detector_solver, extraction_solver);
00168   }
00169
00170   { // STAGE 1.1: (optional) Print template attributes
00171     utils::printFormatted("PROBE TEMPLATE ATTRIBUTES");
00172
00174     SFEPalmTemplateVersion version = {};
00175     error = sfePalmTemplateVersion(&probe_palm_template, &version);
00176     utils::checkError(error);
00177
00178     std::cout « "Version of the probe template: " « version.major « '.'
00179               « version.minor « '.' « version.patch « std::endl;
00181   }
00182
00183   size_t gallery_size = 10;
00184   std::vector<SFEPalmTemplate> palm_template_gallery(gallery_size);
00185   { // STAGE 2: Create palm templates gallery
00186     utils::printFormatted("PALM GALLERY EXTRACTION");
00187
00188     auto matching_palm_template =
00189         extract_template(image_match, detector_solver, extraction_solver);
00190
00191     auto non_matching_palm_template =
00192         extract_template(image_non_match, detector_solver, extraction_solver);
00193
00194     // Fill the gallery with non-matching templates for demonstration purposes
00195     for (size_t i = 0; i < gallery_size; i++) {
00196       palm_template_gallery[i] = non_matching_palm_template;
00197     }
00198
00199     // Add matching template to the gallery at index 5
00200     palm_template_gallery[5] = matching_palm_template;
00201   }
00202
00204   size_t candidate_count = 1;
00205   std::vector<SFEPalmTemplateIdentificationCandidate> identification_results(
00206       candidate_count);
00207   { // STAGE 3: Identify the probe template
00208     utils::printFormatted("1:N IDENTIFICATION");
00209
00210     error = sfePalmTemplateIdentify(
00211         &probe_palm_template, palm_template_gallery.data(),
00212         palm_template_gallery.size(), identification_threshold,
00213         identification_results.data(), &candidate_count, 4);
00214     utils::checkError(error);
00215
00216     if (candidate_count == 0) {
00217       std::cout « "No candidates found above identification threshold "
00218                 « identification_threshold « std::endl;
00219       return 0;
00220     }
00221
00222     std::cout « "Found " « identification_results.size()
00223               « " candidates above identification threshold "
00224               « identification_threshold « std::endl;
00225     for (auto &result : identification_results)
00226       std::cout « "Template index: #" « result.index
00227                 « ", score: " « result.score « std::endl;
00228   }
00230
00232   { // STAGE 4: 1:1 matching with top candidate to showcase 1:1 matching
00233     utils::printFormatted("1:1 MATCHING");
00234
00235     if (identification_results.size() == 0) {
00236       std::cout « "No candidates found above identification threshold "
00237                 « identification_threshold « std::endl;
00238       return 0;
00239     }
00240
00241     // Since it's sorted vector by score, the best candidate is the first one
00242     auto match_palm_template =
00243         palm_template_gallery[identification_results[0].index];
00244
00245     float match_confidence;
00246     error = sfePalmTemplateMatch(&probe_palm_template, &match_palm_template,
00247                                  &match_confidence);
00248     utils::checkError(error);
00249
00250     std::cout
00251         « "Matching score of probe template with the best template, score: "
00252         « match_confidence « std::endl;
00253   }
00255
00256   utils::printFormatted("FINISHED");
00257 }
```

9.16 D:/glab/da0a89/1/include/sfe_core.h File Reference

File containing API of sfe_core library as part of SFE Toolkit.

```
#include <stddef.h>
#include <stdint.h>
```

Data Structures

- struct SFEImage

  Raw owned raster image representation, HWC|BGR order.
- struct SFEBbox

  Bounding box.
- struct SFEOrientedBbox

  Oriented bounding box.
- struct SFESolverParameter

  Solver parameter used to configure solvers.
- struct SFEEntity

  Face 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.

Typedefs

- typedef enum SFEHwidMethod SFEHwidMethod

  Supported methods for Hardware ID (HWID) computation.
- typedef enum SFELicenseSource SFELicenseSource

  License sources.
- 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 SFEBbox SFEBbox

  Bounding box.
- typedef struct SFEOrientedBbox SFEOrientedBbox

  Oriented bounding box.
- 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

  Face 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.

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 SFELicenseSource {
  SFE_LICENSE_SOURCE_NONE = 0 , SFE_LICENSE_SOURCE_FILE = 1 , SFE_LICENSE_SOURCE_MEMORY = 2 , SFE_LICENSE_SOURCE_TOKEN = 3 ,
  SFE_LICENSE_SOURCE_ENV = 4 }

  License sources.
- enum SFEImageFormat {
  SFE_IMAGE_FORMAT_PNG = 0 , SFE_IMAGE_FORMAT_JPEG = 1 , SFE_IMAGE_FORMAT_GIF = 2 ,
  SFE_IMAGE_FORMAT_WEBP = 3 ,
  SFE_IMAGE_FORMAT_PNM = 4 , SFE_IMAGE_FORMAT_TIFF = 5 , SFE_IMAGE_FORMAT_TGA = 6 ,
  SFE_IMAGE_FORMAT_DDS = 7 ,
  SFE_IMAGE_FORMAT_BMP = 8 , SFE_IMAGE_FORMAT_ICO = 9 , SFE_IMAGE_FORMAT_HDR = 10 ,
  SFE_IMAGE_FORMAT_OPENEXR = 11 ,
  SFE_IMAGE_FORMAT_FARBFELD = 12 , SFE_IMAGE_FORMAT_AVIF = 13 }

  Supported image formats.

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 (SFELicenseSource *out_license_source)

  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 and SFELicenseSource will be populated.
- 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 sfeSolverCreateWithParameters (const char *solver_file, const SFESolverParameter *solver_↩ parameters, size_t solver_parameters_count, SFESolver *out_solver)

  Create new solver from solver file.
- SFEError sfeSolverCreate (const char *solver_file, SFESolver *out_solver)

  Create new solver from solver file.
- void sfeSolverFree (SFESolver solver)

  Free memory associated with `SFESolver`.
- SFEError sfeImageLoad (const unsigned char *data, size_t data_len, SFEImage *out_image)

  Load `SFEImage` from raw image data of various formats. Eg. PNG, JPEG ..
- SFEError sfeImageSave (SFEImageView image, SFEImageFormat image_fromat, unsigned char *out_data, size_t *in_out_data_len)

Save SFEImage into a buffer encoding it to specified image_format.

- SFEError sfeImageInfo (const unsigned char *data, size_t data_len, size_t *out_width, size_t *out_height, SFEImageFormat *out_image_format)

  Try to get information from image data without fully decoding it.
- void sfeImageFree (SFEImage image)

  Free memory associated with SFEImage.
- SFEError sfeImageColorTranspose (SFEImageView 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.

9.16.1 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.

9.16.2 Typedef Documentation

9.16.2.1 SFEBbox

```
typedef struct SFEBbox SFEBbox
```

Bounding box.

9.16.2.2 SFEEntity

```
typedef struct SFEEntity SFEEntity
```

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

9.16.2.3 SFEEntityIdentificationCandidate

```
typedef struct SFEEntityIdentificationCandidate SFEEntityIdentificationCandidate
```

Candidate for identification by entity.

9.16.2.4 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 121 of file sfe_core.h.

9.16.2.5 SFEHwidMethod

```
typedef enum SFEHwidMethod SFEHwidMethod
```

Supported methods for Hardware ID (HWID) computation.

9.16.2.6 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.

9.16.2.7 SFEImageFormat

```
typedef enum SFEImageFormat SFEImageFormat
```

Supported image formats.

9.16.2.8 SFEImageView

```
typedef SFEImage SFEImageView
```

Raw raster image view, HWC|BGR order.

Note

For owning variant use `SFEImage`.

Definition at line 80 of file sfe_core.h.

9.16.2.9 SFELicenseSource

```
typedef enum SFELicenseSource SFELicenseSource
```

License sources.

9.16.2.10 SFEOrientedBbox

```
typedef struct SFEOrientedBbox SFEOrientedBbox
```

Oriented bounding box.

9.16.2.11 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 126 of file sfe_core.h.

9.16.2.12 SFESolverParameter

```
typedef struct SFESolverParameter SFESolverParameter
```

Solver parameter used to configure solvers.

9.16.2.13 SFETemplateIdentificationCandidate

```
typedef struct SFETemplateIdentificationCandidate SFETemplateIdentificationCandidate
```

Candidate for identification by template.

9.16.3 Enumeration Type Documentation

9.16.3.1 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 18 of file sfe_core.h.

```
00018                             {
00019   SFE_HWID_METHOD_AUTO = 0,
00020   SFE_HWID_METHOD_DISK_ID = 1,
00021   SFE_HWID_METHOD_MAC = 2,
00022   SFE_HWID_METHOD_SERIAL_NUMBER = 3,
00023   SFE_HWID_METHOD_IMEI = 4,
00024   SFE_HWID_METHOD_SM_BIOS = 5,
00025   SFE_HWID_METHOD_AMAZON = 6,
00026   SFE_HWID_METHOD_APP_ID = 7,
00027   SFE_HWID_METHOD_PHY = 8,
00028   SFE_HWID_METHOD_MAC_ADDRESS = 9,
00029   SFE_HWID_METHOD_SYS = 10,
00030   SFE_HWID_METHOD_ANDROID_ID = 11,
00031 } SFEHwidMethod;
```

9.16.3.2 SFEImageFormat

```
enum SFEImageFormat
```

Supported image formats.

Enumerator

SFE_IMAGE_FORMAT_PNG	
SFE_IMAGE_FORMAT_JPEG	
SFE_IMAGE_FORMAT_GIF	
SFE_IMAGE_FORMAT_WEBP	
SFE_IMAGE_FORMAT_PNM	
SFE_IMAGE_FORMAT_TIFF	
SFE_IMAGE_FORMAT_TGA	
SFE_IMAGE_FORMAT_DDS	
SFE_IMAGE_FORMAT_BMP	
SFE_IMAGE_FORMAT_ICO	
SFE_IMAGE_FORMAT_HDR	
SFE_IMAGE_FORMAT_OPENEXR	
SFE_IMAGE_FORMAT_FARBFELD	
SFE_IMAGE_FORMAT_AVIF	

Definition at line 48 of file sfe_core.h.

```
00048                             {
00049   SFE_IMAGE_FORMAT_PNG = 0,
00050   SFE_IMAGE_FORMAT_JPEG = 1,
00051   SFE_IMAGE_FORMAT_GIF = 2,
00052   SFE_IMAGE_FORMAT_WEBP = 3,
00053   SFE_IMAGE_FORMAT_PNM = 4,
00054   SFE_IMAGE_FORMAT_TIFF = 5,
00055   SFE_IMAGE_FORMAT_TGA = 6,
00056   SFE_IMAGE_FORMAT_DDS = 7,
00057   SFE_IMAGE_FORMAT_BMP = 8,
00058   SFE_IMAGE_FORMAT_ICO = 9,
00059   SFE_IMAGE_FORMAT_HDR = 10,
00060   SFE_IMAGE_FORMAT_OPENEXR = 11,
00061   SFE_IMAGE_FORMAT_FARBFELD = 12,
00062   SFE_IMAGE_FORMAT_AVIF = 13,
00063 } SFEImageFormat;
```

9.16.3.3 SFELicenseSource

```
enum SFELicenseSource
```

License sources.

Enumerator

SFE_LICENSE_SOURCE_NONE	License was not read.
SFE_LICENSE_SOURCE_FILE	License was read from a file.
SFE_LICENSE_SOURCE_MEMORY	License was read from memory.
SFE_LICENSE_SOURCE_TOKEN	License was read from a USB token.
SFE_LICENSE_SOURCE_ENV	License was read from an environment variable.

Definition at line 34 of file sfe_core.h.

```
00034                               {
00036   SFE_LICENSE_SOURCE_NONE = 0,
00038   SFE_LICENSE_SOURCE_FILE = 1,
00040   SFE_LICENSE_SOURCE_MEMORY = 2,
00042   SFE_LICENSE_SOURCE_TOKEN = 3,
00044   SFE_LICENSE_SOURCE_ENV = 4,
00045 } SFELicenseSource;
```

9.16.4 Function Documentation

9.16.4.1 sfeErrorFree()

```
void sfeErrorFree (
            SFEError error )
```

Free memory associated with `SFEError`.

Parameters

in	*error*	Error to free.

9.16.4.2 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.

9.16.4.3 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).

9.16.4.4 sfeImageColorTranspose()

```
SFEError sfeImageColorTranspose (
            SFEImageView image )
```

Transpose RGB<->BGR color channel order.

Parameters

in	*image*	Image to color transpose.

Returns

Error

9.16.4.5 sfeImageFree()

```
void sfeImageFree (
            SFEImage image )
```

Free memory associated with SFEImage.

Parameters

in	*image*	Image to free.

Examples

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

9.16.4.6 sfeImageInfo()

```
SFEError sfeImageInfo (
            const unsigned char * data,
            size_t data_len,
            size_t * out_width,
            size_t * out_height,
            SFEImageFormat * out_image_format )
```

Try to get information from image data without fully decoding it.

Parameters

in	*data*	Image data to load image from.
in	*data_len*	Size of the image data in bytes.
out	*out_width*	Width of the image data.
out	*out_height*	Height of the image data.
out	*out_image_format*	Format of the image data.

Returns

Error in case of unknown image type.

9.16.4.7 sfeImageLoad()

```
SFEError sfeImageLoad (
            const unsigned char * data,
            size_t data_len,
            SFEImage * out_image )
```

Load SFEImage from raw image data of various formats. Eg. PNG, JPEG ..

Parameters

in	*data*	Image data to load 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_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, and example_palm_identify.cpp.

9.16.4.8 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.

9.16.4.9 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

9.16.4.10 sfeImageSave()

```
SFEError sfeImageSave (
            SFEImageView image,
            SFEImageFormat image_fromat,
            unsigned char * out_data,
            size_t * in_out_data_len )
```

Save SFEImage into a buffer encoding it to specified image_format.

Parameters

in	*image*	Image to save.
in	*image_format*	Image format to encode to.
out	*out_data*	Image data to save 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, and example_face_liveness.cpp.

9.16.4.11 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.

9.16.4.12 sfeLicenseValidate()

```
SFEError sfeLicenseValidate (
            SFELicenseSource * out_license_source )
```

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 and SFELicenseSource will be populated.

Note

If no license source information is needed, a null pointer can be supplied.

Parameters

in	*info*	Pointer to a `SFELicenseSource`.

Returns

Upon failure, the returned error will contain a descriptive message for the reason behind the failure.

9.16.4.13 sfeSolverCreate()

```
SFEError sfeSolverCreate (
            const char * solver_file,
            SFESolver * out_solver )  [inline]
```

Create new solver from solver file.

Parameters

in	*solver_file*	Solver file to create the solver from.
out	*out_solver*	New solver.

Returns

Error in case the solver fails to load.

Deprecated Use `sfeSolverCreateWithParameters` instead.

Definition at line 222 of file sfe_core.h.

```
00223                                                    {
00224   return sfeSolverCreateWithParameters(solver_file, NULL, 0, out_solver);
00225 }
```

9.16.4.14 sfeSolverCreateWithParameters()

```
SFEError sfeSolverCreateWithParameters (
            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_identify.cpp, example_face_identify_entity.cpp, example_face_liveness.cpp, example_face_track.cpp, example_iris_identify.cpp, and example_palm_identify.cpp.

9.16.4.15 sfeSolverFree()

```
void sfeSolverFree (
            SFESolver solver )
```

Free memory associated with `SFESolver`.

Parameters

in	*solver*	Solver to free.

Examples

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

example_iris_identify.cpp, and example_palm_identify.cpp.

9.16.4.16 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.

9.17 sfe_core.h

Go to the documentation of this file.

```
00001
00008 #pragma once
00009
00010 #include <stddef.h>
00011 #include <stdint.h>
00012
00013 #ifdef __cplusplus
00014 extern "C" {
00015 #endif
00016
00018 typedef enum SFEHwidMethod {
00019   SFE_HWID_METHOD_AUTO = 0,
00020   SFE_HWID_METHOD_DISK_ID = 1,
00021   SFE_HWID_METHOD_MAC = 2,
00022   SFE_HWID_METHOD_SERIAL_NUMBER = 3,
00023   SFE_HWID_METHOD_IMEI = 4,
00024   SFE_HWID_METHOD_SM_BIOS = 5,
00025   SFE_HWID_METHOD_AMAZON = 6,
00026   SFE_HWID_METHOD_APP_ID = 7,
00027   SFE_HWID_METHOD_PHY = 8,
00028   SFE_HWID_METHOD_MAC_ADDRESS = 9,
00029   SFE_HWID_METHOD_SYS = 10,
00030   SFE_HWID_METHOD_ANDROID_ID = 11,
00031 } SFEHwidMethod;
00032
00034 typedef enum SFELicenseSource {
00036   SFE_LICENSE_SOURCE_NONE = 0,
00038   SFE_LICENSE_SOURCE_FILE = 1,
00040   SFE_LICENSE_SOURCE_MEMORY = 2,
00042   SFE_LICENSE_SOURCE_TOKEN = 3,
00044   SFE_LICENSE_SOURCE_ENV = 4,
00045 } SFELicenseSource;
00046
00048 typedef enum SFEImageFormat {
00049   SFE_IMAGE_FORMAT_PNG = 0,
00050   SFE_IMAGE_FORMAT_JPEG = 1,
00051   SFE_IMAGE_FORMAT_GIF = 2,
00052   SFE_IMAGE_FORMAT_WEBP = 3,
00053   SFE_IMAGE_FORMAT_PNM = 4,
00054   SFE_IMAGE_FORMAT_TIFF = 5,
00055   SFE_IMAGE_FORMAT_TGA = 6,
00056   SFE_IMAGE_FORMAT_DDS = 7,
00057   SFE_IMAGE_FORMAT_BMP = 8,
00058   SFE_IMAGE_FORMAT_ICO = 9,
00059   SFE_IMAGE_FORMAT_HDR = 10,
00060   SFE_IMAGE_FORMAT_OPENEXR = 11,
00061   SFE_IMAGE_FORMAT_FARBFELD = 12,
00062   SFE_IMAGE_FORMAT_AVIF = 13,
00063 } SFEImageFormat;
00064
```

```
00069 typedef struct SFEImage {
00071   size_t width;
00073   size_t height;
00075   unsigned char *data;
00076 } SFEImage;
00077
00080 typedef SFEImage SFEImageView;
00081
00083 typedef struct SFEBbox {
00086   float x;
00089   float y;
00092   float width;
00095   float height;
00096 } SFEBbox;
00097
00099 typedef struct SFEOrientedBbox {
00102   float center_x;
00105   float center_y;
00108   float width;
00111   float height;
00114   float angle;
00115 } SFEOrientedBbox;
00116
00121 typedef void *SFEError;
00122
00126 typedef void *SFESolver;
00127
00129 typedef struct SFESolverParameter {
00131   const char *name;
00133   const char *value;
00134 } SFESolverParameter;
00135
00138 typedef struct SFEEntity {
00139   uint8_t uuid[16];
00140 } SFEEntity;
00141
00143 typedef struct SFETemplateIdentificationCandidate {
00145   float score;
00147   size_t index;
00148 } SFETemplateIdentificationCandidate;
00149
00151 typedef struct SFEEntityIdentificationCandidate {
00153   float score;
00155   SFEEntity entity;
00156 } SFEEntityIdentificationCandidate;
00157
00161 const char *const sfeVersion();
00162
00165 void sfeErrorFree(SFEError error);
00166
00172 const char *sfeErrorMessage(SFEError error);
00173
00185 SFEError sfeHwidGet(char *out_hwid, size_t *in_out_hwid_len,
00186                     SFEHwidMethod method);
00187
00197 SFEError sfeLicenseValidate(SFELicenseSource *out_license_source);
00198
00205 void sfeLicenseSet(const unsigned char *data, const size_t data_len);
00206
00213 SFEError sfeSolverCreateWithParameters(
00214     const char *solver_file, const SFESolverParameter *solver_parameters,
00215     size_t solver_parameters_count, SFESolver *out_solver);
00216
00222 inline SFEError sfeSolverCreate(const char *solver_file,
00223                                 SFESolver *out_solver) {
00224   return sfeSolverCreateWithParameters(solver_file, NULL, 0, out_solver);
00225 }
00226
00229 void sfeSolverFree(SFESolver solver);
00230
00239 SFEError sfeImageLoad(const unsigned char *data, size_t data_len,
00240                       SFEImage *out_image);
00241
00250 SFEError sfeImageSave(SFEImageView image, SFEImageFormat image_fromat,
00251                       unsigned char *out_data, size_t *in_out_data_len);
00252
00260 SFEError sfeImageInfo(const unsigned char *data, size_t data_len,
00261                       size_t *out_width, size_t *out_height,
00262                       SFEImageFormat *out_image_format);
00263
00266 void sfeImageFree(SFEImage image);
00267
00271 SFEError sfeImageColorTranspose(SFEImageView image);
00272
00279 SFEError sfeImageResize(SFEImageView image, size_t width, size_t height,
00280                         SFEImage *out_image);
00281
```

```
00289 SFEError sfeImageResizeWithAspectRatio(SFEImageView image, size_t width,
00290                                        size_t height, SFEImage *out_image);
00291
00292 #ifdef __cplusplus
00293 } // extern "C"
00294 #endif
```

9.18 D:/glab/da0a89/1/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 SFEFaceDetect

  Face detect struct.
- struct SFEFaceLandmarks

  Face landmarks struct.
- struct SFEFaceTemplate

  Face template struct.
- struct SFEFaceTemplateVersion

  Face template version struct.
- struct SFEFaceArea

  Face area struct.
- struct SFEFaceHeadPose

  Face head pose struct containing angle rotations of head.
- struct SFEFaceQualityAttributes

  Face quality attributes struct.
- struct SFEFaceCrop

  Face crop struct.
- struct SFEFaceTracked

  Tracked entity struct.

Macros

- #define SFE_FACE_LANDMARK_COUNT 23
- #define SFE_FACE_TEMPLATE_SIZE 522

Typedefs

- typedef SFEEntity SFEFaceEntity
- typedef SFETemplateIdentificationCandidate SFEIdentificationResult

  Candidate for identification by template.
- typedef SFETemplateIdentificationCandidate SFEFaceTemplateIdentificationCandidate
- typedef SFEEntityIdentificationCandidate SFEFaceEntityIdentificationCandidate

  Candidate for identification by entity.
- typedef struct SFEFaceDetect SFEFaceDetect

  Face detect struct.
- typedef enum SFEFaceLandmarkType SFEFaceLandmarkType

  Face landmark types.
- typedef struct SFEFaceLandmarks SFEFaceLandmarks

  Face landmarks struct.
- typedef struct SFEFaceTemplate SFEFaceTemplate

  Face template struct.
- typedef struct SFEFaceTemplateVersion SFEFaceTemplateVersion

  Face template version 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 enum SFEFaceDetectAccuracyType SFEFaceDetectAccuracyType

  Supported 2 detection accuracy types.
- typedef struct SFEFaceCrop SFEFaceCrop

  Face crop struct.
- typedef void * SFEFaceTracker

  Tracker is a ByteTrack implementation for face tracking across.
- typedef struct SFEFaceTracked SFEFaceTracked

  Tracked entity 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 SFEFaceDetectAccuracyType { SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE = 0 , SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED = 1 }

  Supported 2 detection accuracy types.

- enum SFEFaceTrackedState { SFE_FACE_TRACKED_STATE_NEW = 0 , SFE_FACE_TRACKED_STATE_TRACKED = 1 , SFE_FACE_TRACKED_STATE_LOST = 2 , SFE_FACE_TRACKED_STATE_REMOVED = 3 }

  Tracked state.

Functions

- SFEError sfeFaceDetect (SFESolver solver, SFEImageView image, float threshold, SFEFaceDetect *out_faces, size_t *in_out_face_count)

  Detect faces in the source image.

- SFEError sfeFaceLandmarks (SFESolver solver, SFEImageView image, SFEBbox bbox, SFEFaceLandmarks out_face_landmarks[SFE_FACE_LANDMARK_COUNT])

  Detect 23 landmarks of the face detected in the area of source image marked with bounding box.

- 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, SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT], float raw_detection_confidence, 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 sfeFaceTemplateIdentify (SFEFaceTemplate *probe_face_template, SFEFaceTemplate *templates_gallery, size_t gallery_size, float matching_score_threshold, SFEFaceTemplateIdentificationCandidate *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], float *out_liveness_score)

  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 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, SFEFaceDetectAccuracyType detection_mode, size_t min_face_size, size_t max_face_size, size_t *out_img_width, size_t *out_img_height)

  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], uint32_t 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.
- SFEError sfeFaceTrackerCreate (float track_threshold, float high_threshold, float match_threshold, uint64_t max_time_lost, SFEFaceTracker *out_tracker)

  Create a new tracker.
- SFEError sfeFaceTrackerFree (SFEFaceTracker tracker)

  Free the tracker.
- SFEError sfeFaceTrackerUpdate (SFEFaceTracker tracker, SFEFaceDetect *detections, size_t detections↩ _count, SFEFaceTracked *out_tracked, size_t *out_tracked_count)

  Update the tracker.
- SFEError sfeFaceTrackerLost (SFEFaceTracker tracker, SFEEntity *out_entities, size_t *in_out_entities_↩ count)

  Get lost entities after update.
- SFEError sfeFaceTrackerRemoved (SFEFaceTracker tracker, SFEEntity *out_entities, size_t *in_out_↩ entities_count)

  Get removed entities after update.

9.18.1 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.

9.18.2 Macro Definition Documentation

9.18.2.1 SFE_FACE_LANDMARK_COUNT

```
#define SFE_FACE_LANDMARK_COUNT 23
```

Examples

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

Definition at line 57 of file sfe_face.h.

9.18.2.2 SFE_FACE_TEMPLATE_SIZE

```
#define SFE_FACE_TEMPLATE_SIZE 522
```

Definition at line 135 of file sfe_face.h.

9.18.3 Typedef Documentation

9.18.3.1 SFEFaceArea

```
typedef struct SFEFaceArea SFEFaceArea
```

Face area struct.

9.18.3.2 SFEFaceCrop

```
typedef struct SFEFaceCrop SFEFaceCrop
```

Face crop struct.

9.18.3.3 SFEFaceDetect

```
typedef struct SFEFaceDetect SFEFaceDetect
```

Face detect struct.

9.18.3.4 SFEFaceDetectAccuracyType

```
typedef enum SFEFaceDetectAccuracyType SFEFaceDetectAccuracyType
```

Supported 2 detection accuracy types.

9.18.3.5 SFEFaceEntity

```
typedef SFEEntity SFEFaceEntity
```

Deprecated Use SFEEntity instead, this type will be removed in future

Definition at line 19 of file sfe_face.h.

9.18.3.6 SFEFaceEntityIdentificationCandidate

```
typedef SFEEntityIdentificationCandidate SFEFaceEntityIdentificationCandidate
```

Candidate for identification by entity.

Deprecated Replace with SFEEntityIdentificationCandidate

Definition at line 31 of file sfe_face.h.

9.18.3.7 SFEFaceHeadPose

```
typedef struct SFEFaceHeadPose SFEFaceHeadPose
```

Face head pose struct containing angle rotations of head.

9.18.3.8 SFEFaceLandmarks

```
typedef struct SFEFaceLandmarks SFEFaceLandmarks
```

Face landmarks struct.

9.18.3.9 SFEFaceLandmarkType

```
typedef enum SFEFaceLandmarkType SFEFaceLandmarkType
```

Face landmark types.

9.18.3.10 SFEFaceQualityAttributes

```
typedef struct SFEFaceQualityAttributes SFEFaceQualityAttributes
```

Face quality attributes struct.

9.18.3.11 SFEFaceTemplate

```
typedef struct SFEFaceTemplate SFEFaceTemplate
```

Face template struct.

9.18.3.12 SFEFaceTemplateIdentificationCandidate

```
typedef SFETemplateIdentificationCandidate SFEFaceTemplateIdentificationCandidate
```

Deprecated Replace with SFETemplateIdentificationCandidate

Definition at line 27 of file sfe_face.h.

9.18.3.13 SFEFaceTemplateVersion

```
typedef struct SFEFaceTemplateVersion SFEFaceTemplateVersion
```

Face template version struct.

9.18.3.14 SFEFaceTracked

```
typedef struct SFEFaceTracked SFEFaceTracked
```

Tracked entity struct.

9.18.3.15 SFEFaceTracker

```
typedef void* SFEFaceTracker
```

Tracker is a ByteTrack implementation for face tracking across.

Warning

Use `sfeFaceTrackerFree` to release memory associated with the tracker.

Definition at line 386 of file sfe_face.h.

9.18.3.16 SFEIdentificationResult

```
typedef SFETemplateIdentificationCandidate SFEIdentificationResult
```

Candidate for identification by template.

Deprecated Replace with SFETemplateIdentificationCandidate

Definition at line 23 of file sfe_face.h.

9.18.4 Enumeration Type Documentation

9.18.4.1 SFEFaceDetectAccuracyType

```
enum SFEFaceDetectAccuracyType
```

Supported 2 detection accuracy types.

Enumerator

SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE	
SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED	

Definition at line 329 of file sfe_face.h.

```
00329                                       {
00330   SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE = 0,
00331   SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED = 1,
00332 } SFEFaceDetectAccuracyType;
```

9.18.4.2 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 60 of file sfe_face.h.

```
00060                                   {
00061   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER = 0,
00062   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER = 1,
00063   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER = 2,
00064   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER = 3,
00065   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER = 4,
00066   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER = 5,
00067   SFE_FACE_LANDMARK_TYPE_NOSE_ROOT = 6,
00068   SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM = 7,
00069   SFE_FACE_LANDMARK_TYPE_NOSE_TIP = 8,
00070   SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM = 9,
00071   SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM = 10,
```

```
00072   SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER = 11,
00073   SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER = 12,
00074   SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER = 13,
00075   SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE = 14,
00076   SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE = 15,
00077   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END = 16,
00078   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END = 17,
00079   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END = 18,
00080   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END = 19,
00081   SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE = 20,
00082   SFE_FACE_LANDMARK_TYPE_CHIN_TIP = 21,
00083   SFE_FACE_LANDMARK_TYPE_LEFT_EDGE = 22,
00084 } SFEFaceLandmarkType;
```

9.18.4.3 SFEFaceTrackedState

```
enum SFEFaceTrackedState
```

Tracked state.

Enumerator

SFE_FACE_TRACKED_STATE_NEW	
SFE_FACE_TRACKED_STATE_TRACKED	
SFE_FACE_TRACKED_STATE_LOST	
SFE_FACE_TRACKED_STATE_REMOVED	

Definition at line 389 of file sfe_face.h.

```
00389                           {
00390   SFE_FACE_TRACKED_STATE_NEW = 0,
00391   SFE_FACE_TRACKED_STATE_TRACKED = 1,
00392   SFE_FACE_TRACKED_STATE_LOST = 2,
00393   SFE_FACE_TRACKED_STATE_REMOVED = 3,
00394 };
```

9.18.5 Function Documentation

9.18.5.1 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.

9.18.5.2 sfeFaceCrop()

```
SFEError sfeFaceCrop (
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            uint32_t 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.

9.18.5.3 sfeFaceDetect()

```
SFEError sfeFaceDetect (
            SFESolver solver,
            SFEImageView image,
            float threshold,
            SFEFaceDetect * out_faces,
            size_t * in_out_face_count )
```

Detect faces in the source image.

Parameters

in	*solver*	face detection solver
in	*image*	source image
in	*threshold*	detection confidence threshold - range <0,1>
out	*out_faces*	pointer to an array of detected faces with detection confidence above the threshold, detected faces are sorted by detection confidence
in,out	*in_out_face_count*	IN: maximum number of faces to detect OUT: real number of detected faces

Returns

Error in case of failed detection.

Examples

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

9.18.5.4 sfeFaceDetectInputSize()

```
SFEError sfeFaceDetectInputSize (
            SFEImageView image,
            SFEFaceDetectAccuracyType detection_mode,
            size_t min_face_size,
            size_t max_face_size,
            size_t * out_img_width,
            size_t * out_img_height )
```

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_img_width*	Recommended image width
out	*out_img_height*	Recommended image 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.

9.18.5.5 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.

9.18.5.6 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.

9.18.5.7 sfeFaceLandmarks()

```
SFEError sfeFaceLandmarks (
            SFESolver solver,
```

```
            SFEImageView image,
            SFEBbox bbox,
            SFEFaceLandmarks out_face_landmarks[SFE_FACE_LANDMARK_COUNT] )
```

Detect 23 landmarks of the face detected in the area of source image marked with bounding box.

Parameters

in	*solver*	face landmarks solver
in	*image*	source image
in	*bbox*	bounding box of 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_identify.cpp, example_face_identify_entity.cpp, and example_face_liveness.cpp.

9.18.5.8 sfeFaceLivenessPassive()

```
SFEError sfeFaceLivenessPassive (
            SFESolver solver,
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            float * out_liveness_score )
```

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_score*	OUT: normalized score of passive liveness

Returns

Error in case of failed passive liveness score calculation

Examples

example_face_liveness.cpp.

9.18.5.9 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.

9.18.5.10 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.

9.18.5.11 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.

9.18.5.12 sfeFaceTemplateExtract()

```
SFEError sfeFaceTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
            float raw_detection_confidence,
            SFEFaceTemplate * out_face_template )
```

Extract template from source image and given face landmarks.

Parameters

in	*solver*	template extraction solver
in	*image*	source image
in	*face_landmarks*	face landmarks detected in source image
in	*raw_detection_confidence*	raw value of face detection confidence of detected face (used in template quality calculation)
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.

9.18.5.13 sfeFaceTemplateIdentify()

```
SFEError sfeFaceTemplateIdentify (
            SFEFaceTemplate * probe_face_template,
            SFEFaceTemplate * templates_gallery,
            size_t gallery_size,
            float matching_score_threshold,
```

```
            SFEFaceTemplateIdentificationCandidate * 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
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.

9.18.5.14 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.

9.18.5.15 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

9.18.5.16 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.

9.18.5.17 sfeFaceTrackerCreate()

```
SFEError sfeFaceTrackerCreate (
            float track_threshold,
            float high_threshold,
            float match_threshold,
            uint64_t max_time_lost,
            SFEFaceTracker * out_tracker )
```

Create a new tracker.

Parameters

in	*track_threshold*	Tracking threshold
in	*high_threshold*	High threshold
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.

9.18.5.18 sfeFaceTrackerFree()

```
SFEError sfeFaceTrackerFree (
            SFEFaceTracker tracker )
```

Free the tracker.

Parameters

in	*tracker*	Tracker to free

Examples

example_face_track.cpp.

9.18.5.19 sfeFaceTrackerLost()

```
SFEError sfeFaceTrackerLost (
            SFEFaceTracker 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*	OUT: number of lost entities

Returns

Error in case of failed lost entities retrieval

Examples

example_face_track.cpp.

9.18.5.20 sfeFaceTrackerRemoved()

```
SFEError sfeFaceTrackerRemoved (
            SFEFaceTracker 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*	OUT: number of removed entities

Returns

Error in case of failed removed entities retrieval

Examples

example_face_track.cpp.

9.18.5.21 sfeFaceTrackerUpdate()

```
SFEError sfeFaceTrackerUpdate (
            SFEFaceTracker tracker,
            SFEFaceDetect * detections,
            size_t detections_count,
            SFEFaceTracked * out_tracked,
            size_t * out_tracked_count )
```

Update the tracker.

Parameters

in	*tracker*	Tracker to update
in	*detections*	Detections to update the tracker with
in	*detections_count*	Number of detections
out	*out_tracked*	OUT: pointer to the array of tracked entities
out	*out_tracked_count*	OUT: number of tracked entities

Returns

Error in case of failed update

Examples

example_face_track.cpp.

9.19 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 DETECTION
00017
00019 typedef SFEEntity SFEFaceEntity;
00020
00023 typedef SFETemplateIdentificationCandidate SFEIdentificationResult;
00024
00025 // @brief Candidate for identification by template
00027 typedef SFETemplateIdentificationCandidate SFEFaceTemplateIdentificationCandidate;
00028
00031 typedef SFEEntityIdentificationCandidate SFEFaceEntityIdentificationCandidate;
00032
00034 typedef struct SFEFaceDetect {
00036   SFEBbox bbox;
00038   float confidence;
00040   float raw_confidence;
00041 } SFEFaceDetect;
00042
00053 SFEError sfeFaceDetect(SFESolver solver, SFEImageView image, float threshold,
00054                        SFEFaceDetect *out_faces, size_t *in_out_face_count);
00055
00056 // FACE LANDMARKS
00057 #define SFE_FACE_LANDMARK_COUNT 23
00058
00060 typedef enum SFEFaceLandmarkType {
00061   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER = 0,
00062   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER = 1,
00063   SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER = 2,
00064   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER = 3,
00065   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER = 4,
00066   SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER = 5,
00067   SFE_FACE_LANDMARK_TYPE_NOSE_ROOT = 6,
00068   SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM = 7,
00069   SFE_FACE_LANDMARK_TYPE_NOSE_TIP = 8,
00070   SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM = 9,
00071   SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM = 10,
00072   SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER = 11,
00073   SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER = 12,
00074   SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER = 13,
00075   SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE = 14,
00076   SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE = 15,
00077   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END = 16,
00078   SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END = 17,
00079   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END = 18,
00080   SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END = 19,
00081   SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE = 20,
00082   SFE_FACE_LANDMARK_TYPE_CHIN_TIP = 21,
00083   SFE_FACE_LANDMARK_TYPE_LEFT_EDGE = 22,
00084 } SFEFaceLandmarkType;
00085
00087 typedef struct SFEFaceLandmarks {
00089   SFEFaceLandmarkType landmark_type;
00091   float confidence;
00093   float x;
00095   float y;
00096 } SFEFaceLandmarks;
00097
00107 SFEError
00108 sfeFaceLandmarks(SFESolver solver, SFEImageView image, SFEBbox bbox,
00109                  SFEFaceLandmarks out_face_landmarks[SFE_FACE_LANDMARK_COUNT]);
00110
00117 SFEError sfeFaceMaskConfidence(
00118     const SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00119     float *out_mask_confidence);
00120
00130 SFEError sfeFaceSize(SFEImageView image,
00131                      SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00132                      float *face_size);
00133
00134 // FACE TEMPLATE
00135 #define SFE_FACE_TEMPLATE_SIZE 522
00136
00138 typedef struct SFEFaceTemplate {
00139   uint8_t data[SFE_FACE_TEMPLATE_SIZE];
```

```
00140 } SFEFaceTemplate;
00141
00143 typedef struct SFEFaceTemplateVersion {
00145   uint8_t version_major;
00147   uint8_t version_minor;
00148 } SFEFaceTemplateVersion;
00149
00158 SFEError
00159 sfeFaceTemplateExtract(SFESolver solver, SFEImageView image,
00160                        SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00161                        float raw_detection_confidence,
00162                        SFEFaceTemplate *out_face_template);
00163
00169 SFEError sfeFaceTemplateMatch(SFEFaceTemplate *template1,
00170                               SFEFaceTemplate *template2,
00171                               float *out_matching_score);
00172
00177 SFEError sfeFaceTemplateQuality(SFEFaceTemplate *face_template,
00178                                 float *out_face_template_quality);
00179
00184 SFEError
00185 sfeFaceTemplateVersion(SFEFaceTemplate *face_template,
00186                        SFEFaceTemplateVersion *out_face_template_version);
00187
00201 SFEError
00202 sfeFaceTemplateIdentify(SFEFaceTemplate *probe_face_template,
00203                         SFEFaceTemplate *templates_gallery, size_t gallery_size,
00204                         float matching_score_threshold,
00205                         SFEFaceTemplateIdentificationCandidate *out_candidates,
00206                         size_t *in_out_candidates_count, size_t thread_count);
00207
00225 SFEError sfeFaceEntityIdentify(SFEFaceTemplate *probe_face_template,
00226                                SFEFaceTemplate *templates_gallery,
00227                                SFEEntity *entities_gallery, size_t gallery_size,
00228                                float matching_score_threshold,
00229                                SFEEntityIdentificationCandidate *out_candidates,
00230                                size_t *in_out_candidates_count,
00231                                size_t thread_count);
00232
00239 SFEError
00240 sfeFaceLivenessPassive(SFESolver solver, SFEImageView image,
00241                        SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00242                        float *out_liveness_score);
00243
00245 typedef struct SFEFaceArea {
00247   float size;
00249   float area;
00252   float area_in_image;
00253 } SFEFaceArea;
00254
00262 SFEError sfeFaceArea(SFEImageView image,
00263                      SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00264                      SFEFaceArea *out_area);
00265
00267 typedef struct SFEFaceHeadPose {
00270   float pitch;
00273   float yaw;
00276   float roll;
00277 } SFEFaceHeadPose;
00278
00285 SFEError
00286 sfeFaceHeadPose(SFEImageView image,
00287                 SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00288                 SFEFaceHeadPose *out_head_pose);
00289
00291 typedef struct SFEFaceQualityAttributes {
00296   float sharpness;
00301   float brightness;
00307   float contrast;
00313   float unique_intensity_levels;
00314 } SFEFaceQualityAttributes;
00315
00323 SFEError sfeFaceQualityAttributes(
00324     SFEImageView image,
00325     SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00326     SFEFaceQualityAttributes *out_quality_attributes);
00327
00329 typedef enum SFEFaceDetectAccuracyType {
00330   SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE = 0,
00331   SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED = 1,
00332 } SFEFaceDetectAccuracyType;
00333
00349 SFEError sfeFaceDetectInputSize(SFEImageView image,
00350                                 SFEFaceDetectAccuracyType detection_mode,
00351                                 size_t min_face_size, size_t max_face_size,
00352                                 size_t *out_img_width, size_t *out_img_height);
00353
```

```
00355 typedef struct SFEFaceCrop {
00358   uint32_t face_size_extension;
00360   SFEBbox crop_box;
00362   SFEImage crop_image;
00363 } SFEFaceCrop;
00364
00378 SFEError sfeFaceCrop(SFEImageView image,
00379                      SFEFaceLandmarks face_landmarks[SFE_FACE_LANDMARK_COUNT],
00380                      uint32_t face_size_extension, float max_face_size,
00381                      SFEFaceCrop *out_crop);
00382
00386 typedef void *SFEFaceTracker;
00387
00389 enum SFEFaceTrackedState {
00390   SFE_FACE_TRACKED_STATE_NEW = 0,
00391   SFE_FACE_TRACKED_STATE_TRACKED = 1,
00392   SFE_FACE_TRACKED_STATE_LOST = 2,
00393   SFE_FACE_TRACKED_STATE_REMOVED = 3,
00394 };
00395
00397 typedef struct SFEFaceTracked {
00399   SFEFaceDetect tracked;
00401   uint64_t id;
00403   SFEEntity uuid;
00405   SFEFaceTrackedState state;
00406 } SFEFaceTracked;
00407
00414 SFEError sfeFaceTrackerCreate(float track_threshold, float high_threshold,
00415                               float match_threshold, uint64_t max_time_lost,
00416                               SFEFaceTracker *out_tracker);
00417
00420 SFEError sfeFaceTrackerFree(SFEFaceTracker tracker);
00421
00429 SFEError sfeFaceTrackerUpdate(SFEFaceTracker tracker, SFEFaceDetect *detections,
00430                               size_t detections_count,
00431                               SFEFaceTracked *out_tracked,
00432                               size_t *out_tracked_count);
00433
00439 SFEError sfeFaceTrackerLost(SFEFaceTracker tracker, SFEEntity *out_entities,
00440                             size_t *in_out_entities_count);
00441
00447 SFEError sfeFaceTrackerRemoved(SFEFaceTracker tracker, SFEEntity *out_entities,
00448                                size_t *in_out_entities_count);
00449
00450 #ifdef __cplusplus
00451 } // extern "C"
00452 #endif
```

9.20 D:/glab/da0a89/1/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 SFEIrisKeypoint

  Iris keypoint struct.
- struct SFEIrisDetect

  Iris detection struct.
- struct SFEIrisTemplate

  Iris template struct.
- struct SFEIrisTemplateVersion

  Iris template version struct.
- struct SFEIrisTracked

  Tracked entity struct.

Macros

- #define SFE_IRIS_KEYPOINT_COUNT 8

  Iris keypoints count.
- #define SFE_IRIS_TEMPLATE_SIZE 522

  Iris template size in bytes.

Typedefs

- typedef SFEEntity SFEIrisEntity
- typedef SFETemplateIdentificationCandidate SFEIrisTemplateIdentificationCandidate
- typedef SFEEntityIdentificationCandidate SFEIrisEntityIdentificationCandidate
- typedef enum SFEIrisKeypointType SFEIrisKeypointType

  Iris keypoint types.
- typedef struct SFEIrisKeypoint SFEIrisKeypoint

  Iris keypoint struct.
- typedef struct SFEIrisDetect SFEIrisDetect

  Iris detection struct.
- typedef struct SFEIrisTemplate SFEIrisTemplate

  Iris template struct.
- typedef struct SFEIrisTemplateVersion SFEIrisTemplateVersion

  Iris template version struct.
- typedef void * SFEIrisTracker

  Tracker is a ByteTrack implementation for iris tracking across multiple frames.
- typedef struct SFEIrisTracked SFEIrisTracked

  Tracked entity struct.

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 SFEIrisTrackedState { SFE_IRIS_TRACKED_STATE_NEW = 0 , SFE_IRIS_TRACKED_STATE_TRACKED = 1 , SFE_IRIS_TRACKED_STATE_LOST = 2 , SFE_IRIS_TRACKED_STATE_REMOVED = 3 }

  Tracked state.

Functions

- SFEError sfeIrisDetect (SFESolver solver, SFEImageView image, SFEIrisDetect *out_iris_detection)

  Detect iris in the source image.
- SFEError sfeIrisTemplateExtract (SFESolver solver, SFEImageView image, const SFEIrisDetect *in_↩ 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.
- SFEError sfeIrisTrackerCreate (float track_threshold, float high_threshold, float match_threshold, uint64_t max_time_lost, SFEIrisTracker *out_tracker)

  Create a new tracker.
- SFEError sfeIrisTrackerFree (SFEIrisTracker tracker)

  Free the tracker.
- SFEError sfeIrisTrackerUpdate (SFEIrisTracker tracker, SFEIrisDetect *detections, size_t detections_count, SFEIrisTracked *out_tracked, size_t *out_tracked_count)

  Update the tracker.
- SFEError sfeIrisTrackerLost (SFEIrisTracker tracker, SFEEntity *out_entities, size_t *in_out_entities_count)

  Get lost entities after update.
- SFEError sfeIrisTrackerRemoved (SFEIrisTracker tracker, SFEEntity *out_entities, size_t *in_out_entities_count)

  Get removed entities after update.

9.20.1 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.

9.20.2 Macro Definition Documentation

9.20.2.1 SFE_IRIS_KEYPOINT_COUNT

```
#define SFE_IRIS_KEYPOINT_COUNT 8
```

Iris keypoints count.

Definition at line 52 of file sfe_iris.h.

9.20.2.2 SFE_IRIS_TEMPLATE_SIZE

```
#define SFE_IRIS_TEMPLATE_SIZE 522
```

Iris template size in bytes.

Definition at line 78 of file sfe_iris.h.

9.20.3 Typedef Documentation

9.20.3.1 SFEIrisDetect

```
typedef struct SFEIrisDetect SFEIrisDetect
```

Iris detection struct.

9.20.3.2 SFEIrisEntity

```
typedef SFEEntity SFEIrisEntity
```

Deprecated Use SFEEntity instead, this type will be removed in future

Definition at line 19 of file sfe_iris.h.

9.20.3.3 SFEIrisEntityIdentificationCandidate

```
typedef SFEEntityIdentificationCandidate SFEIrisEntityIdentificationCandidate
```

Deprecated Replace with SFEEntityIdentificationCandidate

Definition at line 25 of file sfe_iris.h.

9.20.3.4 SFEIrisKeypoint

```
typedef struct SFEIrisKeypoint SFEIrisKeypoint
```

Iris keypoint struct.

9.20.3.5 SFEIrisKeypointType

```
typedef enum SFEIrisKeypointType SFEIrisKeypointType
```

Iris keypoint types.

9.20.3.6 SFEIrisTemplate

```
typedef struct SFEIrisTemplate SFEIrisTemplate
```

Iris template struct.

9.20.3.7 SFEIrisTemplateIdentificationCandidate

```
typedef SFETemplateIdentificationCandidate SFEIrisTemplateIdentificationCandidate
```

Deprecated Replace with SFETemplateIdentificationCandidate

Definition at line 22 of file sfe_iris.h.

9.20.3.8 SFEIrisTemplateVersion

```
typedef struct SFEIrisTemplateVersion SFEIrisTemplateVersion
```

Iris template version struct.

9.20.3.9 SFEIrisTracked

```
typedef struct SFEIrisTracked SFEIrisTracked
```

Tracked entity struct.

9.20.3.10 SFEIrisTracker

```
typedef void* SFEIrisTracker
```

Tracker is a ByteTrack implementation for iris tracking across multiple frames.

Warning

Use `sfeIrisTrackerFree` to release memory associated with the tracker.

Definition at line 175 of file sfe_iris.h.

9.20.4 Enumeration Type Documentation

9.20.4.1 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 28 of file sfe_iris.h.

```
00028                                {
00029   SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT = 0,
00030   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT = 1,
00031   SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT = 2,
00032   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT = 3,
00033   SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT = 4,
00034   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT = 5,
00035   SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT = 6,
00036   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT = 7,
00037 } SFEIrisKeypointType;
```

9.20.4.2 SFEIrisTrackedState

```
enum SFEIrisTrackedState
```

Tracked state.

Enumerator

SFE_IRIS_TRACKED_STATE_NEW	
SFE_IRIS_TRACKED_STATE_TRACKED	
SFE_IRIS_TRACKED_STATE_LOST	
SFE_IRIS_TRACKED_STATE_REMOVED	

Definition at line 178 of file sfe_iris.h.

```
00178                              {
00179   SFE_IRIS_TRACKED_STATE_NEW = 0,
00180   SFE_IRIS_TRACKED_STATE_TRACKED = 1,
00181   SFE_IRIS_TRACKED_STATE_LOST = 2,
00182   SFE_IRIS_TRACKED_STATE_REMOVED = 3,
00183 };
```

9.20.5 Function Documentation

9.20.5.1 sfeIrisDetect()

```
SFEError sfeIrisDetect (
            SFESolver solver,
            SFEImageView image,
            SFEIrisDetect * out_iris_detection )
```

Detect iris in the source image.

Parameters

in	*solver*	iris detection solver
in	*image*	source image
out	*out_iris_detection*	pointer to the iris detection struct where a detected iris will be saved

Returns

Error in case of failed detection.

Examples

example_iris_identify.cpp.

9.20.5.2 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
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.

9.20.5.3 sfeIrisTemplateExtract()

```
SFEError sfeIrisTemplateExtract (
```

```
            SFESolver solver,
            SFEImageView image,
            const SFEIrisDetect * in_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*	iris detection detected in the source image
out	*out_iris_template*	extracted iris template

Returns

Error in case of failed extraction.

Examples

example_iris_identify.cpp.

9.20.5.4 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
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.

9.20.5.5 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.

9.20.5.6 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.

9.20.5.7 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.

9.20.5.8 sfeIrisTrackerCreate()

```
SFEError sfeIrisTrackerCreate (
            float track_threshold,
            float high_threshold,
            float match_threshold,
            uint64_t max_time_lost,
            SFEIrisTracker * out_tracker )
```

Create a new tracker.

Parameters

in	*track_threshold*	Tracking threshold
in	*high_threshold*	High threshold
in	*match_threshold*	Match threshold
in	*max_time_lost*	Maximum time lost
out	*out_tracker*	OUT: pointer to the created tracker

9.20.5.9 sfeIrisTrackerFree()

```
SFEError sfeIrisTrackerFree (
            SFEIrisTracker tracker )
```

Free the tracker.

Parameters

in	*tracker*	Tracker to free

9.20.5.10 sfeIrisTrackerLost()

```
SFEError sfeIrisTrackerLost (
            SFEIrisTracker 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*	OUT: number of lost entities

Returns

Error in case of failed lost entities retrieval

9.20.5.11 sfeIrisTrackerRemoved()

```
SFEError sfeIrisTrackerRemoved (
            SFEIrisTracker 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*	OUT: number of removed entities

Returns

Error in case of failed removed entities retrieval

9.20.5.12 sfeIrisTrackerUpdate()

```
SFEError sfeIrisTrackerUpdate (
            SFEIrisTracker tracker,
            SFEIrisDetect * detections,
            size_t detections_count,
            SFEIrisTracked * out_tracked,
            size_t * out_tracked_count )
```

Update the tracker.

Parameters

in	*tracker*	Tracker to update
in	*detections*	Detections to update the tracker with
in	*detections_count*	Number of detections

Returns

Error in case of failed update

9.21 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 DETECTION
00017
00019 typedef SFEEntity SFEIrisEntity;
00020
00022 typedef SFETemplateIdentificationCandidate SFEIrisTemplateIdentificationCandidate;
00023
00025 typedef SFEEntityIdentificationCandidate SFEIrisEntityIdentificationCandidate;
00026
00028 typedef enum SFEIrisKeypointType {
00029   SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT = 0,
00030   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT = 1,
00031   SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT = 2,
00032   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT = 3,
00033   SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT = 4,
00034   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT = 5,
00035   SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT = 6,
00036   SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT = 7,
00037 } SFEIrisKeypointType;
00038
00040 typedef struct SFEIrisKeypoint {
00042   SFEIrisKeypointType keypoint_type;
00044   float confidence;
00046   float x;
00048   float y;
00049 } SFEIrisKeypoint;
00050
00052 #define SFE_IRIS_KEYPOINT_COUNT 8
00053
00055 typedef struct SFEIrisDetect {
00057   SFEBbox iris_bounding_box;
00059   SFEBbox pupil_bounding_box;
00061   float confidence;
00063   SFEIrisKeypoint keypoints[SFE_IRIS_KEYPOINT_COUNT];
00064 } SFEIrisDetect;
00065
00072 SFEError sfeIrisDetect(SFESolver solver, SFEImageView image,
00073                        SFEIrisDetect *out_iris_detection);
00074
00075 // IRIS TEMPLATE EXTRACTION AND MATCHING
00076
00078 #define SFE_IRIS_TEMPLATE_SIZE 522
00079
00081 typedef struct SFEIrisTemplate {
00082   uint8_t data[SFE_IRIS_TEMPLATE_SIZE];
00083 } SFEIrisTemplate;
00084
00086 typedef struct SFEIrisTemplateVersion {
00088   uint8_t version_major;
00090   uint8_t version_minor;
00091 } SFEIrisTemplateVersion;
00092
00099 SFEError sfeIrisTemplateExtract(SFESolver solver, SFEImageView image,
00100                                 const SFEIrisDetect *in_detection,
```

```
00101                                SFEIrisTemplate *out_iris_template);
00102
00108 SFEError sfeIrisTemplateMatch(const SFEIrisTemplate *template1,
00109                               const SFEIrisTemplate *template2,
00110                               float *out_matching_score);
00111
00116 SFEError
00117 sfeIrisTemplateVersion(const SFEIrisTemplate *iris_template,
00118                        SFEIrisTemplateVersion *out_iris_template_version);
00119
00124 SFEError sfeIrisTemplateQuality(const SFEIrisTemplate *iris_template,
00125                                 float *out_iris_template_quality);
00126
00139 SFEError
00140 sfeIrisTemplateIdentify(const SFEIrisTemplate *probe_iris_template,
00141                         const SFEIrisTemplate *templates_gallery,
00142                         size_t gallery_size, float matching_score_threshold,
00143                         SFETemplateIdentificationCandidate *out_candidates,
00144                         size_t *in_out_candidates_count, size_t thread_count);
00145
00162 SFEError sfeIrisEntityIdentify(const SFEIrisTemplate *probe_iris_template,
00163                                const SFEIrisTemplate *templates_gallery,
00164                                const SFEEntity *entities_gallery,
00165                                size_t gallery_size,
00166                                float matching_score_threshold,
00167                                SFEEntityIdentificationCandidate *out_candidates,
00168                                size_t *in_out_candidates_count,
00169                                size_t thread_count);
00170
00175 typedef void *SFEIrisTracker;
00176
00178 enum SFEIrisTrackedState {
00179   SFE_IRIS_TRACKED_STATE_NEW = 0,
00180   SFE_IRIS_TRACKED_STATE_TRACKED = 1,
00181   SFE_IRIS_TRACKED_STATE_LOST = 2,
00182   SFE_IRIS_TRACKED_STATE_REMOVED = 3,
00183 };
00184
00186 typedef struct SFEIrisTracked {
00188   SFEIrisDetect tracked;
00190   uint64_t id;
00192   SFEEntity uuid;
00194   SFEIrisTrackedState state;
00195 } SFEIrisTracked;
00196
00203 SFEError sfeIrisTrackerCreate(float track_threshold, float high_threshold,
00204                               float match_threshold, uint64_t max_time_lost,
00205                               SFEIrisTracker *out_tracker);
00206
00209 SFEError sfeIrisTrackerFree(SFEIrisTracker tracker);
00210
00216 SFEError sfeIrisTrackerUpdate(SFEIrisTracker tracker, SFEIrisDetect *detections,
00217                               size_t detections_count,
00218                               SFEIrisTracked *out_tracked,
00219                               size_t *out_tracked_count);
00220
00226 SFEError sfeIrisTrackerLost(SFEIrisTracker tracker, SFEEntity *out_entities,
00227                             size_t *in_out_entities_count);
00228
00234 SFEError sfeIrisTrackerRemoved(SFEIrisTracker tracker, SFEEntity *out_entities,
00235                                size_t *in_out_entities_count);
00236
00237 #ifdef __cplusplus
00238 } // extern "C"
00239 #endif
```

9.22 D:/glab/da0a89/1/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 SFEPalmKeypoint

Palm keypoint struct.

- struct SFEPalmDetect

  Palm detection struct.
- struct SFEPalmTemplate

  Palm template struct.
- struct SFEPalmTemplateVersion

  Palm template version struct.
- struct SFEPalmTracked

  Tracked entity struct.
- struct SFEPalmQualityAttributes

  Palm quality attributes struct.
- struct SFEPalmAttributes

Macros

- #define SFE_PALM_KEYPOINT_COUNT 8

  Palm keypoints count.
- #define SFE_PALM_TEMPLATE_SIZE 522

  Palm template size in bytes.

Typedefs

- typedef SFEEntity SFEPalmEntity
- typedef SFETemplateIdentificationCandidate SFEPalmTemplateIdentificationCandidate
- typedef SFEEntityIdentificationCandidate SFEPalmEntityIdentificationCandidate
- typedef struct SFEPalmKeypoint SFEPalmKeypoint

  Palm keypoint struct.
- typedef enum SFEHandPosition SFEHandPosition

  Palm hand enum.
- typedef enum SFEHandOrientation SFEHandOrientation

  Palm orientation enum.
- typedef struct SFEPalmDetect SFEPalmDetect

  Palm detection struct.
- typedef struct SFEPalmTemplate SFEPalmTemplate

  Palm template struct.
- typedef struct SFEPalmTemplateVersion SFEPalmTemplateVersion

  Palm template version struct.
- typedef void * SFEPalmTracker

  Tracker is a ByteTrack implementation for palm tracking across multiple frames.
- typedef struct SFEPalmTracked SFEPalmTracked

  Tracked entity struct.
- typedef struct SFEPalmQualityAttributes SFEPalmQualityAttributes

  Palm quality attributes struct.
- typedef struct SFEPalmAttributes SFEPalmAttributes

Enumerations

- 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 SFEPalmTrackedState { SFE_PALM_TRACKED_STATE_NEW = 0 , SFE_PALM_TRACKED_STATE_TRACKED = 1 , SFE_PALM_TRACKED_STATE_LOST = 2 , SFE_PALM_TRACKED_STATE_REMOVED = 3 }

  Tracked state.

Functions

- SFEError sfePalmDetect (SFESolver solver, SFEImageView image, float threshold, SFEPalmDetect *out_palm_detection, size_t *in_out_palm_count)

  Detect palm in the source image.
- SFEError sfePalmTemplateExtract (SFESolver solver, SFEImageView image, const SFEPalmDetect *in_detection, SFEPalmTemplate *out_palm_template)

  Extract template from source image and given palm detection.
- 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 SFEPalmDetect *palm_detect, float *out_liveness_score)

  Passive liveness score calculation.
- SFEError sfePalmTrackerCreate (float track_threshold, float high_threshold, float match_threshold, uint64_t max_time_lost, SFEPalmTracker *out_tracker)

  Create a new tracker.
- SFEError sfePalmTrackerFree (SFEPalmTracker tracker)

  Free the tracker.
- SFEError sfePalmTrackerUpdate (SFEPalmTracker tracker, const SFEPalmDetect *detections, size_t detections_count, SFEPalmTracked *out_tracked, size_t *out_tracked_count)

  Update the tracker.
- SFEError sfePalmTrackerLost (SFEPalmTracker tracker, SFEEntity *out_entities, size_t *in_out_entities_count)

  Get lost entities after update.
- SFEError sfePalmTrackerRemoved (SFEPalmTracker tracker, SFEEntity *out_entities, size_t *in_out_entities_count)

  Get removed entities after update.

- SFEError sfePalmQualityAttributes (SFEImageView image, const SFEPalmDetect *palm_detect, SFEPalmQualityAttributes *out_quality_attributes)

  Calculate palm image quality attributes from given source image and palm detection.
- SFEError sfePalmAttributes (SFESolver solver, SFEImageView image, const SFEPalmDetect *palm_detect, SFEPalmAttributes *out_attributes)

  Calculate palm image quality attributes from given source image and palm detection.

9.22.1 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.

9.22.2 Macro Definition Documentation

9.22.2.1 SFE_PALM_KEYPOINT_COUNT

```
#define SFE_PALM_KEYPOINT_COUNT 8
```

Palm keypoints count.

Definition at line 39 of file sfe_palm.h.

9.22.2.2 SFE_PALM_TEMPLATE_SIZE

```
#define SFE_PALM_TEMPLATE_SIZE 522
```

Palm template size in bytes.

Definition at line 85 of file sfe_palm.h.

9.22.3 Typedef Documentation

9.22.3.1 SFEHandOrientation

```
typedef enum SFEHandOrientation SFEHandOrientation
```

Palm orientation enum.

9.22.3.2 SFEHandPosition

```
typedef enum SFEHandPosition SFEHandPosition
```

Palm hand enum.

9.22.3.3 SFEPalmAttributes

```
typedef struct SFEPalmAttributes SFEPalmAttributes
```

9.22.3.4 SFEPalmDetect

```
typedef struct SFEPalmDetect SFEPalmDetect
```

Palm detection struct.

9.22.3.5 SFEPalmEntity

```
typedef SFEEntity SFEPalmEntity
```

Deprecated Use SFEEntity instead, this type will be removed in future

Definition at line 19 of file sfe_palm.h.

9.22.3.6 SFEPalmEntityIdentificationCandidate

```
typedef SFEEntityIdentificationCandidate SFEPalmEntityIdentificationCandidate
```

Deprecated Replace with SFEEntityIdentificationCandidate

Definition at line 26 of file sfe_palm.h.

9.22.3.7 SFEPalmKeypoint

```
typedef struct SFEPalmKeypoint SFEPalmKeypoint
```

Palm keypoint struct.

9.22.3.8 SFEPalmQualityAttributes

```
typedef struct SFEPalmQualityAttributes SFEPalmQualityAttributes
```

Palm quality attributes struct.

9.22.3.9 SFEPalmTemplate

```
typedef struct SFEPalmTemplate SFEPalmTemplate
```

Palm template struct.

9.22.3.10 SFEPalmTemplateIdentificationCandidate

```
typedef SFETemplateIdentificationCandidate SFEPalmTemplateIdentificationCandidate
```

Deprecated Replace with SFETemplateIdentificationCandidate

Definition at line 23 of file sfe_palm.h.

9.22.3.11 SFEPalmTemplateVersion

```
typedef struct SFEPalmTemplateVersion SFEPalmTemplateVersion
```

Palm template version struct.

9.22.3.12 SFEPalmTracked

```
typedef struct SFEPalmTracked SFEPalmTracked
```

Tracked entity struct.

9.22.3.13 SFEPalmTracker

```
typedef void* SFEPalmTracker
```

Tracker is a ByteTrack implementation for palm tracking across multiple frames.

Warning

Use `sfePalmTrackerFree` to release memory associated with the tracker.

Definition at line 189 of file sfe_palm.h.

9.22.4 Enumeration Type Documentation

9.22.4.1 SFEHandOrientation

```
enum SFEHandOrientation
```

Palm orientation enum.

Enumerator

SFE_ORIENTATION_UNKNOWN	
SFE_ORIENTATION_PALMAR	
SFE_ORIENTATION_DORSAL	

Definition at line 49 of file sfe_palm.h.

```
00049                                 {
00050   SFE_ORIENTATION_UNKNOWN = 0,
00051   SFE_ORIENTATION_PALMAR = 1,
00052   SFE_ORIENTATION_DORSAL = 2,
00053 } SFEHandOrientation;
```

9.22.4.2 SFEHandPosition

```
enum SFEHandPosition
```

Palm hand enum.

Enumerator

SFE_HAND_UNKNOWN	
SFE_HAND_LEFT	
SFE_HAND_RIGHT	

Definition at line 42 of file sfe_palm.h.

```
00042                               {
00043   SFE_HAND_UNKNOWN = 0,
00044   SFE_HAND_LEFT = 1,
00045   SFE_HAND_RIGHT = 2,
00046 } SFEHandPosition;
```

9.22.4.3 SFEPalmTrackedState

```
enum SFEPalmTrackedState
```

Tracked state.

Enumerator

SFE_PALM_TRACKED_STATE_NEW	
SFE_PALM_TRACKED_STATE_TRACKED	
SFE_PALM_TRACKED_STATE_LOST	
SFE_PALM_TRACKED_STATE_REMOVED	

Definition at line 192 of file sfe_palm.h.

```
00192                                 {
00193   SFE_PALM_TRACKED_STATE_NEW = 0,
00194   SFE_PALM_TRACKED_STATE_TRACKED = 1,
00195   SFE_PALM_TRACKED_STATE_LOST = 2,
00196   SFE_PALM_TRACKED_STATE_REMOVED = 3,
00197 };
```

9.22.5 Function Documentation

9.22.5.1 sfePalmAttributes()

```
SFEError sfePalmAttributes (
            SFESolver solver,
            SFEImageView image,
            const SFEPalmDetect * palm_detect,
            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	*palm_detect*	palm detection
out	*out_attributes*	structure containing hand_orientation, hand_position, quality

Returns

Error in case of failed attributes calculation

9.22.5.2 sfePalmDetect()

```
SFEError sfePalmDetect (
            SFESolver solver,
            SFEImageView image,
            float threshold,
            SFEPalmDetect * out_palm_detection,
            size_t * in_out_palm_count )
```

Detect palm in the source image.

Parameters

in	*solver*	palm detection solver
in	*image*	source image
in	*threshold*	detection threshold - range <0,1>
out	*out_palm_detection*	pointer to a storage where for each detected palm
in,out	*in_out_palm_count*	IN: Available space in the out_palm_detection OUT: Number of detected palms saved to out_palm_detection

Returns

Error in case of failed detection.

Examples

example_palm_identify.cpp.

9.22.5.3 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.

9.22.5.4 sfePalmLivenessPassive()

```
SFEError sfePalmLivenessPassive (
            SFESolver solver,
            SFEImageView image,
            const SFEPalmDetect * palm_detect,
            float * out_liveness_score )
```

Passive liveness score calculation.

Parameters

in	*solver*	passive liveness solver
in	*image*	source image
in	*palm_detect*	palm detection
out	*out_liveness_score*	OUT: normalized score of passive liveness

Returns

Error in case of failed passive liveness score calculation

9.22.5.5 sfePalmQualityAttributes()

```
SFEError sfePalmQualityAttributes (
            SFEImageView image,
            const SFEPalmDetect * palm_detect,
            SFEPalmQualityAttributes * out_quality_attributes )
```

Calculate palm image quality attributes from given source image and palm detection.

Parameters

in	*image*	source image
in	*palm_detect*	palm detection
out	*out_quality_attributes*	structure containing `sharpness`, `brightness`, `hotspots_score`

Returns

Error in case of failed quality attributes calculation

Deprecated Use `sfePalmAttributes` instead. Since version 2.0.0.

9.22.5.6 sfePalmTemplateExtract()

```
SFEError sfePalmTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            const SFEPalmDetect * in_detection,
            SFEPalmTemplate * out_palm_template )
```

Extract template from source image and given palm detection.

Parameters

in	*solver*	palm extraction solver
in	*image*	source image
in	*in_detection*	palm detection detected in the source image
out	*out_palm_template*	extracted palm template

Returns

Error in case of failed extraction.

Examples

example_palm_identify.cpp.

9.22.5.7 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.

9.22.5.8 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.

9.22.5.9 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.

9.22.5.10 sfePalmTrackerCreate()

```
SFEError sfePalmTrackerCreate (
            float track_threshold,
            float high_threshold,
            float match_threshold,
            uint64_t max_time_lost,
            SFEPalmTracker * out_tracker )
```

Create a new tracker.

Parameters

in	*track_threshold*	Tracking threshold
in	*high_threshold*	High threshold
in	*match_threshold*	Match threshold
in	*max_time_lost*	Maximum time lost
out	*out_tracker*	OUT: pointer to the created tracker

9.22.5.11 sfePalmTrackerFree()

```
SFEError sfePalmTrackerFree (
            SFEPalmTracker tracker )
```

Free the tracker.

Parameters

in	*tracker*	Tracker to free

9.22.5.12 sfePalmTrackerLost()

```
SFEError sfePalmTrackerLost (
            SFEPalmTracker 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*	OUT: number of lost entities

Returns

Error in case of failed lost entities retrieval

9.22.5.13 sfePalmTrackerRemoved()

```
SFEError sfePalmTrackerRemoved (
            SFEPalmTracker 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*	OUT: number of removed entities

Returns

Error in case of failed removed entities retrieval

9.22.5.14 sfePalmTrackerUpdate()

```
SFEError sfePalmTrackerUpdate (
            SFEPalmTracker tracker,
            const SFEPalmDetect * detections,
            size_t detections_count,
            SFEPalmTracked * out_tracked,
            size_t * out_tracked_count )
```

Update the tracker.

Parameters

in	*tracker*	Tracker to update
in	*detections*	Detections to update the tracker with
in	*detections_count*	Number of detections

Returns

Error in case of failed update

9.23 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 DETECTION
00017
00019 typedef SFEEntity SFEPalmEntity;
00020
00022 typedef SFETemplateIdentificationCandidate
00023     SFEPalmTemplateIdentificationCandidate;
00024
00026 typedef SFEEntityIdentificationCandidate SFEPalmEntityIdentificationCandidate;
00027
00029 typedef struct SFEPalmKeypoint {
00031   float x;
00033   float y;
00035   float confidence;
00036 } SFEPalmKeypoint;
00037
00039 #define SFE_PALM_KEYPOINT_COUNT 8
00040
00042 typedef enum SFEHandPosition {
00043   SFE_HAND_UNKNOWN = 0,
00044   SFE_HAND_LEFT = 1,
00045   SFE_HAND_RIGHT = 2,
00046 } SFEHandPosition;
00047
00049 typedef enum SFEHandOrientation {
00050   SFE_ORIENTATION_UNKNOWN = 0,
00051   SFE_ORIENTATION_PALMAR = 1,
00052   SFE_ORIENTATION_DORSAL = 2,
00053 } SFEHandOrientation;
00054
00056 typedef struct SFEPalmDetect {
00058   SFEBbox bounding_box;
00060   float confidence;
00062   SFEHandPosition hand_position;
00064   SFEHandOrientation hand_orientation;
00066   SFEPalmKeypoint keypoints[SFE_PALM_KEYPOINT_COUNT];
```

```
00067 } SFEPalmDetect;
00068
00078 SFEError sfePalmDetect(SFESolver solver, SFEImageView image, float threshold,
00079                        SFEPalmDetect *out_palm_detection,
00080                        size_t *in_out_palm_count);
00081
00082 // PALM TEMPLATE EXTRACTION AND MATCHING
00083
00085 #define SFE_PALM_TEMPLATE_SIZE 522
00086
00088 typedef struct SFEPalmTemplate {
00089   uint8_t data[SFE_PALM_TEMPLATE_SIZE];
00090 } SFEPalmTemplate;
00091
00093 typedef struct SFEPalmTemplateVersion {
00095   uint8_t major;
00097   uint8_t minor;
00099   uint8_t patch;
00100 } SFEPalmTemplateVersion;
00101
00108 SFEError sfePalmTemplateExtract(SFESolver solver, SFEImageView image,
00109                                 const SFEPalmDetect *in_detection,
00110                                 SFEPalmTemplate *out_palm_template);
00111
00117 SFEError sfePalmTemplateMatch(const SFEPalmTemplate *template1,
00118                               const SFEPalmTemplate *template2,
00119                               float *out_matching_score);
00120
00125 SFEError
00126 sfePalmTemplateVersion(const SFEPalmTemplate *palm_template,
00127                        SFEPalmTemplateVersion *out_palm_template_version);
00128
00142 SFEError
00143 sfePalmTemplateIdentify(const SFEPalmTemplate *probe_palm_template,
00144                         const SFEPalmTemplate *templates_gallery,
00145                         size_t gallery_size, float matching_score_threshold,
00146                         SFETemplateIdentificationCandidate *out_candidates,
00147                         size_t *in_out_candidates_count, size_t thread_count);
00148
00166 SFEError sfePalmEntityIdentify(const SFEPalmTemplate *probe_palm_template,
00167                                const SFEPalmTemplate *templates_gallery,
00168                                const SFEEntity *entities_gallery,
00169                                size_t gallery_size,
00170                                float matching_score_threshold,
00171                                SFEEntityIdentificationCandidate *out_candidates,
00172                                size_t *in_out_candidates_count,
00173                                size_t thread_count);
00174
00181 SFEError sfePalmLivenessPassive(SFESolver solver, SFEImageView image,
00182                                 const SFEPalmDetect *palm_detect,
00183                                 float *out_liveness_score);
00184
00189 typedef void *SFEPalmTracker;
00190
00192 enum SFEPalmTrackedState {
00193   SFE_PALM_TRACKED_STATE_NEW = 0,
00194   SFE_PALM_TRACKED_STATE_TRACKED = 1,
00195   SFE_PALM_TRACKED_STATE_LOST = 2,
00196   SFE_PALM_TRACKED_STATE_REMOVED = 3,
00197 };
00198
00200 typedef struct SFEPalmTracked {
00202   SFEPalmDetect tracked;
00204   uint64_t id;
00206   SFEEntity uuid;
00208   SFEPalmTrackedState state;
00209 } SFEPalmTracked;
00210
00217 SFEError sfePalmTrackerCreate(float track_threshold, float high_threshold,
00218                               float match_threshold, uint64_t max_time_lost,
00219                               SFEPalmTracker *out_tracker);
00220
00223 SFEError sfePalmTrackerFree(SFEPalmTracker tracker);
00224
00230 SFEError sfePalmTrackerUpdate(SFEPalmTracker tracker,
00231                               const SFEPalmDetect *detections,
00232                               size_t detections_count,
00233                               SFEPalmTracked *out_tracked,
00234                               size_t *out_tracked_count);
00235
00241 SFEError sfePalmTrackerLost(SFEPalmTracker tracker, SFEEntity *out_entities,
00242                             size_t *in_out_entities_count);
00243
00249 SFEError sfePalmTrackerRemoved(SFEPalmTracker tracker, SFEEntity *out_entities,
00250                                size_t *in_out_entities_count);
00251
00253 typedef struct SFEPalmQualityAttributes {
```

```
00255   float sharpness;
00257   float brightness;
00259   float hotspots_score;
00260 } SFEPalmQualityAttributes;
00261
00270 SFEError
00271 sfePalmQualityAttributes(SFEImageView image, const SFEPalmDetect *palm_detect,
00272                          SFEPalmQualityAttributes *out_quality_attributes);
00273
00274 typedef struct SFEPalmAttributes {
00277   float hand_position;
00281   float hand_orientation;
00283   float quality;
00284 } SFEPalmAttributes;
00285
00294 SFEError sfePalmAttributes(SFESolver solver, SFEImageView image,
00295                            const SFEPalmDetect *palm_detect,
00296                            SFEPalmAttributes *out_attributes);
00297
00298 #ifdef __cplusplus
00299 } // extern "C"
00300 #endif
```

9.24 D:/glab/da0a89/1/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 SFETattooDetect

  Tattoo detection struct.
- 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 SFEEntity SFETattooEntity
- typedef struct SFETattooDetect SFETattooDetect

  Tattoo detection struct.
- typedef struct SFETattooTemplate SFETattooTemplate

  Tattoo template struct.
- typedef struct SFETattooTemplateVersion SFETattooTemplateVersion

  Tattoo template version struct.

Functions

- SFEError sfeTattooDetect (SFESolver solver, SFEImageView image, float threshold, SFETattooDetect *out_tattoo_detection, size_t *in_out_tattoo_count)

  Detect tattoo in the source image.
- SFEError sfeTattooTemplateExtract (SFESolver solver, SFEImageView image, const SFETattooDetect *in_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.

9.24.1 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.

9.24.2 Macro Definition Documentation

9.24.2.1 SFE_TATTOO_TEMPLATE_SIZE

```
#define SFE_TATTOO_TEMPLATE_SIZE 522
```

Tattoo template size in bytes.

Definition at line 45 of file sfe_tattoo.h.

9.24.3 Typedef Documentation

9.24.3.1 SFETattooDetect

```
typedef struct SFETattooDetect SFETattooDetect
```

Tattoo detection struct.

9.24.3.2 SFETattooEntity

```
typedef SFEEntity SFETattooEntity
```

Deprecated Use SFEEntity instead, this type will be removed in future

Definition at line 19 of file sfe_tattoo.h.

9.24.3.3 SFETattooTemplate

```
typedef struct SFETattooTemplate SFETattooTemplate
```

Tattoo template struct.

9.24.3.4 SFETattooTemplateVersion

```
typedef struct SFETattooTemplateVersion SFETattooTemplateVersion
```

Tattoo template version struct.

9.24.4 Function Documentation

9.24.4.1 sfeTattooDetect()

```
SFEError sfeTattooDetect (
            SFESolver solver,
            SFEImageView image,
            float threshold,
            SFETattooDetect * out_tattoo_detection,
            size_t * in_out_tattoo_count )
```

Detect tattoo in the source image.

Parameters

in	*solver*	tattoo detection solver
in	*image*	source image
in	*threshold*	detection threshold - range <0,1>
out	*out_tattoo_detection*	pointer to a storage where for each detected tattoo
in,out	*in_out_tattoo_count*	IN: Available space in the out_tattoo_detection OUT: Number of detected tattoos saved to out_tattoo_detection

Returns

Error in case of failed detection.

9.24.4.2 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.

9.24.4.3 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.

9.24.4.4 sfeTattooTemplateExtract()

```
SFEError sfeTattooTemplateExtract (
            SFESolver solver,
            SFEImageView image,
            const SFETattooDetect * in_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	*in_detection*	tattoo detection detected in the source image
out	*out_tattoo_template*	extracted tattoo template

Returns

Error in case of failed extraction.

9.24.4.5 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.

9.24.4.6 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.

9.24.4.7 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.

9.24.4.8 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.

9.25 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 DETECTION
00017
00019 typedef SFEEntity SFETattooEntity;
00020
00022 typedef struct SFETattooDetect {
00024   SFEBbox bounding_box;
00026   float confidence;
00027 } SFETattooDetect;
00028
00038 SFEError sfeTattooDetect(SFESolver solver, SFEImageView image, float threshold,
00039                          SFETattooDetect *out_tattoo_detection,
00040                          size_t *in_out_tattoo_count);
00041
00042 // TATTOO TEMPLATE EXTRACTION AND MATCHING
00043
00045 #define SFE_TATTOO_TEMPLATE_SIZE 522
00046
00048 typedef struct SFETattooTemplate {
00049   uint8_t data[SFE_TATTOO_TEMPLATE_SIZE];
00050 } SFETattooTemplate;
00051
00053 typedef struct SFETattooTemplateVersion {
00055   uint32_t version;
00056 } SFETattooTemplateVersion;
00057
00064 SFEError sfeTattooTemplateExtract(SFESolver solver, SFEImageView image,
00065                                   const SFETattooDetect *in_detection,
00066                                   SFETattooTemplate *out_tattoo_template);
00067
00073 SFEError sfeTattooTemplateMatch(const SFETattooTemplate *template1,
00074                                 const SFETattooTemplate *template2,
00075                                 float *out_matching_score);
00076
00081 SFEError
00082 sfeTattooTemplateVersion(const SFETattooTemplate *tattoo_template,
00083                          SFETattooTemplateVersion *out_tattoo_template_version);
00084
00098 SFEError
00099 sfeTattooTemplateIdentify(const SFETattooTemplate *probe_tattoo_template,
00100                           const SFETattooTemplate *templates_gallery,
00101                           size_t gallery_size, float matching_score_threshold,
00102                           SFETemplateIdentificationCandidate *out_candidates,
00103                           size_t *in_out_candidates_count, size_t thread_count);
00104
00122 SFEError sfeTattooEntityIdentify(const SFETattooTemplate *probe_tattoo_template,
00123                                  const SFETattooTemplate *templates_gallery,
00124                                  const SFEEntity *entities_gallery,
00125                                  size_t gallery_size,
00126                                  float matching_score_threshold,
00127                                  SFEEntityIdentificationCandidate *out_candidates,
00128                                  size_t *in_out_candidates_count,
00129                                  size_t thread_count);
00130
00137 SFEError sfeTattooTemplateExport(const SFETattooTemplate *tattoo_template,
00138                                  uint8_t *bytes,
```

```
00139                                   size_t *size);
00140
00146 SFEError sfeTattooTemplateImport(const uint8_t *bytes,
00147                                   size_t size,
00148                                   SFETattooTemplate *out_tattoo_template);
00149
00150 #ifdef __cplusplus
00151 } // extern "C"
00152 #endif
```

9.26 D:/glab/da0a89/1/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_TEMPLATE;

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
```

```
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &recommended_width, &recommended_height);
    utils::checkError(error);
  };
}

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,
                SFEFaceDetect &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 = sfeFaceDetect(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 = sfeSolverCreateWithParameters(
        solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &detector_solver);
    utils::checkError(error);

    // Initialize landmarks detection solver
    error = sfeSolverCreateWithParameters(
        solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &landmarks_solver);
    utils::checkError(error);

    // Initialize template extraction solver
    error = sfeSolverCreateWithParameters(
        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 = sfeImageLoad(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
  }

  SFEFaceDetect 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.bbox,
                             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, landmarks.data(),
                                   detected_face.raw_confidence,
                                   &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;
  }
}

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<SFEFaceDetect> 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 = sfeImageLoad(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].bbox, 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, landmarks[i].data(),
          detected_faces[i].raw_confidence, &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<SFEIdentificationResult> 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 = sfeImageLoad(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.
      uint32_t face_size_extension = 2;
      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 = sfeImageSave(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].bbox,
                         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 = sfeImageSave(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_TEMPLATE;

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
    SFEError error = sfeFaceDetectInputSize(image,
      SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, face_size_min,
                                            face_size_max, &recommended_width,
                                            &recommended_height);
    utils::checkError(error);
  };
}

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 = sfeImageLoad(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);
  }

  SFEFaceDetect detected_face = {};
  size_t detection_count = 1;
  { // STAGE 3: Detect face in the image

    // Detect face in the image
    error = sfeFaceDetect(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.bbox,
                             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, landmarks.data(),
                               detected_face.raw_confidence, &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 = sfeSolverCreateWithParameters(
      solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  // Initialize landmarks detection solver
  error = sfeSolverCreateWithParameters(
      solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);

  // Initialize template extraction solver
  error = sfeSolverCreateWithParameters(
      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<SFEFaceEntity> 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<SFEFaceEntityIdentificationCandidate> 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
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &recommended_width, &recommended_height);
    utils::checkError(error);
  };
}

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<SFEFaceDetect> 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<SFEFaceDetect> detected_faces(detection_count);
  { // STAGE 2: Detect face in the image

    // Detect faces in the image
    error = sfeFaceDetect(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 SFEFaceTrackedState &state) {
  switch (state) {
  case SFE_FACE_TRACKED_STATE_NEW:
    os « "NEW";
    break;
  case SFE_FACE_TRACKED_STATE_TRACKED:
    os « "TRACKED";
    break;
  case SFE_FACE_TRACKED_STATE_LOST:
    os « "LOST";
    break;
  case SFE_FACE_TRACKED_STATE_REMOVED:
    os « "REMOVED";
    break;
  default:
    os « "UNKNOWN";
    break;
  }
  return os;
}

std::ostream &operator«(std::ostream &os, const SFEFaceTracked &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<SFEFaceDetect> &detections, SFEFaceTracker 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<SFEFaceTracked> 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 =
      sfeFaceTrackerUpdate(tracker, detections.data(), detections.size(),
                           tracked_faces.data(), &tracked_faces_count);
  utils::checkError(error);
  tracked_faces.resize(tracked_faces_count);

  // Get lost faces
  error = sfeFaceTrackerLost(tracker, lost_faces.data(), &lost_faces_count);
  utils::checkError(error);
  lost_faces.resize(lost_faces_count);

  // Get removed faces
  error = sfeFaceTrackerRemoved(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 = sfeSolverCreateWithParameters(
        solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &detector_solver);
    utils::checkError(error);
  }

  std::vector<SFEFaceDetect> 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 = sfeImageLoad(image_data.data(), image_data.size(), &image);
      utils::checkError(error);
    }

    SFEFaceDetect detected_face = {};
    { // STAGE 2.2: Detect face in the image
      // Detect a face in the image
      detected_faces = detectFaces(image, detector_solver);
    }
  }

  SFEFaceTracker tracker{};
  DEFER(sfeFaceTrackerFree(tracker));
  { // STAGE 3: Track faces
    utils::printFormatted("TRACK FACES");
    error = sfeFaceTrackerCreate(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
    SFEError error = sfeFaceDetectInputSize(
        image,
        SFEFaceDetectAccuracyType::SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE,
        face_size_min, face_size_max, &recommended_width, &recommended_height);
    utils::checkError(error);
  };
}

inline void printFaceDetectionInfo(SFEFaceDetect &detected_face) {
  std::cout << "SFEFaceDetect{ " << std::endl;
  std::cout << "  confidence: " << detected_face.confidence << std::endl;
  std::cout << "  SFEBbox{ " << std::endl;
  std::cout << "    x: " << detected_face.bbox.x << std::endl;
  std::cout << "    y: " << detected_face.bbox.y << std::endl;
  std::cout << "    width: " << detected_face.bbox.width << std::endl;
  std::cout << "    height: " << detected_face.bbox.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 = sfeSolverCreateWithParameters(
      solver_face_detect.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &detector_solver);
  utils::checkError(error);

  // Initialize landmarks detection solver
  error = sfeSolverCreateWithParameters(
      solver_face_landmarks.c_str(), SOLVER_PARAMETERS.data(),
      SOLVER_PARAMETERS.size(), &landmarks_solver);
  utils::checkError(error);

  // Initialize face liveness solver
  error = sfeSolverCreateWithParameters(
      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 = sfeImageLoad(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);
}
SFEFaceDetect detected_face = {};
{ // STAGE 3: Detect face in the image
  size_t detection_count = 1;
  // Detect face in the image
  error = sfeFaceDetect(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.bbox,
                           landmarks.data());
  utils::checkError(error);
}

float liveness_score = 0.0f;
{ // STAGE 5: Liveness check
  utils::printFormatted("LIVENESS CHECK");
  error = sfeFaceLivenessPassive(liveness_solver, image, landmarks.data(),
                                 &liveness_score);
  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.bbox, 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 = sfeImageSave(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_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 = sfeImageLoad(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
  }

  SFEIrisDetect detected_iris = {};
  { // STAGE 2: Detect iris in the image

    // Detect iris in the image
    error = sfeIrisDetect(detector_solver, image, &detected_iris);
    utils::checkError(error);

    if (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 = sfeSolverCreateWithParameters(
        solver_iris_detect.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &detector_solver);
    utils::checkError(error);

    // Initialize template extraction solver
    error = sfeSolverCreateWithParameters(
        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<SFEIrisTemplateIdentificationCandidate> 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");
}
```

10.6 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;

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

float identification_threshold = 0.6f;

SFEPalmTemplate extract_template(std::string image_path,
                                 SFESolver &detector_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 = sfeImageLoad(image_data.data(), image_data.size(), &image);
    utils::checkError(error);
  }

  SFEPalmDetect 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 = sfePalmDetect(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;
  }

  SFEPalmTemplate palm_template;
  { // STAGE 3 Extract probe palm template
    // Use template extraction solver to create new palm
    // template
    error = sfePalmTemplateExtract(extraction_solver, image, &detected_palm,
                                   &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 « "-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("-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 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 extraction_solver{};
  DEFER(sfeSolverFree(extraction_solver));
  { // STAGE 1.1: loading solvers
    utils::printFormatted("LOADING SOLVERS");
    // Initialize Palm detection solver
    error = sfeSolverCreateWithParameters(
        solver_palm_detect.c_str(), SOLVER_PARAMETERS.data(),
        SOLVER_PARAMETERS.size(), &detector_solver);
    utils::checkError(error);
    error = sfeSolverCreateWithParameters(
        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, 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, extraction_solver);

  auto non_matching_palm_template =
      extract_template(image_non_match, detector_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<SFEPalmTemplateIdentificationCandidate> 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");
}
```

Index

angle
    SFEOrientedBbox, 92
area
    SFEFaceArea, 73
area_in_image
    SFEFaceArea, 73

bbox
    SFEFaceDetect, 76
bounding_box
    SFEPalmDetect, 94
    SFETattooDetect, 102
brightness
    SFEFaceQualityAttributes, 80
    SFEPalmQualityAttributes, 97

center_x
    SFEOrientedBbox, 92
center_y
    SFEOrientedBbox, 92
Changelog, 7
confidence
    SFEFaceDetect, 76
    SFEFaceLandmarks, 78
    SFEIrisDetect, 86
    SFEIrisKeypoint, 87
    SFEPalmDetect, 94
    SFEPalmKeypoint, 96
    SFETattooDetect, 102
contrast
    SFEFaceQualityAttributes, 80
crop_box
    SFEFaceCrop, 74
crop_image
    SFEFaceCrop, 74

D:/glab/da0a89/1/changelog.md, 105
D:/glab/da0a89/1/example/example_face_identify.cpp, 105, 115
D:/glab/da0a89/1/example/example_face_identify_entity.cpp, 121, 129
D:/glab/da0a89/1/example/example_face_liveness.cpp, 133, 142
D:/glab/da0a89/1/example/example_face_track.cpp, 146, 153
D:/glab/da0a89/1/example/example_iris_identify.cpp, 157, 163
D:/glab/da0a89/1/example/example_palm_identify.cpp, 166, 171
D:/glab/da0a89/1/include/sfe_core.h, 174, 187
D:/glab/da0a89/1/include/sfe_face.h, 189, 209
D:/glab/da0a89/1/include/sfe_iris.h, 211, 222
D:/glab/da0a89/1/include/sfe_palm.h, 223, 236
D:/glab/da0a89/1/include/sfe_tattoo.h, 238, 244
D:/glab/da0a89/1/readme.md, 245
data
    SFEFaceTemplate, 81
    SFEImage, 85
    SFEIrisTemplate, 89
    SFEPalmTemplate, 98
    SFETattooTemplate, 103
Deprecated List, 63
detectFace
    example_face_identify.cpp, 106
detectFaces
    example_face_track.cpp, 147
detection_threshold
    example_face_identify.cpp, 113
    example_face_identify_entity.cpp, 127
    example_face_liveness.cpp, 140
    example_face_track.cpp, 152

entity
    SFEEntityIdentificationCandidate, 72
example_face_identify.cpp
    detectFace, 106
    detection_threshold, 113
    face_size_max, 113
    face_size_min, 113
    getRecommendedImageSize, 107
    identification_threshold, 113
    image_gallery, 113
    image_probe, 114
    main, 107
    printHelp, 112
    solver_face_detect, 114
    solver_face_landmarks, 114
    solver_face_template, 114
    SOLVER_PARAMETERS, 115
example_face_identify_entity.cpp
    detection_threshold, 127
    extractTemplate, 121
    face_size_max, 127
    face_size_min, 127
    gallery, 127
    getRecommendedImageSize, 122
    identification_threshold, 127
    image_probe, 128
    main, 124
    printHelp, 126

solver_face_detect, 128
solver_face_landmarks, 128
solver_face_template, 128
SOLVER_PARAMETERS, 128
example_face_liveness.cpp
detection_threshold, 140
face_size_max, 140
face_size_min, 140
getRecommendedImageSize, 133
image_probe, 140
main, 134
prinFaceAreaInfo, 138
printFaceDetectionInfo, 138
printFaceHeadPose, 138
printFaceQualityAttributes, 139
printFaceSize, 139
printHelp, 139
printMaskConfidence, 140
solver_face_detect, 141
solver_face_landmarks, 141
solver_face_liveness, 141
SOLVER_PARAMETERS, 141
example_face_track.cpp
detectFaces, 147
detection_threshold, 152
face_size_max, 152
face_size_min, 152
frame, 147
getRecommendedImageSize, 148
image_probe, 152
main, 149
operator<<, 150, 151
printHelp, 152
solver_face_detect, 153
SOLVER_PARAMETERS, 153
example_iris_identify.cpp
extract_template, 158
identification_threshold, 161
image_match, 161
image_non_match, 161
image_probe, 162
main, 158
printHelp, 161
solver_iris_detect, 162
solver_iris_template, 162
SOLVER_PARAMETERS, 162
example_palm_identify.cpp
extract_template, 166
identification_threshold, 170
image_match, 170
image_non_match, 170
image_probe, 170
main, 167
printHelp, 169
solver_palm_detect, 170
solver_palm_extraction, 170
SOLVER_PARAMETERS, 170
extract_template
example_iris_identify.cpp, 158
example_palm_identify.cpp, 166
extractTemplate
example_face_identify_entity.cpp, 121

face_size_extension
SFEFaceCrop, 74
face_size_max
example_face_identify.cpp, 113
example_face_identify_entity.cpp, 127
example_face_liveness.cpp, 140
example_face_track.cpp, 152
face_size_min
example_face_identify.cpp, 113
example_face_identify_entity.cpp, 127
example_face_liveness.cpp, 140
example_face_track.cpp, 152
Features, 31
frame
example_face_track.cpp, 147

gallery
example_face_identify_entity.cpp, 127
getRecommendedImageSize
example_face_identify.cpp, 107
example_face_identify_entity.cpp, 122
example_face_liveness.cpp, 133
example_face_track.cpp, 148

hand_orientation
SFEPalmAttributes, 93
SFEPalmDetect, 94
hand_position
SFEPalmAttributes, 93
SFEPalmDetect, 95
height
SFEBbox, 69
SFEImage, 85
SFEOrientedBbox, 92
hotspots_score
SFEPalmQualityAttributes, 97

id
SFEFaceTracked, 83
SFEIrisTracked, 90
SFEPalmTracked, 100
identification_threshold
example_face_identify.cpp, 113
example_face_identify_entity.cpp, 127
example_iris_identify.cpp, 161
example_palm_identify.cpp, 170
image_gallery
example_face_identify.cpp, 113
image_match
example_iris_identify.cpp, 161
example_palm_identify.cpp, 170
image_non_match
example_iris_identify.cpp, 161
example_palm_identify.cpp, 170

image_probe
example_face_identify.cpp, 114
example_face_identify_entity.cpp, 128
example_face_liveness.cpp, 140
example_face_track.cpp, 152
example_iris_identify.cpp, 162
example_palm_identify.cpp, 170
index
SFETemplateIdentificationCandidate, 104
iris_bounding_box
SFEIrisDetect, 86

keypoint_type
SFEIrisKeypoint, 87
keypoints
SFEIrisDetect, 86
SFEPalmDetect, 95

landmark_type
SFEFaceLandmarks, 78

main
example_face_identify.cpp, 107
example_face_identify_entity.cpp, 124
example_face_liveness.cpp, 134
example_face_track.cpp, 149
example_iris_identify.cpp, 158
example_palm_identify.cpp, 167
major
SFEPalmTemplateVersion, 99
minor
SFEPalmTemplateVersion, 99

name
SFESolverParameter, 101

operator<<
example_face_track.cpp, 150, 151
Overview, 1

patch
SFEPalmTemplateVersion, 99
pitch
SFEFaceHeadPose, 77
prinFaceAreaInfo
example_face_liveness.cpp, 138
printFaceDetectionInfo
example_face_liveness.cpp, 138
printFaceHeadPose
example_face_liveness.cpp, 138
printFaceQualityAttributes
example_face_liveness.cpp, 139
printFaceSize
example_face_liveness.cpp, 139
printHelp
example_face_identify.cpp, 112
example_face_identify_entity.cpp, 126
example_face_liveness.cpp, 139
example_face_track.cpp, 152
example_iris_identify.cpp, 161
example_palm_identify.cpp, 169
printMaskConfidence
example_face_liveness.cpp, 140
pupil_bounding_box
SFEIrisDetect, 86

quality
SFEPalmAttributes, 93

raw_confidence
SFEFaceDetect, 76
roll
SFEFaceHeadPose, 77

score
SFEEntityIdentificationCandidate, 72
SFETemplateIdentificationCandidate, 104
sfe_core.h
SFE_HWID_METHOD_AMAZON, 179
SFE_HWID_METHOD_ANDROID_ID, 179
SFE_HWID_METHOD_APP_ID, 179
SFE_HWID_METHOD_AUTO, 179
SFE_HWID_METHOD_DISK_ID, 179
SFE_HWID_METHOD_IMEI, 179
SFE_HWID_METHOD_MAC, 179
SFE_HWID_METHOD_MAC_ADDRESS, 179
SFE_HWID_METHOD_PHY, 179
SFE_HWID_METHOD_SERIAL_NUMBER, 179
SFE_HWID_METHOD_SM_BIOS, 179
SFE_HWID_METHOD_SYS, 179
SFE_IMAGE_FORMAT_AVIF, 179
SFE_IMAGE_FORMAT_BMP, 179
SFE_IMAGE_FORMAT_DDS, 179
SFE_IMAGE_FORMAT_FARBFELD, 179
SFE_IMAGE_FORMAT_GIF, 179
SFE_IMAGE_FORMAT_HDR, 179
SFE_IMAGE_FORMAT_ICO, 179
SFE_IMAGE_FORMAT_JPEG, 179
SFE_IMAGE_FORMAT_OPENEXR, 179
SFE_IMAGE_FORMAT_PNG, 179
SFE_IMAGE_FORMAT_PNM, 179
SFE_IMAGE_FORMAT_TGA, 179
SFE_IMAGE_FORMAT_TIFF, 179
SFE_IMAGE_FORMAT_WEBP, 179
SFE_LICENSE_SOURCE_ENV, 180
SFE_LICENSE_SOURCE_FILE, 180
SFE_LICENSE_SOURCE_MEMORY, 180
SFE_LICENSE_SOURCE_NONE, 180
SFE_LICENSE_SOURCE_TOKEN, 180
SFEBbox, 176
SFEEntity, 176
SFEEntityIdentificationCandidate, 176
SFEError, 176
sfeErrorFree, 180
sfeErrorMessage, 180
sfeHwidGet, 181
SFEHwidMethod, 177, 178
SFEImage, 177
sfeImageColorTranspose, 181

SFEImageFormat, 177, 179
sfeImageFree, 182
sfeImageInfo, 182
sfeImageLoad, 182
sfeImageResize, 183
sfeImageResizeWithAspectRatio, 184
sfeImageSave, 184
SFEImageView, 177
sfeLicenseSet, 184
SFELicenseSource, 178, 180
sfeLicenseValidate, 185
SFEOrientedBbox, 178
SFESolver, 178
sfeSolverCreate, 185
sfeSolverCreateWithParameters, 186
sfeSolverFree, 186
SFESolverParameter, 178
SFETemplateIdentificationCandidate, 178
sfeVersion, 187
sfe_face.h
SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE, 196
SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED, 196
SFE_FACE_LANDMARK_COUNT, 193
SFE_FACE_LANDMARK_TYPE_CHIN_TIP, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EDGE, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE, 196
SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM, 196
SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM, 196
SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM, 196
SFE_FACE_LANDMARK_TYPE_NOSE_ROOT, 196
SFE_FACE_LANDMARK_TYPE_NOSE_TIP, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END 196
SFE_FACE_TEMPLATE_SIZE, 193
SFE_FACE_TRACKED_STATE_LOST, 197
SFE_FACE_TRACKED_STATE_NEW, 197
SFE_FACE_TRACKED_STATE_REMOVED, 197
SFE_FACE_TRACKED_STATE_TRACKED, 197
SFEFaceArea, 193
sfeFaceArea, 197
SFEFaceCrop, 193
sfeFaceCrop, 198
SFEFaceDetect, 193
sfeFaceDetect, 198
SFEFaceDetectAccuracyType, 193, 195
sfeFaceDetectInputSize, 199
SFEFaceEntity, 193
SFEFaceEntityIdentificationCandidate, 194
sfeFaceEntityIdentify, 199
SFEFaceHeadPose, 194
sfeFaceHeadPose, 201
SFEFaceLandmarks, 194
sfeFaceLandmarks, 201
SFEFaceLandmarkType, 194, 196
sfeFaceLivenessPassive, 202
sfeFaceMaskConfidence, 202
SFEFaceQualityAttributes, 194
sfeFaceQualityAttributes, 203
sfeFaceSize, 203
SFEFaceTemplate, 194
sfeFaceTemplateExtract, 204
SFEFaceTemplateIdentificationCandidate, 195
sfeFaceTemplateIdentify, 204
sfeFaceTemplateMatch, 205
sfeFaceTemplateQuality, 205
SFEFaceTemplateVersion, 195
sfeFaceTemplateVersion, 206
SFEFaceTracked, 195
SFEFaceTrackedState, 197
SFEFaceTracker, 195
sfeFaceTrackerCreate, 206
sfeFaceTrackerFree, 207
sfeFaceTrackerLost, 207
sfeFaceTrackerRemoved, 207
sfeFaceTrackerUpdate, 208
SFEIdentificationResult, 195
SFE_FACE_DETECT_ACCURACY_TYPE_ACCURATE
sfe_face.h, 196
SFE_FACE_DETECT_ACCURACY_TYPE_BALANCED
sfe_face.h, 196

SFE_FACE_LANDMARK_COUNT
sfe_face.h, 193
SFE_FACE_LANDMARK_TYPE_CHIN_TIP
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EDGE
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_CENTER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_INNER_CORNER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYE_OUTER_CORNER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_INNER_END
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_LEFT_EYEBROW_OUTER_END
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_CENTER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_LEFT_CORNER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_LOWER_EDGE
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_RIGHT_CORNER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_MOUTH_UPPER_EDGE
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_NOSE_BOTTOM
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_NOSE_LEFT_BOTTOM
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_NOSE_RIGHT_BOTTOM
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_NOSE_ROOT
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_NOSE_TIP
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EDGE
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_CENTER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_INNER_CORNER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYE_OUTER_CORNER
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_INNER_END
sfe_face.h, 196
SFE_FACE_LANDMARK_TYPE_RIGHT_EYEBROW_OUTER_END
sfe_face.h, 196
SFE_FACE_TEMPLATE_SIZE
sfe_face.h, 193
SFE_FACE_TRACKED_STATE_LOST
sfe_face.h, 197
SFE_FACE_TRACKED_STATE_NEW
sfe_face.h, 197
SFE_FACE_TRACKED_STATE_REMOVED
sfe_face.h, 197
SFE_FACE_TRACKED_STATE_TRACKED
sfe_face.h, 197
sfe_features.md, 105
SFE_HAND_LEFT
sfe_palm.h, 229
SFE_HAND_RIGHT
sfe_palm.h, 229
SFE_HAND_UNKNOWN
sfe_palm.h, 229
SFE_HWID_METHOD_AMAZON
sfe_core.h, 179
SFE_HWID_METHOD_ANDROID_ID
sfe_core.h, 179
SFE_HWID_METHOD_APP_ID
sfe_core.h, 179
SFE_HWID_METHOD_AUTO
sfe_core.h, 179
SFE_HWID_METHOD_DISK_ID
sfe_core.h, 179
SFE_HWID_METHOD_IMEI
sfe_core.h, 179
SFE_HWID_METHOD_MAC
sfe_core.h, 179
SFE_HWID_METHOD_MAC_ADDRESS
sfe_core.h, 179
SFE_HWID_METHOD_PHY
sfe_core.h, 179
SFE_HWID_METHOD_SERIAL_NUMBER
sfe_core.h, 179
SFE_HWID_METHOD_SM_BIOS
sfe_core.h, 179
SFE_HWID_METHOD_SYS
sfe_core.h, 179
SFE_IMAGE_FORMAT_AVIF
sfe_core.h, 179
SFE_IMAGE_FORMAT_BMP
sfe_core.h, 179
SFE_IMAGE_FORMAT_DDS
sfe_core.h, 179
SFE_IMAGE_FORMAT_FARBFELD
sfe_core.h, 179
SFE_IMAGE_FORMAT_GIF
sfe_core.h, 179
SFE_IMAGE_FORMAT_HDR
sfe_core.h, 179
SFE_IMAGE_FORMAT_ICO
sfe_core.h, 179
SFE_IMAGE_FORMAT_JPEG
sfe_core.h, 179
SFE_IMAGE_FORMAT_OPENEXR
sfe_core.h, 179
SFE_IMAGE_FORMAT_PNG
sfe_core.h, 179
SFE_IMAGE_FORMAT_PNM
sfe_core.h, 179
SFE_IMAGE_FORMAT_TGA
sfe_core.h, 179
SFE_IMAGE_FORMAT_TIFF
sfe_core.h, 179
SFE_IMAGE_FORMAT_WEBP

sfe_core.h, 179
sfe_iris.h
SFE_IRIS_KEYPOINT_COUNT, 213
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT, 216
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT, 216
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT, 216
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT, 216
SFE_IRIS_TEMPLATE_SIZE, 213
SFE_IRIS_TRACKED_STATE_LOST, 216
SFE_IRIS_TRACKED_STATE_NEW, 216
SFE_IRIS_TRACKED_STATE_REMOVED, 216
SFE_IRIS_TRACKED_STATE_TRACKED, 216
SFEIrisDetect, 214
sfeIrisDetect, 216
SFEIrisEntity, 214
SFEIrisEntityIdentificationCandidate, 214
sfeIrisEntityIdentify, 217
SFEIrisKeypoint, 214
SFEIrisKeypointType, 214, 215
SFEIrisTemplate, 214
sfeIrisTemplateExtract, 217
SFEIrisTemplateIdentificationCandidate, 215
sfeIrisTemplateIdentify, 218
sfeIrisTemplateMatch, 219
sfeIrisTemplateQuality, 219
SFEIrisTemplateVersion, 215
sfeIrisTemplateVersion, 219
SFEIrisTracked, 215
SFEIrisTrackedState, 216
SFEIrisTracker, 215
sfeIrisTrackerCreate, 220
sfeIrisTrackerFree, 220
sfeIrisTrackerLost, 221
sfeIrisTrackerRemoved, 221
sfeIrisTrackerUpdate, 221
SFE_IRIS_KEYPOINT_COUNT
sfe_iris.h, 213
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_LEFT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_INNER_RIGHT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_LEFT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_BOTTOM_RIGHT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_LEFT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_INNER_RIGHT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_LEFT
sfe_iris.h, 216
SFE_IRIS_KEYPOINT_TYPE_TOP_RIGHT
sfe_iris.h, 216
SFE_IRIS_TEMPLATE_SIZE
sfe_iris.h, 213
SFE_IRIS_TRACKED_STATE_LOST
sfe_iris.h, 216
SFE_IRIS_TRACKED_STATE_NEW
sfe_iris.h, 216
SFE_IRIS_TRACKED_STATE_REMOVED
sfe_iris.h, 216
SFE_IRIS_TRACKED_STATE_TRACKED
sfe_iris.h, 216
SFE_LICENSE_SOURCE_ENV
sfe_core.h, 180
SFE_LICENSE_SOURCE_FILE
sfe_core.h, 180
SFE_LICENSE_SOURCE_MEMORY
sfe_core.h, 180
SFE_LICENSE_SOURCE_NONE
sfe_core.h, 180
SFE_LICENSE_SOURCE_TOKEN
sfe_core.h, 180
SFE_ORIENTATION_DORSAL
sfe_palm.h, 229
SFE_ORIENTATION_PALMAR
sfe_palm.h, 229
SFE_ORIENTATION_UNKNOWN
sfe_palm.h, 229
sfe_palm.h
SFE_HAND_LEFT, 229
SFE_HAND_RIGHT, 229
SFE_HAND_UNKNOWN, 229
SFE_ORIENTATION_DORSAL, 229
SFE_ORIENTATION_PALMAR, 229
SFE_ORIENTATION_UNKNOWN, 229
SFE_PALM_KEYPOINT_COUNT, 226
SFE_PALM_TEMPLATE_SIZE, 226
SFE_PALM_TRACKED_STATE_LOST, 229
SFE_PALM_TRACKED_STATE_NEW, 229
SFE_PALM_TRACKED_STATE_REMOVED, 229
SFE_PALM_TRACKED_STATE_TRACKED, 229
SFEHandOrientation, 226, 228
SFEHandPosition, 226, 229
SFEPalmAttributes, 227
sfePalmAttributes, 230
SFEPalmDetect, 227
sfePalmDetect, 230
SFEPalmEntity, 227
SFEPalmEntityIdentificationCandidate, 227
sfePalmEntityIdentify, 230
SFEPalmKeypoint, 227
sfePalmLivenessPassive, 231
SFEPalmQualityAttributes, 227
sfePalmQualityAttributes, 232
SFEPalmTemplate, 227

sfePalmTemplateExtract, 232
SFEPalmTemplateIdentificationCandidate, 228
sfePalmTemplateIdentify, 232
sfePalmTemplateMatch, 233
SFEPalmTemplateVersion, 228
sfePalmTemplateVersion, 234
SFEPalmTracked, 228
SFEPalmTrackedState, 229
SFEPalmTracker, 228
sfePalmTrackerCreate, 234
sfePalmTrackerFree, 234
sfePalmTrackerLost, 235
sfePalmTrackerRemoved, 235
sfePalmTrackerUpdate, 235
SFE_PALM_KEYPOINT_COUNT
sfe_palm.h, 226
SFE_PALM_TEMPLATE_SIZE
sfe_palm.h, 226
SFE_PALM_TRACKED_STATE_LOST
sfe_palm.h, 229
SFE_PALM_TRACKED_STATE_NEW
sfe_palm.h, 229
SFE_PALM_TRACKED_STATE_REMOVED
sfe_palm.h, 229
SFE_PALM_TRACKED_STATE_TRACKED
sfe_palm.h, 229
sfe_solvers.md, 105
sfe_tattoo.h
SFE_TATTOO_TEMPLATE_SIZE, 239
SFETattooDetect, 240
sfeTattooDetect, 240
SFETattooEntity, 240
sfeTattooEntityIdentify, 241
SFETattooTemplate, 240
sfeTattooTemplateExport, 241
sfeTattooTemplateExtract, 242
sfeTattooTemplateIdentify, 242
sfeTattooTemplateImport, 243
sfeTattooTemplateMatch, 243
SFETattooTemplateVersion, 240
sfeTattooTemplateVersion, 243
SFE_TATTOO_TEMPLATE_SIZE
sfe_tattoo.h, 239
SFEBbox, 69
height, 69
sfe_core.h, 176
width, 69
x, 70
y, 70
SFEEntity, 70
sfe_core.h, 176
uuid, 71
SFEEntityIdentificationCandidate, 71
entity, 72
score, 72
sfe_core.h, 176
SFEError
sfe_core.h, 176
sfeErrorFree
sfe_core.h, 180
sfeErrorMessage
sfe_core.h, 180
SFEFaceArea, 72
area, 73
area_in_image, 73
sfe_face.h, 193
size, 73
sfeFaceArea
sfe_face.h, 197
SFEFaceCrop, 74
crop_box, 74
crop_image, 74
face_size_extension, 74
sfe_face.h, 193
sfeFaceCrop
sfe_face.h, 198
SFEFaceDetect, 75
bbox, 76
confidence, 76
raw_confidence, 76
sfe_face.h, 193
sfeFaceDetect
sfe_face.h, 198
SFEFaceDetectAccuracyType
sfe_face.h, 193, 195
sfeFaceDetectInputSize
sfe_face.h, 199
SFEFaceEntity
sfe_face.h, 193
SFEFaceEntityIdentificationCandidate
sfe_face.h, 194
sfeFaceEntityIdentify
sfe_face.h, 199
SFEFaceHeadPose, 76
pitch, 77
roll, 77
sfe_face.h, 194
yaw, 77
sfeFaceHeadPose
sfe_face.h, 201
SFEFaceLandmarks, 78
confidence, 78
landmark_type, 78
sfe_face.h, 194
x, 79
y, 79
sfeFaceLandmarks
sfe_face.h, 201
SFEFaceLandmarkType
sfe_face.h, 194, 196
sfeFaceLivenessPassive
sfe_face.h, 202
sfeFaceMaskConfidence
sfe_face.h, 202
SFEFaceQualityAttributes, 79
brightness, 80

contrast, 80
sfe_face.h, 194
sharpness, 80
unique_intensity_levels, 80
sfeFaceQualityAttributes
sfe_face.h, 203
sfeFaceSize
sfe_face.h, 203
SFEFaceTemplate, 81
data, 81
sfe_face.h, 194
sfeFaceTemplateExtract
sfe_face.h, 204
SFEFaceTemplateIdentificationCandidate
sfe_face.h, 195
sfeFaceTemplateIdentify
sfe_face.h, 204
sfeFaceTemplateMatch
sfe_face.h, 205
sfeFaceTemplateQuality
sfe_face.h, 205
SFEFaceTemplateVersion, 82
sfe_face.h, 195
version_major, 82
version_minor, 82
sfeFaceTemplateVersion
sfe_face.h, 206
SFEFaceTracked, 83
id, 83
sfe_face.h, 195
state, 83
tracked, 83
uuid, 84
SFEFaceTrackedState
sfe_face.h, 197
SFEFaceTracker
sfe_face.h, 195
sfeFaceTrackerCreate
sfe_face.h, 206
sfeFaceTrackerFree
sfe_face.h, 207
sfeFaceTrackerLost
sfe_face.h, 207
sfeFaceTrackerRemoved
sfe_face.h, 207
sfeFaceTrackerUpdate
sfe_face.h, 208
SFEHandOrientation
sfe_palm.h, 226, 228
SFEHandPosition
sfe_palm.h, 226, 229
sfeHwidGet
sfe_core.h, 181
SFEHwidMethod
sfe_core.h, 177, 178
SFEIdentificationResult
sfe_face.h, 195
SFEImage, 84
data, 85
height, 85
sfe_core.h, 177
width, 85
sfeImageColorTranspose
sfe_core.h, 181
SFEImageFormat
sfe_core.h, 177, 179
sfeImageFree
sfe_core.h, 182
sfeImageInfo
sfe_core.h, 182
sfeImageLoad
sfe_core.h, 182
sfeImageResize
sfe_core.h, 183
sfeImageResizeWithAspectRatio
sfe_core.h, 184
sfeImageSave
sfe_core.h, 184
SFEImageView
sfe_core.h, 177
SFEIrisDetect, 85
confidence, 86
iris_bounding_box, 86
keypoints, 86
pupil_bounding_box, 86
sfe_iris.h, 214
sfeIrisDetect
sfe_iris.h, 216
SFEIrisEntity
sfe_iris.h, 214
SFEIrisEntityIdentificationCandidate
sfe_iris.h, 214
sfeIrisEntityIdentify
sfe_iris.h, 217
SFEIrisKeypoint, 87
confidence, 87
keypoint_type, 87
sfe_iris.h, 214
x, 88
y, 88
SFEIrisKeypointType
sfe_iris.h, 214, 215
SFEIrisTemplate, 88
data, 89
sfe_iris.h, 214
sfeIrisTemplateExtract
sfe_iris.h, 217
SFEIrisTemplateIdentificationCandidate
sfe_iris.h, 215
sfeIrisTemplateIdentify
sfe_iris.h, 218
sfeIrisTemplateMatch
sfe_iris.h, 219
sfeIrisTemplateQuality
sfe_iris.h, 219
SFEIrisTemplateVersion, 89

sfe_iris.h, 215
version_major, 89
version_minor, 89
sfeIrisTemplateVersion
sfe_iris.h, 219
SFEIrisTracked, 90
id, 90
sfe_iris.h, 215
state, 90
tracked, 91
uuid, 91
SFEIrisTrackedState
sfe_iris.h, 216
SFEIrisTracker
sfe_iris.h, 215
sfeIrisTrackerCreate
sfe_iris.h, 220
sfeIrisTrackerFree
sfe_iris.h, 220
sfeIrisTrackerLost
sfe_iris.h, 221
sfeIrisTrackerRemoved
sfe_iris.h, 221
sfeIrisTrackerUpdate
sfe_iris.h, 221
sfeLicenseSet
sfe_core.h, 184
SFELicenseSource
sfe_core.h, 178, 180
sfeLicenseValidate
sfe_core.h, 185
SFEOrientedBbox, 91
angle, 92
center_x, 92
center_y, 92
height, 92
sfe_core.h, 178
width, 92
SFEPalmAttributes, 93
hand_orientation, 93
hand_position, 93
quality, 93
sfe_palm.h, 227
sfePalmAttributes
sfe_palm.h, 230
SFEPalmDetect, 94
bounding_box, 94
confidence, 94
hand_orientation, 94
hand_position, 95
keypoints, 95
sfe_palm.h, 227
sfePalmDetect
sfe_palm.h, 230
SFEPalmEntity
sfe_palm.h, 227
SFEPalmEntityIdentificationCandidate
sfe_palm.h, 227
sfePalmEntityIdentify
sfe_palm.h, 230
SFEPalmKeypoint, 95
confidence, 96
sfe_palm.h, 227
x, 96
y, 96
sfePalmLivenessPassive
sfe_palm.h, 231
SFEPalmQualityAttributes, 96
brightness, 97
hotspots_score, 97
sfe_palm.h, 227
sharpness, 97
sfePalmQualityAttributes
sfe_palm.h, 232
SFEPalmTemplate, 97
data, 98
sfe_palm.h, 227
sfePalmTemplateExtract
sfe_palm.h, 232
SFEPalmTemplateIdentificationCandidate
sfe_palm.h, 228
sfePalmTemplateIdentify
sfe_palm.h, 232
sfePalmTemplateMatch
sfe_palm.h, 233
SFEPalmTemplateVersion, 98
major, 99
minor, 99
patch, 99
sfe_palm.h, 228
sfePalmTemplateVersion
sfe_palm.h, 234
SFEPalmTracked, 99
id, 100
sfe_palm.h, 228
state, 100
tracked, 100
uuid, 100
SFEPalmTrackedState
sfe_palm.h, 229
SFEPalmTracker
sfe_palm.h, 228
sfePalmTrackerCreate
sfe_palm.h, 234
sfePalmTrackerFree
sfe_palm.h, 234
sfePalmTrackerLost
sfe_palm.h, 235
sfePalmTrackerRemoved
sfe_palm.h, 235
sfePalmTrackerUpdate
sfe_palm.h, 235
SFESolver
sfe_core.h, 178
sfeSolverCreate
sfe_core.h, 185

sfeSolverCreateWithParameters
sfe_core.h, 186
sfeSolverFree
sfe_core.h, 186
SFESolverParameter, 101
name, 101
sfe_core.h, 178
value, 101
SFETattooDetect, 102
bounding_box, 102
confidence, 102
sfe_tattoo.h, 240
sfeTattooDetect
sfe_tattoo.h, 240
SFETattooEntity
sfe_tattoo.h, 240
sfeTattooEntityIdentify
sfe_tattoo.h, 241
SFETattooTemplate, 102
data, 103
sfe_tattoo.h, 240
sfeTattooTemplateExport
sfe_tattoo.h, 241
sfeTattooTemplateExtract
sfe_tattoo.h, 242
sfeTattooTemplateIdentify
sfe_tattoo.h, 242
sfeTattooTemplateImport
sfe_tattoo.h, 243
sfeTattooTemplateMatch
sfe_tattoo.h, 243
SFETattooTemplateVersion, 103
sfe_tattoo.h, 240
version, 104
sfeTattooTemplateVersion
sfe_tattoo.h, 243
SFETemplateIdentificationCandidate, 104
index, 104
score, 104
sfe_core.h, 178
sfeVersion
sfe_core.h, 187
sharpness
SFEFaceQualityAttributes, 80
SFEPalmQualityAttributes, 97
size
SFEFaceArea, 73
solver_face_detect
example_face_identify.cpp, 114
example_face_identify_entity.cpp, 128
example_face_liveness.cpp, 141
example_face_track.cpp, 153
solver_face_landmarks
example_face_identify.cpp, 114
example_face_identify_entity.cpp, 128
example_face_liveness.cpp, 141
solver_face_liveness
example_face_liveness.cpp, 141
solver_face_template
example_face_identify.cpp, 114
example_face_identify_entity.cpp, 128
solver_iris_detect
example_iris_identify.cpp, 162
solver_iris_template
example_iris_identify.cpp, 162
solver_palm_detect
example_palm_identify.cpp, 170
solver_palm_extraction
example_palm_identify.cpp, 170
SOLVER_PARAMETERS
example_face_identify.cpp, 115
example_face_identify_entity.cpp, 128
example_face_liveness.cpp, 141
example_face_track.cpp, 153
example_iris_identify.cpp, 162
example_palm_identify.cpp, 170
Solvers, 57
state
SFEFaceTracked, 83
SFEIrisTracked, 90
SFEPalmTracked, 100

tracked
SFEFaceTracked, 83
SFEIrisTracked, 91
SFEPalmTracked, 100

unique_intensity_levels
SFEFaceQualityAttributes, 80
uuid
SFEEntity, 71
SFEFaceTracked, 84
SFEIrisTracked, 91
SFEPalmTracked, 100

value
SFESolverParameter, 101
version
SFETattooTemplateVersion, 104
version_major
SFEFaceTemplateVersion, 82
SFEIrisTemplateVersion, 89
version_minor
SFEFaceTemplateVersion, 82
SFEIrisTemplateVersion, 89

width
SFEBbox, 69
SFEImage, 85
SFEOrientedBbox, 92

x
SFEBbox, 70
SFEFaceLandmarks, 79
SFEIrisKeypoint, 88
SFEPalmKeypoint, 96

y

SFEBbox, 70
SFEFaceLandmarks, 79
SFEIrisKeypoint, 88
SFEPalmKeypoint, 96
yaw
SFEFaceHeadPose, 77