Team Roadrunner Project Report

Arnav Dhamija Luca Scheuer Saumya Shah

***Abstract*—This paper explains our approach to a develop a racing strategy to compete in head to head racing of F1/10 Autonomous Race Cars in simulated environment. It discusses our solutions to various aspects of developing an end-to-end racing strategy. The first task being raceline optimization to find an optimal path to follow. It then extends to local planning by selecting a path which is dynamically feasible and at the same time collision free, and finally use Model Predictive Control for this reference tracking. We also present lane switching strategy to promote overtakes. The method described was successful in racing while racing at the speed of 4.5 m/s and tackling various scenarios arising in head-to-head race situations.**

Index Terms*—Autonomous vehicle, Model Predictive Control, CMA-ES, Raceline Optimization, RRT

I. Introduction

The task given to us for our final project was to develop an agent which could race competitively in a head-to-head race with another agent in a virtual ROS environment. The agents in question are simulated versions of the F1/10 race cars we used in the first half of the semester. To simulate the real cars, these agents are equipped with a virtual planar LIDAR scanner which returns distance measurements 360° around the car at a resolution of $\frac{1}{3}^\circ$. The agent can also access its own and the opponent agent's precise location on the map during the race.

Our challenge was to develop a robust, real-time approach which could 1) track an optimized raceline trajectory at close to the maximum speed of 4.5m/s with minimal drift, 2) avoid crashing into obstacles, such as walls of the track and the opponent car, and 3) overtake the opponent car when it can be done safely. Our solution, described in Section III, to these three key problems is using Model Predictive Control (MPC) to track an optimized raceline trajectory, generated using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). For overtaking, we generated multiple concentric trajectories which the agent can switch to tracking in realtime. We have also discussed our results of using this approach in Section IV.

II. Related Work

The work on superoptimization tool chain for autonomous racing by Achin Jain et. al. in [6] tackles the problem of optimization of various parameters including raceline and dynamics of the car for racing. They demonstrate the optimization of parameters using CMA-ES with the laptimes as the fitness scores and implementing it in parallel exploiting the independent nature of CMA-ES rollouts. This formed the basis of our raceline optimization approach.

Liniger et. al. in [1], presents two methods for achieving optimization and control in real time. The Hierarchical Receding Horizon Control (HRHC) method breaks the task in two stages, path planning and tracking using MPC over receding horizon. The other method which paper presents, is using Model Predictive Contouring Control (MPCC) [2] in context of racing, which combines both the tasks by using a controller that generates trajectories which are projected on the centerline spline and the least cost path is chosen as optimal path. This paper also shows how the methods can be extended for obstacle avoidance by using dynamic programming to generate the optimal corridor and passing the corridor as the constraints for optimization. Our approach, described in the following section, is inspired by the HRHC method.

III. Methods

The following subsections discuss the various aspects of our racing strategy:

A. Raceline Optimization

The first step to racing would be to optimize the raceline as much as possible. We approached this task using CMA-ES for optimization as discussed in [6]. The key contribution of our approach is formulating a simplified ptimization problem by using polar co-ordinates instead of the Cartesian co-ordinates with boxes transformed and fit to fill up the free space around the center-line on the track. We begin by using the centerline spline indicated by the blue line in the figure below and sparely sampling it indicated by the small red dots. We then use $(r, \theta)_i$

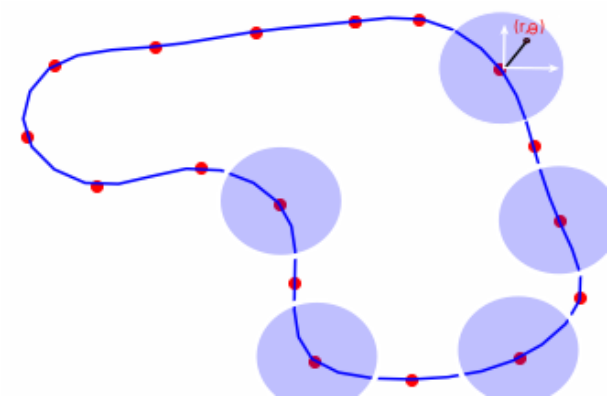

Fig. 1. CMA-ES optimization with center-line reference. Blue shaded circles indicate constraints on the maximum radius of search

relative to each of these points as the parameters of the optimization problem. Reducing the problem to polar co-ordinates with constraints on radius helps in avoiding the transformation of the bounding boxes. The maximum radius bound for each of these reference points is analytically

computed. The result of each CMA-ES iteration converted to the Cartesian coordinates using simple polar to Cartesian conversion. A regularized cubic spline is used to fit a smooth spline over these points. A velocity profile is then generated as a preset function of curvature. The time to complete the trajectory is used as a measure of fitness, which virtually leads to a minimum curvature trajectory. The optimization was then performed over $(r, \theta)_i$ pairs for each of these points using CMA-ES.

Result

Fitness curve Final Path

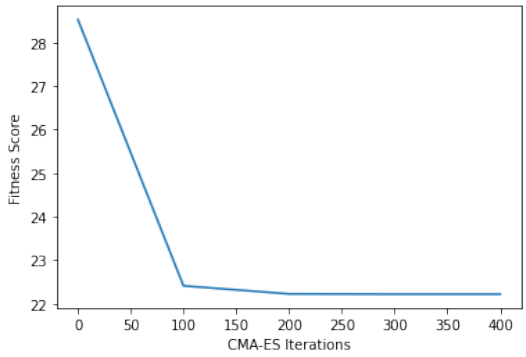

Fig. 2. Fitness vs Optimization iterations

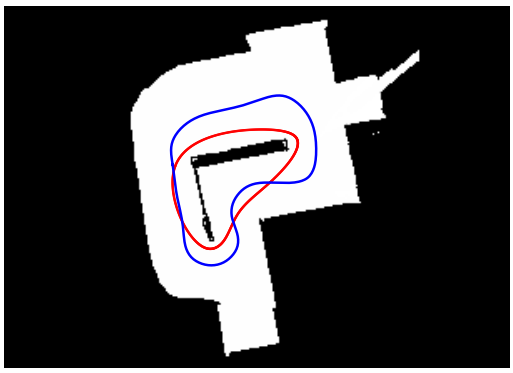

Fig. 3. CMA-ES Optimized trajectory (Red) vs Centerline (Blue)

This helped us win the single-player, no obstacle race in Milestone 2 with a two lap time of 10.94s.

B. Local Planner

The local planning approach involved the following two tasks:

1) Local Path Selection: A set of nine paths were generated to offline to be used as local trajectories which track the global CMA-ES optimized trajectory. The set of 9 paths are constant curvature arcs as shown in Figure 4.

The behavior of the car was empirically analyzed and the velocities corresponding to different steering angles were set to minimize drift. The dynamically feasible trajectories

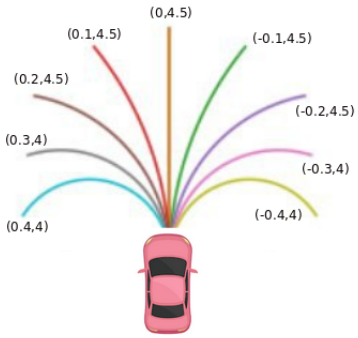

Fig. 4. Local paths for nine (steering angle, speed) pairs

corresponding to the nine (steering angle, speed) pairs were calculated and sampled with the simulator time step size of 0.01 seconds.

Depending on the state of the car, one of these paths were selected and later tracked. A lane switching mechanism selected the lane to be tracked as discussed in the section following this. Once the lane or the global trajectory to be followed is decided, one of these paths is selected using on the following criteria in order:

1) Collision check: Each of the segments of these paths was checked for collision using the efficient Bresenham's Line Drawing algorithm over the occupancy grid.
2) The end point of this local path and the closest point on the global path that was being tracked is checked for collision. This was added for robustness by preventing selection of paths leading the car to points points on other sides of the wall when the car was far from the global path.
3) If the above two tests are passed, a fitness value is calculated as follows:

$$F = dist(P, G) - \beta * dist(L, G) \tag{1}$$

where:
$dist(A, B)$ represents Euclidean distance between points A and B,
P is the current position of the car,
L is the end point of the local path,
G is the point on global path closest to L, and
β is a hyper parameter which weighed the progress along the global path against being close to the global path.

4) Lastly, to avoid cases of the agent erroneously selecting a goal point in the clockwise direction, the path is checked for following the anticlockwise direction along with maximization of fitness value. This added more robustness to the selection of the local trajectory.

Result

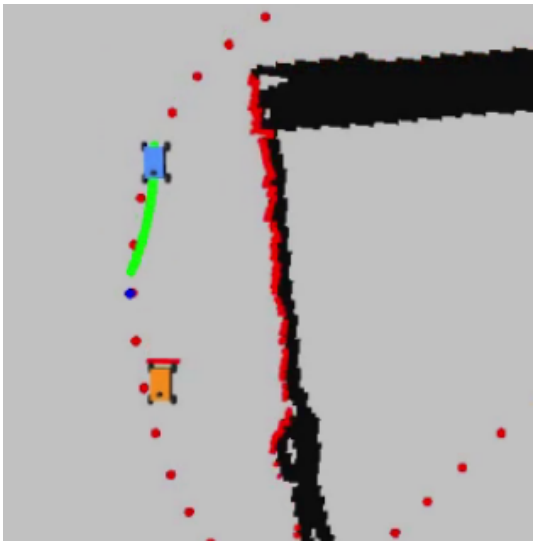

Fig. 5. Local path (green) tracking the global path (red dots)

2) Lane Switching: The CMA-ES optimized trajectory was expanded outwards concentrically to create trajectories 0.25m apart. These trajectories were used as lanes for the agent to travel on as others became blocked. The lanes were ranked from best to worst based on their distance from the optimal trajectory. Two conditions could cause a lane switch.

The first condition was if the current lane is blocked. Blocked was defined as stepping forwards and backwards a paramaterized distance along the lane from the closest point to the agent. If any segment within this distance along the lane was blocked in the occupancy grid, the lane was considered blocked.

The second condition was if the current lane was not the best lane and had been traveled on for longer than a paramaterized tracking distance. This way if the innermost lane was blocked the agent would transfer to another lane and stay there for some time before attempting to merge back to the best lane. If this distance was set to 0m the car could oscillate between the best lane and the first non-blocked lane.
Lane switching was accomplished by stepping through the list of lanes from best to worst and checking if the lane was blocked. The first non-blocked lane was selected as the current lane and the local path selection was used to follow that lane.

This lane switching behavior allowed the agent to perform overtakes of slower vehicles and more effectively avoid large static obstacles before the local paths could intersect the obstacle.

C. Model Predictive Control

The MPC formulation consisted of a simple quadratic cost for state and four types of constraints. The four constraint types were dynamics, minimum and maximum input constraints, follow the gap cone constraints, and skid constraints.

To solve our formulation, we converted the MPC problem to a standard quadratic program (QP).

$$
\begin{aligned}
u_0^* = \arg\min_{x_k, u_k} \; & (x_N - x_{r,N})^T Q_N (x_N - x_{r,N}) + \\
& \sum_{k=0}^{N-1} ((x_k - x_{r,k})^T Q (x_k - x_{r,k}) + \\
& (u_k - u_r)^T R (u_k - u_r)) \\
\text{subject to} \; : & \\
x_{k+1} &= A x_k + B u_k \\
x_{\min} &\le x_k \le x_{\max} \\
u_{\min} &\le u_k \le u_{\max} \\
x_0 &= \bar{x}
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
\text{minimize} \quad & \tfrac{1}{2} z^T H z + f^T z \\
\text{subject to} \quad & l \le C z \le u
\end{aligned}
\tag{3}
$$

In addition to constraints, the dynamics of the car had to be modeled. The combination of constraints and a model allowed for the controller to select inputs that would allow the agent to travel along the desired local path.

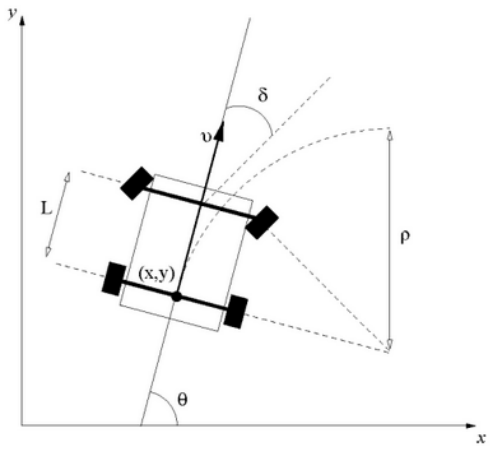

Fig. 6. Kinematic Model representation

1) Modeling Dynamics: A simple kinematic vehicle model was used for capturing the non-holonomic behavior of the system. The state x and input u were as defined below:

$$
\begin{aligned}
x &= [\text{x,y}, \psi]^T \\
u &= [v, \delta]^T
\end{aligned}
\tag{4}
$$

where,
(x,y) are coordinates of the car on the map,
ψ is the orientation,
v is the velocity, and
δ is the steering angle

The non-linear dynamics were formulated as follows:

$$\begin{aligned}\dot{x} &= v * cos(\psi)\\ \dot{y} &= v * sin(\psi)\\ \dot{\psi} &= v * tan(\delta)/L\end{aligned} \tag{5}$$

The above system was linearized about the current position and previous input. The model was then discretized using Forward Euler Discretization using the time step of the previous state to the current one. The following formulation was the results. It proved to be fairly accurate if discretization time step Δt was below 200ms [7]:

$$x_{t+1} = (A\Delta t + I)x_t + B\Delta t u_t + C$$

where,

$$A = \begin{pmatrix} 0 & 0 & -v_{ref}sin(\psi_{ref}) \\ 0 & 0 & v_{ref}cos(\psi_{ref}) \\ 0 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} cos(\psi_{ref}) & 0 \\ sin(\psi_{ref}) & 0 \\ tan(\delta_{ref})/L & vsec^2(\delta_{ref})/L \end{pmatrix} \tag{6}$$

$$C = \begin{pmatrix} v_{ref}\psi_{ref}sin(\psi_{ref}) \\ -v_{ref}\psi_{ref}cos(\psi_{ref}) \\ -\delta_{ref}v_{ref}sec^2(\delta_{ref})/L \end{pmatrix}$$

*The subscript **ref** denotes parameters are corresponding to the reference state and action for linearization.*

This model was used because it was simple enough, yet effective in capturing most of the behavior which allowed us to use large time horizons while still solving the MPC problem in real-time.

2) Quadratic Program Construction: To construct the quadratic program (QP) a method was used to create large sparse matricies that could represent the whole time horizon as one variable. For each time step in the time horizon (N) there is a state x_t and an input u_t. Additionally, there is one state appended at the end known as the terminal state, x_N. This series of states and inputs is combined to become the single optimization variable z:

$$z = [x_0, x_1, ...x_{N-1}, x_N, u_0, ...u_{N-1}, u_{N-1}] \tag{7}$$

This allows all states and inputs to be optimized as one variable, allowing for existing quadratic optimization libraries to use. The final size of this optimization variable is:

$$size(x) * (N + 1) + size(u) * N \tag{8}$$

3) Cost Function: For each time step in the horizon there is a cost associated with each state, Q and a cost associated with each input, R. To convert the problem to a QP the original cost function's quadratic forms can be expanded.

$$\begin{aligned} &x_N^T Q x_N + x_{r,N}^T Q x_{r,N} - 2x_{r,N}^T Q x_N \\ &\sum_{k=0}^{N-1} (x_k^T Q x_k + x_{r,k}^T Q x_{r,k} - 2x_{r,k}^T Q x_k + \\ &\qquad u_k^T R u_k + u_r^T R u_r - 2u_r^T R u_k) \end{aligned} \tag{9}$$

Each term multiplied only by $x_{r,k}$ or u_r is constant and can be removed as it does not affect the optimization:

$$\begin{aligned} &x_N^T Q x_N - 2x_r^T Q x_N \\ &\sum_{k=0}^{N-1} (x_k^T Q x_k - 2x_{r,k}^T Q x_k + \\ &\qquad u_k^T R u_k - 2u_r^T R u_k) \end{aligned} \tag{10}$$

Looking back at equation 3, the equation can be broken down into parts multiplied by $\frac{1}{2}$ and those not. Similarly this can be done for equation 10.

To create H from the QP for use with our new optimization variable z, the quadratic costs are put into the block diagonal form shown below, one Q per state, and one R per input.

$$H = \left[\begin{array}{ccc|ccc} Q & & & & & \\ & \ddots & & & & \\ & & Q & & & \\ \hline & & & R & & \\ & & & & \ddots & \\ & & & & & R \end{array}\right] \tag{11}$$

This results in a square matrix with row and column size equal to the optimization variable z. The variable f from the QP is calculated using the desired state and input as well as Q and R.

$$f = \begin{bmatrix} -Qx_{r,0} \\ \vdots \\ -Qx_{r,N} \\ -Ru_r \\ \vdots \\ -Ru_r \end{bmatrix} \tag{12}$$

The desired states given to each step in the horizon are taken from the local trajectory planner. The desired inputs are constant. Generally, the desired input was set to full throttle and zero steering.

4) MPC Constraints: To encode constraints into the optimization problem they must be in the form from equation 3. l and u are upper and lower bound vectors for the constraints. C is a linear constraint matrix that allows any affine combination of the components of the z vector. This means constraints based on each state or input individually, combinations of states and inputs, or combinations of states and inputs from multiple time steps can be realized. With these affine functions, all the desired constraints were realized.

Below is the C matrix split among the constituent constraint types.

$$C = \left[\begin{array}{c|c} \multicolumn{2}{c}{dynamics} \\ \hline & input \\ \hline cone & \\ \hline & skid \end{array}\right]$$

$$l = \begin{bmatrix} dynamics \\ input \\ cone \\ skid \end{bmatrix}, u = \begin{bmatrix} dynamics \\ input \\ cone \\ skid \end{bmatrix} \tag{13}$$

5) Dynamics Constraints: The dynamics constraints were realized using a standard state space structure which can be realized as an affine function as shown below.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ Ax_k - x_{k+1} + Bu_k &= 0 \\ \begin{bmatrix} A & -I & B \end{bmatrix} \begin{bmatrix} x_k \\ x_{k+1} \\ u_k \end{bmatrix} &= 0 \end{aligned} \tag{14}$$

From that single update function we can reformulate it into a block matrix which can be multiplied by the z vector, resulting in the state update constraints. The first row is reserved for the initial state.

$$C_{dynamics} = \left[\begin{array}{cccc|ccc} -I & & & & 0 & & \\ A & \ddots & & & B & \ddots & \\ & \ddots & \ddots & & & \ddots & 0 \\ & & A & -I & & & B \end{array}\right] \tag{15}$$

Because the dynamics are equality constraints, the lower and upper bounds are equal. The first state does not contain the dynamics and is set to the current position of the vehicle. Every other state is set to a constant. Normally for a linear system this is set to zero. As this system is non-linear and subsequently linearized, the bounds are set to a linearizing coefficient, C discussed in equation 6, and seen below.

$$l_{dynamics} = u_{dynamics} = \begin{bmatrix} -x_0 \\ -C \\ \vdots \\ -C \end{bmatrix} \tag{16}$$

6) Input Constraints: The input constraints are a more straightforward formulation than the dynamics. A block of the linear constraint matrix corresponding to all the inputs in z is set to identity and the corresponding upper and lower bound are set to the limits as shown below. In the case of this race the upper bound on velocity is set to 4.5m/s and the lower bound is set to 0. The steering angle constraints are set to the minimum and maximum that the car will allow.

$$C_{input} = \left[\begin{array}{ccc|ccc} 0 & & & I & & \\ & \ddots & & & \ddots & \\ & & 0 & & & I \end{array}\right]$$

$$l_{input} = \begin{bmatrix} u_{min} \\ \vdots \\ u_{min} \end{bmatrix}, u_{input} = \begin{bmatrix} u_{max} \\ \vdots \\ u_{max} \end{bmatrix} \tag{17}$$

7) Cone Constraints: While not used during the final race, a cone constraint was tested with the MPC formulation. This consisted of finding the largest gap within a prescribed field of view within in the lidar scan. This gap in the lidar scans defined a cone from the vehicle's lidar that intersected the edges of the gap, creating two affine constraints using the x and y positions of the vehicle as variables. Both constraints are formulated as half spaces where one side is allowable and the other is not, thus creating a cone. Each constraint consisted of an A and a b to create the limit.

$$Ax \leq b \tag{18}$$

These linear constraints were then extended to each step in the time horizon.

$$C_{cone} = \left[\begin{array}{ccc|ccc} A_1 & & & 0 & \dots & 0 \\ A_2 & & & & & \\ & \ddots & & \vdots & \ddots & \vdots \\ & & A_1 & & & \\ & & A_2 & 0 & \dots & 0 \end{array}\right]$$

$$l_{cone} = \begin{bmatrix} \infty \\ \infty \\ \vdots \\ \infty \\ \infty \end{bmatrix}, u_{cone} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_1 \\ b_2 \end{bmatrix} \tag{19}$$

A limitation with the cone constraint came when the car needed to drive around a corner that was not yet visible to the lidar. This would result in the local trajectory not being followed fully, and the car riding along the side of the cone constraint that allowed the cost to be minimized, but the desired trajectory was not taken. Also, since the path was already checked for collision, we decided to deactivate the constraint.

8) Skid Constraints: The final constraint type implemented was skid constraints. These limit the steering angle as speed increases or vice versa as an affine constraint based on inputs. To simulate the model used in the simulator the code from the simulator was converted to Python. An array of input velocities and steering angles was generated, and the subsequent steady state steering angle was calculated using the model. The results of this can be seen in figure 7.

An allowable slip angle was determined, and a two-dimensional slice of the surface was plotted. This plot is a curve, so a linear approximation was drawn between the two end points of the plot. In figure 8 the selected slip angle is 0.3 radians.

The two points from this line are used as inputs into the ROS parameter file and four constraints are generated creating an allowable parallelogram in the constraint space of steering and velocity. The constraints are parallel because the limit must be met for positive and negative steering angles as well as

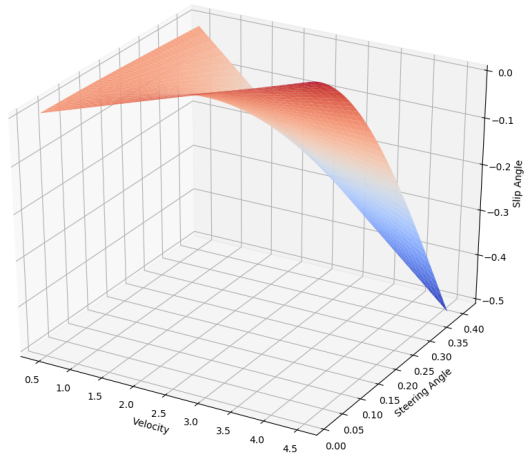

Fig. 7. The Slip Angle as a Function of Velocity and Steering Angle

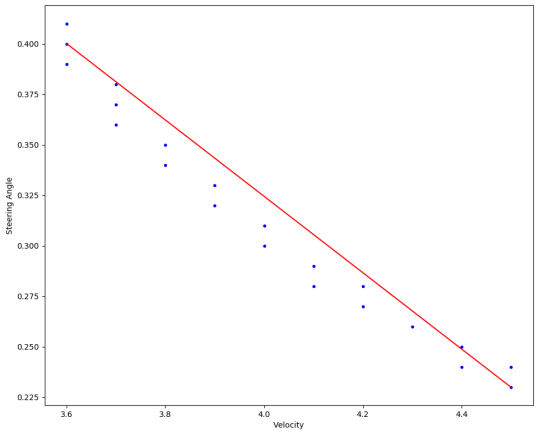

Fig. 8. Allowable Steering Angle vs. Velocity for a Slip Angle of 0.3 Radians

velocities. Because these constraints are parallel they can be represented with two affine functions if both upper and lower bounds are used. These constraints are added to the bottom of the linear constraint matrix.

$$C_{cone} = \left[\begin{array}{ccc|ccc} 0 & .. & 0 & A1 & & \\ & & & A2 & & \\ \vdots & \ddots & \vdots & & \ddots & \\ & & & & & A1 \\ 0 & .. & 0 & & & A2 \end{array}\right]$$

$$l_{skid} = \begin{bmatrix} b_{min,1} \\ b_{min,2} \\ \vdots \\ b_{min,1} \\ b_{min,2} \end{bmatrix} \quad u_{skid} = \begin{bmatrix} b_{max,1} \\ b_{max,2} \\ \vdots \\ b_{max,1} \\ b_{max,2} \end{bmatrix} \tag{20}$$

Different limits were tested, but for the final race the limit was set to the point where the car could be at full steering and full speed. While the safety of the vehicle would technically be compromised without these limits, the local trajectories were already selected to be dynamically feasible. This constraint could be useful as more difficult trajectories such as those from the output of a non-splined RRT* are loaded into the desired MPC states.

Result

The MPC formulation proved to be effective in the race as well as in general obstacle avoidance. A visualization of the the output of MPC can be seen in figure 9. The blue line represents the car's location and orientation, the red line represents the commanded throttle, and the black line's angle with respect to the blue line represents the steering angle. This visualization is sparse and does not show every state, but allows the user to see where the agent believes it will travel with the given inputs.

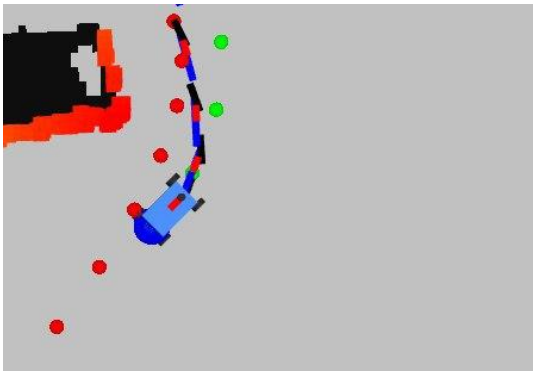

Fig. 9. Visualized MPC Trajectory

D. Other Approaches

We pursued various other approaches in parallel to our main approach. While these approaches were not used in our final implementation, we felt that these approaches could be useful starting points for future implementations.

a) RRT:* Our Plan B was to use Rapidly-exploring Random Trees* (RRT*) [5] with Pure Pursuit for solving the problems discussed in Section I. From the RRT lab assignment, we observed that the main shortcoming of sampling based trajectory planning approaches is that they do not take the dynamics of the vehicle into account, resulting in jerky and often infeasible trajectories. The steering angle dependent velocity profile needs to be carefully chosen so the trajectories can be closely followed by the vehicle. Moreover, the random nature of RRT* implies that the current trajectory can switch very rapidly between consecutive scan updates, making it difficult to track. Lastly, RRT* will not generate optimal trajectories at the low number of iterations required to run the algorithm in real-time. We tackled each of these issues independently in our modified version of RRT*.

For solving the issue of rapidly changing trajectories, our RRT* implementation stops updating the path once a feasible path has been found within the specified number of iterations. This path is followed using pure pursuit until either 1) the current path is no longer collision-free due to updates of the

occupancy grid from new scan data or 2) the agent is close to the last point of the current trajectory.

The generated trajectories may still not be suitable for tracking due to the kinks generated between each pair of consecutive line segments in an RRT* path. We used the C++ `xtensor-interpolate` package for fitting a regularized cubic spline between the points of the RRT* trajectory. As we can generate the interpolated trajectory with a large number of points, the pure pursuit tracker can track it much better than the original RRT* path.

As mentioned above, RRT* trajectories are typically sub-optimal in path length for smaller number of iterations (for faster performance). To improve the optimality of the RRT* trajectory, we implemented the randomized shortcutting approach described in [4]. Although this algorithm is designed for reducing the path length of RRT* trajectories in the higher dimensional configuration space of manipulators, this approach also works well for these 2D RRT* trajectories. An example of our improved RRT* trajectory in action is in Figure 10.

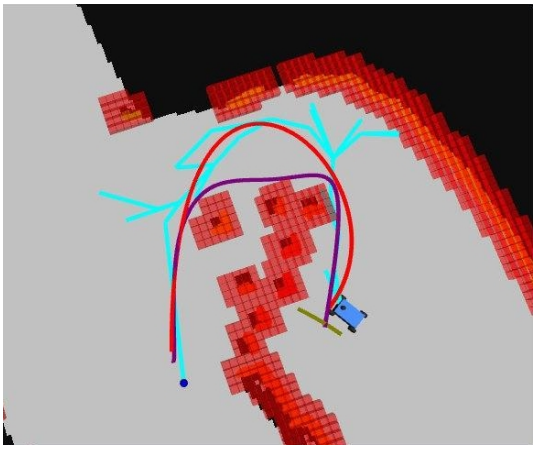

Fig. 10. Improvements to RRT* - Cubic spline interpolated RRT* trajectory (Red) and corresponding shortcut trajectory (Purple)

Result

To improve the tracking accuracy, we attempted setting the RRT* trajectory as the reference trajectory for the MPC subroutine. However, the agent's tracking performance of the RRT* trajectory using our MPC implementation was not stable enough to be used in the final project.

b) Opponent Prediction: Predicting the position of the opponent at a future point in time is essential for developing a good overtaking trajectory. In this case, we had access to the opponent's current location and velocity on the map during the race. Conventional approaches to opponent prediction use a weighted sum of the linear model (assume opponent continues on its current bearing with the current velocity) and pure pursuit model (assume opponent follows a pre-defined trajectory). We felt that this approach would be inaccurate as we had no way of knowing the raceline an opponent agent would be following in advance. Hence, we investigated using a Gaussian process model trained on a combination of optimal and sub-optimal trajectories. These trajectories were generated by injecting Gaussian noise in the CMA-ES subroutine.

Fig. 11. Trajectories used for training the Gaussian process model

Gaussian processes are stochastic processes which are trained on the training data to create a multivariate normal distribution. Similar to K-NN or kernel regression, a Gaussian process predicts a value by measuring the similarity of the query with points in the training data. Gaussian processes can also return the uncertainty of the prediction with the predicted value.

For predicting the opponent's position in the future, we followed the approach described in [3]. We trained two separate Gaussian process models to predict $\frac{dx}{dt}$ and $\frac{dy}{dt}$ given the opponent agent's current position (x_k, y_k). t refers to the knots in the generated splines in Figure 11. Once we have predicted the derivatives of x_k and y_k with respect to t, we can find the heading of the opponent car by dividing these terms to get $\frac{dy}{dx}$. Assuming the magnitude of opponent agent's velocity remains constant, we can use the predicted heading to estimate its next position (x_{k+1}, y_{k+1}). We can then repeat this procedure for predicting the position of the opponent n steps into the future by using the newly estimated (x_{k+1}, y_{k+1}) as the query to our Gaussian process model.

The benefit of using a Gaussian process model in this application is that we can adjust the weighting of the Gaussian process model predicted trajectory and the linear velocity model depending on the confidence of the Gaussian process model in its prediction. For instance, if the Gaussian process model has a has a very low confidence in its prediction, we can switch to predicting only using the linear velocity model.

Result

We found that the Gaussian process model was able to make real-time predictions of the trajectory of the default opponent agent with good accuracy upto 2.5 seconds in the future. At 4.5m/s, an agent can cover almost half a lap in 2.5 seconds, making the duration of predictions more than sufficient for our purposes. However, this approach had two main drawbacks which made it difficult to integrate with our planner - 1) the Gaussian process model requires nearly 3GB of RAM at runtime for prediction using the 500 splines we trained it on and 2) the opponent agent's trajectory must be roughly similar

to the splines used for training. Predictions became very inaccurate whenever the opponent was out of the regions of the splines in Figure 11. Nevertheless, this approach can provide an interesting alternative to weighting the linear velocity and pure pursuit model for a single trajectory. Gaussian Process Regression Model conditioned on the previous states of the opponent can also add to more accuracy of prediction.

IV. Other Results

A. RRT* vs Model Predictive control approach

We used RRT* at much safer speeds (since time was not a part of evaluation) as our approach for the milestone 3 which was a safety evaluation based on collision frequency for 10 roll-outs of two laps for 4 maps with increasing difficulty. We then compared our result of using RRT* with the later developed MPC approach which was used in the final race. The following are results:

	RRT*		MPC	
Map	Time (2 laps)	Success rate (out of 10 rollouts)	Time (2 laps)	Success rate (out of 10 rollouts)
1	41.76	10	12.49	10
2	37.77	10	12.72	10
3	55.36	10	20.60*	10*
4	50.13	6	13.09	10

**The map 3 results for MPC do not correspond to the exact final submission. The final submission had a collision check which prevented selecting a path leading to the other side of the wall which had to be disabled to get this result. Otherwise, the car did not complete any lap for map 3.*

B. Robustness analysis for drift control

Since our model was based on Kinematic Vehicle Model, it did not account for drift. To accommodate that, a skid constraint was added as described in section III-C8. We performed experiments to test the effectiveness of this constraint for minimizing drift. The goal was to allow the MPC formulation to prevent the agent from getting into situations where the model was no longer accurate. We track the average error of sum of absolute difference between the MPC prediction of the next state and the actual next state. The following are the results for the performance (total time) and the average error, total error for two laps:

Friction Coeff.	Without slip constraint			With slip constraint		
	Total Error (m)	Total Time (s)	Average Error (m/s)	Total Error (m)	Total Time (s)	Average Error (m/s)
0.3	24.72	13.47	1.83	23.50	16.10	1.46
0.4	21.57	12.72	1.69	20.10	15.21	1.32
0.523 (default)	20.52	12.47	1.64	19.15	14.56	1.31
1	16.80	11.81	1.42	17.28	13.80	1.25

The slip constraint for the above results was calculated by approximating the steering angle versus velocity linearly for the slip angle of 0.1. As seen from the results above, the average error i.e. the prediction of the model vs actual outcome is consistently better when the slip constraints are used. Although, they were deactivated for the race as the total time for two laps is much lower without the slip constraint as it did not make the solution reduce speed in order to minimize drift. However, on applying the slip constraint, the drift is reduced but at the expense of higher lap times.

V. Conclusions and Future Work

The MPC formulation described in this report turned out to be extremely effective when compared to previous control attempts. While it seems robust in simulation based on varying friction coefficients, the most exciting thing to test would be the MPC controller on the real car. While the robustness of the controller can be tested in simulation with many different variables, it is unclear what settings would really show how the car would react in the real world. It would be very interesting to know if this simple model formulation could effectively navigate the real vehicle.

If the real vehicle wasn't able to navigate effectively with the current model, the next thing to test would be more complex models that could account for vehicle dynamics beyond the no slip mechanics that were implemented.

Through the course of the class many different methods were attempted to make the agent traverse a course faster and more effectively. The more naive methods at the beginning didn't require any sort of prior knowledge while later methods began to rely more heavily on the knowledge of some sort of course. In the future it would be interesting to create a global planner that could take advantage of the robust controller described in this report and use it to navigate a course without any prior knowledge of the environment, whether it was using a simple follow the gap method, a complicated locating and mapping method, or an amalgamation of any method covered in the class.

References

[1] Alexander Domahidi Alexander Liniger and Manfred Morari. Optimization-based autonomous racing of 1:43 scale rc cars. November 2017.

[2] Chris Manzie Denise Lam and Malcolm Good. Model predictive contouring control. December 2010.

[3] Sepideh Afkhami Goli, Behrouz H. Far, and Abraham O. Fapojuwo. Vehicle Trajectory Prediction with Gaussian Process Regression in Connected Vehicle Environment. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2018-June(Iv):550–555, 2018.

[4] Kris Hauser and Victor Ng-thow hing. Fast Smoothing of Manipulator Trajectories using Optimal Bounded-Acceleration Shortcuts. pages 3–8.

[5] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, abs/1105.1186, 2011.

[6] Achin Jain Joseph Auckley Kim Luong Matthew O'Kelly, Hongrui Zheng and Rahul Mangharam. Tech report: Tunercar: A superoptimization toolchain for autonomous racing. September 2019.

[7] PhD Thesis Pedro Lima. Optimization-based motion planning and model predictive control for autonomous driving. December 2018.