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.

In this tutorial, you’ll learn how to use the SDK to detect and track known as well as unknown objects in a video sequence or scene. Specifically, 2D object tracking, feature-based 3D tracking, edge-based 3D tracking from CAD models, and instant 3D tracking using the SDK.
Contents

- Overview
  2D Image Tracking
  Feature-based 3D Maps
  Edge-Tracking
  Instant 3D Tracking
- Code Sample Files
- Creating a Session
  Initializing 2D-Image Tracking
  Initializing Feature-based 3D Maps
  Capturing the Tracking Values
  Cleaning Up the Pipeline
  Running the Code Samples
- To learn more
Overview

This tutorial shows you how to detect and track objects using various techniques. It also explains how to capture 3D Map data to track an object using the feature-based 3D Map object tracking. Every algorithm returns 3D tracking parameters such as translation and rotation in the form of TrackingValues data structure.

The four kinds of tracking techniques are:

2D Image Tracking

This algorithm tracks using a reference image. The algorithm tracks a given image in a video sequence or scene and returns the 3D tracking parameters.

Feature-based 3D Maps

This algorithm can track any real-world 3D object. The tracking is based on a 3D feature map "slam" file that can be generated using the Metaio Toolbox application located in RSSDK/contrib/Metaio directory.

Edge-Tracking

This algorithm can better track objects with features that are hard to detect. Specifically, objects with low-texture or highly specular (reflective) properties or those changing their appearances over time (e.g., a color change after a building has been painted) or those in different/dynamic lighting conditions. The algorithm uses a 3D CAD model, mesh model, or 3D-point cloud target as input.

The Metaio Toolbox application located in the RSSDK/contrib/Metaio directory processes the 3D model to extract edge features and create an "xml" 3D map as input for tracking.

Please refer to the Edge Tracking section of the SDK Reference Manual for its implementation.
**Instant 3D Tracking**

This algorithm enables you to create a point cloud (3d map) of a scene on the fly and immediately use it as a tracking reference. The user doesn't need any input map file or image to track objects. The algorithm learns the surroundings on-the-fly and tracks objects in a scene automatically and in real time.

Please refer to the [3D Instant Tracking](#) section of the SDK Reference Manual for its implementation.

**Code Sample Files**

You can use either procedural calls (used in this tutorial) or event callbacks to implement object tracking techniques, and code samples are provided for both (see Table 1). Using event callbacks is usually preferred when developing console applications; procedural calls are often used for GUI applications.

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

<table>
<thead>
<tr>
<th>Code Sample</th>
<th>For more information, see:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Tracking using procedural calls Files: main_2DObjectTracking.cpp, main_FeatureBased3DTracking.cpp</td>
<td>This Tutorial. Also see <a href="#">Object Tracking Using the SenseManager Procedural Functions</a> section in the SDK Reference Manual.</td>
</tr>
</tbody>
</table>
Creating a Session

The first step when creating an application that uses the Intel RealSense SDK is to create a session and an instance of the `PXCSenseManager` using `CreateInstance`.

1. Enable the object tracking module using `EnableTracker`.
2. Retrieve an instance of the object tracking module using `QueryTracker`.
3. Initialize the pipeline using `Init`.

```cpp
#include <pxcsensemanager.h>
#include "pxctracker.h"

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

// Enable the tracking module in the multimodal pipeline
sts = psm->EnableTracker();
if (sts < PXC_STATUS_NO_ERROR) {
    wprintf_s(L"Unable to enable tracking module\n");
    return 2;
}

//retrieve tracking module if ready
PXCTracker *trackingAnalyzer = psm->QueryTracker();
if (!trackingAnalyzer) {
    wprintf_s(L"Unable to retrieve tracker\n");
    return 3;
}

// initialize the PXCSenseManager pipeline
if(psm->Init() < PXC_STATUS_NO_ERROR) return 4;
```
Initializing 2D-Image Tracking

1. Initialize an instance of **Tracking2DRenderer** utility class provided in the tutorial.
2. Allocate a **filepath**, the location of the image of the object to track.
3. Allocate a **cosID** (object identifier) that can be associated with the particular image **filepath**.
4. The SDK will assign a particular **cosID** to the image **filepath**.
5. Use **Set2DTrackFromFile** function to set the tracking configuration to 2D-image tracking and retrieve the SDK assigned **cosID** to use an identifier.

A sample image is provided in the **Resources** directory for tracking.

```c
// initialize the Tracking2DRenderer
Tracking2DRenderer *renderer = new Tracking2DRenderer(L"2D OBJECT TRACKING");

// allocate a cosID and relate it to the path of the file
pxcCHAR *filename = L"Resources/Intel.png";
pxcCHAR filepath[1024];
pxcUID cosID;
wcsncpy_s<1024>(filepath, filename, 1024);

// set the kind of tracking configuration - 2D Object Tracking and let the SDK assign a cosID
sts = trackingAnalyzer->Set2DTrackFromFile(filepath, cosID);
if (sts < PXC_STATUS_NO_ERROR) {
    wprintf_s(L"Failed to set tracking configuration\n");
    return 5;
}
```

For tracking multiple objects simultaneously you can have multiple **cosID**'s associated with multiple file paths.
---

## Initializing Feature-based 3D Maps

Make sure to export the *.slam file of an object to track using the Metaio Toolbox provided in the SDK and place it inside Resources directory in order to successfully run the code sample. Also replace the filename correspondingly e.g. “Resources/myfilename.slam”.

1. Initialize an instance of **Feature3DTrackingRenderer** utility class provided in the tutorial.
2. Allocate a **filepath**, the location of the (.slam) file, which is the 3D map data of the object to track.
3. Allocate a **cosID, firstID, lastID** (object identifiers) that can be associated with the particular **filepath**.
4. Set the feature-based 3D tracking algorithm for a particular **filepath** using the **Set3Dtrack** function.

If there is more than one reference, the function returns the first and last object identifiers of the added reference objects. If not, use only the **firstID** as the primary object identifier by assigning it to **cosID**.

```c++
// initialize the Feature3DTrackingRenderer
Feature3DTrackingRenderer *renderer = new Feature3DTrackingRenderer (L"3D OBJECT TRACKING");

// allocate a cosID and relate it to the path of the file
pxcCHAR *filename = L"Resources/UNKNOWN";
pxcCHAR filepath[1024];
pxcUID firstID, lastID;
pxcUID cosID;
wcsncpy_s(filepath, filename, 1024);

// set the kind of tracking configuration - 3D Object Tracking and let the SDK assign a cosID
sts = trackingAnalyzer->Set3DTrack(filepath, firstID, lastID);
cosID = firstID;

if (sts < PXC_STATUS_NO_ERROR) {
    wprintf_s(L"Failed to set tracking configuration\n");
    return 5;
}
```

---

**Intel® RealSense™ SDK Object Tracking Tutorial**
Capturing the Tracking Values

1. Create a loop to process the data.
2. In every iteration of the loop, first use the `AcquireFrame (true)` function:
   a. TRUE (aligned) to wait for all modules to be ready in a given frame; else
   b. FALSE (unaligned) whenever any of the processing modules signal
3. Loop through all the detected objects in a given frame using `QueryNumberTrackingValues`.
4. Query and retrieve the `TrackingValues` corresponding to a `cosID` using `QueryTrackingValues`.
5. Use the quality parameter of the `TrackingValue` to set the threshold to decide the whether to consider a tracked object for rendering based on the quality of tracking.
6. Use `DrawTrackingValues` to render the tracked data provided in the OpenGLRenderer class.
7. Use `QueryTrackerSample` to retrieve the tracker image sample and render the specific color image of the sample using `updateFrame`.
8. Release the frame to process the next frame through the `ReleaseFrame` function.

```c
while (psm->AcquireFrame(true) >= PXC_STATUS_NO_ERROR) {
    // loop through all detected objects
    for (pxcI32 i = 0; i < trackingAnalyzer->QueryNumberTrackingValues(); i++) {
        // query and retrieve the trackedData corresponding to the cosID
        PXCTracker::TrackingValues trackedData;
        trackingAnalyzer->QueryTrackingValues(cosID, trackedData);

        // threshold beyond which one can consider trackedData as valid
        if (trackedData.quality > 0) {
            renderer->DrawTrackingValues(&trackedData);
        }
    }
    // retrieve available tracker image sample
    const PXCCapture::Sample *sample = psm->QueryTrackerSample();
    // get the color data
    image = sample->color;
    // render the frame
    if (!renderer->RenderFrame(image)) break;
    // release or unlock the current frame to fetch the next frame
    psm->ReleaseFrame();
}
```
Cleaning Up the Pipeline

After your application is done tracking and rendering, you must “clean up”:

1. Delete the render instance using `delete`.
2. Release any session and processing module instances using `Release()` on the `PXCSenseManager` instance.

```cpp
// delete the Renderer instance
delete renderer;

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

Now you have all the information to use the object tracking techniques provided in the Intel® RealSense™ SDK.
Running the Code Samples

You can run these code samples two ways:

1. Build and run the **2DObjectTracking**, **FeatureBased3DTracking** samples in Visual Studio*.  
2. Run the executables found in the “Release” subfolder of the tutorial code sample directory.

Figures 1 and 2 show the 2DObjectTracking and FeatureBased3DTracking output from the tutorial code sample.

![Figure 1. 2D Object Tracking](image1.png)

![Figure 2. FeatureBased3DTracking](image2.png)

Figures 3 and 4 shows the use of Metaio Tracker Toolbox to capture the “.slam” data file

![Figure 3. Setting up Metaio Tracker Toolbox](image3.png)

![Figure 4. Saving Tracked (.slam) file using Metaio Tracker Toolbox](image4.png)
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.