

Proposal of a Low Cost Navigation System for Starter Teams in Autonomous Racecar Competitions

Lima, Bruno S. * Moreno, Ubirajara F. **

* *Department of Mechanical Engineering, Federal University of Santa Catarina, SC, Brazil (e-mail: bruno.szdl@gmail.com).*

** *Department of Automation and Systems, Federal University of Santa Catarina, SC, Brazil (e-mail: ubirajara.f.moreno@ufsc.br).*

Abstract: This article is intended for student teams which aim to enter a autonomous racecar competition. Many starter teams have neither a consolidated knowledge in autonomous system nor a great financial support. Most papers focused on autonomous racecars introduce advanced systems with expensive hardware. Therefore, this work proposes a low-cost and simple navigation system for the autonomous racing problem. This paper focus on a specific event of the Formula Student Driverless competition, but it can serve as basis for other events and competitions. The development of the system is done using the framework ROS and simulated in Gazebo. From this perspective, this work can serve as a guide for starter teams to design their first prototype.

Keywords: Autonomous Vehicles, State Estimation, Localization, Map-building, Trajectory and Path Planning.

1. INTRODUCTION

Autonomous racecar competitions for student teams have been taken place in some countries for years, such as the Formula Student Driverless (FSD) in Germany and the Formula Student Autonomous Competition (FSAC) in China. The FSD is part of the Formula Student Germany, which also encompass the Formula Student Combustion and Formula Student Electric. The Driverless class was introduced in 2017, and since then several teams have made a great improvement on their projects.

With the advancement of technology and popularization of artificial intelligence (AI) and autonomous systems, many students teams wish to design their own vehicles and enter those competitions. Starter teams usually do not have neither a consolidated knowledge in autonomous system nor a great financial support. When they read the papers from the teams which have been for years in the competition, they come across with complex systems and expensive hardware, such as Kabzan et al. (2019), Bader and Hofmann (2017) and Chen et al. (2018), not knowing how to begin the project. It is hard to find a work that helps starter teams to enter this field.

This article aims to fill this gap between the starter teams and the competition, proposing simple and low cost solutions for a team to develop its first prototype quickly combined with a platform of simulation to evaluate the performance of the algorithms. These solutions are not the ones which will win the competition, nor are they the most optimized. However they will help the team to have the base knowledge and a prototype, so that they can run their experiments and realize which areas they have to focus on the most. Over time the team can acquire knowledge

and propose more optimized solution, leading to a more competitive car.

Although several different areas are necessary to be designed in order to have the car ready for the race, this article focus on four aspects of navigation: computer vision, state estimation, self localization and mapping (SLAM) and path planning. Furthermore, the system proposed is specific for the trackdrive event of FSD and it follows the 2020 rules. However, it can serve as basis for other events and competitions.

This paper is structured as follows. In Section 2 it is described the trackdrive event of the FSD and the overview of the proposal for the low cost navigation system. Section 3 introduces the suggested tools for the car's software development. In Section 4 it is described in more details the computer vision, state estimation, SLAM, path planning and control subsystems of the proposal. Results for simulations run by a starter team are shown in Section 5. Finally 6 is reserved for the conclusion of the work.

2. SCENARIO AND SYSTEM PROPOSAL

The trackdrive event is one of the dynamic events of the FSD and it is the most valued one in terms of points. It consists in a track bounded by cones with different functions and colors. The vehicle has no prior knowledge of the track, so it has to run a first lap in order to map the track. After that it can plan the optimal path, to run laps with the minimum possible time. After 10 laps are concluded, the vehicle stops.

The specifications of the cones are shown in Figure 1, however the small orange cone with a single white stripe is not used for this event. A layout of the track for





			
big orange cone two white stripes	small orange cone single white stripe	small yellow cone single black stripe	small blue cone single white stripe
WEMAS 307.610500.00.00	WEMAS 400.000013.00.00	WEMAS 400.000013.01.10	WEMAS 400.000043.00.00
285 mm × 285 mm × 505 mm 1.05 kg		228 mm × 228 mm × 325 mm 0.45 kg	

Figure 1. Specifications of the cones used in the Formula Student Driverless events.
 Source: FSG Competition Handbook 2020

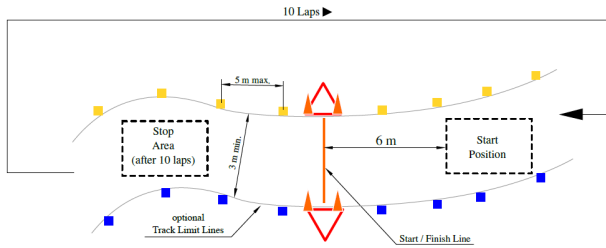


Figure 2. Layout of the track for the trackdrive event.
 Source: FSG Competition Handbook 2020

the trackdrive event is illustrated in Figure 2. It can be seen from Figure 2 the function of each cone: the small yellow cones mark the right border of the track; the small blue cones delimit the left border; the big orange cones determine the start and finish line. The minimum track width is 3m and the maximum distance between two cones of the same color is 5m. The Figure also shows the start position of the vehicle and the stop area after 10 laps.

For a simple and low-cost solution for the trackdrive event of the FSD, the sensors represent a large amount of the budget, so they are supposed to be in a few number. The sensors have two main functions in the car, which are to measure the state of the vehicle and to perceive the environment. There are two sensors, that combined can take care of first function, the GPS and IMU. The IMU is, in general, more accurate than the GPS, but its error build up quadratically over time. Because of this, the GPS needs to be introduced for correcting the IMU measurement in the state estimation.

The perceiving of the environment, meant for the cone detection can be done by a stereo camera. A pair of mono cameras could also be used, but it would require much more caution to calibrate and assemble the sensing platform. A LiDAR could also be useful, but it has a very high price. So, if the team does not have enough money, it is possible to use just a camera. In general, if the team is able to afford more sensors, LiDARs, wheel encoders and Ground Speed Sensors can make the state estimation and cone detection systems more robust.

From the chosen sensors, it is possible to build the system's architecture. The diagram of the proposal can be seen in Figure 3. In this architecture the GPS and IMU measurements serve as inputs for the State Estimation subsystem, while the images from the camera serve as input for the Computer Vision subsystem. Other sensors can be added depending on the availability.

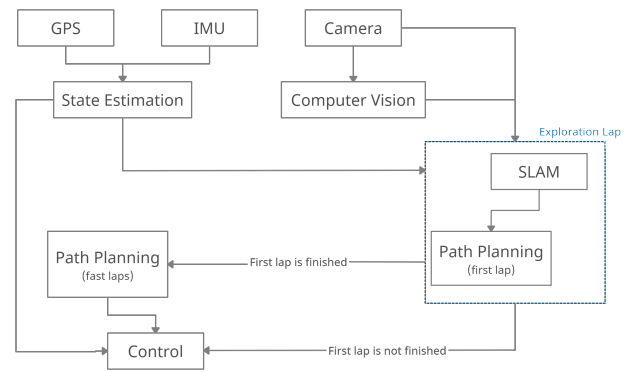


Figure 3. System's architecture

The State Estimation subsystem estimates the state of the vehicle using the Extended Kalman Filter (EKF) and sends this information to other subsystems. The Computer Vision subsystem uses a CNN to detect the cones in the vehicle's field of view (FOV).

The SLAM subsystem is responsible for creating and updating the map of the track which will be used in the future laps. While the car is mapping the track it has to move in order to complete a first lap. The Path Planning (exploration lap) subsystem decides to where the vehicle must go to complete the lap being aware of only a part of the map. The SLAM and Path Planning (exploration lap) subsystems compose a single subsystem called Exploration Lap. The Path planning (fast laps) subsystem is used to find the optimal trajectory for the first laps and the velocity and acceleration profiles.

Finally, a Control subsystem is used to send the command to the actuators in order to make the vehicle moves and it receives information from the State Estimation and from the Exploration Lap in the first lap. In the others laps, the information is given by the Path Planning (fast laps) subsystem instead of Exploration Lap.

3. ROBOT DEVELOPMENT TOOLS

There are many tools a team can use for the development of a robot's software, such as the Robotics Toolbox for MATLAB, the Microsoft Robotics Developer Studio and the Mobile Robot Programming Toolkit. However, there is another option with a great adhesion among software developers for robotics called Robot Operating System (ROS) and it has a native simulator named Gazebo. It is suggested for the teams to used these tools for the reasons discussed in the next subsections.

3.1 Robot Operating System

ROS is a robotics middleware suite. It contains a range of frameworks for robot software development and a lot of libraries and tools to support the project. It is an open source project and has a large and participatory community, hence it is very simple to find straightforward documentation and answers to eventual questions. There are a lot of ROS courses and classes available on internet and also many solutions already implemented by the community for well known and widespread problems of the robotics field. This middleware is used by many companies

and student teams, including the ones competing in FSD. It can be installed following the steps in

<https://www.ros.org/install/>

By the time of this work, it is recommended to use ROS melodic, for it has more consolidate documentation and packages. There are three concepts widely used in ROS that must be known for the understanding of the proposal, nodes, topics and messages.

- **Nodes:** A node, represented in yellow in Figure 4 is an algorithm which process some information. It is able to receive information as input and also to hand information as output. The node can represent a sensor, actuator, controller or anything that process information. For instance, a camera can be modeled as a node which receives information from the environment (light), transforms it into another type of information (image) and sends it to another node.
- **Messages:** A message represents the type of the information. There are numerous information types, it can be an image, a desired velocity, a distance from an obstacle, etc. The developer can even create his own types of message. Each message has a predefined structure. The message Twist, for instance, used in order to pass information of velocity, is composed by two three dimensional vectors, one of which stands for the linear velocity and the other one stands for angular acceleration. It is not necessary to have information of all the variables in the message, one can leave unknown values as zero.
- **Topics:** The node sends the message somewhere and receives the message from somewhere. This “place” is called topic, shown in blue in Figure 4. There are other ways to transmit information in ROS, such as services and actions, but it is not needed to use them for now. The topic is like a bulletin board, any node can publish messages into the topic and any node can read messages from the topic, which is called subscribe to the topic. Each topic only allows one type of message, but a message can be sent to multiple topics. The messages in the topics can be published and read in real time.

A robot software can be modeled in ROS using only these three concepts. As long as this work aims to propose a simple solution, only them are recommended.

Most of ROS tutorials and applications use omnidirectional robots or differential drive two-wheeled robots, but the racecar is a four-wheeled robot with Ackermann steering. The team can build its own robot from scratch, or it can use the catvehicle implemented by Bhadani et al. (2018), available in

<https://github.com/sprinkjm/catvehicle>

The team must modify the properties of the car to meet the competition rules.

3.2 Gazebo

ROS has a fully integrated simulator called Gazebo. It is hard to think about ROS without thinking of this simulator. It utilizes high-quality graphics and a very accurate physics engine. It is widely used for quick robot

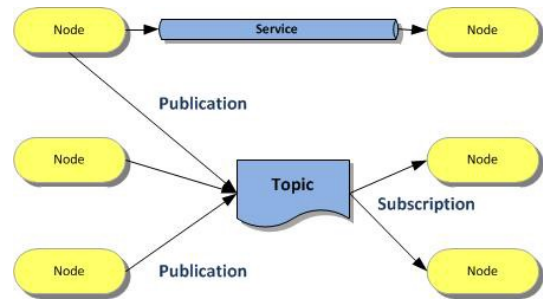


Figure 4. Representation of the information exchange between nodes by topics and services.
Source: Towards Data Science

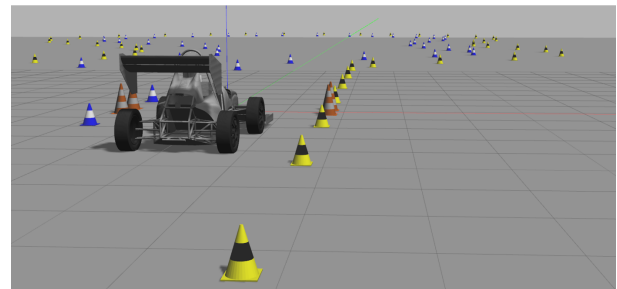


Figure 5. Car and cones models in Gazebo.

development and tests, even by teams of FSD and it has a large community. The simulator is very easy to use and has several plug-ins for the majority of the sensors used in autonomous cars. Figure 5 shows a screenshot of the simulator, containing a racecar and the cones marking the track. Robots and models can be created in Gazebo following the tutorials in

<http://gazebosim.org/tutorials>

3.3 Sensoring

This subsection describe how to simulate the three sensors defined in Section 2 in Gazebo. For more information it is recommended to visit

http://gazebosim.org/tutorials?tut=ros_gzplugins

GPS The GPS can be modeled using the ROS package `hector_gazebo_plugins`. It contains a plug-in that simulates a GNSS receiver. However, this GPS publishes messages of the `sensor_msgs/NavSatFix` type, composed of latitude, longitude and altitude information, and we will need messages of the `nav_msgs/Odometry` type for the state estimation. In order to do this conversion, the ROS package `gps_common`, available in

http://wiki.ros.org/gps_common

can be used. It has a node that reads `sensor_msgs/NavSatFix` and publishes `nav_msgs/Odometry` into the `/odom` topic.

IMU The same ROS package used for simulating the GPS can be used for the IMU. The `hector_gazebo_plugins` contains a plug-in that simulates an IMU sensor. This sensor publishes messages of the `sensor_msgs/Imu` type, composed of linear accelerations and angular velocities.

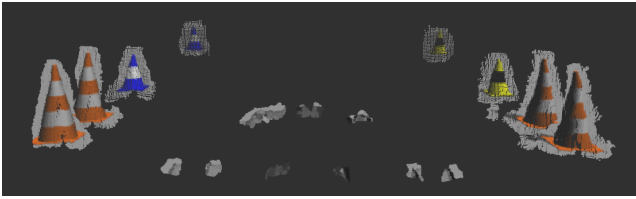


Figure 6. Pointcloud generated using the stereo_image_proc package.

This format can be used for the state estimation, so it needs no additional package.

Camera There is no plug-in in Gazebo that can be used for the simulation of a stereo camera. However, there are plug-ins for mono cameras and it is possible to mimic a stereo camera using two mono cameras. A plug-in for mono cameras can be found in the standard gazebo packages. The package stereo_image_proc, available in

http://wiki.ros.org/stereo_image_proc

reads sensor_msgs/Image messages from the cameras and is capable of publish a pointcloud of the environment. An example of this pointcloud can be seen in Figure 6.

4. SUBSYSTEMS

This subsection details the subsystems shown in Figure 3.

4.1 Computer Vision

The Computer Vision subsystem receives images from the camera in real-time and detects the cones in it, along with their colors. For the detection of the cones, a Convolutional Neural Network (CNN) must be trained. It is recommended to use the YOLO architecture Bochkovskiy et al. (2020) to build the CNN, and train it using the FSO CO Dodel et al. (2021) dataset. This dataset can be accessed by making a contribution with a set of images. Then, the CNN can be integrated with Gazebo with the ROS package darknet_ros, available in

https://github.com/Tossy0423/yolov4-for-darknet_ros

The package has a node which reads the topic in which the camera publishes the images, and then publishes the bounding boxes of cones into another topic. More information about the package can be found in the GitHub page. An example of the output of the CNN can be seen in Figure 7.

4.2 State Estimation

The State Estimation subsystem estimates the state of the vehicle. For the robot to localize itself or map the environment, it must have a good estimation of its state. The state, according to Simon (2006), is the set of variables which completely represents the condition of the robot in an instant of time. For an autonomous car the state is composed of the position, velocity and orientation.

To estimate the state of the vehicle it is recommended to use the EKF. This filter does a probabilistic estimation

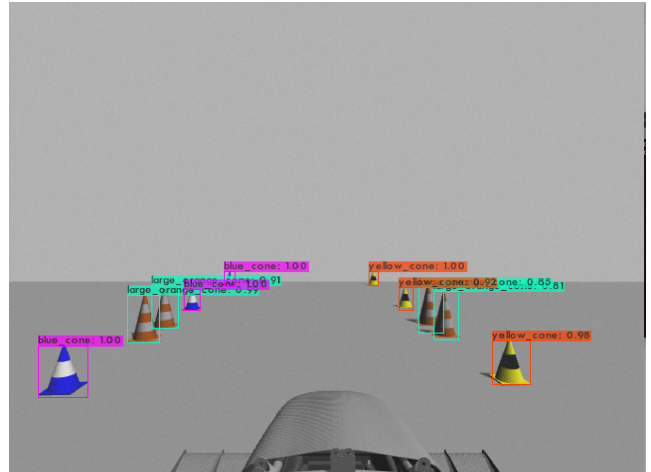


Figure 7. Bounding boxes for the cones.

and updates it in real-time. It has two phases, prediction and update. The prediction is done based on the motion model of the vehicle, using the measures of internal sensors and the kinematic and dynamic equations of motion. The update phase corrects the state based on the measure model based on the external sensors.

As it can be seen in Figure 8, the prediction phase can be run using the measurements from the IMU sensor as the input for the motion model, which predicts the state of the vehicle. The measurement model can receive the measurements from the GPS and its output is used to update the predicted state. In the next instant of time, the updated state will be used to predict a new state. This Figure illustrate the classical formulation for the EKF, whose equations can be found in many books, such as Simon (2006).

This filter is recommended because it is widely used in mobile robots and by other teams from FSD. There are numerous implementation of this method, even in ROS. Besides that, Xue and Schwartz (2013), showed that it is a better option for real-time and low cost applications. The results are not better than methods like Unscented Kalman Filter or Particle Filter, but it is more simple and faster.

To implement the EKF in ROS it is recommended to use the package

http://wiki.ros.org/robot_localization

It has a node that reads the messages published by the sensors, and it publishes the estimated state into another topic. More information can be found on the website. This package was developed for an omnidirectional robot, so the motion model must be modified to correspond to an Ackermann steering model.

4.3 Exploration Lap

Although there are several methods for SLAM widely accepted and used for mobile robots, such as EKF-SLAM (Smith and Cheeseman (1986)) and Fast-SLAM (Montemerlo and Thrun (2007)), they are not very simple to understand and implement for the track scenario. It is recommended for the starter teams to develop their own

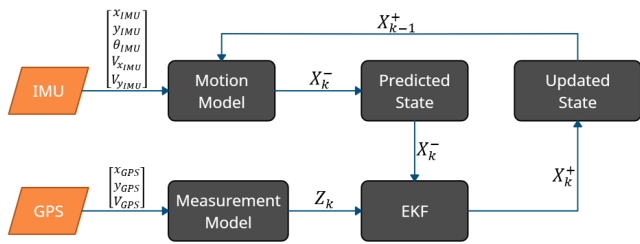


Figure 8. Diagram representing the EKF's information flow.

SLAM method in a simple way focused only in the characteristics of the trackdrive event.

The map serves for the vehicle to plan its trajectory. To do this, the racecar needs only the estimation of the boundaries of the track. As the boundaries of the track are marked by the cones, the map can be composed just by the position of the cones.

A node can publish the position of the cones and the vehicle in real-time using the information given by the camera and by the Computer Vision subsystem. The cones ahead of the vehicle and their colors can be detected by the computer vision. Then it is needed to match a pixel from each bounding box, such as the central pixel or a pixel below the center, to its corresponded point in the pointcloud provided by the camera. The pointcloud will show the distance from the cone to the camera. After that, the state of the vehicle can be used to transform the position of the cone detected by the car into a global position. Another node can read the global position of the cones to build a map and update it as the node receives more information.

However, to complete the exploration lap, the vehicle must move across the whole track to the finish line. It is not possible to plan a path from the beginning to the end of the track because it is not completely known. So the vehicle must plan its next trajectories based only on the partial map it currently has.

To do this, a suggestion is to draw lines connecting blue cones to yellow cones or vice versa, as shown in Figure 9. The strategy of when to draw or not draw these lines can be defined by the team. At the center of these lines, nodes can be generated. If the mapping is sufficiently accurate, the nodes will be generated near the central line of the track. So all the vehicle has to do is follow these nodes as they are generated. To decide which node to follow, a search strategy can be implemented, such as Uniform cost search or A* search.

When the vehicle finds the finish line node, it means the first lap was completed. The map can then be saved in a file containing the estimated position of the cones and the nodes. This map will be used in the trajectory planning for the fast laps.

A common problem the team can face while exploring the lap is the lack of visibility in tight turns as shown in Figure 10. In these kind of situations the vehicle can't see the internal cones of the turn and might not be able to draw lines connecting cones and generate nodes. The team must be aware of this possibility and implement rules for the vehicle to follow in this scenario. For instance, making the

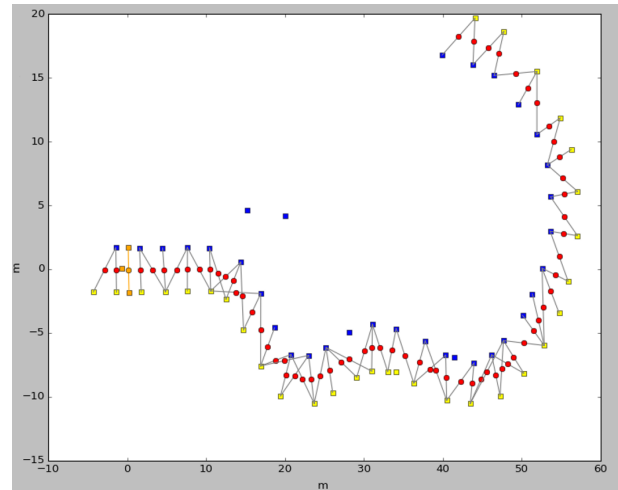


Figure 9. Example of map being built. The blue and yellow squares represent the estimated position of the cones with their respective colors. The red circles are the nodes the car can follow.

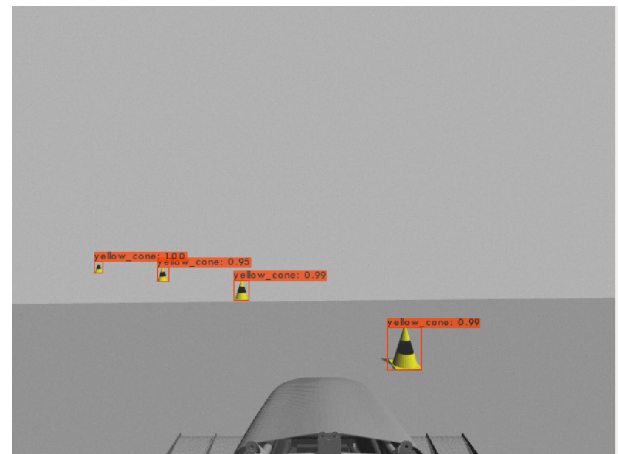


Figure 10. Example of tight turn. There is no blue cone in the FOV of the vehicle.

turn left if there is no node to follow and only yellow cones in front of the vehicle, and the same thing for blue cones, but turning right, can be a simple working solution.

4.4 Path Planning

After completing the first lap, the vehicle has a full map of the lap, so it can plan a full trajectory for the fast laps. The optimal path is the path which allows the car to complete the track within the minimum time possible. Braghin et al. (2008) suggests that the optimal path can be the one with the minimum length. However, this path leads to tight turns, forcing the vehicle to decelerate, losing time. With that in mind Braghin then suggests that the optimal path could be the one with the minimum curvature, but he concludes that the optimal path is an interpolation of the minimum curvature and the minimum length paths.

Heilmeier et al. (2019), based on the Braghin's work, developed a new method to optimise the minimum curvature path and it proved to be the most optimised path. Heilmeier formulates this optimisation problem which varies the raceline points r along the track widths, in

order to minimize the curvature. The raceline points \mathbf{r}_i are defined as

$$\mathbf{r}_i = \mathbf{p}_i + \alpha_i \mathbf{n}_i, \quad (1)$$

where $\mathbf{p}_i = [x_i, y_i]^T$ is a point of the track's centerline and $[x_i, y_i]^T$ are its coordinates. The parameter \mathbf{n}_i is a unit length normal vector, perpendicular to the track's centerline and α_i is used to move the point along the normal vector, always between the track's borders. So,

$$\alpha_i \in \left[-w_{tr,left,i} + \frac{w_{veh}}{2}, w_{tr,right,i} - \frac{w_{veh}}{2} \right], \quad (2)$$

where $w_{tr,left,i}$ represents the width of the left part of the track given a specific point i , and $w_{tr,right,i}$ the width of the right part. The parameter w_{veh} corresponds to the width of the vehicle.

The raceline is then defined by third order spline interpolations of the points r . For instance, the x-part of the spline and its derivatives with respect to t can be described as follows

$$x_i(t) = a_i + b_i t + c_i t^2 + d_i t^3, \quad (3)$$

$$x'_i(t) = b_i + 2c_i t + 3d_i t^2, \quad (4)$$

$$x''_i(t) = 2c_i + 6d_i t, \quad (5)$$

where a_i , b_i , c_i and d_i are the spline parameters and t is the normalised curvilinear parameter along one spline segment at the distance s starting at the distance s_{i0} , so

$$t_i(s) = \frac{s - s_{i0}}{\Delta s_i}, \quad (6)$$

The optimisation is done minimizing the discrete curvature κ_i^2 of the splines summed along the raceline with N points, as shown in Eq. 7

$$\underset{[\alpha_1 \dots \alpha_N]}{\text{minimize}} \quad \sum_{i=1}^N \kappa_i^2(t), \quad (7)$$

where κ_i is expressed as

$$\kappa_i = \frac{x'_i y''_i - y'_i x''_i}{(x_i'^2 + y_i'^2)^{\frac{3}{2}}}, \quad (8)$$

where x_i and y_i are the x-part and y-part of the spline. It is suggested for the team to test different methods and choose the one that gives the best results in terms of lap time. However, to use these method, it is needed to have an estimation of the boundaries of the track and the centerline. This estimation can be obtained using the nodes of the track's map. As long as the nodes are probably at the centerline of the track, the least squares method can be used in order to estimate this center line. After generating this center line, an internal and an external offset can be done based on the minimum track's width imposed by the competition rules.

The center line and the estimated boundaries can be seen in Figure 11. The boundaries generated pass through most of the cones and they are very close to the actual boundaries. In this case, the track has the minimum possible width, but in a wider track, the generated boundaries would still represent the minimum width, resulting in a more secure estimation. Another way to estimate the boundaries could be to create a curve, using the least squares method, passing close to the cones.

Figure 12 shows the minimum curvature path in the previous track, generated using the estimated boundaries.

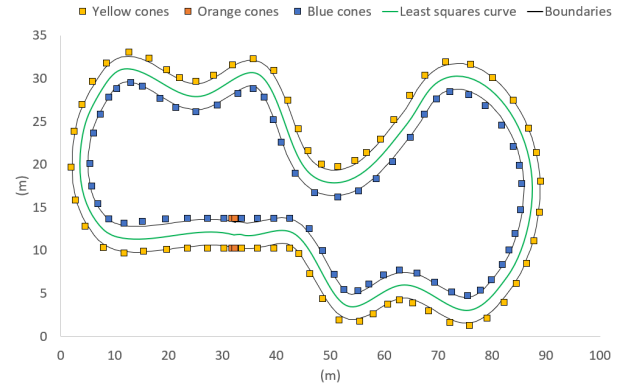


Figure 11. Example of the estimation of the track's boundaries. The yellow and blue squares mark the actual position of the cones. The green curve was obtained by the least squares method. The black lines are offsets of the green line.

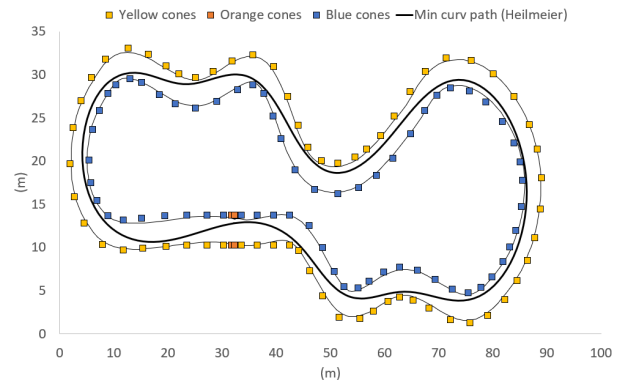


Figure 12. Example of optimal path generated using Heilmeier's method, represented by the black curve. The yellow and blue squares mark the actual position of the cones.

The minimum safe distance between the car and the track's boundary can be modified, according to the team's need.

4.5 Control

The control of the vehicle can be divided in two parts. During the exploration lap, it can be implemented a pure-pursuit controller in order to make the vehicle follow the chosen node of the map. This can be done until the vehicle find the initial node, meaning that it has completed the first lap.

For the fast laps, a PID controller can be used to find the error between the desired state and the actual state. The desired state can be obtained from the optimal path coordinates and the velocity and acceleration profiles, while the actual state is obtained from the State Estimation subsystem.

5. EXPERIMENTS AND RESULTS

The system proposed by this work is being used by Ampera Racing, a starter team from Federal University of Santa Catarina. Simulations were run following the system propose, using the recommended development tools.

Table 1. Maximum position error for different sensing configurations.

$\sigma_{GPS_{x,y}}^2$ (m^2)	GPS		IMU			E_{max} (m)
	$\sigma_{GPS_V}^2$ (m^2/s^2)	f (Hz)	σ_a^2 (m^2/s^4)	σ_θ^2 (rad^2/s^2)	f (Hz)	
0,5	0,5	1	5e-3	10e-5	100	1,77
0,5	0,5	1	1e-3	10e-5	100	1,21
0,5	0,5	1	5e-3	10e-5	200	1,68
0,5	0,5	1	1e-3	10e-5	200	0,87
0,5	0,5	10	5e-3	10e-5	100	1,25
0,5	0,5	10	1e-3	10e-5	100	0,78
0,5	0,5	10	5e-3	10e-5	200	1,11
0,5	0,5	10	1e-3	10e-5	200	0,49

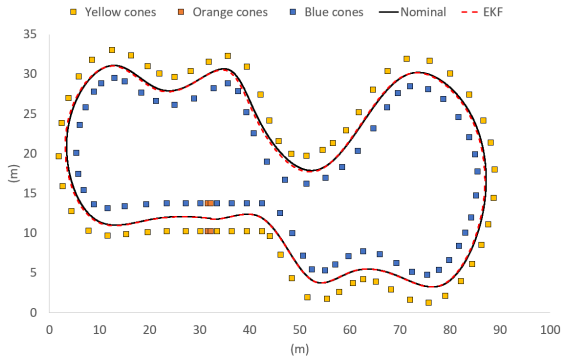


Figure 13. Position estimated by EKF. The true position is represented by the black line and the EKF estimated position by the red line.

For the state estimation, the sensors used were the suggested ones, GPS and IMU. In the experiments, the car was manually controlled on the track while the true and estimated position were measured. The variance of the sensors was decreased and the frequency increased until the maximum difference between the estimated position and the true position was less than 0.5m. As shown in Table 1, for the maximum error to be less than 0.5m, the GPS needed to have a frequency of 10Hz and a maximum variance of $0,5m^2$ for the position measurements, while the IMU had to have a frequency of 200Hz and maximum variances of $10^{-3}m^2/s^4$ for the acceleration measurements and $10^{-5}rad^2/s^2$ for the angular velocities measurements. The Figure 13 shows the true position of the car and the estimated position during one experiment.

For the exploration of the track, the team had two trained CNNs, called YOLOv4 *normal* and YOLOv4 *tiny*, in reference to the size of the CNN. Both of them were tested to define which one should be used for the next experiments. For the tests, the car was manually controlled along the track while the average FPS and the total classification errors were measured. As can be seen in Table 2, the YOLOv4 *tiny* had a much higher average FPS when compared to YOLOv4 *normal*. This FPS will surely decrease when experiments are done in a real prototype, but the YOLOv4 *tiny* will be better in terms of FPS nonetheless, so it was chosen to be implemented. The quantity of classification errors from both CNNs were very close to each other, but this problem was solved plotting in the map only blue cones detected on the left side of the image, and yellow cones detected on the right side of the image.

Table 2. Comparison of CNNs.

CNN	avg. FPS	Classification errors
YOLOv4 <i>normal</i>	32	4
YOLOv4 <i>tiny</i>	119	5

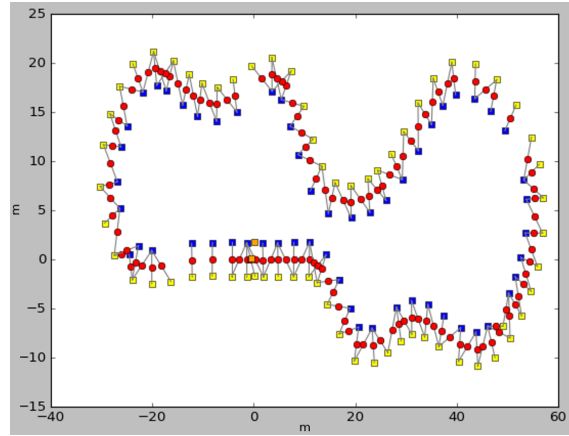


Figure 14. Complete track map.

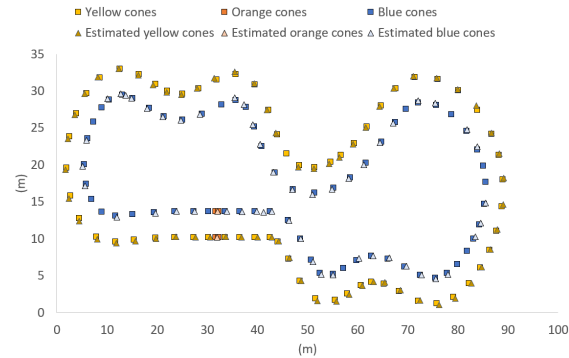


Figure 15. Estimated cones position. The true position is represented by squares and the estimated position by triangles.

By following the suggestions of this work and by creating good heuristics for the first lap exploration, the car was able to complete the lap in the simulations. A video of one simulation can be seen in

<https://youtu.be/2lnkiTIOLY>

where the car completed the track successfully. Figure 14 shows the complete track map, built by the system using the connecting lines and nodes. And Figure 15 shows both true and estimated position of the cones.

For the trajectory planning, the four methods cited by this paper, minimum path, minimum curvature and interpolation from Braghin et al. (2008) and minimum curvature from Heilmeier et al. (2019), were tested in five different tracks, which can be seen in Figure 16. The tracks have different sizes and forms, so it is possible to conclude which method is better in general. The results of the experiments can be seen in Table 3.

A vehicle can run the Heilmeier minimum curvature's path and Braghin's interpolated path approximately at the same time, and faster than Braghin's minimum length and minimum curvature. But Heilmeier's path generation

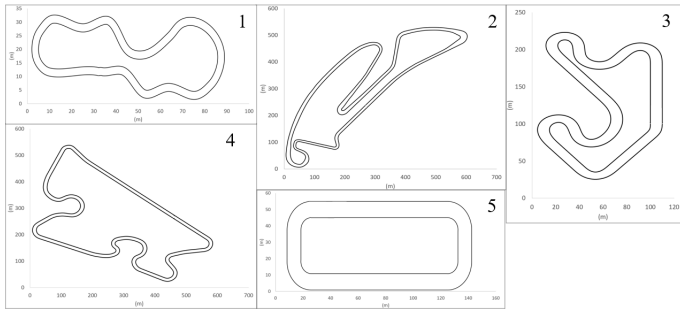


Figure 16. Tracks used in the experiments.

Table 3. Comparison of trajectories.

Track	Trajectory	Lap time (s)	Solver time (s)
1	Min. length	19,96	0,001
	Min. curv. (Braghin)	17,66	0,001
	Min. curv. (Heilmeier)	16,85	0,003
	Interpolation	16,90	5,246
2	Min. length	88,75	0,839
	Min. curv. (Braghin)	81,67	0,553
	Min. curv. (Heilmeier)	79,96	1,590
	Interpolation	79,47	191,325
3	Min. length	37,08	0,016
	Min. curv. (Braghin)	35,74	0,017
	Min. curv. (Heilmeier)	33,54	0,054
	Interpolation	33,39	16,728
4	Min. length	85,77	0,452
	Min. curv. (Braghin)	79,31	0,425
	Min. curv. (Heilmeier)	78,04	1,184
	Interpolation	77,80	79,394
5	Min. length	19,17	0,001
	Min. curv. (Braghin)	18,24	0,002
	Min. curv. (Heilmeier)	15,51	0,006
	Interpolation	15,18	14,212

is faster. While Braghin’s path needs several second to be calculated, Heilmeier’s needs milliseconds. Although Heilmeier’s path needs more time to be generated than the Braghin’s minimum length and curvature, the time saved in the laps compensate it. The car will run nine fast laps, so this saved time will be multiplied by nine. So it is recommended to use Heilmeier’s method to plan the track’s optimal trajectory. Finally, the control subsystem was not yet developed by the team.

6. CONCLUSION

The aim of this work was to propose a simple and low cost navigation system for starter teams with little knowledge in autonomous cars and a limited budget. Suggestions were made for the tools to be used in the development of the vehicle’s software and how to implement the navigation system, covering the following topics: Computer vision, State Estimation, SLAM, Path Planning and Control. The results of the proposed integration of algorithms, evaluated in the experiments were satisfactory, with a good perspective of increase of the performance.

This navigation system is currently being implemented by the Ampera Racing team at Federal University of Santa Catarina, Brazil. This team, which already competes in the electric class of Formula SAE Brazil, is an example of

a starter team that wishes to design its first autonomous vehicle for the FSD competition, having no great knowledge in this field and a limited budget.

Following the steps and recommendations written in this paper, and consulting the information in the sources cited, a team is able to develop and simulate its first prototype. Future works can evaluate the performance of the system in a real environment. This real scenario will contain more uncertainties and noises, and by doing experiments, the team can realize which areas need more attention and study, and the system can be improved and became more complex over time. During the development of new strategies, it is recommended to build the blocks of the system in a modular manner, so the team can work with the modules separately, improving time efficiency, and put them together afterwards.

REFERENCES

- Bader, M. and Hofmann, A. (2017). Design of an autonomous race car for the formula student driverless (fsd).
- Bhadani, R.K., Sprinkle, J., and Bunting, M. (2018). The CAT vehicle testbed: A simulator with hardware in the loop for autonomous vehicle applications. *Electronic Proceedings in Theoretical Computer Science*, 269, 32–47. doi:10.4204/eptcs.269.4. URL <https://doi.org/10.4204/eptcs.269.4>.
- Bochkovskiy, A., Wang, C.Y., and Liao, H.Y.M. (2020). Yolov4: Optimal speed and accuracy of object detection. ArXiv:2004.10934.
- Braghin, F., Cheli, F., Melzi, S., and Sabbioni, E. (2008). Race driver model. *Computers & Structures*, 86(13–14), 1503–1516. doi:10.1016/j.compstruc.2007.04.028. URL <http://dx.doi.org/10.1016/j.compstruc.2007.04.028>.
- Chen, T., Li, Z., He, Y., Xu, Z., Yan, Z., and Li, H. (2018). From perception to control: An autonomous driving system for a formula student driverless car.
- Dodel, D., Schötz, M., and Vödisch, N. (2021). Fsoco: The formula student objects in context dataset. ArXiv:2012.07139.
- Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M., and Lohmann, B. (2019). Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10), 1497–1527. doi:10.1080/00423114.2019.1631455. URL <http://dx.doi.org/10.1080/00423114.2019.1631455>.
- Kabzan, J., Valls, M.I., Reijgwart, V., and Hendriks, H.F.C. (2019). *Amz driverless: The full autonomous racing system*. arXiv:1905.05150.
- Montemerlo, M. and Thrun, S. (2007). *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*. Springer.
- Simon, D. (2006). *Optimal State Estimation*. John Wiley & Sons, Inc.
- Smith, R.C. and Cheeseman, P. (1986). Bayesian filtering: From kalman filters to particle filters, and beyond. *International journal of Robotics Research*, 5(4), 56–58.
- Xue, Z. and Schwartz, H. (2013). A comparison of several nonlinear filters for mobile robot pose estimation. In *Proceedings of 2013 IEEE International Conference on Robotics and Automation*.