Intel® RealSense™ SDK

Face Tracking Tutorial

With the Intel® RealSense™ SDK, you have access to robust, natural human-computer interaction (HCI) algorithms such as face tracking, finger tracking, gesture recognition, speech recognition and synthesis, fully textured 3D scanning and enhanced depth augmented reality.

With the SDK you can create Windows* desktop applications that offer innovative user experiences.

This tutorial shows how to enable the face tracking algorithms, namely: face location detection, landmark detection, pose detection, alert notification and expression detection. The tutorial also includes a basic rendering utility to view the output data.

Contents

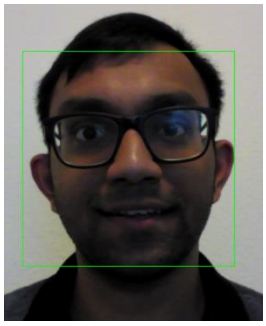
Overview

The Intel RealSense SDK contains input/output modules and algorithm modules. In this tutorial, you'll see how to use the algorithms in the face tracking module to collect and render face location and landmark detection data and to collect and display pose detection and expression detection data on your screen.

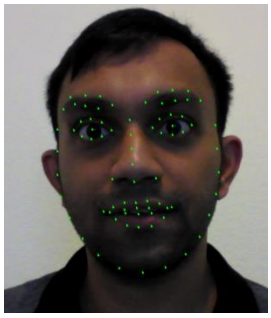
The **PXCFaceModule** interface of the SDK is the representation of the face tracking module. **PXCFaceData** interfaces abstract the face location detection, landmark detection, pose detection and expression detection data of a tracked face.

Each of the five face detection algorithms provide specific type of data:

Face Location Detection – Applications can locate a face (or multiple faces) inside a rectangle. This data can be used to count how many faces are in the current video sample and find their general locations.



Landmark Detection – This data, comprising 78 key landmark points, is often used to further identify separate features (eyes, mouth, eyebrows, etc.) of a detected face. One example usage is to determine a user's eye location in applications that change perspectives based on where the user is looking on the screen or using the feature points to create a face avatar.



Pose Detection – This data estimates the face orientation (in degrees) of a face once it is detected. Face orientation is measured three ways as: roll, pitch, and yaw (see Figure 1). Your application can use this data in many ways. For example, the perspective of the scene may change based on the user's face orientation to simulate a 3D effect.

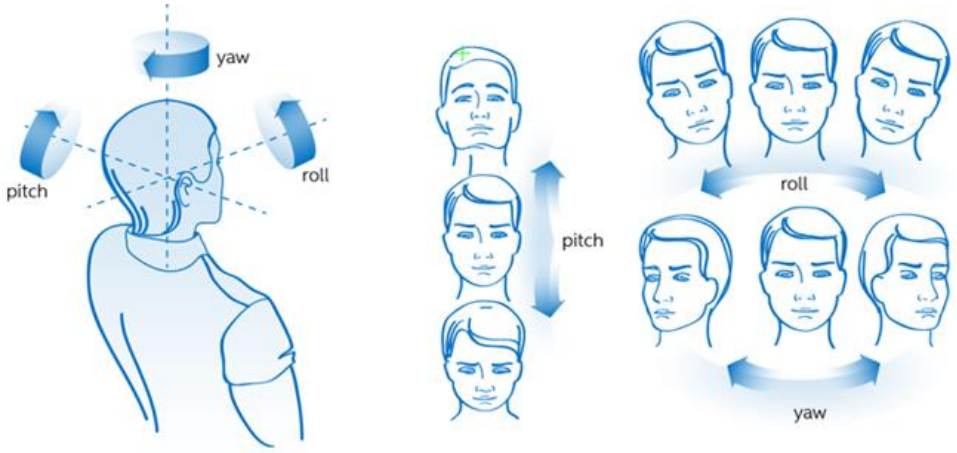


Figure 1. Head orientation measured as: roll, pitch, and yaw

Expression Detection – This module calculates the scores for a few supported expressions that are defined by eyes, mouth, eye-brows and head.

Face Recognition – This feature compares the current face with a set of reference face images in a recognition database to determine the user's identification.

Face Alert Notification– This feature helps notify the application with useful information on the face status, such as; face detected, face lost, face occluded or if face is out of camera's field of view.

Code Sample Files

You can use either procedural calls or event callbacks to capture face data, and relevant code samples showing both are listed in Table 1. Using event callbacks is usually preferred when developing console applications; procedural calls are often used for GUI applications. This tutorial's code samples use procedural calls.

The facetracking_render.cpp file, provided with the tutorial code samples, contains the **FaceRender** utility class that renders an image sample that contains face tracking data. It is provided with the tutorial so that you do not have to create your own face rendering algorithm.

Executable files (.exe) are provided in the Release subfolder in the code sample directory.

Table 1: Face Tracking Code Samples

Code Sample	**For explanation, see:**
Face tracking using procedural calls Code Sample File: main_face_tracking_procedural.cpp	This Tutorial. Also see Face Tracking using the SenseManager Procedural Functions section of the SDK Reference Manual.
Face tracking using event callbacks	Face Tracking using the SenseManager Callback Functions section of the SDK Reference Manual.

Creating a Session for Face Tracking

The SDK core is represented by two interfaces:

- **PXCSession:** manages all of the modules of the SDK
- **PXCSenseManager:** organizes a pipeline by starting, stopping, and pausing the operations of its various modalities.

The first step when creating an application that uses the Intel RealSense SDK is to create a session. A session can be created explicitly by creating an instance of **PXCSession**. Each session maintains its own pipeline that contains the I/O and algorithm modules.

Another way of creating a session is by creating an instance of the **PXCSenseManager** using **CreateInstance**. The PXCSenseManager implicitly creates a session internally.

Create an instance of **pxcStatus** used for error checking in the rest of the program.

1. Initialize an instance of the **FaceRender** utility class so that you can render the captured image samples.
2. Create a session and instance of the **PXCSenseManager** to add the functionality of the Intel RealSense SDK to your application.

```
  //SDK provided utility class used for rendering
 #include "faceanalysis_render.h"
{
        // Error checking
        pxcStatus sts;

        // initialize the UtilRender instances
        FaceRender *renderer  = new FaceRender (L"Procedural Face Tracking");
        renderer->SetMaxFaces (MAX_FACES);

        // create the PXCSenseManager
        PXCSenseManager *psm=0;
        psm = PXCSenseManager::CreateInstance();
        if (!psm) {
                wprintf_s(L"Unable to create the PXCSenseManager\n");
                return 1;
         }
}
```

Initializing the Pipeline

1. Enable the face tracking using **EnableFace**.
2. Query the face Module instance using **QueryFace** on the **PXCSenseManager** instance to configure the face tracking module.
3. Initialize the pipeline using **Init**.
4. Retrieve **PXCFaceData** instance from the face module using **CreateOutput**.
5. Retrieve an active configuration from the face module using **CreateActiveConfiguration**.
6. Set the tracking mode using **SetTrackingMode**.

You can determine if you want to enable 2D face tracking, which captures only color input data or 3D face tracking, which captures color and depth data.

```
// Enable face analysis in the multimodal pipeline:
sts = psm->EnableFace();
if (sts < PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Unable to enable Face Analysis\n");
        return 2;
}
//retrieve face Module if ready
PXCFaceModule* faceAnalyzer = psm->QueryFace();
if (!faceAnalyzer) {
        wprintf_s(L"Unable to retrieve face results\n");
        return 3;
}
// initialize the PXCSenseManager
if(psm->Init() < PXC_STATUS_NO_ERROR) return 4;

PXCFaceData* outputData = faceAnalyzer->CreateOutput();

PXCFaceConfiguration* config = faceAnalyzer->CreateActiveConfiguration();
config->
SetTrackingMode(PXCFaceConfiguration::TrackingModeType::FACE_MODE_COLOR_PLUS_DEPTH);
```

Face Detection Configuration

1. Enable the detection algorithm by setting **detection.isEnabled** to **True**.
2. Set the **maxTrackedFaces** in order to enable tracking more than one face.

```
config->detection.isEnabled = true;
config->detection.maxTrackedFaces = MAX_FACES;
```

Face Landmark Detection Configuration

1. Enable the landmark detection algorithm by setting **landmarks.isEnabled** to **true**.
2. Set the **maxTrackedFaces** in order to enable tracking more than one face.

```
config->landmarks.isEnabled = true;
config->landmarks.maxTrackedFaces = MAX_FACES;
```

Face Expression Detection Configuration

1. Enable the expression detection algorithm.
2. Enable all the expressions.
3. Set the **maxTrackedFaces** in order to enable tracking more than one face.

```
config->QueryExpressions()->Enable();
config->QueryExpressions()->EnableAllExpressions();
config->QueryExpressions()->properties.maxTrackedFaces = MAX_FACES;
```

Refer to EnableExpression to enable specific face expressions.

Face Pose Detection Configuration

1. Enable the pose detection algorithm by setting **pose.isEnabled** to **true**.
2. Set the **maxTrackedFaces** in order to enable tracking more than one face.

```
config->pose.isEnabled = true;
config->pose.maxTrackedFaces = MAX_FACES;
```

Face Alert Event Notification Configuration

Enable the alerts using **EnableAllAlerts**. Refer to EnableAlert to enable specific alerts.

```
config->EnableAllAlerts();
```

Apply the Configurations

Apply changes to the active configuration using **ApplyChanges**.

```
config->ApplyChanges();
```

Streaming the Face Tracking Algorithms

1. Acquire Frames in a loop using **AcquireFrame(true)**:
    a. TRUE to wait for all processing modules to be ready in a given frame; else
    b. FALSE whenever any of the processing modules signal.
2. In every iteration of the frame loop first **Update()** on the face data to update the face data.
3. Create the necessary data structures such as **PXCFaceData::DectectionData**, rectangle, **PXCFaceData::Landmarkdata**, **PXCFaceData::LandmarkPoint** array, **numPoints**, **PXCFaceData::ExpressionsData, PXCFaceData::ExpressionsData::FaceExpressionResult**, **PXCFaceData::PoseData, PXCFaceData::PoseEulerAngles**.
4. On the facedata **QueryNumberOfDetectedFaces** to iterate through every face detected.
5. Retrieve a face tracked using **QueryFaceByIndex** and check if it's a valid face.
6. At the end of every frame loop use **ReleaseFrame** to acquire a fresh frame in the next iteration.

```
// stream data
while (psm->AcquireFrame(true)>=PXC_STATUS_NO_ERROR)
{
        outputData->Update();

        /* Detection Structs */
        PXCFaceData::DetectionData *detectionData;
        PXCRectI32 rectangle;

        /* Landmark Structs */
        PXCFaceData::LandmarksData* landmarkData;
        PXCFaceData::LandmarkPoint* landmarkPoints;
        pxcI32 numPoints;

        /* Expression Structs */
        PXCFaceData::ExpressionsData *expressionData;
        PXCFaceData::ExpressionsData::FaceExpressionResult expressionResult;

        /* Pose Structs */
        PXCFaceData::PoseData *poseData;
        PXCFaceData::PoseEulerAngles angles;

        // iterate through hands
        pxcU16 numOfFaces = outputData->QueryNumberOfDetectedFaces();

        for(pxcU16 i = 0 ; i < numOfFaces; i++){
                // get face data by index
                PXCFaceData::Face *trackedFace = outputData->QueryFaceByIndex(i);
                if(trackedFace != NULL) {
                        // Retrieve specific face data,
                }
        }
        psm->ReleaseFrame();
}
```

Face Detection Data

1. Retrieve the detection data using **QueryDetection**.
2. Use **QueryBoundingRect** to get the tracked face rectangle dimensions.
3. Use **SetDetection** of the renderer and pass the **face index** and **rectangle** to render.

```
// for loop that iterates through detected faces
{       // if tracked face is not null
        {
                /* Query Detection Data */
                detectionData = trackedFace->detectionData();
                if(detectionData!=NULL)
                {
                        /* Get rectangle of the detected face and render */
                        if(detectionData->QueryBoundingRect(&rectangle))
                        renderer->SetDetection(i,rectangle);
                }
        }
}
```

Face Landmark Detection Data

1. Retrieve the landmark data using **QueryLandmarks**.
2. Use **QueryNumPoints** to initialize the **landmarkPoints** array.
3. Use **QueryPoints** to retrieve the 78 landmark points from the landmark data.
4. Use **SetLandmark** to pass the **face index**, the **landmark array**, and the **number of points** to render.

```
// for loop that iterates through detected faces
{       // if tracked face is not null
        {
                /* Query landmark Data */
                landmarkData = trackedFace->QueryLandmarks();
                if(landmarkData!=NULL)
                {
                        /* Get number of points from Landmark data*/
                        numPoints = landmarkData->QueryNumPoints();
                        /* Create an Array with the number of points */
                        landmarkPoints = new PXCFaceData::LandmarkPoint[numPoints];
                        /* Query Points from Landmark Data and render */
                        if(landmarkData->QueryPoints(landmarkPoints))
                                        renderer->SetLandmark(i,landmarkPoints,numPoints);
                }
        }
}
```

Face Expression Detection Data

1. Retrieve the expression data using **QueryExpressions**.
2. **QueryExpression** with the specific expression type.
3. Use **SetExpression** and pass the **face index**, **trackedFace**, **intensity**, and **expression type**.

```
// for loop that iterates through detected faces
{       // if tracked face is not null
        {
                /* Query expression Data */
                expressionData = trackedFace->QueryExpressions();
                if(expressionData!=NULL)
                {
                        /* Get expression of the detected face and render */
                        if(expressionData->
                        QueryExpression(PXCFaceData::ExpressionsData::EXPRESSION_MOUTH_OPEN,
                        &expressionResult))
                                renderer->SetExpression(i,trackedFace, expressionResult.intensity,
                                PXCFaceData::ExpressionsData::EXPRESSION_MOUTH_OPEN);

                }
        }
}
```

Face Pose Detection Data

1. Retrieve the pose data using **QueryPose**.
2. Use **QueryPoseAngles** to fill the angle data structure with **yaw**, **roll**, **and pitch** values.
3. Use **SetPose** to render and pass the **face index**, **trackedFace**, and **angles**.

```
// for loop that iterates through detected faces
{       // if tracked face is not null
        {
                /* Query Detection Data */
                poseData = trackedFace->QueryPose();
                if(poseData!=NULL)
                {
                        /* Retrieve the angles and render */
                        if(poseData->QueryPoseAngles(&angles))
                                renderer->SetPose(i,trackedFace,angles);
                }
        }
}
```

Face Alert Event Notification Data

1. Create an instance of **PXCFaceData::AlertData**.
2. Iterate through fired alerts using **QueryFiredAlertsNumber** and populate the alert data using **QueryFiredAlertData**.
3. Using **alertData.label** you can compare in a switch statement all available alerts under the **PXCFaceData** alert type.

```
// while loop; AcquireFrame
{
        // iterate through Alerts
        PXCFaceData::AlertData alertData;
        for(int i = 0 ; i <outputData->QueryFiredAlertsNumber()  ; ++i)
        {
                if(outputData->QueryFiredAlertData(i,&alertData)==PXC_STATUS_NO_ERROR)
                {
                        // Display last alert - see AlertData::Label for all available alerts
                        switch(alertData.label)
                        {
                                case PXCFaceData::AlertData::ALERT_NEW_FACE_DETECTED:
                                {
                                        wprintf_s(L"Last Alert: Face %d Detected\n", alertData.facId);
                                        break;
                                }
                                case PXCFaceData::AlertData::ALERT_FACE_LOST:
                                {
                                        wprintf_s(L"Last Alert:  Face % Lost\n", alertData.facId);
                                        break;
                                }
                                // cases for other possible face alert label types
                        }
                }
        }
}
```

Rendering the Frame

1. Create an instance of **PXCImage** to render the video stream.
2. Retrieve a sample using **QueryFaceSample**.
3. Retrieve the specific image or frame type from the sample by saving it in a variable of type **PXCImage**.

- Make sure to enable all the possible required streams in the **initializing pipeline phase**. If not enabled, the sample types will **return null**.
- Use the **RenderFrame** method provided in the custom renderer to render all the algorithms.

```
PXCImage *colorIm = NULL;

//while loop per frame
{
        // retrieve all available image samples
        PXCCapture::Sample *sample = psm->QueryFaceSample();

        // retrieve the image or frame by type
        colorIm = sample->color;

        // render the frame
        if (!renderer->RenderFrame(colorIm)) break;
}
```

Cleaning Up the Pipeline

1. Delete the FaceRender instance using **delete** at the end of the application.
2. If used, also release any instance of **PXCFaceConfiguration** with **Release**.
3. It is very important to close the last opened session, in this case the instance of **PXCSenseManager** by using **Release** in order to indicate the camera to stop streaming.

```
//In Main
{
        //while loop per frame
        {

        }
        //End of while loop per frame

        // delete the FaceRender instance
        delete renderer;

        // close the Face Configuration instance
        config->Release();

        // close the last opened stream and release any session
        //and processing module instances
        psm->Release();

        return 0;

}
//End of Main
```

Running the Code Sample

You can run these code samples two ways:

1. Build and run the **4_RawDataCapture** sample in Visual Studio*.
2. Run the executables found in the “Release” subfolder of the tutorial code sample directory.

Figure 2 shows a rendered color image frame that displays the face location detection data as a rectangle around the found face and the landmark detection data as the 78 collected landmark points, Mouth Open Expression data, and Roll, Yaw, Pitch Pose data.

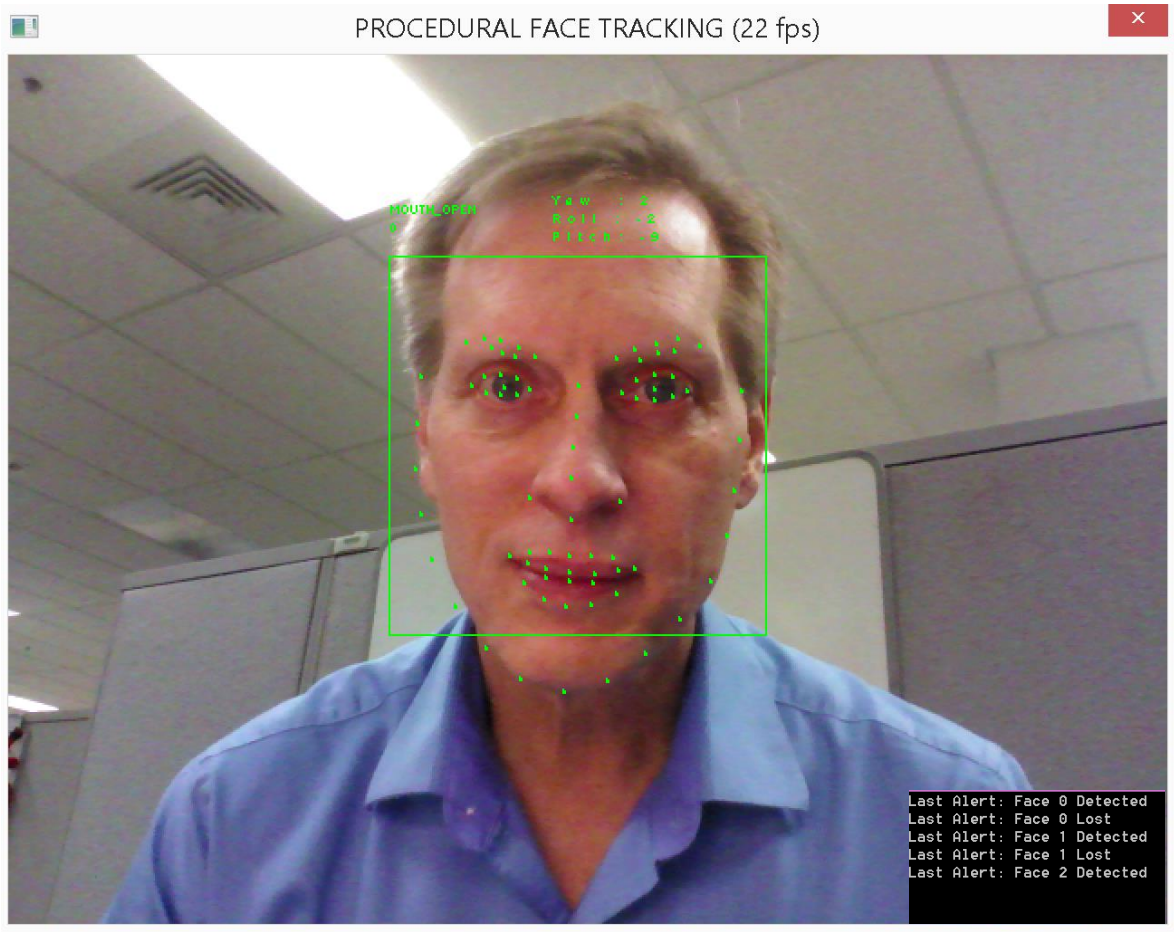


Figure 2. Rendered Color Image Sample with all the algorithms

To learn more

- The SDK Reference Manual is your complete reference guide and contains API definitions, advanced programming techniques, frameworks, and other need-to-know topics.

- To learn about Face Recognition, a feature that compares the current face with a set of reference pictures in a recognition database to determine the user's identification please to refer to Face Recognition section of the SDK Reference Manual.

- You can also tweak the device settings to get the best face tracking results by changing certain device properties:

```
PXCCapture::Device* device = psm->QueryCaptureManager()->QueryDevice();
if (device)
{
   device->SetDepthConfidenceThreshold(1);
   device->SetIVCAMMotionRangeTradeOff(21);
   device->SetIVCAMFilterOption(6);
}
```