AUTONOMOUS MINATURE CAR FOR ROOM EXPLORATION AND OBJECT SEARCH

Tyler Hattori*, Xuanni Huo*, Sabrina Maldonado*,
Parker Napier*, Wyatt Swist*, Sean Anderson†
Department of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA
thattori@ucsb.edu, xuanni@ucsb.edu, smmaldonado@ucsb.edu
pnapier@ucsb.edu, wyatt_swist@ucsb.edu, seananderson@ucsb.edu

Faculty Advisor:
João Hespanha

ABSTRACT

We demonstrate the use of a miniature car to autonomously search an unknown environment and successfully find a target object. With inspiration from the F1TENTH project, we use a Raspberry Pi to run localization, mapping, and path planning software. Our software creates a map of the car's surroundings and avoids obstacles through simultaneous localization and mapping software generated using LiDAR scans and odometry data. Using the resulting map, our search algorithm calculates points for the car to explore based on distance and probability of the target being near those points. The car's path updates until the target has been detected using a camera. With this project, we aim to enhance the utility of autonomous vehicles for search and identification.

INTRODUCTION

As a senior capstone project at University of California Santa Barbara, the Department of Electrical and Computer Engineering conducts year-long group projects in partnership with external institutions: this project in particular was funded by the International Foundation for Telemetering. The team built a miniature car based on open-source designs and developed software that enables autonomous exploration in an unknown building and location of a target object.

The team leveraged hardware designs and basic software from the open-source community F1TENTH. F1TENTH is an international organization with the goal of fostering a community geared towards improving autonomous system technology. F1TENTH focuses on racing with miniature cars at

*The authors contributed equally to this work.
† Graduate student mentor in the Department of Electrical and Computer Engineering.

1/10 the scale of full-sized race cars, and holds international conventions where teams can race against each other [1]. F1TENTH also seeks to cultivate interest in communications, robotics, and controls through educational materials. F1TENTH offers open-source materials to anyone interested in order to help teams build their cars and therefore have more time to devote to creating algorithms. This means that many cars are uniform in their hardware and teams can spend more time making significant improvements in their software and algorithms.

Our specific project car, which we refer to as Velma, builds off the basic F1TENTH platform. The team aimed to develop a miniature car capable of autonomously navigating through an unknown environment while searching for a particular object. This problem is challenging due to the car not having any prior knowledge about the layout of the area it is searching, in our case a building with multiple rooms. In order to find a desired object, the car needs to be able to efficiently move through rooms while not crashing, map the area for keeping track of searched space, and identify the target object from variable distances and poses. We accomplished this goal by using off-the-shelf hardware components from the F1TENTH community, and developed software that would enable the required search capabilities. We recount the hardware architecture and focus the majority of this paper on the novel software developments. The final version of Velma is capable of autonomously searching an unknown interior environment with multiple rooms and successfully identify and approach a target object. In addition to the main searching algorithm, Velma is capable of several other driving modes that further expand the capabilities of the car in the realm of autonomous driving.

HARDWARE

Our system is characterized by three main subsystems interacting with each other to achieve the desired task. Each subsystem is a set of hardware components that communicate sensor data or execute commands through a software interface with the central computing unit.

We designed our system around the Raspberry Pi 4–a microprocessor with the capability to manage our overall network. The first of our three subsystems is a camera that can detect objects and colors, which is represented by the Pixy. The second subsystem is a sensor that can detect distances and recall a complete mapping of the area it has observed, represented by the SLAMTEC RPLIDAR Laser Range Scanner (LiDAR). The third subsystem is a remote control car, represented by the Traxxas Slash 4WD Electric Short Course Truck. These subsystems interact with each other via a software framework set up by Robot Operating System (ROS), which simplifies the software design process and assists with wireless control of the system. Additionally, we are able to remotely access the Raspberry Pi 4 through the Virtual Network Connection (VNC) across a wireless network. This allows us to remotely interact with our entire system from a laptop while guaranteeing the safety of the car at all times. The vehicle is powered by two batteries: one for the drive motor and Raspberry Pi and one for the LiDAR specifically. Figure 1 depicts the final build of the car with all components attached.

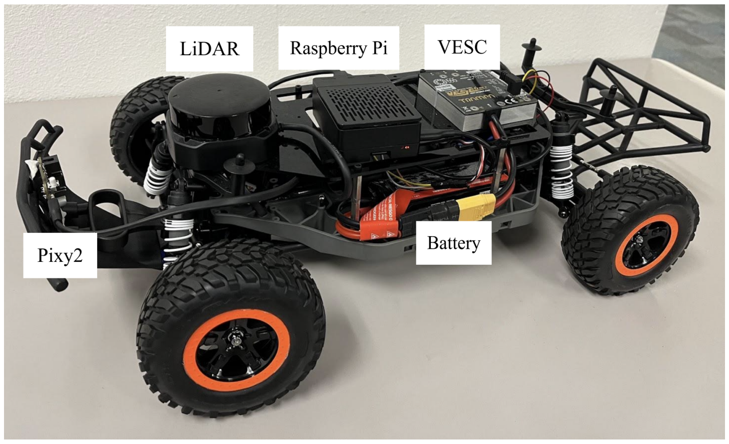


Figure 1: The sensing and compute components are mounted on top of the car, with the LiDAR at the front, and the actuators internally mounted on the chassis.

A. Sensing

The LiDAR scans the environment in a 360 degree rotation and uses a simple time-of-flight calculation to determine the distance to an object in every direction. The LiDAR is the crucial component regulating collision avoidance and overall navigation. The F1TENTH project uses LiDAR as the main sensor for some of the movement algorithms and is also the primary sensor for our room mapping software. As it constructs a map of the environment in real time, the LiDAR detects obstacles and walls in the environment where the rest of our software moves the car to avoid them. The LiDAR module for our project is a RPLiDAR S2, which our team chose for its high refresh rate and improved distance calculations. The LiDAR is powered separately from the other components using a portable battery bank.

The Pixy is an imaging sensor that uses filtering algorithms to detect colors in its field of view. Once an object is detected, the Pixy will continue tracking the object and report its position in terms of planar pixel coordinates within the camera frame. The image processing onboard the camera module requires minimal CPU space, thus saving valuable space for more computationally intensive components, such as LiDAR mapping. The Pixy acts as the main source of identification of the object, allowing the Pi to send commands to the other peripherals to form a path toward the object.

B. Actuation

The speed controller controls the drive and steering motors. It is connected to the Pi through a USB, from which it receives commands to control the motors. Our project uses the VESC 6 Mk. 5 as the speed controller because of its thorough documentation on the F1TENTH website and compatibility with Pi computers. The motors used are the servo steering motor and brushless drive motor, which are common on racing remote controlled cars. The speed controller also reads the (rotations per minute) RPM of the drive motor and angle of the steering rack, which allows our car to accurately report its linear and angular velocity.

C. Computation and Networking

The Raspberry Pi, which is mounted in the center of the car, serves as the central processing unit for our system and has three major I/O elements connected via USB. The Pixy object detection camera and RPLIDAR S2 sensor act as input data streams to the Pi, while the VESC is a bilateral stream of data including drive commands and read back of driving status. The Raspberry Pi utilizes these resources to process information and execute an algorithm to allow the robot to meet our goals. The Pi also maintains a wireless connection with our laptops via a virtual network computing (VNC) connection. The Raspberry Pi also runs the main coding structure for the robot through ROS, which will be discussed in detail in the next section.

SOFTWARE

The framework for Velma was created in large part by our team with some help from F1TENTH and other online resources [2] [3]. Our software framework also includes capabilities outside of our original project goal. ROS is the overall coding framework used to run the various code and algorithms that operate the car [4]. ROS operates by running code in snippets known as nodes that publish and listen to topics, through which different nodes can communicate. ROS also allows for real-time visualization of data in the system and the integration of various devices together under one coding structure.

D. Architecture

Figure 2 depicts a simplified version of our overall ROS structure. The centerpiece of our network is a multiplexer that publishes to the main drive command topic based on what driving option the mux controller decides is relevant. The multiplexer, or mux, controller makes this decision mainly based on input from the manual controllers, which is either a keyboard command from a connected keyboard or from a website that sends commands over a local network. The multiplexer also subscribes to the sensor data as it implements the emergency braking for safety. The multiplexer controller does this by obtaining the car's current location, orientation, velocity, and the current LiDAR map, and calculates if the car's projected location will coincide with the location of an obstacle. If this returns true, the driving mux stops publishing to the VESC until a drive command is registered in the opposite direction. As the car moves, the sensor data is updated. The self-driving algorithms use this data to determine the content of each of their own drive commands, which are constantly being fed to the driving mux in case they are specifically called for by the user. For the manual controllers, they must each publish to both the mux controller and the driving

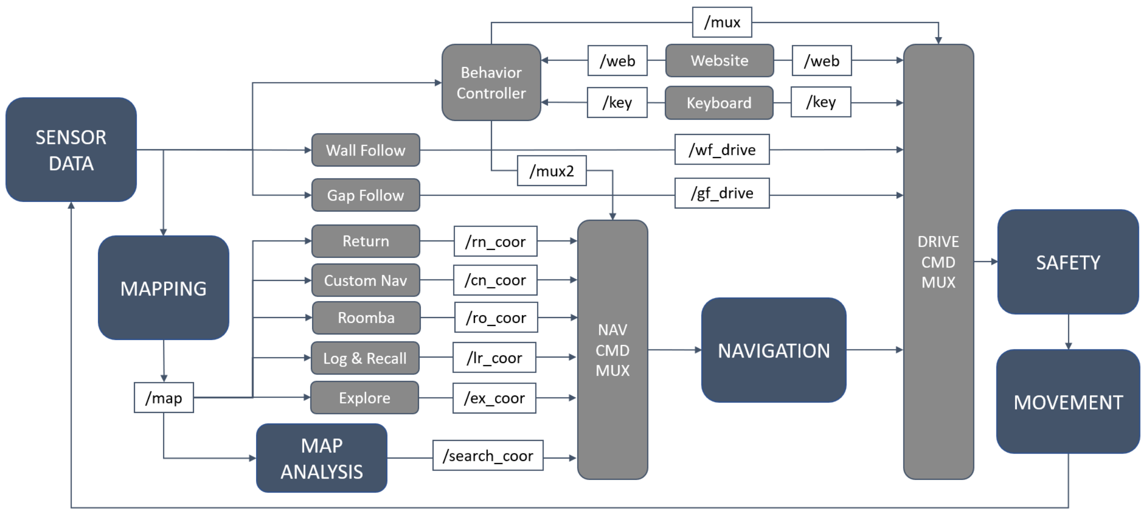


Figure 2: The graph depicts flow of information through the ROS network interacting with our software stacks (navy blue blocks).

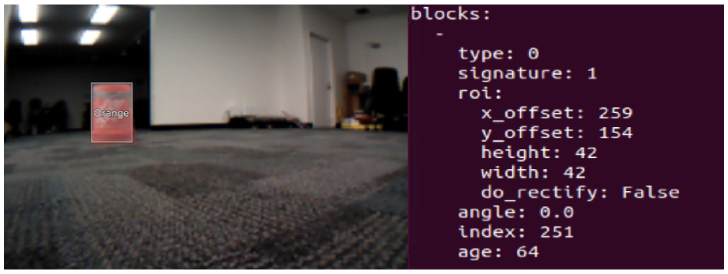
mux itself. This is set up so that when the mux controller receives a command to activate manual control, it will create and publish drive messages based on input from that same manual controller indefinitely. Once a drive message is sent to the VESC, the VESC will operate the drive motor and steering servo. Then, as the car drives forward, the sensors will update again in a way that a continuous loop is created in our system.

E. Target detection

The Pixy is our primary means of detecting the target object. It publishes the area, in pixels, of the rectangle where the target is detected, the pixel coordinates of the center of this rectangle, and the time since detected. The source code for extracting this data from a Pixy camera comes from the Pixy's Wiki website, but this code was adapted for our ROS network. The position of the target object in relation to the car when seen by the camera can be determined by the area and location of the box created by the Pixy software. Figure 3 depicts the Pixy view of the target object and the box the software draws on the detected object. Through testing, we established a relation between the size and position of the drawn box and the location of the target relative to the car, and we use this relation to navigate the car to the target once it has been seen by the camera.

F. Mapping and navigation

The car keeps track of its position in the environment through simultaneous localization and mapping (SLAM) [5]. This program takes the LiDAR scans of the environment and uses them to generate a map of the car's surroundings. As the car moves around the room, SLAM uses the



```
blocks:
  -
    type: 0
    signature: 1
    roi:
      x_offset: 259
      y_offset: 154
      height: 42
      width: 42
      do_rectify: False
    angle: 0.0
    index: 251
    age: 64
```

Figure 3: On the left, we show a screen capture of the Pixy camera's view and identification of the target object. On the right, the Pixy translates the identified object to the corresponding data.

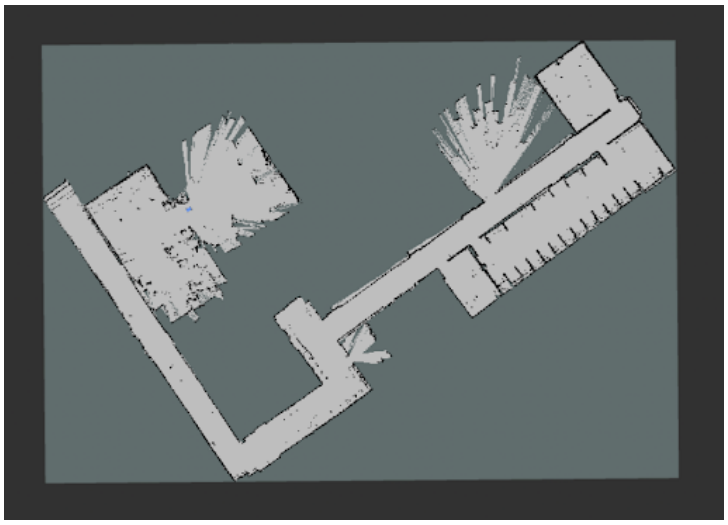


Figure 4: From the SLAM algorithm, we can generate a high fidelity map of an indoor environment that is continuously updated as the robot travels.

linear and angular velocity readings from the VESC to determine where on the map the car has moved. SLAM utilizes scans of the environment from new positions to update its map to include new areas that have been seen and increase the map accuracy of spaces already seen. SLAM also utilizes its map and new scans to update the position of the car on the map and correct small positional errors. This process also makes use of transforms to relate the various coordinate frames of the different sensors with each other to create a cohesive map. Figure 4 shows an example of a SLAM map generated by our vehicle.

Velma is capable of several driving modes which can be divided into two major categories: simple driving and map-based navigation. There are three simple driving algorithms, the first of which is *manual control* where the user uses the keyboard or website to send drive commands. The second simple algorithm is *patrol* which uses the LiDAR to maintain a consistent distance with a nearby wall and drive forward along it. The final simple algorithm is *advance*, which uses the LiDAR to drive towards the furthest point it detects without hitting any obstacles.

The map-based navigation algorithms are the more complex of the self-driving algorithms and utilize what is known as the navigation stack. The navigation stack accepts a navigation goal in the form of coordinates on the SLAM map as inputs and outputs drive commands to navigate to the specific point while avoiding obstacles. Velma's algorithm also includes various fixes from publicly available navigation packages that allow for a traditional steering setup and allow it to make multiple point turns to turn around. The coordinates that are passed to the navigation stack are determined by the navigation multiplexer. In total, there are six algorithms that use the navigation stack, the first of which is *return* which navigates the car back to its starting position. The second algorithm is *roomba*, which picks a random mapped point to travel to. The third algorithm is *navigate* where the user selects a point manually in the software for the car to path to. *Recall* is the fourth algorithm where the car records various points along a path traveled using manual control and then traces its path backwards along those points. The *explore* algorithm is the fifth map-based algorithm where the car travels to areas not yet mapped continuously to create a complete map of the environment. The final map-based algorithm is *search* which is the main goal of the project and is discussed in further detail in the next section.

SEARCH ALGORITHM

The searching algorithm continuously assigns values to points on the SLAM map-based on how relevant a point is for search purposes, with higher values being more valuable for the purpose of searching. This set of values is known as the aggregate value map and takes a weighted average of a set of value maps, each designed to take into account a particular criteria, which will be detailed next. The car then selects the highest value on the aggregate value map to travel to and if the target is not identified, Velma updates the maps based on the new information and selects a new point to navigate to. This process is repeated until the target is found.

To create the final value map, known as the aggregate value map and depicted in Figure 5, Velma first generates five separate value maps that each represent a different important criteria for efficient searching. The first value map takes into account whether an area has been searched or not by assigning higher values to clusters of points in the currently known area not yet observed by the car. The second value map operates similarly to the *explore* algorithm such that known points that are on the frontiers of unmapped open space are assigned a nonzero value. Additionally, the third value map accounts for the last time a point was observed: as time progresses, each point's value increases up to a saturation value based on how long it has been since the point was observed. This encourages the car to re-search points if the target has not been identified after searching a space. In order to encourage exploration of points that are easy to reach, the fourth value map assigns higher weights to points that are within a certain distance and radius of the front of the car than points beyond this range. The final individual value map considers obstacles: obstacles are assigned zero-value. All of these maps are continuously generated in real-time to be used in the aggregate value map, thus determining the next navigation goal. In practice, both the map weights and each map's parameters can be fine-tuned for a particular environment to enhance the searching efficiency.

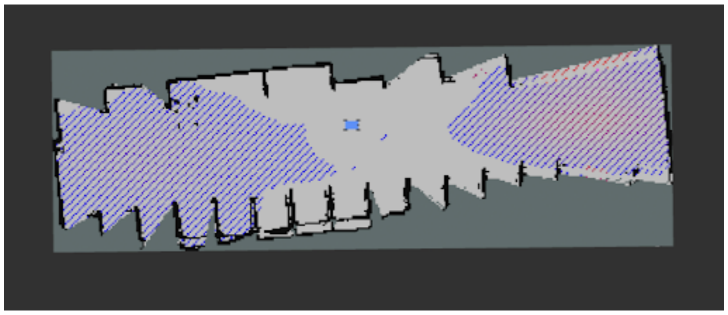


Figure 5: The aggregate value map overlaid on the current SLAM map illustrates the overall criteria for selecting a new navigation goal. The red points indicate a higher value of relevance for the search algorithm than the blue points

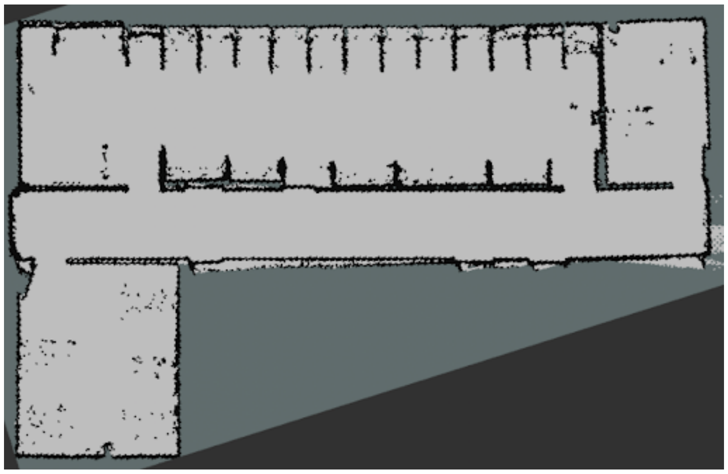


Figure 6: The testing environment for the algorithm included a central room with multiple partitions for desks, a hallway, and two side rooms.

When the Pixy camera detects the target object, which in our case is a large orange bucket, the searching algorithm then selects the navigation goal to be the position of the target. The position of the target is calculated from the Pixy camera data. The car then navigates to the bucket and, once within a few centimeters of the target, the car stops and the searching algorithm ends.

EXPERIMENTAL RESULTS

To test the search algorithm, we use an orange bucket as the target object, which was chosen for its large size and distinctive color with respect to the environment. The testing environment consisted of various rooms and hallways within an old gym building on the University of California, Santa Barbara campus. Figure 6 shows a complete SLAM map of our testing environment.

In this searching environment, Velma was able to successfully find the target in seven out of every ten trials, with successful runs taking less than three minutes with the total distance traveled on the

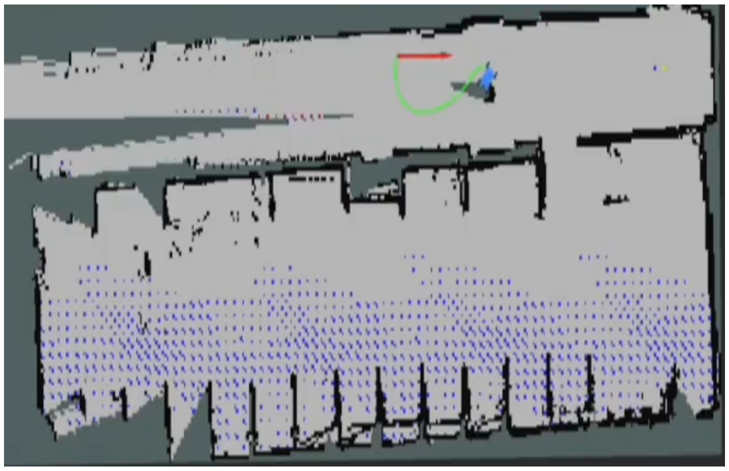


Figure 7: A successful search trial generates a map of the environment traversed by the robot and the aggregate value map created at the conclusion of the run.

order of 15 meters. Figure 7 depicts the SLAM map and aggregate value map generated at the end of a successful trial. This result shows that our project goal was successful in searching a complex environment to find a target object.

CONCLUSIONS

In conclusion, we achieved our initial goal of creating an autonomous room searching robot. By combining accurate SLAM map generation, image projection techniques using a color sensing camera, and intelligent decision making software, we created a self-driving miniature car that can perform an interesting and compelling task in the real world. Our team sees this project as a research exercise that highlights the capabilities of autonomous vehicles while also identifying potential issues in such designs and creating suggestion to address them. We hope that this project serves as a resource to other groups that are attempting similar projects and as an inspiration for those looking to further explore the world of autonomy.

Our testing highlighted aspects of the search algorithm that we could improve to increase its success rate and efficiency. The current navigation goal selection continuously updates, which can lead to oscillatory searching between similarly valued points. By changing the algorithm to only update when a goal is reached, we can circumvent this issue. Additionally, in order for maps to be generated faster and allow for trials to be repeated quicker, we can incorporate extra computing power or optimize the code itself for speed. Finally, we could further tune the adjustable parameters in the searching algorithm to increase the efficiency of the overall search algorithm.

REFERENCES

[1] "About F1Tenth." Web Page. https://f1tenth.org/about.html. Accessed: 07-Dec-2022.

[2] "cst0/gpio." GitHub. https://github.com/cst0/gpio_control. Accessed: 12-Jan-2023.

[3] "F1Ttenth Repository." GitHub. https://github.com/f1tenth. Accessed: 18-Nov-2022.

[4] "ROS Wiki." Web Page. http://wiki.ros.org/ROS/Tutorials. Accessed: 07-Dec-2022.

[5] "Slamtec/rplidar_ros." GitHub. https://github.com/Slamtec/rplidar_ros. Accessed: 18-Nov-2022.