An Introduction to Intel® RealSense™ Visual SLAM and the T265 Tracking Camera

Version 1.0

Anders Grunnet-Jepsen, Michael Harville, Brian Fulkerson, Daniel Piro, Shirit Brook, Jim Radford

1. Introduction

Location awareness is arguably one of the most fundamental and important senses needed to survive and navigate in a 3D world. The ability to move around, forage, and to return to one's home or nest is ingrained in all animals, so the challenge in robotics becomes "How do we mimic and possibly even enhance this innate ability *electronically*?" In Computer Science, this field of study is commonly referred to as "Simultaneous Localization and Mapping" or SLAM. It encompasses the study of instruments and algorithms that try to solve for an agent's (aka robot or autonomous vehicle) location and orientation in 3D space, while simultaneously creating and updating a "mental" map of the unknown surrounding environment. Sometimes this is also referred to as "inside-out tracking" of the environment to determine the agent's own movement, to distinguish it from "outside-in" motion capture (aka mo-cap) or object tracking achieved with arrays of stationary cameras surrounding the object to be tracked. One of the most important aspects of SLAM for many applications, is the ability to fully track all 6-degrees of freedom (6DoF), meaning one's own spatial location (X, Y, Z) as well as angular orientation (pitch, yaw, roll).

Mature solutions do exist in the market today for some constrained mapping problems. For example, Global Positioning Systems (GPS) and assisted navigation are now so wide spread that practically every cell-phone has a receiver embedded, essentially obsoleting the need to learn how to read and navigate with maps. However, the accuracy of GPS is currently limited to roughly 10m. Moreover, the update rate is fairly slow at ~10Hz, they only work well outdoors where they are able to receive electronic triangulation signals from GPS satellites, and they suffer from multi-path interference due to reflection from mountains and buildings. Another sensor that has seen truly amazing development in recent years is the inertial measurement unit, or IMU. This consists of a combination of gyroscopes and accelerometers, usually built with tiny integrated MEMs (micro-electromechanical systems) components, able to quickly measure changes in its own orientation and acceleration. These have also now made their way into essentially every cell phone and can be purchased individually for about $1. On the other range of the price, power and size spectrum, Light Detection and Ranging systems (LiDARs) have become very popular in the race to develop fully autonomous cars. LiDARs usually consist of spinning/scanning laser beams that employ short optical pulses or frequency modulation in order to measure the distance to surrounding object with millimeter scale accuracy.

The technology we will focus on here is called Visual Inertial Odometry (VIO), and it is really the closest electronic equivalent to how most animals sense the world – using CMOS sensors to act as eyes to see the surrounding environment; an IMU that acts as the inner ear to sense balance and orientation; and compute to act as the brain that fuses the information into instantaneous location and mapping. VIO has become one of the most promising SLAM technologies to emerge in recent years, and algorithms and variants continue to evolve and improve at a high pace. For example, today most augmented/virtual reality headsets (including the Microsoft HoloLens, Oculus Quest, or Magic Leap One) and cell phones (Apple's AR Kit and Google's AR Core) utilize their own versions of VIO to accurately and with low latency track the full 6DoF of the device. Moreover, VIO is now starting to see more wide-spread adoption in robotics as well. In contrast to the incumbent Lidar solutions, VIO systems are significantly smaller, cheaper, and lower power, and they offer potential for superior *re-localization*, by recognizing and re-centering their location on a large map based on visual features, and not just ranging and geometry.

In the next sections we will explore more how VIO works, and we will introduce the Intel RealSense Tracking Camera T265 and Tracking module T261 that are intended to serve as complete turn-key embedded SLAM sensor solution for use in AR/VR headsets, Robotics and Drones. We will explain some key SLAM system performances metrics, and we will show how easy it is to get started with the T265 camera using the Intel RealSense Viewer, and interfacing to it programmatically using the open source Intel RealSense SDK.   This whitepaper is meant to serve as a quick introduction and overview, leaving more in-depth treatment to other technical documents.

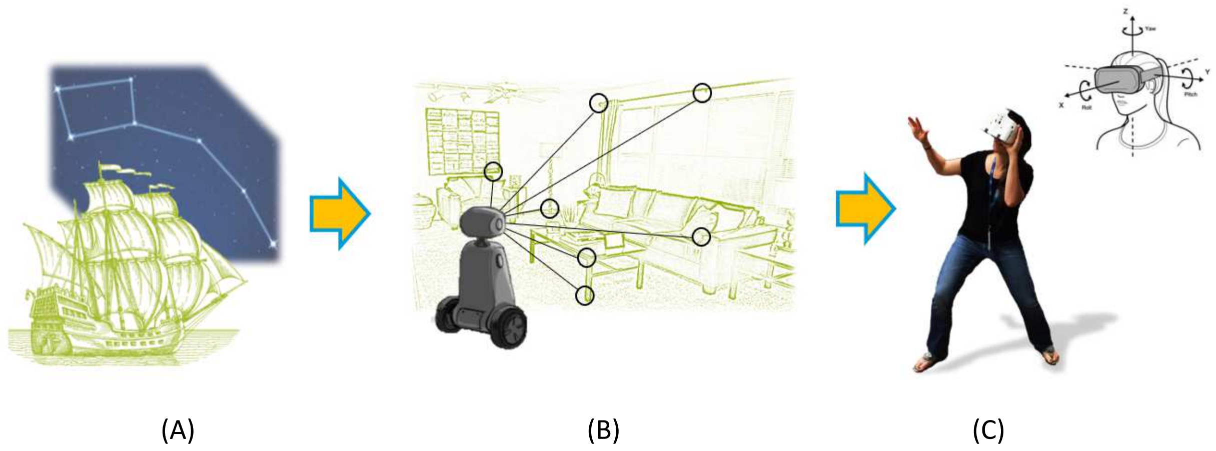


Fig. 1. Skilled sailors used to navigate according to the stars (A). Visual Inertial Odometry tries to accomplish low-latency, high-speed, robust and accurate full 6DOF tracking that can be used for robots (B) and for AR/VR headsets (C)

2. Beginner's guide to Visual-Inertial-Odometry

The best way to start understanding visual-inertial-odometry is to first examine the "visual" part. The core concept of operation harkens back to pre-compass and pre-GPS days, when humans would navigate according to the stars, as shown in Figure 1A. By *recognizing* special constellations or specific stars, people could find North (i.e. direction) as well as calculate their own latitude and longitude (i.e. location), usually using look-up tables.

In Computer Vision, we strive to accomplish the same task in the fastest and most robust way. We start by taking an electronics picture of our surroundings. This will typically consist of millions of pixels, so we use a "feature detection algorithm" to immediately condense the informational content to the extent that we essentially are left with only a *few hundred named points*. In other words, the image is now replaced with a black picture with a few white points that each have an identity, just like looking at a sky at night. So, what is a "feature"? A feature could be something as simple as saying "find all corners in the image". In reality, this is an extremely active area of research, and feature detection algorithms abound with names like SIFT, LIFT, Harris Corner, Shi-Tomasi corner, ORB, BLOB, SURF, KAZE, FAST and many more. Once the features are detected, they all have to be "named" by an algorithm that gives each feature a "descriptor" or "feature vector". We repeat this process for each new image frame that is received and compare and track features in consecutive frames (or certain key frames) using a *feature tracking* algorithm. We emphasize that each feature is actually mapped to 3D space, and not just in 2 dimensions. Figure 2 shows an example of an image with detected features.

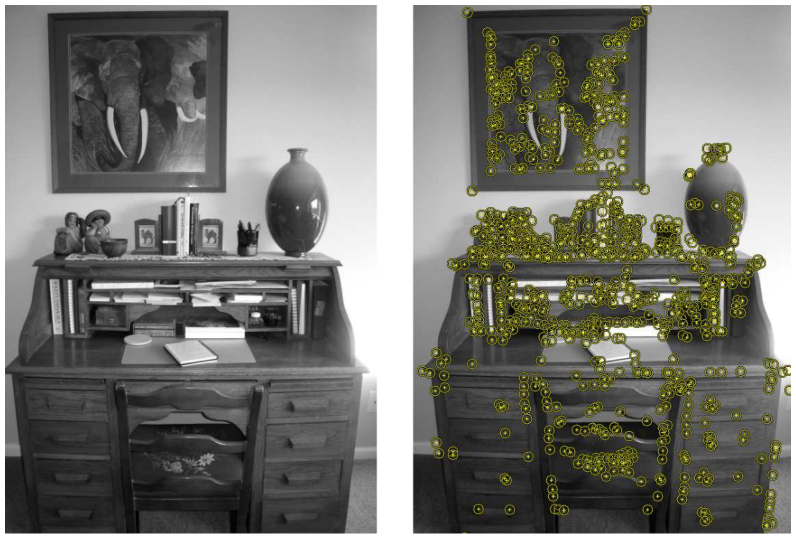


Fig. 2. Example of "Harris Corner Features" detected in an image. Reproduced from ref. 1.

This all works amazingly well, but state-of-the-art solutions are constantly evolving to improve performance. For example, one of the issues with using a single camera (aka. Mono-SLAM) is that it does not solve for scale very well. This means that if the camera is moved 1m forward, it does not have a way of relating the observed movement to 1m, as it could just as well be 1cm. The two most common ways to solve this problem is to either use calibrated stereo cameras, or an IMU. Another significant challenge is being able to achieve high frame rates and low latency tracking. While it is certainly possible to operate camera sensors at higher frame rates, the algorithms are actually extremely computational expensive (meaning compute and power hungry) and the faster you run a camera the less light it can get. It turns out that a better engineered approach is to use a high-speed IMU that is surprisingly good at tracking speed and orientation for *short periods* of time (within 10s of milliseconds), but suffers from serious drift if operated over longer periods of time, like several seconds. The addition of the IMU is actually the reason for the "I" ("Inertial") in VIO. Table 1 gives a good summary of how Visual and IMU odometry complement each other nicely. Furthermore, if the sensor is connected to a vehicle that provides odometry readings (like speed), this can also be used to enhance the tracking accuracy. Finally, since the main premise of the tracking is being able to see features, it stands to reason that most systems try to use optical sensors that have a very wide field-of-view so they can see as many features as possible. This is especially useful when the sensor is moved close to a white wall, that inherently has no features. With a fisheye lens, it should still be possible to track features in the periphery of the vision even when moving very close to a featureless object.

	Visual Odometry	IMU Odometry
Update Frequency	Low	High
Stability over Time	High	Low
Scene Dependent	Yes	No
Re-localization	Yes	No
CPU Utilization	High	Low

Table 1. Visual and IMU odometry techniques complement each other. In particular, IMU data is available at higher frequency than the visual odometry. On the other hand, IMU data alone suffers from high amount of aggregated drift over time, something that can be rectified via infrequent updates from visual odometry. Moreover, visual odometry relies on presence of features in the scene, while IMU is relatively independent of the environment.

In summary, the best solutions today tend to combine information from multiple fisheye imagers with sensor readings from an IMU and/or vehicle's odometry if available. It will then fuse all the information together in real time and track the sensors own 6DoF with extremely low latency. Figure 1B and 1C are examples of using such an embedded sensor for ego-centric tracking of 6DOF for a robot and AR/VR headset respectively.

Fig. 3. The Intel RealSense Tracking Camera T265 is a complete embedded SLAM solution that use Visual Inertial Odometry (VIO) to track its own orientation and location (6DoF) in 3D space.

The Intel RealSense Tracking Camera T265, shown in Figure 3, is a complete stand-alone solution that leverages state-of-the-art algorithms to output 6DoF tracking based on VIO. The hardware design includes a set of wide Field-of-View stereo fisheye cameras with ~165 degree circular FOV captured by ~800 pixel diameter monochrome global shutter cameras. It also incorporates a hardware synched Bosch BMI055 200Hz Gyro and 62.5Hz accelerometer and a powerful Intel® Movidius™ Myriad™ 2 Vision Processing Unit (VPU). The embedded processor runs the entire SLAM algorithm onboard, analyzes the stereo images and fuses all sensor information together into 6DoF tracking, all while using less than 1.5W of power. The camera pose information is provided over USB at a rate of 200Hz and can easily be interfaced to almost any host platform using the Open Source Intel RealSense SDK, as detailed in Section 4.

This device is intended for customers looking for a hassle-free stand-alone SLAM system. For those who want to develop or deploy their own tracking algorithms, they can obtain the raw data (cameras and IMU) from the T265, or build on top of our Intel RealSense Depth Camera D400 family which provide similar sensor capabilities (albeit with smaller FOV) and also output depth maps. However, note that any SLAM algorithm would need to be run on the host platform and not embedded and this will increase the weight, power budget and cost of the target platform.

3. Performance Metrics

To benchmark the performance of 6DoF trackers the most widely used key performance indicators (KPIs) are Drift and Floor Position Error. The drift is the accumulated error in an 'unknown' environment when re-localization is not activated. It is calculated as the percentage of error per the length of the path that was traveled. The Floor Position Error is the root-mean-square error of the floor report position compared to the actual position in a known environment, when re-localization is active. It is measured in cm.

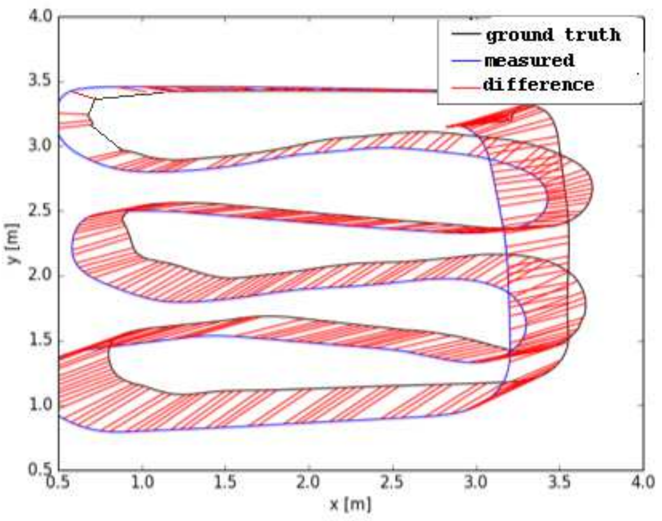


Fig. 4. Example of SLAM motion trajectory errors, showing ground truth and estimated location (here in 2D). KPIs like floor position error and drift can be measured from these types of recordings.

One extremely important property of a good SLAM system is its ability to *re-localize* to a known or pre-generated map. This is sometimes referred to as solving the “kidnapped robot” problem. Basically, when a robot is initially powered on, or if it loses power, or if its view is obstructed for long periods of time during motion, it is possible that the sensor will lose track of where it is. If it has an internal map, the relocalization error describes the ability of a system to recognize previous environments and reposition itself on that map. This is especially important to tracking motion on large scale or through multiple rooms. A good system relocalizes with centimeter absolute accuracy and has the option to relocalize frequently and on a predefined cadence. Another very important aspect of feature map generation, is whether the maps can be shared across multiple agents. For example, it would be desirable to have that any robot or autonomous vehicle could benefit from the fact that another robot has already mapped out that area. For this to happen, each robot must be able to not only capture, but also export and share their maps, and more importantly, the detected features should look the same on each robot. The T265 allows for this cooperative mapping.

Two additional performance criteria are host CPU utilization and memory usage. Since the T265 SLAM algorithm runs on the device itself (i.e. on an embedded vision processor), it uses very little of both of these, leaving space and compute power for applications on the host, even when running on weak platforms.

Please refer to the T265 datasheet [ref 2] for the target KPIs.

4. Getting Started with T265

4.1 Calibration & Initialization

The T265 Tracking Camera has been calibrated at the factory and these parameters can be obtained from the Intel RealSense SDK. Some of these parameters can change when the camera is started, and so the algorithm inside the T265 estimates them on startup. It assumes the device is not moving when it is started and will only return “high” confidence once the device has been moved sufficiently to estimate its internal parameters. Both very slow and very fast movements hinder this initialization and should be avoided.

4.1a Mounting considerations

The device should be rigidly attached to the target platform, two M3 screw holes are provided on the back of the case to make this easier. In applications where the sensor might experience high vibration (such as a drone) or temporary shocks (such as a robot), it is best to attach the T265 to a vibration dampened mount.

The T265 also provides M2 screw holes to firmly attach the micro USB 3 cable, and these should be used if possible to prevent USB disconnections.

4.1b Coordinate systems

The pose provided by the T265 is provided relative to the center of the two cameras, as shown in Figure 4b. The pose is then the transformation from this reference frame to a gravity aligned reference frame centered on the starting position, with Y pointing up, -Z pointing forward relative to the initial camera axis, and X pointing to the right (see Figure 4c).

In an application, you usually need to know the pose of something other than the T265. For instance, in a VR headset, you may want to know the position of the center of rendering. The transformation between the T265 and the target coordinate system should be carefully measured and applied.

The application may also wish to know the position at some time in the future (for instance, at the time when the next frame of a VR scene is planned to be rendered). The Intel RealSense SDK provides an example called **pose-predict** which shows how to use the T265's velocity and acceleration estimates to predict the position of the device in the future.

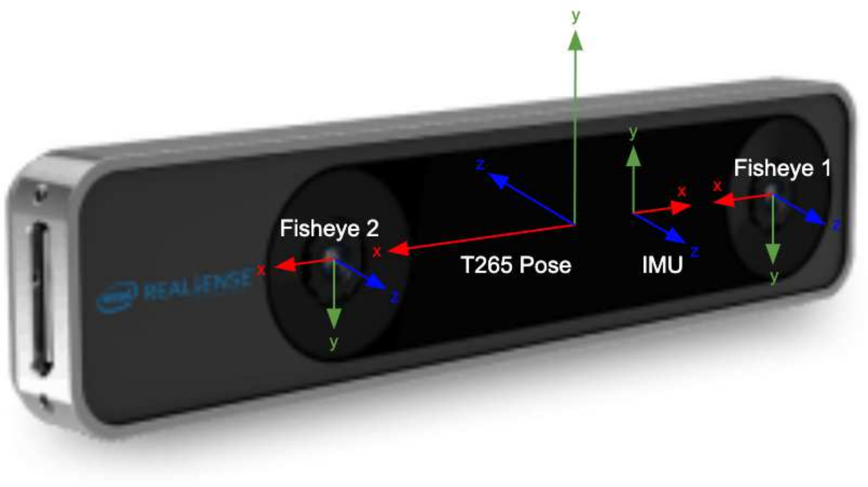


Fig. 4b. Coordinate systems of the T265. The exact transformations between the sensors can be obtained from the Intel RealSense SDK.

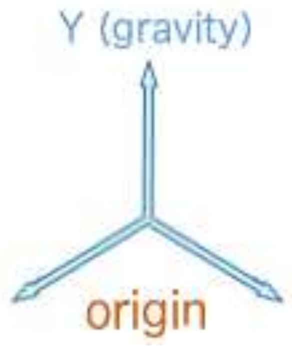


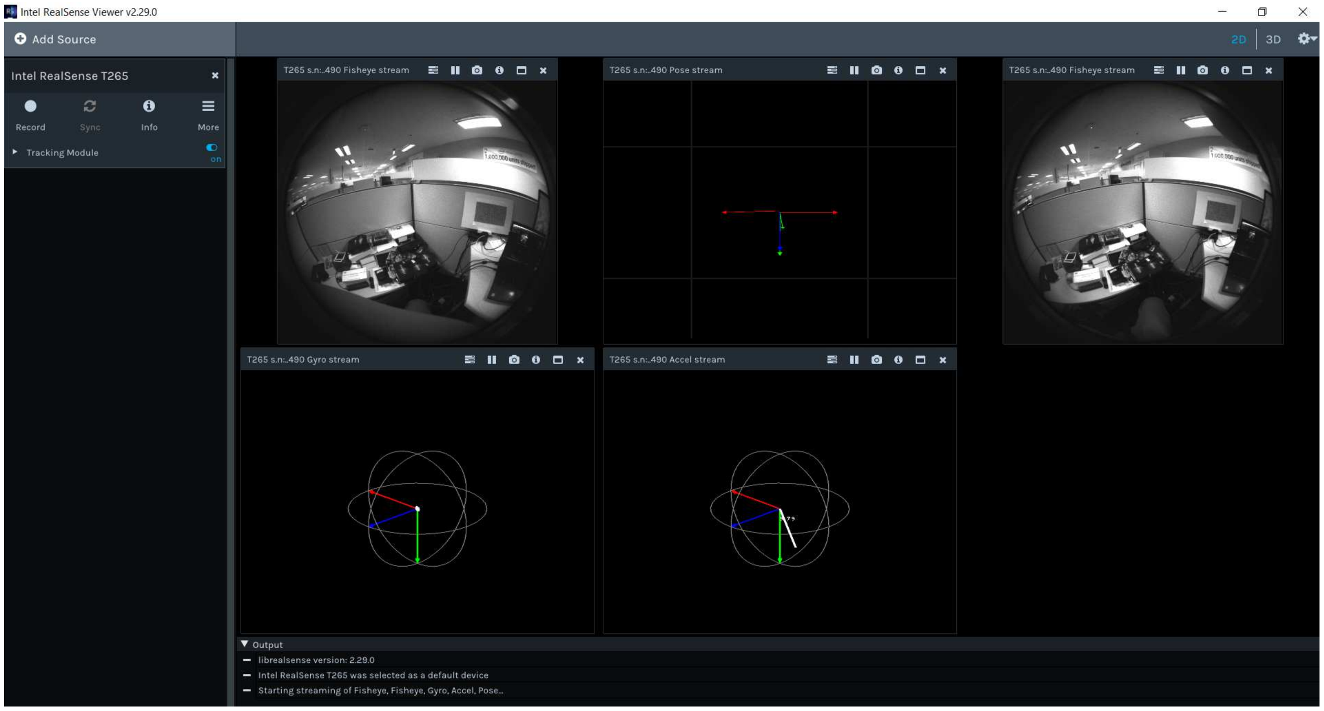
Fig. 4c. The origin coordinate system for T265.

4.1c Timestamps

The T265 reports the time for all sensors in RS2_TIMESTAMP_DOMAIN_GLOBAL_TIME. This uses a time synchronization mechanism between the host computer and the T265 to provide all sensor data timestamps in millisecond accurate host clock time.

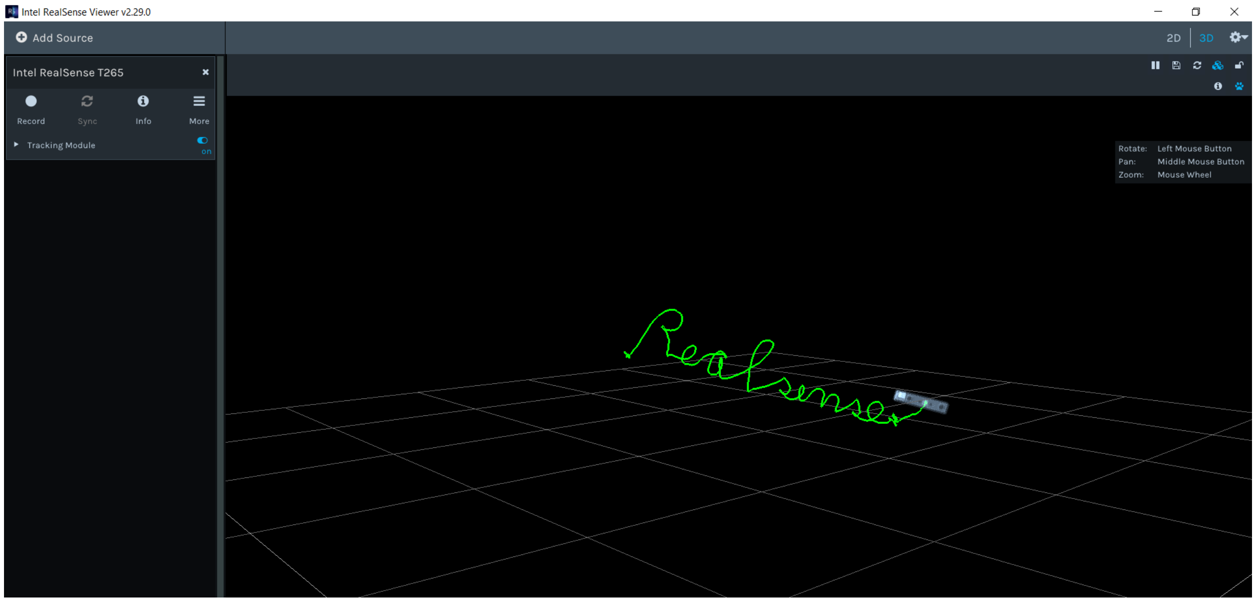
4.2 Intel RealSense Viewer

The best way to get familiar with the T265 Tracking Cameras is to run the Intel® RealSense™ Viewer application which is included in the Intel® RealSense™ SDK 2.0. This application visualizes the pose output via a graphical user interface. Please refer to https://www.intelrealsense.com/tracking-camera-t265/ and T265 on GitHub for further information and code examples.


Intel RealSense Viewer v2.29.0
Add Source
2D
3D
Intel RealSense T265
Record
Sync
Info
More
Tracking Module
on
T265 s.n:..490 Fisheye stream
T265 s.n:..490 Pose stream
T265 s.n:..490 Fisheye stream
T265 s.n:..490 Gyro stream
T265 s.n:..490 Accel stream
Output

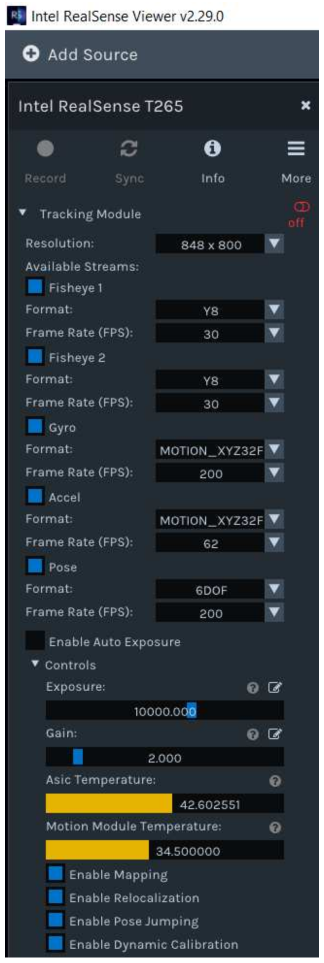
```
librealsense version: 2.29.0
Intel RealSense T265 was selected as a default device
Starting streaming of Fisheye, Fisheye, Gyro, Accel, Pose...
```

Fig. 5A. The Intel RealSense Viewer, shown here, is a great starting point for getting familiar with the Intel RealSense Tracking Camera T265. In the "2D" view, the different raw sensor inputs are visualized.



```
Rotate:   Left Mouse Button
Pan:      Middle Mouse Button
Zoom:     Mouse Wheel
```

Fig. 5B. In the Intel RealSense Viewer, switching to the 3D view visualizes the pose via a graphical user interface. This point-cloud view can be rotated in 3D.


Intel RealSense Viewer v2.29.0

Add Source

Intel RealSense T265

Tracking Module
Resolution: 848 x 800
Available Streams:
Fisheye 1
Format: Y8
Frame Rate (FPS): 30
Fisheye 2
Format: Y8
Frame Rate (FPS): 30
Gyro
Format: MOTION_XYZ32F
Frame Rate (FPS): 200
Accel
Format: MOTION_XYZ32F
Frame Rate (FPS): 62
Pose
Format: 6DOF
Frame Rate (FPS): 200
Enable Auto Exposure
Controls
Exposure: 10000.000
Gain: 2.000
Asic Temperature: 42.602551
Motion Module Temperature: 34.500000
Enable Mapping
Enable Relocalization
Enable Pose Jumping
Enable Dynamic Calibration

Fig. 5C. In the Intel RealSense Viewer it is possible to adjust parameters to see their effect. Furthermore, relocalization and mapping can be enabled.

4.3 Intel RealSense SDK API

The primary software interface to the Intel® RealSense™ Tracking Camera T265 is Intel® RealSense™ SDK 2.0, which may be obtained from http://intelrealsense.com/developers/

The Intel® RealSense™ SDK 2.0 is cross-platform and open source and can also be used to drive other Intel® RealSense™ products, such as the D400 series of depth cameras. It is best supported on Windows and Linux platforms, with somewhat reduced functionality on Mac.

The SDK 2.0 also contains a variety of other sample applications (with source code and documentation) showing how to use Intel® RealSense™ devices, including the T265 tracking camera. The ROS (Robot Operating System) can also be used to interact with Intel® RealSense™ devices. https://github.com/IntelRealSense/realsense-ros/ contains ROS integration, tools, and sample applications built on top of Intel® RealSense™ SDK 2.0. All of these code samples can be used directly in testing, modified to suit testing purposes, or serve as inspiration for new applications built by users.

For the Intel® RealSense™ Tracking Camera T265 , the most basic sample applications are found in the "pose" and "pose-predict" folders. These two samples show how to obtain pose data via polling and callback interfaces, respectively. Within the Intel® RealSense™ Software ROS repository, the T265RealsenseNode class is a good place to start, and several samples for how to drive the T265 via ROS may be found here.

4.4 Challenges

We are continuously improving the stability, robustness and tracking performance of the T265. You can always obtain the latest version of the firmware by updating to the newest version of the Intel RealSense SDK. There are a few challenges we want to highlight which apply to all SLAM systems, including the T265.

- **Scene appearance and geometry:** SLAM algorithms typically create a model of the world that relies on "landmarks", or relative configurations of features detected by their sensors, in order to navigate through that world. For example, the Intel® RealSense™ Tracking Camera T265 relies in part on "interesting" visual features seen by its cameras. If the scene has too few features, such as when the cameras faces a white wall or an empty room with textureless surfaces, it can be difficult for the sensor to create a map. On the other hand, scenes with excessive visual texture or repeating visual patterns can occasionally cause confusion in mapping and navigation.
- **Dynamic scenes:** If many objects (such as people) are moving around within the scene, the SLAM system may have difficulty creating a map of landmarks that remain stationary in 3D.
- **Lighting:** Without enough light, the Intel® RealSense™ Tracking Camera T265 may not be able to detect any visual features at all. At moderately low light, sensor noise can impact image quality, and longer exposure times can lead to image blur. Very strong global lighting, or pointing the camera at light sources, can also result in saturation of some or all of the image. No visual features are detected within saturated image regions, and spurious features may be created along the edges of these regions.
- **Reflections:** Specular surfaces in the scene, such as mirrors or glass windows, can cause perception of "phantom" landmarks that appear to be located at incorrect 3D locations.
- **Device motion:** At high speeds, motion blur can cause image data to become largely useless. Sudden accelerations, either in position or orientation, can exceed measurement limits of the IMU, or break assumptions underlying the visual-inertial sensor fusion algorithms.
- **Timing, and compute resources:** Real-time visual-inertial fusion algorithms are very sensitive to the relative timing of their input signals, so if multiple device clocks are involved, any asynchrony that is not properly handled can have tremendous negative impact on results. Similarly, temporary short-falls or interrupts in compute resources can cause damaging data drops or skipping of critical algorithm steps. The architecture of the Intel® RealSense™ Tracking Camera T265 takes care of most of these issues, as compared to many other SLAM systems, but customers must still take care when integrating it with applications on their host computers.

4.5 Beyond SLAM

There are at least two other important aspects that should be considered when developing autonomous navigation systems. When moving around in a 3D world, beyond mapping, the next most important task is collision avoidance. This usually entails measuring continuously the exact 3D distance to all obstacles in the path. While a VIO system actually does create a "sparse" depth-map by tracking feature points, it is generally not sufficient for robust collision avoidance. For this reason, we usually recommend pairing the T265 with an Intel RealSense depth camera, like models D435 or D415, which will provide a very dense and accurate depth map that can be used in 3D Reconstruction to build a full 3D model of the environment, or for object scanning. The Intel RealSense Viewer is a great place to start with seeing how these sensors can behave together. More information can be found at https://www.intelrealsense.com/depth-and-tracking-combined-get-started/.

Another aspect to consider is how to navigate extremely large spaces. One approach that is supported natively on the T265 is "April Tag" tracking, as described in Ref 3. April tags are a type of open source 2D bar code (or fiducial marker) that can be tracked very well in 3D space to give location and orientation of the tags, or of the agent relative to the tag. April tags are especially appealing as they are rotation, shear, lighting, scale, and translation invariant. Since there are many unique codes, this affords a system an option to relocalize based on a different modality from the Visual SLAM. Moreover, it is easy for users to print out April Tags and mark different areas or objects. A code example for using April Tags can be found here: https://github.com/IntelRealSense/librealsense/tree/master/examples/pose-apriltag.

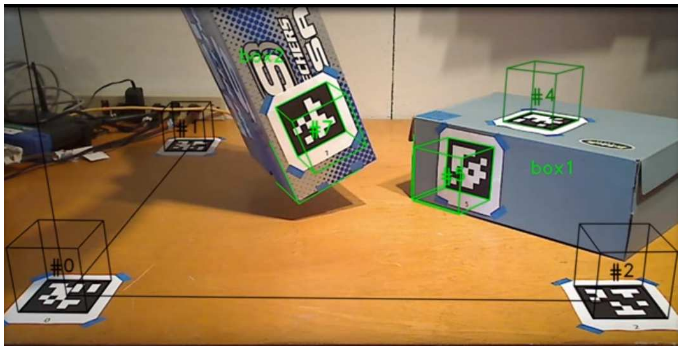


Fig. 6. Example of tracking of April Tags. Screenshot reproduced from Video in Ref 4. April Tag tracking is included in the T265.

5. Summary

Many new and exciting applications are enabled by having accurate, high-speed, low-latency "inside-out" tracking of 6DoF position and orientation. With a small form factor, low power, and good open-loop tracking, a tracking device can, for example, be mounted in helmets for AR/VR usages, or on robots for positional tracking. With the ability to relocalize to maps, large area coverage and recall and be enabled, and maps can even be shared between multiple agents. The T265 Tracking Camera is an excellent out-of-the-box solution that should enable quick prototyping and familiarity with performance and programming using the RealSense Viewer app and SDK. The T261 camera module is available for more custom integration.

References

1. https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)
2. https://dev.intelrealsense.com/docs/tracking-camera-t265-datasheet
3. https://april.eecs.umich.edu/software/apriltag.html

4. https://www.youtube.com/watch?v=Y8WEGGbLWlA