SLAM Algorithms

Syamprasad Karyattuparambil Rajagopalan

Abstract

This report describes a study of the Simultaneous Localization and Mapping (SLAM) problem, which is key to solving the autonomous navigation problem, useful for many robots. The first part of the study deals with the implementation of existing SLAM packages on an unmanned aerial vehicle (UAV). These packages are implemented using Robot Operating System (ROS) and uses visual techniques to execute the SLAM. The second part explains the steps, software architecture and development of an Extended Kalman Filter (EKF) based SLAM algorithm. Since the algorithm has not been implemented on a robot, dummy readings from odometry and sensors are used to run the algorithm through one complete operation cycle.

I. Introduction

Robots are increasingly becoming a part of modern human life, and replacing humans in undertaking many tedious and dangerous tasks. This requires human like capabilities such as manipulation, vision, audio perception, walking and navigation. Among these abilities, navigation is a key requirement of many robots including delivery robots, UAVs and self-driving cars. Humans make use of their senses to understand the environment and map it so as to better navigate it. They are also aware of their own position and movements in this environment which enables them to move to desired locations. The robots try to use a similar approach called Simultaneous Localization and Mapping (SLAM).

SLAM mainly consists of two main parts: mapping and localization. Mapping refers to the process of modeling and understanding the environment while localization is the process of estimating robot's own pose in the environment [16]. The robot uses odometry information to estimate the extend to which it has moved in an environment during a particular period. It uses a set of sensors to map the environment or world in which it is moving. The world consists of many landmarks or markers that can be recognized by the robot. When the sensors detect these markers, the information about their position is used to correct the estimate of robot pose obtained from odometry.

This report initially explains the implementation of existing ROS based SLAM packages which will provide an understanding of how these techniques work and what are the results expected from them. The next part explains the software architecture and steps in development of a SLAM algorithm using C++ programming language.

II. Literature Review

The beginning of probabilistic SLAM techniques can be traced back to the 1986 IEEE Robotics and Automation Conference held in San Francisco, California[7]. This conference identified that consistent probabilistic mapping is a fundamental problem in robotics, and that its computational and conceptual problems/challenges must be addressed. The paper [15] describes the creation of a stochastic map and the estimation of uncertain relative spatial relationship among the objects in the map. The paper [9] is an important work which describes the idea of simultaneous map building and localization as a chicken and egg problem. It was identified that the EKF based solution would require the use of a joint state consisting of robot pose and landmark positions. The solution uses a set of sonar sensors to learn environment features from the initial position of the robot and later, uses this information to obtain a more precise estimation of the robot location.

One of the major issues with the conventional EKF based SLAM approach is its computational complexity. This limits the number of landmarks that can be handled by this approach to a few hundred. A method called FastSLAM[11] was developed to deal with this problem. This method decomposes the problem into a robot localization problem, and a collection of landmark estimation problems that are conditioned on the robot pose estimate.

Some of the popular 2D SLAM packages available under ROS has been listed in [14]. It points out that the HectorSLAM focuses on real time estimation of robot movement, exploiting high update rates and low measurement noise in modern LIDARs. It also lists Gmapping as the most widely used SLAM package for robots across the world. This package makes use of particle filter based algorithms and hence, is computationally complex.

Two other popular SLAM techniques which were recently developed are LSD-SLAM[8] and ORB-SLAM[12]. The LSD-SLAM is a large scale direct monocular SLAM method which is designed to run on the CPU. This method detects scale-drift and also implements a novel, probabilistic approach to incorporate noise on the estimated depth maps into tracking. On the other hand, ORB-SLAM is a feature based monocular SLAM method which is designed for real-time operation in large environments, with real-time loop closing as well as camera relocalization abilities. This method uses same set of features for all tasks: tracking, mapping, relocalization and loop closing.

III. Implementation of an Existing SLAM Package

The primary goal of this study is to implement the tum_ardrone[6][17] package on a Parrot AR. Drone 2.0 Power Edition[4]. The basic architecture used by this drone to implement autonomous navigation has been shown in the figure 1.

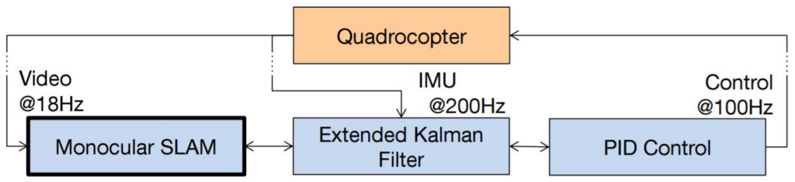


Fig. 1: Basic Architecture of Quadcopter Navigation

The quadcopter uses a wireless connection to send images, at the rate of 18Hz, to a Monocular SLAM package running on a separate computer. The Monocular SLAM relies on PTAM (Parallel Tracking and Mapping) library[3]. This library can use a single camera to obtain 3D features and estimate camera poses. The data from this package is fused with the IMU (Inertial Measurement Unit) data coming in from the quadcopter, using Extended Kalman Filter. This provides a better estimation of the state of the world and the quadcopter, which can be used to issue control commands to the quadcopter.

The tum_ardrone package implements the above mentioned architecture and consists of three main nodes. The node *drone_stateestimation* is used for state estimation and contains PTAM implementation. The node *drone_autopilot* acts as a controller while the node *drone_gui* provides a Graphical User Interface (GUI) for the control purposes. The package also has two main launch files. These launch files make use of the nodes mentioned earlier. The file *ardrone_driver.launch* initializes a driver which acts as an interface to the actual hardware. The second launch file is the *tum_ardrone.launch* which should be launched after the driver is successfully connected to the drone. This will provide three windows: *tum_ardrone GUI*, *PTAM Drone Map View* and *PTAM Drone Camera Feed*. The GUI provides a keyboard or joystick interface for the user to fly and control the drone in a desired manner. The Map View window provides the map built by the package while the Camera Feed shows what the front camera of the drone sees along with the features detected.

The 3D feature map, localization and other results obtained from the implementation of the package has been discussed in section V.

IV. EKF SLAM Development

This section explains the development of a basic SLAM architecture making use of EKF. First, the mathematical steps involved in the process has been explained using an example. This is followed by the software implementation of these steps.

A. *SLAM Example*

Consider the 2D ((x, y)) toy world shown in the figure 2. The robot is shown as the blue circle along with its coordinate system. The world coordinate axes are shown in black. The red crosses indicate the markers or landmarks in the map while the green line indicates the path followed by the robot. It is assumed that the number of landmarks and their data association is known.

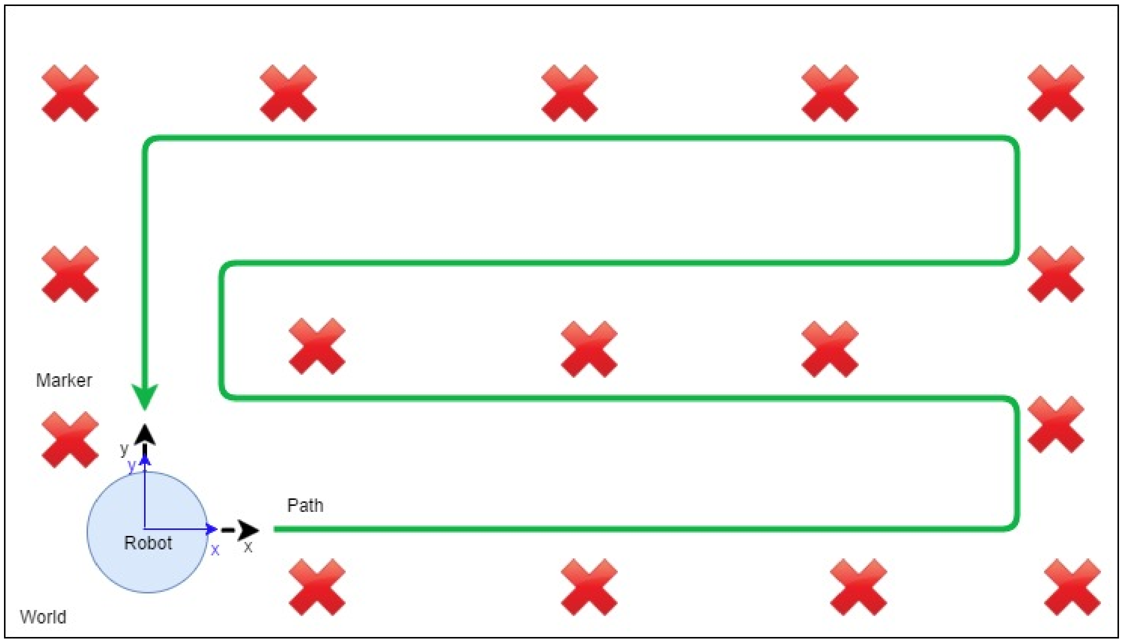


Fig. 2: Toy world for the SLAM problem with markers, robot and its path

Consider the Extended Kalman Filter(EKF) algorithm (inputs = μ_{t-1}, Σ_{t-1}, z_t) [16] [18] [17]:

1) $\bar{\mu}_t = g(\mu_{t-1}, u_t)$
2) $\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + Q$
3) $K_t = \bar{\Sigma}_t H_t^T(H_t\bar{\Sigma}_t H_t^T + Q_t)^{-1}$
4) $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
5) $\Sigma_t = (I - K_tH_t)\bar{\Sigma}_t$
6) return μ_t, Σ_t

The term μ_t is an estimate of the world state X_t and the term Σ_t denotes the corresponding covariance matrix. This matrix denotes how certain the algorithm is, about the estimates. The term z_t denotes the sensor observations.

The world state consists of the robot pose and the marker positions. The robot pose consists of 2D position ((x, y)) and orientation (θ). The marker positions are denoted by (mnx, mny) where n is the number of the marker. Initially, when the robot is turned on, the world coordinate axes are assumed to be aligned with the robot's local coordinate axes. The robot pose is assumed to be (0, 0, 0). Also, the robot does not know the marker positions and those positions are also assumed to be zero. However, since it is evident that all markers are not at the origin, we denote this uncertainty in position by entering an infinite term in the appropriate places of covariance matrix. The covariance matrix will have a dimension of (2N + 3, 2N + 3) where N is the number of total landmarks/markers. This matrix is of the form:

$$\begin{bmatrix} \Sigma_{xx(3,3)} & \Sigma_{xm(3,2N)} \\ \Sigma_{mx(2N,3)} & \Sigma_{mm(2N,2N)} \end{bmatrix}$$

Thus, at time t = 0, the world state and covariance matrix looks like:

$$\mu_0 = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \end{bmatrix}^T \tag{1}$$

$$\Sigma_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \infty & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \infty \end{bmatrix} \tag{2}$$

As the robot moves, the motion model can be used to predict its new pose. This model can be either velocity based or odometry based. This corresponds to step 1 in the EKF algorithm. The velocity based model gives the estimation of the new position as shown below:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} (cos(\theta)V_{tx} - sin(\theta)V_{ty})dt \\ (sin(\theta)V_{tx} + cos(\theta)V_{ty})dt \\ \omega_t dt \end{bmatrix} \tag{3}$$

The values V_{tx}, V_{ty} are velocities in x and y direction while ω_t is angular velocity and dt is the time interval. Note that these values are in local coordinate frame. The multiplication with the rotation matrix takes them to the global coordinate frame. However, the prediction step requires a vector of dimension 2N + 3, not 3. Hence a new matrix, F_x is introduced as shown below:

$$\begin{bmatrix} x' \\ y' \\ \theta' \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \end{bmatrix}^T \begin{bmatrix} (cos(\theta)V_{tx} - sin(\theta)V_{ty})dt \\ (sin(\theta)V_{tx} + cos(\theta)V_{ty})dt \\ \omega_t dt \end{bmatrix} \tag{4}$$

The next step is to estimate the covariance matrix and this requires Jacobian of the motion model which is as shown below:

$$G_{tX} = \begin{bmatrix} 1 & 0 & (-sin(\theta)V_{tx} - cos(\theta)V_{ty})dt \\ 0 & 1 & (cos(\theta)V_{tx} - sin(\theta)V_{ty})dt \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$G_t = \begin{bmatrix} G_{tX} & 0 \\ 0 & I \end{bmatrix} \tag{6}$$

Then the covariance matrix is given by:

$$\bar{\Sigma}_t = \begin{bmatrix} G_{tX} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \Sigma_{xx(3,3)} & \Sigma_{xm(3,2N)} \\ \Sigma_{mx(2N,3)} & \Sigma_{mm(2N,2N)} \end{bmatrix} \begin{bmatrix} G_{tX}^T & 0 \\ 0 & I \end{bmatrix} + F_X^T R_{tX} F_X \tag{7}$$

The R_{tX} is the process or prediction noise. The matrix F_X and its transpose is used to bring this matrix into 2N+3 dimension space. This completes the prediction step.

The next part of the EKF deals with the correction step and requires an observation model. The correction step uses the sensor to observe the landmarks and uses their location to correct the estimate of world (and robot) state. It is assumed that the robot knows the number of landmarks. An important problem in this context is data association. Suppose the robot sees a particular landmark and identifies it as a landmark, without data association, it is difficult to verify which landmark it is and whether it has been observed before. To solve this problem, a simple data association technique is considered. The landmarks in the toy world will be simple cross figures in red color that can be identified with an RGB-D camera. The position of a landmark will be initiated when the sensor first sees the landmark and makes a calculation of its position. As the SLAM proceeds, the uncertainty associated with this position reduces and it approaches the truth. To keep the data association simple, the landmarks are placed far enough such that any new observation after observing known number of landmarks will be discarded if it is not within a specified range of a previously known landmark. Otherwise, it is considered to be same as the landmark it is close to, which leads to correction in the position of that landmark. Once the landmark is identified, the depth sensor in the RGB-D camera can be used to measure the position.

The sensor provides an observation as a tuple of range and orientation:

$$\hat{z_{ti}} = \begin{bmatrix} r_{ti} & \phi_{ti} \end{bmatrix}^T \tag{8}$$

If the landmark j has not been initiated, it shall be done as shown below:

$$\begin{bmatrix} \bar{\mu_{jx}} \\ \bar{\mu_{jy}} \end{bmatrix} = \begin{bmatrix} \bar{\mu_{tx}} \\ \bar{\mu_{ty}} \end{bmatrix} + \begin{bmatrix} r_{ti}cos(\phi_{ti} + \bar{\mu_{t\theta}}) \\ r_{ti}sin(\phi_{ti} + \bar{\mu_{t\theta}}) \end{bmatrix} \tag{9}$$

The first matrix on the left gives the position coordinates of the landmark j while the first matrix on the right gives the position estimate of the robot. In future this landmark position coordinates will be used to calculate the expected observation:

$$\delta = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = \begin{bmatrix} \bar{\mu_{jx}} - \bar{\mu_{tx}} \\ \bar{\mu_{jy}} - \bar{\mu_{ty}} \end{bmatrix} \tag{10}$$

$$q = \delta^T \delta \tag{11}$$

$$h(\bar{\mu_t}) = \begin{bmatrix} \sqrt{q} \\ atan2(\delta_y, \delta_x) - \bar{\mu_{t\theta}} \end{bmatrix} \tag{12}$$

The function h is called the observation model. The further steps require the calculation of the Jacobian of this observation model:

$$H_{ti}^{(low)} = \frac{1}{q} \begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix} \tag{13}$$

Once again, a matrix F_{Xj} is used to bring this low dimensional Jacobian to 2N + 3 dimensional space.

$$H_t = H_{ti}^{(low)} \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & \dots & 0 \end{bmatrix} \tag{14}$$

Once H_t is available, the Kalman gain is calculated. This is the factor that decides how much the observation model and motion model should be trusted. The noise in the sensor model, represented by a (2, 2) matrix Q, also plays a role in this calculation.

$$K_t = \bar{\Sigma_t} H_t^T (H_t \bar{\Sigma_t} H_t^T + Q_t)^{-1} \tag{15}$$

In the next step the estimation of world state is corrected by comparing what the sensor actually sees and what the sensor model predicts.

$$\mu_t = \bar{\mu_t} + K_t(z_t - h(\bar{\mu_t})) \tag{16}$$

The next step is the correction of the covariance matrix:

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma_t} \tag{17}$$

Thus the world state and covariance matrix for the current time step has been obtained. This world state consists of the robot pose and also the position of the landmarks. The robot pose in all time steps can be calculated and plotted to obtain the path traversed by the robot, thus completing the localization. The mapping can be completed by plotting the landmarks at the obtained landmark positions.

B. *EKF SLAM Software Architecture*

The EKF SLAM is being implemented using C++. The figure 3 shows the current software architecture. There are three main classes: World State, Robot and Sensor. There is an additional class called Marker which is not part of Sensor class, but is only used by it. There are separate files for global variables, functions and main function.

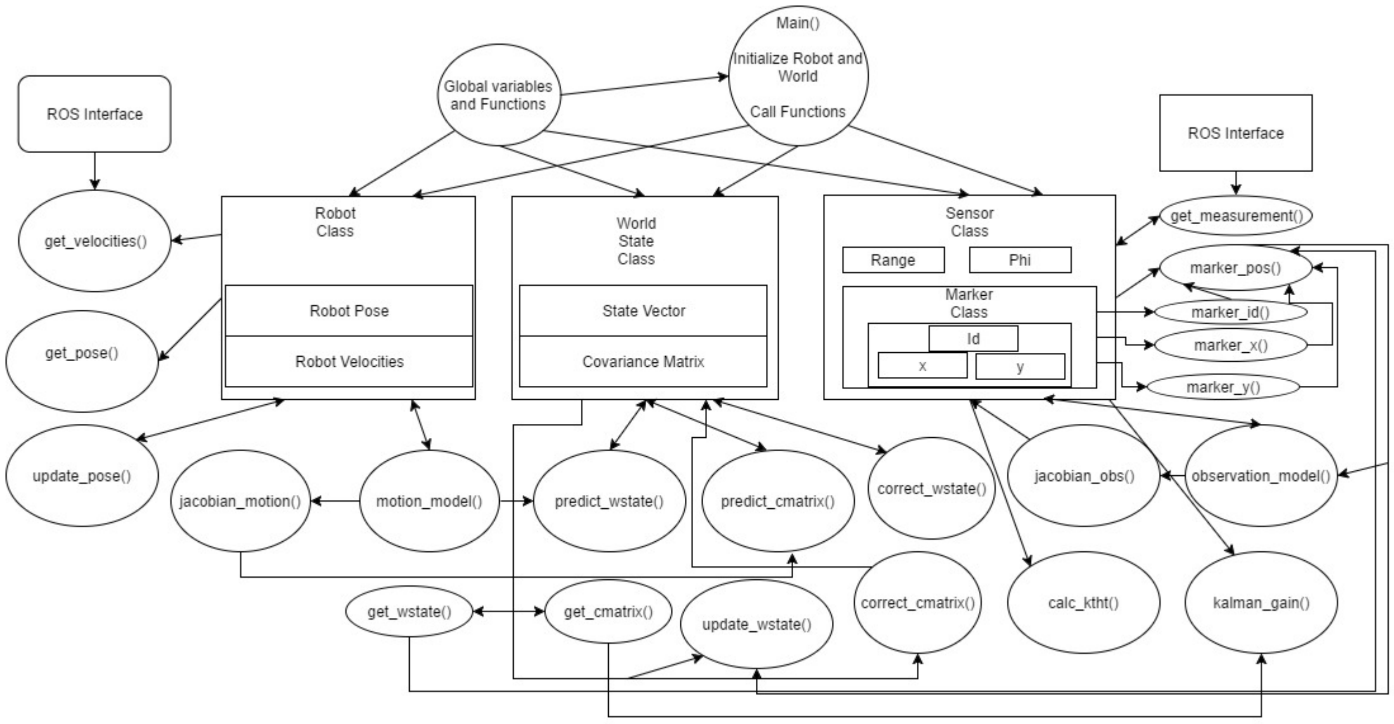


Fig. 3: Software Architecture

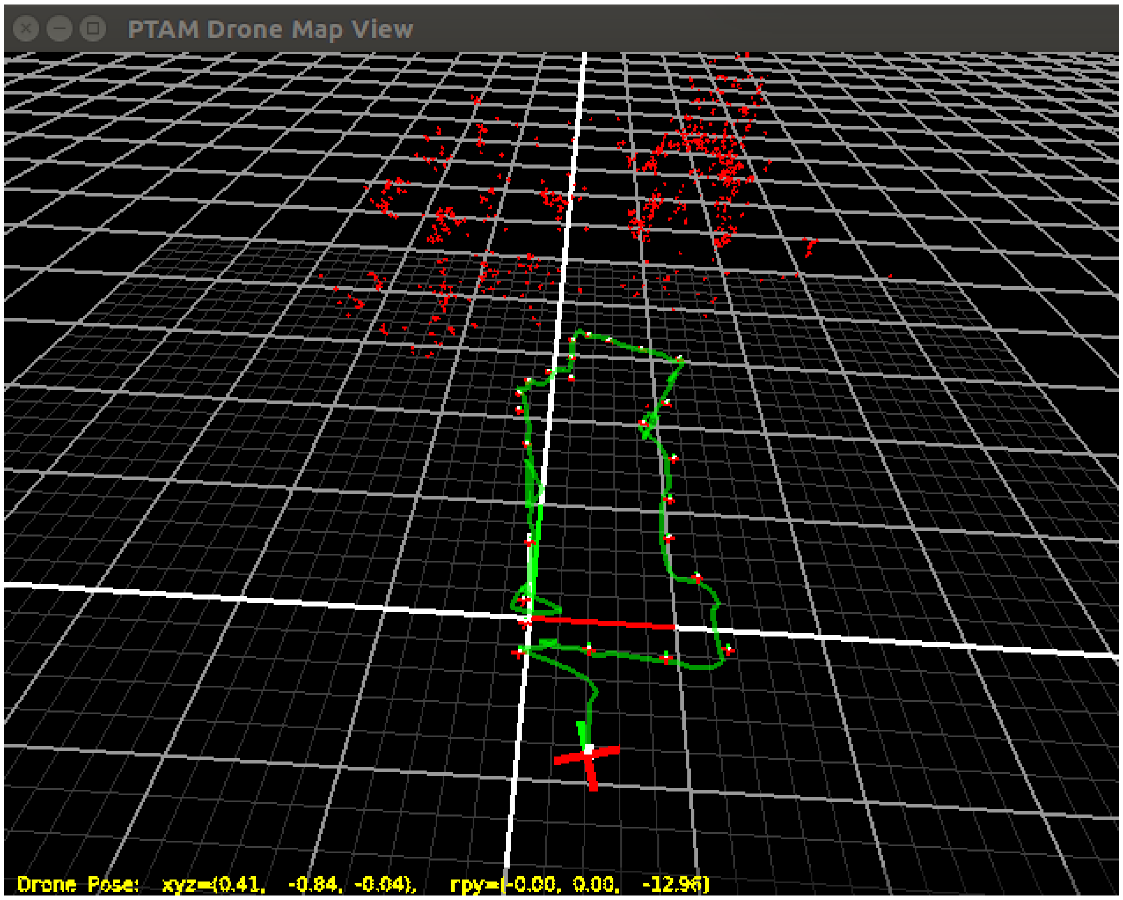
The main function initializes the robot object (*segway_rmp*), of Robot class, with initial pose and velocity with values zero. The world state and the covariance matrix are also initialized similarly by creating a world state object (*segway_world*), of World State class. The functions *get_pose* and *get_velocities* return matrices with robot pose and robot velocities. The robot velocities are obtained from its ROS interface. Once the velocities are obtained, the *motion_model* function can be called to estimate the new robot pose. This pose is used to update the prediction of world state, using *predict_wstate*. The *jacobian_motion* calculates the Jacobian of motion model and updates the prediction of covariance matrix via *predict_cmatrix*. This completes the prediction step of EKF.

The next stage is the correction step and involves the sensor observations. The *get_measurements*, a function of Sensor class, will obtain sensor measurements using ROS interface. The markers observed, if uninitialized, will be initialized using *marker_pos*. The same function is responsible for updating world state with the new marker positions and calling the *observation_model* function. The *observation_model* calculates the expected observation and calls the *jacobian_obs* function. The *jacobian_obs* calculates the Jacobian for the observation model. Once this is done, the control returns to *marker_pos* which returns predicted or expected observation. The *kalman_gain* uses this Jacobian and the covariance matrix to calculate Kalman gain. The *correct_wstate* function uses predicted observation, actual observation (measurements from the sensor) and kalman gain to correct the world state. The function *calc_ktht* calculates the product of kalman gain and the jacobian of observation, required for the correction of covariance matrix. The *correct_cmatrix* function is used to correct the covariance matrix. The *update_pose* function will update the robot pose with the corrected values from the world state vector.

This completes the correction step and one loop of the program execution. Further discussion on the topic is available in section V.

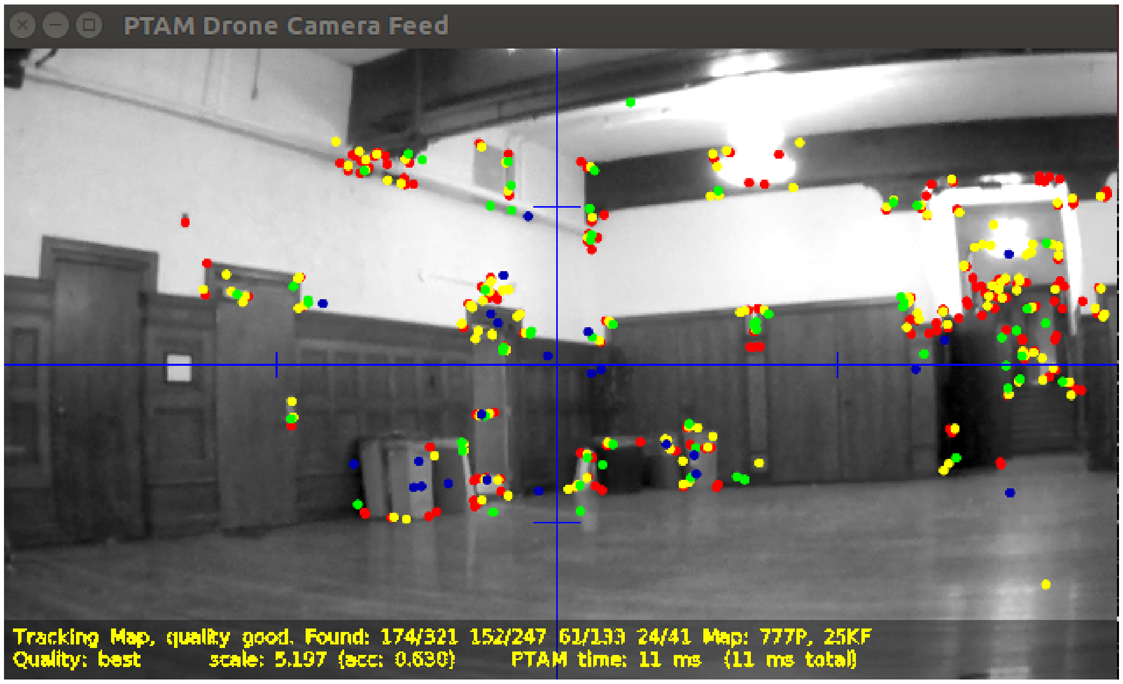
V. Results

This section first discusses the result of the work described in section III. The figure 4 shows the 3D feature map obtained from implementing the tum_ardrone SLAM packages on a Parrot AR Drone 2.0. The visual features such as corners are detected using FAST[2] algorithm and tracked using KLT Tracker [10][19]. The estimation is improved by linking the output from PTAM with IMU data via EKF. The figure 5 shows the camera of the drone detecting the 3D features.



Drone Pose: xyz=(0.41, -0.84, -0.04), rpy=(-0.00, 0.00, -12.96)

Fig. 4: The green line indicates the path of the drone while the red points indicate the 3D features



```
Tracking Map, quality good. Found: 174/321 152/247 61/133 24/41 Map: 777P, 25KF
Quality: best        scale: 5.197 (acc: 0.630)        PTAM time: 11 ms   (11 ms total)
```

Fig. 5: The camera feed from the drone

The result of the work described in section IV is a C++ code available at [13]. The code has been tested using Codeblocks IDE[1]. The functions *get_velocities* and *get_measurement* have dummy values of velocities, and range and phi respectively; to simulate the execution of one cycle of EKF SLAM. The time interval used in many calculations has also been set to a random value for the same purpose.

VI. Future Work

The next step would be to provide ROS interface with the robot hardware, such as Segway RMP 200[5], to eliminate the use of dummy values for velocities and sensor observations. A 3D Depth sensing camera could be a very suitable sensor for this approach. The motion model, which is currently velocity based, may be changed to odometry based, depending on the robot platform. Another important aspect is the detection and data association of the landmarks in the sensor observation. A simple approach could be to use specific landmarks throughout the environment, that can be easily detected by simple computer vision algorithms. An assumption can be made that the landmarks are far enough such that any detection or observation within a particular range of a particular landmark can be associated with that landmark only; thus circumventing the problem of data association.

It is also important to develop tools to visualize the results of SLAM, such as those used in tum_ardrone package. There are other advanced research paths possible where the existing SLAM packages can be compared with what is being developed in this work. Another possible area of research would be how to reduce the computational complexity of the SLAM solutions. Overall, the SLAM is a very challenging, critical problem and this study is only an attempt to understand the basics of this problem.

References

[1] Code::Blocks IDE. `http://www.codeblocks.org/`.
[2] FAST Algorithm for Corner Detection. `http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html`.
[3] Parallel Tracking and Mapping. `http://www.robots.ox.ac.uk/~gk/PTAM/`.

[4] Parrot AR. Drone 2.0 Power Edition. https://www.parrot.com/us/drones/parrot-ardrone-20-power-%C3%A9dition#ar-drone-20-power-edition.

[5] Segway RMP 200. http://rmp.segway.com/tag/segway-rmp-200/.

[6] TUM ARDRONE package. http://wiki.ros.org/tum_ardrone.

[7] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[8] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[9] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee, 1991.

[10] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[11] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598, 2002.

[12] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[13] Syamprasad K Rajagopalan. Github Repository for Directed Research on SLAM Algorithms. https://github.com/syamprasadkr/dr_slam/tree/skr.

[14] Joao Machado Santos, David Portugal, and Rui P Rocha. An evaluation of 2d slam techniques available in robot operating system. In *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pages 1–6. IEEE, 2013.

[15] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. *arXiv preprint arXiv:1304.3111*, 2013.

[16] Cyrill Stachniss. SLAM Course. https://youtu.be/U6vr3iNrwRA?list=PLgnQpQtFTOGQrZ4O5QzbIHgl3b1JHimN_.

[17] Jürgen Sturm. Autonomous Navigation for Flying Robots. https://courses.edx.org/courses/course-v1:TUMx+AUTONAVx+2T2015/info.

[18] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[19] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.