**Basic Solutions**

**Method 1**

We tried a naive approach of giving the entire feature set and directly predicting one of the 5 actions [UP, DOWN, RIGHT, LEFT, No\_Control]. The vehicle was taking a right turn but not at the junction, it was colliding with the edges of the road either before the junction or after the junction and also in this way we are not even telling the vehicle to avoid collisions with other vehicles. We have given you a simple mlp code for it, but not used it, to try it for yousellf make appropriate changes in the commander.py file explained further below. We would encourage you all to try this simple model and think of why it can fail (Few of the reasons were mentioned in the initial document which was shared).

**Method 2 - (We are sending this code)**

Debugging a problem in ML is tough (Understanding why the vehicle was not taking a proper right turn using the above method), in such scenarios it always helps to simplify the problem.

We divide the problem into 2 parts, One is the steering control which is responsible for taking right and left turns and the other is acceleration control responsible for controlling acceleration to avoid collisions. We present a very basic solution where the steering behaviour is controlled by a regressor and the acceleration is controlled by a simple physics and basic optimization technique.

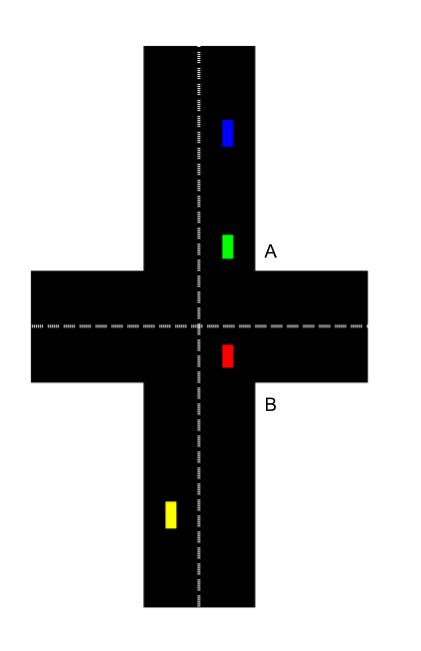
**PART 1 - Controlling the Steering**

We train a simple Regressor which can predict the orientation of the vehicle given the input features.

**Training the Model**

Input features :-

1. X Pos of the Vehicle
2. Y Pos of the Vehicle
3. X Velocity of the Vehicle
4. Y Velocity of the Vehicle
5. Y Distance of the vehicle from A( Mentioned in the image given in the initial document)
6. X Acceleration of the Vehicle
7. Y Acceleration of the Vehicle
8. X Distance of the vehicle from A
9. Absolute distance of the car from B
10. Angle at which car approaches the junction.



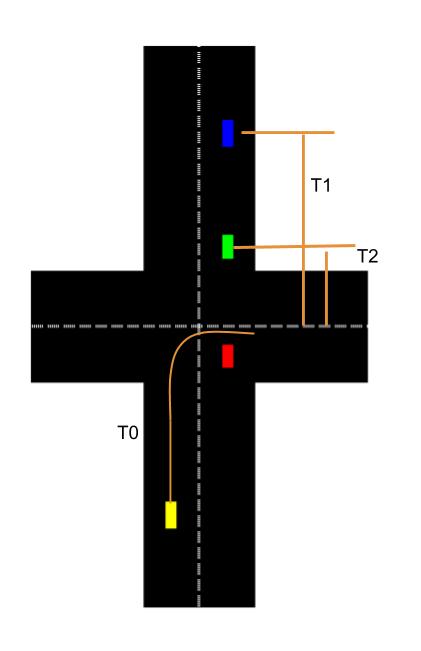
Using this we create a regressor which predicts the orientation given the above the features. The code for training the regressor is present in the jupyter notebook **train\_orientation** in the **models/** folder. The code is well commented so you will get the complete idea of how the regressor is trained by simply going through the code.

The generated model gets saved in **models/params** folder. Next, **src/commander.py** given the set of features will output the next orientation.(i.e to take RIGHT or LEFT) using the model we created.

The model works well to get the car to the correct path if it hasn’t deviated much, can you tackle those challenges as well? Also, you may want to hardcode some scenarios (real life autonomous systems have a lot of hardcoding as well). The scenarios might include taking a right turn if you cross a particular location and the orientation of the car is still not in accordance. It also includes scenarios such as stopping the car from entering the right lane of the straight road too early.

**PART 2 - Controlling the acceleration of the vehicle to avoid collision**

From the below figure we can see that the vehicle B takes T1 time to reach the junction, vehicle C takes T2 time to reach the junction (We can calculate this because we know velocity and distance of the vehicles and they follow a straight line).



Let Vehicle X (Yellow vehicle) take T0 time to reach the junction. We define an optimization problem as follows:

Given T1 and T2 select T0 such that it is far away from T1 and T2. To get such a T0, we are using a cost function F (defined below). F is constructed in such a way that its value increases as T0 is nearer to T1 or T2. The problem gets reduced to minimizing F to get optimal T0.

The F function chosen is as follows.(There are many possibilities of such a function).

F = Gaussian(mean = T1, standard deviation = 10) + Gaussian(mean = T2, standard deviation = 10)

We can notice that at t = T1 or T2 the above function gives higher values.

We now do an exhaustive search over various values of T0 and find the T0 which gives the least value of the cost function.

Once we have T0 we essentially know at what time the vehicle should reach the junction to avoid collision. We have the initial velocity all we need is to compute the distance of the vehicle to the destination to find the acceleration (using s = ut + ½\*a\*t\*t).

We calculate the distance of the road intersection from the Target vehicle by assuming it to take a fixed path (Straight line for sometime and then a smooth quarter circle as shown in the figure.). Using this formula for distance we approximate the distance the vehicle needs to cover in time T0 derived from the above optimization problem.

Using the distance, time T0 and velocity we calculate acceleration to be given to the vehicle at that given time. We repeat this process at each instance and help the vehicle move without colliding. (Since we are approximating the distance the vehicle can still collide on few instances.)

The code for this is present in **src/commander\_physics.py**

The set of actions the vehicle can perform is given in **actions.py**

**For running this model**

Run src/main.py

For switching between manual/network mode where you can control the vehicle or leave it to the network press ‘m’.

**IMPORTANT NOTE:**

The data that we collected has 2 values calculated wrong: feature indices 18,19, that is the current acceleration of the vehicle X. However, it can easily be corrected by taking a diffrence between the current frame and previous frame velocity and we advice you to use that instead of using the one in the dataset. The current code calculates it the right way. If you want to train on a small model you can generate yout own trips again and use “python src/createDataset.py” command to convert your trips into a single file which you can use to train your model.

We encourage you to change and experiment with both commander\_physics.py and the orientation model.