

# Centerline Estimation

## Masters Project Report

by Ashok Vaishnav (1152142) and Akshay Agrawal (1152254)

### Introduction

Driverless cars are getting more and more popular in these times. There have been numerous experiments going on and prototypes are being developed in the field as well. Many car manufacturing companies have already advanced and released such cars. Driverless cars can not only be used for everyday purpose but also as a part of sport, such as motor racing. In Germany, Formula Student(Raceyard) is a competition for just the same wherein every year student teams compete using self-designed and developed vehicles.

This project focuses on a section Autonomous Driving Car in ROS for Formula Student majorly in ROS pipeline. There are four sections in which the pipeline is divided.

1. Cone Detection: Using sensors like camera and lidar, cone color and its position is estimated wrt to the car either from a simulation or the real world
2. SLAM: Taking in the detected position of cones, a 2D map of the race track is supposed be created after the first lap. Also, localizing the car position at all times.
3. Centerline Estimation: It estimates and gives a path which is supposed to be followed during the race. It comes from accumulation of the centre points of the cones.
4. Driver: Tuning parameters such as speed, steering angle which helps the driver to complete the lap in an optimal time.

The goal is to focus on section 3, which deals with calculation of the centreline. The goal is achieved by using techniques such as interpolation, classification using SVM, and visualising with matplotlib. Centerline calculation can be split into two cases.

1. The first, where the live centerline calculation needs to be done for the very first lap. In this lap, no cones are identified and we do not have a clear picture of the racecourse/ map. As the car moves forward it starts detecting the cone and stores them in the pointcloud, and a centerline is calculated simultaneously giving it a path. Upon completion of the first lap, a map is formed or rather a pointcloud that identifies the cones for the left and right borders of the track by their colors i.e., blue and red. The start and the end cones are identified by orange color.
2. The second case calculates a robust centerline, by using this complete data. This second iteration gives out a better and complete centerline as it has the accumulated data of all the

cones in the plane. It considers and handles anomalies caused, such as missing or misidentified data.

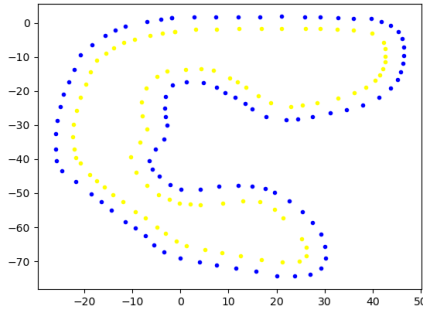


Figure 1: Track plot

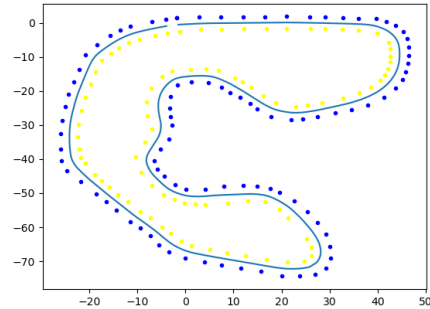


Figure 2: With centerline

Figure 1 shows a sample track, whereas Figure 2 portrays the tracks with the estimated centerline. As we can see in the figure, it has created a path in the middle of the track. Be it a straight line or a curve the points in the center have been accumulated forming a Centerline.

There might occur some cases of augmented cones which includes

1. Missing cones - Ideally the number of cones on the left and right boundary of the track should remain the same. But in such cases, some random cones might be missing from the pointcloud while recording the data, which might not have been picked up by the sensors.
2. Misidentified cones - A set of random cones which are miscolored. They can be labeled as blue instead of red or vice versa.
3. Unsorted/shuffled pointcloud - For calculating the centerline we need to know the order in which the cones proceed to identify the map and direction of the race. Which creates the need for sorting the cones.

## Existing System

The project uses a remote server Thinlinc having a Linux client and ROS has been set up on this client. Gazebo a simulation tool inputting parameters of the Raceyard project and converting them to simulations based on real-world physics is used. Also, RVIZ being a simulation tool uses this data from Gazebo and creates a visual simulation on the ROS environment. A python virtual environment is created on which the whole ROS is present.

A working simulation was already present at the beginning having multiple sections present for different steps of the pipeline. Different languages are being used and are combined to get the result. This task is being done by Cmake.

One of the crucial directories on which the project is rooted is the parameters directory. It

contains the .yaml format files written for initialization of the parameters, combining different sections of the project into one, as well as control over real-time values and cheat(dummy) values being used for the simulation. Initially, it uses a set of cheat cone driver as well as the pointcloud parameter that is predefined in ROS for the simulation.

The part for the centerline had been written in C++, which created a dummy centerline for the car. To get this centerline it uses cheat driver cones applying some basic mathematical formulae. Densification had been done so that the centerline becomes smooth and complete.

## Implementation

The implementation can be segregated into two sections.

### Section 1 by Ashok Vaishnav

For achieving the final goal of estimating centerline, the process has to tackle several issues or anomalies which may be present in the input data set. Because of this, pre-processing of input data set is necessary and followed by calculating centerline.

The process is now divided into two parts :

1. Pre-processing of input
2. Estimation of centerline

#### 1. Pre-processing of input

For a better estimation of centerline, pre-processing of the available data(cones) is required. The pre-processing consist of tasks such as:

- Cones color classification for misidentified/unidentified color of cones
- Removal of redundant cones
- Order the list of spatially unordered cones

##### 1. Cones color classification:

In a track/map, cone positions are non-linear(x and y increasing and decreasing to form a path). Since, there are many classification models available for non-linear data, one of them being SVM(Support Vector Machine) classification model with "radial basis function kernel"(for polynomial or non-linear data) can be used to fit a track/map data. This will introduce the 3rd dimension adding a plane between the data to classify them accordingly.

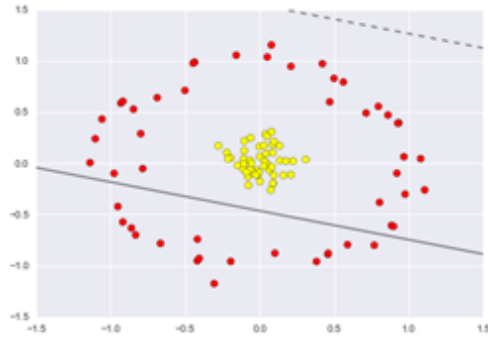


Figure 3

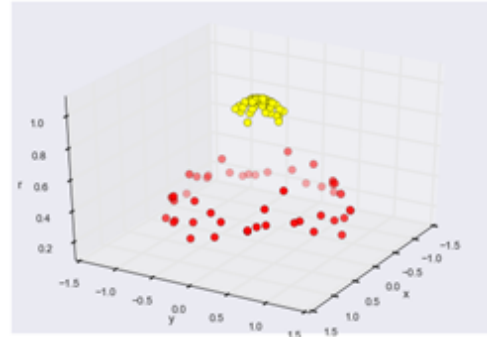


Figure 4. Picture reference:

<https://jakevdp.github.io/PythonDataScienceHandbook/suppor-vector-machines.html>

## 2. Spatial ordering of cones:

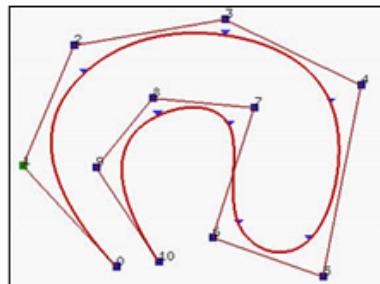
There can be a case where there are duplicated and unordered cones in the list/array. In this second stage of pre-processing, duplicate cones are removed and then being passed to order them in sequence to make a perfect path to iterate. If cones are already in order then they remain unchanged otherwise they are ordered by iterating each cone in the list and by estimating their distance to other cones and swapping their positions.

## 2. Estimation of centerline

Now, here cones are in an expected format to estimate centerline. For getting more accurate centerline there should be more data and that is where the interpolation of cone data using BSpline is introduced.

### 1. Generating interpolated data using BSpline

This process has become necessary to generate a more accurate centerline as this gives interpolated data with a length of 10 times(can be changed) the number of cones. This is being used for both blue and yellow colored cones. It smoothens(spline) the path and also ignores any noisy point in the path.





## 2. Centerline calculation



This is the final stage where the centerline is estimated. The process iterates each data point in interpolated data on either side which has lesser data length and finds the nearest point in the other list of data points and then evaluates the center point between those two points and we get centerline. Following the smooth data points on the path, there is a smooth and interpolated centerline.

## Functions Description

### Key Functions

 Function	 Description
<b>get_centerline(side1, side2)</b>	<u>It evaluates the centerline points between both sides 'side1' and 'side2'.</u>
<b>getBSpline(list)</b>	<u>Interpolate the polynomial geometrical points list with the length of 10 times larger the input list..</u>
<b>get_sorted(list)</b>	<u>Sort the given coordinates list according to their spatial location and return the sorted list.</u>
<b>clasify_cones(X, y, TestSet)</b>	<u>Classify the cones into two different lists using SVM(support vector machine) mode with radial basis function.</u>

### Supporting Functions

 Function	 Description
<b>remove_dup(list)</b>	<u>Removes redundant(duplicate) data from input list and returns the list.</u>
<b>orient_clockwise(list)</b>	<u>Will return the clockwise oriented list(in some cases after sorting the cones are in anti-clockwise orientation).</u>
<b>return_list(list)</b>	<u>Returns x,y coordinate in [x,y] format.</u>
<b>expand_xy(list)</b>	<u>Expands the x,y coordinates from the input list to two different lists of x and y.</u>
<b>nearest(side2, n_side2, side1, x)</b>	<u>Finds the nearest point in side2 from side1 point having index 'x'.</u>
<b>cones_to_xy(cones)</b>	<u>Expands the cone objects list to x,y position of cones.</u>
<b>bind_xy(x, y)</b>	<u>Binds the x,y values together to represent a geometrical point</u>
<b>distance(p1, p2)</b>	<u>Calculates the distance between two geometrical points</u>
<b>pre_process_cones(blue_cones, yellow_cones, noisy_cones)</b>	<u>A wrapper function for pre-processing processes.</u>

## Section 2 by Akshay Agrawal

### Platform Migration

The tasks proposes a need for Deep Learning in future, reasoning to which the platform had to be migrated from C++ to python. Python has many inbuilt libraries for Linear Algebra, Interpolation as well as Deep Learning, making the job easier and faster. A major part while doing so is to get the Subscriber/Publisher function running, as the published value is the output for the task.

**Issues faced:** Parameter type which needs to be passed in the Publisher function.

**Solution:** This issue can be solved using an object of the type Map which can be found under `src/rotyard_common/msg`. The output is specified in a fixed format that is defined in the `Map.msg`. In this directory, there are a couple of types that have been specified and are used throughout the project.

### Additional Files



In the directory `rotyard_common/src/rotyard_common_scripts/` there are files where a few functions have already been defined which can be referred while using the pointcloud. One of the functions that is used, is to convert the cone color to string from type RGB (`common_methods.py`). This is an important step when sorting the cones into list/arrays of left and right track i.e. boundaries of the track, colors being blue and red. It is recommended to look up this directory before starting to write the actual code or make changes in the existing ones.

**Note:** The pointcloud from ROS comes in a tuple format which is immutable so a copy needs to be made for changes or using these functions to it.

### Augmentations

Live data recordings using sensors can produce a pointcloud which has various anomalies and augmentations that need to be dealt with. During simulations, we use an ideal pointcloud which does not contain any augmentations. To make the centerline robust and test the program against this, we need to produce these errors ourselves. Different functions were created which can be called to add these anomalies.

#### Functions

 Function	 Description
<u><a href="#">Unsee cones</a></u>	For creating missing cones.
<u><a href="#">Mislabel</a></u>	Misinterpreting the cone colors.
<u><a href="#">Randomize</a></u>	Changing the order of the cones, randomizing it.

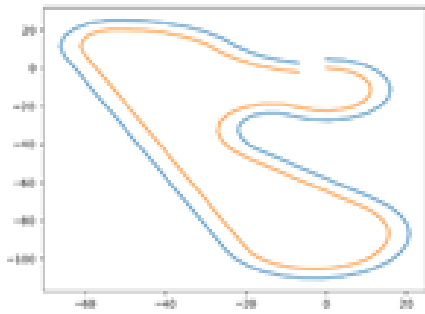


Figure 6. Missing points

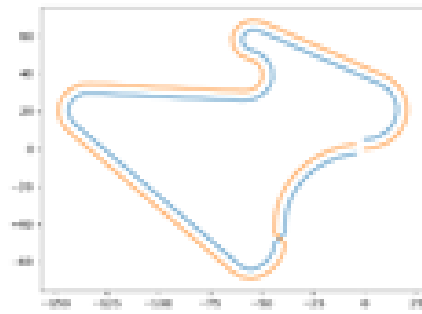


Figure 7. Mislabeled data

## Testing and Visualisation

The code for centerline estimation had to be experimented with against different tracks and visualize the output to check for robustness. To this, the input tracks (cone coordinates) must be loaded and used. The input in .yaml format needs to be loaded to be processed.

### Functions used

Aa Name	Tags
<code>load_track()</code>	Loads a yaml file, creating a list for different cone colors.
<code>get_track()</code>	Loads N-number of tracks at once with track number of them.

The testing needed to be done for several tracks which had to be visualized and recorded. We started with Wandb which is a tool for logging the work, getting system stats, and demonstrating signs of progress.

**Problem:** Wandb is primarily used for logging the parameters for machine learning and was unable to plot all of the track and centerline findings in one plane.

**Solution:** Matplotlib was used for visualizing the centerline and saving every plot. All the track files are loaded and iterated for this purpose.

# Experiments and Results

Below are the results based on a provided sample map “FSG”.

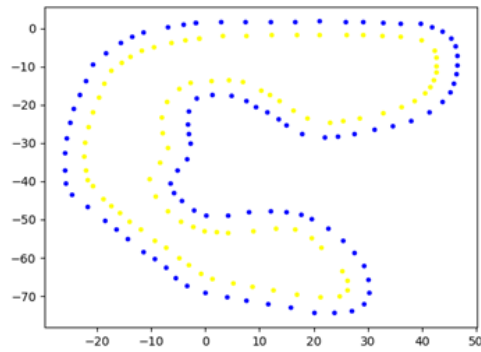


Figure 8. FSG Map

## 1. Pre-processing

- Cones color classification

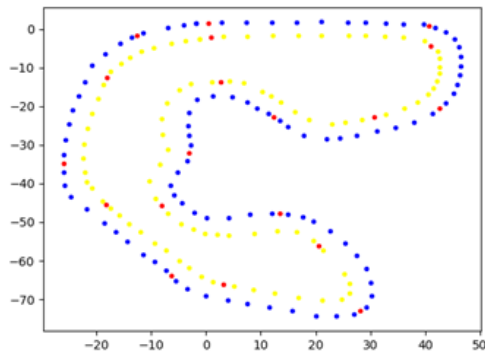


Figure 9 Red points represents misidentified colored cones

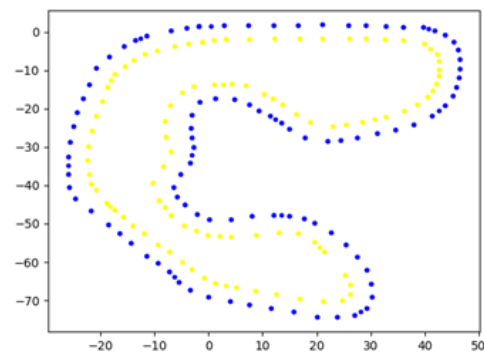


Fig.10 Cones classified using SVM classification model

- Spatial Ordering



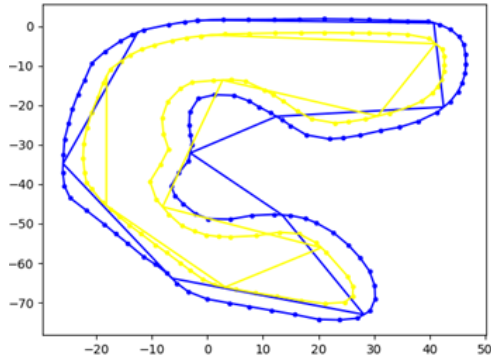


Fig.11 Cones are classified but they are not spatially ordered in the list

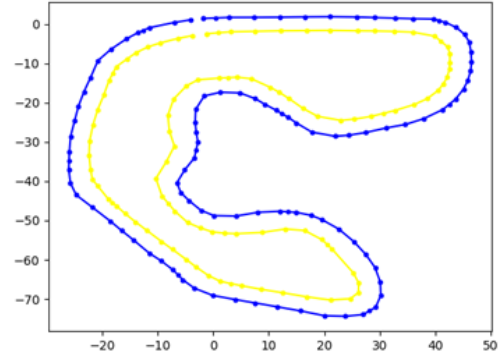


Fig.12 Cones are ordered and also duplicate free(see the disconnected path in the end)

## 2. Estimation of Centerline

- Generating interpolated data using BSpline

### Estimation of Centerline

- Generating interpolated data using BSpline

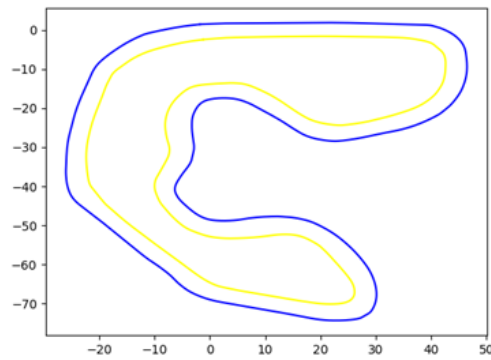


Fig.13 Cones are interpolated using BSpline

- Estimated Centerline

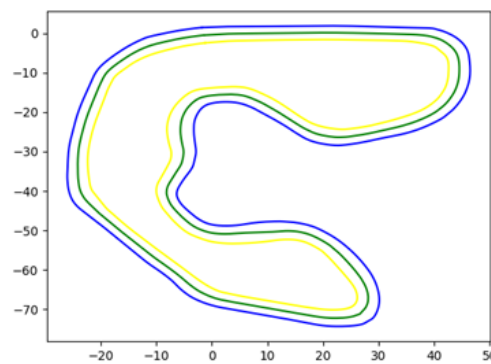


Fig.14 Centerline between the smooth spline path

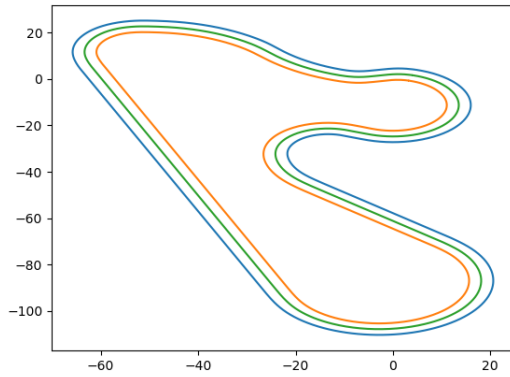


Fig 15. In presence of missing points corresponding to Fig. 6

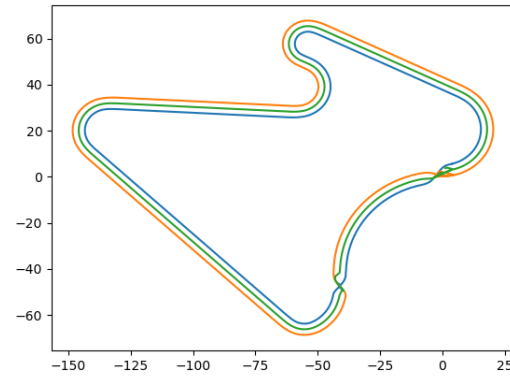


Fig16. Centerline when data is mislabeled corresponding to Fig. 7

## Conclusion

In this project, we successfully implement the centerline for the driver. A smooth centerline discarding the noise in data is achieved using the BSpline interpolation method. The platform was migrated from C++ to python to make the code more efficient and simpler. Having the centerline estimation in python enables an option for Deep Learning. The anomalies in the input data namely missing cones, mislabeled data are handled as well as sorting of the input data is carried out by spatial ordering of cones. Various iterations are visualized on different tracks depicting the accuracy of the centerline.

The calculation of centerline can be taken further a few steps by trying to implement Deep Learning. For the application of deep learning, CNN can be used by training the model with input images of track visualization and output with centerline estimated. DNN cannot be considered as it requires the number of outputs that will be calculated. While training a model, the number of input and output should be fixed. But in the case of centerline, the track can be of variable length and the number of cones on the left and right side of the track may differ.