Intel® RealSense™ SDK

Hand 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 hand tracking module of the SDK and enable its features, such as hand and finger tracking, robust 22-point skeletal joint tracking, and gesture recognition.

The module can also display event alert notifications in your application, and a basic rendering utility is provided to view the output data.

Contents

- **Overview**

    Hand Tracking Data
    Gesture Tracking
    Hand Alert Notification
    Blob Extractor
    Contour Extractor
    Data Smoothing Utility

- **Code Sample Files**
- **Creating a Session for Hand Tracking**
- **Initializing the Pipeline**

    Hand Tracking
    Gesture Tracking Configuration
    Alert Event Notification Configuration

- **Streaming the Hand Tracking Data**

    Joint data
    Gesture Data
    Alert Event Notification Data

- **Rendering the Frame**
- **Cleaning up the Pipeline**
- **Running the Code Sample**
- **To learn more**

Overview

The Intel RealSense SDK contains input/output modules and algorithm modules. In this tutorial, you'll see how to use one of the algorithm modules in particular the hand tracking module. To learn more about I/O modules, see the Capturing Raw Streams tutorial.

The hand module handles the tracking of hands in the scene, allowing you to reconstruct the 3D skeleton of the hands, recognize various hand gestures and get notifications for interesting events. The **PXCHandModule** interface is used to set up the configuration and output data of up to two hands. The **PXCHandConfiguration** can be used to set up different tracking options, enable gesture and alert notification and select the required outputs.

Hand Tracking Data

Applications can locate and return a **hand data** for up to two hands through the **PXCHandData** interface.

The hand data is exposed to the application through the **iHand** interface that provides the following data:

Mass Center: Image and world coordinates of the calculated hand image mass center.

Extremity Points: Special tracking points such as the closest, left-most, top-most, right-most, bottom-most and center points which form the boundaries of the mask of the hand silhouette.

Body Side: Whether it is a left or right hand.

Palm Orientation: Estimation of where the hand is facing.

Tracked Joints: Position and rotations of the user's hand in world and image coordinates.

Normalized Joints: Posture of user's hand without changing dimensions of the hand, i.e., the distances between each joint (bone-length) are always the same.

Finger Data: Degree of foldedness and the radius of a particular fingertip.

Segmentation Image: A mask of the hand silhouette allowing to separate a hand from the background image.

Hand Openness: (0-100 value) indicating if all fingers are completely folded to all fingers fully spread.

Gesture Tracking

Using the **PXCHandData** interface, you can also access gesture information. Certain defined hand movements, also known as hand-gestures, can be enabled for detection.

Gesture Name	Illustration	Description
spreadfingers		The hand is open facing the camera with fingers pointing upwards.
fist		Fingers folded into a fist.
tap		Hand moving towards the camera (along z-axis) as if one is pressing a button.
thumb_down		The hand is closed with the thumb finger pointing down.
thumb_up		The hand is closed with the thumb finger pointing up.
two_fingers_pinch_open		The thumb finger and the index finger touch each with vertical orientation of the hand.
v_sign		Index and middle fingers extended in upwards direction.

full_pinch	All fingers extended and touching the thumb. The pinched fingers should be pointing to the screen.
swipe	Swipe to the left with the right hand or swipe to the right with the left hand. The palm is facing the side and fingers are more or less towards the camera. The hand swiping must be inside the camera's field of view.
wave	Move the hand to the left and then the right. Repeat the action if necessary.

Hand Alert Notification

This module helps notify your application of certain useful information:

Hand detection and tracking: Inform the application that a hand is detected and is tracked.

Hand calibration: Inform the application that hand calibration data is available.

Tracking Boundaries: Inform the application when the tracked hand goes out or is about to go out of the tracking boundaries.

Blob Extractor

The blob extractor provides information about image blobs within certain depth value range. The blob extractor does not handle joint tracking or gesture tracking.

Contour Extractor

The contour extractor provides information about the contour of the image blobs. Just like the blob extractor contour extractor does not handle joint tracking or gesture tracking.

Data Smoothing Utility

The data smoothing utility can be used to reduce data noise. The utility interface enables data types for 1D, 2D, 3D data points with support for Stabilizer smoother, Weighted smoother, Quadratic smoother and Spring smoother algorithms.

Code Sample Files

You can use either procedural calls or event callbacks to capture hand 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 handtracking_render.cpp file, provided with the tutorial code samples, contains the **HandRender** utility class that renders an image sample that contains hand tracking data. It is provided with this tutorial so that you do not have to create your own hand rendering algorithm.

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

Table 1: Hand Tracking Code Samples

Code Sample	**For explanation, see:**
Hand tracking using procedural calls Code sample file: main_hand_tracking_procedural.cpp	This Tutorial. Also see Hand Tracking using the SenseManager Procedural Functions section of the SDK Reference Manual.
Hand tracking using event callbacks	Hand Tracking using the SenseManager Callback Functions section of the SDK Reference Manual.
Gesture recognition using procedural calls or events	Gesture Recognition Data section of the SDK Reference Manual.
Alert notification using procedural calls or events	Handle Alert Notification section of the SDK Reference Manual.

Creating a Session for Hand 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.

1. Initialize an instance of the **HandRender** 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.

```
#include "handanalysis_render.h"

{
        // initialize the UtilRender instances
        HandRender *renderer  = new HandRender(L"Procedural Hand Tracking");

        // 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

Hand Tracking

Create an instance of type **pxcStatus** for error checking throughout the application.

1. Enable the hand analysis using **EnableHand**.
2. Retrieve an instance of hand module using **QueryHand** on the instance of PXCSenseManager.
3. Initialize the pipeline using **Init** and retrieve an instance of handData using **CreateOutput** on the hand module.

```
// error checking Status
pxcStatus sts;

// enable hand analysis in the multimodal pipeline
sts = psm->EnableHand();
if (sts < PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Unable to enable Hand Tracking\n");
        return 2;
 }

// retrieve hand results if ready
PXCHandModule* handAnalyzer = psm->QueryHand();
if (!handAnalyzer) {
        wprintf_s(L"Unable to retrieve hand results\n");
        return 2;
}

// initialize the PXCSenseManager
if(psm->Init() < PXC_STATUS_NO_ERROR) return 3;

PXCHandData* outputData = handAnalyzer->CreateOutput();
```

Create an instance of **PXCHandConfiguration** using the hand module to configure the hand.

```
PXCHandConfiguration* config = handAnalyzer->CreateActiveConfiguration();
```

Gesture Tracking Configuration

Enable all the gestures using **EnableAllGestures** on the PXCHandConfiguration instance. You can also enable specific gestures using EnableGesture.

```
config->EnableAllGestures();
```

Alert Event Notification Configuration

Similarly, enable specific alerts using **EnableAlert**. Refer to EnableAllAlerts to enable all alerts for the hand module.

```
config->EnableAlert(PXCHandData::AlertType::ALERT_HAND_DETECTED);
config->EnableAlert(PXCHandData::AlertType::ALERT_HAND_NOT_DETECTED);
```

After configuring the hand module, apply the changes to the active configuration using **ApplyChanges**.

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

Streaming the Hand Tracking Data

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 use **Update()** on the hand data instance to update the hand data.
3. Create the required data structures for the renderer class provided in the tutorial such as the two-dimensional **PXCHandData::JointData** array.
4. At the end of every iteration of the frame loop make sure to release the acquired frame using **ReleaseFrame** to process the next frame.

```
 //Stream data
while (psm->AcquireFrame(true)>=PXC_STATUS_NO_ERROR)
 {

        //update the output data to the latest availible
        outputData->Update();

        // create data structs for storing required data
        PXCHandData::JointData nodes[2][PXCHandData::NUMBER_OF_JOINTS]={};

        //refer to upcoming sections to learn how to retrieve and render joint data
        //also gesture notifications, handSide and alerts

        // release or unlock the current frame to go fetch the next frame
        psm->ReleaseFrame();
}
```

Joint data

1. Retrieve the number of detected hands in the current frame using **QueryNumberOfHands** on the updated hand data instance.
2. Loop through the total number of detected hands.
3. Use **QueryHandData** to populate the **PXCHandData::iHand** instance**,** with appropriate access order type.
4. On a given instance of **iHand** you can loop through and **QueryTrackedJoint** with a specific **JointType** to extract individual joints.

   Use the render utility DrawJoints() provided in the tutorial to render the joints.

```
//while loop: AcquireFrame
{

	pxcU16 numOfHands = outputData->QueryNumberOfHands();
	for(pxcU16 i = 0 ; i < numOfHands ; i++)
	{
		// get hand joints by time of appearence
		PXCHandData::IHand* handData;
		if(outputData->QueryHandData(PXCHandData::ACCESS_ORDER_BY_TIME,i,handData) ==
		PXC_STATUS_NO_ERROR)
		{
			PXCHandData::JointData jointData;
			// iterate through Joints
			for(int j = 0; j < PXCHandData::NUMBER_OF_JOINTS ; j++)
			{
				handData->QueryTrackedJoint((PXCHandData::JointType)j,jointData);
				nodes[i][j] = jointData;
			}
		}
	}

	renderer->DrawJoints(nodes);
	//ReleaseFrame
}
```

Gesture Data

1. Iterate through all the fired gestures using **QueryFiredGesturesNumber** and populate the gesture data using **QueryFiredGestureData**.
2. Access an **iHand** instance associated with a gesture using **QueryHandDataById** on a hand data instance.
3. Use **QueryBodySide** on the retrieved iHand instance to detect whether the gesture was fired by a right or left hand.

Use the render utility NotifyGestures() provided in the tutorial to display the gestures.

```
//while loop: AcquireFrame
{

	// iterate through fired gestures
	int numOfGestures = outputData->QueryFiredGesturesNumber();
	if(numOfGestures>0){
		// iterate through fired gestures
		for(int i = 0; i < numOfGestures; i++){
			// initialize data
			PXCHandData::GestureData gestureData;
			// get fired gesture data
			if(outputData->QueryFiredGestureData(i,gestureData) ==
			PXC_STATUS_NO_ERROR){
				// get hand data related to fired gesture
				PXCHandData::IHand* handData;
				if(outputData->QueryHandDataById(gestureData.handId,handData) ==
			PXC_STATUS_NO_ERROR){
					// save gesture only if you know that its right/left hand
					if(handData->QueryBodySide() !=
					PXCHandData::BodySideType::BODY_SIDE_UNKNOWN)
						renderer->NotifyGestures(handData-
						>QueryBodySide(),gestureData.name);
					}

				}
			}
		}
	//ReleaseFrame

}
```

Alert Event Notification Data

1. Iterate through fired alerts in the current frame using **QueryFiredAlertsNumber** and populate the alert data using **QueryFiredAlertData**.
2. Using **alerData.label** you can compare in a switch statement all available alerts under the **PXCFaceData** alert type.

Use the render utility NotifyAlerts() provided in the tutorial to display the alerts.

```
//while loop: AcquireFrame
{
        // iterate through Alerts
        PXCHandData::AlertData alertData;
        for(int i = 0 ; i <outputData->QueryFiredAlertsNumber(); i++)
        {
                sts = outputData->QueryFiredAlertData(i,alertData);
                if(sts==PXC_STATUS_NO_ERROR)
                {
                        // Display last alert - see AlertData::Label for all available alerts
                        renderer->NotifyAlerts(alertData.label);

                }
        }//ReleaseFrame

}
```

Rendering the Frame

1. Create an image instance for the depth sample used to stream the data.
2. Retrieve a hand sample using **QueryHandSample**.
3. Retrieve the specific frame type from the sample, by saving it in a variable of type **PXCImage**.

    - Use the **RenderFrame** method provided in the custom renderer to render the depth image as the background.

```
PXCImage *depthim = NULL;
//while loop per frame
{

        // retrieve all available image samples
        PXCCapture::Sample *sample = psm->QueryHandSample();

        // retrieve the image or frame by type
        depthim = sample->depth;

        // render the frame
        if (!renderer->RenderFrame(depthIm)) break;

        //ReleaseFrame

}
```

Cleaning up the Pipeline

1. Make sure to delete the **HandRenderer** instance using **delete**.
2. If used, also release any instance of **PXCHandConfiguration** using **Release** on it.
3. It is very important to close the last opened session, in this case the instance of **PXCSenseManager** by using **Release** to indicate the camera to stop streaming.

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

	}
	//End of while loop per frame

	// delete the HandRender instance
	delete renderer;

	// close the Hand 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 the code sample two ways:

1. Build and Run the **3_Hands_Tracking sample** in Visual Studio*.
2. Run the executables in the "Release" subfolder of the tutorial code sample directory.

Figure 1 shows a rendered depth image frame that displays the hand tracking data as the 22 tracked joints, Alert data, and Gesture Data along with the Body side of each gesture.

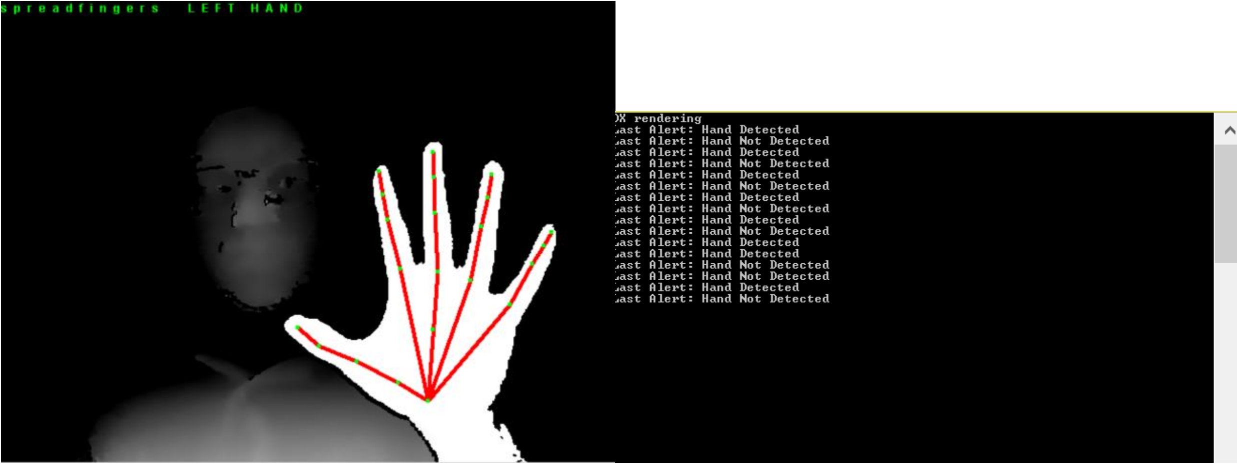

Figure 1. Rendered Depth Sample with Hand Tracking data, Alert data, and Gesture data

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.
- The Intel RealSense SDK allows the hand tracking module to adapt to a user's hand over time. Check out how to save **hand calibration data** for specific users in the Hand Calibration Data section of the SDK Reference Manual.
- For more ways to capture Hand Data refer to its member functions.