SLAM TECHNIQUES AND IMPLEMENTATION FOR AERIAL ROBOTICS

JERRIN BRIGHT
Vellore Institute of Technology, Chennai, India

Abstract: Simultaneous Localization and Mapping or SLAM is a computational process of reconstructing or creating a map with the help of a robot or an unmanned system of an unknown environment for navigation using the map it generates while simultaneously localizing the position of the robot/ unmanned system. All the types of localization, mapping techniques and SLAM implementations are explained in detail in this paper. More importance is drawn towards explaining sensor fusion using EKF to localize the aerial robot and mapping using Visual-Inertial SLAM approaches to map the unknown environment for autonomous navigation.

Keywords: SLAM, Localization, Mapping, Optimization.

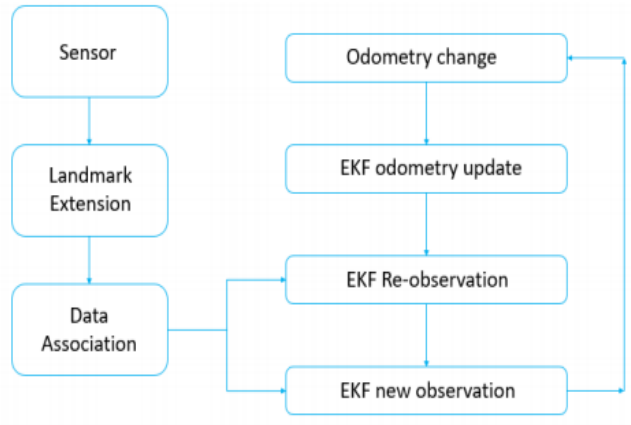
WHAT IS SLAM?

Building maps and localizing a vehicle/ robot in a world at the same time simultaneously is termed as SLAM. It uses Mapping, Localization and Pose Estimation algorithms to build map out of unknown environments. GPS doesn't work well indoors. Even outdoors, it is accurate up to a few meters. This kind of accuracy is too low for autonomous applications. Thus, to address these issues, SLAM was developed. The major advantages of SLAM are:

- SLAM uses many types of sensors like laser and camera (visual also termed as V-SLAM).
- Indoor mapping and location identification are easier with SLAM.
- SLAM can produce 3D images of the surroundings

ARCHITECTURE OF SLAM

SLAM is twofold as the name suggests, it needs to construct or update map of an environment while simultaneously keeping track of the object location.



The basic steps involved in SLAM are:

- Landmark Extraction - The algorithm will terminate only when the population of the particles converging are features which can easily be re-observed and distinguished from the environment. These are used by the robot to find out where it is (to localize itself).

- Data association - The problem of data association is that of matching observed landmarks from different (laser) scans with each other.
- State estimation - The Extended Kalman Filter is used to estimate the state (position) of the robot from odometry data and landmark observations
- State update.
- Landmark update.

Before a robot can answer the question of what the environment looks like given a set of observations, it needs to know from which locations these observations have been made. At the same time, it is hard to estimate the current position of a vehicle without a map. A good map is needed for localization while an accurate pose estimate is needed to build a map.

APPLICATIONS OF SLAM

There are a widespread use of SLAM and some of the common usage to start with are the following:

- Autonomous vehicles/ UAVs
- Autonomous Underwater vehicles
- Planetary rovers
- Domestic robots like Roomba
- Self-driving cars

METHODS OF SLAM

VISUAL SLAM

A technology which uses single camera based on computer vision theory. That's a technology using camera images are called Visual SLAM. In the initial Visual SLAM, the position of the camera was tracked through matching feature point in the image and then 3D map was generated. However, this method has the drawback that the processing speed is slow because matching feature point and updating map has to be performed for all the image frames. As a result, Visual SLAM now uses keyframes to improve performance by parallelizing the tracking and mapping process.

LIDAR SLAM

A LiDAR-based SLAM system uses a laser sensor paired with an IMU to map a room similarly to visual SLAM, but with higher accuracy in one dimension. LiDAR measures the distance to an object (for example, a wall or chair leg) by illuminating the object with multiple transceivers. Each transceiver quickly emits pulsed light, and measures the reflected pulses to determine position and distance. Because of how quickly light travels, very precise laser performance is needed to accurately track the exact distance from the robot to each target. This requirement for precision makes LiDAR both a fast and accurate approach. However, that's only true for what it can see.

LOCALIZATION

EXTENDED KALMAN FILTER

EKF is used to estimate the 3D pose of a robot, based on (partial) pose measurements coming from different sources. It combines measurement from different wheel odometry, IMU sensors and visual odometry. Most real-world problems involve nonlinear functions. In most cases, the system is looking

into some direction and taking measurement in another direction. This involves angles and sine, cosine functions which are nonlinear functions which then lead to problems. This issue was the reason to go for EKF from KF. Linear equations give gaussian results when gaussian is applied but nonlinear equations will not give gaussian. And we also know that, in real-world all problems are nonlinear. Therefore, the solution for this problem will be to approximate the nonlinear equations to linear equations. So, after doing this approximation, what we get is the Extended Kalman Filter. This approximation can be done using various tools in statistics like the Taylor series. That's, Gaussian on the nonlinear function will be done followed by taking the mean first and then performing a number of derivatives to approximate it. The first derivative of a Taylor series is called as Jacobian Matrix. This Jacobian Matrix converts the nonlinear curve to linear function. The mathematical steps involved in this EKM are:

- Prediction Step

Estimates of the current position along with noises (the uncertainties).

- Update Step

Nonlinear Measurements coming from the sensor.

$y = z - h(x')$

This is the difference between measured and actual value.

- Mapping Function

Mapping is specified between the Cartesian and Polar coordinates.

$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y / p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

- Kalman Gain

This is basically the weight given to the measurements and the current state estimate.

$S = HjP'HjT + R$

$K = P'HjTS{-}1$

- Jacobian Matrix

$$H_j = \begin{bmatrix} \frac{\partial \rho}{\partial p_x} & \frac{\partial \rho}{\partial p_y} & \frac{\partial \rho}{\partial v_x} & \frac{\partial \rho}{\partial v_y} \\ \frac{\partial \varphi}{\partial p_x} & \frac{\partial \varphi}{\partial p_y} & \frac{\partial \varphi}{\partial v_x} & \frac{\partial \varphi}{\partial v_y} \\ \frac{\partial \dot{\rho}}{\partial p_x} & \frac{\partial \dot{\rho}}{\partial p_y} & \frac{\partial \dot{\rho}}{\partial v_x} & \frac{\partial \dot{\rho}}{\partial v_y} \end{bmatrix}$$

- Polar Cordinates

Thus, using these Jacobian matrices, we will retrieve polar coordinates, that's from cartesian coordinates.

ISSUES WITH EFK:

- If the initial state estimated is wrong, the filter will diverge quickly.
- Impact of imperfect models.

PARTICLE FILTER LOCALIZATION

Also called as Monte Carlo localization (MCL). With a given map, the algorithm attempts to determine the position and the orientation of the robot with the help of particle filters. Particles are basically the location the robot guesses to be present compared to the current position. Thus, this estimated position or the particle
is compared with the readings from the sensor data and convergence/ divergence is analyzed. Ultimately, the sensor readings have to converge with the particle (estimated position). So basically, how this works is that the actual and particle positions will be estimated and then probability will be calculated, then when the car starts to move the irrelevant particles will be differentiated and then removed. Note: Before estimating the particle, landmark coordinates have to be fixed based on which rough calculations using the sensor is laid.
First let's discuss how to generate the particles. The steps involved are:

- Initializing the landmarks and the world size
- Depending on the world size, random position of the actual robot is estimated.
- Checking if this position is then in the bounds of the world size.
- Find the distance of the robot from the landmark.
- Finally moving the robot and thus returning new position.

Note: Noises are also taken into consideration. Noises indicate the uncertainties caused on the robot on its environment.

Secondly, we will discuss how to Measure the Probability. As discussed earlier, probability is determined with the actual position and the particle position obtained. Thus, particles close to the actual robot will have high probability and far-off particles will have less probability value. Once probability is estimated, we will go to the third step that's the Resampling. Resampling is simply removing of the less probability positions or the particles from the hypothesis/ algorithm being processed. The main reason for removing unwanted/ fewer probable positions is:

- Reduces the depletion of the particles.
- Processing with less probable cases/ scenarios is economical
- High probable positions will not be segregated, which increases tracking of useless hypothesis.

Once resampling is done, there will be only one more step to be done, Determining the Orientation. That is, even though we found out the closest location, we will have to determine the orientation of the robot. Let's take a scenario of after 100 iterations, there comes a situation with only 2 particles A and B, A in the direction of the robot itself. So now, when time increases, the probability measured with respect to particle B will be more as it moves away from the robot itself. So, after some more iterations, Particle B will be filtered out. Thus, we can say that when time increases (thereby increasing iterations), we will be able to determine the exact orientation of our robot. Thus, only one particle will be left out which will have high probability which is the particle closest to the location and orientation.

APPLICATION OF PFL

In FASTSLAM:
The FASTSLAM nomenclature consists of:

- DATA – Input of the robot (go left, right) and the Measurements (distance with respect to the landmarks, etc.)

- STATE – Position of the robot and the landmark

Thus, in FASTSLAM, both the input of the robot and the measurements are calculated at the same time in each and every step. Also, the landmarks are considered stationery. And that the state will have to estimate the value of both the robot and the landmark.

COMPARING EKF WITH PFL

Particle Filter Localization when compared with the Effective Kalman Filter is better based on the following problem; The EKF will represent only one single Gaussian, that's one particular position of the robot. This means that if there are in case two different places where the robot is estimated to be, the EKF will filter one which is most likely to occur. But this goes well only if the chosen position is correct. Otherwise, the complete hypothesis will fail. This is where Particle Filter Localization gives an edge over EKF.

In simple words, PFL will keep in track of all the hypothesis planned and estimates simultaneously and based on the probability estimation done in PFL as discussed previously, it filters out the less likely position (less probability particles) and continues its tracking with new particle observation. This factor as mentioned above makes Particle Filter Localization more effective and robust when compared to Effective Kalman Filter in terms of localization.

HISTOGRAM FILTER LOCALIZATION

Histogram Filter Localization is a grid-based approach that is analogous to midpoint rectangular integration. It is an approximation of Bayes filter. Bayes Filter is an algorithm that estimates the probability distribution of a robot position conditioned on the series of observations that's the camera images and velocity commands. This posterior over states is called as belief. Thus, a Histogram Filter Localization is called as a type or an approximation of Bayes filter as it represents the belief as histogram that's splitting the world with one probability value per state. Also, Histogram Filter Localization is a non-parametric filter. That's it doesn't rely on fixed functional forms. Also, the number of samples biases the speed of the algorithm and quality of the filter in non-parametric filters. The grid will have 2 state variables. One state variable map to the x-axis and other state variable to the y-axis. Once grids are created, the iterations over all the cells of the grid has to be done, do that belief will be updated upon sensor inputs.

if d is a measurement z **then**
  $\eta = 0$
  for all x **do**
    $Bel'(x) = p(z|x)Bel(x)$
    $\eta = \eta + Bel'(x)$
  end for
  for all x **do**
    $Bel'(x) = \eta^{-1}Bel'(x)$
  end for
else if d is a action u **then**
  for all x **do**
    $Bel'(x) = \int p(x|u, x')Bel(x')dx'$
  end for
end if
return $Bel'(x)$

So basically, just like the landmarks in particle filter localization, here in histogram filter localization we will need few references. The robot will move with respect to these references. The robot while moving,

will calculate the position probability of the histogram filter, cell after cell in the grid. This filter will integrate the speed input and range observations from the references for the localization task.

DISADVANTAGES OF HFL

The probability of each particular possible state can't be found. We can only be able to find the probability of the state in a certain region of the world.

COMPARING HFL AND PFL

When comparing Histogram Filter Localization with Particle filter Localization, one major difference observed is that with less time, that's a smaller number of particles and a smaller number of grids which depends on the computation speed of the device, the Particle filter Localization is observed to outperform Histogram Filter Localization. Thus, in situations where computational power is severely restricted, the particle filter can outperform histogram filter.

PATH PLANNING

Path planning is one of the basic operations needed to implement robot navigation. So, once we localize our robot as discussed, using various methods like EFK, PFL or HFL, we will have to do path planning. That's once we know the destination point cloud and localize our position, we will have to plan the route the robot/ drone takes in-order to reach the destination. This planning operation is called as Path planning. It can only be done if the map of the world is known beforehand. The path the robot takes has to be collision-free. There are a lot of path planning methods available. In ROS, we use the move_base package which moves the robot from its current position to the goal position with the help of other navigation nodes. The move_base package basically links the local and global planner for path planning. Thus, it will subscribe the goal topic which will be the input of the navigation stack. Once the goal is subscribed, it will be passed to the global_planner, local_planner, recovery_behavior and costmaps which in return will generate the output that's the cmd_vel sending it to the base controller for moving the robot for achieving the goal pose. Now we will discuss about all the links attached to the move_base. Some of the important links are:

- global_planner –

Path planning is done here using algorithms like A* and Dijkstra also calculates the shortest path possible for traversing.

- local_planner –

Uses odometry and sensor readings and sends appropriate cmd_vel values to the robot controller for accomplishing the plan drafted by the global planner.

- rotate_recovery –

Takes a 360 degree turn and decides the path most suitable to traverse. This is especially used when the robot collides with an obstacle.

- Costmaps –

They create grids on the environment and then navigate accordingly. We know the robot can only plan if it knows the map. Therefore, costmaps create grids on the map and each cell will have a probability which will help the controller to understand if there is an obstacle or it is free.

- Map_server -

It is used to save the map and also load it when needed to navigate.

- AMCL -

Helps in localizing the robot in the map. It basically uses particle filter to localize the robot in an environment with the help of probability theory as discussed earlier in TASK3. AMCL can work only with laser scans.

- Gmapping –

An implementation of the FASTSLAM algorithm. Takes in laser scan data and odometry to build the map of the environment.

Thus, the working of the navigation stack is as follows:

- Localizing the map -Using AMCL / EFK / HFL
- Setting the goal
- Sending the goal and Path planning

Sending the velocity to the robot controller

NODE BASED OPTIMAL ALGORITHMS

Dijkstra's Algorithm

Finds shortest path in a graph where weights of the edges are already known. It is a form of dynamic programming. It finds the shortest path using local path cost. When applying in 3D space, a 3D weighted graph must be built first; then it searches the whole graph to find the minimum cost path. It has 2 sets, one containing the vertices included in the shortest path tree and other set with the ones not in the tree. The steps involved in this algorithm are:

- Creating a set shortest path tree
- Assigning a distance value to all vertices
- Check the adjacent vertices of the lowest vertex and pick the vertex with minimum distance.
- Update the distance value
- Pick a vertex not in the path tree and check its adjacent vertices
- Repeat the steps until the path tree does include all vertices given.

A-Star Algorithm

In maps the A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state). A* algorithm has 3 parameters:

- g: the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
- h: it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.
- f: it is the sum of g and h. So, f = g + h

Thus, this algorithm works in the following way, it will calculate this f value and find the smallest f valued cell and move to that particular cell. This process will be continued until it reaches the destination, that's the goal cell.

D-Star Algorithm

D* resembles A* but is dynamic (its cost can change in the traversing to reach goal). That is, it will assume that there is no obstacle in a particular grid and once it
approaches the grid, it will dynamically adjust and adds information to the map and if necessary, re-plans the entire path from the current coordinates. This process is repeated until the robot reaches the destination. There are 3 types of D* algorithm namely original D*, Focused D* and D* Lite. All these are basically combinations of different path planning algorithms like A*. D* is an incremental search algorithm used for path planning. D* resembles A* but is dynamic (its cost can change in the traversing to reach goal). That is, it will assume that there is no obstacle in a particular grid and once it approaches the grid, it will dynamically adjust and adds information to the map and if necessary, re-plans the entire path from the current coordinates. This process is repeated until the robot reaches the destination. There are 3 variants with the D* algorithm, that's:

- D* - Original Algorithm
- Focused D* - It is a modification of the original D* algorithm. It is a combination of A* and the original D* algorithm.
- D* Lite – An incremental Heuristic search algorithm built on the lifelong planning A* algorithm.

D* algorithm has a lot of benefits:

- Optimal and Complete
- More efficient than A*
- Reduces computational costs
- There won't be much of impact of local changes on the path

Theta-Star Algorithm

Theta * is an any-angle path planning algorithm that is based on the A* search algorithm. They search for a path in the space between two points and takes turn in any angle. The resulting path will be towards the goal with fewer turns. Algorithms like the A* will be limited only within the grids thus will produce indirect and jagged paths. It interleaves smoothening with the search. There are many variants of the theta* algorithm. Some of them are:

- Lazy theta*
- Incremental Phi*

A* VS THETA*

A* algorithms is slower due to many edges and many line-of-sight checks compared to the Theta* algorithm. Slower in terms of constructing the map/ graph. The results of a study concluded that the A* and Basic Theta*algorithm has the same completeness criteria and has time complexity which is relatively same, the A* algorithm has the advantage of optimality in fewer number of nodes searched, whereas the Basic Theta*algorithm has the advantage of the optimality the shortest results.

ROS PACKAGES FOR 3D LOCALIZATION

- Mcl_3dl

A ROS node that performs 3D localization system for robots with 3D LIDARS. It implements point cloud-based Monte Carlo localization aka particle filter localization that uses a reference point cloud as the map. It receives the reference point clouds and localizes the 6 DOF that's the x, y, z, yaw, pitch, roll and yaw poses assisted by the motion prediction from the odometry.

- Hdl_localization

Uses the Unscented Kalman Filter based estimation. It first estimates the sensor pose from IMU data and performs multi-threaded normal distribution transform scan between global point cloud and input point clouds to correct the estimated pose.

- Amcl3d

It is a problematic algorithm to localize a robot in 3D. It uses Monte Carlo localization just like Mcl_3dl. It uses laser sensor and radio-range sensors to localize the UAV with the help of a map. Thus, it with a defined map calculates the weight of the particle and localizes with the probability distributions. It occurs in 3 steps just the way in any particle filter localization that's prediction, update and resampling.

- Rtabmap_ros

Rtabmap stands for Real-time appearance-based mapping. A RGBD SLAM approach based on the global loop closure detection with real-time constraints. It is used to create a 3D point cloud in an environment and also to create a 2D occupancy grid map usually used for navigation. It can be used along with a Kinect, stereo camera or a 3D lidar.

- Dynamic Robot Localization

The dynamic_robot_localization offers 3 DoF and 6 DoF localization using PCL and allows dynamic map update using OctoMap. It's a modular localization pipeline, that can be configured using yaml files.

MOTION PLANNING

GRAPH BASED PLANNING

Grassfire

When a robot moves in a known environment, the floor will be split into grids/ blocks. Coordinates will be assigned to the grids. And then each grid will be assigned a binary value depending on if the grid is empty or not. Now we start from the goal location and mark it as 0 value. Each of the four neighboring cells that is empty is marked a consecutive value 1. Now the neighbors of the cells that are marked as 1, are given the next value 2. This goes on till the start location is reached. Now each value of the cells surrounding the goal location represents the number of steps required by the robot to reach the goal location from this cell. Therefore, successfully selecting the minimum neighboring value will lead the robot to the goal location via the shortest path.

Configuration Space

Let as assume a point x in the C-Space. There is a collision detection function called CollisionCheck which will return a binary value depending on what lies in the grid the robot is visualizing. For example, the CollisionCheck will return 1 if there is a collision with an obstacle and 0 if x is a free space. The collision or free space will be found by considering the obstacle and the robot as triangles. Then we will check when the robot is placed at a particular point cloud within the C-Space, if a triangle still prevails or a polygon is formed. If a polygon is formed, it means the robot is coinciding with the obstacle, thus the CollisionCheck will return 1 or if there is only a triangle, it will return 0 thus reflecting it is a free space.

SAMPLING BASED PLANNING METHODS

Rapidly-Exploring Random Tree

The Rapidly-exploring Random Tree is actually quite straight forward. Points are randomly generated and connected to the closest available node. Each time a vertex is created, a check must be made that the vertex lies outside of an obstacle. Furthermore, chaining the vertex to its closest neighbor must also avoid obstacles. The algorithm ends when a node is generated within the goal region, or a limit is hit.

Probabilistic Road Map

A new technique to deal with choosing points. Here, we will randomly choose points in the C-Space instead of uniform selection, assuming we will get the structure of the free space from these random points. Thus, on every iteration, a new set of points are randomly assigned in the C-Space and the CollisionCheck will return 0 or 1 depending on whether free space or collision. Thus, in every turn when it gets a free space, it will try to forge a new configuration and closest existing sample. The problem with probabilistic road map is that there can be a possibility that the robot might fail to find a path even if it exists. Thus, this kind of approach is not considered a complete path planning algorithm.

BIONIC PATH PLANNING ALGORITHMS

Ant Colony Optimization Algorithm

ACO is an intelligence-optimized algorithm that simulates the heuristic mechanism of the shortest route based on pheromone in the process of ants foraging for food. ACO uses all paths of the entire ant colony to describe the solution space of an optimization problem, and obtains the best path through positive feedback based on pheromone. The idea of ACO directly maps the robot's path optimization problem, which is easy to understand and has very intuitive results.

Particle Swarm Optimization

It optimized by continuously iterating to improve the candidate solution with regards to given measure of quality. It will have some particles, which will be moving in the environment/ search space with a particular position and velocity. Each particle will be influenced by its local best-known position which are updated as better positions by other particles. This will move the swarm towards the better solution ultimately. It is metaheuristic as it makes few/ no assumptions about the problem optimized.

Genetic Algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. The algorithm will terminate only when the population of the particles converge.

SENSOR FUSION

Combining two or more sensor data in a way that generates a better understanding of the system is called as sensor fusion. Thus, sensor fusion does the sensing and perceiving part for the autonomous systems.

That is, it will take multiple sensor readings, combining it to get a better model, so that the system can plan and act efficiently.
The major advantages of sensor fusion are:

- Increase in the quality of data by removes constant noises
- It increases reliability (even if one fails, we will get value but not as accurate as before obviously).
- Unmeasurable states can be measured (one camera can't measure the distance between itself and an object, but 2 cameras together can).
- Increasing range (like having many ultrasonic around a car for close distance detection)

Kalman filters are one of the most efficient sensor fusion algorithms used. The advantage of Kalman filters is that the mathematical model of the system is already built in the filter, so fusing of sensors can be done and map quality can be enhanced. Kalman filters will operate in 2 steps:

- Predict
- Measurement
- Update

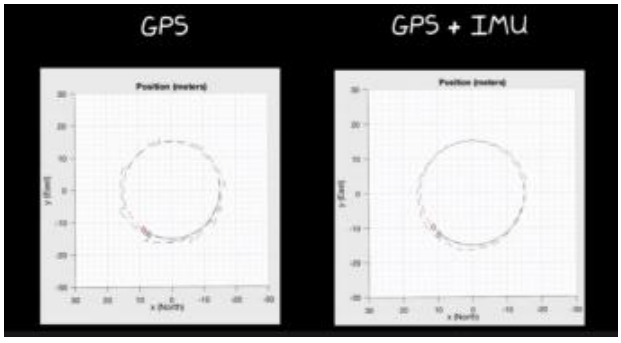
Predict:

$$p(x_k|y_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|y_{1:k-1})dx_{k-1}$$

Update:

$$p(x_k|y_{1:k}) = \frac{p(y_k|x_k)p(x_k|y_{1:k-1})}{p(y_k|y_{1:k-1})}$$

Prediction is based on the previous knowledge of the robot and kinematic equations. Measurement is getting the readings form the sensors. Update is updating the value based on the predicted and measured readings.
This predict equation uses the position of the previous time-step k-1 together with the motion model to predict the current state xk. This will be updated in the update equation shown above using the Bayes theorem thus combining the measured and predicted states. Now we will have the posterior distribution. Similarly, we can now consider this posterior as the previous posterior and find the next posterior. Thus, this method is iterated for further measurements. It will first measure using a sensor and then uses a mathematical model to compare it with the measured values from the sensors. Thus, it estimates the state from the confidence attained from the predicted and measured values. This way, the sensor will run asynchronized measurements. IMU is a fusion of accelerometer, magnetometer and gyroscope. IMU is often fused with GPS in drones.



In the image seen above, when we use only GPS, the prediction is drastically running away from the ground truth due to the slow update rate of the GPS. Whereas when IMU is added the prediction will be corrected with the fast update rate of the IMU sensor against that of the GPS. Thus, once the sensor readings converge, we will get a better prediction and thus a better state estimation.

LIDAR CAMERA FUSION

LIDAR has an advantage of sharp accuracy in ranging and thus results in good mapping. Coming to Visual sensors, they provide a very dense map and comparatively cheap and consumes less power. But it also has its cons, including poor depth estimation, small range for stereo-visions, difficult to use RGBD cameras outside due to the lightings. Thus, considering both its pros and cons, a fusion of these two sensors would prove to be very efficient in SLAM. Two important considerations when it comes to fusion of sensors are the calibration of the sensors to be fused and the approaches/ architecture of fusing the sensors. One of the approaches of the fusion is:

- Camera resolution is much higher than a lidars but less depth information. Thus, a gaussian process regression was performed to interpolate the missing depth values using the lidar. Thus, Lidars where used to initiate the features declared in the images.
- When the visual tracking is unsuccessful, the Lidar pose can be used to localize the point clouds data of the RGBD camera to build the 3D map.

A few sensor fusion methods are:

- Kalman filtering
- Optimization based methods

Kalman filtering is again divided into 2 types:

- Loosely coupled (fuse processed information's from individual sensors)
- Tightly coupled (fuse raw measurements directly)

Optimization-based methods are mostly tight-coupled.
The advantages of using Kalman filter for sensor fusion are:

- Sensor data fusion can be easily implemented using Kalman filters as it already has mathematical models inbuilt.
- Increases the on-line estimation by reducing the noise and bias.

Camera-GPS-IMU sensor fusion for autonomous flying

The above-mentioned research paper, explains state estimation for aerial robots in particular using the sensor fusion of Camera-GPS-IMU using EKF filters for ease and robustness. Generally, the state estimation of an UAV is done with the fusion of GPS and IMU. But GPS has a lot of disadvantages. Some of the major disadvantages are slow update rate, low accuracy with position drift. Thus, the paper suggests using camera thus fused with the IMU-GPS combination to increase the update rate and position drift of the estimation process. One notable advantage with this fusion of camera along with GPS and IMU using EKF as observed from the experimental study of the author is that, even when GPS fails due to some reception glitches it was observed that the IMU-Camera fusion gave very reasonable estimates that is very close to the IMU-GPS-Camera fusion. Thus, the author concludes by saying that the Camera-IMU-GPS sensor fusion can enhance accuracy and robustness of the aerial robots.

"Vision-Controlled Micro Flying Robots from System Design to Autonomous Navigation and Mapping in GPS-Denied Environments"

In this paper the author first implemented Visual EKF based SLAM. But the absolute scale in this technique has to be manually set by the user. Thus, the system was extended by adding an IMU sensor. Thus, the estimate of the absolute scale will be automatically set while calibrating. Also, the author used GPS only in the starting of the system to initialize the scaling factor with respect to the ground truth. The main usage of GPS in the beginning was to align all the states for a simpler comparison with the ground truth and also that it would be done quickly. GPS was no longer used as input in the EKF framework apart from the initialization phase.

VISUAL SLAM

Visual SLAM is a type of SLAM that leverages 3D vision to perform localization and mapping of the unknown environment. Thus, using visual inputs of the camera, the position and the orientation of the robot with respect to its surrounding is calculated and mapped simultaneously. Visual Odometry (VO) is the process of estimating the motion of the robot with the help of sequence of images of the environment. VO is divided into monocular and stereo camera methods which can be further divided into feature making, feature tracking and optical flow techniques. Feature tracking is the process of tracking and monitoring particular features in between adjacent images like the window or a door in a particular image. It is basically used to monitor variations in motion in between frames. Feature matching is the process of extracting individual features and matching those features over multiple frames. They are significantly useful when there is a change in the appearance of the features occurring due to monitoring over a longer sequence. Some of the feature extraction techniques are:

- Harris Corner Detection
- FAST Corners
- SIFT Features
- SURF Features
- 3D Descriptors

FAST CORNER

It is one of the fastest feature extraction technique which could be used for extracting features and then to track and map. It is best for live real-time application point of view with efficient computation.

METHOD:

It takes a pixel (p) from the image and circle it with 16 pixels called as the Bresenham circle as the first step to detect corners. Now we will label each pixel from 1 to 16. Then check random N labels in the circle if the intensity of the labeled pixel is brighter than the pixel around which 16 pixels has been selected. Thus, the conditions for the pixel p to be a corner is that:

- The intensity of x should be greater than the intensity of p + threshold
- Or the intensity of x should be less than the intensity of p – threshold

The main significant step in this method is the point where we decide the N and the threshold. N is the pixels out of the 16 pixels to be considered. More the N, more accurate but more computation. N is usually taken as 12. So, this trade-off has to be considered smartly to get desirable results. The difference between the pixel p and the surrounding N pixel values is calculated and stored in a variable V. Now 2 adjacent

keypoints will be considered and its corresponding V value will be calculated. The lowest V value will be castoff. This way multiple interest points will be avoided.

DISADVANTAGE:

It will not work properly if there are no contrasting pixels around the center of the pixel p. Thus, we will have to add a blur using Gaussian filters. This will give less precise but a corner.
The major disadvantages of corner detection algorithms are the scaling invariance. That's when a corner is zoomed in, it will represent an edge more than a corner. Thus, came the SIFT or Scale Invariant Feature Transform which solves this scaling issue. The major advantage of this algorithm is that it is invariant to

- Image brightness, contrast
- Rotation
- Scaling

SIFT

The working of SIFT algorithm can be split into 5 topics:

- Scale-Space Extrema Detection
- Keypoint Localization
- Orientation Assignment
- Keypoint Descriptor
- Keypoint Matching

Scale-Space Extrema Detection

The process involved in this step is described below:

- The image is blurred using Gaussian filters.
- Octave is taken. That's a pixel will be selected and compared with the 8 pixels around it. These 9 pixels are considered to be an octave.
- Now 2 more octaves above and below the pixel we took is taken.
- Now, we will check if there is a local extremum and if yes, then it will be considered a potential keypoint.

Keypoint Localization

This process majorly does 2 important things. They are:

- Removing edges using eigen values and ratios
- Removing the extrema of intensities lesser than the threshold values set by us

Thus, removal of low contrast keypoints and edge keypoints happens in this step.

Orientation Assignment

In this step orientation will be assigned to the keypoints. With these orientations for the keypoints being created, an orientation histogram will be derived from the orientations. And when the orientation is more than 80% for a keypoint that keypoint will be taken into consideration for calculating the orientation.

Keypoint Descriptor

It is the step where vectors are derived from the orientation assignment step. The size of the vector is the number of keypoints * 128. The algorithm will create a 16*16 neighbor around the keypoint in which each subblock will have 4*4 pixels. Now, each of this pixel will have 8 pixels around it. Thus, 8*4*4 gives 128.

Keypoint Matching

It is the process where keypoints are matched between images irrespective of the rotation, contrast or the scaling. This is done by identifying the nearest neighbors and doing ratio analysis between closest and second closest neighbors. Also, if this ratio is greater than 0.8, then the keypoints are rejected. This is done to avoid closest match of neighbors which happens due to noises.

BRIEF

It uses binary strings as feature point descriptor. In other words, instead of the 128-bin vector in the SIFT, we will use binary strings in BRIEF. BRIEF stands for Binary Robust Independent Elementary Features. It easily outperforms SIFT in terms of speed and recognition rate. Here, the neighbors around the keypoint are called patches. Thus, the major function of BRIEF is to identify the patches around the keypoint and convert it into a binary vector, so that they can represent an object. Thus, each keypoint will be described with the help of the binary 1's and 0's. One major consideration about the BRIEF is that it is very noise sensitive as it deals closely with pixel-level images. Thus, smoothening is very important to reduce the noise from the image. Smoothening is done using Gaussian Kernel in BRIEF. Once smoothening is done, the next step is the feature descriptor.

$$\text{Where } \tau(p;x,y) \text{ is defined as:}$$
$$\tau(p;x,y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases}$$
$$p(x) \text{ is the intensity value at pixel } x.$$

Now the problem will come when we will have to calculate the n (x, y), that's the x, y pairs. This (x, y) pairs are called random pair inside the patch. N is the length of the binary vector and τ is the binary test response of the vector. Finding the random pair can be easily done if the length of the binary vector is decided as we can pick from the patch easily then. There are 5 different methods used to calculate the length. They are uniform (GI), gaussian (GII), gaussian (GIII), coarse polar grid (GIV), coarse polar grid (GV).

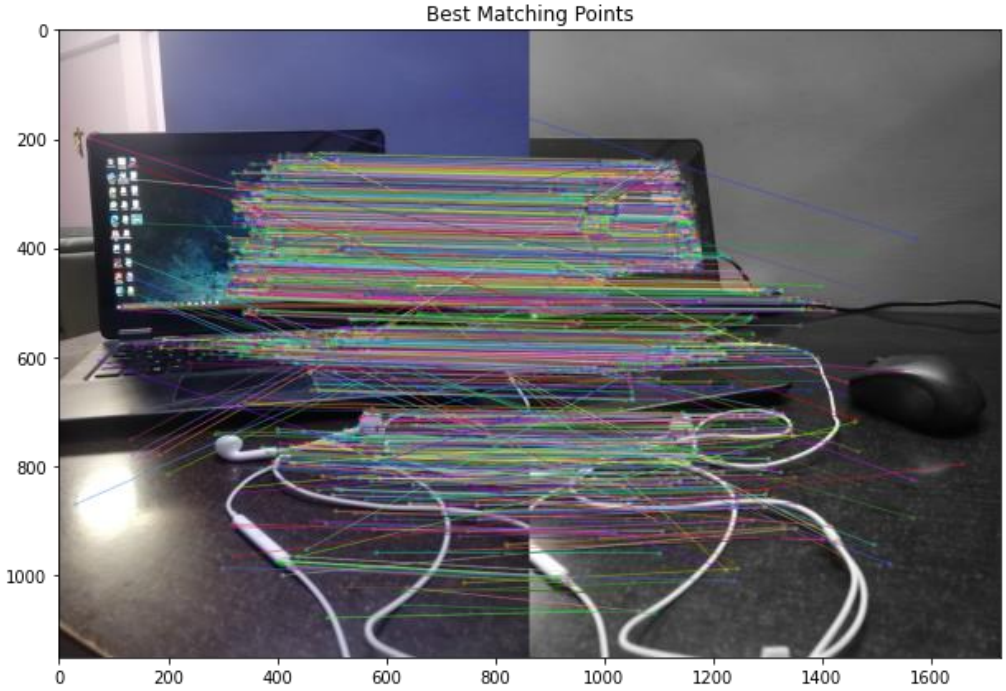
ADVANTAGES

Faster than SIFT in terms of speed and recognition rate. Also, BRIEF relies on less intensity difference between the image patches.

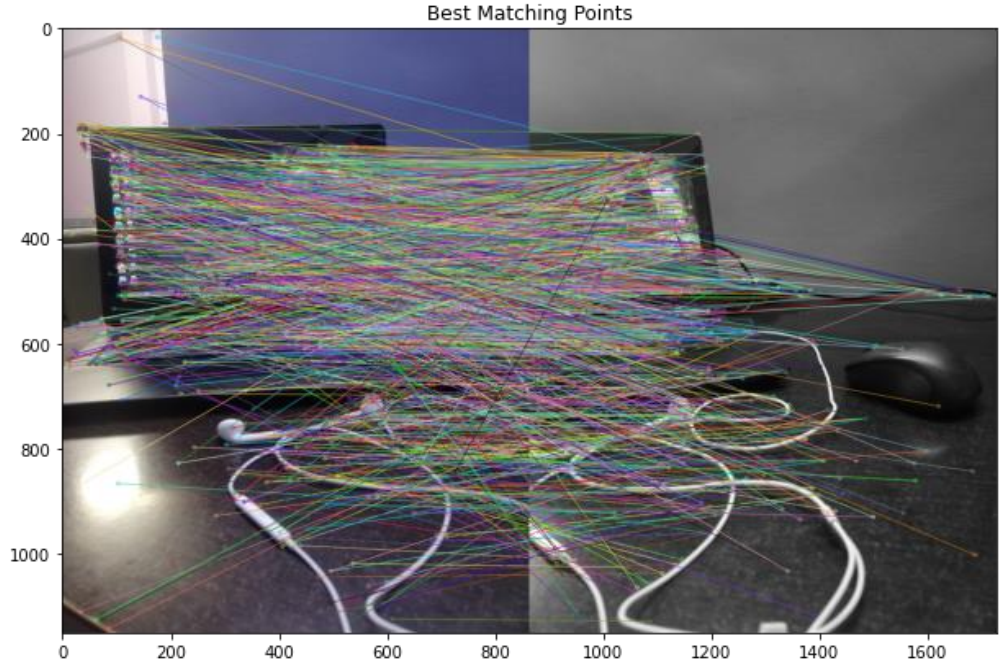
ORB (Oriented FAST and Rotated BRIEF)

FAST algorithm is used to calculate and find the keypoints from the image. Thus, it was most often used to detect and differentiate edges, corner and flat surfaces. It doesn't calculate the orientation just like SIFT method. Thus, thus algorithm ORB, solves this issue. Also FAST isn't invariant of the scaling of the

image. ORB also solves this issue. The computation of ORB is done using BRIEF algorithm., Thus it is a mixture of both FAST and BRIEF making it better than SIFT in terms of speed and computation.



Number of Matching Keypoints Between the Training and Query Images: 2455
Implementation with FAST detectors and BRIEF descriptors



Number of Matching Keypoints Between the Training and Query Images: 1871
Implementation using SIFT feature extractors

Development of a real-life EKF based SLAM system for mobile robots employing vision sensing

In this paper, the author used stereo camera (two webcams) for a ground robot to implement the 'observe' part of the system. The 'observe' step consists of image feature selection, tracking of the features selected. The feature selection is selecting of the patches/ windows which can be used for tracking efficiently in subsequent frames. Here, for feature extraction they used a technique called KLT/ Kanade-Lucas-Tomasi Tracker. The major significance of this KLT tracker is that one of the stereo cameras (let's take right) will be used to obtain features via patches/ windows and the camera at left will be used to track the features obtained from processing the image from the right camera. More pixels are chosen as a single pixel will be hard to keep track, due to varies issues like contrast in intensities or confusions due to noises in the data. Thus, N patches/ windows are preferred for tracking the features. This way the features can be selected and tracked.

The next important step is calculating the 3D distance between our location and the feature. The line of sight of both the cameras intersects at a point say for instance P. Now, with this intersection points the point clouds can be obtained.

Once the landmark is identified and tracked, they can be initialized in map utilizing the usual procedure in EKF-based SLAM algorithm. That's the algorithm goes as such:

- Waypoints will be initialized for navigation.
- Traversing of the robot and performing the predict step of EKF
- Observing- Extracting the features and tracking of the features and also 3D distance calculating of the features from the robot.
- Add the features attained to the unknown map of the environment. Then continue the normal observe and update step of the EKF algorithm.
- Now, repeat the same steps again and again updating the new landmarks/ features to the map we started constructing in the fourth step.
- These steps should now be iterated till the final stated waypoint is reached.

VSLAM approaches:

- Feature based approach
- Direct approach
- RGBD Approaches

FEATURE BASED APPROACH

This is again divided into 2 approaches- Filter based and bundle based. Example for filter based is MonoSLAM and for bundle adjustment it is PTAM.

MonoSLAM

In MonoSLAM, camera motion and 3D structure of an unknown environment are simultaneously estimated using an extended Kalman filter. camera motion and 3D positions of feature points are represented as a state vector in EKF and the tracking of this features is observed in MonoSLAM. Various methods for this process of feature detection and tracking have been discussed earlier. The issue with this kind of approach is that when the size of the room/ world increases, the state vector will also increase causing more computation. Thus, BA algorithms were introduced to overcome these issues.

PTAM

It is an BA algorithm/ technique. Bundle Adjustment can be defined as simultaneous refining of the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera employed to acquire the images. The tracking and mapping are done parallelly to save the computation cost. PTAM is a hybrid of:

- BA part
- Localization with known landmarks
- VO

The basic approach of the PTAM is:

- Initializing the map using the five-point algorithm. Five-point algorithm is essentially taking 5 points in between 2 images and the rotational and translational matrix along with the essential matrix is found.
- Camara poses are estimated from the previously obtained feature points by projecting on the image thus making a correspondence.
- 3D position of the feature points is estimated using the triangulation method.
- Tracking by randomized tree-based classifiers for searching the nearest keypoint.

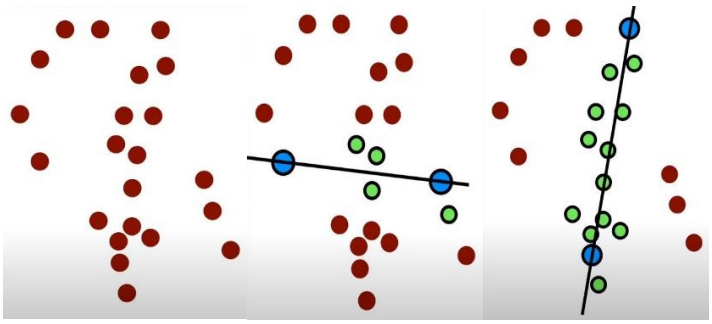
BA-BASED APPROACH VS EKF-BASED APPROACH

It is known that more the keypoints, more will be the accuracy and robustness. But more the keypoints/ features selected more will be the computation cost. But due to the parallel tracking of the BA based approaches, they can handle a greater number of keypoints than the filter-based approaches with similar computation cost. Thus, BA-Based approaches are more robust than filter-based approaches.

RANSAC

RANSAC stands for Random Sample Consensus. Deals with removing outliers from inliers contained in a data. No real-world sensor readings are perfect. Thus, we use RANSAC which is a simple trial and error method with groups the inliers and outliers separately. Thus, it helps us to throw away the outliers and work with inliers alone which will save our computation and time. It involves a 4-step processing, and they are:

- Sampling
- Compute
- Score
- Repeat



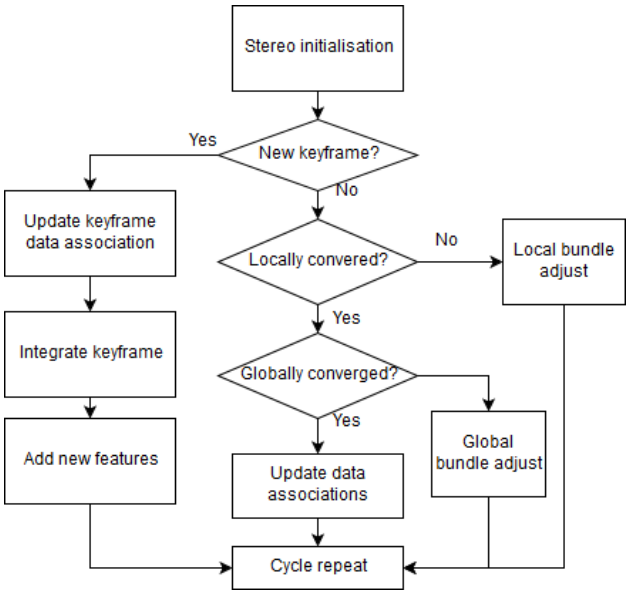
1 2 3

Let's assume there are 2D points (Figure 1) in which a line has to be fitted. Now we will sample and randomly take one line marked in figure 2 as our inlier. Now we will compute the number of supporting inliers satisfying the line we drew. This will be done by parallelly extending imaginary lines on both the side of the line assumed to be the inlier with a uniform distance of **δ.** Now each and every point lying inside these imaginary lines constitute to the scoring of the inliers. Now for the figure 2 there are 4 inliers.

Now we will repeat the steps mentioned above repeatedly and the score will be overwritten with the best score after every iteration. In the 3rd image, that's after some iterations, we got a line which has 12 points satisfying that line model which is the best score to be obtained. Thus, it will be best fit for the line. This way we can eliminate all the outliers easily.

PTAM

PTAM is an BA algorithm/ technique. Bundle Adjustment can be defined as simultaneous refining of the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera employed to acquire the images. The tracking and mapping are done parallelly to save the computation cost.



5 POINTS OF PTAM:

- Tracking and mapping is separated and run in 2 parallel threads
- Mapping is based on keyframes, which are processed using BA
- The map is densely initialized from a 5-Point Algorithm
- New points are initialized with an epipolar search.
- Large numbers of points are mapped.

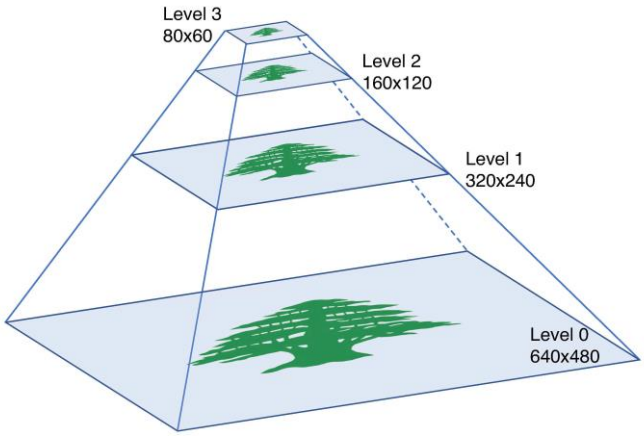
TRACKING AND MAPPING

It is done parallelly in different threads (nowadays, all computers have more than one core). The major advantage of processing separately is:

- Tracking will not be slaved to the map making procedure
- More thorough processing can be done as the computational burden to update a map in every frame.
- Every single frame isn't necessary to be used, as there will be a lot of redundance, especially when the robot is hovering.

Now, we will start explaining tracking and mapping individually with the architecture in detail.

TRACKING



4-level pyramid representation of an image

PTAM generated a 4-level pyramid representation of every frame it obtains as shown in the above figure. It uses this structured data to enhance the features robustness towards scale changes and to increase the convergence ratio of the pose estimates. Each level is formed by blurring the level before it and sub-sampling by a factor of 2.

IMAGE ACQUISITION

The captured images are converted into 8 bits per pixel greyscale for tracking purposes. On each of the pyramid levels, FAST-10 corner detector algorithm is run resulting in a blob-like cluster of corner regions.

CAMERA POSE AND PROJECTION

Projecting map points into the image is vital for tracking. This is done by first converting the world coordinate frame to a camera-centered coordinate frame. This conversion is done by left multiplication with the camera pose.

CW represents frame C from frame W. J^{th} point of the map is represented by pj. The transformation between the camera-centered coordinate frame and the world is called $E_{CW.}$ Thus, in the first equation we will find a particular point on the map in the coordinate frame W.

$$p_{jC} = E_{CW} p_{jW}$$

CamProj() is a camera calibrated projection model used to projecting the points in the camera frame to the image.

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \text{CamProj}(E_{CW}\boldsymbol{p}_{iW})$$

The below equations show the pin-hole camera projection parameters including focal length, principal points and the distortion

$$\text{CamProj}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \frac{\mathrm{r}'}{\mathrm{r}} \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix}$$

$$r = \sqrt{\frac{x^2 + y^2}{z^2}}$$

$$r' = \frac{1}{\omega} \arctan(2r \tan\frac{\omega}{2})$$

The major requirement of tracking is to observe change in the change in the position of the map with respect to change in the camera pose. This is shown in the below equation which is done using the left multiplication with respect to the camera motion which is represented by the letter M.

$$E'_{CW} = ME_{CW} = \exp(\boldsymbol{\mu})E_{CW}$$

μ is the 6-vector parameter used here, representing 3 translational and 3 rotation axis and magnitude.

PATCHING

To find a single point p in the map, a fixed range image search is done around the points of the predicted image. To perform this search, the corresponding patch must first be warped to take account of viewpoint changes between the patch's first observation and the current camera position. So, the warping matrix is shown below where u and v are the horizontal and vertical pixel displacement in the patch's source pyramidal level and u_c and v_c are pixel displacement of the current camera frame.

$$A = \begin{bmatrix} \frac{\partial u_c}{\partial u_s} & \frac{\partial u_c}{\partial v_s} \\ \frac{\partial v_c}{\partial u_s} & \frac{\partial v_c}{\partial v_s} \end{bmatrix}$$

The differential of the A matrix tells us in which pyramid level, the patch should be searched.

POSE UPDATE

From the patch observations obtained the pose of the camera can be computed. From each observation, patches pose is found with an assumption of noise measured. Thus, we will calculate the pose by iterating continuously till convergence is observed from any set of measurements.

Obj is the objective function and **σ** represents the robust estimate of the standard deviation of the distribution derived.

$$\boldsymbol{\mu}' = \underset{\boldsymbol{\mu}}{\text{argmin}} \sum_{j \in S} \text{Obj}\left(\frac{|\boldsymbol{e}_j|}{\sigma_j}, \sigma_T\right)$$

$$\boldsymbol{e}_j = \begin{pmatrix} \hat{u}_j \\ \hat{v}_j \end{pmatrix} - \text{CamProj}\left(\exp\left(\boldsymbol{\mu}\right) E_{CW}\boldsymbol{p}_j\right)$$

The pose update is thus computed iteratively by minimizing the objective function of the reprojection error.

TRACKING

At first, a frame will be taken from the live video and a prior estimate for the camera is generated from the motion model. Depending on the prior pose estimate the map points will be visualized. To increase the tracking resilience to rapid camera movement the patch search and pose update will be done twice in PTAM. Both the cycle is explained below:

- A small amt (50) of coarse scale features will be searched in the image. It is done in the highest level of the pyramid representation of the image. From these coarse matched features, the camera pose will be measured and updated.
- A large number of points (1000) are now re-projected from the remaining potential points obtained and searched in the image over a smaller range.

MAPPING

Map building occurs in 2 steps. They are:

- Building the initial map using stereo technique.
- Map will be continuously refined and expanded as new keyframes are added by the tracking system.

These 2 steps are individually explained in the coming pages.

INITIAL MAP

We use 5-point algorithm to initialize the map initially. The initial map will consist of only 2 keyframes taken by user upon which the 5-point algorithm and the RANSAC can estimate and triangulate the base map.

REFINING AND EXPANDING MAP

As the camera moves away from the 2 keyframes formed in the initial map, new keyframes and map features are thus added to the system to allow the map to expand.

REFINEMENTS

BUNDLE ADJUSTMENTS

It is defined as simultaneous refinement of the 3D coordinate describing the scene geometry and the optical characteristics of the camera giving input images. It is basically applied as the very last step of feature-based 3D reconstruction. It is used to remove the noise pertaining the image features observed.

DATA ASSOCIATION REFINEMENT

Once BA has converged, data association will be done. New measurements will be made in the old keyframes. When new features are added, they will be checked if it is an inlier or outlier. If outlier it

will be cast off the map. This will be very useful in cases where tracking goes wrong which happens. If the measurements are given low weights by the M-estimator in the BA refinement process, they will be considered to be outlier. A second chance is given by the data association refinement, where its keypoint will be re-measured in the keyframe with a very tight search region. If outlier even after this measurement, it will be cast off permanently. If it is an inlier it will be added to the map.

This process is given the least priority in the hierarchy. Only if there is no process at hand to be done, this refinement will be done. That's, only if there are no other keyframes to be measured, it is done and once a new keyframe comes in it will be abruptly stopped.

VISUAL INERTIAL SLAM

Adding inertial sensors to the camera to estimate robust pose of a robot is VI-SLAM. IMUs are thus fused to IMU data for localization. It is divided into 2 types:

FILTERING BASED

It is again divided into loosely coupled and tightly coupled depending on the sensor fusion type. The loosely coupled method usually fuses the IMU to estimate the orientation and change in position. The tightly coupled method fuses the state of the camera and IMU together into the motion and observation equation and then performs state estimation.
VI-SLAM has 3 strategies for feature tracking:

- Feature Extraction + Descriptor Matching
- Feature Extraction + Filter Based Tracking
- Feature Extraction + Optical Flow Tracking

Feature extraction in VI-SLAM can be done using Harris, FAST, ORB, SIFT and SURF. When it comes to feature tracking as mentioned above one of the 3 types will be used. Descriptor matching and filter based matching methods model the target area in the current frame and predict positions by finding similar area in the model in the next frame. Optical flow is an efficient way of estimating the motion. It relates the movement in the image brightness mode and expresses the change in an image.

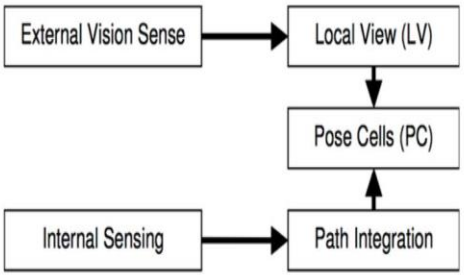
OPTIMIZATION BASED

Some of the optimization-based techniques are:
OKVIS is a keyframe based VI-SLAM technique that will combine the IMU and reprojection error terms into the cost function to optimize the system. For initializing and matching, it acquires the IMU measurement thus having a preliminary uncertain estimate. Harris corner detector combined with BRISK descriptor is used for extracting features. Initially, keypoints are stereo-triangulated and inserted into a local map. Then brute-force matching of the map landmarks are done and the outliers are removed using chi-square test. RANSAC is not used as it is computationally expensive.
VIORB is a mono tightly coupled VI-SLAM based on ORB-SLAM. Basically, it will contain the front end of ORB SLAM, along with optimization backend, loop closure and relocation. Also, bundle adjustment is used to optimize the keyframes in the local map.
VINS-mono uses the front end of KLT optical flow to track Harris corner, while the backend uses sliding window for nonlinear optimization. Optical Flow recovers image motion at each pixel from spatial-temporal image brightness variation. So, the system consists of estimation, local BA without

relocalization, loop closure and global pose optimization. In the front-end RANSAC is used to remove the outliers.
Stereo Multi-State Constraint Kalman filter system (S-MSCKF), which is a state-of-the-art VIO system with low computational cost and good robustness. Synchronized stereo images and IMU messages are fused and a real time 6DOF pose estimation of the IMU frame is generated.

RATSLAM

It is a biologically inspired approach hoping for robotic navigation without the requirement of costly sensors and computationally intensive algorithms. It is a SLAM system based on the combination of appearance based visual scene matching, competitive attractor networks and semi-metric topological map representation. Attractor networks are neural networks that Is designed to converge a stable pattern of activation across its unit. It takes in mono images and odometry as standard messages. It integrates the odometry data along with the landmark sensing using a competitive attractor network thus getting a consistent representation of the environment. The robot thus navigates just like the rat moves in an unknown environment and when it finds a known landmark, the neuron will spike and note the errors and keep in check just mimicking the rat's approach.



Pose Cell

Pose cell encodes the robot's orientation and location at the same time. So, when the robot sees the landmark, it will precisely encode its place using the pose cell and keep errors in check.

Local View

Local View encodes the visual information's in a one-dimensional array of cells. Here, using camera data, locations are estimated as patterns by an appearance-based view recognition. Thus, camera information's are matched with the growing dataset. Once an image feature is recognized, appropriate cell is activated.

Mapping and Re-Localization

Here, the mapping is done by associate learning and the re-localization is done by adding the activation to the pose cell. So once, an activity is found in the local view, the RATSLAM algorithm will send it to the pose cell, by which the correct pose estimate is made.