Solace: A Persistent, Performant Localization and Mapping Technique for Controlling Exploratory Autonomous Vehicles

Charles Zhao and Grey Golla

***Abstract*—Currently, methods exist for path planning within known areas, as well as for mapping out new areas. One algorithm for localization within known areas is Adaptive Monte Carlo Localization (AMCL), and one algorithm for mapping out new areas is Gmapping. However, maps created by Gmapping are not persistent, i.e., one cannot build upon a map previously generated by Gmapping, and must instead start creating a new map from scratch. In this paper, we propose a method for building persistent Gmapping maps for path planning within both known and unknown areas by combining Gmapping and AMCL. To implement our method, we built and programmed a 1/8th scale R/C car to drive autonomously to a given goal coordinate, determining the optimal path to the goal using its internal map, or mapping out unknown areas until a path was found to the goal. The car can thus find paths to goals in both known and unknown areas, while simultaneously constructing an updated map that can be used for finding future goals.**

I. Background

Several companies including Google, Mercedes Benz, and Audi have begun to develop their own autonomous or semi-autonomous vehicles [7]. Of these, Google is perhaps the most well-known for its autonomous vehicles. Autonomous vehicles have immense potential to save lives and time. As shown by Google's track record, it is much less likely for a devoted computer system to malfunction than for an easily distracted human. A study by the Eno Center for Transportation predicts that if just 10% of all cars in the U.S. were self-driving, about 211,000 accidents would be prevented, around 1,100 lives would be saved, and related costs would be reduced by about $38 billion per year [9]. Additionally, given computers precision, autonomous vehicles would not only be able to drive faster, but also closer together, thus reducing congestion and also making travel more environmentally friendly as people spend less time in the car.

One efficient simultaneous localization and mapping algorithm is Gmapping. Current implementations of Gmapping must start from scratch on every run because they cannot keep a persistent map, i.e. they cannot localize the car within a given map. Our goal is to solve this problem by building on top of Gmapping.

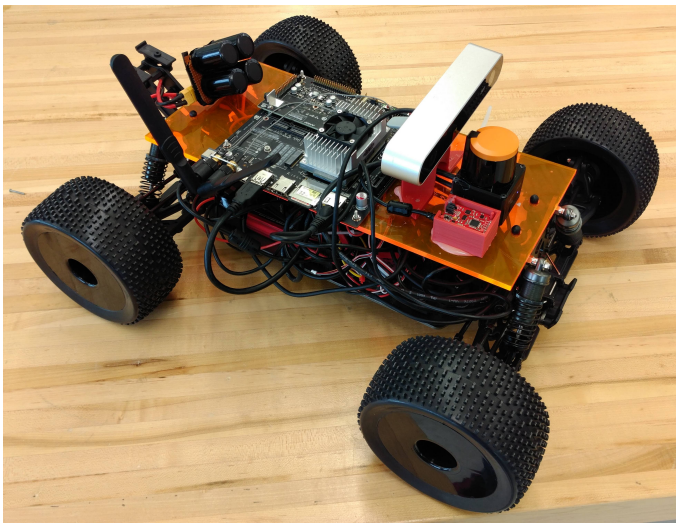


Fig. 1: A photo of Solace, with the various hardware components visible.

II. Methodology

We mounted several devices on a 1/8th scale car. These include an Nvidia Jetson TX1 "supercomputer," a Vedder Electronic Speed Controller (VESC), a Hokuyo UST10-LX 2D Laser Rangefinder, a ZED 3D Stereo Camera, and a Sparkfun Razor 9-DOF IMU. The Nvidia Jetson TX1, a low-power ARM computer combined with an Nvidia graphics card, runs our navigation and environment processing code on top of the Robot Operating System (ROS) Kinetic and Ubuntu 14.04. The VESC controls the motor while also providing odometry data. The ZED stereo camera provides a 3D point cloud of the space in front of the vehicle,

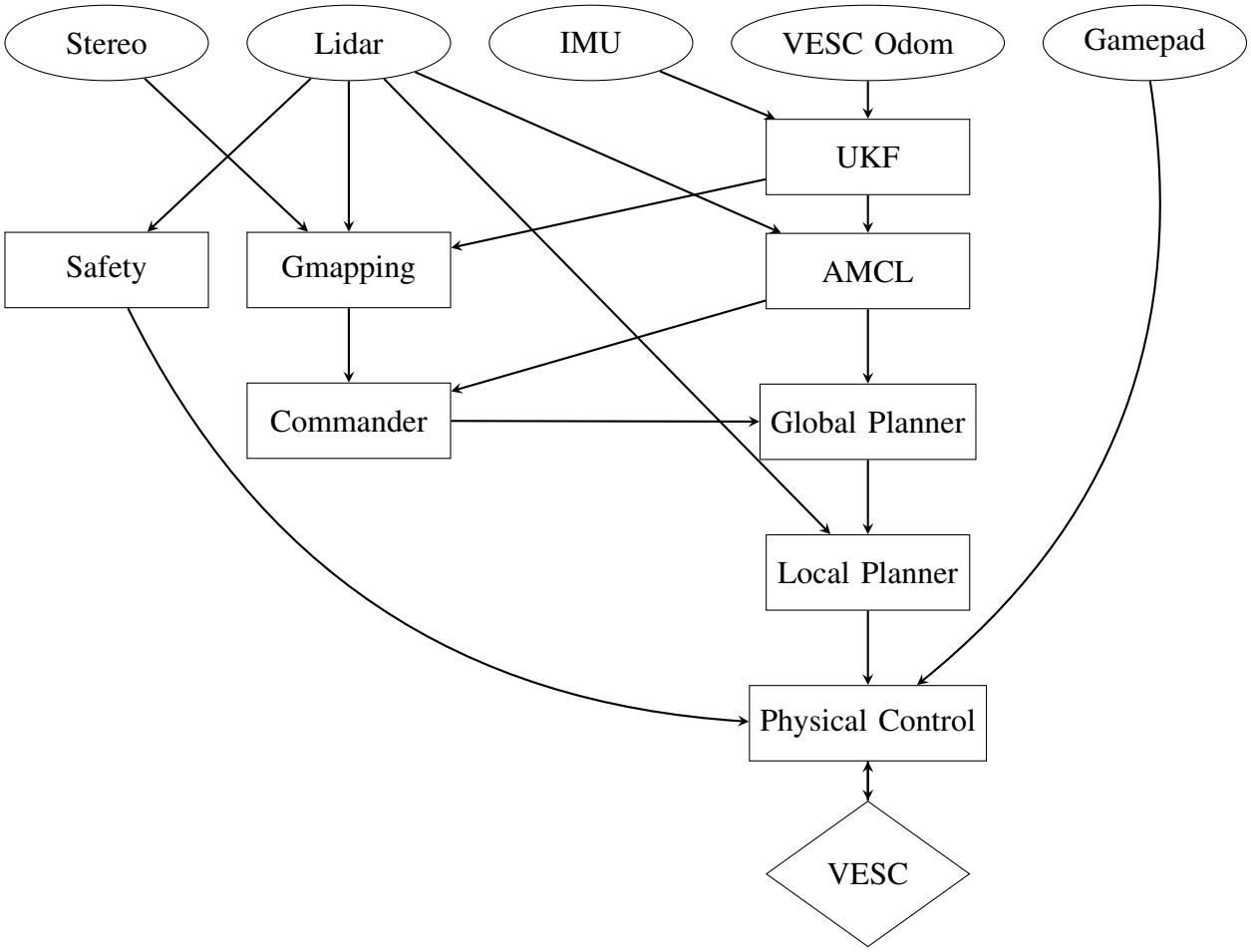


Fig. 2: Solace ROS software architecture. Ellipses represent inputs, rectangles represent software nodes, and diamonds represent outputs.

with a maximum range of 20m and a 110° field of view. The 2D Lidar provides a 270° scan of depth information, with a maximum range of 30m, and returning 1081 measurements per rotation (4 measurements per degree) and rotating at 40Hz. The IMU reports physical movement of the car in 3D space, including linear and angular acceleration, velocity, and position.

We programmed the car to first localize itself within an internal map. Given a goal coordinate, the car then tries to find the shortest path to the goal using its internal map. If its map is incomplete between the car and the goal, the car autonomously explores unknown areas, continuously updating its internal map until the goal coordinate is reached. Planning is naturally split into two separate tasks: local planning and global planning. The local planner avoids static and dynamic objects, while the global planner determines an overall path for the car to follow within its internal map.

In the following sections, we describe in detail localization, mapping, global path planning, local path planning, and our technique of combining known and unknown path planning. See Figure 2 for the full software architecture.

A. Localization

In order to navigate the environment, the car must first be able to determine where in the environment it is, and determine how it is moving in the environment. The simplest way that the movement of the car can be determined is odometry. Odometry uses internal measurements of the robots state (proprioceptive data) to estimate the change in its position over time. We use our brushless motor speed controller as well as a nine degree of freedom Inertial Measuremnet Unit for our odometry. Because both of these sensors are expected to have a random error

in their measurements, we combine them together using an Unscented Kalman Filter (UKF). A UKF is an extension of a basic Kalman Filter that handles nonlinear systems more effectively. Kalman filters estimate the actual value of a variable from multiple error-prone measurements of that variable. To do this, it models each measurement as a gaussian (or normal) distribution of possible values of the state. By finding the intersection of each distribution, it finds the most likely value of the data. It also keeps track of the estimated change in state variable over time by the same method, and transforms its old gaussian state estimate into another distribution that is also taken into account.

Unfortunately, even with these statistical tricks, odometry data is expected to drift over time, which means that its error (the difference between the robots expected position and orientation and its actual location in the world) is expected to increase without bound. To counteract this, we must use exterioceptive data to provide a correction. Exterioceptive data is taken in from the outside world, such as Lidar scan data.

We use Adaptive Monte Carlo Localization (AMCL) to localize the car within a given map, i.e., to determine the robots pose relative to the map. AMCL uses a particle filter that combines proprioceptive data from the unscented Kalman filter described above with exteroceptive data from the Lidar. The particle filter keeps track of a number of candidate poses, which are each possible poses of the car assigned a certain probability.

Whenever odometry data is received, all the candidate poses are updated accordingly. For example, if the odometry data states that the car has moved forward one meter, all the candidates poses will move forward one meter. Whenever external observations are received, which happens less frequently due to the greater computational resources required, the data from odometry is corroborated and adjusted accordingly by matching current laser scans to previous laser scans (a technique called laser scan matching, see Figure 4), and particles are either added to or removed from the filter. If the car is less confident about its pose, there will be a greater number of particles that are also more widespread, and if the car is more confident about its pose, there will be a smaller number of particles that are clumped tightly together. In our configuration, AMCL uses at most 2000 particles (when very lost), and at least 500 particles (when confident). See Figure 3.

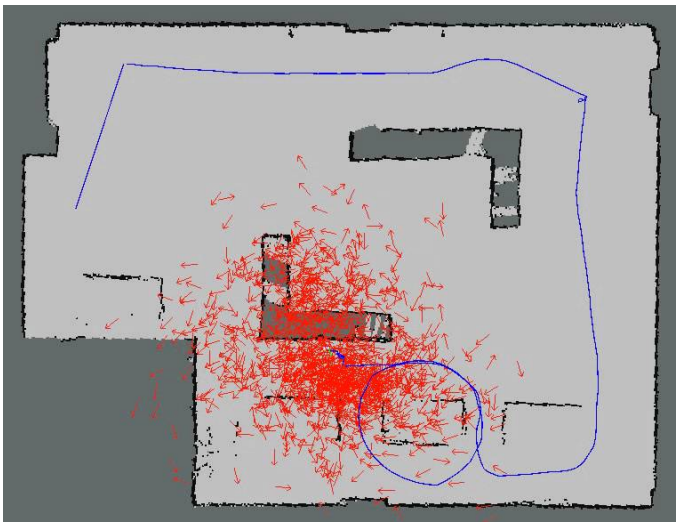


Fig. 3: An example of AMCL in action. Each of the red arrows represents a candidate pose. This particular robot is quite lost, as there are many candidate poses and they are quite widespread. [12]

B. Mapping

For mapping, we use a simultaneous localization and mapping (SLAM) algorithm called Gmapping [2]. This algorithm uses a Rao-Blackwellized particle filter, with each possible position and trajectory of the car represented as a particle [6]. Gmapping solves the problem of reducing the number of particles to more efficiently and accurately predict the car's actual position and trajectory. This algorithm improves on other kinds of algorithms that only use odometry data by also considering the car's current observations, using Lidar data to perform laser scan matching (see Figure 4).

Rao-Blackwellization takes advantage of the fact that the probability of the car having a certain position and trajectory is equal to the probability of the car having that trajectory times the probability of the car having that position given that trajectory. Formally, if x_k, m_k, z_k, and u_k are the car's trajectory, map, observations, and odometry measurements, respectively, at time k, then

$$p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) = \\ p(x_{1:t}|z_{1:t}, u_{1:t-1}) \cdot p(m|x_{1:t}, z_{1:t}).$$

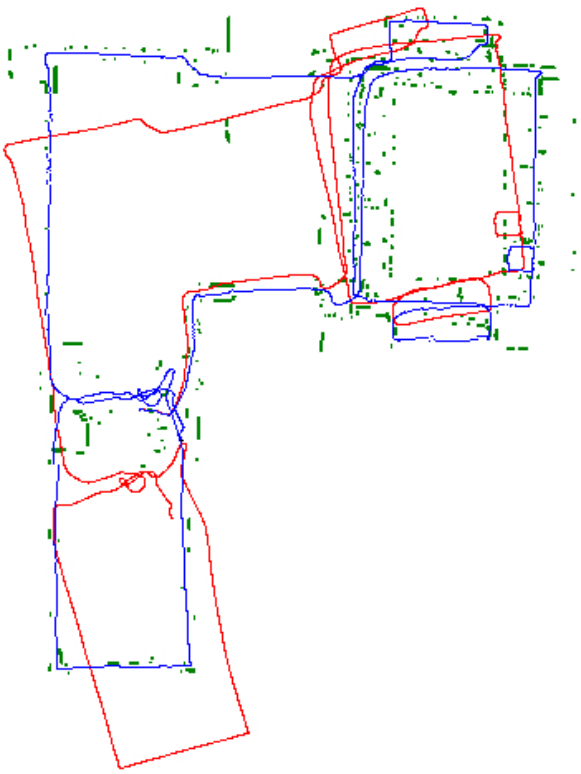


Fig. 4: An example of laser scan matching. [4]

Since the car's position is strongly dependent on the car's trajectory, Rao-Blackwellization results in greater efficiency [6]. Once the car has localized itself using this algorithm, i.e. determined its position and trajectory, Gmapping will constantly provide updated maps to the navigation stack.

Gmapping returns occupancy grids. In our configuration, these occupancy grids have a resolution of 5cm, and cells that have lower than a 19.6% chance of being occupied are marked as free, cells that have greater than a 65% chance of being occupied are marked as occupied, and all other cells are marked as unknown. See Figure 5.

C. Global Path Planner

The global path planner's goal is to find a "rough" path from the car's initial point to it's goal position. The global planner finds a path through the current world map that the car has already generated and localized itself within. To find a path, we use the Rapidly Exploring Random Trees (RRT*) algorithm. This algorithm randomly generates a tree, where each branch is a valid movement of the car. The tree growth is biased toward the goal, and branches are

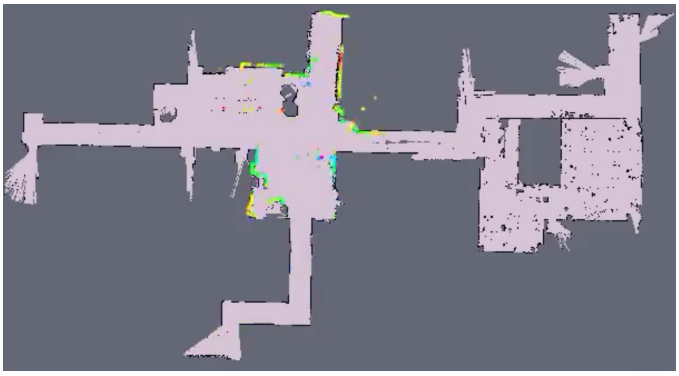


Fig. 5: A map our car generated with Gmapping. Black cells are occupied, dark areas are unknown, light areas are free, and colored dots are the live view of the Lidar at the point the screenshot was taken.

removed from the tree if they violate any constraints we put on the movement of the car (for example, going through a wall or turning too quickly). The final path is created as soon as a branch of the tree touches the goal position. See Figure 6 for an example of RRT* navigation through a simple environment.

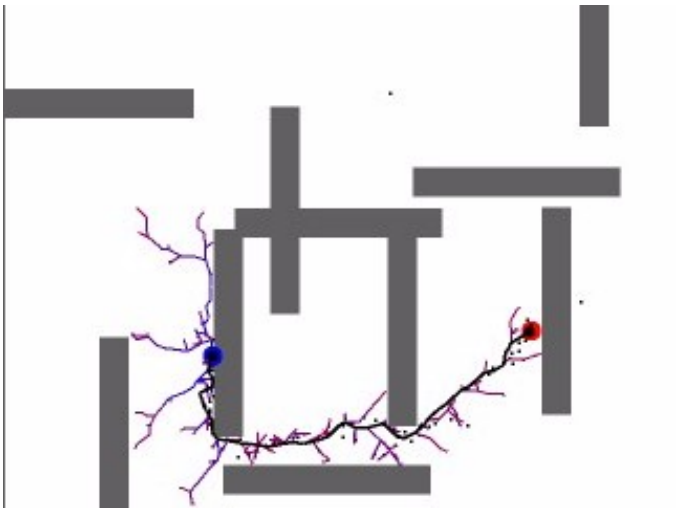


Fig. 6: An example of RRT* finding a path from the blue dot to the red dot. [10]

D. Local Path Planner

The local path planner ensures that the car does not hit obstacles not in the car's known map, such as obstacles smaller than the global map's resolution (5cm), and dynamic obstacles such as people

walking around. The local path planner generates a 6m x 6m costmap centered around the car, with a resolution of 1cm. For determining a local path, we use a technique called potential fields, where we treat the car as a positively charged particle and then treat each of the points returned by the Lidar as another positively charged particle. This causes the car to stay away from obstacles and to take the path of least resistance, reacting especially strongly to obstacles that are nearby. See Figure 7.

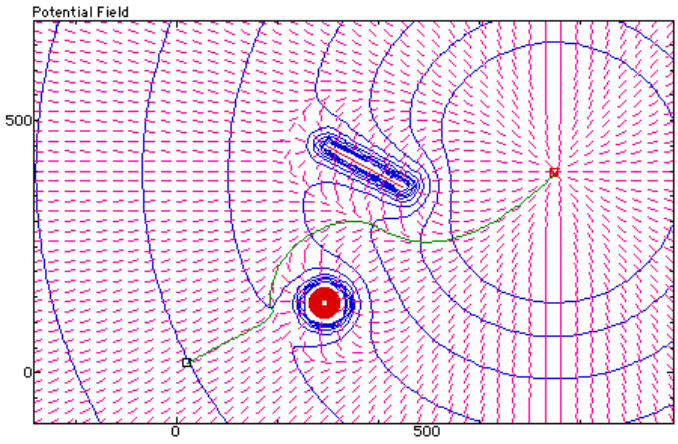


Fig. 7: A visualization of potential fields. The arrows indicate the direction of the gradient, and their point of convergence at the upper right is the goal coordinate. The red circle and line are obstacles. The green line represents the robot's planned path. The blue contour lines are lines of equal potential. [1]

In our implementation of potential fields, we do not simulate physical potential fields exactly, and only use the analogy of electrostatic potentials to make the explanation easier. For example, Coulomb's Law states that the electrostatic force between two point charges falls off as $1/r^2$, where r is the distance between the particles. However, we have the force fall of as $1/r^{1.8}$. This has the effect of making far-away points have a more noticeable effect on the car's path, rather than having only very close obstacles have a significant effect.

To cause the car to tend to move in a certain direction, we create a constant large positively charged particle accordingly. For example, to cause the car to tend to move forward, we create a large positively charged particle behind the car. The local planner uses the A* shortest path algorithm to calculate a local path within the 6m x 6m costmap, which decides the placement of this large charged particle.

Since the car may get stuck if it reaches an equilibrium in the potential fields, especially since the car is not holonomic, we also have a recovery behavior that removes the constant charged particle if the car has not moved for 3 seconds.

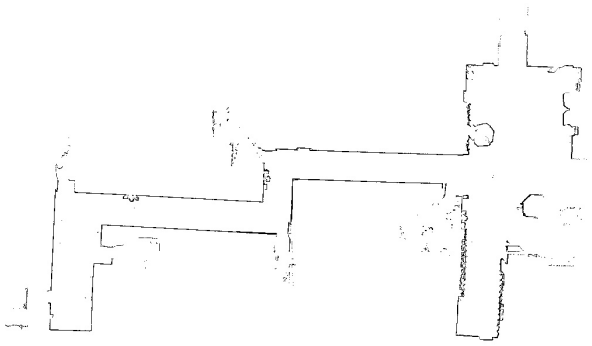
E. Combining Known and Unknown Area Path Planning

AMCL is able to localize the car within known maps, and thus allows the global path planner to plan a path through known areas. Gmapping is able to create new maps and localize the car within these new maps, thus allowing the global path planner to plan a path through new areas. To combine these two algorithms, we implement a method of stitching new maps together. This allows us to always use AMCL for localization and to use Gmapping to update the known map.

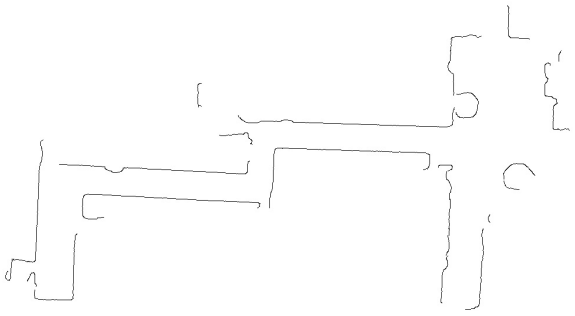
The car must always start in a known area, and Gmapping is always running, even when the car is not lost. This is necessary because Gmapping is only able to construct completely new maps and not build upon previously made maps, so this causes the map generated by Gmapping to at least partially overlap with the known map. For stitching, we use an automatic stitching program called OpenPano [11]. Due to the overlap between the map created by Gmapping and the known map, OpenPano can find matching key points in both maps and therefore stitch them together.

Due to the imperfect nature of our hardware, and any hardware for that matter, the maps generated by Gmapping are quite noisy. For example, there are tiny obstacles where there should not be anything, which could be due to someone walking by at the time of the mapping, and there are many small gaps in the obstacles, such as in walls. This noise made OpenPano unable to stitch maps together. To filter out most of this noise, we use OpenCV to first apply a Gaussian blur with a kernel size of 21 x 21, and then perform Canny edge detection with a minimum threshold of 10, a maximum threshold of 115, and an aperture size of 3 [8]. The large Gaussian blur along with Canny's high maximum threshold removes small "phantom" obstacles, while Canny's very low minimum threshold patches small gaps in obstacles. See Figure 8.

Since stitching requires significant computational resources, we only want to perform stitching when



(a) An unaltered map generated by Gmapping.



(b) The same map after applying a Gaussian blur and Canny edge detection.

Fig. 8: The before and after images from our smoothing method described in Section II-E.

the car is mapping out new areas. To determine when the car is lost, i.e. in a new area, we use the number of particles in AMCL's particle filter. We decide that the car is lost if AMCL has kept track of more than 1500 particles for more than 3 seconds. (See Section II-A for details about AMCL's particle filter.)

Our last optimization for mapping involves controlling the car's speed. Gmapping publishes a topic called entropy, which is an "estimate of the entropy of the distribution over the robot's pose (a higher value indicates greater uncertainty)" [5]. Therefore, we have the car's speed be proportional to $1/entropy$ so that the car drives more slowly when Gmapping is less certain of the car's pose.

III. Results

We found that odometry was extremely unreliable, although the unscented Kalman filter improved odometry dramatically. The Lidar, as expected, was much more reliable and precise. We found the ZED stereo camera to be less useful than the Lidar, as we only considered 2D space, so we did not need all the data that the ZED reported. Additionally, the ZED requires more computational power than the Jetson could provide in order to update frequently enough to be useful. Finally, the ZED driver had a serious issue that made it mostly incompatible with the rest of our software stack.

By combining odometry with laser scan matching, AMCL and Gmapping both worked quite well in localizing the car. Gmapping's probabilistic approach to occupancy and position made it much more effective at localizing the car, but AMCL was still able to do reasonably well.

We were able to successfully implement a local planner using potential fields and a global planner using the ROS navigation stack. We were also able to map out new areas autonomously by combining Gmapping and potential fields. However, we were unable to perform map stitching successfully, as OpenPano was not able to find matching key points. We believe this is due to unfiltered noise produced by Gmapping, as well as distortions in these maps that cause maps to not align perfectly. For example, in Figure 5, the right angles are not exactly $90°$.

IV. Conclusions

Although we were not able to get map stitching to work, we believe it is definitely possible with improved smoothing and stitching algorithms. For example, a better method of smoothing the maps could be to use a probabilistic Hough line transform to only extract lines longer than a certain length. As for stitching, a more complex but informed approach could be to incorporate information about the car's last position and orientation when determining how to stitch maps together.

If our method is improved upon and implemented on a larger scale, we believe it could have very useful practical applications in disaster relief in the future, as well as any other task that requires mapping out areas.

V. Acknowledgements

We would like to thank our research lab directors, Dr. Shane Torbert and Dr. Peter Gabor, for guiding us through our research. We would also like to thank MIT Beaver Works for introducing us to ROS and

autonomous vehicles, as well as Winter Guerra for assisting us with the potential fields code.

REFERENCES

[1] Calerga. (n.d.). Potential Field [Digital image]. Retrieved June 10, 2017, from https://www.calerga.com/products/Sysquake/robotnav.html

[2] Chrysanthakopoulos, G., & Shani, G. (2010). Augmenting appearance-based localization and navigation using belief update. Retrieved October 14, 2016, from ACM Digital Library.

[3] Davies, A. (2016, February 29). Google's Self-Driving Car Caused Its First Crash. Retrieved April 24, 2016, from http://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/

[4] [Feature based map and corrected robot pose using laser scan data as input.]. (n.d.). Retrieved February 1, 2017, from https://rvlab.icg.tugraz.at/

[5] Gerkey, B. (n.d.). Gmapping (Version Kinetic) [Program documentation]. Retrieved June 11, 2017, from http://wiki.ros.org/gmapping?distro=kinetic

[6] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. N.p.: n.p., n.d. PDF.

[7] Kennedy, B. (2014, July 25). Top 5 Companies For Autonomous Vehicle Technology. Retrieved April 24, 2016, from https://finance.yahoo.com/news/top-5-companies-autonomous-vehicle-143652992.html

[8] OpenCV (Version 2.4) [Computer software]. (n.d.). Retrieved June 11, 2017, from http://opencv.org/

[9] Preparing a Nation for Autonomous Vehicles (Rep.). (n.d.). Retrieved April 24, 2016, from Eno Center for Transportation website: https://www.enotrans.org/wp-content/uploads/2015/09/AV-paper.pdf

[10] RRT [Digital image]. (n.d.). Retrieved June 12, 2017, from https://lightlordhippo.files.wordpress.com/2012/11/rrt.jpg

[11] Wu, Y. (n.d.). OpenPano [Computer software]. Retrieved May 23, 2017, from https://github.com/ppwwyyxx/OpenPano

[12] YTU BM Robotics. (2014, August 5). [ROS AMCL localization]. Retrieved June 10, 2017, from https://www.youtube.com/watch?v=9J2oc_hvON0