California Polytechnic State University Pomoma
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

F1TENTH Senior Design Project

Report for Fall 2024

Prepared by
Jared Alanis, Jesse Araiza, Emmanuel Marcial, Mike Plata,
Daniel Sneddon, Jordan Tejada, Alec Voong

Present to
Prof. Anas Salah Eddin

Date
12/13/2024

Abstract:

The CPP F1TENTH senior design project focuses on the development and implementation of full-scale Formula 1 challenges to 1:10-scale autonomous race cars, providing a platform for research in robotics and autonomous systems. The team's goal is to design, build, and program an autonomous vehicle that can readily adapt to complex environments with minimal human intervention. To accomplish this, the autonomous vehicle makes use of a Nvidia Jeston Orin Nano as its on-board computer, which communicates with a VESC 6 MKV motor controller, Intel RealSense D345i camera, and a Hokuyo UST-10LX LiDAR sensor. All communication between the various components is facilitated using ROS2, with testing facilitated in a simulated environment. The camera and LiDAR sensor provide real-time environmental data, enabling the vehicle to accomplish tasks such as Automatic Emergency Braking (AEB), object detection, wall/gap following, and Simultaneous Localization and Mapping (SLAM) with the use of advanced algorithms. Recent SLAM improvements include graph-based pose estimation, loop closure for map refinement, and particle filtering for precise localization. A newly implemented Follow Gap algorithm dynamically selects navigation points to enhance obstacle avoidance, though further refinements are being explored to address challenges in highly complex environments like Levine_obs.

Additional system enhancements include camera calibration, lane detection, and object recognition, with deployment via TensorRT for high-performance inference. By integrating modular hardware and advanced software algorithms into a cohesive framework, the CPP F1TENTH project contributes to the development of scalable and accessible autonomous vehicles while advancing research in intelligent navigation systems.

Table of Contents:

List of Tables:

Introduction:

With the advancements made in machine learning and artificial intelligence, the industry of autonomous vehicles has rapidly grown in recent years. As the industry has grown, many demands have come from consumers and governing bodies. Demands include low cost, ease of accessibility for research and development, and high safety standards. Under the F1TENTH platform, there has been a concerted effort to develop a solution to these demands. The use of $1/10^{th}$ scale RC cars removes the need for high budgets and large testing facilities while allowing hardware development of autonomous driving vehicles and progressing algorithmic complexity and robustness. F1TENTH competitions allows various teams to compete against each other to see what hardware and software is most beneficial to meet the demands of the consumers and governing bodies.

Background:

The F1TENTH platform addresses the growing demands in the autonomous vehicle industry by offering a low-cost, accessible solution for research and development while maintaining a focus on safety and innovation. Autonomous driving technology has traditionally required significant resources, including high-budget testing facilities and full-scale vehicles, which limit accessibility for smaller research teams and academic institutions. By utilizing $1/10^{th}$ scale RC cars, the F1TENTH initiative reduces these barriers, enabling more teams to participate in the development of cutting-edge hardware and software systems.

Previous research in autonomous driving has primarily focused on large-scale vehicles, emphasizing real-world applications and commercial deployment. Projects like the DARPA Grand Challenge have paved the way for advancements in autonomous navigation, mapping, and control. However, these efforts required substantial financial investment and infrastructure. At the same time, smaller platforms such as TurtleBot and Robotic Operating System (ROS)-enabled robots have focused on educational and research purposes but lack the scalability and competitive environment provided by F1TENTH.

The F1TENTH platform builds on these efforts by providing a robust, adaptable system for testing algorithms in a competitive setting. Teams worldwide have developed algorithms for collision avoidance, simultaneous localization and mapping (SLAM), and path planning, contributing to a growing body of knowledge. Despite this, a gap remains in addressing the challenges of seamlessly integrating modular hardware and software solutions for robust real-time decision-making in dynamic environments. Additionally, existing research often lacks a focus on affordability and scalability, critical for broader adoption of autonomous driving technologies.

This project seeks to address these gaps by developing and improving algorithms for navigation, obstacle avoidance, and localization while optimizing the hardware configuration to balance performance and cost. By building on the advancements made by the F1TENTH community, this project explores new approaches to SLAM, such as graph-based localization and particle filtering, and implements innovative obstacle avoidance algorithms like Follow Gap to enhance performance in complex environments. These contributions aim to advance the state of

autonomous vehicle research while making the technology more accessible and effective for a broader audience.

Requirements and Specifications:

Using the Restricted Class rules for July 7th,2024 of the F1TENTH competition[6] the following table shows the recommended hardware vs. the equipped hardware of the vehicle.

Table 1. Recommended and Equipped Specifications

Specification	Recommended	Equipped
Chassis	1:10 Traxxas	Traxxas Slash 4x4 Platinum Edition
Computation Unit	Nvidia Jetson Xavier NX | Orin Nano | TX2 | Nano. INtel NUC, Raspberry Pi	Nvidia Jetson Orin Nano
LiDAR	Hokuyo UTM-39LX	Hokuyo UST-10LX
Camera	Logitech C270 | C920, Raspberry Pi Camera Module V2, Arducaam, Intel Realsense, ZED	Intel Realsense D345i
Engine[1]	Velineon 3500kV	Velineon 3500kV
Sensors[2]	None	None
Tires	None	Traxxas 2.2 inch
Battery	4s Rated	5800 mAh 2s LiPo

[1]*Only one engine is allowed per vehicle.
[2]*No indoor GPS sensors are allowed.

Design:

Hardware:

The Traxxas Slash 4x4 Platinum Edition was modified by previous teams working on the project. A deck was 3D modeled to fit the electrical hardware required for autonomous driving and after prototyping was completed, the deck was constructed from ½" acrylic sheet. The VESC 6 MKV was chosen as the speed controller for the car. The controller is chosen because it exceeded the standards of previous years requirements.

For the powerboard circuit, several requirements must be met. The draw current and power output must be adequate enough for the Lidar, Microcontoller, and Camera components. The voltage and current requirements are listed in the table below.

Table 2. Voltage and Current Draw for Components

Component	Voltage Requirement(V)	Current Requirement(mA)
LiDar (Hokuyo UST-10LX)	12	150(450[1])
Microcontroller (Nvidia Jetson Orin Nano)	5	1400(2000[2])
Camera (Intel Realsense D345i)	5	700

[1]*450 mA is drawn on start up

[2]*If on 10W power rating, 2000mA drawn

To meet these requirements we are using a LM7805 to supply voltage to the microcontroller and camera, while the LiDAR has its voltage supplied by a PDQ30-Q24-S12-D. Below, the input voltage, output voltage and max current are given for each voltage regulator.

Table 3. Voltage and Current Parameters of Voltage Regulators

Regulator	V_{IN}(V)	V_{OUT}(V)	I_O MAX(A)
LM7805	7.5(min)	5	2.5
PDQ30-Q24-S12-D	9-36	12	2.4

With the regulators in series, the total current from the components must not exceed the lowest value of I_O MAX, which is 2.4A. This constraint leaves the only viable option for the Jetson to be on the 7W power rating, resulting in a total current draw of 2.25A, which is less than the 2.4A limit. All values were taken from datasheets provided by the manufacturer.

Software:

The F1TENTHS platform offers several labs to build a basis for creating and testing various software components of the autonomous rc car, labs 4 and 5 were developed upon these past few months. Lab 4 focuses on the implementation of a follow the gap algorithm (FTG) for our vehicle. This algorithm is important when it comes to competitive racing, as it is a key feature

for a F1TENTH vehicle to maintain high speed while simultaneously avoiding potential hazards such as an obstacle or an opponent.

Lab 5 involves the implementation of SLAM on the vehicle. When choosing which SLAM package to use, we compared Google Cartographer and SLAM toolbox. We decided on using the SLAM toolbox package as our SLAM algorithm since it has many capabilities and it could be easily integrated with ROS2. SLAM toolbox would be able to perform 2D mapping through a graph based approach and implement a relative based approach during localization to identify its position on the map. We decided on using the LiDAR sensor data to track the position of obstacles and identify its location on the map. The package would also need to include Nav2 for localization. This allows the vehicle to navigate and generate a map autonomously.

Integration and Test Results:

FTG:

We were tasked with improving or replacing the existing implementation of Lab 4, which was inefficient at the time. The FTG algorithm itself relies on preprocessing the received Lidar data by smoothing the data in order to reduce noise which may impact the algorithm's performance, while also pairing the lidar points with angle points in front of the car. After preprocessing the data we then find the closest point to the vehicle, this is the point where we will create a "bubble" around, all other points inside this bubble are set to 0, this signals this area is a hazard to avoid. Any non-zero point is then considered free space, a sequence of free space is considered a gap, the next step is to find the largest consecutive sequence of free space, or the largest gap. The largest gap is where the vehicle will find the best point for its steering angle, following the gap.

Our FTG implementation performed well in simulation on most maps, including maps with obstacles present. The algorithm struggles in the more complex obstacle map, as the vehicle can potentially select a point that appears to be safe but in reality it is not. This is because the algorithm is not taking into account the physical width of the car, which causes the car to collide with the obstacle on its corners or clip it with its sides. Various steps were taken to correct this error. Penalizing points in a gap with a high angle, signals to the vehicle that there is a corner or obstacle close to that point, reducing the likelihood that the vehicle will consider the gap as a viable route. The introduction of the penalty system improved the simulation performance and reduced the amount of collisions, but slight clipping into obstacles in certain cases still exists. If improvement for this issue is still desired it is suggested to instead focus on implementing the Disparity Extender Algorithm, which is a tweak to FTG and designed to address this "disparity" issue but also more efficient than the basic FTG algorithm. The DEA algorithm has its limits as well, specifically on a wider track where the vehicle can be close to the narrower side of the track at a turn but instead of following the turn it can choose to follow the longest path instead, which would be the corner of the map. Therefore it is likely that the larger focus should be on more efficient algorithms, such as Pure Pursuit and SLAM, where limits such as these aren't encountered.

SLAM:

Our integration of SLAM involved installing SLAM toolbox onto our ROS2. SLAM toolbox can be run in different modes such as online synchronous or online asynchronous. The difference between the modes is the quality of map generation and its localization. Synchronous mapping provides the ability to map and localize in a space keeping a buffer of measurements to add to the SLAM problem. [7] By contrast, the asynchronous mode will only process new measurements when the last measurement is completed and the new update criteria are met. [7] The synchronous mode works better when generating a smaller map and during offline mapping, but we would run the online asynchronous mode for better identification of the vehicle's position. We also need to make sure that the sensor_msgs/LaserScan has the scan topic subscribed. Through our test on the simulation, it had resulted in an error of running SLAM toolbox and this was likely because of the parameters that were set as default in online asynchronous mode or the transform variables in our TF tree. Once we modified the SLAM file, the package ran without any errors, but there were no sensor lines generated on the simulation. We could not identify if the robot was mapping without the sensor, or if it was not mapping at all.

ROS2:

We decided to create a separate simulation by installing all the necessary packages of ROS2 and the simulator such as Gazebo and Rviz. We then had included all the description files for the vehicle. Once the SLAM toolbox mode file was launched, it displayed the robot in Rviz and Gazebo with an edited world. The display on Gazebo shows the lines that the lidar sensor sends out to recognize if there is a wall or object in the way. On the Rviz simulator, it displays what the vehicle sees and starts the mapping process. We were able to manually control the vehicle in the simulation and as the vehicle moves, it generates a map of its surroundings.

The transition to ROS 2 Humble Hawksbill significantly impacted our project. Humble extended support to Ubuntu 22.04, necessary for our transition to the NVIDIA Orin Nano. This release also required C++14 or later and supported Python 3.10, aligning with our new hardware requirements. We observed enhanced real-time performance and more robust Quality of Service (QoS) handling in Humble.

The rclcpp and rclpy libraries gained valuable features in Humble, including improved lifecycle node utilities, synchronization primitives, and introspection capabilities. These advancements streamlined our development and debugging processes. The ROS2 launch system received substantial enhancements, simplifying our launch configurations. The ROS2 bag tool improved recording and replay features, aiding in our test analysis. These tooling improvements contributed to increased efficiency in our development workflow.

Humble introduced new standard message sets and improved custom message type handling, facilitating data exchange between our vehicle's nodes. The updated Nav2 stack and perception libraries, such as image_pipeline, provided valuable tools for our autonomous vehicle's navigation and sensor processing. Humble replaced deprecated packages from Foxy, ensuring our project remains current with the latest advancements in robotics.

This transition also necessitated a review of our existing nodes, including the Safety Node, Wall Following Node, Gap Following Node, and SLAM Toolbox waypoint logger node. For the waypoint logger, we updated the quaternion-to-Euler angle conversion to utilize the tf_transformations library, ensuring compatibility with potential future changes in the tf2_ros library. Additionally, we replaced all print() statements with self.get_logger().info() calls across our nodes, aligning with ROS2 best practices. This change improved code maintainability and better integrated our logging with the ROS 2 framework, allowing for more effective filtering and visualization of log messages, facilitating debugging and analysis.

While the core functionality of subscribing to /scan, processing LiDAR data, and publishing to /drive remained largely unchanged for the Safety, Wall Following, and Gap Following nodes, we made several adjustments to enhance compatibility and performance with the new ROS 2 release. We leveraged Humble's improved QoS profiles by using qos_profile_sensor_data for the /scan subscription, ensuring efficient and timely delivery of high-frequency sensor data, crucial for our autonomous vehicle's performance. Furthermore, we considered implementing parameterization to enhance node flexibility. By exposing key parameters like bubble_radius, corner_speed, and other gains as ROS 2 parameters, we enabled dynamic reconfiguration during runtime, facilitating rapid experimentation and tuning without code recompilation.

We carefully reviewed the handling of LaserScan data, ensuring accurate processing of angle_min, angle_max, and angle_increment for precise wall-following error calculation. This review included verifying the correct handling of potential noise or missing data in the LiDAR readings. Similarly, for the waypoint logger, we reviewed its behavior in the Humble environment to ensure compatibility with the updated libraries and message definitions.

Finally, we thoroughly tested the nodes in the Humble environment, leveraging enhanced debugging tools such as rqt with its plugins, ros2 topic echo, and ros2 topic hz. These tools provided valuable insights into node behavior, aiding in the identification and resolution of potential issues.

Enhanced tutorials, API documentation, and community resources within Humble proved invaluable for our team. The numerous enhancements in Humble, coupled with the necessary code adaptations, significantly improved the platform for our project, enabling us to develop a more efficient and reliable autonomous racing vehicle

Standards and Constraints:

Constraints:
The constraints for the project were set by the F1TENTHS competition. Ensuring the components of the car were within the requirements of the Restricted Class. Major areas of constraint coming from team members were time, knowledge, and project organization.

Time was the largest constraint on most members. Many of the members were full time students and employees. The outside responsibilities made it difficult to be able to concurrently work on the project. An optional weekly meeting was made with the advisor, but this was not enough to keep most members informed of the status of the project. To accommodate this challenge, a

Discord server is used to issue announcements for the groups, as well as to provide an environment in which members can help one another and with labs given by the F1TENTH competition. The server allows everyone to continuously be part of the project allowing the project to always be developed upon.

Many of the members had no prior experience working with autonomous vehicles. This caused a delay in learning the various softwares we were using when we initially joined the project. The project consisted of virtual machines, containers, python simulations and more. This resulted in us spending most of the first semester learning how the project was configured and how to navigate the software followed by the development of the software this semester.

To aid with project organizations, the team was divided into 3 subcategories this semester. The delegation of duties will be discussed in the next section. To organize the guides and scripts for the software, a Github repository has been in use for several iterations of the team. This semester, the repository was redefined for ease of use and understanding.

Standards:
We continued with the standards set by previous team configurations, using agile development principles. The standards set by agile development ensure group communication and workflow are easily accessible for all members. With use of the Discord server and GitHub repository, an asynchronous work schedule was seamlessly created. With the primary focus of the project being the development of the FTG and SLAM algorithms, we slowly iterated the processes to overcome challenges that arose from each iteration. The careful development of the processes allowed for ample time for all team members to understand changes and improvements made on the codes.

Project Planning and Task Definition:

The introduction to the project contained several foreign topics for many of the team members. The first semester of the project consisted of us installing the required softwares, and getting the first 3 labs of the F1TENTH platform running on our machines. The project was divided into multiple groups afterwards, with one focused on hardware, several focused on lab development, and a final group focused on optimization of the completed labs. The hardware team was able to complete the modification of the chassis with a custom acrylic deck. Several labs were slightly improved but ran into several errors.

This past semester, the onboarding process of new members was easily streamlined with the help of video tutorials from previous team members. Within the first couple of weeks the project was divided into three main teams. Team 1 was focused on lab 4 and removing collision errors. Team 2 was focused on lab 5 and developing a solution of mapping the vehicle and its surroundings. Team 3 was focused on researching ROS2 and how to implement Humble Hawksbill into the project and the changes it would bring.

Conclusion:

The CPP F1TENTH senior design project team successfully continued on the development of an autonomous driving vehicle in compliance with the F1TENTH's platform restrictions. The advancements made on the FTG and SLAM strategies have placed the project closer to its goal of competing in an official race competition. The transition to Humble Hawksbill improved the debugging capability of current nodes and will be valuable in future node development. Although collision in the FTG algorithm still occurs, further development using strategies discussed in the report would continue to reduce error in a dynamic environment. The vehicle is the closest it's been to be able to autonomously operate. Future iterations of the team will be able to have the vehicle operate with the current foundation left by our team, contributing to the needs project and expanding on algorithmic development.

References:

[1] Hokuyo, "UST-10LX Datasheet." [Online]. Available: https://autonomoustuff.com/-/media/Images/Hexagon/Hexagon%20Core/autonomoustuff/pdf/hokuyo-ust-10lx-datasheet. [Accessed: Dec. 13, 2024].

[2] Nvidia, "Jetson Orin Nano Datasheet." [Online]. Available: https://drive.google.com/file/d/1QRjinl0oWDu4lvRjRMz2J8PZ5_FNkhKV/view. [Accessed: Dec. 13, 2024].

[3] Intel, "Realsense D345i Datasheet." [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf. [Accessed: Dec. 13, 2024].

[4] CUI Inc., "PDQ30-Q24-S12-D Datasheet." [Online]. Available: https://www.cui.com/product/resource/pdq30-d.pdf. [Accessed: Dec. 13, 2024].

[5] Texas Instruments, "LM7805 Datasheet." [Online]. Available: https://www.ti.com/lit/ds/symlink/lm340.pdf. [Accessed: Dec. 13, 2024].

[6] F1TENTH, "F1TENTH Rule Set." [Online]. Available: https://iros2024-race.f1tenth.org/rules.html. [Accessed: Dec. 13, 2024].

[7] S. Macenski et al., "SLAM Toolbox," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.02783. [Accessed: Dec. 13, 2024].

[8] ROS, "ROS 2 Documentation: Foxy Fitzroy." [Online]. Available: https://docs.ros.org/en/foxy/Releases-Foxy-Fitzroy.html. [Accessed: Dec. 13, 2024].

[9] ROS, "ROS 2 Documentation: Humble Hawksbill." [Online]. Available: https://docs.ros.org/en/humble/Releases-Humble-Hawksbill.html. [Accessed: Dec. 13, 2024].