SURGE 2019 PROJECT REPORT
Indian Institute of Technology, Kanpur

Project : Visual Simultaneous Localization And Mapping (SLAM)

Date of Submission:

July 5, 2019

Submitted by:

Piyush Patel

Surge Rollno. S190102

Department of Civil Engineering IIT KANPUR

Under the guidance of:

Professor Salil Goel

Department of Civil Engineering IIT KANPUR

Contents

1 About Myself

Myself Piyush Patel, and I am a third year Undergraduate student in Department of Civil Engineering at IIT Kanpur. You can know more about my work by visiting my Github Account and my LinkedIn Account

2 Abstract

Simultaneous localization and mapping (SLAM) is a technique applied in artificial intelligence mobile robot for a self-exploration in numerous geographical environment. SLAM becomes fundamental research area in recent days as it promising solution in solving most of problems which are related to the self-exploratory oriented artificial intelligence mobile robot field. For example, the capability to explore without any prior knowledge on environment and without any human interference. The unique feature in SLAM is that the process of mapping and localization is done concurrently and recursively. Since SLAM introduction, many SLAM algorithms have been proposed to apply SLAM technique in real practice. The aim of this project is to provide a low cost solution in terms of computation and easy to deploy, ready made package.

3 About Project

3.1 Introduction:

The challenges faced in a self-exploratory oriented autonomous mobile robot is the environment factors which have numerous complex geographical landmarks and obstacles. Other challenges are capability of the mobile robot to explore and navigate without any knowledge of unknown environment, to generate its own map for the environment, to be able to recognize its own position, landmark and any obstacles, to make decision based on the new environment data received and to be able to navigate through the unknown environment without human interference.

Keeping in mind that indoor and remote environments are blind spots for our GPS, also in outdoor environment the capability of GPS to precisely locate an object is of meters level accuracy. Even for the best GPS modules (RTK GPS for example) can locate object with 50 centimeters inaccuracy in outdoor environment, which is not a good deal for a self-exploring MAV . So, our localization(the ability to know one's position with respect to environment) cannot be based on GPS entirely, we need to localize the camera locally, Since we are already using a sensor(camera) to do the mapping the localization can also be performed using the camera. Thus, localization and mapping are coupled. This kind of procedure is called Visual SLAM (Simultaneous Localization and Mapping).

In this project I have used a RGBD camera for the above purpose. The camera had an inbuild IMU.

3.2 Localization

Once we have a map of the environment, Camera/ MAV also needs to know its current location within the map, which may or may not be geo-referenced. The sensors used additionally for this purpose are accelerometer and gyroscope along with the camera(localization using feature detection and matching). The camera uses feature detection and tracking to calculate the velocity, orientation and position of the camera in the unknown environment. The IMU data has an accelerometer which is used to get the ground's direction.

3.3 Mapping

Using onboard sensors, the vehicle must perceive the environment and build a map of it, in order to identify objects such as obstacles and targets. The sensors used for this purpose usually include laser scanners and

different types of cameras. In this project we preferred to use an RGBD camera over a monocular camera to do the job. Since stereo camera's are much more accurate in estimating the distances of object than monocular cameras.

4 System Architecture

4.1 Hardware Description

4.1.1 Intel Realsense D435i

Intel RealSense depth camera D435i combines the robust depth sensing capabilities with the addition of an inbuilt inertial measurement unit (IMU). It has inbuilt accelerometer and gyroscope, giving the acceleration and angular velocity respectively. Other available depth cameras in market like Kinect are too heavy to be used on UAVs.
Few advantages of Intel Realsense are:

- It is lightweight and small.
- It has inbuilt Imu which give synchronous time stamped data to align with high quality depth maps
- It is ROS compatible, which means it can be used in Robotics applications
- It can give RGB images of resolution 1920 x 1080 at 30fps
  Disadvantage :
- Its depth image and pointcloud data is not as accurate as kinect's camera

The whole algorithm was developed using Intel's i5 8th gen processor with 8GB RAM. The package is still not tested on real UAV.

4.2 Software Description

The Realsense Camera publishes colored RGB images and depth images over ROS on /camera/color/image_raw and /camera/depth/image_rect_raw rostopics respectively. It also publishes RGB pointcloud on rostopic /camera/color/depth/points.

The accelerometer and gyroscope's data is published on /camera/accel/sample and /camera/gyro/sample respectively. Both accelerometer's and gyroscope's data is first combined under single rostopic /imu/data_raw which is then used to calculate the orientation of the camera with respect to the fixed (world) frame using imu_filter_madgwick algorithm.

The RGB and depth image rostopics are subscribed by the orb_slam2_rgbd package, using feature detection and matching in two simultaneous image frames it determines the pose of the camera, which is published on /orb_slan2_rgbd/pose

The octomap subscribes the pointcloud data from realsense camera and uses transformations published over ROS to generate a probabilistic 3D occupancy grid map in realtime.

A detailed flow chart of rostopics and rosnodes used in the system is given in the following chart

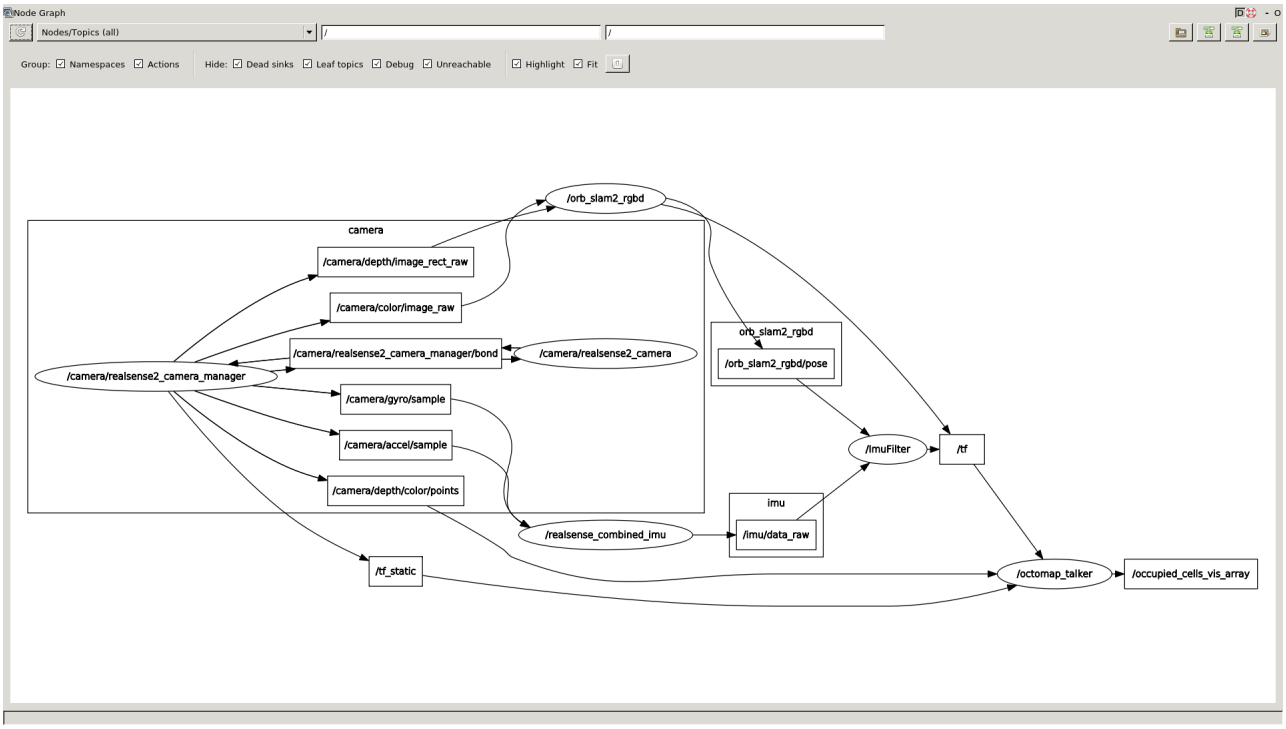


Figure 1: Graph for ROS Topics and ROS Nodes

A detailed representation of the nodes is given here in the table below:

TOPIC SUBSCRIBED	NODE	TOPIC PUBLISHED
rgb and depth images	orb_slam2_rgbd	UAV's position in world frame
accelerometer and gyroscope's data	realsense_imu_comb	combined imu data without orientation
combined imu data without orientation	imu_filter_madgwick	imu with orientation
point cloud and transformations	octomap	occupied markers
imu data with orientation and UAV's position from orb_slam2_ros	kalman_filter	filtered data

4.3 Integration : ROS

The Robot Operating System (ROS) [4] is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.
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. Even though ROS is still a framework that isn't a standalone OS or an RTOS and isn't the only framework for robots, it seems to be adopted widely and have a large developers community.
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. More information can be found on ROS Website

5 Package Description

5.1 Orb Localization

5.1.1 What is Orb?

Orb [3] is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. It is faster than SIFT and SURF feature detector[5].

To know the local position of the camera in the environment we can double integrate accelerometer's data to get the position but during the course of time the uncertainty gets accumulated and the position gets highly inaccurate. This amount of uncertainty in indoor environment can lead to crashing of UAV. To overcome this problem visual localization was needed. The ORB Localization solves this problem very well:

- The program reads a pair of images – one of which is the colour image taken by the camera, while the other is the depth-map generated using the stereo image pair.
- The program looks for a special kind of features called an ORB feature within the colour image. These features can be detected very quickly and can be uniquely identified under most circumstances.
- The depth-map is used to find the location of each of these features in 3D space (with respect to the current position).
- Each ORB feature is mapped against the features already present in the map in order to find the correct position. ORB features can be matched very quickly in contrast to many popular image descriptors.
- An EKF (Extended-Kalman-Filter) based approach is used to estimate the new pose of the camera, based on the location of the matched features.
- The (unmatched) new features are added to this map, thus expanding the map slightly

This process is repeated for each frame. Since ORB features can be detected and matched very quickly, this method is very efficient and can thus work in real-time.

Although this a SLAM algorithm, the map generated by it is not useful for our purposes (as will be explained in a later section). Thus, we use this only for localization. A different approach is used for mapping the environment.

5.1.2 How does ORB works?

FAST (Features from Accelerated and Segments Test): Given a pixel p in an array fast compares the brightness of p to surrounding 16 pixels that are in a small circle around p. Pixels in the circle is then sorted into three classes (lighter than p, darker than p or similar to p). If more than 8 pixels are darker or brighter than p than it is selected as a keypoint. So keypoints found by FAST gives us information of the location of determining edges in an image.

However, FAST features do not have an orientation component and multiscale features. So orb algorithm uses a multiscale image pyramid. An image pyramid is a multiscale representation of a single image, that consist of sequences of images all of which are versions of the image at different resolutions. Each level in

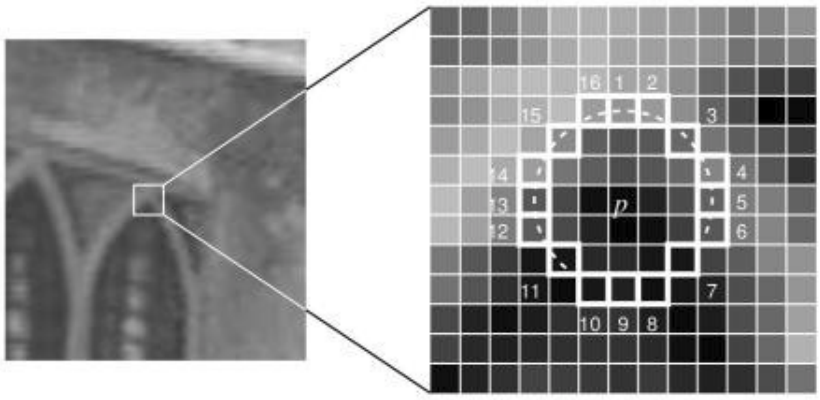


Figure 2: FAST

the pyramid contains the downsampled version of the image than the previous level. Once orb has created a pyramid it uses the fast algorithm to detect keypoints in the image. By detecting keypoints at each level orb is effectively locating key points at a different scale. In this way, ORB is partial scale invariant.

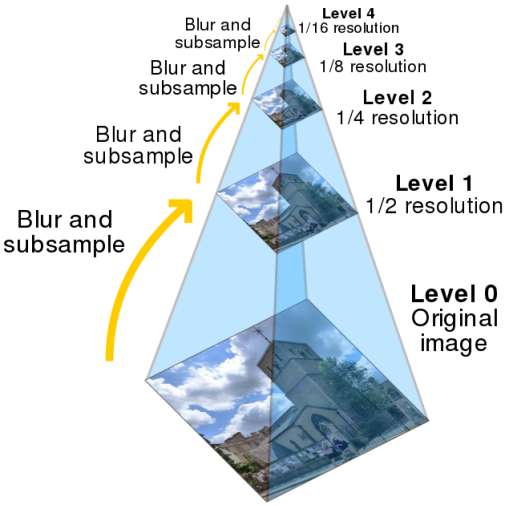


Figure 3: Image Prramid of FAST feature detector

BRIEF : Brief takes all keypoints found by the fast algorithm and convert it into a binary feature vector so that together they can represent an object. Binary features vector also know as binary feature descriptor is a feature vector that only contains 1 and 0. Brief deals with the image at pixel level so it is very noise-sensitive. By pre-smoothing the patch, this sensitivity can be reduced, thus increasing the stability and repeatability of the descriptors.

Computation of R, t : In order to determine the rotation matrix R and translation vector t, it uses Levenberg-Marquardt non-linear least squares minimization to minimize the following sum:

$$\epsilon = \sum_{\mathcal{F}^t, \mathcal{F}^{t+1}} (\mathbf{j_t} - \mathbf{PTw_{t+1}})^2 + (\mathbf{j_{t+1}} - \mathbf{PT^{-1}w_t})^2$$

Figure 4: Levenberg-Marquardt Equation

Ft,Ft+1 : Features in the rgb image at time t and t+1 jt,jt+1: 2D Homogeneous coordinates of the features Ft,Ft+1
wt,wt+1: 3D Homogeneous coordinates of the features Ft,Ft+1
P: 34 Projection matrix of left camera
T: 44 Homogeneous Transformation matrix

5.2 Octomap

[1] Point clouds store large amounts of measurement points and hence are not memory efficient. The most simple 3D occupancy map representation is a 3D voxel grid. This is basically a binary array with 3 dimensions. Each element of the array corresponds to a cubical volume in 3D space. Each element can take a value from 0 to 1 depending on the calculated occupancy probablity. If the probablity of the element exceeds a certain threshold (say 0.5) the element is filled and maked as occupied.

However, such a representation faces two drawbacks:

- It has a very large memory requirement (a cubical room with a side of 10m will require 8 million voxels if the size of each voxel is 5cm).
- It cannot be updated easily – once the size of the map grows and becomes larger than the initial grid, the entire map must be rearranged (this is because memory is 1 dimensional, but the array is 3 dimensional).

Thus, OctoMap uses a special data structure for storing occupancy. This data structure is called an Octree. This resolves both of the issues mentioned above (the details are given in the original paper). Additionally, it provides advantages such as multi-resolution maps. Octomaps are based on an octree and is designed to meet the following requirements:

- **Full 3D model:** The map is able to model arbitrary environments without prior assumptions about it. The representation models occupied areas as well as free space. Unknown areas of the environment are implicitly encoded in the map. While the distinction between free and occupied space is essential for safe robot navigation, information about unknown areas is important, e.g., for autonomous exploration of an environment.
- **Updateable:** It is possible to add new information or sensor readings at any time. Modeling and updating is done in a probabilistic fashion. This accounts for sensor noise or measurements which result from dynamic changes in the environment, e.g., because of dynamic objects. Furthermore, multiple robots are able to contribute to the same map and a previously recorded map is extendable when new areas are explored.
- **Compact:** The map is stored efficiently, both in memory and on disk. It is possible to generate compressed files for later usage or convenient exchange between robots even under bandwidth constraints.

The basic procedure of octomap is:

- The algorithm takes a (localized) 3D point cloud as input.

- For each point present in the point cloud, it traces a ray from the origin of the point cloud to the point. In our case, this origin is the optical centre of the camera (so that the ray represents the path traced by the light entering the camera).
- For every voxel that intersects this ray, its occupancy value is probabilistically updated, in order to indicate that it is a free region.
- The voxel at the end of the ray has its occupancy probabilistically updated so as to reflect the fact that it is occupied.

This procedure is repeated for every point in the point cloud. Then this entire process is repeated for the next point cloud and so on.

The details of how occupancy values are probabilistically are as follows. In essence, the occupancy values stored at each voxel correspond to the log-likelihood of its occupancy. A naïve sensor model is assumed for the camera (constant probability of hitting or missing a surface). If we use the equations for conditional probability based on this model, it can be shown that the log-likelihood of occupancy increases/decreases by a fixed value based on whether the voxel is occupied/unoccupied.

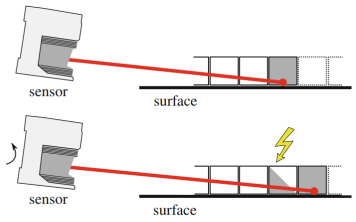


Fig. 10 A laser scanner sweeps over a flat surface at a shallow angle by rotating. A cell measured occupied in the first scan (*top*) is updated as free in the following scan (*bottom*) after the sensor rotated. Occupied cells are visualized as *gray boxes*, free cells are visualized in *white*

(a) How occupacy is determined

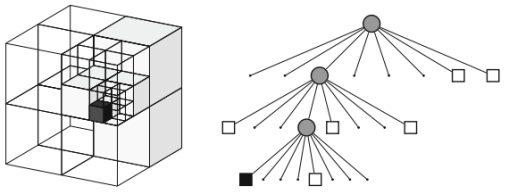


Fig. 2 Example of an octree storing free (*shaded white*) and occupied (*black*) cells. The volumetric model is shown on the *left* and the corresponding tree representation on the *right*

Fig. 3 By limiting the depth of a query, multiple resolutions of the same map can be obtained at any time. Occupied voxels are displayed in resolutions 0.08, 0.64, and 1.28 m

(b) Octree Representation

What is an octree ?

An octree is a hierarchical data structure for spatial subdivision in 3D. Each node in an octree represents the space contained in a cubic volume, usually called a voxel. This volume is recursively subdivided into eight sub-volumes until a given minimum voxel size is reached, as illustrated in Fig. 2. The minimum voxel size determines the resolution of the octree. Since an octree is a hierarchical data structure, the tree can be cut at any level to obtain a coarser subdivision if the inner nodes are maintained accordingly. An example of an octree map queried for occupied voxels at several resolutions is shown here:

5.3 Kalman Filter

You can use a Kalman filter in any place where you have uncertain information about some dynamic system, and you can make an educated guess about what the system is going to do next. Even if messy reality comes

along and interferes with the clean motion you guessed about, the Kalman filter will often do a very good job of figuring out what actually happened.

Kalman filters are ideal for systems which are continuously changing. They have the advantage that they are light on memory (they don't need to keep any history other than the previous state), and they are very fast, making them well suited for real time problems and embedded systems.

6 Alogrithm Description

The algorithm developed and used in this package is described in detail here. The source code can be found at github.com/da-piyushpatel/slam

6.1 Camera Model

A camera projects a 3D surrounding to a 2D image. The project equation corresponding to this transformation is:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$x = \frac{x'}{w}, y = \frac{y'}{w}$$

Figure 5: Pinhole camera model equation

Here, X, Y, Z are the real-world coordinates, and x, y are the image coordinates. The first matrix is called the intrinsic matrix and is often calibrated using a chessboard with known dimensions. The second matrix is a rotation + translation matrix, corresponding to the transformation from the world frame to the camera frame. The parameters f_x and f_y correspond to the focal length of the camera, while c_x and c correspond to the location of the optical centre on the image sensor. It is important to note that this is an inherently non-recoverable transformation since it transforms a point in 3D space to a pixel location on a 2D image (each pixel can correspond to any point along a line). Thus, we require extra information in order to recover the 3D representation, so that we can create a map. This brings us to the next step – depth generation using stereo images.

6.2 Depth map from Stereo Image Pair

As humans have capability of depth perception using binocluar vision. A pair of camera rigidly fixed at a certain distance with all its parameters known, can be used to approximate the depth map. Objects closer to the camera subtend greater angle and appear at two different locations in the left and right images, while objects farther away appear to be at the same position in both the images. For this reason stereo camera has limited range.

The alogrithm used for depth estimation is:

- Both images are divided into a square window of size 8 * 8.
- Each square region in both the images is compared. It is matched with the one which has lowest SSD (Sum of Square of differences) of intensity value compared to the window being evaluated. It is assumed that these two windows correspond to the same point in 3D space.

- Once this process is complete we use camera model equations to get a 3D line corresponding to each window.
- For every pair of matched window the intersection of the two lines corresponding to the two windows gives the location of 3d point in the 3D space.

A highly optimized mathematical model is used to perform the depth estimation natively on the realsense camera. A detailed reference material can be found here

6.3 Conversion to 3D point cloud

If you can reverse the camera model equations which are used to project 3D view to 2D view, we can generate a 3D model by combining the depth value of each pixel. This optimized algorithm is natively implemented on the realsense camera itself. The RGBXYZ pointcloud thus generated is shown below:

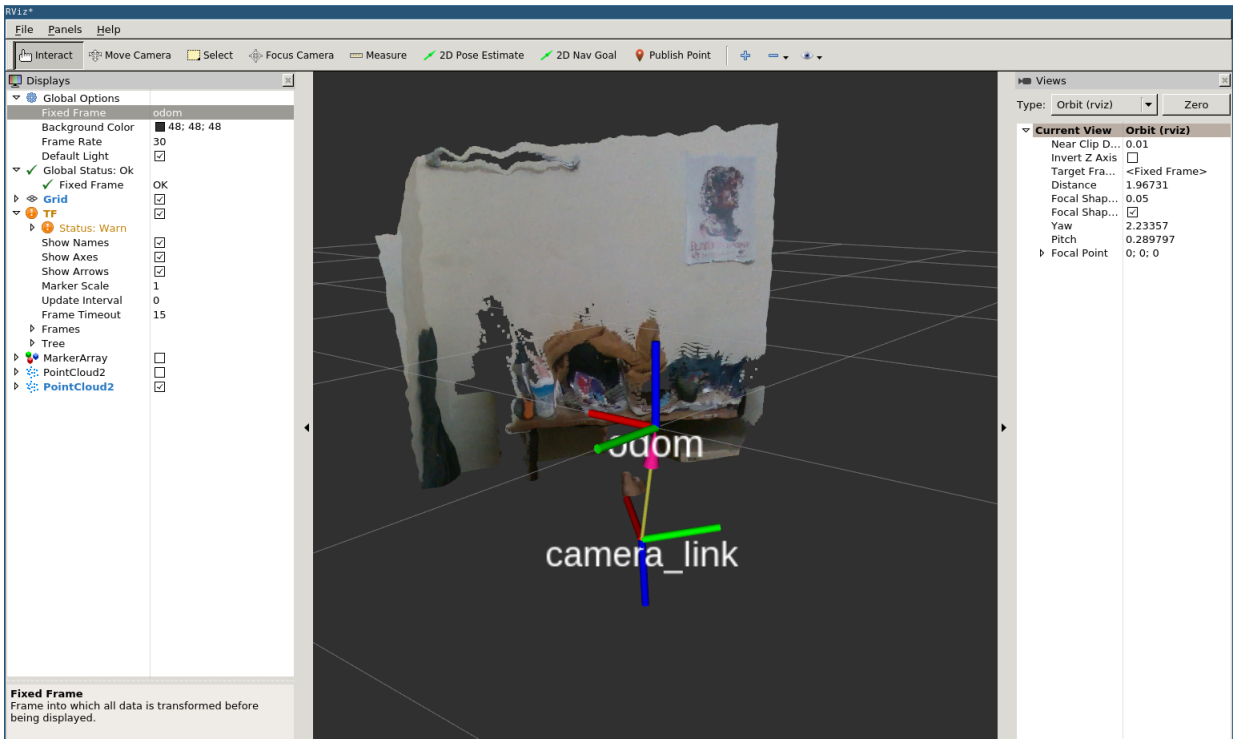


Figure 6: Intel Realsense's PointCloud map

6.4 Calculating camera's orientation

The algorithm used for calculating the orientation of camera using accelerometer's and gyroscope's data is imu_filter_madgwick [2] . The algorithm suggests use of magnetometer also to know the true north and provide more accurate orientation with global alignment. But, due to lack of magnetometer we are only using accelerometer and gyroscope only.

The estimated quaternion is calculated by integrating the quaternion derivative, which is calculated using the angular velocity data of gyroscope. Source

6.5 Transformations

TF is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time. Our imu_tools package publishes the transformation between the camera_link frame (camera frame) and odom frame (world frame).

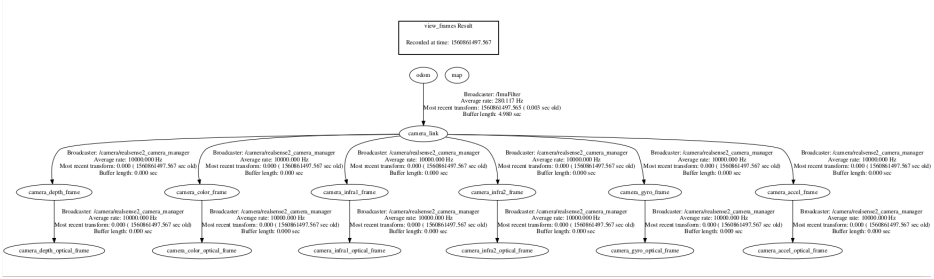


Figure 7: Transformations between various frames

6.6 Feature Detection and Matching

The RGB image captured by the RGB-D camera is converted to grayscale first, and a scale pyramid scheme of the image is then established using Gaussian kernels with different scale factors, such that features can be extracted from each level to enhance robustness. The ORB algorithm utilizes the FAST (features from accelerated segment test) detector to extract feature points from each level of the image scale pyramid. The basic idea of the FAST detector is to determine a feature point by comparing the intensity threshold to the grayscale gradient between a center pixel and pixels in its circular neighborhood .

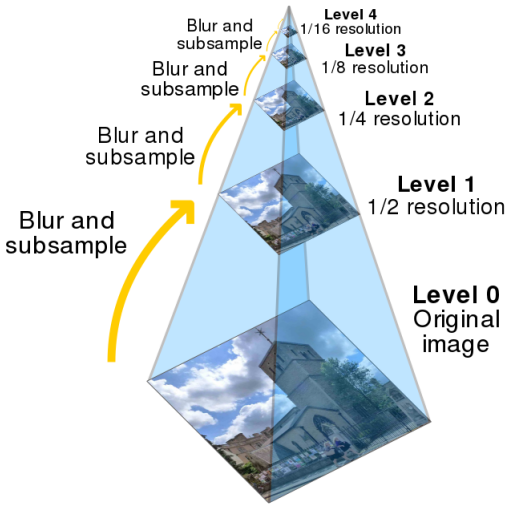


Figure 8: Image Pyramid

Once the features are extracted from consecutive images, the feature matching procedure is performed to generate feature correspondence. Once the 2D image feature matches are extracted by the OFC-ORB procedure, these feature matches are corresponded to 3D-space using their corresponding depth data from

the depth image, yielding two sets of 3D point clouds with known correspondences. Given these consecutive 3D feature correspondences captured at different time steps, the MAV's relative motion from the prior to the subsequent time step can be estimated based on the transformation of the two 3D point clouds. The concept of relative motion estimation is illustrated in the figure below: The optimal solution of R(rotation)

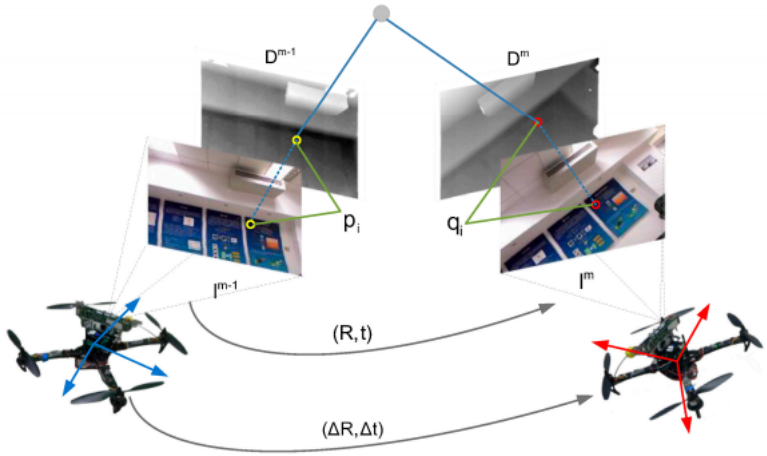


Figure 9: Relative motion Estimation using

and t(translation) in a least-square sense can be obtained by minimizing the following error function:

$$\sum_{i}^{n} || \, p - (Rq + t) \, ||^2$$

It can be concluded from the above calculation procedure that the performance of relative motion estimation relies heavily on the quality of extracted feature correspondences, and it is thus highly sensitive to outliers.

6.7 Linear Kalman Filter

The noisy position estimates are filtered using a Linear Kalman Filter algorithm (**??**). Newtonian mechanics have been used to formulate the system model, with the acceleration varying linearly between two consecutive iterations. This acceleration is measured by the Realsense's on board IMU, and used in the prediction step. Then the predicted data is fused with the incoming position data, in the measurement step. The filtered position estimate is more accurate as compared to the unfiltered position estimate, because data from both the sensors is being used to calculate the belief of the position of the robot. Figure **??** depicts the noisy position data , compared to the filtered data.

Input : (X_{k-1},P_{k-1}, U_k, Z_k, F_k, B_k, Q_k, H_k, R_k): X_{k-1} - initial belief vector ;P_{k-1} - initial covariance matrix; U_k - control vector; Z_k - measurement vector; F_k - state transition model; B_k - control input model; Q_k - process noise covariance; H_k - observation model; R_k - observation noise covariance

Output: (X_k, P_k):X_k - final belief vector; P_k - final covariance matrix

$P_k \longleftarrow$ Identity Matrix; $X_k \longleftarrow$ Identity Vector

repeat

Prediction step

$\hat{X_p} \leftarrow F_k * X_{k-1} + B_k * U_k$

$\hat{P_k} \leftarrow F_k * P_{k-1} * F_k^T + Q_k$

Update Step

$K \leftarrow P_k * H_k^T * (H_k * P_k * H_k^T + R_k)^{-1}$

$X_k \leftarrow \hat{X_k} + K * (Z_k - H_k * \hat{X_k})$

$P_k \leftarrow \hat{P_k} - K * H_k * \hat{P_k}$

until *end of input*;

Figure 10: Linear Kalman Filter Algorithm

7 Results & Validation

The end results obtained from various modules are described here:

7.1 Features extracted and used to determine odometry

The rectified RGB image with the features extracted for the matching and position estimation is shown here. The green squares are the Orb features extracted from the image.

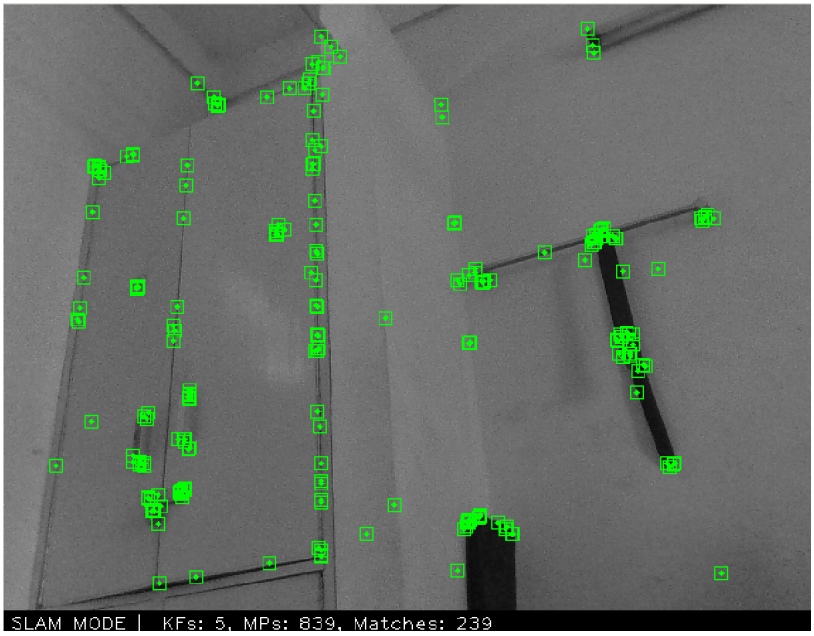


Figure 11: Orb features detected in rgb image

7.2 Accuracy of Visual Odometry

The visual odometry was tested in three cases. In two of the cases the actual path followed by the the camera in world frame is shown in RED and the data generated by Visual Odometry for position estimation is shown in the blue.

The data from the Visual Odometry is not very accurate but it gives a good estimate of the position keeping in mind that there is no GPS available in the area (indoor or remote environment) and the position estimation is done only through the camera and no external sensors are used.

For the following case the camera was moved in a straight line for 140 centimeters along the x axis. The predicted movement along the x-axis is 130 centimeters, it is accurate with an error of 10 centimeters. There is also some error in the predicted movement along the y-axis upto 10 centimeters.

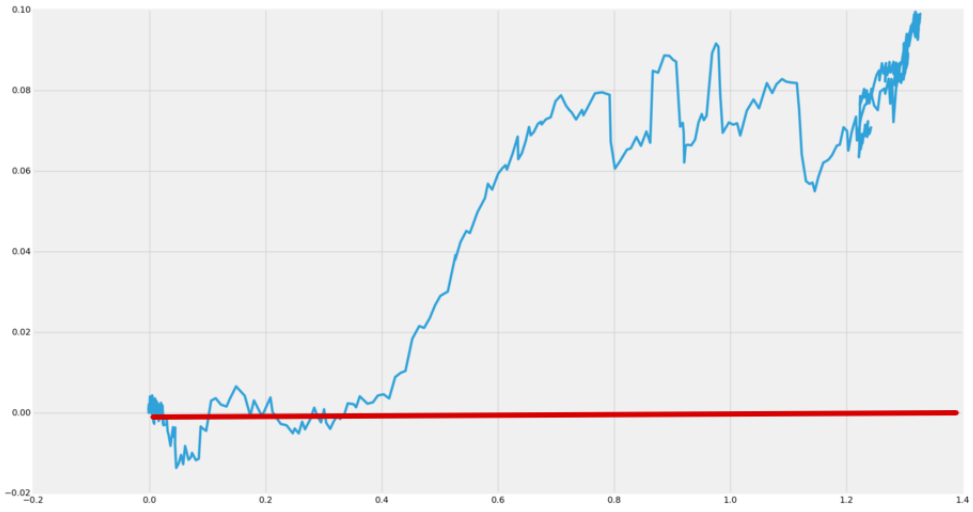


Figure 12: Straight Line: RED is the actual path followed, BLUE is the path determined by Visual odometry

In case of square path, it can be seen there was an initial error in the predicted position which kept on accumulating with the time and the predicted path showed significant deviation in the end.

In the third case the camera was moved in a long gallery for about 22 meters along the x-axis of the camera's frame and then the same path was traced back to the origin. The deviation in recorded data from its initial position was about 13 centimeters.

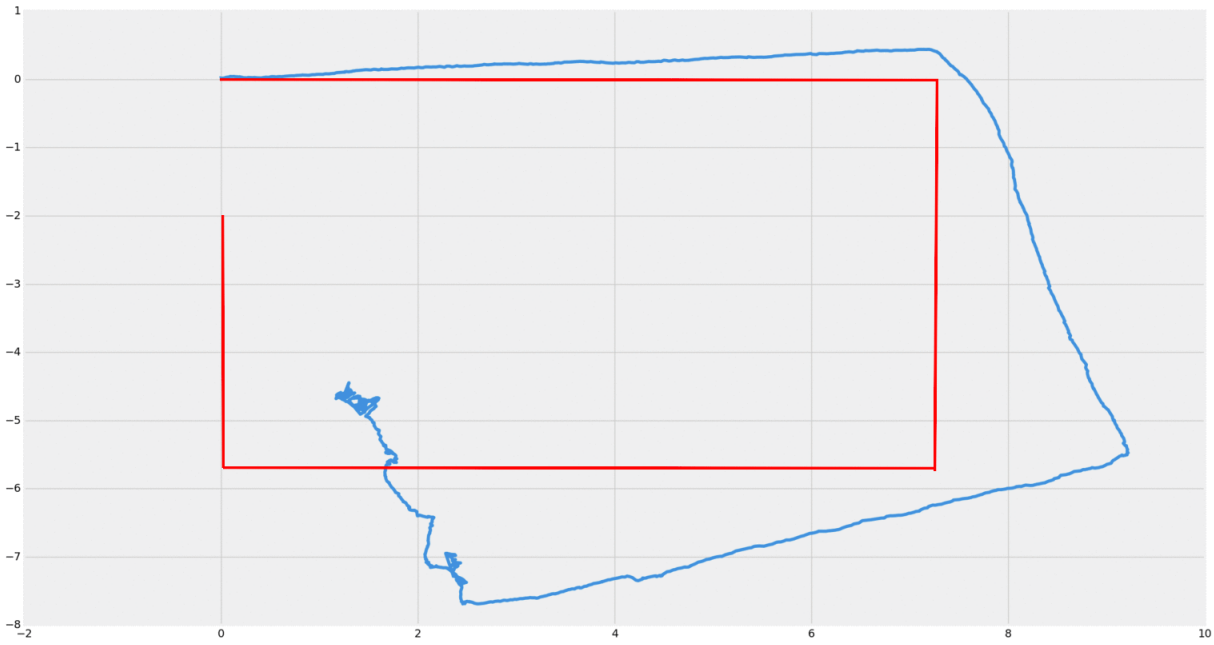
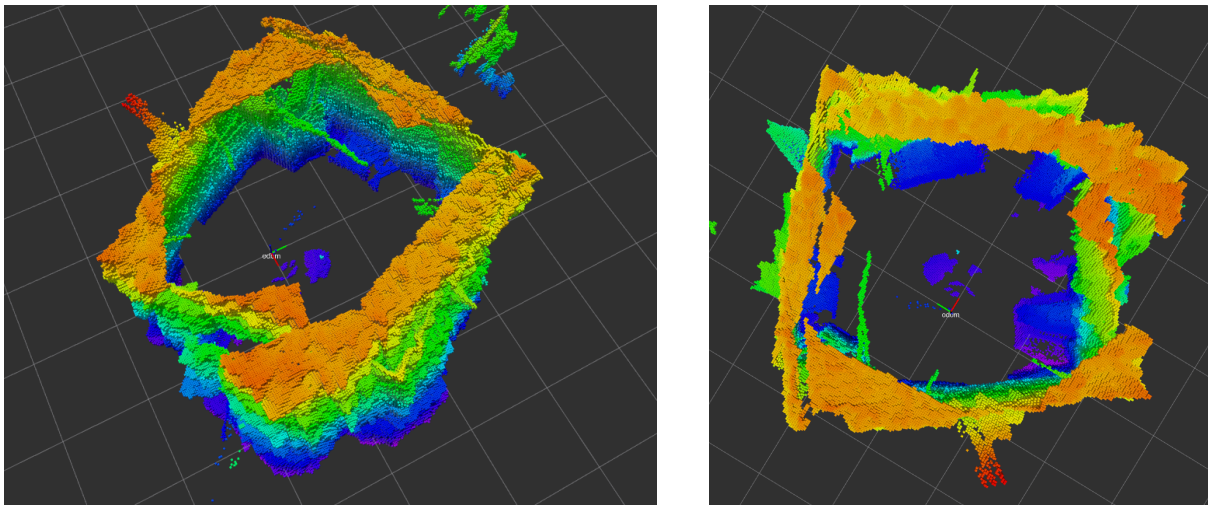


Figure 13: Square path: RED is the actual path followed, BLUE is the path determined by Visual odometry

7.3 3D Octomap

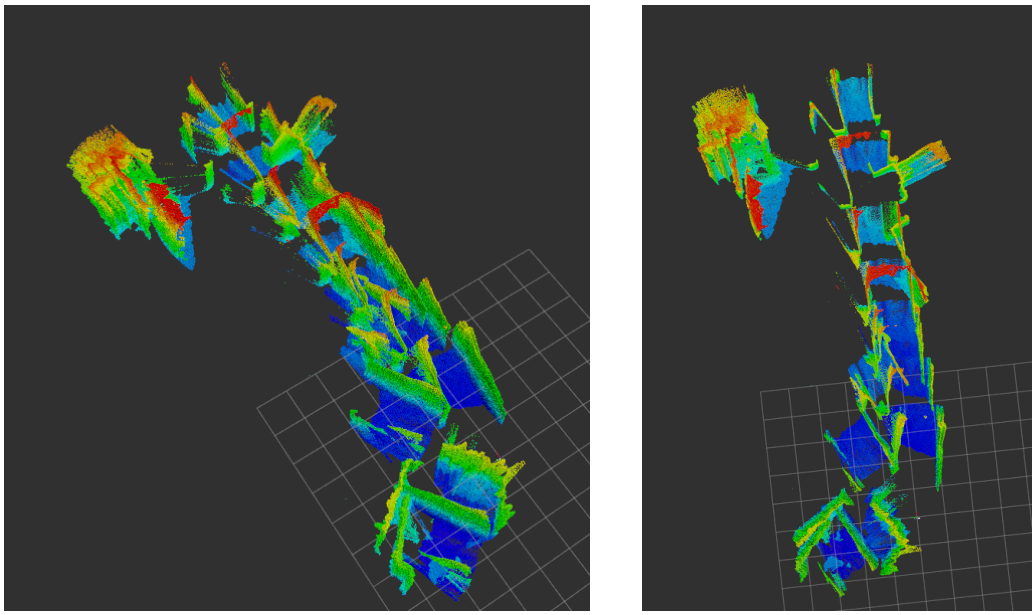
The whole package was tested to create octomap of some locations. These Octomaps may not be very pleasing to eyes initially as they are not RGB coloured and only geometric informations is visible in it. But these maps are still very useful for a exploring Ground or Aerial Vechile whose main aim is to navigate through obstacles.



(a) Isometric View

(b) Top View

Figure 14: 3D Octomap of a Square Room



(a) Isometric View (b) Top View

Figure 15: 3D Octomap of a Gallery with left side supported by pillars and right side supported by Wall

The algorithm was tested in PK Kelkar Library, as the book shelves provided a decent amount of features for localization, also the close proximity of the shelves helped us understand better the usefulness of the algorithm in very narrow places like caves and buildings.

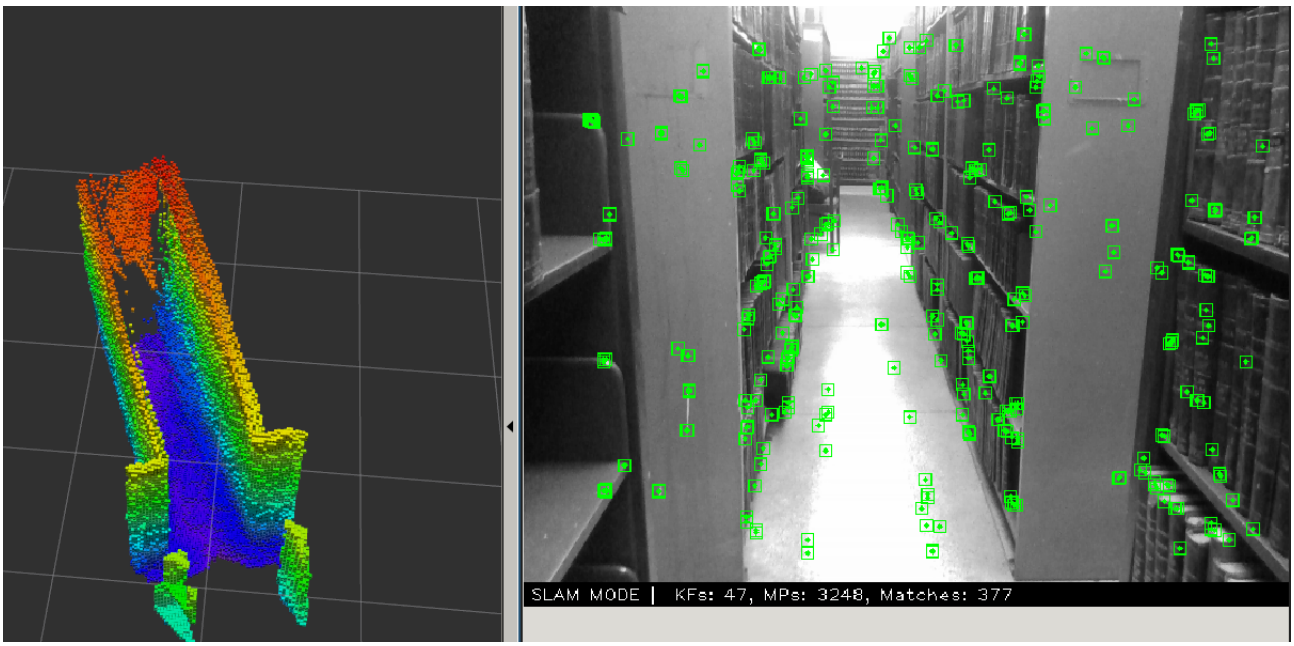


Figure 16: Left: Octomap of Book Shelves, Right: Among the Book Shelves of PK Kelkar Library

To validate the distance measured by octomap is accurate: A table was used as an object and

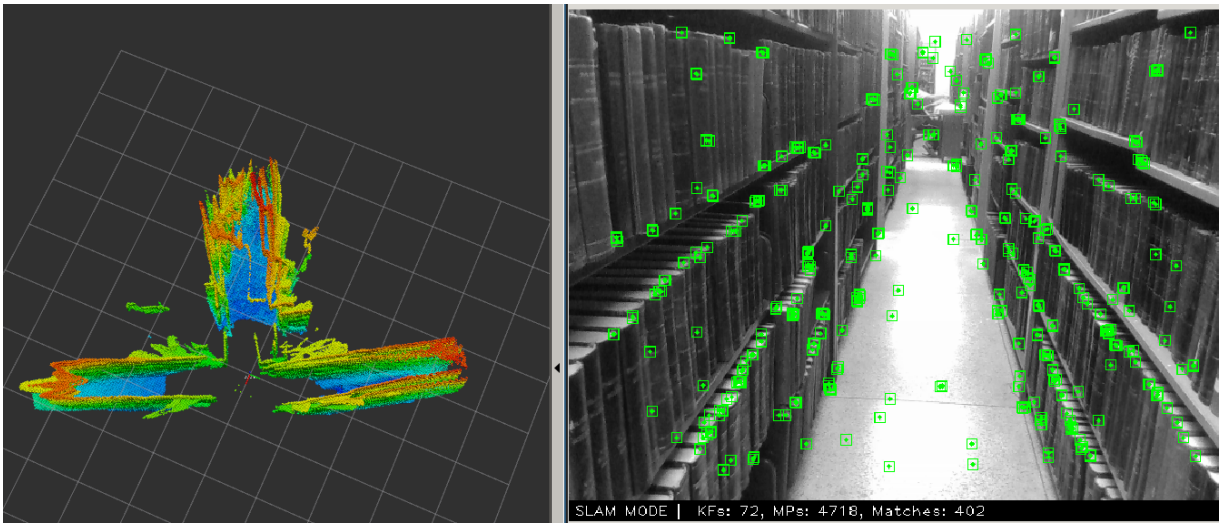


Figure 17: Standing at a junction of three corridors of shelves the camera was rotated and an octomap was build

length of its one side was compared in actual and the length determined by octomap. Each voxel in the octomap was set to define a length of 2 centimeters. A total of 52 voxels laid end to end from their diagonal were measured, measuring to a total of 147centimeters. The actual length of table was 150 centimeters.

(a) Table used for measuring the side length

(b) Octomap of the Table

Figure 18: The front side of the table in Real and Octomap were measured and compared

8 Future Work

- To fill the voxels in the octomap with colors. Currently each voxel is colored based on its z(height) value.
- To implement loop closure, this will enable the position estimation to correct to correct itself once it sees the same place it has visited again.
- To implement the Convolutional Neural Network based SLAM with monocular camera, this will enable us to implement SLAM algorithm with a low cost USB camera.

References

[1] Armin Hornung et al. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* (2013). Software available at `http://octomap.github.com`. DOI: `10.1007/s10514-012-9321-0`. URL: `http://octomap.github.com`.

[2] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. "Estimation of IMU and MARG orientation using a gradient descent algorithm". In: *2011 IEEE International Conference on Rehabilitation Robotics*. June 2011, pp. 1–7. DOI: `10.1109/ICORR.2011.5975346`.

[3] R. Mur-Artal and J. D. Tardós. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (Oct. 2017), pp. 1255–1262. ISSN: 1552-3098. DOI: `10.1109/TRO.2017.2705103`.

[4] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, May 2009.

[5] E. Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2564–2571. DOI: `10.1109/ICCV.2011.6126544`.