Oregon State University
College of Engineering

CS Capstone Design Document

May 14, 2020

Global Formula Racing Vehicle Stack

Prepared for

Global Formula Racing

Kyle O'Brien ______________ Signature ______ Date

Prepared by

Group 49
GFR Capstone

Monica Holliday ______________ Signature ______ Date

Mitchell Schenk ______________ Signature ______ Date

Yuchen Wen ______________ Signature ______ Date

Abstract

For the Global Formula Racing team at Oregon State University to compete and be competitive in the Formula SAE driverless vehicle competition, a high performance autonomous vehicle needs to be designed and created. To achieve the goals of the team, the autonomous system will need to be robust, reliable, and capable of directing the car around a track as fast as possible. The system will need to gather accurate observations about the environment and the location of the vehicle, create a map with these observations, and plan a path that produces a fast lap time. This document details the design of the Simultaneous Localization and Mapping(SLAM) software, which will be be an implementation of an algorithm called FastSLAM2.0 with improvements to data association and particle weighting. The document also details the design of the trajectory planning software, which will be comprised of systems for estimating boundaries and triangulating from these boundaries to find the desired path.

Contents

1 Introduction

1.1 Purpose

The purpose of the software described in this document is to function as an autonomous racing vehicle capable of competing in and winning the Formula SAE Driverless competitions during the summer of 2020. Creating an autonomous vehicle is not a simple task as there are many challenges requiring robust and often complicated solutions. The vehicle needs to be able to sense its surroundings, identify things that it needs to avoid, map its environment, determine its own location, and plan a path towards a desired destination. Each of these elements come with their own complications, requiring great care in designing and implementing solutions. In addition to creating software that enables the autonomous vehicle to compete in competition, it should perform so that the final autonomous system is accurate and fast enough to win the competitions it is entered in.

1.2 Scope

The software created for this project is limited to the tasks of identifying landmarks, creating a map of the environment, determining its location on the map, and planning a trajectory around the track. In a sense, this software is not concerned with anything analog as it only needs to create a virtual map, find its location in the virtual map, and plan a path through the virtual map. The software will not have to handle low level specifics like explicit controls to motors and actuators, filtering out noise in lidar point clouds, or determining details of cones from an image captured by the onboard camera. The design and implementation of these elements of the autonomous system will be carried out by other project teams.

Due to the details and rules of the competition the environment in which the autonomous vehicle operates in is fairly limited. This environment consists of a small race track which does not vary much in width and is marked by cones of only a few colors. Fortunately this provides some simplifications in creating software for an autonomous vehicle like knowing that the only landmarks necessary to define the track are cones, with blue ones on the left side, and yellow on the right. Ultimately the autonomous racing car only needs to work in this strictly controlled environment and will not need to handle any other cases. This assumption is factored into the design in a number of ways detailed throughout this document.

1.3 Intended Audience

The intended audience of this document is technical and organizational stakeholders on the Global Formula Racing(GFR) team. GFR will use this software on the vehicles that it creates and enters in the Formula SAE competitions. Additionally, GFR team members will use this document to understand the system so that they can justify their design choices during development and to judges at competitions. This document will also serve as a base of knowledge for future GFR team members working on this software in future years so that they do not need to repeat the research and work done in creating it.

2 Definitions

- **SLAM** Simultaneous Localization and Mapping
- **GFR** Global Formula Racing
- **SAE** Society of Automotive Engineers

- **ROS** Robot Operating System
- **Lidar** Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure ranges
- **CGAL** Computational Geometry Algorithms Library

3 Design Components

3.1 Simultaneous Localization and Mapping

3.1.1 Design Overview

An autonomous car needs to know the details of its environment and where in that environment it is so that it can operate correctly. The Simultaneous Localization and Mapping(SLAM) system on the autonomous vehicle will gather data as the car moves and use the input to create a map and determine its own location on the map. Since this project is limited to the scope of an autonomous race car operating on a track marked by cones, the cones will be the landmarks from which the map is constructed.

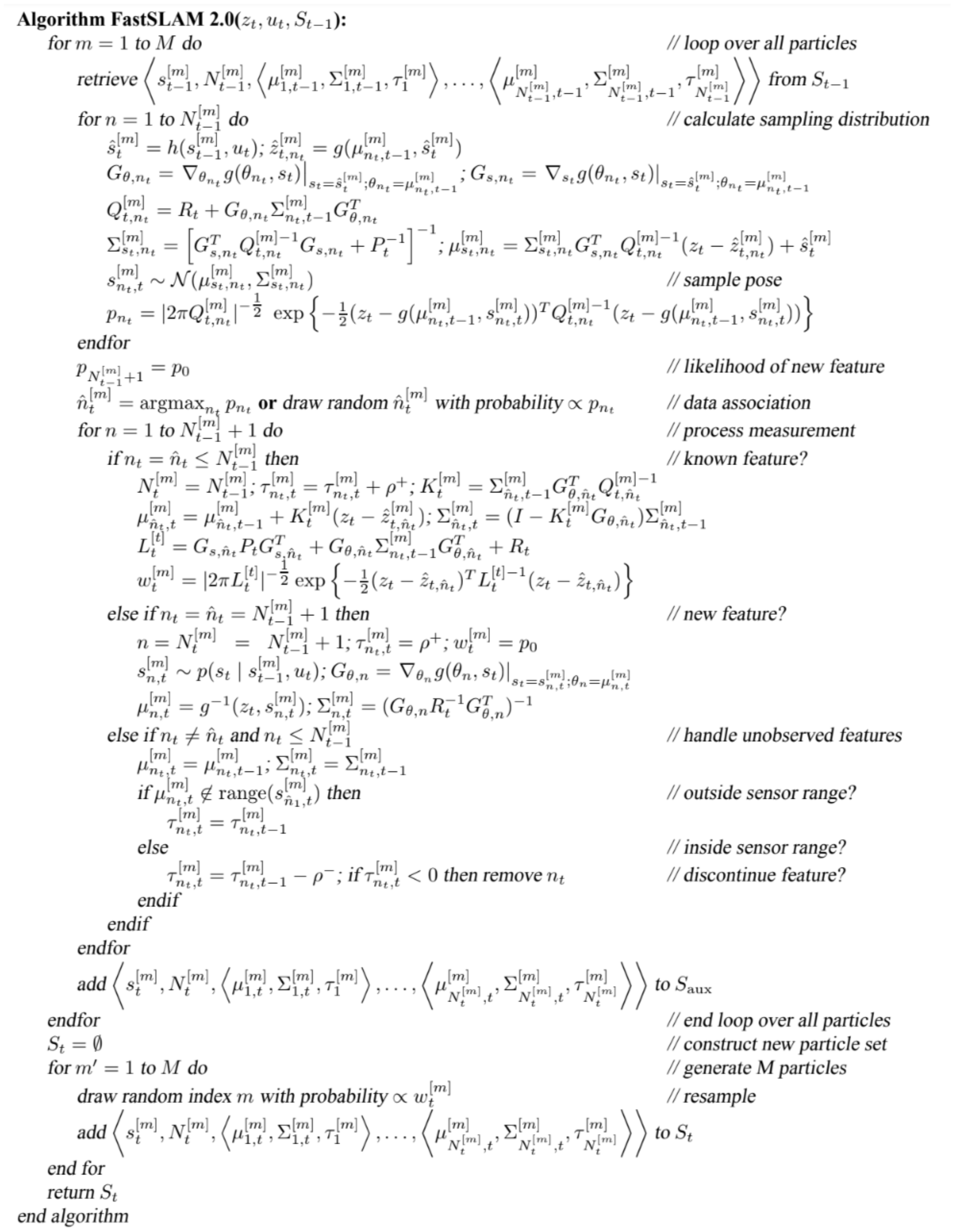
The SLAM system receives input from a number of sources including a camera, lidar sensors, and the odometry software on the vehicle. The output of the SLAM system should provide a map consisting of cone locations and the location of the vehicle. First, the team will write tests to benchmark the SLAM system from the 2019 team. Next, different implementations of SLAM will be developed with improvements to some of the major parts of the SLAM process including particle sampling, data association, and particle weighting. Lastly, this new system will be benchmarked and compared to the previous system so that elements from each version can be combined to create the most accurate and reliable final product.

3.1.2 Design Elements

Before this project began there was an implementation of SLAM that was created by the team that worked on the autonomous vehicle for the 2019 competition. The implementation, known as FastSLAM1.0, performed all steps of the SLAM process which include sampling particles using information from previous iterations, updating landmark estimates, evaluating particles, and resampling particles for use in the next iteration. While not perfect, this code does work but there are still many opportunities to make improvements. Detailed in the following sections are the parts of the SLAM process that will be improved, what the improvements are, and how they will be implemented.

3.1.2.1 Particle Sampling

The first step of the SLAM process is sampling particles to use throughout the rest of the iteration. In the SLAM paradigm a “particle” is an estimate of the current map and the vehicles pose on that map. Sampling particles simply means generating a number of guesses that represent possible states of the map and the vehicles pose. This is done by utilizing particles from the previous iteration and the most current information on the vehicles movement since this last iteration. To improve upon the current implementation of this process the implementation of the SLAM algorithm will be changed from FastSLAM1.0. The new implementation, known as FastSLAM2.0, accounts for measurements to landmarks when sampling, generating more particles that have a higher probability of accurately representing the real environment[1]. Using a particle filter, particles from the previous iteration will be combined with the new location of the vehicle and the vehicle's observations to create a number of new particles. Including the observations in this process creates more accurate particles and as a result creates a more accurate overall process which is highly desirable in the autonomous racing application. The pseudo code for the FastSLAM2.0 algorithm is included below.



```
Algorithm FastSLAM 2.0($z_t, u_t, S_{t-1}$):
    for $m = 1$ to $M$ do                                                                // loop over all particles
        retrieve $\left\langle s_{t-1}^{[m]}, N_{t-1}^{[m]}, \left\langle \mu_{1,t-1}^{[m]}, \Sigma_{1,t-1}^{[m]}, \tau_1^{[m]} \right\rangle, \ldots, \left\langle \mu_{N_{t-1}^{[m]},t-1}^{[m]}, \Sigma_{N_{t-1}^{[m]},t-1}^{[m]}, \tau_{N_{t-1}^{[m]}}^{[m]} \right\rangle \right\rangle$ from $S_{t-1}$
        for $n = 1$ to $N_{t-1}^{[m]}$ do                                               // calculate sampling distribution
            $\hat{s}_t^{[m]} = h(s_{t-1}^{[m]}, u_t); \hat{z}_{t,n_t}^{[m]} = g(\mu_{n_t,t-1}^{[m]}, \hat{s}_t^{[m]})$
            $G_{\theta,n_t} = \nabla_{\theta_{n_t}} g(\theta_{n_t}, s_t)\big|_{s_t=\hat{s}_t^{[m]};\theta_{n_t}=\mu_{n_t,t-1}^{[m]}}; G_{s,n_t} = \nabla_{s_t} g(\theta_{n_t}, s_t)\big|_{s_t=\hat{s}_t^{[m]};\theta_{n_t}=\mu_{n_t,t-1}^{[m]}}$
            $Q_{t,n_t}^{[m]} = R_t + G_{\theta,n_t}\Sigma_{n_t,t-1}^{[m]}G_{\theta,n_t}^T$
            $\Sigma_{s_t,n_t}^{[m]} = \left[G_{s,n_t}^T Q_{t,n_t}^{[m]-1} G_{s,n_t} + P_t^{-1}\right]^{-1}; \mu_{s_t,n_t}^{[m]} = \Sigma_{s_t,n_t}^{[m]} G_{s,n_t}^T Q_{t,n_t}^{[m]-1}(z_t - \hat{z}_{t,n_t}^{[m]}) + \hat{s}_t^{[m]}$
            $s_{n_t,t}^{[m]} \sim \mathcal{N}(\mu_{s_t,n_t}^{[m]}, \Sigma_{s_t,n_t}^{[m]})$                                // sample pose
            $p_{n_t} = |2\pi Q_{t,n_t}^{[m]}|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - g(\mu_{n_t,t-1}^{[m]}, s_{n_t,t}^{[m]}))^T Q_{t,n_t}^{[m]-1}(z_t - g(\mu_{n_t,t-1}^{[m]}, s_{n_t,t}^{[m]}))\right\}$
        endfor
        $p_{N_{t-1}^{[m]}+1} = p_0$                                                       // likelihood of new feature
        $\hat{n}_t^{[m]} = \text{argmax}_{n_t} p_{n_t}$ or draw random $\hat{n}_t^{[m]}$ with probability $\propto p_{n_t}$   // data association
        for $n = 1$ to $N_{t-1}^{[m]} + 1$ do                                           // process measurement
            if $n_t = \hat{n}_t \leq N_{t-1}^{[m]}$ then                                   // known feature?
                $N_t^{[m]} = N_{t-1}^{[m]}; \tau_{n_t,t}^{[m]} = \tau_{n_t,t}^{[m]} + \rho^+; K_t^{[m]} = \Sigma_{\hat{n}_t,t-1}^{[m]} G_{\theta,\hat{n}_t}^T Q_{t,\hat{n}_t}^{[m]-1}$
                $\mu_{\hat{n}_t,t}^{[m]} = \mu_{\hat{n}_t,t-1}^{[m]} + K_t^{[m]}(z_t - \hat{z}_{t,\hat{n}_t}^{[m]}); \Sigma_{\hat{n}_t,t}^{[m]} = (I - K_t^{[m]} G_{\theta,\hat{n}_t})\Sigma_{\hat{n}_t,t-1}^{[m]}$
                $L_t^{[t]} = G_{s,\hat{n}_t} P_t G_{s,\hat{n}_t}^T + G_{\theta,\hat{n}_t}\Sigma_{n_t,t-1}^{[m]} G_{\theta,\hat{n}_t}^T + R_t$
                $w_t^{[m]} = |2\pi L_t^{[t]}|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_{t,\hat{n}_t})^T L_t^{[t]-1}(z_t - \hat{z}_{t,\hat{n}_t})\right\}$
            else if $n_t = \hat{n}_t = N_{t-1}^{[m]} + 1$ then                            // new feature?
                $n = N_t^{[m]} = N_{t-1}^{[m]} + 1; \tau_{n_t,t}^{[m]} = \rho^+; w_t^{[m]} = p_0$
                $s_{n,t}^{[m]} \sim p(s_t \mid s_{t-1}^{[m]}, u_t); G_{\theta,n} = \nabla_{\theta_n} g(\theta_n, s_t)\big|_{s_t=s_{n,t}^{[m]};\theta_n=\mu_{n,t}^{[m]}}$
                $\mu_{n,t}^{[m]} = g^{-1}(z_t, s_{n,t}^{[m]}); \Sigma_{n,t}^{[m]} = (G_{\theta,n} R_t^{-1} G_{\theta,n}^T)^{-1}$
            else if $n_t \neq \hat{n}_t$ and $n_t \leq N_{t-1}^{[m]}$                      // handle unobserved features
                $\mu_{n_t,t}^{[m]} = \mu_{n_t,t-1}^{[m]}; \Sigma_{n_t,t}^{[m]} = \Sigma_{n_t,t-1}^{[m]}$
                if $\mu_{n_t,t}^{[m]} \notin \text{range}(s_{\hat{n}_1,t}^{[m]})$ then         // outside sensor range?
                    $\tau_{n_t,t}^{[m]} = \tau_{n_t,t-1}^{[m]}$
                else                                                                     // inside sensor range?
                    $\tau_{n_t,t}^{[m]} = \tau_{n_t,t-1}^{[m]} - \rho^-$; if $\tau_{n_t,t}^{[m]} < 0$ then remove $n_t$   // discontinue feature?
                endif
            endif
        endfor
        add $\left\langle s_t^{[m]}, N_t^{[m]}, \left\langle \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, \tau_1^{[m]} \right\rangle, \ldots, \left\langle \mu_{N_t^{[m]},t}^{[m]}, \Sigma_{N_t^{[m]},t}^{[m]}, \tau_{N_t^{[m]}}^{[m]} \right\rangle \right\rangle$ to $S_{\text{aux}}$
    endfor                                                                               // end loop over all particles
    $S_t = \emptyset$                                                                    // construct new particle set
    for $m' = 1$ to $M$ do                                                               // generate M particles
        draw random index $m$ with probability $\propto w_t^{[m]}$                       // resample
        add $\left\langle s_t^{[m]}, N_t^{[m]}, \left\langle \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, \tau_1^{[m]} \right\rangle, \ldots, \left\langle \mu_{N_t^{[m]},t}^{[m]}, \Sigma_{N_t^{[m]},t}^{[m]}, \tau_{N_t^{[m]}}^{[m]} \right\rangle \right\rangle$ to $S_t$
    end for
    return $S_t$
end algorithm
```

Fig. 1. FastSLAM 2.0 Algorithm pseudo-code[1]. Difference from fastSLAM1.0 can bee seen in the first for loop where the particles are sampled.

3.1.2.2 Data Association

For both the pose sampling and particle weighting steps of the SLAM process associations between landmarks stored in the map and ones that have been measured need to be made. Making an association simply means determining which observed landmark corresponds to which landmark that is stored in the map so that the difference can be calculated and used for a variety of things. To do this the software needs to consider things like the euclidean distance between the measured and stored landmarks, the covariance for both, and even the color of the landmarks. In the current implementation this is done with a Mahalanobis distance which accounts for the locations of both the stored and measured landmark and the covariance of the measured one. The stored/measured landmark pair with the smallest Mahalanobis distance is the pairing that is used for the association. The only exception is if the smallest distance is above a predetermined threshold it is assumed that this measured landmark is one that is not stored in the map and a new landmark is added.

The Mahalanobis distance can be skewed if variables are highly correlated, instead of this the Hellinger distance will be used instead. The Hellinger distance finds the difference between two probability distributions. In this all cones when looking for an association will be considered their own probability distribution. As the car uses its lidar and camera sensors to identify cones, the position of the cones is not exact, treating this as a probability distribution allows for the covariance to be used in the distance calculation. The lower the Hellinger distance the more probable the distributions are similar and are the same cone.

The improved method for determining the likelihood that an observed landmark correlates to a stored one can be used to implement a greedy data association scheme. This considers all of the observed landmark - stored landmark correspondence probabilities before making associations rather than simply taking the best one for each observation. The previous implementation used a rudimentary naive association scheme which while simple to implement does not produce the most accurate associations. Implementing a true greedy association scheme will produce better associations and as a result a more accurate map and vehicle position.

While the greedy association scheme will be an improvement it has its own pitfalls and does not allow it to consider an entire map where many measurements are associated with landmarks, potentially resulting in a sub-optimal set of associations. This can lead to making the wrong decision for an association which can permanently impact the algorithm's result in all future decisions. To remedy this the software will be modified to use Joint Compatibility Branch and Bound which uses Multi-Target Data Association where the association is built into a tree allowing for association between multiple measurements and landmarks. The association will be evaluated using the Mahalanobis and Bhattacharyya distances but the algorithm will look for the best set of associations for the entire map rather than greedily selecting single associations. The branch and bound tree is used to make this process more efficient. The tree is traversed to find the best solution, but previous hypothesis' of the best pairings are left on the tree allowing it to be expanded from these other nodes if new data is found that agrees with another node's pairings. Branch and Bound is typically used with integer linear programming, but can be used similarly here. This allows for the best solution to be kept and solutions worse than the best solution and with no room for improvement can be pruned off of the tree, prohibiting the algorithm from wasting time. [3]

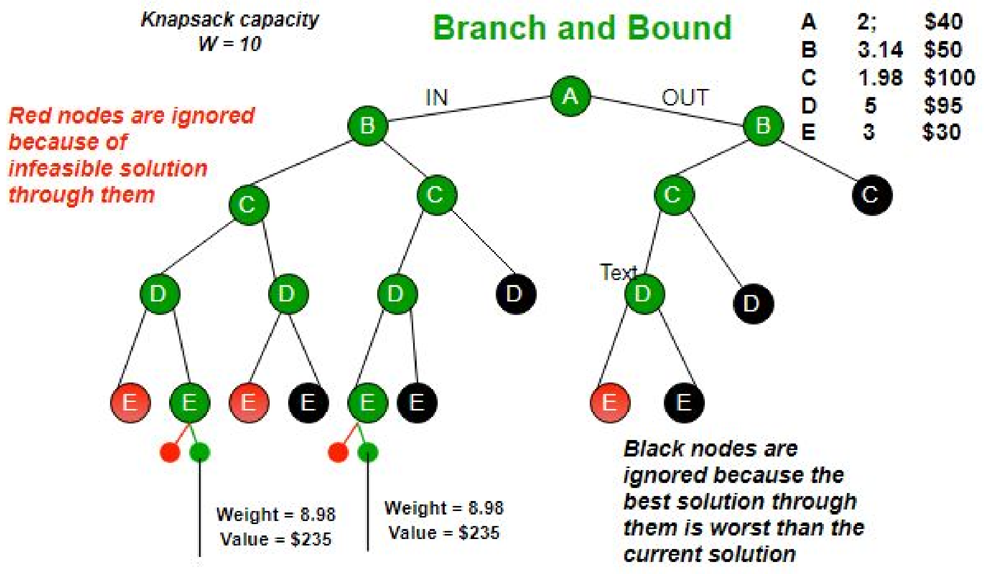

Knapsack capacity
W = 10
Branch and Bound
A 2; $40
B 3.14 $50
C 1.98 $100
D 5 $95
E 3 $30
IN
OUT
Red nodes are ignored because of infeasible solution through them
Text
Weight = 8.98
Value = $235
Weight = 8.98
Value = $235
Black nodes are ignored because the best solution through them is worst than the current solution

Fig. 2. Example of Branch and Bound Tree Implemented for Integer Programming [4].

3.1.2.3 Particle Weighting

The third major step of the SLAM process is particle weighting. This is when the various estimates of the robot's map and pose are evaluated to see which ones most closely represent reality. Particles that are determined to have better weights, indicating they are more realistic, will be used during the next iteration of the SLAM routine. To accomplish this task the software will evaluate the similarity between the landmark estimates stored in the map and the landmarks that were observed for a given SLAM iteration. Assuming that stored and observed landmarks have already been associated, the difference in angle between each pair and the vehicle will be summed for all landmarks with an associated observation. This results in a system where the smallest weight value is the best. While simple, this comparison will provide a fairly accurate weight with a very low computational cost. Due to the nature of the sensors on the vehicle, the directional sensing of observed landmarks is very accurate but the ranging is less so. Performing a more complicated comparison like a Mahalanobis distance between the stored and observed landmark would not give a much better comparison since the covariance related to the possible range of the landmark is so high, and would be considerably more expensive to compute.

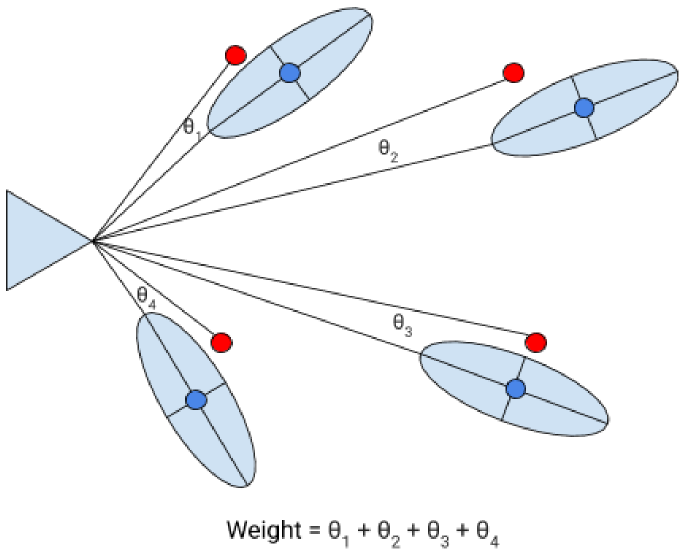


Fig. 3. Angle Difference Particle Weighting. Red circles are landmarks in map and blue circles are observed landmarks with uncertainty drawn with the light blue ellipse.

In addition to summing the angles between the observed and stored landmarks, some simple track validity checks can be performed to ensure that the generated map conforms to the rules. The Formula Student competition has very strictly defined rules related to the layout of the track including a minimum track width and a minimum distance between cones on each side of the track. This means that no two cones should be closer than a known minimum distance. The software will compare all the distances between pairs of cones, verifying that they are not too close together. If a pair of cones is too close together the weight of the particle will be multiplied by a large factor so that it will be evaluated as worse when compared to other particles.

3.1.2.4 Benchmarking

In order for the final product to contain the best versions of each part of the algorithm there needs to be a way to measure the accuracy and performance of these parts. Having statistics to compare for each portion will allow for an educated decision to be made. The accuracy of the algorithm is an important part of this decision, but the performance in terms of CPU usage, memory usage, and time will also be taken into account.

The algorithm is not useful unless it processes information under the constraints of moving at the speed and providing direction in time for the car to navigate the course. It also must run on the processor in the car alongside the other autonomous system algorithms, ideally using as little CPU and memory as possible. To track this metrics on the time for the algorithm to process and how much CPU and memory it is using at a given time will be gathered at different stages of the SLAM algorithm and outputted in a plot. With each variation of the algorithm the performance benchmarks can be run and the plots can be compared between the versions. With this and based on the accuracy of the algorithm the combination of performance and accuracy can be decided for the final version.

To evaluate the end product of the SLAM process a special visualization will be created using the simulator so that the creation and refinement of the map can be tracked. This will be accomplished by using what already exists in the simulator, displaying only the position of the car, the actual location of the cones, where SLAM things the cones are, current observations, and which landmarks these are associated with. All of these elements are already a part of the simulator except showing which observations are associated with which landmarks. This will be implemented by duplicating the drawing of the cones in the SLAM map, excluding those that were not associated, and changing the color to something easily identifiable. Selecting which things to show and not show in the simulator can be implemented with a simple configuration file, and the addition of the cones indicating data associations will be added by duplicating how the SLAM map cones are currently drawn but excluding cones that were not associated. This special visualization will be displayed in a separate window from the rest of the simulator so that it can be easily tracked when necessary.

3.1.3 Design Rationale

3.1.3.1 Particle Sampling

Adding measurements to the information available during the sampling of particles, creating more accurate particles, is especially important when control accuracy is low and measurement accuracy is high. Since each of the particles is more likely to accurately represent the real environment the algorithm can get away with using less particles altogether. This would reduce the computational cost of running SLAM altogether which frees up computational resources for other tasks. The biggest drawback of FastSLAM2.0 compared to version 1.0 is the complexity in mathematical systems used to implement the sampling considering the measurements of landmarks. This means that it would take more time to develop, and will likely be harder to troubleshoot once implemented. Ultimately FastSLAM2.0 creates a more accurate end result coming at the cost of the additional complexity, but in the case of the autonomous racing vehicle it is a worthwhile trade-off.

3.1.3.2 Data Association

Combining the Bhattacharyya distance for the cone color probability comparison with the Mahalanobis distance for the location point and probability comparison provides the most holistic measure for the similarity between a stored and observed landmark. This is more accurate than the Euclidean distance or just the Mahalanobis distance and is the implementation that will be used as it will provide a very accurate evaluation without adding too much computational complexity or memory usage.

A true greedy association scheme is a large improvement on a simple naive scheme and will provide more accurate associations and thus a more accurate result. This will be implemented first as a better baseline which will be compared later to other methods. Joint Compatibility Branch and Bound has been proven to be the best algorithm for data association within SLAM, but it is a much more complicated algorithm than even the greedy scheme. While it is more complicated it does have noticeable performance benefits that make it worth the trouble of implementing.

3.1.3.3 Particle Weighting

Evaluating which particles are accurate representations of the real world and which are not is an important step in the SLAM process. Due to the nature of the sensors on the vehicle where the directional measure of landmarks is very accurate but the distance measurement is less so presents a special circumstance. Using a complicated comparison like a Mahalanobis distance would provide a good evaluation but is highly tasking to calculate, and is hardly an improvement on simply comparing the difference in angle between the observed and stored landmark. Since the landmarks will have already been associated from an earlier part of the SLAM process the software can assume the the distance is reasonably close but the only reliable metric is the angle. This angle comparison is computationally cheap in comparison to a Mahalanobis distance and will provide particle weights that are almost exactly as accurate for a much lower cost.

3.1.3.4 Benchmarking

Last year the software was designed and implemented without metrics, instead they relied on the team's visual approximations on how well the algorithm performed. An engineer should base these decisions on numerical metrics allowing for an educated decision to be made on what to choose. The team does not currently have any tests to gather metrics that could be used to make decisions on the design process. Due to this, it is a worthwhile investment to create tests to gather metrics to compare the current software with the future improved version. The payoff of the metrics should allow for an improved final product and as an added bonus, easy points in the competitions this car is built for. These competitions put a considerable amount of weight into the engineering process of the car, and showing that this process is valued and carried out with tests and metrics will put the team ahead of the competition.

3.1.4 Design Tools

All of the software that will run on the vehicle including SLAM will be built using C++ and ran in ROS. ROS is a design constraint chosen by the Global Formula Racing team. ROS is capable of running simulations and allows for the execution of tests that can be used to evaluate and improve the entire system. Once the software has been verified via the simulator and tests in ROS, and the competition car has been built, the system will be physically tested on the car in a real environment.

3.2 Trajectory

3.2.1 Design Overview

After the Simultaneous Localization and Mapping(SLAM) software analyzes the environmental information and locates the vehicle and the cones marking the course, the self-driving racing car needs to use this information to determine the track boundaries, and plan the trajectory the vehicle needs to follow. From the trajectory the necessary controls needs to be generated to ensure that the vehicle can follow the planned trajectory.

The trajectory planning system obtains a map containing cone positions and the vehicle position from the SLAM system. With this information the track boundaries are estimated, then these boundary cones are used to create a series of triangles from on the interior of the track from which the mid-line of the track is determined. This mid-line will be the trajectory the car will follow. The first task of the team is to benchmark the 2019 boundary estimation algorithm to create a performance baseline and a framework with which future versions of the code can be evaluated with. This will then be followed by modifications to the algorithm for boundary estimation to create new versions capable of generating a better boundary estimate and trajectory. The updated algorithms will then be compared to the original to see the difference in reliability and performance.

3.2.2 Design Elements

3.2.2.1 Boundary Estimation Update

The new boundary estimation algorithm inserts a pair of virtual cones on both sides of the vehicle as a starting point after obtaining the map and vehicle position information from the SLAM software. Then these two virtual cones are used as a starting point to capture other cones at a certain angle and a certain distance. These nodes are then structured into two trees, and find the normal of the line between every two nodes. Next, the angle of the line segment of the two nodes connected to the other line intersecting the normal is checked. The closer that the angle is to 90 degrees, the larger that the weight of this cone pair is. When it is ensured that the distance between the two segments from the left and right boundary trees is not greater than a certain value, the two pairs of segments with the largest weight are selected as the boundary.

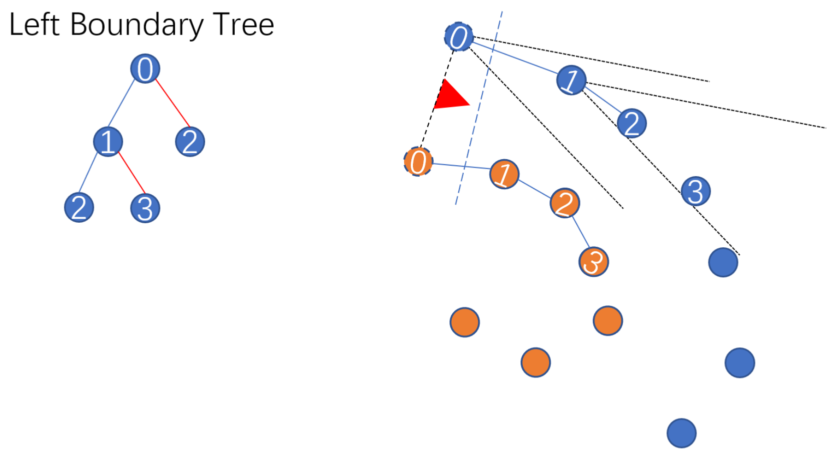


Fig. 4. Boundary Estimation Visualisation: The Boundary tree structure and strategy of cone searching. The red branch of Boundary tree have worse weight than blue one.

3.2.2.2 Triangulation

After the left and right track boundaries are generated, it is necessary to perform Delaunay triangulation on the two consecutive point sets. This triangulation should produce a series of triangles in the interior of the track which can then be used to generate the planned trajectory. Using the GGAL library, a cone with color information can be placed into an old inner triangulated container, and then the cones can be joined in two-track boundaries to form a series of triangles. These triangles need to have at least one vertex from the opposite track boundary. After the triangles are generated, the midpoint of the triangle sides that are not along the track boundary produces a series of points in the middle of the track. Linking these points produces a midline of the track which is the path that the vehicle will take. The midline of a track is not traditionally the fastest path around the track but due to uncertainty in the exact location of the cones marking the boundary the team has decided to prioritize not hitting cones over cutting corners.

3.2.2.3 Edit Distance Benchmarking

The team's current work involves benchmarking the boundary detection algorithm created by last year's team. After generating two sequences of cones, representing the boundaries, a data structure is created by the program that containing the sequences where each cone has a unique identifier. By performing an edit distance calculation on the order of these identifiers and the order of the real boundaries, the accuracy of the boundaries generated by the algorithm can be calculated. The test procedure uses the Damerau–Levenshtein distance, which allows four operations on the sequence, including deleting, adding, substituting, and swapping elements in the sequence, and outputting the minimum number of edits necessary to make the two sequences the same.

$$d_{a,b}(i,j) = \min \begin{cases} 0 & \text{if } i = j = 0 \\ d_{a,b}(i-1,j) + 1 & \text{if } i > 0 \\ d_{a,b}(i,j-1) + 1 & \text{if } j > 0 \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} & \text{if } i,j > 0 \\ d_{a,b}(i-2,j-2) + 1 & \text{if } i,j > 1 \text{ and } a[i] = b[j-1] \text{ and } a[i-1] = b[j] \end{cases}$$

Fig. 5. Damerau-Levenshtein Distance algorithm.

3.2.3 Design Rationale

3.2.3.1 Boundary Estimation Update

This boundary estimation method can simultaneously estimate the left and right boundary pairs in one calculation at a time. Fast boundary estimation is necessary when estimating the track boundaries in real-time. However, the problem with this algorithm is that it is not easy to determine the angle and distance of the cone tree at the beginning. The maximum and minimum width of the track also requires multiple experiments to optimize. Despite the potential issues this method will still be fast and accurate once some experimentation to determine threshold values has been completed.

3.2.3.2 Triangulation

In order to generate a midline between the existing two boundaries of the track, triangulation is an accurate and easy to calculate solution. However, a normal triangulation could use three cones from the same side of the track to generate the triangle, which fails to create a side with a midpoint along the midline of the track, and thus ruins the planned trajectory. By using the Delaunay triangulation, it is ensured that a triangle will not consist of only cones from one boundary, which avoids generating the incorrect triangle show as the red triangle in the figure below.

3.2.3.3 Edit Distance Benchmarking

The edit distance is the methods that has been selected to benchmark the boundary estimation because it can be easily implemented and provides an accurate measure of the similarity between actual and expected output. Further more, the algorithm can not only test the accuracy of the algorithm but also test the robustness of the algorithm through a large number of tests. Robustness is just as important as accuracy for the software since it is imperative that the autonomous vehicle never completely fails to ensure safety of those around the vehicle.

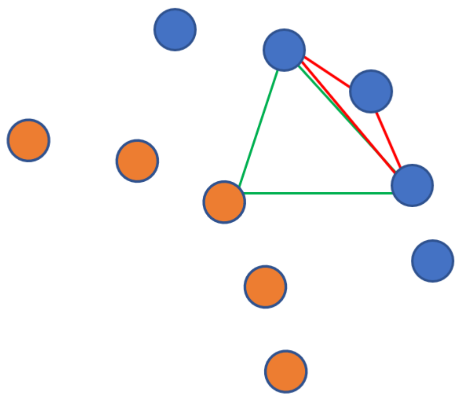


Fig. 6. Comparison of previous implementation of triangulation(Red) and the Delaunay triangulation(Green).

3.2.4 Design Tools

All of the software that will run on the vehicle including SLAM will be built using C++ and ran in ROS. ROS is a design constraint chosen by the Global Formula Racing team. ROS is capable of running simulations and allows for the execution of tests that can be used to evaluate and improve the entire system. Once the software has been verified via the simulator and tests in ROS, and the competition car has been built, the system will be physically tested on the car in a real environment. Some of the tests relating to the evaluation of the boundary estimations will be developed in Python.

References

[1] M. Montemerlo and S. Thrun, “Simultaneous localization and mapping with unknown data association using FastSLAM,” *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*.

[2] K. G. Derpanis, “The Bhattacharyya Measure,” *York University Department Of Electrical Engineering and Computer Science*, 20-Mar-2008. [Online]. Available: http://www.cse.yorku.ca/ kosta/CompVis_Notes/bhattacharyya.pdf.

[3] Giorgio Grisetti, Cyrill Stachniss, Kai Arras and Wolfram Burgard, Robotics 2 Data Association, University of Freiburg, pp. 33–35.

[4] Branch and Bound Algorithm, Geeks for Geeks, https://www.geeksforgeeks.org/branch-and-bound-algorithm/.