REAL-TIME AUTONOMOUS MINIATURE CAR PERCEPTION AND CONTROL FOR PACKAGE DELIVERY

Selim Karahan*, Eduardo Lopez*, Brian Montoya*,
Justice Shepard*, Haodong Wu*, Sean Anderson†
Department of Electrical and Computer Engineering
University of California Santa Barbara
Santa Barbara, CA
skarahan@ucsb.edu, eduardolopez@ucsb.edu, brianmontoya@ucsb.edu,
justiceshepard@ucsb.edu, haodongwu@ucsb.edu, seananderson@ucsb.edu

Faculty Advisor:
João Hespanha

ABSTRACT

Autonomous vehicles increasingly play an important role in daily life. Low-stakes use cases can enable people's familiarity and confidence in these systems, helping to establish public trust. The Neutronomous project aims to enable food and package delivery via a low-cost, easy-to-fabricate one-tenth scale autonomous car. Three Raspberry Pis execute real-time parallel computations for sensing and localization via an environmentally robust LiDAR, and state estimation and high-level planning via an IMU-enabled GPS. This build is ideal for a self-driving robot to navigate college and corporate campuses, as well as small towns. The Neutronomous car's practical and accessible implementation paves the way for continued integration of autonomous vehicles in public spaces.

INTRODUCTION

The Department of Electrical and Computer Engineering at University of California Santa Barbara facilitates year-long undergraduate senior capstone projects in which teams of students work on a project that is often scoped and funded by an industry partner. In this case, the project was funded by the International Foundation for Telemetering.

The team aimed to develop a small-scale autonomous car based on the F1TENTH framework developed at the University of Pennsylannia [1]. Autonomous vehicles are becoming more common in daily life as well as popular culture but still face significant challenges for adoption. Technical challenges include difficult-to-predict road conditions, sensor interference, and variable weather

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

conditions. Hardware and overall cost of engineering development impose financial burdens, and the public needs to be able to trust autonomous systems. The team aimed to address the latter challenge by prototyping a small-scale car to efficiently deliver packages on campus settings. By safely demonstrating obstacle avoidance, the car would thus increase public confidence in such technology.

The F1TENTH project is an open-source project utilized in universities. The project enables students and researchers alike to learn about perception, planning, control, and the role of autonomous systems in society. The project is taught in the form of a course with the end goal of racing, hence the F1 (i.e. Formula One) in the name. Additionally, the scale of the car is one-tenth of a standard car.

Through the use of ROS packages and hardware inspired by the original F1TENTH project, the main technical challenges lay in integrating the hardware components and developing efficient code for real-time control. This paper aims to convey the hardware and software configuration as well as technical challenges faced during development.

VEHICLE HARDWARE

The vehicle components were all off-the-shelf in order to make the final product accessible and affordable. The components used were similar to those suggested on the F1TENTH website (final build in Figure 1). The main differences were the use of an IMU and GPS, no camera, and Raspberry Pi computers instead of a NVIDIA Jetson NX. The choice of components was based on a desire to use common sensors (i.e. IMU/GPS and LiDAR) as well as supply-chain availability (i.e. Raspberry Pi in lieu of the NVIDIA Jetson).

A. Sensing

At the sensing level, we mounted a BerryGPS-IMU on one of the single board computers and a RPLiDAR A3M1 on the front of the vehicle. The advantage of the GPS integrated with IMU is a compact form-factor that can be mounted on the Raspberry Pi. The LiDAR system provides 2D 360 degree scans of the surroundings with 20m range outdoors and 25 meter range indoors at a 10Hz scanning frequency. A key feature of the RPLiDAR product is its resistance to daylight interference allowing for both indoor and outdoor use.

B. Computers

For computation we used three Raspberry Pi 4 single board computers. Each has a 1.5GHz 64-bit ARM processor with 4GB of RAM. As illustrated in Figure 2, each Pi performed a specific task: sensing and low-level tasks, LiDAR processing and SLAM, and trajectory planning. Due to high computational needs for SLAM as well as the trajectory planning, it made sense to split this computation across multiple boards. While more powerful single board computers are on the market, the Raspberry Pi has a strong development community and is cost-effective. Another advantage of parallel computation is that it allows for parallel development of subcomponents (e.g.

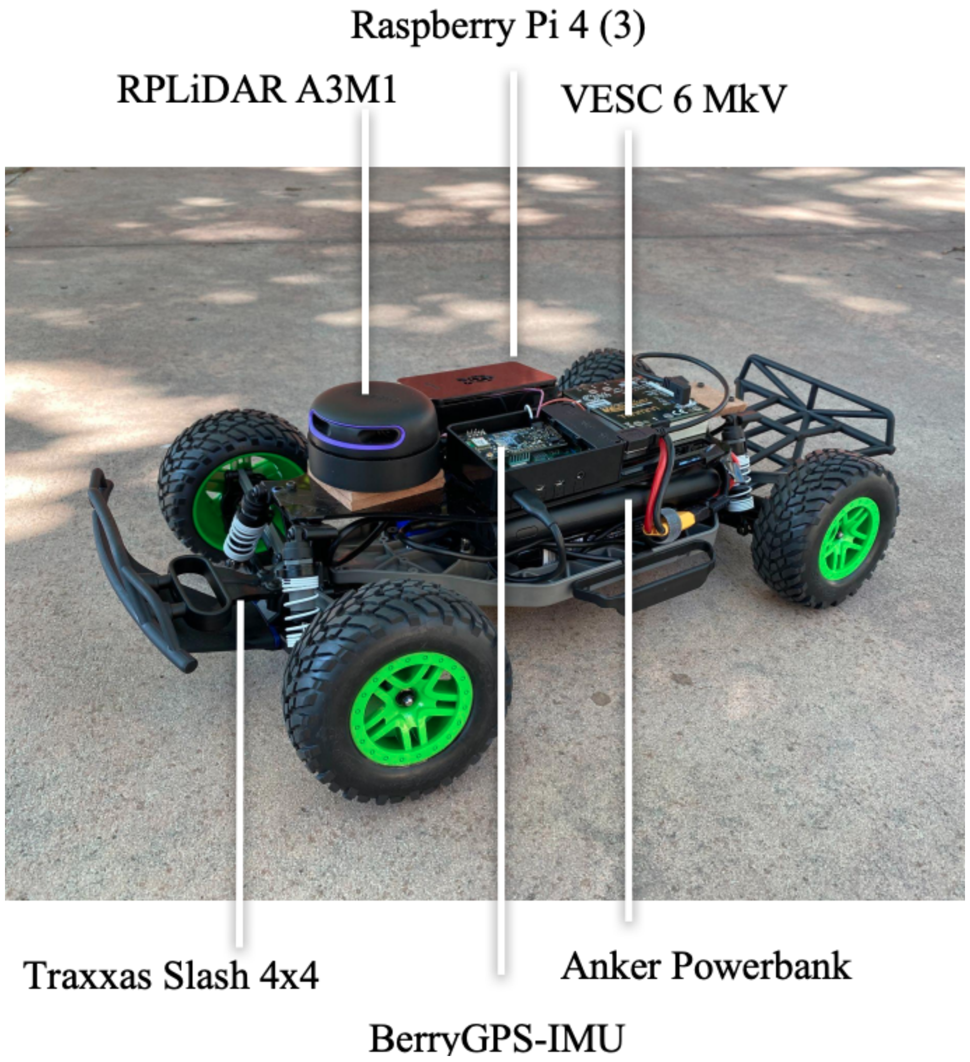

Figure 1: The fully-built car is shown with the LiDAR on the front of the vehicle.

the LiDAR was set up and tested independently before integrating with the rest of the stack). This allowed the team to test, develop code, and tune the sensors independently from one another during early stages of the process.

C. Vehicle base and actuators

Starting from the base, the team used a Traxxas Slash 4x4, which ships with the wheels, suspension, radio transmitter, 12V lithium polymer battery, remote control, antenna, electronic speed controller (ESC), Titan 12T 550 brushed DC motor, and a built-in steering servo. The radio transmitter and antenna were removed, with the ESC being replaced by a Vedder ESC (VESC). The stock product allows for speeds over 60 mph if desired. In order to mount the computers, LiDAR, and VESC on the car, we laser-cut a platform that would elevate these components above the motor and battery. The finished car is shown in Figure 1. An additional (Anker) power-bank is included for the purpose of separating the power supply of the car from the computers. This was simply to eliminate the need for a voltage converter.

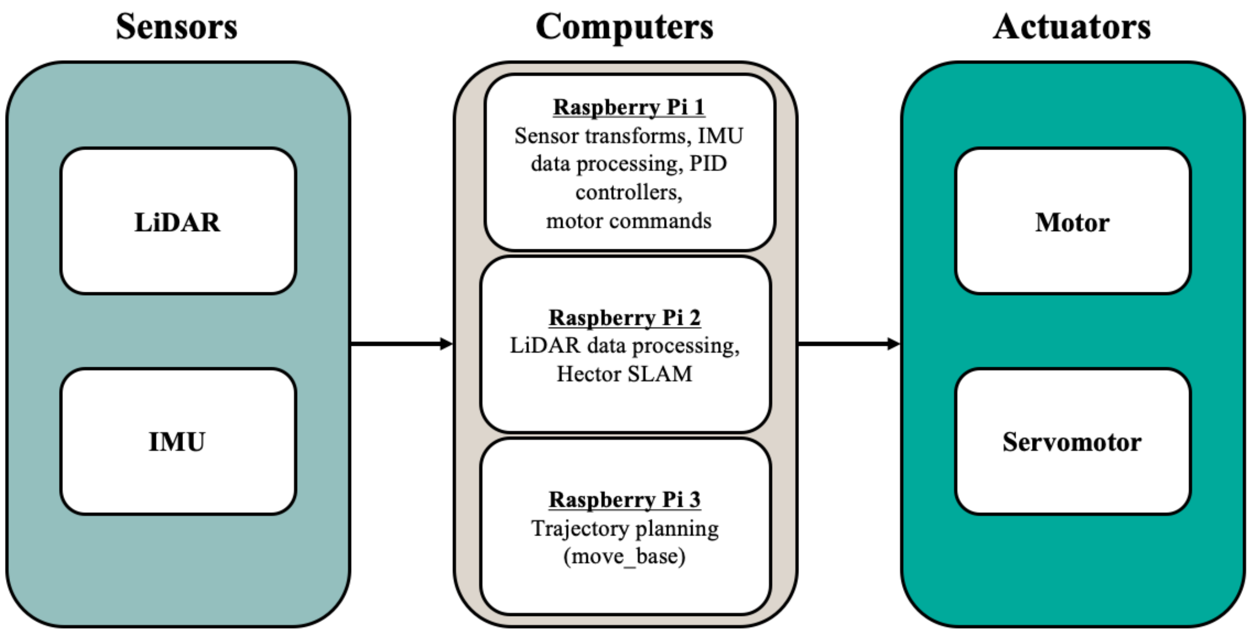
Sensors
Computers
Actuators
LiDAR
IMU
Raspberry Pi 1
Sensor transforms, IMU data processing, PID controllers, motor commands
Raspberry Pi 2
LiDAR data processing, Hector SLAM
Raspberry Pi 3
Trajectory planning (move_base)
Motor
Servomotor

Figure 2: The sensors feed data to a corresponding Raspberry Pi, which after performing computations, outputs a control signal to the actuators.

SOFTWARE

The team utilized ROS [2] packages to enhance the adaptability of the system as well as to enable off-the-shelf algorithms for perception and control. The overall software architecture is illustrated in Figure 3.

D. Sensing

The team focused on indoor navigation to start, such that the RPLiDAR and IMU provided the sensing. The GPS in the GPS-IMU only functions appropriately outdoors. The IMU was used for generating odometry data, namely pose and linear and angular velocity. By using the RPLiDAR, the team could use simultaneous localization and mapping (SLAM), which estimates the car's position in a local map (i.e. the area that is visible to the RPLiDAR) as well as the car's pose. An example of accurate mapping in the lab space is shown in Figure 4(a). With the intention of running in real-time, the SLAM algorithm allows for continuously building the local map as the car moves throughout the space of interest. In particular, the team used the ROS Hector SLAM package [3]. Hector SLAM is a two-dimensional SLAM system based on a robust scan matching technique, suiting the two-dimensional scans coming from the RPLiDAR.

In addition, because the SLAM data was useful for odometry purposes in addition to generating

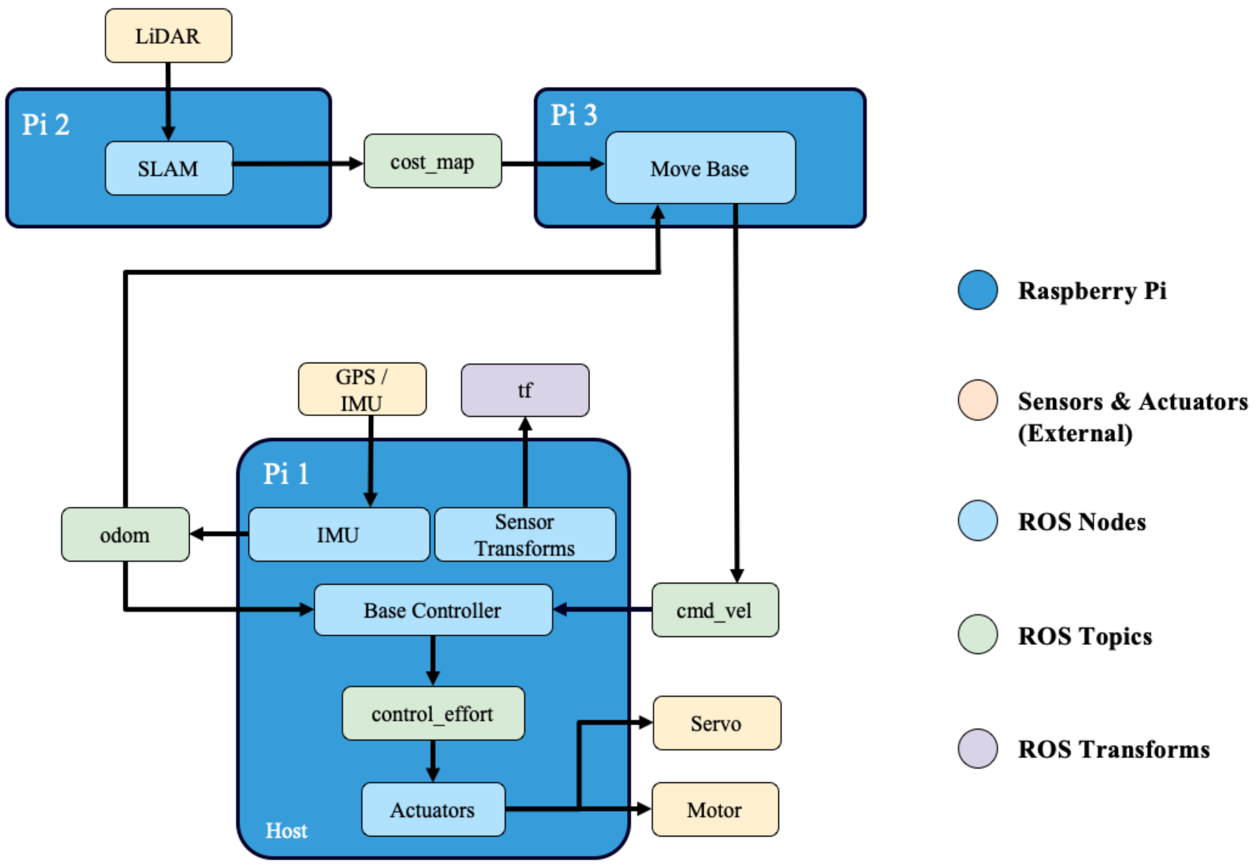

Figure 3: The ROS software configuration is shown superimposed on the Pi boards where computations would take place.

local cost maps, the team utilized the *robot_localization* ROS package [4]. This package enabled sensor fusion between the IMU and the SLAM output by using an extended Kalman filter (EKF). This generates more accurate odometry data. For example, IMUs rely on accelerometers and therefore do no generate useful information when the car is moving at constant velocity in a straight line. Meanwhile, the linear velocity can be inferred from the rate of change in the local cost map from the SLAM algorithm. In this way, sensor fusion can help to improve the state estimates of the system.

It's worth noting that the GPS can be used to improve state estimates when outdoors as well as provide the location of the car when traversing large areas. Due to the focus on indoor navigation for the duration of the project, the GPS was not utilized.

An issue that the team ran into when working on SLAM was distortion of the local cost map when turning (Figure 4(b)). The result of this was an inaccurate cost map. The underlying reasons for this distortion were ultimately unclear but were potentially related to the Raspberry Pi being under-powered for handling the RPLiDAR's data or more fundamental configuration issues when setting up Hector SLAM.

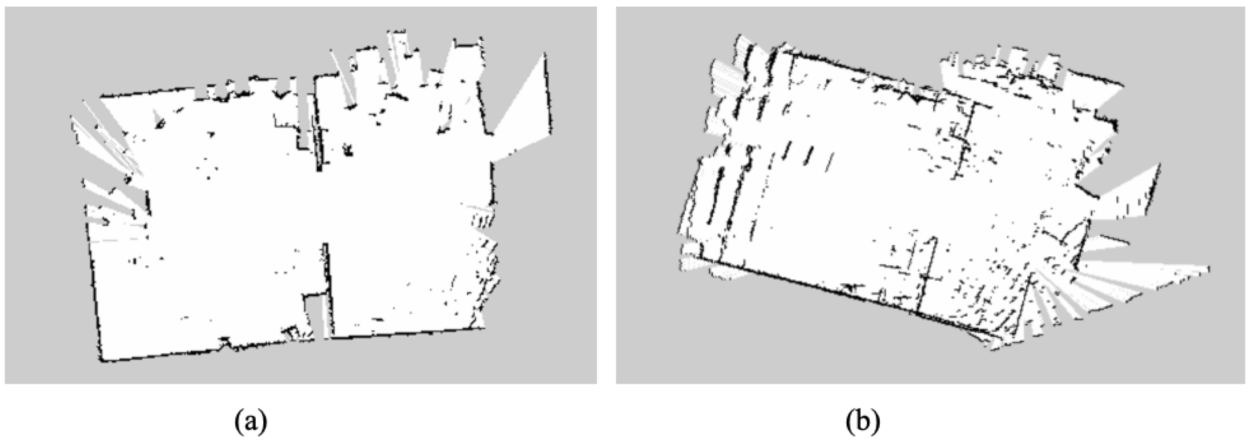

Figure 4: (a) Hector SLAM is able to accurately map the lab space where multiple small objects such as chairs and desks occlude small sections. (b) The team encountered issues when turning the car, resulting in inaccurate maps.

E. Planning and Control

The planning and control system contained three levels. The two highest levels planned the path of the car, and this was executed using the ROS package *move_base* [5]. Generally, *move_base* uses two cost maps. What is known as the "global cost map" provides a general map of the area capturing larger, static features. In the context of this project, this can be generated from exploring a region of interest, such as a campus, by using manual control while running the SLAM algorithm. By using the GPS for localization on the global cost map, waypoints could be determined for where to go next. The lower level planning (second level) utilized the "local cost map". This captures real-time information such as dynamic obstacles and local features left out of the more granular global version. The actual path planning algorithm used is A^*. Lastly, *move_base* is configured to output a velocity and angular velocity reference. Due to the system being nonholonomic, this was transformed into a velocity and steering angle reference. In turn, a PID controller communicating with the VESC tracked the reference linear velocity, and similarly a PID controller working with the servo tracked the reference steering angle.

F. Integration

The team developed the subsystems in parallel with the aim to integrate them using the communication flexibility provided by ROS. The architecture depicted in Figure 3 indicates the information flow. In particular, SLAM using the raw data from the LiDAR publishes to *cost_map*, which is the local cost map. The local cost map, also known as an occupancy map, defines whether a point on the mapped space is occupied. The data is stored in a matrix with zero corresponding to empty and one corresponding to occupied. By using Bayesian inference, the entries become probabilities of occupancy rather than simply boolean. The local cost map gives the path planner, *move_base*, information about local obstacles and can be used for sensor fusion. At the same time, the GPS and IMU are providing odometry data to the *odom* topic, which is read by *move_base*. Utilizing these two topics, *move_base* publishes the aforementioned references to *cmd_vel*. The references are then transformed in the base controller to desired velocity and steering angle. In order for the base controller to utilize feedback, it subscribes to the *odom* topic to get the current linear velocity. Since there are not sensors to read the actual heading angle, the current steering angle can be inferred from the combined linear and angular velocity. Lastly, the base controller (both PIDs) publish to *control_effort*. The control signal is sent to the servo and motor.

It is worth noting that since the sensors are not all in the same place on the car, the *tf* topic allows for transforms such that all computations use the same coordinate frame.

When integrating the subsystems, the team faced a few challenges. At the sensing level, the team encountered persistent issues with establishing an accurate cost map due to the distortion when turning. This limited testing to driving in straight lines or very slow turns. Finally, during the final days of the project, after getting *move_base* to output reference signals, the base controller would not compile due to issues in ROS. This led to the inability to fully test the autonomy of the car. As a result of this, the team was only able to demonstrate the performance of the car moving under manual control. This manual control was instated earlier on in the project to allow for developing global cost maps. By accessing the Pi via secure shell protocol (SSH), the team controlled the speed and turning of the car via keyboard commands.

CONCLUSIONS

The capstone project aimed to develop a miniature autonomous car for campus-scale delivery purpose. While the end result did not meet the original goal, the team was able to gain valuable experience in working with sensing/perception, planning and control, and software development with ROS. Furthermore, by considering the objective of furthering public trust in autonomous systems, the team engaged with the question of what role autonomous systems play in society. In this way, the team was successful in meeting the goals of the F1TENTH project. By using different components than the well-documented ones on the F1TENTH website, the team set forth a greater challenge for itself. If given more time in the future, it would be rewarding to finish the car and demonstrate the capabilities originally set forth.

ACKNOWLEDGEMENTS

The team would like to thank Destin Wong, who was a teaching assistant for the course, as he contributed much time and valuable insight throughout the project. We would also like to thank the International Foundation for Telemetering for funding the project.

REFERENCES

[1] B. Zheng and J. Betz, “F1TENTH.” https://f1tenth.org/, 2022.

[2] I. Open Source Robotics Foundation, “ROS.” https://ros.org/, 2022.

[3] S. Kohlbrecher, “hector_slam - ROS Wiki.” http://wiki.ros.org/hector_slam, 2022.

[4] T. Moore, “robot_localization - ROS Wiki.” http://wiki.ros.org/robot_localization, 2022.

[5] E. Marder-Eppstein, “move_base - ROS Wiki.” http://wiki.ros.org/move_base, 2022.