Tech Review

Thomas Korsness

CS461

Group 61/62

GFR SLAM

Autonomous vehicles need to take in information from sensors in order to know how to react. These sensors get data that is used to calculate the environment the vehicle is in, as well as where it is located within that environment. This process is known as SLAM, or Simultaneous Localization and Mapping. In this paper we will be taking a look at options for three different topics. The first topic is SLAM method, the second topic is method implementation, and the final topic is simulation and testing.

CONTENTS

1 SLAM Method 3
1.1 Kalman Filter . . . 3
1.2 Particle Filter . . . 3
1.3 Graph Based . . . 3

2 Graph Based 4
2.1 g2o . . . 4
2.2 SSA . . . 5
2.3 LAGO . . . 5

3 Simulation and Testing 5
3.1 RVIZ . . . 5
3.2 Gazebo . . . 5
3.3 Unity . . . 6

4 Conclusion 6

1 SLAM METHOD

SLAM, or Simultaneous Localization and Mapping, is a common problem in the world of robotics and has many well tested solutions. The three methods of SLAM that we will be looking at are Kalman filter, particle filter, and graph based.

1.1 Kalman Filter

Kalman filters have been the go to for SLAM in the past. They have become less popular as of late but still offer a viable solution to the SLAM problem. A kalman filter is an estimation algorithm. What we want the kalman filter to do is estimate the boundaries of a race track, as well as estimate the location of the car inside the race track. Kalman filters are able to take data from multiple sources and use it to make the best guess for the position of the car or track. This is useful because sensor data tends to drift, so the more sensors, the more accurate the estimation. Kalman filter are also iterative. This means that the estimation that the kalman filter produces, is then used in the next filtering process. This along with the sensor data allows the kalman filter to produce a more accurate estimate. As this process repeats the estimation becomes more and more accurate. The kalman filter is a good option because it doesn't take much computational power compared to some SLAM options.

1.2 Particle Filter

Particle filters are currently one of the more popular methods of SLAM. A particle filter will plot a large number of particles indicating the position of the car or track. Each of these are estimating the location at a single point in time. Every time the filter gets a new set of data, the particles update. The particles move to a new location indicating the most likely position of the car according to the data. A large clump of these particles in one area indicates that there is a high probability that this is where the car is located. This is shown in figure 1. More particles are present in the location that the robot thinks it is in. Particle filters also have weights associated with each particle. These weights indicate that the particle has more confidence in its estimate of the location. The downside of the particle filter is the amount of particles. The more particles, the more computationally expensive.

1.3 Graph Based

Graph based SLAM is a less popular method to solve the SLAM problem. In graph based SLAM, a graph is created and it creates nodes that represent the location of the car as well as landmarks sensed by the car. There are also edges which are lines drawn between two nodes. The lines are constraints that are measured by sensors on the car. These constraints tell us how far the vehicle travelled between nodes(2). While this method is easy to understand, they can become computationally expensive when a lot of nodes are tracked. In our case, we have a large number of cones outlining a race track. These cones are used as landmarks and a new node and edge would be made for each one. This makes graph based difficult with our application.

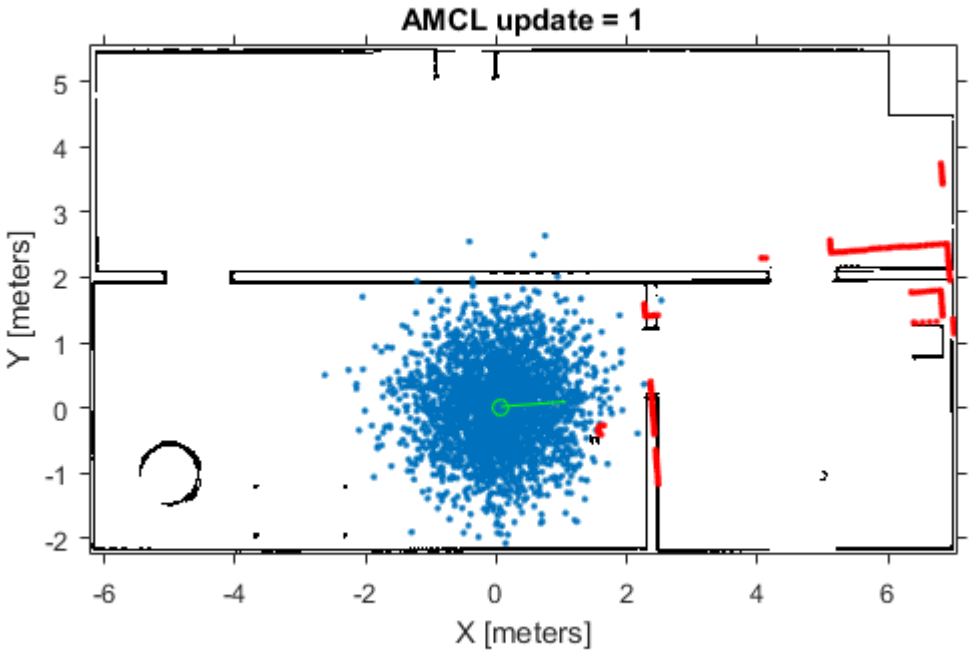

Fig. 1. Particle Filter (1)

2 Graph Based

Graph SLAM may not be as popular as particle, but it still has plenty of implementations that solve the SLAM problem. As mentioned above, graph slam uses nodes and edges to build a map and localize the robot within it. We will be looking at three different implementations of graph SLAM and how they differ. The first is g2o, the second is SSA, and the final implementation is LAGO.

2.1 g2o

G2o is a general solution to graph-based nonlinear error functions. This means it has applications outside of SLAM such as bundle adjustment. G2o is a C++ framework, which works well for us because ROS works with Python and C++. G2o also allows for the addition of problems to the open source code. This makes it very easy to customize for specific applications. This implementation of graph SLAM also allows for both 2D and 3D versions. This is helpful because 2D is simpler to implement while 3D is more robust. G2o gives the user a choice between the two(3).

2.2 SSA

SSA is another C++ tool for SLAM applications. SSA refines the pose of the robot in an iterative manner. When doing this SSA will produce an accurate 2D laser-based map (4). SSA only builds 2D maps, which keeps thing simple, but can create issues down the line. If the ground that the car is travelling on is not completely flat, the map will be slightly off. For our driverless car application, the ground is meant to be flat, but there is always uncertainty involved on paved roads, so a 3D system would be preferable.

2.3 LAGO

LAGO is a linear approximation for graph SLAM. Similar to SSA, LAGO is a 2D implementation(5). One of the big benefits of LAGO is its ability to function without an accurate first guess of location. This has been an issue for SLAM implementations in the past. A downside to LAGO is that it doesn't completely solve the SLAM problem. It works in conjunction with another graph based SLAM implementation. LAGO deals with linear approximation. This means that LAGO will optimize the graph and optimize the connections, or edges, between nodes. The fact that LAGO doesn't completely solve SLAM makes it more difficult to work with because we would have to connect two different implementations.

3 Simulation and Testing

SLAM is a large part of the overall success of the driverless car. We will not be able to test on the car for quite some time, so some simulations are necessary in order to get the result we want. SLAM is also a difficult application to test for. We need to know exact locations of cones to compare against our estimates. This is really only possible in a simulated environment. The three simulation software packages we will be looking at are Rviz, Gazebo, and Unity.

3.1 RVIZ

Rviz is a 3D vizualization tool that is used in ROS. ROS, or robot operating system, is being used to help different portions of the car communicate with each other. Rviz will display what the car is seeing. Without a simulation software, we would have to debug by looking at number representing a 3D world which can become very difficult. One of the benefits of rviz is that it is a ROS package. All the programming we do will be inside ROS so this will make it easier to implement with our code.

3.2 Gazebo

Gazebo is a simulation software that can simulate robots and their environments. Gazebo is also a ROS package which will make implementation easier. Another benefit to Gazebo is that there is already a package built that simulates a formula student driverless race car. This will speed things up for us as we will not need to construct a new environment in the software. With this software we will be able to build a track and have exact locations of cones. This will allow us to test our SLAM results. With this we will also be able to set performance metrics. For example we can make a goal of having our estimated map within one inch of of the actual map. This would be almost impossible outside of a simulation.

3.3 Unity

Unity is a 3D game engine, but has applications beyond just making games. There are quite a few open source libraries that allow us to simulate driverless cars with lidar. ROS is also commonly used in conjunction with ROS packages. A downside to Unity is that it's a huge piece of software, and it takes a lot of work to master. It may not be in our best interests to pursue such a large program when we have approaching deadlines.

4 Conclusion

After researching these technology options, the team has landed on a few decisions. The SLAM method we chose is the particle filter. We chose it because of it's more robust capabilities in comparison to the kalman filter. We also chose to use a feature based mapping system. This means we will not be using any of the graph based open source packages. For the simulation we chose to go with Gazebo. We chose it because it already has a package that works for driverless car simulation. We know this software works so this will cut down on testing time.

References

[1] mathworks, "turtlebot using a monte carlo localization," 2010. [Online]. Available: https://jp.mathworks.com/help/robotics/examples/localize-turtlebot-using-monte-carlo-localization.html

[2] C. S. W. B. Giorgio Grisetti, Rainer Kummerle, "A tutorial on graph-based slam," University of Freiburg, Tech. Rep., 2010. [Online]. Available: http://www2.informatik.uni-freiburg.de/ stachnis/pdf/grisetti10titsmag.pdf

[3] H. S. K. K. W. B. Rainer Kummerle, Giorgio Grisetti, "g2o: A general framework for graph optimization," Tech. Rep., 2011. [Online]. Available: http://ais.informatik.uni-freiburg.de/publications/papers/kuemmerle11icra.pdf

[4] Kudan. (2011) Ssa: Sparse surface adjustment 2d. [Online]. Available: https://openslam-org.github.io/ssa2d.html

[5] J. A. C. Luca Carlone, Rosario Aragues and B. Bona, "A fast and accurate approximation for planar pose graph optimization," Tech. Rep., 2014. [Online]. Available: https://journals.sagepub.com/doi/abs/10.1177/0278364914523689