Internship Report

Evaluating the Performance of Intel Realsense T265
Xsens Technologies B.V.

Gunish Alag
s2148439
M.Sc. Mechanical Engineering

UNIVERSITY
OF TWENTE.

Contents

Preface

It is a pleasure to submit my internship report towards partial fulfillment of the Masters' course of Mechanical engineering. A summary of the accomplished tasks and milestones throughout the internship has been presented in this document. On a personal level, this internship presented me with multiple challenging tasks, which essentially led to a steep learning curve. My programming skills have been sharpened further and I have witnessed an overall development in my personality.

I would genuinely like to thank my organization supervisor, Mr. Fabian Girrbach and Mr. Aditya Tiwari , who have been an immense inspiration and thoroughly supportive throughout my tenure constantly providing me with their esteemed guidance. I would like to thank my parents for making me capable enough to take on these challenges sportingly. Indeed there were local minimas and pitfalls along the way but attempted to make sure that eventually, everything falls into its place, which fortunately happened. So it almost seems like someone upstairs is happy with me for granting me with wonderful people and opportunities.

1 Introduction

Mobile robots require a combination of accuracy and low latency in their state estimation in order to achieve stable and robust trajectory. However, due to the power and payload constraints, state estimation algorithms must provide these qualities under the computational constraints of embedded hardware. Cameras and inertial measurement units (IMUs) satisfy these power and payload constraints, so visual-inertial odometry (VIO) algorithms are popular choices for state estimation in these scenarios, in addition to their ability to operate without external localization from motion capture or global positioning systems. Being able to track the position and orientation (generally referred to as 6 degrees of freedom) in real time with highly accurate state estimation and low latency does not only have application in robotics but also in the field of Virtual Reality (VR), Augmented Reality (AR) and indoor navigation[4-7].
To have a broad range of application not limited by GPS data one approach is to use visual inertial odometry (VIO)[1-3]. VIO fuses together information coming through camera and reading based on IMU together to give an estimate of the position and orientation in the unknown environment. Applications like Autonomous driving, UAVs, Deep Sea vehicles and many more depend on the data coming through such sensors which provide information of position and orientation based on previous measurements and current understanding of the environment. Data coming through sensors is processed to calculate the current position and understanding of the environment based upon speed estimates, time and course , the information of which is coming through sensors.

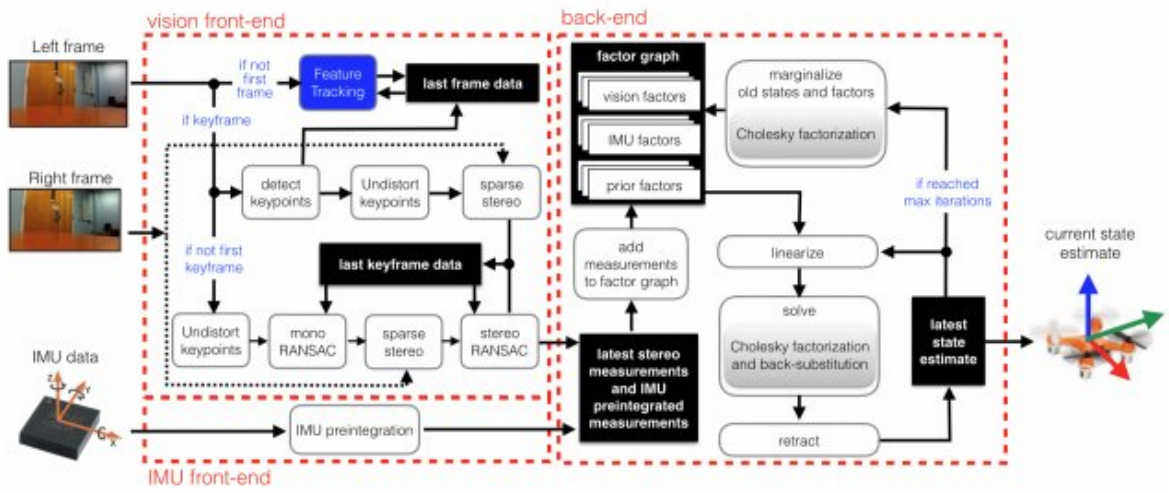


Figure 1: Basic functioning of visual inertial odometery

1.1 Applications

In the modern times where computers are being equipped with every possible feature for complete autonomy state estimation is a key task. Many applications [4-8] in the modern times require robots to be smart enough to navigate through any environment, this is where the concept of state estimation comes in handy and provides for an accurate navigation and localization based on data coming in through reliable sensors.
The measurements are taken at predetermined intervals and the duration between the intervals varies according to the requirement. In applications any known fixed point is taken as the reference point. If a known point is not available then any estimated point is considered as the reference point. Some applications for motion tracking and state estimation are listed below:

1.1.1 Autonomous Underwater Vehicles

The ability of an autonomous vehicle to predict its current position is an essential scientific application. This particular task becomes even more complicated by opacity to all low frequency EM waves, making most commonly used technologies ineffective for this particular application. So it highly depends on acoustic navigation or dead reckoning technologies. Dead-reckoning methods integrate a vehicle's velocity in time to obtain an updated location. In order to dead-reckon, the vehicle must know both the direction and speed of its travel. The simplest methods use a magnetic compass to determine direction, and use speed through the water as a proxy for Earth-referenced speed. However, the large number of error sources for magnetic compasses make measurement of heading to better than a degree accuracy technically challenging.

1.1.2 Autonomous driving

When it comes to applications of autonomous driving [8] on public roads it leads to high risk scenarios, like the crashing of Tesla Model S on Autopilot. Hence it is very crucial that the system that the cars are working on to predict its position and understand the surroundings in real time the sensors used are of high accuracy and reliability. Dead reckoning fuses together dead reckoning with model predictive controller through extended kalman filtering (EKF). Hence the tools used for dead reckoning are of importance and the sensory data coming in through has to be accurate and reliable.

2 Task Description

The objective of the assignment is to benchmark the performance of Realsense T265 for motion tracking as it could be a reliable and accurate solution to such problems at low cost which is bench marked against HTC Vive which is already an established tracking system but is limited by the range of HTC Vive base stations as it can only estimate accurate measurements for distance up to 5m and not beyond. Cheap and reliable solutions are of key interest in the research community for dead reckoning as no prior knowledge of the terrain or the environment is known and it is very important that the technology used for various applications in these circumstances could provide an accurate measurements of position, velocity in real time.
The following are the tasks assigned to me for the duration of the internship :

1. Design a rigid holder for different sensor systems using 3D rapid prototyping techniques.
2. Develop a software framework and data pipeline in Robotics Operating System (ROS) and visualize the data from different sensors.
3. Develop a calibration procedure to identify constant transformation between sensor systems.
4. Evaluate the performance of Realsense T265 for indoor navigation.

To achieve the internship tasks I first began with research on different methods that can be adopted to compare trajectories from two different sensors. Two methods were found as explained in section 6 which can be used to compare the performance of the sensors.

As initially there was no mount on which both the sensors could be mounted we began with the first approach mentioned in section 6. It identifies a constant static transformation between the two trajectories and gives us the drift between the two sensors. Providing us with the error between the two sensors which can be considered as the drift between the two sensors.

Next a solid mount for the sensors was designed using Solidworks and was realized through 3D printing. With the mount in place the static transformation between the sensors was known and the methodology with ROS transformations could be used to evaluate the actual error between the two sensors at each point of time. With the mount realized, ROS nodes were created to provide for static transformations between both the sensors to bring them to the same coordinate frame.

Next was the task to decide on the experiments that had to be done on the combined setup of Realsense and HTC Vive tracker such that it would involve testing of the Realsense sensor in scenarios ranging from easy to moderate, the scenarios of the experiment are explained later in detail in section 8.

With the experiments and the setup ready, it was decided to use different data sets for each experiment with different internal parameters so the best set of parameters could be evaluated and what is the affect of changing internal parameters of the Realsense T265.

3 Realsense T265

The sensor consists of two fisheye lens cameras, Inertial measurement unit (IMU) and an Intel Movidius Myraid visual processing unit (VPU). Using the provided sensors the Realsense T265 enables solutions to the problem of dead reckoning as a stand alone device. It provides for tracking in all 6 Degrees of Freedom. The device fuses input from the fisheye lens sensors and the IMU together to provide for a highly accurate real time position with low power consumption. It can be used with offboard processing units such as Raspberry Pi providing long battery life for mobile robotics.

Figure 2: Realsense T265

3.1 Working Of Realsense T265

The Intel RealSense Tracking Camera T265 enables solutions to these challenges as a stand-alone, six-degrees-of-freedom (6Dof) inside-out tracking sensor. The device fuses inputs from multiple sensors and offloads image processing and computer vision from the host system to provide highly accurate, real-time position tracking with low latency and low power consumption.
The Intel RealSense Tracking Camera T265 is designed for flexibility of implementation and to work well on small-footprint mobile devices such as lightweight robots and drones. It is also optimized for small-scale computers such as Raspberry Pi, and for connectivity with devices such as mobile phones or augmented reality headsets.
The Intel Movidius Myriad 2 VPU is a system-on-chip component that is purpose-built for image processing and computer vision at very high performance per watt, including for space-constrained implementations. Its key architectural features are the following:

1. Vector processor cores are optimized for machine vision workloads.
2. Hardware accelerators increase throughput for imaging and computer vision.
3. General-purpose RISC CPU cores coordinate and direct workloads and interaction with external systems.

The Intel RealSense Tracking Camera performs inside-out tracking, meaning that it does not depend on external sensors for its understanding of the environment. The tracking is based primarily on information gathered from two on board fish-eye cameras.The wide field of view from each camera sensor helps keep points of reference visible to the system for a relatively long time, even if the platform is moving quickly through space.A key strength of visual-inertial odometry is that the various sensors available complement each other. The images from the visual sensors are supplemented by data from an on board inertial measurement unit (IMU), which includes a gyroscope and accelerometer. The aggregated data from these sensors is fed into simultaneous localization and mapping (SLAM) algorithms running on the Intel Movidius Myriad 2 VPU for visual-inertial odometry.The SLAM algorithm identifies sets of salient features in the environment, such as a corner of a room or object that can be recognized over time to infer the device's changing position relative to those points.Together, the combination of sensors provides

higher accuracy than would be possible using just one type. The visual information also prevents long-term drift, or the accumulation of small errors in navigation calculations over time, which would cause inaccuracy of position information. The IMU operates at a higher frequency than the cameras, allowing for quicker response and recognition by the algorithm to changes in the device's position.

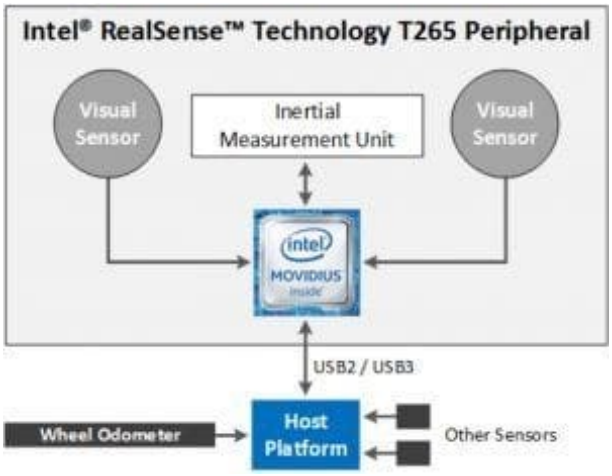


Figure 3: Realsense Working

3.2 Intel Parameters

Realsense allows us to have certain parameters switched based on which the pose estimation of the device changes as it changes what data from the sensor is used to estimate the pose of the device.

3.2.1 3D Mapping

The internal map allows the device to recognize places it's been before so that it can give a consistent pose for the next location. The map has a fixed finite size and will keep the most often/recently seen locations. Without this it will be operating on completely open loop and will accumulate drift over time. The device will use the map to close modest loops and recover from small drifts.

3.2.2 Pose Jumping

With pose jumping the internal map allows the device to recognize places it's been before so that it can give a consistent pose for that location. The map has a fixed finite size based on the computer that it is coupled with and will keep the most often/recently seen locations till it doesn't run out of memory and will replace the oldest feature coming in with the latest features. Without this it will be operating completely open loop and will accumulate more drift. The device will use the map to close modest loops and recover from small drifts.

3.2.3 Enable Re-localization

This allows the device to solve the Kidnapped Robot Problem, i.e. it will allow connecting the current map to a loaded map or connecting the current map to itself after accumulating a large drift, say after closing a large circle or covering the camera and walking around, this will enable the sensor to recognise the places where it has been before and recognise its current position based on that map. This is independent of jumping the pose. When fooled this feature can lead to much larger errors than are likely via the basic mapping.

4 Holder Design

To get comparable data from both the sensors simultaneously the sensors had to be mounted on a common holder of which the static transformation would already be known to us. The sensors to be mounted on this holder were :

1. Realsense T265 (108 x 25 x 13 mm)
2. HTC Vive Tracker (dia. - 99.65 mm)
3. Xsens MTi 300 with Chameleon 3 monocular camera (44x 52 x 60 mm)

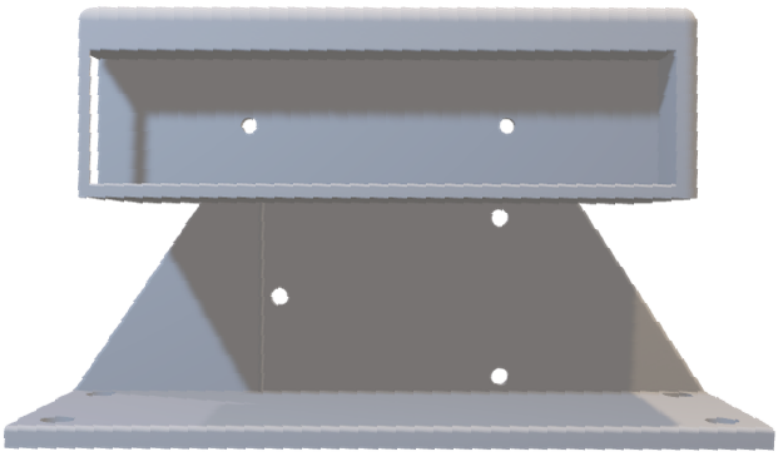


Figure 4: Final Holder Design

To have the least amount of static transformations between the sensors it was attempted to have the center of mass of all the 3 sensors on a single axis. Hence the final design as shown in Figure 4. was finalized as all the three sensors are aligned in the z axis with transformation only in the x and y axis without any rotation. This would help us later in trajectory alignment which is explained in section 6. The design was realized using 3D printing technology at a 40 percentage fill rate for possible high production rate. The material used for the rapid prototyping was thermoplastic named Polylacitic Acid (PLA).

5 ROS

The Robot Operating System (ROS) is a standalone tool that can be used to develop a code for a robot. The framework of ROS eliminates the need to gather various software drivers, third party tools for computer vision and other simulation tools. It is of great use as all the tools required to program a robot are available at a single place, which can be used to program different robots for different applications without going through the hassle of finding different frameworks and tools for that particular application.ROS is an OS in concept because it provides all the services that any other OS does—like hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

5.1 Working of ROS

In general, ROS consists of code and tools that help your project's code run and do the required job—including the infrastructure for running it, like messages passing between processes.

ROS is designed to be a loosely coupled system where a process is called a node and every node should be responsible for one task. Nodes communicate with each other using messages passing via logical channels called topics. Each node can send or get data from the other node using the publish/subscribe model.

The primary goal of ROS is to support code reuse in robotics research and development so you can find a built-in package system. Again, keep in mind that ROS is not an OS, a library, or an RTOS. It's a framework using the concept of an OS.

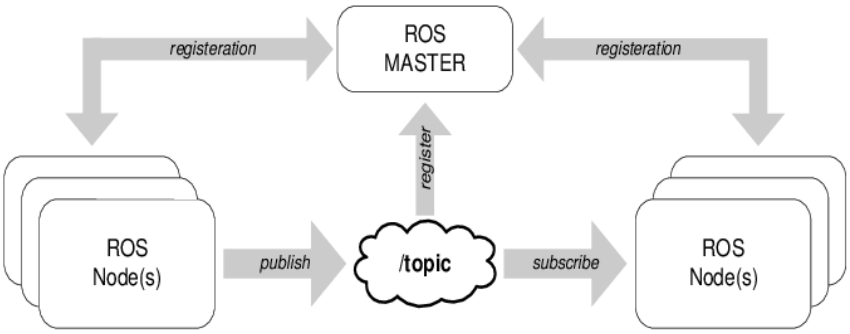


Figure 5: ROS Node communication through topics

6 Sensor Data Comparison Methods

As we began exploring the options on how to compare the results from both sensors it became evident that there were two approaches that could be used for comparison

1. **Trajectory alignment**
   From the data coming through the sensors an unknown transformation is estimated based on the complete trajectories that are recorded. Through this estimated transformation trajectories are aligned to the same coordinate frame. After aligning the two trajectories using the estimated transformation the difference left between the two trajectories is the drift or in our case the error between the sensors.
2. **ROS Transformations**
   With the static transformations known between the two sensors, ROS nodes were created to compensate for the static transformation, thus bringing both the sensors to the same coordinate frame initially through which after a prolonged experiment it could be seen what has been the drift in the realsense sensor

6.1 Static Transformation Identification

Without known static transformation between the two sensors it wouldn't be possible to have both the sensors in the same coordinate frame or Tf tree. Hence by recording the dataset from both the sensors simultaneously (over the same path) and aligning the two trajectories together by a calculated static transformation it would be possible to estimate the performance of one trajectory with respect to the other. To calculate the parameters that provide for a static transformation there were four possible methodologies that could be used to estimate the parameters namely

1. SVD Method
2. Horns Method
3. Transformation using Unit quaternions
4. Transformation using Dual quaternions

To calculate static transformation between two trajectories SVD and Horn's methodologies were realised during the time of the internship.

6.1.1 SVD Method

Developed by Umeyama et.al. [10], this method helps in finding a transformation matrix between two trajectories that have a constant transformation between them. Example if two trajectories from two different sensors are recorded but the coordinate axis for the sensors is different, this method could help find the constant transformation between the two trajectories to represent them in the same coordinate frame. The algorithm uses least square estimation to estimate the transformation between two trajectories. The algorithm is explained in detail below:

Input data

$\{\bar{p}_i\}_{i=0}^{N-1}$ = Ground truth

$\{p_i\}_{i=0}^{N-1}$ = Estimate

Output
Transformation parameters (s, R, T) that maximize $\Sigma_{i=0}^{N-1} \left\| p_i - sR\overline{p_i - t} \right\|^2$

Steps

1. Mean for both the trajectories is calculated

$$\mu_p = \frac{1}{N}\Sigma_{i=0}^{N-1} p_i \quad \mu_{\bar{p}} = \frac{1}{N}\Sigma_{i=0}^{N-1} \bar{p}_i$$

2. Next construct the correlation matrix between the estimate and the estimate and the groundtruth centered around origin.

$$\Sigma = \frac{1}{N}\Sigma_{i=0}^{N-1} \left(p_i - \mu_p\right) \left(\left(\bar{p}_i - \mu_{\bar{p}}\right)^T\right.$$

3. Singular value decomposition is performed on the correlation matrix to obtain the rotation matrix.

   $\Sigma = UDV^T$

   If $\det(U)\det(V) < 0$ then

   $\mathrm{W} = \mathrm{diag}(1, 1, -1)$

   Else $\mathrm{W} = I_{3\times 3}$

4. $\mathrm{R} = UWV^T$ provides us with the rotation matrix between the two trajectories

5. $\mathrm{s} = \frac{1}{\sigma_8^2}\mathrm{trace}(DW)$ where $\sigma_p^2 = \frac{1}{N}\Sigma_{l=0}^{N-1} \left\|p_l - \mu_p\right\|^2$ provides us with the scale between the two trajectories if the scale is not known.

6. $\mathrm{T} = \mu_p - \mathrm{SR}\mu_{\bar{p}}$ from the estimated parameters

6.1.2 Horn's Method

Horns Method [11] follows quite similar steps as shown above except for how to calculate the rotation matrix and compute the scaling between the two trajectories. In this method the same maximization problem is taken but to estimate the parameters instead of using singular value decomposition, eigen decomposition is used. The detailed algorithm for the estimation of the parameters is shown below:
Input data

$\{\bar{p}_i\}_{i=0}^{N-1}$ = Ground truth

$\{p_i\}_{i=0}^{N-1}$ = Estimate

Output

Transformation parameters $(\mathrm{s}, \mathrm{R}, \mathrm{T})$ that maximize $\Sigma_{i=0}^{N-1} \left\|p_i - sR\overline{p_i - t}\right\|^2$

1. Mean for both the trajectories is calculated

$$\mu_p = \frac{1}{N}\Sigma_{i=0}^{N-1} p_i \quad \mu_{\bar{p}} = \frac{1}{N}\Sigma_{i=0}^{N-1} \bar{p}_i$$

2. Next construct the correlation matrix between the estimate and the estimate and the groundtruth centered around origin.

   $H = \Sigma^T = \frac{1}{N}\Sigma_{i=0}^{N-1} \left(p_i - \mu_p\right) \left(\left(\bar{p}_i - \mu_{\bar{p}}\right)^T\right.$

3. Assuming that the correlation matrix is non singular, it can be decomposed using eigenvalue decomposition.

   $\mathrm{H} = \mathrm{US}$
   $\mathrm{U} = \mathrm{HS}^{-1}$
   where $\mathrm{S} = \left(\mathrm{H}^{\mathrm{T}}\mathrm{H}\right)^{1/2}$

4. $\widehat{R} = H\left(\frac{\mu_1\mu_1^T}{\sqrt{\lambda_1}} + \frac{\mu_2\mu_2^T}{\sqrt{\lambda_2}} + \frac{\mu_1\mu_1^T}{\sqrt{\lambda_3}}\right)$ where $\{\lambda_i\}$ and $\{\mu_i\}$ are the eigenvalues and eigen vectors respectively obtained from the S matrix.

Special cases if the H matrix is singular and has less than full rank then the above solution does not hold true. To avoid this the calculation of the rotation matrix is slightly modified. Assuming the third eigenvalue is zero then the modified rotation matrix can be calculated as follows:

$\widehat{R} = HS^{+} \pm \frac{x}{\sqrt{|\mathrm{race}(x)|}}$
where $S^{+} = \left(\frac{\mu_1\mu^T}{\sqrt{\lambda_1}} + \frac{\mu_2\lambda_2^T}{\sqrt{\lambda_2}}\right)$
$\mathrm{X} = \left[\left(\mathrm{MS}^{+}\right)\left(\mathrm{MS}^{+}\right)^{T} - \mathrm{I}\right]\mu_3\mu_3^T$

6.2 ROS Transformations

With the transformation known of both the sensors on the holder, it would be easier to use ROS transformation in order to bring both the sensors to the same coordinate frame. This helps in estimating the pose error between the two sensors, as after moving in different shapes and orientations the data coming through the sensors could provide us with a reliable error between the poses of the two sensors.

After bringing both the sensors in the same coordinate frame under the same TF tree a virtual position of the realsense T265 was placed based on the static transformations. This was done to calculate the drift of the realsense from its original expected position to its current position. This method would have an advantage over the previously used method as it could be implemented real time and does not require any information from the sensors except for the static transformation between the sensors based on the holder they are mounted on. And is not susceptible to errors unless the holder is deformed or used in extreme conditions where the relative position of the sensors is subject to change.

6.3 Method used in our experimentation

After creating scripts(appendix listing 5) for all the three methods it was decided to use the ROS transformations as if try estimate the transformation between the two sensors using SVD or Horns method it also tries to model the error that is accumulated. As if there is a drift between the sensors at any point of time the estimation method will try and find the best set of parameters that closely relate the two trajectories. Hence it was decide to use the ROS transformation and develop a new node in ROS that looks a constant transformation between the two sensors providing constant information how far the sensor has drifted which can help understand the working and shortcoming of the realsense t265 in various applications.

7 Performance Evaluation Methods

To be able to evaluate the performance between the two sensors some method of comparing the trajectories from the sensors was required. To do so the following two evaluation methods [14] were used.

7.1 Absolute Trajectory Error

Absolute Trajectory Error (ATE) – The root mean square error between the ground truth (HTC Vive) and the estimate (Realsense T265) calculated for all the points over the entire trajectory.. Benefit of using ATE for evaluation is it provides a single number that can be used to compare the error between multiple trajectories and helps understanding which trajectory performs best in comparison to the ground truth.

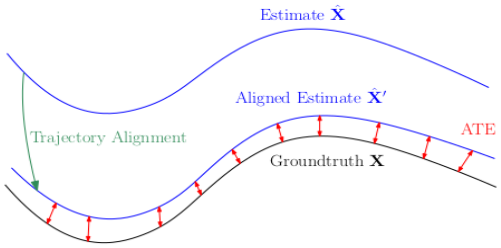


Figure 6: Absolute Trajectory Error

7.2 Relative Pose Error

The basic idea of relative error is that the estimation quality can be evaluated by measuring the relative relations between the states at different times. Since the relative error does not generate a single number but a collection of errors for all the sub-trajectories that satisfy certain criteria, statistics such as the median, average and percentiles can be calculated, which gives more information than Absolute Trajectory Error.

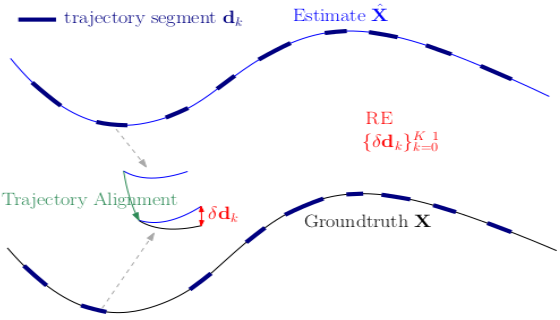


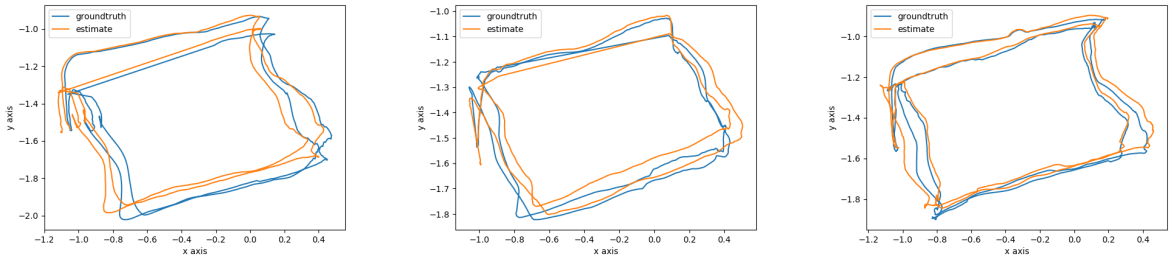
Figure 7: Relative Pose Error

8 Experimental Comparison

To be able to evaluate the performance of Realsense T265 against HTC Vive tracker it was essential to run experiments on the setup in different environments and comparing the performance of different internal settings of the realsense T265 parameters as well. So for each experiment three different datasets were recorded with different sets of parameters.

8.1 Dataset I

After initialization the setup was moved in a square shape of 1x 1.5 m dimensions without rotating the sensors at all (human errors are expected but no major drift in orientation).The entire procedure was repeated 2 times and was placed at the initial position and orientation from where it was initialized. The trajectory paths for the dataset with the varying parameters in shown in Figure 8.



(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 8: Dataset I

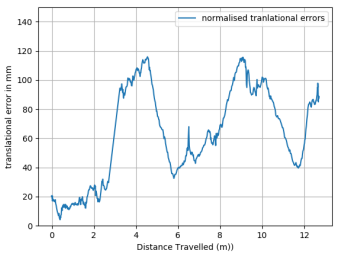
8.1.1 Dataset I Results

With the datasets recorded the error matrices can be constructed (absolute trajectory error and relative pose error) for both the position and the orientation of the sensors based on where the realsense sensor is expected to be and where it actually is.

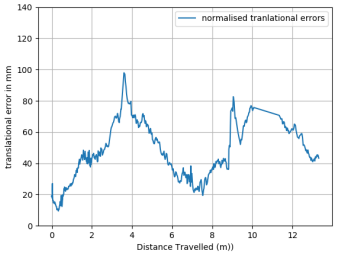
8.1.2 Absolute Trajectory Error Dataset I

From the Table 1 and the Figure 9(a), it can be seen that the error in the estimation of the position are accumulated over time if 3D mapping of the Realsene T265 is disabled. This can be explained by the fact that when the realsense T265 operates without trying to rectify the errors it has accumulated over time as it does not use the prior information from the visual features that it had recorded earlier. The same can be attributed to the fact the sensors is not able to correct itself for any drift caused in its orientation as can be seen in Fig 10.

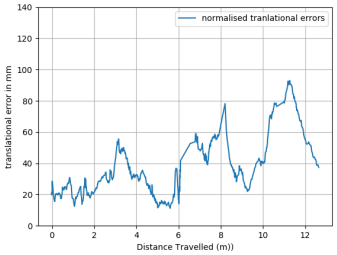
As for the 3D mapping enabled and the pose jumping enabled it can be seen both the parameters perform quite close to each other as both are trying to compensate for errors accumulated over time and rectifying them based on the visual features that the sensor has seen before, which helps in providing with a better estimate of the current position and orientation.



(a) 3D Mapping Disabled



(b) 3D Mapping Enabled

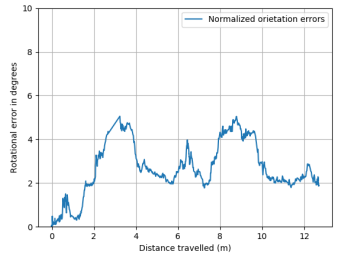


(c) Pose Jumping Enabled

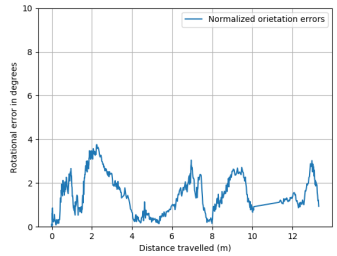
Figure 9: Dataset I Absolute Trajectory Error - Translation Error in mm

Table 1: Transnational Error Dataset I (in cm).

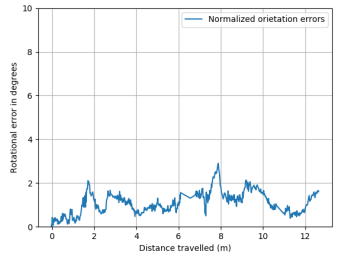
Transnational Errors	**Mapping Disabled**	**Mapping Enabled**	**Mapping Enabled**
RMSE	6.86	4.48	4.05
Mean	5.94	4.04	3.61
Median	5.98	3.94	3.10



(a) 3D Mapping Disabled



(b) 3D Mapping Enabled



(c) Pose Jumping Enabled

Figure 10: Dataset I Absolute Trajectory Error - Orientation Error in degrees

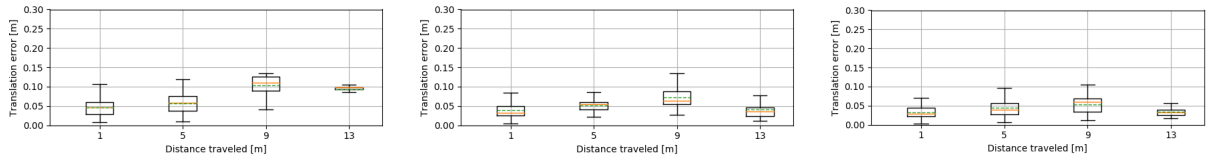
Table 2: Orientation Error Dataset I (in degrees).

Orientation Errors	**Mapping Disabled**	**Mapping Enabled**	**Mapping Enabled**
RMSE	2.61	1.57	1.08
Mean	2.21	1.23	0.93
Median	2.15	1.11	0.96

8.1.3 Relative Pose Error Dataset I

The relative pose error gives us the information how the drifting of the estimated trajectory is accumulated over different sub trajectory lengths. As can be seen in Figure 10 that longer sub trajectory lengths show a higher drift in translation when 3D mapping is disabled for Realsene T265, the reason why such accumulation of error takes place is because the sensor is mostly dependant on the information coming in from the IMU sensors and does not take into account the visual features that it has seen and hence is not able to rectify its positioning.

This can be validated from Table 4, as the errors over large sub trajectory lengths and short sub trajectory lengths for 3D mapping enabled and Pose Jumping enabled mostly remain the same and the error is not accumulated over time, on the other hand for when 3D mapping is disabled the sensors start to accumulate orientation errors as well and only when brought back to the initialization point are the errors reduced but are still not completely 0. However the error is mostly constant for short and large sub trajectory length for when 3D mapping and Pose Jumping is enabled, it cannot be ignored that even with these features being enabled the sensors drifts from the original trajectory and wasn't able to rectify itself of this drift in both translation and orientation.

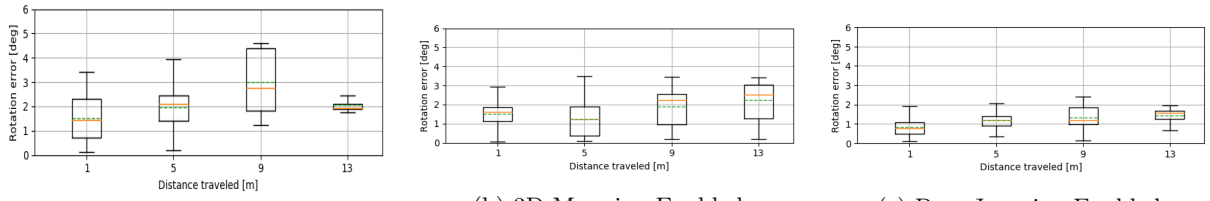


(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 11: Dataset I Relative Pose Error (Translation Error in m)

Table 3: Translation Error at different sub trajectory lengths in cm.

Translation Errors	**Drift at 1m**	**Drift at 5m**	**Drift at 9 m**	**Drift at 13 m**
Mapping Disabled	4.68	5.60	10.4	9.22
Mapping Enabled	3.97	5.18	7.18	4.02
Pose Jumping Enabled	3.26	4.43	5.39	3.36



(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

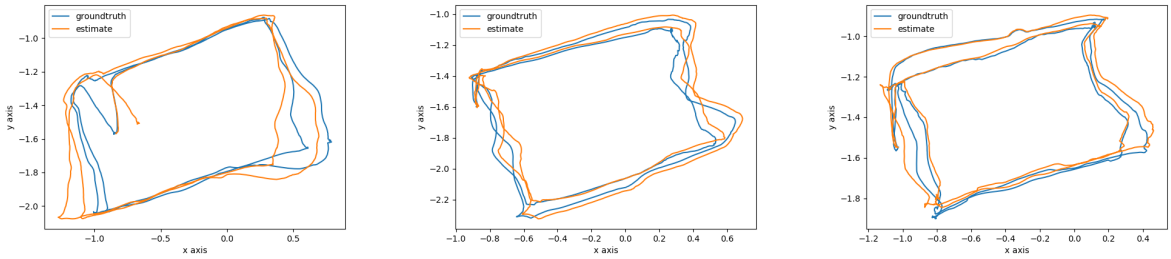
Figure 12: Dataset I Relative Pose Error (Orientation Error in degrees)

Table 4: Orientation Error at different sub trajectory lengths.

Orientation Errors	**Drift at 1m**	**Drift at 5m**	**Drift at 9m**	**Drift at 13m**
Mapping Disabled	1.50	1.95	3.00	2.10
Mapping Enabled	1.50	1.25	1.90	2.25
Pose Jumping Enabled	0.80	1.20	1.30	1.45

8.2 Dataset II

The setup after initialization was moved in a rectangular motion of dimensions 1x 1.5 m twice, after reaching a corner the setup was rotated 90 degrees clockwise. The procedure was repeated twice and was placed back at the point of initialization with the same orientation. Fig 13 provides visual representation of the trajectory followed during recording of the dataset.



(a) 3D mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 13: Dataset II

8.2.1 Dataset II Results

With the datasets recorded the error matrices can be constructed (absolute trajectory error and relative pose error) for both the position and the orientation of the sensors based on where the Realsense sensor is expected to be and where it actually is.

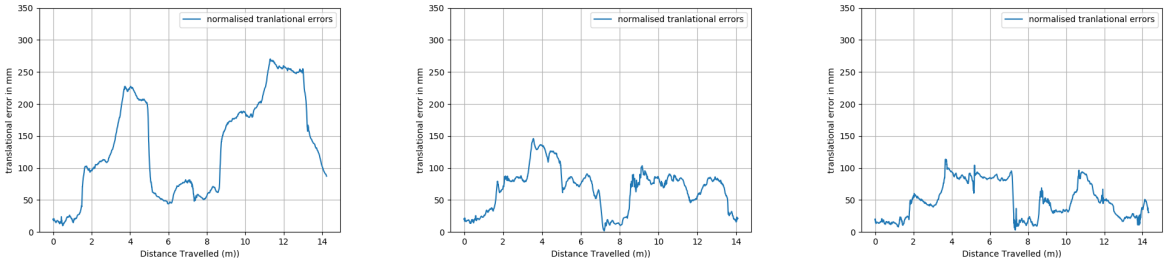
8.2.2 Absolute Trajectory Error Datasect II

As there was rotation introduced in this experiment from Fig 14 and Table 5 it can be seen that the translation error is significantly higher in all the three cases when compared to Dataset I but mostly when the 3D mapping feature in Realsense was disabled. The maximum transnational drift increased more than twice in this case, and as for cases when 3D mapping was enabled the sensor experienced a higher translation drift as well when compared to no rotation. This a was a key flaw that needed to be evaluated as to why this was happening, the problem was tested with a basic experiment and is explained in **section 8.3** . As in use case scenarios if the sensors experiences a drift due to change in orientation it can accumulate largely over time resulting in completely unreliable position estimates.

From Figure 15 the orientation drift between the sensors is evaluated. The sensors do not seem to be highly affected by the parameters for orientation estimation, except for the end result when the sensors are put back to its original place from where they were initialized. As can be seen in Fig 14 and Table 6 that after loop closure the sensors are able to identify its position of initialization and return back to it, which is not the case for when 3D mapping is disabled and it ends up at a higher drift in position and orientation.

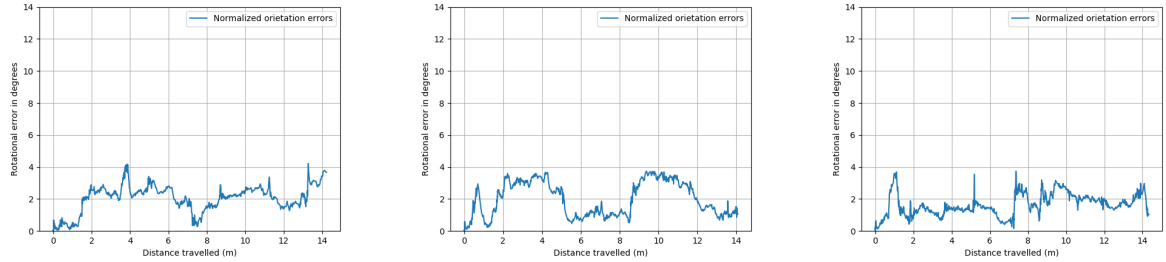
Table 5: Translation Error in m

Transnational Errors	**Mapping Disabled**	**Mapping Enabled**	**Mapping Enabled**
RMSE	14.05	6.60	5.70
Mean	11.30	5.55	4.90
Median	9.40	5.70	4.55



(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 14: Dataset II Absolute Trajectory Error - Translation Error in cm



(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 15: Dataset II Absolute Trajectory Error - Orientation Error in degrees

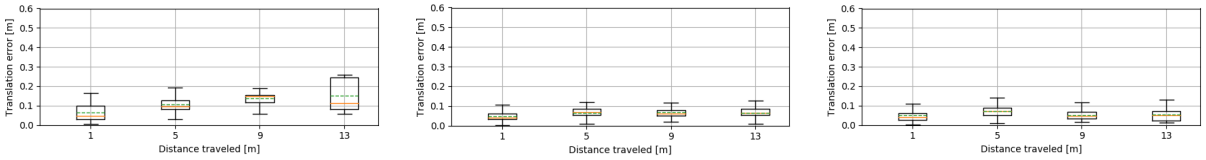
Table 6: Orientation Error in Degrees.

Orientation Errors	**Mapping Disabled**	**Mapping Enabled**	**Mapping Enabled**
RMSE	2.10	2.05	1.65
Mean	1.70	1.70	1.50
Median	2.00	1.30	1.45

8.2.3 Relative Pose Error Dataset II

From Table 7 it can be seen that the translation drift for when 3D mapping is disabled keeps on increasing with increasing sub trajectory lengths, hence without being able to correct for the errors that it accumulates over time the estimation of the position keeps on becoming more unreliable and probably with higher trajectory lengths it might not be possible to depend on the estimate coming in from the sensor. Whereas for when the parameters are enabled the sensors is able to identify the small errors coming in the estimation of position from IMU sensors and is able to correct for small such errors on a regular basis and give a much better estimate.
The estimation of orientation tells a slightly different story as there is a certain drift in orientation in all the three cases the drift is quite similar for all the three cases. The major difference can be seen when pose jumping is enabled as when it identifies itself in a situation where the sensor has been to it identifies the place and orientation it was in and based on that is able to correct for both the things, resulting in a much better estimate for both position and orientation as can be seen in Fig 17(c). However Pose jumping gives a much better result for this particular case it needs to be taken into account that if the features present in the visuals are quite similar throughout the environment then the sensors might be

fooled and can result in a much higher error, and in situations it might be best to use the sensor with only 3D mapping enabled and no pose jumping.

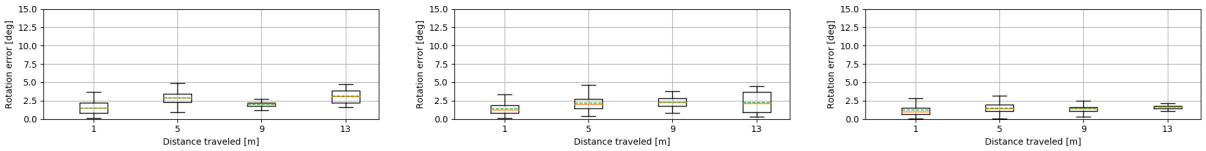


(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 16: Dataset II Relative Pose Error - Translation Error in m

Table 7: Translation Error at different sub trajectory lengths in cm.

Translation Errors	**Drift at 1m**	**Drift at 5m**	**Drift at 9 m**	**Drift at 13 m**
Mapping Disabled	6.55	10.70	13.70	15.20
Mapping Enabled	4.80	6.60	6.70	6.05
Pose Jumping Enabled	5.00	7.15	5.20	5.60



(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 17: Dataset II Relative Pose Error - Translation Error in degrees

Table 8: Orientation Error at different sub trajectory lengths in degrees.

Translation Errors	**Drift at 1m**	**Drift at 5m**	**Drift at 9 m**	**Drift at 13 m**
Mapping Disabled	1.50	2.90	2.00	3.30
Mapping Enabled	1.40	2.25	2.30	2.25
Pose Jumping Enabled	1.25	1.55	1.45	1.65

8.3 Reason for drift on Rotation

Based on the second experiment performed (Dataset II) it was quite evident that rotating the sensors is causing some kind of constant drift between the estimate and actual trajectory and to evaluate this we ran a small experiment with moving the sensors through a line and rotating it 90 degrees and then moving the rotated sensors along the same line to evaluate what kind of drift is caused by rotation between the sensors.

From fig. 18 it can be clearly seen that due to rotating the sensors the realsense starts to drift from its original trajectory and is not able to correct for this drift. It is only after the rotation is compensated for that the transnational drift is reduced. On further inspection of the sensor it was found that when the sensor is rotated it does not rotate about its center but in a circle, causing the drift in its position estimate.

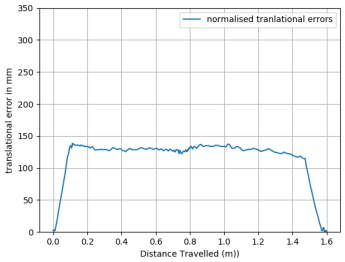


Figure 18: Transnational drift due to rotation.

8.4 Dataset III

Next after having the system tested in known environment it was tested in unknown environment. First both the sensors were initialized in the known environment and moved in rectangle of 1.5 x 2 m dimension, next the setup was taken outside the range of HTC vive environment and moved along a path of 6m length rotated 180 degrees and then returned back to the HTC vive environment. On returning the estimated position of the Realsense was compared with the HTC vive tracker to check if the unknown environment caused drift in position estimation. From Fig 19. the trajectories followed for dataset recording can be seen with changing parameters.

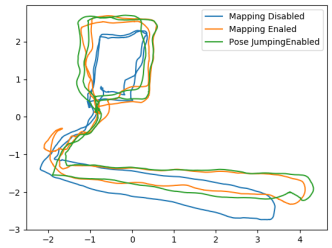


Figure 19: Dataset III

8.4.1 Dataset III Results

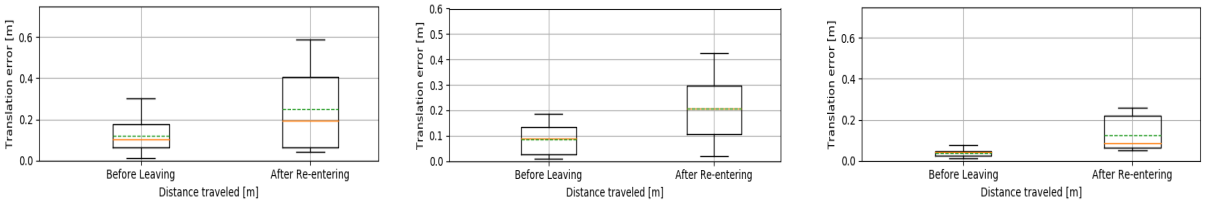
As data for the HTC vive tracker was not available for the entire trajectory the evaluation methods used so far could not be used. Hence to evaluate the performance between the two sensors average error of the drift between he two sensors before leaving the environment and after reentering were used to evaluate the performance of the sensor in this experiment.

8.4.2 Transnational Error

From figure 20 it can be seen that the best estimation of position comes when 3D mapping and pose jumping are enable, having these options disabled results in error accumulation over time. They key parameter that results in accumulation of error is disabling 3D mapping, as without visual features over time the sensor is not able to correct for its position estimate as it depends completely on the inertial sensory data and it is not consistent throughout. Whereas pose jumping is a helpful parameter but only helps when the sensor is used in a completely known environment or a loop closure as it helps the sensor to identify places it has already seen and jump to position it remembers it was in. This feature can result in very reliable estimates or can result in huge errors when fooled through similar features in the environment.

Table 9: Transnational Error in cm.

Transnational Errors	Mapping Disabled	Mapping Enabled	Mapping Enabled
Before Leaving	11.90	8.40	3.90
After Re-entering	25.5	20.60	12.5

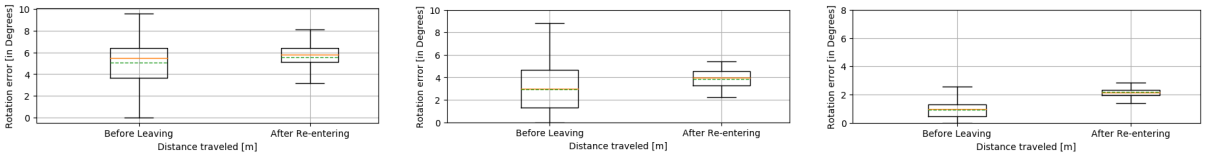


(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 20: Dataset III Translation Error in m

8.4.3 Orientation Error

As can be seen in Table 10 the drift in orientation is quite similar for both when pose jumping is enabled and 3D mapping enabled. Whereas the orientation drift when 3D mapping is disabled is quite high before leaving the HTC vive environment and even after re-entering. Once the sensor drifts to an inconsistent estimate in orientation the sensor is unable to compensate for the error. Whereas having these parameters enabled helps eliminating some of the drift in orientation. Although the percentage increase in orientation drift was higher when the sensors were exposed to unknown environment for when 3D mapping was enabled as well, but was still less than the overall error accumulated when these features were disabled.



(a) 3D Mapping Disabled (b) 3D Mapping Enabled (c) Pose Jumping Enabled

Figure 21: Dataset III Orientation Error in degrees

Table 10: Orientation Error.

Orientation Errors	Mapping Disabled	Mapping Enabled	Pose Jumping Enabled
Before Leaving	5.10	2.90	0.90
After Re-entering	5.55	3.85	2.20

9 Conclusion

From the experiments performed the best set of parameters and their respective errors were compared. On analysing these results it was tried to determine what could be the reason due to which the sensor accumulates error even with highly efficient and accurate sensors being used.

Table 11: Transnational Error in cm.

	Dataset I	**Dataset II**	**Dataset III**
Trajectory Length	1300 cm	1400 cm	3400 cm
RMSE Error	4.05 cm	5.70 cm	12.40 cm
Max Error	9.30 cm	11.35 cm	28.50 cm

1. **Drift accumulation due to Rotation**
   As we have already seen in the experiments performed so far, whenever there is a rotation in around any axis the Realsense tends to rotate in a circle and not about its own axis as how we expect it should. This causes the error to be accumulated over time causing inaccurate readings coming through the sensor.

2. **Inaccurate estimation of position due to bright lighting**
   The sensors were initialized in a known environment and then taken out of range of HTC Vive environment into a completely open lobby. Due to the bright lighting coming in through the windows the position estimate of the Realsense was highly inaccurate. This can be clearly seen from Fig 22. As when the dataset recorded was in a dim light setting the sensor did not drift much from its actual reading which it was estimating. However when the data was recorded in a highly bright setting the system estimated almost a transnational drift of order 1 m. Although the data recorded had a length of around 70 m. However even for a large trajectory accumulation of transnational remains a key problem and the ability of the Realsense T265 to not adapt to lighting situations results in poor performance. This can still be resolved by using physical filters on the Realsene T265 helping the sensor to grasp features in the surroundings and estimate the position of the sensor more reliability but it cannot be a permanent solution to this problem as depending on the point of application the physical filters will have to be modified.

(a) Error when recorded in Dim lighting

(b) Error when recorded in Bright Lighting

Figure 22: Translation Error in m

3. **High drift leading to NaN readings in complete darkness**

   When the sensor was completely deprived of any visual feature in the environment, by covering the sensor with a cloth or by hand, it drifted at a very high pace and resulted in Nan values. This is similar to the previous case when the lighting used was too bright leading to less features being detected. But in this particular case when the sensor is completely covered it cannot detect any visual features and is not able to estimate a position resulting in NaN values.

10 Discussions and Future work

This report dictates the different tasks carried out during the internship at Xsens Technologies. The performance of Realsense T265 was bench-marked against HTC vive tracker with different approaches i.e by estimating a static transformation between the two trajectories by two different methods and by application of ROS transformations to look up constant drift between the actual and the estimated trajectory.

For the mounting of the sensors a design for holder was designed using Solidworks and realized using 3D printing technology. With the holder the static transformation was known and the drift between the sensors could be calculated to evaluate the performance of Realsense in different environments.

For the experiments performed during the duration of the internship enabling 3D mapping and Pose jumping parameters showcased the best results. But for practical applications in varying environments these might not be the best set of parameters. As pose jumping is an unreliable parameter as it can be fooled depending on the visual features present in the environment, similar looking places might confuse the sensor and send inaccurate estimates leading to large error accumulation over time. For Practical implementation enabling 3D mapping and disabling Pose jumping could provide for the best set of results for varying environments as with 3D mapping the sensor is able to correct for small drifts at a high pace and at short intervals resulting in small accumulation of error over time.

The application of the sensors is quite broad and could be used as a standalone tracking and navigation sensors with an on board computer on any robot. The possible working of the sensor with a Raspberry pi on a robot and navigation of the robot through an unknown environment could be a possible future work for better testing of the robustness of the sensor. Apart from this the Realsense provides raw data incoming through Realsense T265 and hence different VIO algorithms could be used to test which performs the best with the provided data from Realsense T265.

Appendices

Codes and Snippets

```

import reading_utilities as ru
import numpy as np
import argparse
import os
import absolute_error as abs
import matplotlib.pyplot as plt
import alignment_util as au
import plotting_utils as pu
import relative_pose_error as rpe


parser = argparse.ArgumentParser(description=''' The path containing the dataset file to be evaluated''')
parser.add_argument('file_path',type = str, help=" path of the file containing data from all the sensors.")
parser.add_argument('bag_name',type = str)
parser.add_argument('vive_topic_name',type = str)
parser.add_argument('realsense_topic_name',type = str)
parser.add_argument('association_type',type= str)
parser.add_argument('results_directory',type = str)
parser.add_argument('error_reading',type = str)
parser.add_argument('vive_topic_name_2',type = str)
args = parser.parse_args()

assert os.path.exists(args.file_path)
alignment_method = 'svd'
pose_gt,t_gt = ru.read_msgs(args.vive_topic_name,args.file_path)


# pose_gt2 = np.delete(pose_gt,np.s_[780:950],0)
# t_gt2 = np.delete(t_gt,np.s_[780:950],0)
pose_es,t_es = ru.read_msgs(args.realsense_topic_name,args.file_path)
pos_realsense,quat_realsense,t_realsense = ru.get_pos_quat_time(pose_es)


if args.association_type == 'associate':
    pose_gt, pose_es = ru.associate_time(t_gt, t_es, 0.02, pose_gt, pose_es)
    pos_es, quat_es, t_es = ru.get_pos_quat_time(pose_es)
    pos_gt, quat_gt, t_gt = ru.get_pos_quat_time(pose_gt)


elif args.association_type == 'resample':
    pose_resample_es = ru.resample_length(t_gt, pose_es)
    pos_es, quat_es, t_es = ru.get_pos_quat_time(pose_resample_es)
    pos_gt, quat_gt, t_gt = ru.get_pos_quat_time(pose_gt)


elif args.association_type == 'interpolate':
    pose_interp = ru.interpolate_values(t_gt, pose_gt, t_es)
    pos_gt, quat_gt, t_gt = ru.get_pos_quat_time(pose_interp)
    pos_es, quat_es, t_es = ru.get_pos_quat_time(pose_es)




accum_dist = ru.get_distance(pos_gt)

e_trans, trans_vec, e_scale_perc, trans_error, scale_error = abs.
    compute_absolute_trans_errors( pos_es,pos_gt)
```

```
58
59 e_rot, rot_error, rot_error_norm,e_rot_norm_2 = abs.compute_absolute_rot_error(quat_es,
       quat_gt)
60
61 sub_l,e_trans_rel, e_trans_rel_perc, e_rot_rel,error_list = rpe.compute_relative_error(
       pos_es, pos_gt, quat_es, quat_gt,file_name=args.bag_name)
62
63 rel_trans_err = []
64 rel_trans_err_perc = []
65 rel_rot_err = []
66 for i in range(0,np.shape(error_list)[0],3):
67     rel_trans_err.append(error_list[i,:])
68     rel_trans_err_perc.append(error_list[i+1, :])
69     rel_rot_err.append(error_list[i+2, :])
70 rel_trans_err = np.transpose(np.array(rel_trans_err))
71 rel_rot_err = np.transpose(np.array(rel_rot_err))
72
73 labels = ['Estimate']
74 colors = ['b']
75 v = np.linalg.norm(trans_vec,axis = 1)
76
77
78 fig = plt.figure(figsize=(6, 2.5))
79 ax = fig.add_subplot(
80     111, xlabel='Distance traveled [m]',
81     ylabel='Translation error [m]')
82 pu.boxplot_compare(ax, sub_l, rel_trans_err, labels, colors)
83 fig.tight_layout()
84 plt.grid(True)
85 plt.ylim([0,.3])
86 fig.savefig(args.results_directory+args.bag_name+'_rel_translation_error.png',
       bbox_inches="tight")
87
88
89
90
91 fig = plt.figure(figsize=(6, 2.5))
92 ax = fig.add_subplot(
93     111, xlabel='Distance traveled [m]',
94     ylabel='Rotation error [deg]')
95 pu.boxplot_compare(ax, sub_l, rel_rot_err, labels, colors)
96 fig.tight_layout()
97 plt.grid(True)
98 plt.ylim([0,6])
99 fig.savefig(args.results_directory+args.bag_name+'_rel_rot_error.png', bbox_inches="tight
       ")
100
101
102 pu.plot_traj(pos_es,pos_gt,args.results_directory+args.bag_name)
103 plt.figure()
104 plt.plot(range(np.shape(t_gt)[0]),pos_gt[:,0],'+')
105 plt.plot(range(np.shape(t_gt)[0]),pos_gt[:,1],'*')
106 plt.plot(range(np.shape(t_gt)[0]),pos_gt[:,2],'--')
107
108
109 plt.figure()
110 pu.plot_errors(accum_dist,trans_vec,"normalised tranlational errors",x_label="Distance
       Travelled (m))",y_label = "translational error in mm")
111
112 plt.ylim([0,140])
113 plt.savefig(args.results_directory+args.bag_name+'_translational_error.png')
114
115 plt.figure()
116 pu.plot_errors(accum_dist,e_rot,"Normalized orietation errors",x_label="Distance
       travelled (m)",y_label = "Rotational error in degrees")
117 plt.ylim([0,10])
```

```
plt.savefig(args.results_directory+args.bag_name+'_rotational_error.png')

plt.show()


ru.writing_errors(trans_error,rot_error_norm,args.results_directory+args.bag_name+"
    _errors")
```

Listing 1: Process to Evaluate the Trajectories

```

import numpy as np
import reading_utilities as ru
import plotting_utils as pu
import alignment_util as au
import matplotlib.pyplot as plt
import relative_pose_error as rpe

import argparse

parser = argparse.ArgumentParser(description=''' The path containing the dataset file to
    be evaluated''')
parser.add_argument('file_path',type = str, help=" path of the file containing data from
    all the sensors.")
parser.add_argument('bag_name1',type = str)
parser.add_argument('bag_name2',type = str)
parser.add_argument('bag_name3',type = str)
parser.add_argument('vive_topic_name',type = str)
parser.add_argument('realsense_topic_name',type = str)
parser.add_argument('association_type',type= str)
parser.add_argument('results_directory',type = str)
parser.add_argument('error_reading',type = str)
parser.add_argument('vive_topic_name_2',type = str)
args = parser.parse_args()



pose_gt_mapping,t_gt_mapping = ru.read_msgs(args.vive_topic_name,args.file_path+args.
    bag_name1)
pose_gt_no_mapping,t_gt_no_mapping = ru.read_msgs(args.vive_topic_name,args.file_path+
    args.bag_name2)
pose_gt_pose,t_gt_pose = ru.read_msgs(args.vive_topic_name,args.file_path+args.bag_name3)

pos_gt_mapping,quat_gt_mapping,t_gt_mapping = ru.get_pos_quat_time(pose_gt_mapping)
pos_gt_no_mapping, quat_gt_no_mapping,t_gt_no_mapping = ru.get_pos_quat_time(
    pose_gt_no_mapping)
pos_gt_pose,quat_gt_pose,t_gt_pose = ru.get_pos_quat_time(pose_gt_pose)


pose_es_mapping,t_es_mapping = ru.read_msgs(args.realsense_topic_name,args.file_path+args
    .bag_name1)
pose_es_no_mapping,t_es_no_mapping = ru.read_msgs(args.realsense_topic_name,args.
    file_path+args.bag_name2)
pose_es_pose,t_es_pose = ru.read_msgs(args.realsense_topic_name,args.file_path+args.
    bag_name3)


pos_es_mapping,quat_es_mapping,t_es_mapping = ru.get_pos_quat_time(pose_es_mapping)
pos_es_no_mapping,quat_es_no_mapping,t_es_no_mapping = ru.get_pos_quat_time(
    pose_es_no_mapping)
pos_es_pose,quat_es_pose,t_es_pose = ru.get_pos_quat_time(pose_es_pose)

plt.figure()
plt.plot(pos_es_no_mapping[:,0],pos_es_no_mapping[:,1],label = 'Mapping Disabled')
plt.plot(pos_es_mapping[:,0],pos_es_mapping[:,1],label = 'Mapping Enaled')
plt.plot(pos_es_pose[:,0],pos_es_pose[:,1],label = 'Pose JumpingEnabled')
plt.legend()
```

```
plt.savefig('../../Alignment/benchmarking_experiments/in_presentation/outside.png')
plt.show()


#
angular_mapping,linear_mapping = ru.read_vel_acc(args.file_path+args.bag_name1,args.
    vive_topic_name_2)
angular_no_mapping,linear_no_mapping = ru.read_vel_acc(args.file_path+args.bag_name2,args
    .vive_topic_name_2)
angular_pose,linear_pose = ru.read_vel_acc(args.file_path+args.bag_name3,args.
    vive_topic_name_2)

norm_ang_map = np.linalg.norm(angular_mapping,axis=1)
mean_ang_map = np.mean(norm_ang_map)
norm_ang_no_map = np.linalg.norm(angular_no_mapping,axis=1)
mean_ang_no_map = np.mean(norm_ang_no_map)
norm_ang_pose = np.linalg.norm(angular_pose,axis=1)
mean_ang_pose = np.mean(norm_ang_pose)


norm_lin_map = np.linalg.norm(linear_mapping,axis=1)
mean_lin_map = np.mean(norm_lin_map)
norm_lin_no_map = np.linalg.norm(linear_no_mapping,axis=1)
mean_lin_no_map = np.mean(norm_lin_no_map)
norm_lin_pose = np.linalg.norm(linear_pose,axis=1)
mean_lin_pose = np.mean(norm_lin_pose)



plt.figure()
plt.plot(range(np.shape(norm_ang_map)[0]),norm_ang_map,label = "map")
# plt.figure()
plt.plot(range(np.shape(norm_ang_no_map)[0]),norm_ang_no_map,label = "no_map")
# plt.figure()
plt.plot(range(np.shape(norm_ang_pose)[0]),norm_ang_pose,label = "pose")
plt.legend()
plt.figure()
plt.plot(range(np.shape(norm_lin_map)[0]),norm_lin_map,label = "map")
# plt.figure()
plt.plot(range(np.shape(norm_lin_no_map)[0]),norm_lin_no_map,label = "no_map")
# plt.figure()
plt.plot(range(np.shape(norm_lin_pose)[0]),norm_lin_pose,label = "pose")
plt.legend()
plt.show()

print("angular velocity  \nno mapping {} \nmapping {} \n pose{}".format(mean_ang_no_map,
    mean_ang_map,mean_ang_pose))
print("linear acceleration \nno mapping {} \nmapping {} \npose {}".format(mean_lin_no_map
    ,mean_lin_map,mean_lin_pose))


# sub_l,e_trans_rel_map, e_trans_rel_perc_map, e_rot_rel_map,error_list_map = rpe.
    compute_relative_error(pos_es_mapping, pos_gt_mapping, quat_es_mapping,
    quat_gt_mapping,file_name=args.bag_name1)
# sub_l,e_trans_rel_no_map, e_trans_rel_perc_no_map, e_rot_rel_no_map,error_list_no_map =
     rpe.compute_relative_error(pos_es_no_mapping, pos_gt_no_mapping, quat_es_no_mapping,
     quat_gt_no_mapping,file_name=args.bag_name2)
# sub_l,e_trans_rel_pose, e_trans_rel_perc_pose, e_rot_rel_pose,error_list_pose = rpe.
    compute_relative_error(pos_es_pose, pos_gt_pose, quat_es_pose, quat_gt_pose,file_name
    =args.bag_name3)
#
# distances = ["1 m no mapping","1 m mapping","1 m pose","5 m  no mapping","5 m mapping
    ","5 m pose","9 m  no mapping","9 m mapping","9 m pose","13 m  no mapping","13 m
    mapping","13 m pose"]
# rel_error = []
# for i in [0,3,6]:
```

```
#     rel_error.append(error_list_no_map[i,:])
#     rel_error.append([error_list_map[i,:]])
#     rel_error.append([error_list_pose[i,:]])
# rel_error = np.transpose(np.array(rel_error))
# labels = sub_l
# colors = ['b']
#
# fig = plt.figure(figsize=(6, 2.5))
# ax = fig.add_subplot(
#     111, xlabel='Distance traveled [m]',
#     ylabel='Translation error [m]')
# pu.boxplot_compare(ax, distances, rel_error, labels, colors)
# fig.tight_layout()
# plt.grid(True)
# # plt.ylim([0,.75])
plt.show()
```

Listing 2: Comparing Datasets

```

import numpy as np
import xsens_common as xs



def compute_absolute_trans_errors(pos_es_aligned,pos_gt):

    trans_vec = (pos_gt-pos_es_aligned)
    e_trans = np.sqrt(np.sum(trans_vec**2,1))
    trans_vec = trans_vec*1000

    motion_gt = np.diff(pos_gt,0)
    motion_es = np.diff(pos_es_aligned,0)

    dist_gt = np.sqrt(np.sum(np.multiply(motion_gt,motion_gt),1))
    dist_es = np.sqrt(np.sum(np.multiply(motion_es,motion_es),1))

    e_scale_perc = np.divide(dist_es,dist_gt)
    trans_error = compute_error_types(e_trans)
    scale_error = compute_error_types(e_scale_perc)

    return e_trans,trans_vec,e_scale_perc,trans_error,scale_error


def compute_absolute_rot_error(q_es_aligned,q_gt):
    quat_error = xs.quatListMulInvQuatList(q_gt, q_es_aligned)
    quat_error = xs.quatListNormalize(quat_error)

    e_rot = xs.quatListToEuler(quat_error)
    e_rot_norm = np.linalg.norm(e_rot, axis=1)
    rot_error = compute_rotation_error(e_rot)
    rot_error_norm = compute_error_types(e_rot_norm)
    return e_rot,rot_error,rot_error_norm,e_rot_norm


def compute_error_types(data_vec):
    stats = dict()
    stats['rmse'] = float(
        np.sqrt(np.dot(data_vec, data_vec) / len(data_vec)))
    stats['mean'] = float(np.mean(data_vec))
    stats['median'] = float(np.median(data_vec))
    stats['std'] = float(np.std(data_vec))
    stats['min'] = float(np.min(data_vec))
    stats['max'] = float(np.max(data_vec))
    stats['num_samples'] = int(len(data_vec))
    return stats

```

```
def compute_rotation_error(rot_error):
    stats = dict()
    stats['rmse roll'] = float(
            np.sqrt(np.dot(rot_error[0], rot_error[0]) / len(rot_error[0])))
    stats['mean roll'] = float(np.mean(rot_error[0]))
    stats['median roll'] = float(np.median(rot_error[0]))
    stats['std roll'] = float(np.std(rot_error[0]))
    stats['min roll'] = float(np.min(rot_error[0]))
    stats['max roll'] = float(np.max(rot_error[0]))
    stats['num_samples'] = int(len(rot_error[0]))

    stats['rmse yaw'] = float(
        np.sqrt(np.dot(rot_error[1], rot_error[1]) / len(rot_error[1])))
    stats['mean yaw'] = float(np.mean(rot_error[1]))
    stats['median yaw'] = float(np.median(rot_error[1]))
    stats['std yaw'] = float(np.std(rot_error[1]))
    stats['min yaw'] = float(np.min(rot_error[1]))
    stats['max yaw'] = float(np.max(rot_error[1]))
    stats['num_samples'] = int(len(rot_error[1]))

    stats['rmse pitch'] = float(
        np.sqrt(np.dot(rot_error[2], rot_error[2]) / len(rot_error[2])))
    stats['mean pitch'] = float(np.mean(rot_error[2]))
    stats['median pitch'] = float(np.median(rot_error[2]))
    stats['std pitch'] = float(np.std(rot_error[2]))
    stats['min pitch'] = float(np.min(rot_error[2]))
    stats['max pitch'] = float(np.max(rot_error[2]))
    stats['num_samples'] = int(len(rot_error[2]))
    return stats
```

Listing 3: Calculating Absolute Trajectory error

```
import reading_utilities as ru
import numpy as np
import argparse
import os
import absolute_error as abs
import matplotlib.pyplot as plt
import alignment_util as au
import plotting_utils as pu
import relative_pose_error as rpe


parser = argparse.ArgumentParser(description=''' The path containing the dataset file to
    be evaluated''')
parser.add_argument('file_path',type = str, help=" path of the file containing data from
    all the sensors.")
parser.add_argument('bag_name',type = str)
parser.add_argument('vive_topic_name',type = str)
parser.add_argument('realsense_topic_name',type = str)
parser.add_argument('association_type',type= str)
parser.add_argument('results_directory',type = str)
parser.add_argument('error_reading',type = str)
parser.add_argument('vive_topic_name_2',type = str)
args = parser.parse_args()

assert os.path.exists(args.file_path)
alignment_method = 'svd'
pose_gt,t_gt = ru.read_msgs(args.vive_topic_name,args.file_path)
pose_gt_before = pose_gt[:740,:]
t_gt_before = t_gt[:740]

pose_gt_after = pose_gt[740:-50,:]
t_gt_after = t_gt[740:-50]
pose_es,t_es = ru.read_msgs(args.realsense_topic_name,args.file_path)
pos_realsense,quat_realsense,t_realsense = ru.get_pos_quat_time(pose_es)
accum_dist = ru.get_distance(pos_realsense)
```

```
34
35 plt.figure()
36 plt.plot(pos_realsense[:,0],pos_realsense[:,1])
37 plt.savefig('../../Alignment/benchmarking_experiments/in_presentation'+args.bag_name)
38
39
40 if args.association_type == 'associate':
41     pose_gt_before, pose_es_before = ru.associate_time(t_gt_before, t_es, 0.02,
      pose_gt_before, pose_es)
42     pos_es_before, quat_es_before, t_es_before = ru.get_pos_quat_time(pose_es_before)
43     pos_gt_before, quat_gt_before, t_gt_before = ru.get_pos_quat_time(pose_gt_before)
44
45     pose_gt_after, pose_es_after = ru.associate_time(t_gt_after, t_es, 0.02,
      pose_gt_after, pose_es)
46     pos_es_after, quat_es_after, t_es_after = ru.get_pos_quat_time(pose_es_after)
47     pos_gt_after, quat_gt_after, t_gt_afer = ru.get_pos_quat_time(pose_gt_after)
48
49
50 elif args.association_type == 'resample':
51     pose_resample_es = ru.resample_length(t_gt, pose_es)
52     pos_es, quat_es, t_es = ru.get_pos_quat_time(pose_resample_es)
53     pos_gt, quat_gt, t_gt = ru.get_pos_quat_time(pose_gt)
54
55
56 elif args.association_type == 'interpolate':
57     pose_interp = ru.interpolate_values(t_gt, pose_gt, t_es)
58     pos_gt, quat_gt, t_gt = ru.get_pos_quat_time(pose_interp)
59     pos_es, quat_es, t_es = ru.get_pos_quat_time(pose_es)
60
61
62 e_trans_before, trans_vec_before, e_scale_perc_before, trans_error_before,
      scale_error_before = abs.compute_absolute_trans_errors( pos_es_before,pos_gt_before)
63 trans_vec_before = np.linalg.norm(np.abs(trans_vec_before),axis = 1)
64
65 e_rot_before, rot_error_before, rot_error_norm_before,e_rot_norm_before = abs.
      compute_absolute_rot_error(quat_gt_before,quat_es_before)
66
67
68
69
70 e_trans_after, trans_vec_after, e_scale_perc_after, trans_error_after, scale_error_after
      = abs.compute_absolute_trans_errors( pos_es_after,pos_gt_after)
71 trans_vec_after = np.linalg.norm(np.abs(trans_vec_after),axis=1)
72
73 e_rot_after, rot_error_after, rot_error_norm_after,e_rot_norm_after = abs.
      compute_absolute_rot_error(quat_gt_after,quat_es_after)
74
75 distances = ["Before Leaving","After Re-entering"]
76 rel_trans_error = []
77 rel_trans_error.append([trans_vec_before/1000])
78 rel_trans_error.append([trans_vec_after/1000])
79 rel_trans_error = np.transpose(np.array(rel_trans_error))
80
81 rel_rot_error = []
82 rel_rot_error.append([e_rot_norm_before])
83 rel_rot_error.append([e_rot_norm_after])
84 rel_rot_error = np.transpose(np.array(rel_rot_error))
85
86
87 labels = ['Estimate']
88 colors = ['b']
89
90 fig = plt.figure(figsize=(6, 2.5))
91 ax = fig.add_subplot(
92     111, xlabel='Distance traveled [m]',
93     ylabel='Translation error [m]')
```

```
pu.boxplot_compare(ax, distances, rel_trans_error, labels, colors)
fig.tight_layout()
plt.grid(True)
plt.ylim([0,.6])
fig.savefig(args.results_directory+args.bag_name+'_trans_drift.png', bbox_inches="tight")

fig = plt.figure(figsize=(6, 2.5))
ax = fig.add_subplot(
    111, xlabel='Distance traveled [m]',
    ylabel='Rotation error [in Degrees]')
pu.boxplot_compare(ax, distances, rel_rot_error, labels, colors)
fig.tight_layout()
plt.grid(True)
plt.ylim([0,10])
fig.savefig(args.results_directory+args.bag_name+'_rot_drift.png', bbox_inches="tight")




plt.show()

mean_before = np.mean(trans_vec_before)/1000
mean_after = np.mean(trans_vec_after)/1000
mean_rot_before = np.mean(e_rot_norm_before)
mean_rot_after = np.mean(e_rot_norm_after)

print('error before leaving the environment is {}: \nerror after return is {}:'.format(
    mean_before,mean_after))


print('Rot error before leaving the environment is {}: \nRot error after return is {}:'.
    format(mean_rot_before,mean_rot_after))
```

Listing 4: Comparison Method For unknown environement

```
import numpy as np
import xmath
import reading_utilities as ru
import absolute_error as abs

def discatnce_from_start(length):
    distances = np.diff(length[:, 0:3], axis=0)
    distances = np.sqrt(np.sum(np.multiply(distances, distances), 1))
    distances = np.cumsum(distances)
    distances = np.concatenate(([0], distances))
    return distances


def boxplot_disctance(max_dist_diff,p_es,p_gt,q_es,q_gt,accum_distances,file_name):
    x = np.array([1,2.5,5,10])
    sub_l = []
    error_list = []
    for l in x:
        sub_l.append(l)
        e_trans, e_trans_perc, e_rot,error_list = relative_calculations(l,max_dist_diff,
    p_es,p_gt,q_es,q_gt,accum_distances,error_list,file_name)
    return sub_l,e_trans, e_trans_perc, e_rot,np.array(error_list)


def compute_comparison_indices_length(distances, dist, max_dist_diff):
    max_idx = len(distances)
    comparisons = []
    for idx, d in enumerate(distances):
        best_idx = -1
        error = max_dist_diff
        for i in range(idx, max_idx):
            if np.abs(distances[i]-(d+dist)) < error:
```

```
                best_idx = i
                error = np.abs(distances[i] - (d+dist))
        if best_idx != -1:
            comparisons.append(best_idx)
    return comparisons


def compute_relative_error(p_es, p_gt, q_es, q_gt,file_name, max_dist_diff=-1):

    accum_distances = discatnce_from_start(p_gt)
    sub_l,e_trans, e_trans_perc, e_rot,error_list = boxplot_disctance(max_dist_diff,p_es,
    p_gt,q_es,q_gt,accum_distances,file_name)
    return sub_l,e_trans, e_trans_perc, e_rot,error_list


def relative_calculations(subtraj_length,max_dist_diff,p_es,p_gt,q_es,q_gt,
    accum_distances,error_list,file_name):
    Tcm = np.identity(4)
    if max_dist_diff <= 0:
        max_dist_diff = 0.2 * subtraj_length
    errors, e_trans, e_trans_perc, e_rot = compute_relative_error_values(p_es, p_gt, q_es
    , q_gt,

    subtraj_length,

    max_dist_diff, accum_distances, Tcm)

    trans_error = abs.compute_error_types(e_trans)
    trans_error_perc = abs.compute_error_types(e_trans_perc)
    rot_error = abs.compute_error_types(e_rot)
    ru.writing_relative_errors(trans_error,rot_error,trans_error_perc,subtraj_length,
                               file_txt="/home/gunisha/Alignment/benchmarking_experiments
    /errors_new/"+file_name+"_error at{} length".format(subtraj_length))
    error_list.append([e_trans])
    error_list.append([e_trans_perc])
    error_list.append([e_rot])

    return e_trans, e_trans_perc, e_rot,error_list


def compute_relative_error_values(p_es,p_gt,q_es,q_gt,subtraj_length,max_dist_diff,
    distances,T_cm):
    if len(distances) == 0:
        distances = discatnce_from_start(p_gt)
    comparisons = compute_comparison_indices_length(distances,subtraj_length,
    max_dist_diff)
    print('number of samples ='+str(len(comparisons)))
    T_mc = np.linalg.inv(T_cm)
    errors = []
    for idx,c in enumerate(comparisons):
        if not c == -1:
            T_c1 = get_rigid_body_trafo(q_es[idx, :], p_es[idx, :])
            T_c2 = get_rigid_body_trafo(q_es[c, :], p_es[c, :])
            T_c1_c2 = np.dot(np.linalg.inv(T_c1), T_c2)


            T_m1 = get_rigid_body_trafo(q_gt[idx, :], p_gt[idx, :])
            T_m2 = get_rigid_body_trafo(q_gt[c, :], p_gt[c, :])
            T_m1_m2 = np.dot(np.linalg.inv(T_m1), T_m2)

            T_m1_m2_in_c1 = np.dot(T_cm,np.dot(T_m1_m2,T_mc))
            T_error_in_c2 = np.dot(np.linalg.inv(T_m1_m2_in_c1),T_c1_c2)
            T_c2_rot = np.eye(4)
            T_c2_rot[0:3,0:3] = T_c2 [0:3,0:3]
            T_error_in_w = np.dot(T_c2_rot,np.dot(
                T_error_in_c2,np.linalg.inv(T_c2_rot)))
```

```
            errors.append(T_error_in_w)

            error_trans_norm = []
            error_trans_perc = []
            error_rot = []
            for e in errors:
                tn = np.linalg.norm(e[0:3,3])
                error_trans_norm.append(tn)
                error_trans_perc.append(tn/subtraj_length *100)
                error_rot.append(compute_angle(e))

    return errors, np.array(error_trans_norm), np.array(error_trans_perc), np.array(
    error_rot)


def get_rigid_body_trafo(quat, trans):
    T = np.eye(4)
    T[0:3,0:3] = xmath.matrix.setQuat(quat)
    T[0:3,3] = trans
    return T

def compute_angle(transform):

    return np.arccos(
        min(1, max(-1, (np.trace(transform[0:3, 0:3]) - 1)/2)))*180.0/np.pi
```

Listing 5: Unit Test between SVD and Horns Method

```
import numpy as np
import xsens_common as xs
import xmath
import math



def align_trajectory(pos_gt,pos_es,method):
    assert np.shape(pos_es)[1] == 3
    assert np.shape(pos_gt)[1] == 3

    s = 1
    if method =='svd':
        s,R,t = align_umeyama(pos_gt,pos_es)
    elif method == 'Horn':
        s,R,t = horns_method(pos_gt,pos_es)
    else:
        assert False, 'Unknown Alignment Method'
    return s,R,t


def align_umeyama(pos_gt,pos_es):
    s=1
    # substract mean to center the data points around origin
    mu_es = pos_es.mean(0)
    mu_gt = pos_gt.mean(0)
    pos_es_zerocentered = pos_es - mu_es
    pos_gt_zerocentered = pos_gt - mu_gt
    n = np.shape(pos_gt)[0]

    # correlation
    C = 1.0/n*np.dot(pos_gt_zerocentered.transpose(), pos_es_zerocentered)
    sigma2 = 1.0/n* np.multiply(pos_es_zerocentered, pos_es_zerocentered).sum()
    U_svd, D_svd, V_svd = np.linalg.linalg.svd(C)
    D_svd = np.diag(D_svd)
    V_svd = np.transpose(V_svd)

    S = np.eye(3)
    if(np.linalg.det(U_svd)*np.linalg.det(V_svd) < 0):
```

```
        S[2, 2] = -1
    R = np.dot(U_svd, np.dot(S, np.transpose(V_svd)))

    s = 1.0/sigma2*np.trace(np.dot(D_svd, S))

    t = mu_gt-np.dot(R, mu_es)
    # print s,R,t

    return s, R, t

def horns_method(pos_gt,pos_es):
    s=1
    mu_es = pos_es.mean(0)
    mu_gt = pos_gt.mean(0)
    pos_es_zerocentered = pos_es - mu_es
    pos_gt_zerocentered = pos_gt - mu_gt
    n = np.shape(pos_es)[0]

    # correlation
    C = np.dot(pos_gt_zerocentered.transpose(), pos_es_zerocentered).T

    M = np.dot(C.transpose(),C)
    e_values,e_vec = np.linalg.eig(M)
    l1 = np.sqrt(e_values[0])
    l2 = np.sqrt(e_values[1])
    l3 =  np.sqrt(e_values[2])
    v1 = np.array(e_vec[:,0])
    v1 = v1.reshape(3,1)
    v2 = e_vec[:,1]
    v2 = v2.reshape(3,1)
    v3 = e_vec[:,2]
    v3 = v3.reshape(3,1)

    S = (1/l1)*(np.dot(v1,v1.transpose())) + (1/l2)*(np.dot(v2,v2.transpose())) + (1/l3)
    *(np.dot(v3,v3.transpose()))
    R = np.dot(C,S)
    s = np.sqrt(np.sum(pos_gt_zerocentered**2)/np.sum(pos_es_zerocentered**2))
    t = mu_gt-(s)*np.dot(R, mu_es)

    return s, R, t


def apply_alignment(pos_es,quat_es,s,R,t):
    pos_es_aligned = []

    quat_rot = xmath.quaternion.setRmat(R)
    for i in range(np.shape(pos_es)[0]):
        pos_es_al = s * xmath.vector.setRotateVecByQuat(np.array(pos_es[i,:]),quat_rot)+t
        pos_es_aligned.append(pos_es_al)
    pos_es_aligned = np.array(pos_es_aligned)
    quat_es_aligned = xs.quatMulQuatList(quat_rot,xs.quatListMulInvQuat(quat_es,quat_rot)
    )
    quat_es_aligned = xs.quatListNormalize(np.array(quat_es_aligned))
    return pos_es_aligned,quat_es_aligned


def rot2eul(R):
    beta = np.degrees(-np.arcsin(R[2,0]))
    alpha = np.degrees(np.arctan2(R[2,1]/np.cos(beta),R[2,2]/np.cos(beta)))
    gamma = np.degrees(np.arctan2(R[1,0]/np.cos(beta),R[0,0]/np.cos(beta)))
    return np.array((alpha, beta, gamma))
def quaternion_to_euler(quat):
    w = quat[:, 0]
    x = quat[:, 1]
    y = quat[:, 2]
    z = quat[:, 3]
```

```
    euler = []
    for i in range(len(w)):
        sqw = w[i] * w[i]
        sqx = x[i] * x[i]
        sqy = y[i] * y[i]
        sqz = z[i] * z[i]
        unit = sqx+sqy+sqz+sqw
        test = x[i]*y[i]+z[i]*w[i]
        Y = math.degrees(math.atan2(2*y[i]*w[i]-2*x[i]*z[i],sqx-sqy-sqz+sqw))
        Z = math.degrees(math.asin(2*test/unit))
        X = math.degrees(math.atan2(2*x[i]*w[i]-2*y[i]*z[i],-sqx+sqy-sqz+sqw))
        if (test>.499*unit):
            Y = math.degrees(2*math.atan2(x[i],w[i]))
        if(test<-.499*unit):
            Y = math.degrees(-2*math.atan2(x[i],w[i]))
        euler.append([X,Y,Z])
    euler = np.array(euler)
    return euler
def store_align_factors(s,R,t):
    factors = dict()
    factors['scale'] = s
    angles = rot2eul(R)
    factors['Rotation Angles'] = angles
    factors['translation'] = t
    return factors
```

Listing 6: SVD and Horn Method Implementation

```


import numpy as np
import alignment_util as au
import reading_utilities as ru
import xsens_common as xs
import xmath
import unittest
import numpy.testing as npt


class TestTrajEval(unittest.TestCase):

    def setUp(self) :
        FOLDER_PREFIX = '../../Alignment/data_collection/'
        bag_name = FOLDER_PREFIX + 'trial.bag'
        vive_name = '/vive/LHR_EF94BE39_pose'
        self.pose_gt, self.t_gt = ru.read_msgs(vive_name, bag_name)
        self.pos_gt, self.quat_gt, self.t_gt = ru.get_pos_quat_time(self.pose_gt)
        self.generate_estimate()

    def generate_estimate(self):
        self.pos_es = []
        self.s = 1
        self.t = np.random.rand(3,)
        self.rot_array = np.random.rand(4,)
        self.rot_array = (self.rot_array)/np.linalg.norm(self.rot_array)
        self.R = xmath.matrix.setQuat(self.rot_array)
        for i in range(np.shape(self.pos_gt)[0]):
            self.pos = self.s * xmath.vector.setRotateVecByQuat( self.pos_gt[i,:],self.
    rot_array) + self.t
            self.pos_es.append(self.pos)
        self.pos_es = np.array(self.pos_es)
        self.quat_es = xs.quatListMulQuat(self.quat_gt,self.rot_array)
        self.add_noise()
    def add_noise(self):
        self.mu_pos, self.sigma_pos = 0, .001
        self.noise_pos = np.random.normal(self.mu_pos, self.sigma_pos, [np.shape(self.
    pos_gt)[0], np.shape(self.pos_gt)[1]])
```

```
        self.pos_es = self.pos_es + self.noise_pos


    def test_align_svd(self):
        s_svd, R_svd, t_svd = au.align_trajectory(self.pos_gt,self.pos_es, 'svd')
        # pos_es_aligned_svd, quat_es_aligned_svd, pos_es_aligned_unscaled_svd = au.apply_alignment(self.pos_es, self.quat_es,
        #
               self.s_svd, self.R_svd, self.t_svd)
        self.compare_rotation(self.R,R_svd)
        self.compare_translation(self.t,t_svd,R_svd)

    def test_align_horn(self):
        s_h,R_h, t_h = au.align_trajectory(self.pos_gt, self.pos_es, 'svd')
        # pos_es_aligned_h, quat_es_aligned_h, pos_es_aligned_unscaled_h = au.apply_alignment(self.pos_es,self.quat_es,
        #
                  self.s_h, self.R_h,
        #
                  self.t_h)
        self.compare_translation(self.t, t_h, R_h)
        self.compare_rotation(self.R, R_h)


    @staticmethod
    def compare_rotation(R_true, R_est):
        npt.assert_array_almost_equal(R_true,R_est.T,decimal=4)

    @staticmethod
    def compare_translation(t_true,t_est, R_est):
        t_true = -t_true
        t_est =  np.dot(R_est.T,t_est)
        npt.assert_array_almost_equal(t_true,t_est,decimal=4)


if __name__ == '__main__':
    unittest.main()
```

Listing 7: Unit Test between SVD and Horns Method

References

[1] Fusco, Giovanni, and James M. Coughlan. "Indoor localization using computer vision and visual-inertial odometry." In International Conference on Computers Helping People with Special Needs, pp. 86-93. Springer, Cham, 2018.

[2] Wang, Yuqin, Biyu Tang, Lingxiang Zheng, and Zhichao Lin. "Autonomous Integrity Monitoring of a vision based pedestrian dead reckoning system."

[3] Rebecq, Henri, Timo Horstschaefer, and Davide Scaramuzza. "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization." (2017): 1-8.

[4] Li, Mingyang, and Anastasios I. Mourikis. "High-precision, consistent EKF-based visual-inertial odometry." The International Journal of Robotics Research 32, no. 6 (2013): 690-711.

[5] Usenko, Vladyslav, Jakob Engel, Jörg Stückler, and Daniel Cremers. "Direct visual-inertial odometry with stereo cameras." In 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1885-1892. IEEE, 2016.

[6] Ramezani, Milad, Debaditya Acharya, Fuqiang Gu, and Kourosh Khoshelham. "Indoor positioning by visual-inertial odometry." ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 4 (2017): 371.

[7] Weinstein, Aaron, Adam Cho, Giuseppe Loianno, and Vijay Kumar. "Visual inertial odometry swarm: An autonomous swarm of vision-based quadrotors." IEEE Robotics and Automation Letters 3, no. 3 (2018): 1801-1807. for applications

[8] Howie M. Choset; Seth Hutchinson; Kevin M. Lynch; George Kantor; Wolfram Burgard; Lydia E. Kavraki; Sebastian Thrun (2005). Principles of Robot Motion: Theory, Algorithms, and Implementation. MIT Press. pp. 285–. ISBN 978-0-262-03327-5. autonomous driving

[9] https://www.intelrealsense.com/tracking-camera-t265

[10] Umeyama S (1991) Least-squares estimation of transformation parameters between two point patterns. IEEE Trans Pattern Anal Machine Intell 13:376–380

[11] Horn BKP, Hilden HM, Negahdaripour S (1988) Closed-form solution of absolute orientation using orthonormal matrices. J Opt Soc Am Ser A 5:1127–1135