SLAM TECHNIQUES AND IMPLEMENTATION FOR AERIAL ROBOTICS

ATOM ROBOTICS
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.
Keywords: SLAM, Localization, Mapping, Optimization.

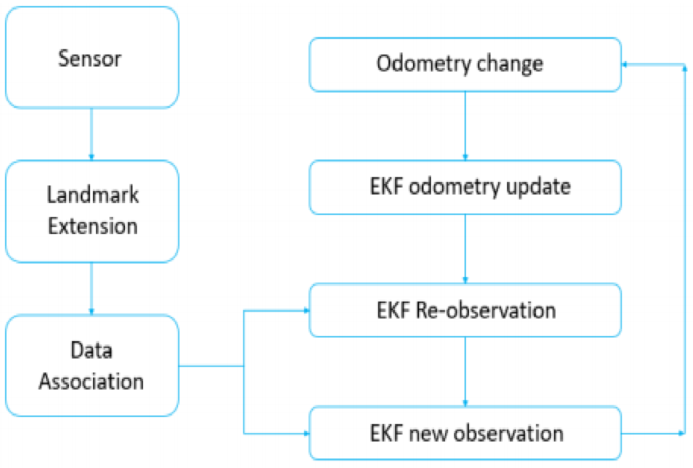
WHAT IS SLAM?

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
- Augmented Reality

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 are:

- 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.

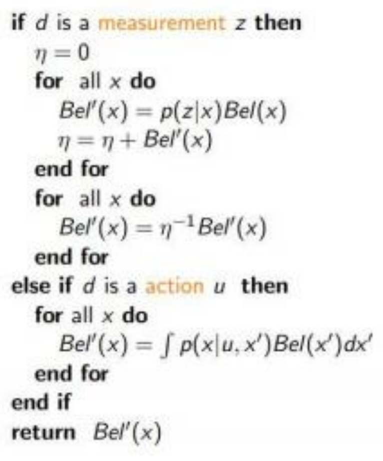
COMPARING EKF WITH PFL

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.