University of Mumbai

Design and Implementation of control systems of an autonomous FSAE vehicle on an RC car

Submitted at the end of semester VII in partial fulfillment of requirements

For the degree of

Bachelors in Technology

by

Aliasgar Merchant

Roll No: 1815037

Bhushan Pawaskar

Roll No: 1815047

Mohommad Saud Shaikh

Roll No: 1815119

Guide

Prof. Atul Saraf

Batch 2018 -2022

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate

This is to certify that the dissertation report entitled **Design and Implementation of Control Systems of an autonomous FSAE vehicle on an RC car** submitted by Aliasgar Merchant, Bhushan Pawasakar, Saud Shaikh at the end of semester VII of LY B. Tech is a bona fide record for partial fulfillment of requirements for the degree of Bachelors in Technology in Mechanical Engineering of University of Mumbai

Prof. Atul Saraf

_____

Guide

Dr. Ramesh Lekurwale

_____

Head of the Department

Dr. Shubha Pandit

_____

Principal

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Certificate of Approval of Examiners

We certify that this dissertation report entitled **Design and Implementation of Control Systems of an autonomous FSAE vehicle on an RC car** is bonafide record of project work done by Aliasgar Merchant, Bhushan Pawasakar, Saud Shaikh during semester VII.

This project work is submitted at the end of semester VII in partial fulfillment of requirements for the degree of Bachelors in Technology in Electronics and Telecommunication Engineering of University of Mumbai.

_____

Internal Examiners

_____

External/Internal Examiners

Date:

Place: Mumbai-77

K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

DECLARATION

We declare that this written report submission represents the work done based on our and / or others' ideas with adequately cited and referenced the original source. We also declare that we have adhered to all principles of intellectual property, academic honesty and integrity as we have not misinterpreted or fabricated or falsified any idea/data/fact/source/original work/ matter in my submission.

We understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

Signature of the Student **Roll No.**	**Signature of the Student** **Roll No.**
Signature of the Student **Roll No.**	**Signature of the Student** **Roll No.**

Date: 22 December 2021

Place: Mumbai-77

Abstract

FSAE (Formula Student) is a student engineering design competition in which student teams design, manufacture, test and race their own formula style single seater racing car. FSG (Formula Student Germany) is an international competition held in Germany every year, where student teams compete from all over the world for the coveted crown of being the best formula student team. From the year 2022, FSG has introduced the driverless cup in which there are points assigned for the driverless events, failing to attend which the teams stand to lose valuable points in the event.

The competition has 4 different events for the driverless category. The events are acceleration, skidpad, autocross and trackdrive. In the acceleration event, the car has to accelerate straight in least time possible. In the skidpad event, the car needs to follow a figure eight track. In autocross and trackdrive the car has to self navigate across a specific path. In order to keep up with the competition, it was necessary for the team to implement autonomous driverless capabilities for the existing race car. Since this was the first time the capabilities were being implemented, the team set a goal to focus on the events of acceleration and skidpad.

The sub teams were divided into perception and sensing, localization and mapping, planning and controls. The controls team was tasked with implementing the control systems for the FSAE race car. Main responsibilities were designing the entire controls hardware for the vehicle and developing their relative control algorithms. The design and analysis for the required hardware was done successfully. The simulation of the control algorithms for the same was done on ROS inside of Linux. For testing the algorithms, a prototype RC car powered by Raspberry Pi was designed which would help validate our simulation results and sensor data.

***Key words*:** FSAE, Control Systems, Autonomous Vehicle, Testing of control algorithms. Actuator selection, Driverless vehicle

Contents

List of Figures

Nomenclature

BLDC	Brushless Direct Current
DV	Driverless
EBS	Emergency Braking System
ECU	Electronic Control Unit
FSAE	Formula Society of Automotive Engineers
FSDS	Formula Student Driverless Simulator
FSSIM	Formula Student Simulator
GPS	Global Positioning System
LIDAR	Light Detection and Ranging
MPC	Model Predictive Control
PID	Proportional Integration Differentiation
RES	Remote Emergency System
ROS	Robot Operating System
F_L	Force at lugs
F_T	Force component along tie rod
F_μ	Frictional force
θ	Angle between steering shaft and tie rod
R_P	Radius of pinion
R_A	Perpendicular distance between Ackermann mount point and steering axis
R_U	Perpendicular distance between contact point and steering axis

Chapter 1: Introduction

This chapter presents the introduction of what is formula student, what is driverless cup, format and events involved in it. It gives a background about the Formula Student Germany event. The chapter also mentions the motivation, scope behind the project and the structure of the report is also presented briefly.

1.1 Background

FSAE, short for Formula SAE, is a worldwide collegiate competition hosted by the Society of Automotive Engineers (SAE). Teams composed of undergraduate and graduate students design, build and compete for a formula-style race car in the FSAE series. The cars that compete here are either combustion or electric or even driverless with every category competing separately. More than a racing competition it is an engineering and project management competition. The event is broadly divided into three segments which are Technical Inspection, Static & Dynamic Events. The Technical Inspection event or the scrutineering event checks the rule compliance of the car, according to the rule book given by the competition. This also ensures that the car is safe enough for the driver in case of any accidents. The Statics and the dynamics event are where the teams score points to win the competition.

The Static event is further divided into 3 events which are Cost and manufacturing event, Design event & Business Plan Presentation. The Cost and manufacturing event is where the judges question our understanding and decision taken for our resource management and our knowledge about manufacturing. In the Design event judges who belong to industrial and racing backgrounds question us about our system understanding and our design trade-offs. The Business plan presentation provides us with the opportunity to make our own business model around our car and present them to the investors who have a business background.

Fig: 1 Official Formula Student Germany Logo

Teams from various universities compete in various competitions across the globe. Formula Student Germany (FSG) is one such event, an international competition in which teams from top universities participate from across the world. The event is held every year in Hockenheimring, Germany and is divided into electric and combustion categories. The teams participate in various dynamics events like the acceleration, skidpad, autocross and endurance events. These test the race-cars under different parameters like acceleration, vehicle dynamics, aerodynamics and so on.

From the year 2022 onwards, the competition has decided to add another challenge for the teams entering the competition. The teams would have to make a driverless vehicle that could navigate the track by itself. This would be a compulsory event, meaning that teams would lose points if they were to not participate in it. This is why it is important for the team to adapt and implement driverless capabilities in the vehicle in order to compete successfully in the competition.

1.2 Driverless Cup

FSG has introduced a driverless cup and has added mandatory points for some dynamic events in the regular category for driverless attempts of the same. Currently, there are 4 events which can be attempted in the driverless category of events:

1) DV Skid Pad
2) DV Acceleration
3) DV Autocross
4) Trackdrive

	CV & EV	DC
Static Events:		
Business Plan Presentation	75 points	-
Cost and Manufacturing	100 points	-
Engineering Design	150 points	150 points
Dynamic Events:		
Skid Pad	50 points	-
DV Skid Pad	75 points	75 points
Acceleration	50 points	-
DV Acceleration	75 points	75 points
Autocross	100 points	-
DV Autocross	-	100 points
Endurance	250 points	-
Efficiency	75 points	-
Trackdrive	-	200 points
Overall	1000 points	600 points

Fig: 2 Score table

Each of these events have a different task to be completed. In the DV Skid Pad event, the race-car has to navigate a figure eight circular track in the fastest time possible. Similarly in the DV Acceleration, it has to accelerate across a straight track in the quickest way. DV Autocross and Trackdrive, both have a set track through which the vehicle must plan its own path and make its way through in the quickest time possible. To aid the race-car in determining the path, the left side of the track will be marked with blue cones and the right track will be marked with yellow cones. To achieve these tasks, the race-car has to have multiple systems working in tandem together, from sensing, perception, localization, mapping to planning and control.

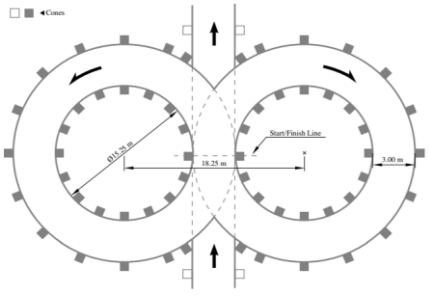


Fig: 3 Skidpad Track layout

Currently, top teams in the competition achieve speeds of more than 60kmph while racing on the track and some teams even perform better than a professional race-car driver due to the immense real time data available at disposal to the driverless computing unit.

Fig: 4 Cones used in competition

1.3 Motivation

The Control System is the vital subsystem for a driverless vehicle. It is responsible for deciding where the actual vehicle will move towards while also ensuring safety by braking in case of emergency. Optimal tuning of the control parameters is essential for a safe smooth functioning of the vehicle controller. The control system also significantly affects the vehicle dynamics and in turn the performance of the vehicle on track. Ensuring a reliable control system which would be safe and pass inspection was the major goal of this project. Getting the mandatory event points for FSG driverless was important, but moreover the immense experience that the team would get by switching to a driverless vehicle was crucial for the team to compete in the competition in future. Doing this from scratch requires lots of data of previously successful attempts and also lots of research into the works of other teams. Thus, in this project we aim to develop the control systems which could be used on an FSAE vehicle from scratch and eventually test and implement it on an actual vehicle in the future.

1.4 Scope

To achieve the goal of implementing control systems on FSAE vehicles, we need to integrate it into the main car such that it is compatible with the subsystems of the car. This means designing the hardware and actuators required as per the steering and pedal geometry. We try to optimize the design so that it is not over designed a lot, however being our first time with this project, having no references we have to keep a certain margin of safety. Vehicle performance parameters were simulated using IPG carmaker and dynamic performance of the vehicle was evaluated based on which suspension and chassis geometry was finalised. The integration was done inside of solidworks based on inputs obtained from other systems of the vehicle. Calculations for hardware selection were done based on these geometry relations. While coding the control algorithms, we simulated and tested those on software in ROS based simulators like FSSIM and FSDS.

1.5 Organization of Report

This report gives a detailed explanation on the design integration for control systems hardware and simulation of its corresponding control algorithms. The first chapter gives an introduction about the FSAE competition and also gives a small intro about the driverless system along with the motivation and reasoning behind the implementing an autonomous system. In the second chapter we give a summary about the research papers and books that we had referred to. The third chapter explains the design procedure, CAD model and important concepts related to hardware components and control algorithms. The fourth chapter is where we show selection criteria and calculations done for the actuators and sensors. Also we show the simulation and code behind the algorithms. In the last chapter we conclude our report and suggest potential future developments and testing that we can do to implement it on the main vehicle.

Chapter 2: Literature Survey

This chapter gives the summary of some of the literature or books that we had referred to. Research papers from FSAE teams were studied and their designing and testing methods are summarized below. The algorithms the teams used for the control systems of their vehicle were studied. These references have been mentioned in the following chapter.

Myung-Wook Park et al. [1] utilized an adaptive pure pursuit algorithm to control the lateral motion of a commercial SUV. Their idea differs from the traditional controller because it takes into account the tracking error generated by large look-ahead distances which cause a fast moving vehicle to cut corners at curved paths. To reduce the tracking error, they employed a PI (Proportional-Integral) gain controller. Using the proposed method, path tracking error is reduced more than the original method at the curved path by 1.5m. However, it is reduced only by 0.3m at a low curvature path

In [2], Hiroki Ohta et al, used simulations and on-field testing to determine the problems in the traditional pure pursuit controller and have thus suggested changes for the same. They found three main problems, the meandering of the vehicle i.e. overshoot and oscillation of the vehicle along the path, cutting corners at curves and unstable steering commands. The look-ahead distance is modified dynamically relative to the velocity. To prevent divergence of the control owing to tracking too near target waypoints, the minimum Lookahead Distance is also parameterized. The linear interpolation approach made the continuous acquisition of the target point from the path possible.

Dhanya S Lal et al. [3] tested the effectiveness of the Pure Pursuit algorithm on paths of different curvatures. The authors explain the linear and non-linear vehicle models and mention the motion equations for the same. The simulation testing is carried out on OCTAVE using both linear and non-linear vehicle models. For a straight track with length of 100m with a linear vehicle model at a velocity of 8m/s, it is observed that the crosstrack error reduces for 1m followed by a constant error for the next 2m followed by reduction in crosstrack error until it becomes 0 at 10m. For a non-linear model, the crosstrack error decreases with distance travelled until it becomes 0 at 12m. For a circular path, the velocity was kept constant at 5 m/s while the steering angle is constrained to be within ± 45° and the steering angle rate is constrained to ± 2.5°. The average cross-track error was found to be 0.88311 m.

In [4], Wei-Jen Wang et al. proposed an advanced pure pursuit algorithm with 2-degree polynomial function in the vehicle velocity and the shortest distance between the reference trajectory and the current vehicle position. Their proposed model takes into account not only the dynamic velocity but also the distance between the tracking path and the ideal path for the calculation of the look-ahead distance. This algorithm calibrates the heading and steering angle of the vehicle and reduces lateral error when the vehicle tracks the ideal path. The analysis has been done for the rectangle paths, including the straight paths and the algorithm reduces the lateral error from 2.65m to 1.2m. Results show that the algorithm is more accurate than the traditional algorithm by improving 54.54%.

Hua Wang et al. [5] propose a path following algorithm based on two fuzzy controllers to adjust the velocity and foresight distance, aiming at the influence of driving velocity of autonomous vehicles and the foresight distance on the trajectory tracking control in pure pursuit mode. The two fuzzy controllers are used in order to optimize the traditional pure pursuit algorithm and improve the tracking performance of the whole system. The results showed that the proposed algorithm can adaptively adjust vehicle's parameters on the corners to track different reference paths quickly and effectively. The average tracking error is 0.042m, and the maximum tracking error is 0.102m on the curvature whereas, the average tracking error is 0.026m and the maximum tracking error is 0.043m on the linear path.

Chapter 3: Fundamental Concepts

The chapter introduces the levels of autonomy involved in autonomous cars. It also talks about the different subsystems of an autonomous cars, about the design flow of control systems for an FSAE car. Component Selection and control algorithms have also been discussed in this chapter

3.1 Working of autonomous vehicle

Autonomous vehicles can be broadly classified into the following categories based on the level of autonomy. These levels go from level 0 to level 5, the last being completely autonomous. Currently, there are some vehicles that achieve level 4 autonomy but still require some emergency intervention in some situations.

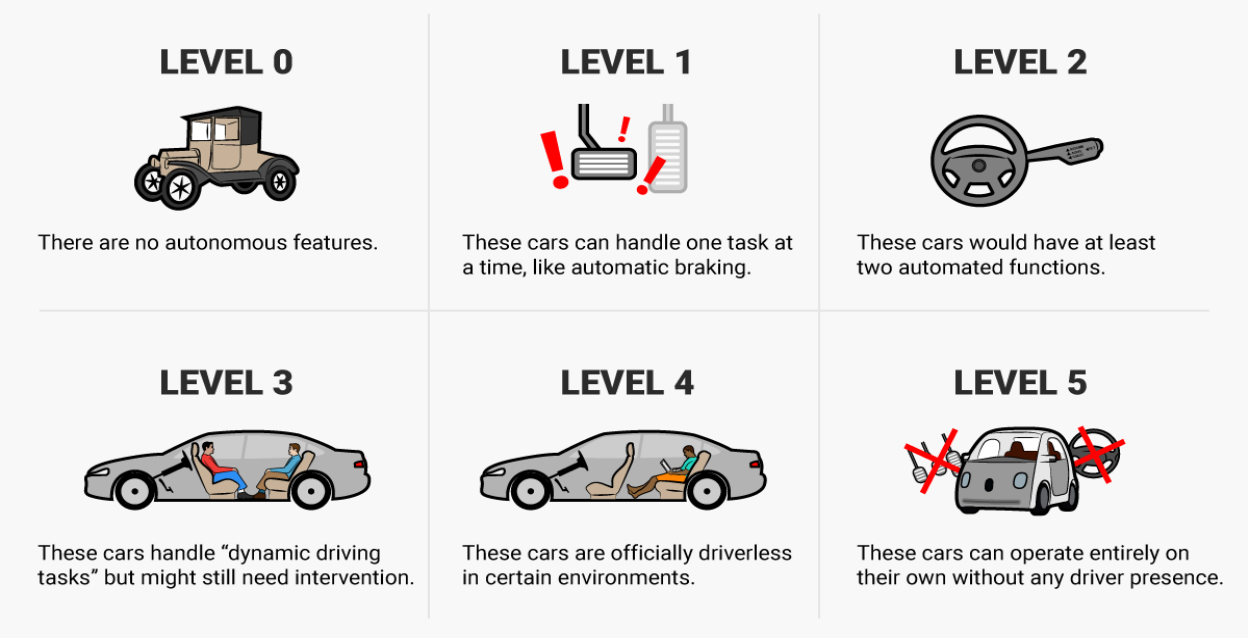


Fig: 5 Levels of Automation

Level 5 vehicles have to tackle multiple challenges like avoid other vehicles on the road, adjust to other human drivers who might behave unexpectedly, react to road signs, signals, trees and other objects on the roads and also account for humans and animals on the road. Then, they need to make a smart decision based on all of these in order to navigate autonomously.

Formula Student on the other hand does not have these unexpected conditions. The system is quite controlled. There are no pedestrians, objects which should not be there. The car has to successfully navigate a track with cones around it to help make it easy to determine the path. Some other events like acceleration and skidpad are even more straightforward in that the track layout is fixed.

In order for a vehicle to navigate the track autonomously, it has to go through a long pipeline of commands. These can be summarised in the flowchart below.

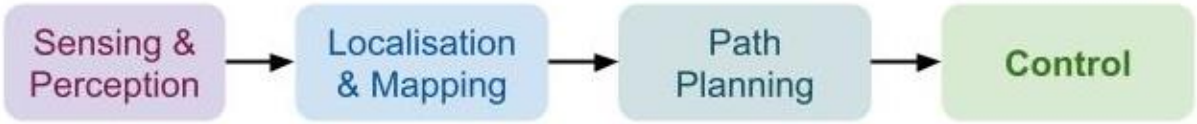


Fig: 6 Autonomous control flowchart

1. Sensing & Perception

The race-car first needs to sense the surroundings. This is done with the help of various sensors mounted on the vehicle. These could include sensors like LIDAR or RADAR or even camera sensors which collect raw pixel data. Other sensors used to collect system data are GPS sensors which give position data and accelerometer sensors which give velocity, acceleration data. This is the first step and involves collecting the data from the environment which could be processed later to get meaningful information from the raw data.

2. Localisation & Mapping

The collected data is then processed and a map is made of the system which can be used for decision making. This is made by state of the art algorithms. Another process that goes hand in hand with mapping the environment is determining the local location of the agent with respect to the system and constantly updating it for real time decision making.

3. Path Planning

After making a map of the system, feasible paths are decided based on certain predetermined parameters. This process requires the system to consider other parameters that might go into optimisation of the paths. Based on the chosen path, a trajectory to follow is then determined.

4. Control

Controls is the last subsystem in the process which has to actuate servo motors or other linear actuators and actually bring about the change in the system to help it move from one state to another state. It also has to take into consideration the feedback from its output and incorporate that back into the loop to make a closed loop system. This helps account for inaccuracies in practical implementation of the control systems.

3.2 Control System

As mentioned before, the control system is responsible for actual actuation of the servo motors and linear actuators present in the system. It also decides on an optimal algorithm for the same based on certain objectives to be achieved by the system. Control systems can be broadly classified into two types:

1. Open Loop Control Systems

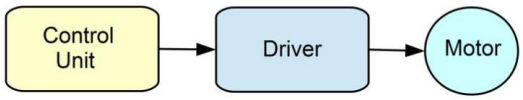


Fig: 7 Open Loop Control System

In this kind of control system, the output doesn't change the action of the control system. The accuracy of this system is low and less dependable. It is very simple, needs low maintenance, has quick operation, and is cost-effective. It doesn't have any feedback.

2. Closed Loop Control Systems

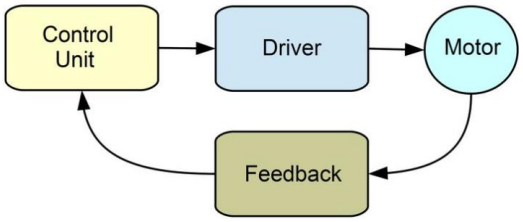


Fig: 8 Closed loop control system

The closed-loop control system can be defined as the output of the system that depends on the input of the system. This control system has one or more feedback loops among its input & output. This system provides the required output by evaluating its input. This kind of system produces the error signal and it is the main disparity between the output and input of the system. The main advantages of the closed-loop control system are that it is accurate and reliable but it is expensive, and requires high maintenance.

With respect to the formula student competition, the control system has to actuate the servo motors for the steering and the linear actuators for the pedals which in turn actuate the input to the motor controller and brakes system respectively. So there are a total of 3 commands which can be given to the controller in order to control the vehicle:

1. Accelerate
2. Brake
3. Steer (left or right)

3.3 Design Flow

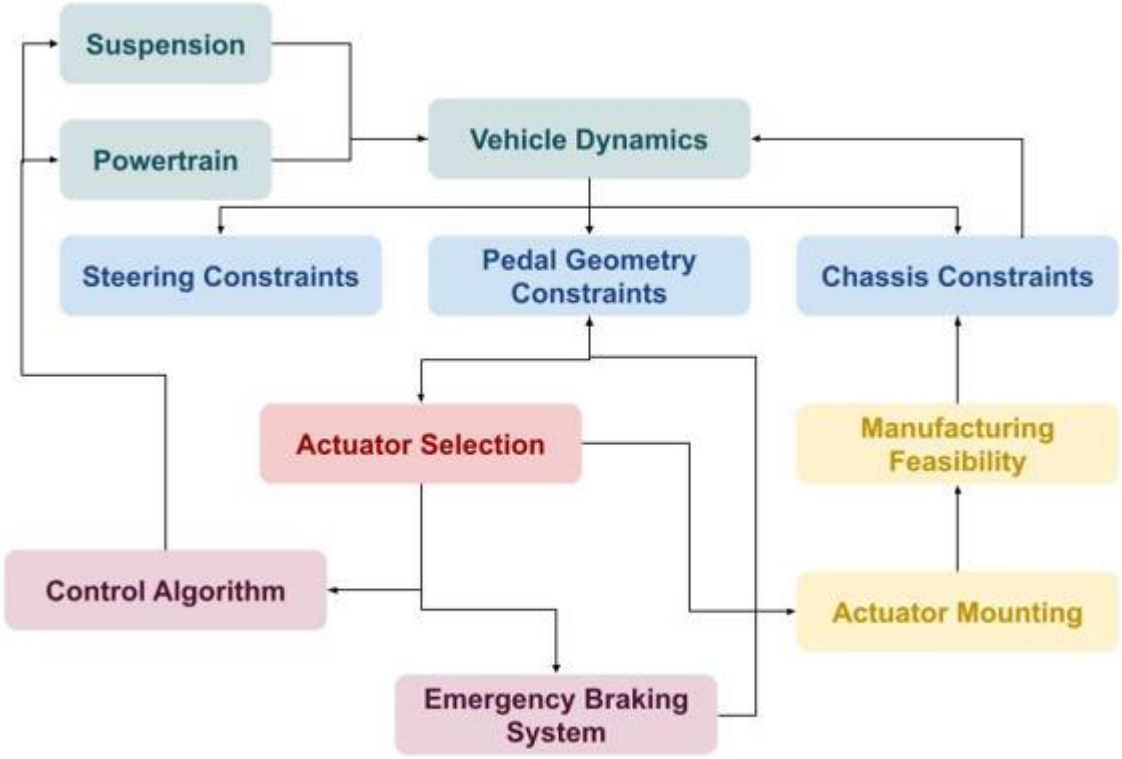


Fig: 9 Design Flow

- While designing, vehicle performance is of utmost importance. It is most affected by vehicle dynamics.
- While iterating the design, performance of the vehicle is checked for different iterations to optimize the performance of the vehicle based on different parameters.
- These parameters are most influenced by suspension and powertrain of the vehicle.
- The vehicle dynamics influences the chassis, ergonomics, pedal geometry and steering geometry of the vehicle.
- These in turn determine the actuators and sensors to be selected.
- The mounting for the same needs to be made based on it and would need to be checked for manufacturing feasibility which might again change chassis design.
- The actuators also influence the control algorithm and emergency braking system which again influences pedal geometry.
- The control algorithm influences how the vehicle behaves under dynamic conditions.

3.4 Block CAD of the racecar

Pedal Geometry

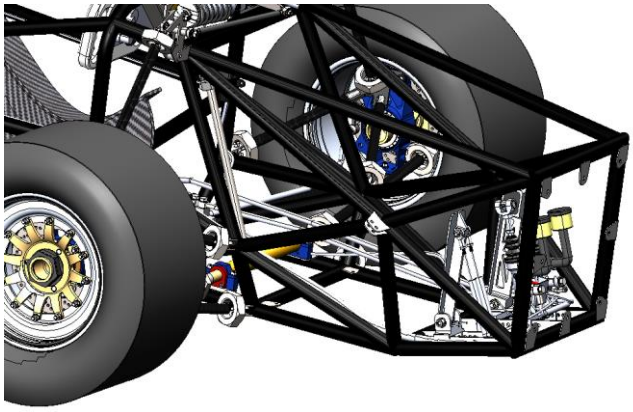


Fig: 10 Pedal Geometry

The braking subteam designs the pedal geometry of the racecar. This is influenced by the following factors:

- Ergonomics of the driver
- Front Space available
- Brake lines routing

This becomes the factor which influences the design of the linear actuator we use to measure the pedal input. It is mapped based on the angle of pedal, the actuator travel and the output to send.

Steering Column

The steering subteam decides on the type of steering type to use (ex. Ackermann vs Davis steering) They have to take a number of considerations regarding the links and mechanisms to use. The design for the previous year had a UV joint for angular links. It was replaced by a bevel gear so that a motor could be integrated for autonomous control of the steering system.

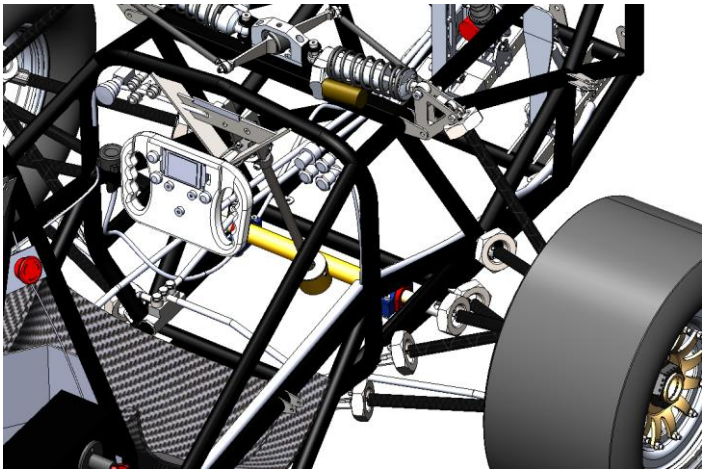


Fig: 11 Steering Column

3.5 Component Selection:

To successfully achieve our task, we required sensors that would provide accurate data to the control algorithms to perform the necessary computations needed and provide the required steering, throttle and braking commands that would be sent to the ECU.

Next the ECU would then send these commands to the actuators which would then actuate the steering, throttle and braking mechanisms to move the vehicle in the required direction.

To actuate the steering mechanism, the team had two options:

1. Directly actuate the rack and pinion mechanism: Upon considering the idea and discussions within the team, we found the idea lacking. This was because the actuators available for linear motion could only move in the forward direction. Backward motion was not possible when current supply was removed and the back-driving force required to pull back the linear actuator would be massive and that would not be feasible for the driver.

2. Control the steering column: This idea seemed feasible as we had an option of obtaining a BLDC motor as it could be back-drivable and in cases this was not possible the team could create a mechanism that could detach the servo-motor.

To actuate the brake and accelerator pedals, the team had to take into account the back-driving force that would be required to be applied by the driver if the actuators remained coupled to the pedals during the manual runs. This was accompanied by factors like the retraction speed and time of the actuator as well as space available behind the pedal. We needed high speed linear actuators as multiple acceleration and braking commands would be sent to the actuator depending on how well the low level PID controller could match up with the required speed. However, we also needed to ensure smooth movements and no jerk.

Additionally, to make the vehicle rule compliant, we have to incorporate an Emergency Braking System (EBS) into our design. According to the competition rules, the EBS system must activate the braking of the vehicle in cases of electric failure or when the RES (Remote Emergency System) is triggered.

To ensure that the control systems would perform the necessary calculations, the team would also require accurate reading from sensors like GPS, Wheel speed sensors and fuse them together through the use of the Localization module to obtain the data like the position of the vehicle on the map, the velocity of the vehicle.

To move around the track autonomously, the vehicle requires waypoints that it has to follow to be sent. This can be done either by:

1. Using a pre-built track
2. Make use of the perception pipeline along with the mapping and Path planning pipelines. This combination is majorly used when we have an unknown track that the vehicle has to traverse.

3.6 Control Algorithms

PID

The PID control algorithm is used for longitudinal control. The term PID stands for Proportional, Integral and Derivative. This algorithm works using a feedback mechanism in which it calculates an error term and applies a correction factor.

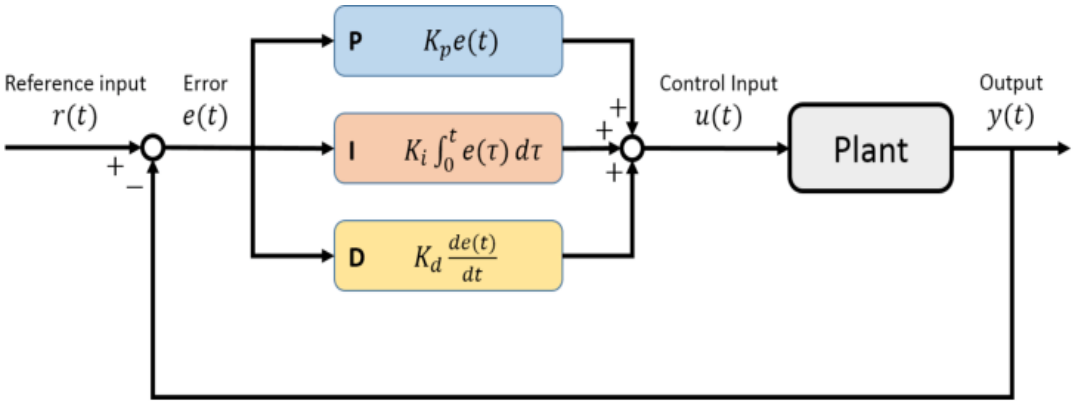


Fig: 12 PID control Algorithm

Block Diagram of a PID Controller

Pure-Pursuit

Pure-Pursuit algorithm is used for lateral control. The Pure-Pursuit control algorithm involves the use of a reference point at some distance in front of the vehicle and calculates an appropriate steering command.

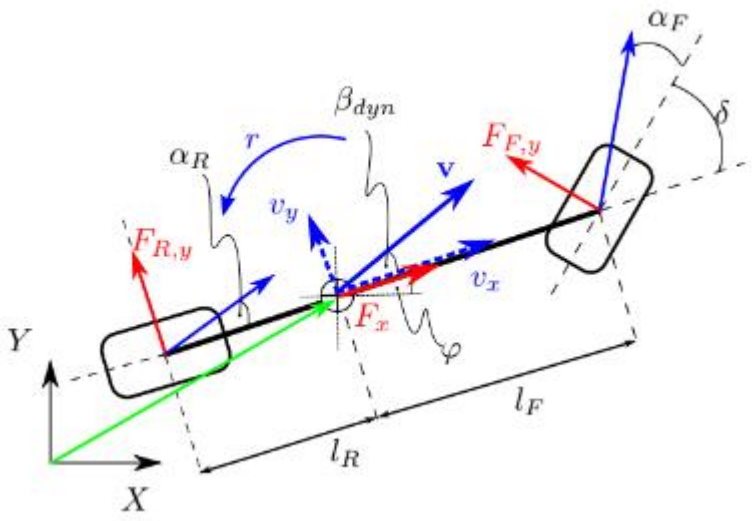
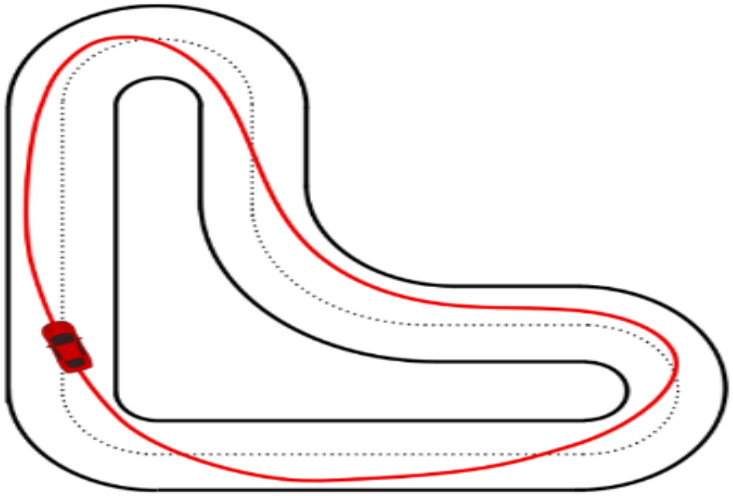


Fig: 13 Kinematic Bicycle Model used in Pure Pursuit

MPC Model based Predictive Control

MPC solves an optimization algorithm to find the optimal control action that drives the predicted output to the reference trajectory while satisfying a set of constraints. MPC is used for both longitudinal and lateral control.



$$\min_x f(x)$$

Raceline curvature

$$\text{s.t.} \quad g_1(x) \leq 0$$

Staying within the track bounds

$$g_2(x) = 0$$

Curvature as a function of tangential distance

Fig: 14 Problem formulation for MPC

Chapter 4: Implementation

This chapter talks about the hand calculations, analysis involved related to the actuators. It also mentions the specifications of the sensors and actuators, their mounting. It briefs about the simulators used for simulating the algorithms.

4.1 Analysis/Calculations for hardware

4.1.1 Steering motor

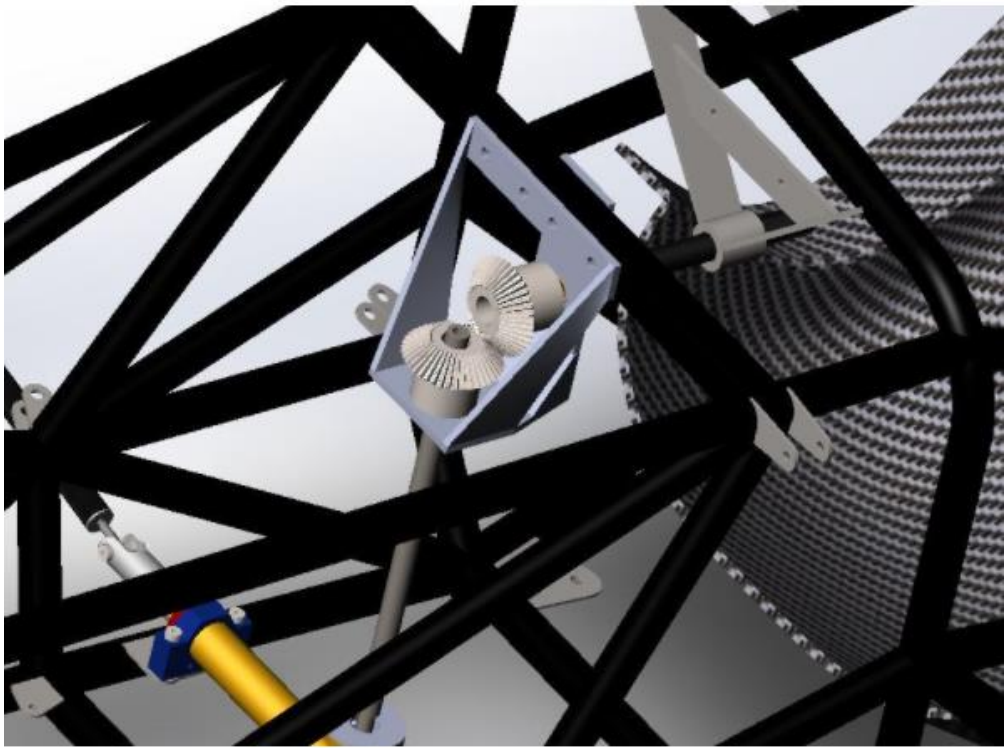


Fig: 15 Steering Motor

For the purpose of steering the vehicle a rack and pinion setup was made, with the steering wheel and pinion being connected via bevel gear. The gear ratio being 1:1 due to manufacturing and speed requirements.

The steering torque at static conditions was found to be 23Nm via experimental testing. This with a 15 rpm speed was required from the motor to be able to properly take corners at high speeds. For this purpose a BLDC motor has been chosen due to its backdrivability as required by the rules. There is an attached gearbox with the motor so as to meet the torque requirements while maintaining a compact size. As of now a suitable motor is still being searched for.

$\boldsymbol{\tau}_{st} = F_L \times R_P$

$F_T = F_L \cos\theta$

$F_T \times R_A \times \sin \alpha = F_\mu \times R_\mu$

Where,

R_P = Radius of pinion

R_A = Perpendicular distance between ackermann mount point and steering axis

R_U = Perpendicular distance between contact point and steering axis

F_T = Force component along tie rod

F_L = Force at lugs

F_μ = Frictional force

θ = Angle between steering shaft and tie rod

$\theta = 9.19°$ mg = 274 x 9.8N

$\alpha = 71.75°$ $\boldsymbol{\mu} = 1$

R_A = 73.25mm

R_U = 27.71mm

$F_T \times R_A \times \sin \alpha = F_\mu \times R_\mu$

$F_T = (F_\mu \times R_\mu) / (R_A \times \sin \alpha)$

= (274 x 9.8 x 27.71) / (4 x 73.25 x sin(71.75))

= 267.4N

$F_T = F_L \cos \theta$

= 267.4 / cos(9.19)

= 270.87N

$\tau = 2 \times F_L \times R_p$

= 2 x 270.87 x 20

= 10.82Nm

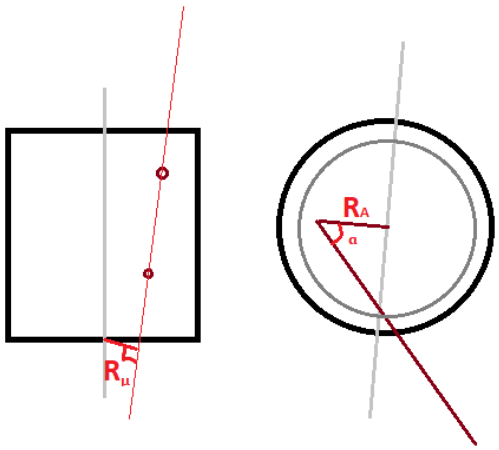


Fig: 16 Wheel Model for steering torque calculations Top View and Side View

4.1.2 Linear Actuator

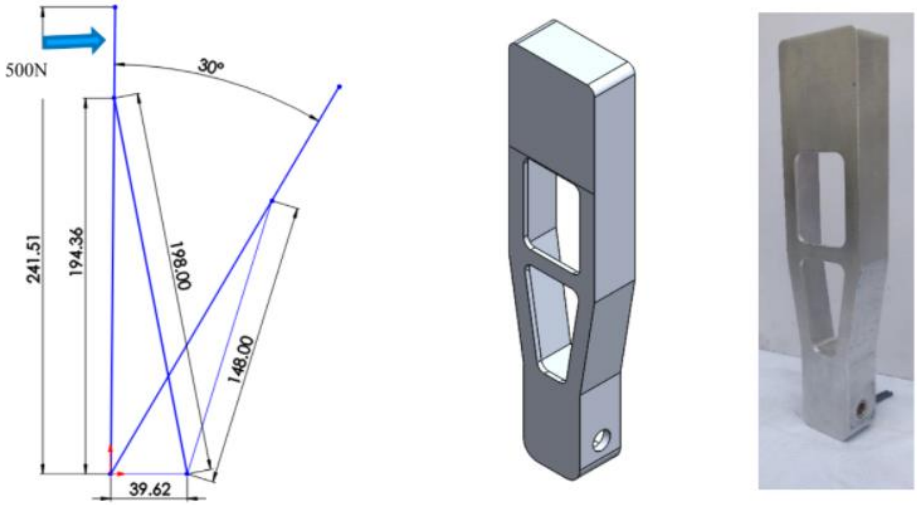


Fig: 17 Pedal geometry

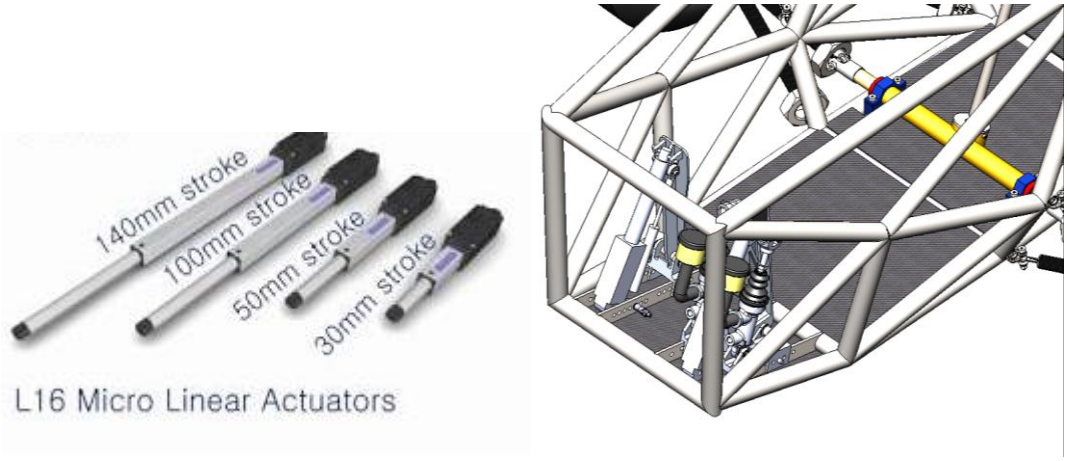


Fig: 18 Linear actuator on car

Driverless braking is made possible using a linear actuator. Many models were referred but the one that fit the specifications was the Actuonix L-16 P linear actuator. Its force and back drivability will be able to actuate the braking system on the track and it also follows all the guidelines specified by the rulebook.

Specifications : -

Stroke :- 100 mm

Force :- 200 N

Gear ratio :- 150:1

Voltage : - 12 V

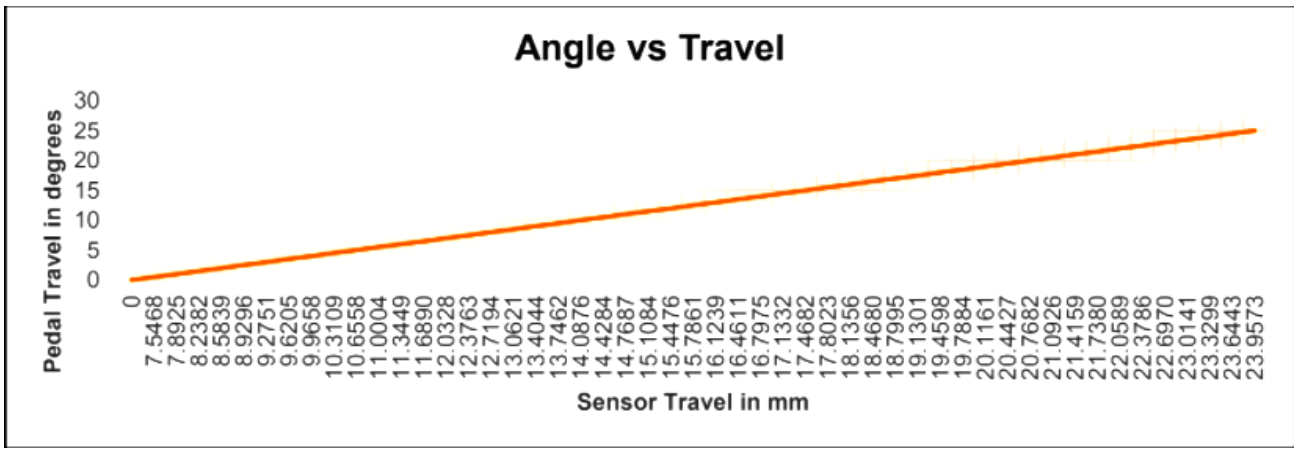


Fig: 19 Angle vs pedal travel

Basic cosine rule is applied to the pedal geometry to obtain the pedal travel and the actuating length of the APPS sensor. However, considering the packaging constraint and weight constraint the sensor travel was kept to be around 30 mm. Which was being simulated on Solidworks using Motion Study tool and the values were verified.

4.1.3 LiDAR Mount

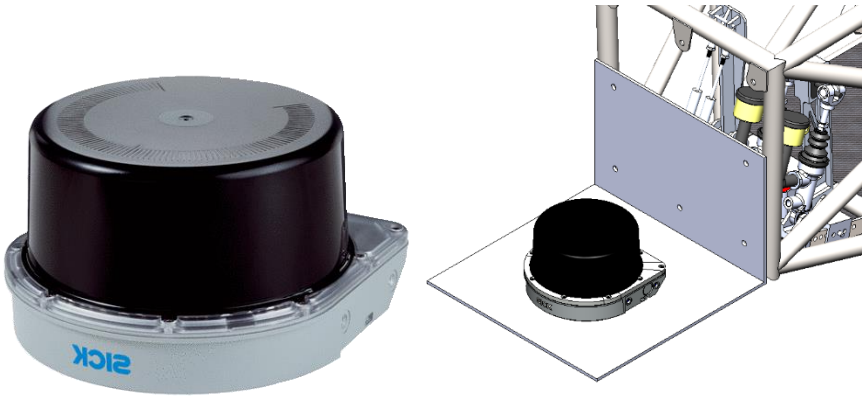


Fig: 20 LiDAR sensor and mount

The LiDAR needed to be on a flat surface and in front of the car so that the ground reference was stable and unobstructed point clouds were obtained. Using common mounting points for AIP, an MS plate was designed for mounting the LiDAR sensor.

4.1.3 GPS and Speed sensor

Fig: 21 SBG inertial sensor

SBG Systems offers a complete line of inertial sensors based on the state-of-the-art MEMS technology, such as Attitude and Heading Reference System (AHRS), Inertial Measurement Unit (IMU), and embedded GPS Inertial Navigation System (INS/GPS). Combined with cutting-edge calibration techniques and advanced embedded algorithms, our inertial sensors are ideal solutions for unmanned vehicle control.

Due to its position being of key importance for proper yaw, pitch and roll rates, the sensor was to be placed roughly near the CG of the vehicle, which comes to be on the slanted face of the battery pack.

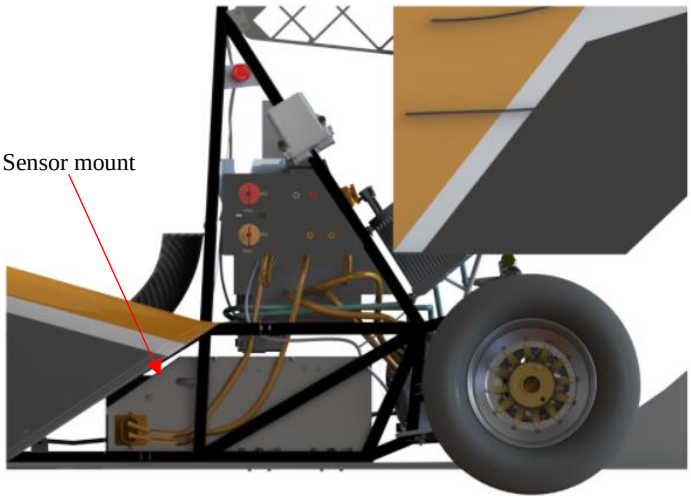


Fig: 22 SBG Sensor mounting point

4.2 Code and Simulation

4.2.1 Simulations

FSSIM

FSSIM is a vehicle simulator dedicated for Formula Student Driverless Competition. It was developed for autonomous software testing purposes by AMZ racing. This simulator is developed and tested on Ubuntu 18.04 and ROS Kinetic and both were installed. With a few modifications, we were able to run it on ROS melodic as well. But the main drawback of this simulator was that there were no sensors available on the car model so we could not obtain sensor data for further processing. Due to this drawback, we use this simulator only for the testing of the Control algorithms.

FSDS

We have also shortlisted the Formula Student Driverless Simulator (FSDS) from various simulators available. FSDS was originally developed by Formula Student Team Delft, MIT Driverless and FSEast. FSDS is a community project with a goal to provide an end-to-end simulation for FS Driverless teams. It simulates all commonly used sensors and is compatible with ROS. Autonomous systems connect to the simulation using ROS topics provided by the ros-bridge. A separate component - the operator - will provide an interface to control the simulation during competition. But one of the drawbacks of this simulator is that it is computationally expensive. We use this simulator to run the Perception, State Estimation, Mapping and Planning module. Therefore, we are using this simulator currently and plan to use this for final testing.

We have implemented PID and Pure-Pursuit algorithms on the FSSIM simulator. Due to some of the drawbacks of FSSIM mentioned previously, we tested our control algorithm on FSDS which allowed us to identify an issue with synchronization between throttle and steering commands which was then corrected.

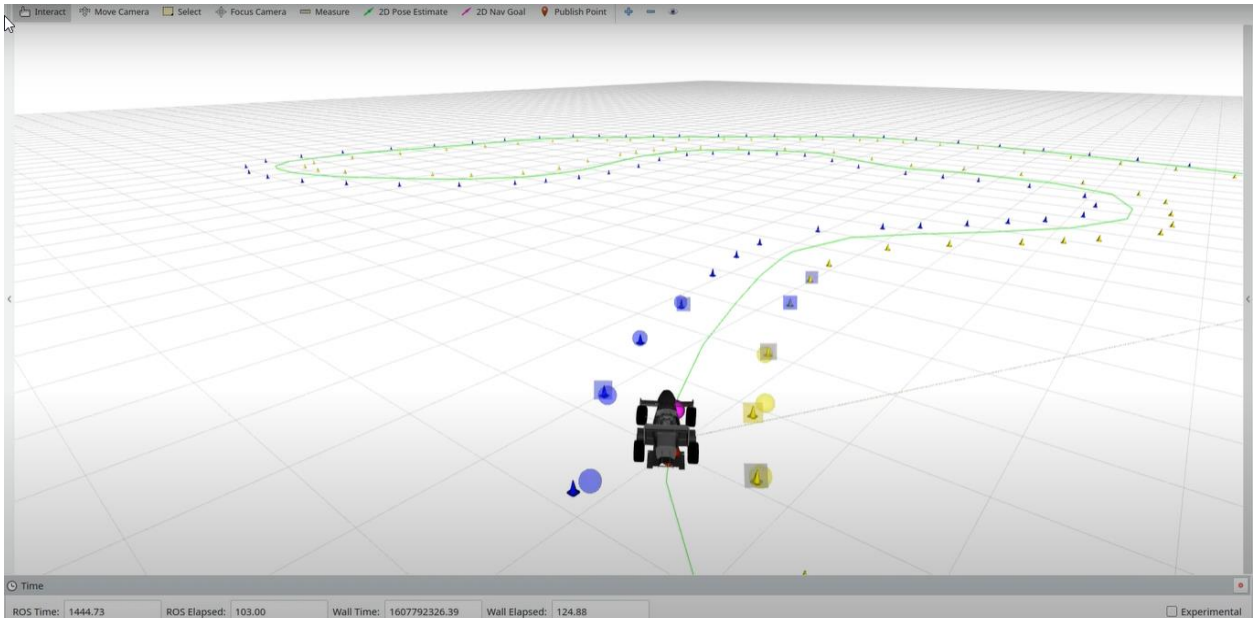


Fig: 23 Pure Pursuit, PID controller implementation on FSSIM

Fig: 24 Pure Pursuit, PID controller implementation On FSDS

https://drive.google.com/file/d/1jKTN7Gvp8moVjb5G9_gmLdyyAqbA2Iin/view?usp=sharing

https://drive.google.com/file/d/12VtwTAKu85L0tb_RZFXB1uAPyvniA8zm/view?usp=sharing
https://drive.google.com/file/d/1DfaSPHhkYkjm1cdjUjqoJb10eBCH_MAH/view?usp=sharing
https://drive.google.com/file/d/1dUBhQEdIIK7uW-esYG1oNgtqGrwFuuaS/view?usp=sharing

Chapter 5: Conclusions and Further Work

This chapter will conclude the work done with respect to the implementation of control systems for an FS vehicle in brief. It will also talk about work left to be done like testing of the algorithms on actual physical model to verify the simulation results

5.1 Conclusion

In this report the design procedure along with how the components were selected for the design was explained. With that procedure we had finalized a BLDC motor for steering actuation. We also finalized Actuonix L16-P as the linear actuator for the design. A block CAD for the race car was used and the hardware was then successfully integrated in the main CAD. Analysis for the same was also carried out. Mounting of the hardware components was also designed. Based on the selections made for the hardware, control algorithms like PID, Pure Pursuit, were developed and tested. Testing of the algorithms was successfully done in ROS based FSDS and FSSim simulators. Planning and trajectory generation algorithms like Vornoi and Djikstra were implemented successfully on the simulator.

5.2 Future Work

5.2.1 Testing of Algorithms

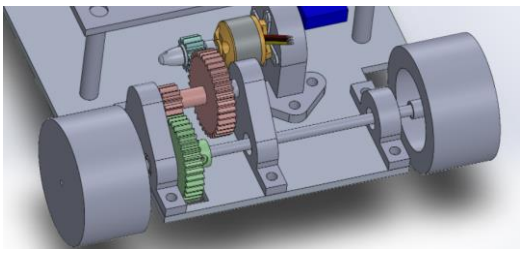


Fig: 25 Model RC powertrain

In order to test the simulated algorithms, we plan on developing a small scale prototype RC car which would be controlled by a raspberry pi. It would have a steering actuation using basic servos and will be accelerated using a bldc motor. The parts would be 3d printed using fused deposition modelling of plastic PLA. The work on it is underway currently.

Doing this will help us verify if the simulated algorithms actually work the same way. We would also need to better incorporate physical inaccuracies and account for them in the control algorithm to correct for them.

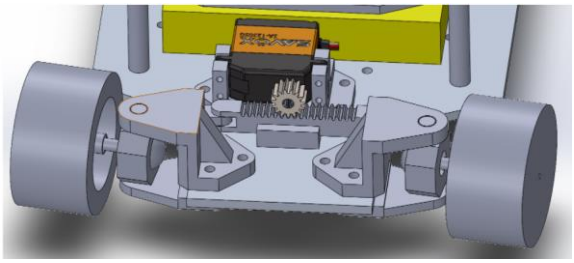


Fig: 26 Model RC Steering System

Our current plan is to also test our sensors and see if real time sensing and mapping algorithms work properly such that the vehicle is able to create a localised map of the environment.

5.2.2 Testing of Hardware

Fig: 27 Linear actuator and Lidar

Servos and linear actuators that will be ordered need to be tested and integrated with the electronics system in order to control them effectively. The specifications of these would need to be verified if they are actually as per the datasheet. Mounting for the same would need to be adjusted and tested if it is feasible with the actual hardware.

REFERENCES:

[1] -> Development of Lateral Control System for Autonomous Vehicle Based on Adaptive Pure Pursuit Algorithm, Myung-Wook Park and Sang-Woo Lee and Woo-Yong Han at 2014 14th International Conference on Control, Automation and Systems (ICCAS 2014) Oct. 22-25, 2014 in KINTEX, Gyeonggi-do, Korea

[2] -> Pure Pursuit Revisited: Field Testing of Autonomous Vehicles in Urban Areas, Hiroki Ohta, Naoki Akai, Eijiro Takeuchi, Shinpei Kato and Masato Edahiro, 2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications

[3] -> Lateral Control of an Autonomous Vehicle Based on Pure Pursuit Algorithm, Dhanya S Lal, Vivek A and Dr. Gopinath Selvaraj, 2017 IEEE International Conference on Technological Advancements in Power and Energy (TAP Energy)

[4] -> The Improved Pure Pursuit Algorithm for Autonomous Driving Advanced System, Wei-Jen Wang, Tusng-Ming Hsu, Tzu-Sung Wu, 2017 IEEE 10th International Workshop on Computational Intelligence and Applications November 11-12, 2017, Hiroshima, Japan

[5] -> Trajectory Tracking and Speed Control of Cleaning Vehicle Based on Improved Pure Pursuit Algorithm, Hua Wang , Xi Chen , Yu Chen , Baoming Li , Zhonghua Miao, Proceedings of the 38th Chinese Control Conference July 27-30, 2019, Guangzhou, China

Appendix A

Code Snippets

Pure pursuit main code

```
#! /usr/bin/env python

import rospy
import sys
import argparse
from geometry_msgs.msg import Polygon
from pure_pursuit_controller import PurePursuitController
import time
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def main():
    print("hello main")

    rospy.init_node('control_pure_pursuit_node')
    controller = PurePursuitController()
    # parser = argparse.ArgumentParser(description = 'Rate')
    # parser.add_argument('--node_rate', default = 10, type = int, required = True, help = 'Add node Rate')
    # args = parser.parse_args(['',join(sys.argv[1:])])
    rate = rospy.Rate(controller.getNodeRate())
    center_line = Polygon()
    while not rospy.is_shutdown():
        controller.run()
        # ani = FuncAnimation(controller.fig, controller.update_plot, init_func=controller.plot_init)
        plt.scatter(x=np.array(controller.x_data), y=np.array(controller.y_data),s = 1.0, c=np.array(controller.color), cmap='hsv',vmin=0,vmax=1)
        plt.ion()
        plt.show()
        plt.draw()
        plt.pause(0.001)
        rate.sleep()


if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass
```

Pure Pursuit Controller code

```
import math
import rospy
import argparse
import sys
from geometry_msgs.msg import Polygon, PolygonStamped, Point32
# from fsd_common_msgs.msg import CarState, CarStateDt, Map, Cone
from fs_msgs.msg import ControlCommand, Track
from visualization_msgs.msg import MarkerArray, Marker
from nav_msgs.msg import Odometry
from geometry_msgs.msg import TwistStamped
import matplotlib.pyplot as plt
import time
import numpy as np


class PurePursuitController():
    def __init__(self):
        print("hello")
        if rospy.has_param("/control_pure_pursuit_node/controller/speed/p"):
            self.speed_p = rospy.get_param("/control_pure_pursuit_node/controller/speed/p")
        else:
            self.speed_p = 0.01
            rospy.logwarn("Did not load controller/speed/p. Standard value is:" +  str(self.speed_p))

        if rospy.has_param("/control_pure_pursuit_node/controller/steering/p"):
            self.steering_p = rospy.get_param("/control_pure_pursuit_node/controller/steering/p")
        else:
            self.steering_p = 0.01
            rospy.logwarn("Did not load controller/steering/p. Standard value is:" +  str(self.steering_p))

        self.control_command = ControlCommand()
        self.center_line = Polygon()
        self.state_ = Odometry()
        self.next_point = Point32()
        self.velocity = TwistStamped()
        self.fig, self.ax = plt.subplots()
        self.ln = plt.scatter(x=np.array([]), y=np.array([]),s = 1.0, c=np.array([]), cmap='hsv',vmin=0,vmax=1)
        self.x_data, self.y_data,self.color = [], [], []
        self.start=time.time()


        self.tem1=0
        self.pub_closest_point_ = rospy.Publisher('/control/pure_pursuit/marker', MarkerArray, queue_size=1)
        self.loadParameters()
        self.subscribeToTopics()
        self.publishToTopics()


    def plot_init(self):
        self.ax.set_xlim(-70, 70)
        self.ax.set_ylim(-50, 100)
        return self.ln

    def update_plot(self,frame):
        end=time.time()
        if end-self.start >=1 and len(self.x_data)!=0 :
            # self.ln.set_data(self.x_data, self.y_data)
            self.ln.set_offsets(np.c_[np.array(self.x_data),np.array(self.y_data)])
            self.ln.set_array(np.array(self.color))
            self.start=end


        return self.ln

    def getNodeRate(self):
        return self.node_rate_

    def createControlCommand(self):
```

```
if not self.center_line.points:
    self.control_command.throttle = 0.0
    self.control_command.steering = 0.0
    return

# a = Point32()
# b = Point32()

it_center_line = min(self.center_line.points, key = lambda i: math.hypot(self.state_.pose.pose.position.x - i.x, self.state_.pose.pose.position.y - i.y))
i_center_line = self.center_line.points.index(it_center_line)
size = len(self.center_line.points)
i_next = (i_center_line+ 2) % size
self.next_point = self.center_line.points[i_next]
print("dx {} dy {}".format(self.next_point.x-self.state_.pose.pose.position.x,self.next_point.y-self.state_.pose.pose.position.y))


self.x_data.append(self.state_.pose.pose.position.x)
self.y_data.append(self.state_.pose.pose.position.y)
self.color.append(0)

beta_est = self.control_command.steering * 0.5
eta = (math.atan2(self.next_point.y - self.state_.pose.pose.position.y ,self.next_point.x - self.state_.pose.pose.position.x) - (self.state_.pose.pose.orientation.z +
#eta = (math.atan2(self.next_point.y - self.state_.pose.pose.position.y ,self.next_point.x - self.state_.pose.pose.position.x) )
ld = math.hypot(self.next_point.x - self.state_.pose.pose.position.x, self.next_point.y - self.state_.pose.pose.position.y)
vel = math.hypot(self.velocity.twist.linear.x, self.velocity.twist.linear.y)
self.control_command.steering = -1*(0.2* math.atan(2.0 / ld * math.sin(eta)*vel))

#self.control_command.steering = -(0.2*math.atan(2.0 *math.sin(eta)/ ld*vel ))
#print(self.control_command.steering)

# temp= self.speed_p * (self.desired_vel - vel)
temp = 0.2*(self.desired_vel - vel)
print(temp)



    if self.tem1<5:
        self.control_command.throttle =1
        self.tem1+=1
    else:
        if temp > 0:
            self.control_command.throttle = temp
            #self.control_command.brake = 0
        else:
            self.control_command.throttle = 0
            #self.control_command.brake = abs(temp)

    self.publishMarkers(it_center_line.x, it_center_line.y, self.next_point.x, self.next_point.y)


def publishMarkers(self, x_pos, y_pos, x_next, y_next):
    markers = MarkerArray()
    marker = Marker()
    marker.color.r            = 1.0
    marker.color.a            = 1.0
    marker.pose.position.x    = x_pos
    marker.pose.position.y    = y_pos
    marker.pose.orientation.w = 1.0
    marker.type               = marker.SPHERE
    marker.action             = marker.ADD
    marker.id                 = 0
    marker.scale.x            = 0.5
    marker.scale.y            = 0.5
    marker.scale.z            = 0.5
    marker.header.stamp       = rospy.Time.now()
    marker.header.frame_id    = "map"
    markers.markers.append(marker)

    marker.pose.position.x = x_next
    marker.pose.position.y = y_next
    marker.color.b         = 1.0
    marker.id              = 1
    markers.markers.append(marker)
    self.pub_closest_point_.publish(markers)
```

```
def loadParameters(self):
    rospy.loginfo("Loading handle parameters")
    if rospy.has_param("/control_pure_pursuit_node/desired_vel"):
        self.desired_vel = rospy.get_param("/control_pure_pursuit_node/desired_vel")
    else:
        self.desired_vel = 1.0
        rospy.logwarn("Did not load desired_vel. Standard value is: " +  str(self.desired_vel))


    if rospy.has_param("/control_pure_pursuit_node/slam_state_topic_name"):
        self.slam_state_topic_name_ = rospy.get_param("/control_pure_pursuit_node/slam_state_topic_name")
    else:
        self.slam_state_topic_name_ = "/fsds/testing_only/odom"
        rospy.logwarn("Did not load slam_state_topic_name. Standard value is: " + str(self.slam_state_topic_name_))


    if rospy.has_param("/control_pure_pursuit_node/waypoints_topic_name"):
        self.waypoints_topic_name_ = rospy.get_param("/control_pure_pursuit_node/waypoints_topic_name")

    else:
        self.waypoints_topic_name_ = "/mapping/waypoints"
        rospy.logwarn("Did not load center_line_topic_name. Standard value is: " + str(self.waypoints_topic_name_))

    if rospy.has_param("/control_pure_pursuit_node/velocity_estimate_topic_name"):
        self.velocity_estimate_topic_name_ = rospy.get_param("/control_pure_pursuit_node/velocity_estimate_topic_name")
    else:
        self.velocity_estimate_topic_name_= "/fsds/gss"
        rospy.logwarn("Did not load velocity_estimate_topic_name. Standard value is: " + str(self.velocity_estimate_topic_name_))


    if rospy.has_param("/control_pure_pursuit_node/control_command_topic_name"):
        self.control_command_topic_name_ = rospy.get_param("/control_pure_pursuit_node/control_command_topic_name")
    else:
        self.control_command_topic_name_= "/fsds/control_command"
        rospy.logwarn("Did not load control_command_topic_name. Standard value is: " + str(self.control_command_topic_name_))



     if rospy.has_param("/control_pure_pursuit_node/node_rate"):
         self.node_rate_ = rospy.get_param("/control_pure_pursuit_node/node_rate")
     else:
         self.node_rate_ = 1
         rospy.logwarn("Did not load node_rate. Standard value is: " + str(self.node_rate_))

     if rospy.has_param("/map_to_waypoints_node/slam_map_topic_name"):
         self.slam_map_topic_name_ = rospy.get_param("/map_to_waypoints_node/slam_map_topic_name")
     else:
         self.slam_map_topic_name_ = "/fsds/testing_only/track"
         rospy.logwarn("Did not load slam_map_topic_name. Standard value is: " + str(self.slam_map_topic_name_))
```

```
def subscribeToTopics(self):
    rospy.loginfo("Subscibe to topics")
    rospy.Subscriber(self.waypoints_topic_name_,data_class=PolygonStamped, queue_size=1, callback=self.updateWaypoints)
    rospy.Subscriber(self.slam_state_topic_name_,data_class=Odometry, queue_size=1, callback=self.slamStateCallback)
    rospy.Subscriber(self.velocity_estimate_topic_name_,data_class=TwistStamped, queue_size=1, callback=self.velocityEstimateCallback)
    rospy.Subscriber(self.slam_map_topic_name_,data_class= Track, queue_size=1, callback=self.mapWayPointConversion)


def publishToTopics(self):
    rospy.loginfo("Publish to topics")
    self.controlCommandPublisher_ = rospy.Publisher(self.control_command_topic_name_, ControlCommand, queue_size=1)


def run(self):
    self.createControlCommand()
    self.sendControlCommand()

def sendControlCommand(self):
    self.control_command.header.stamp = rospy.Time.now()
    self.controlCommandPublisher_.publish(self.control_command)


def slamStateCallback(self, state):
    self.state_ = state

def velocityEstimateCallback(self, velocity_):
    self.velocity = velocity_

def updateWaypoints(self,center_line_stamped):
    self.center_line = center_line_stamped.polygon

def mapWayPointConversion(self,map_):
    yellow_cones=[]
    blue_cones=[]
    for points in map_.track:
        if points.color==0:
            blue_cones.append(points)
        elif points.color==1:
            yellow_cones.append(points)


  x = []
  y = []
  c = []
  for yellow in yellow_cones:

      it_blue = min(blue_cones, key = lambda x: math.hypot(yellow.location.x- x.location.x, yellow.location.y - x.location.y))
      p = Point32()
      p.x = (yellow.location.x + it_blue.location.x) / 2.0
      p.y = (yellow.location.y + it_blue.location.y) / 2.0
      p.z = 0.0
      x.append(p.x)
      y.append(p.y)
      c.append(0.33)
      x.append(yellow.location.x)
      y.append(yellow.location.y)
      c.append(0.16)
      x.append(it_blue.location.x)
      y.append(it_blue.location.y)
      c.append(0.62)


  self.x_data = x
  self.y_data = y
  self.color = c
```