Adaptive Planning with Model Predictive Control

Kausik Sivakumar
M.S.E. Robotics
University of Pennsylvania
Philadelphia, USA

Shubham Gupta
M.S.E. Robotics
University of Pennsylvania
Philadelphia, USA

Ruchi Gupte
M.S.E. Robotics
University of Pennsylvania
Philadelphia, USA

***Abstract*—When considering the task of racing, it is important to consider the multiple factors and conditions the car would be racing in along with a robust racing strategy that is able to satisfy the constraints and variability of the environment. Our project focuses on combining two main strategies that account for both the race-line and how it should be temporarily modified to outmaneuver an opponent. We propose a strategy where the car follows a baseline trajectory, which we plan to generate before the race which gives us ideal x,y,velocity, and yaw inputs. In the case an opponent is detected to be in front of the car, the car's focus is then shifted to local planning where multiple potential trajectories are generated and an alternate best trajectory is chosen. This trajectory is tasked with overtaking the opponent or obstacle. The ideal trajectory is selected by first filtering invalid trajectories and then selecting a valid trajectory which achieves maximum progress along the original raceline. After overtaking, our car then resumes its set baseline trajectory till the opponent is detected in front of it again. In-order to closely follow both the local and baseline and efficiently switch between the two, we incorporte a Model Predictive Control algorithm with a kinematic model. With this method we are able to follow our baseline trajectory for the maximum duration while only computing other trajectory options and taking account of the opponent when they are in the car's field of view.**

***Index Terms*—Adaptive Trajectory Planning, Opponent Overtaking, Model Predictive Control, Obstacle Avoidance**

I. Introduction

Our problem statement is to find an on-the-fly trajectory that is best able to outmaneuver an opponent in front of us and figure out a good switching trajectory that is able to smoothly switch from a baseline trajectory to a local one and vice versa. This is because in the case the opponent is not in our field of view, following a baseline trajectory which could be an optimal one is the most ideal case for completing a race as fast as possible. Since the baseline trajectory can be generated offline, it can be tuned to accurately follow the optimal raceline and take consistent laps around the track. The drawback of this offline trajectory is its lack of adaptability to opponents and newly introduced obstacles. This is where local planning is able to excel, as it not only considers progressing along the raceline, but is also able to account for dynamic changes in the environment. The drawback of always following the local planner is that it tends to follow the baseline loosely and thus might never achieve optimal race times over long periods of time due to inconsistency in best trajectories and limited horizon. This is why a method that is able to combine both these concepts and achieve a smooth transition helps eliminate the drawbacks of of these individual methods.

A smooth transition is achievable only when there is a control algorithm in place that considers both the cars current state and future states in real time. By considering future states, the car is able to smoothly return back to the original baseline over a certain number of time steps rather than directly aiming towards a certain point on the baseline which prevents jerky and rough motions. To achieve this a Model Predictive Control algorithm is one of the most efficient ways of planning for multiple time steps with consideration to the vehicle kinematics and feasible trajectories. This way, when switching from a local trajectory that is away from the baseline to the baseline trajectory itself, the MPC will take a smooth transition trajectory that is able to converge onto the baseline over the next few time steps. Other simpler control algorithms such as Pure Pursuit would cause more aggressive jerks which might cause loss in stability and time.

This paper details the methods which show how the control algorithm is structured with its costs and constraints and the various considerations taken to incorporate it with an adaptive strategy. We will also discuss various optimisations and methods we have used to simplify the planning and trajectory selection process and how such a system is viable for racing competitively and is able to overtake opponents and avoid obstacles at speeds upto 4m/s.

II. Final Race Comparison and Changes

A. Control and Trajectory

Trajectory Generation: Trajectory generated is done by the TUM trajectory generator. We used the minimum curvature approach to optimize the trajectory

1) **Final Race:** In final race, we used Pure pursuit with a look ahead distance for control. The steering angle is just set as proportional to the curvature of the point tracked. For every point in the trajectory, the optimal raceline defines an optimal velocity. Thus, the input velocity to the car is the same as that from the raceline. Another hyperparameter to tune in Pure Pursuit is the look-ahead distance. On the curves, high look ahead distance causes collision with the walls. On the straights, low look-ahead distance results in jerky motion. Thus, we set the look-ahead distance to be a linear function of the curvature defined by the car's current position on the raceline. We defined the equation of the line such that high curvature results in a low look-ahead distance and low curvature results in a high look-ahead distance.

2) **Final Project:** Pure pursuit only follows a particular point at every time step. It doesn't consider the kinematics of the car. This results in non-smooth trajectories. Thus, for our final project we decided to explore the idea of Model Predictive Control which considers kinematics of the car while aiming to track multiple points in the trajectory resulting in smoother trajectories. The cost function and constraints in the MPC formulation are explained in Section III.

B. *Obstacle Avoidance*

1) **Final Race:** Obstacle avoidance on the final race was achieved by gap-follow. Basically you detect an obstacle by thresholding the LIDAR scans to lie within 20°around the x axis of the car's frame. This means that we threshold the LIDAR scan to lie 10°to the left and 10°to the right of the longitudinal axis of the car. An obstacle is set as detected if the LIDAR ranges data that lie within this threshold is below 1 meter. If this is so, the control scheme of the car immediately switches from pure-pursuit (which follows the optimal raceline with a look ahead) to a follow-the-gap algorithm with a disparity extender approach. This makes the local plan maneuver to just follow the maximum gap with the virtual disparity map as obstacles. Follow-the-gap algorithm looks for the maximum gap and tries to follow the center of that gap. This results in slower laps as the maximum gap is not the optimal way to overtake an opponent. Thus for our final project we switched to a local trajectory planner.
2) **Final Project:** In the final project, the obstacles are detected by the same approach as above. However, the difference is in the way we overtake an opponent. For our approach, we decide to switch to a local planner whenever an obstacle is detected. We pre-compute an offline spline map, which is the planning algorithm in our hierarchical planner approach. This planning algorithm uses offline generated splines, which are a roll-out of a particular chosen car velocity and a heading angle over a fixed time horizon and time-step gap (this is set in our case as 20 time steps, each of 0.02 seconds long). We use the bicycle kinematic model to roll-out the trajectory that are feasible to the car model. These splines are transformed to car's coordinates (using the Odom data published by the particle filter. Once we have n splines to choose from, we sub sample valid trajectories from here that don't collide with any obstacle in its horizon after using the disparity extender (for safety distance). Once we have a set of valid trajectories that don't collide, we just use Model Predictive control over the selected spline as the reference trajectory and optimize the inputs corresponding to good tracking (least cost in MPC). This is more efficient in theory and can smoothly traverse thought sharp turns and edges, since we explicitly give it a trajectory to follow for a certain number of time steps. This does not work well at high speeds in the car because of the reasons explained in the Section VII C.

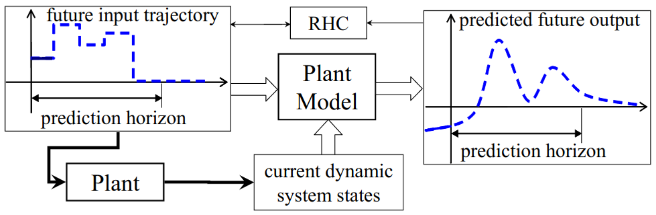

Fig. 1. Model Predictive Control

III. Model Predictive Control

A. *Concept*

Model predictive control is one of the best control strategy available. Hence, we used this state of the art approach. At each time step, the MPC computes control by solving an open-loop optimization problem for the prediction horizon. We apply only the first control action, update the initial state and redo the open loop optimization again. So to put it in brief, we plan for a particular receding horizon (meaning we plan the control action to follow the particular state trajectory, not only the next state). Once we have it we demonstrate only the first control action and observe the state the object reaches due to this action. We again set the initial state as the state the object reached here and redo the optimization step for a trajectory of receding horizon length. The optimization it solves is to follow the trajectory for a receding horizon as close of possible

$$
\begin{aligned}
min \sum_{t=0}^{T-1}((z_t - z_{ref_t})^T Q(z_t - z_{ref_t}) + u_t^T R u_t + \\
(u_{t+1} - u_t)^T R_d (u_{t+1} - u_t)) + \\
(x_T - x_{ref_T}) Q_T (x_T - x_{ref_T}) \\
s.t x_{t+1} = A x_t + b u_t \\
x_0 = \text{current state of the system} \\
x < X_{MAX} \\
u < u_{MAX}
\end{aligned}
$$

z is the state of the vehicle which is of 4 dimensions, namely the x-coordinate, y-coordinate, velocity and the yaw heading. The control (u) is of two dimensions, namely the steering angle and the acceleration input to the car. The model used by the car is the linearized kinematic bicycle model with the rear wheel as the concentrated position of object's center of mass

B. *Vehicle Kinematics*

We followed the kinematic bicycle model which is

$$
\begin{aligned}
x_{t+1} &= x_t + v cos(\phi) dt \\
y_{t+1} &= y_t + v sin(\phi) dt \\
v_{t+1} &= v_t + a dt \\
\phi_{t+1} &= \phi_t + v \frac{tan(\delta)}{L} dt
\end{aligned}
$$

Here δ is the steering angle and ϕ is the heading (yaw) of the car with respect to the world frame

IV. Baseline Trajectory Generation

Since the adaptive planning switches between a pre-generated offline trajectory and a locally generated one, we have the opportunity to create a baseline which is close to the optimal trajectory for that track. This gives us the chance to use computationally heavy but more efficient algorithms for baseline trajectory generation, and we can then tune the adaptive planner to minimize its time on the baseline trajectory and only switch to the local one incase of obstacles or major deviations of the car.
For this project, we used the TUM global optimized trajectory generator with a very conservative estimate of the track width just for tuning purposes. This is so that we could showcase and test our obstacle avoidance and overtaking from various positions of the track. This trajectory generation algorithm gives us a feasible and optimized trajectory given a centerline path and track width. It is able to do this by considering various factors such as max velocity, car width and mass, etc. We used the Minimum curvature method with multiple iterations and set a conservative track width on the left and right side of the centreline as 0.3 m. This gives us an optimized trajectory which does not deviate from the centerline by too much, but optimizes the yaw and velocity values to the most ideal versions for the given path.
Figure 2. gives us the optimal raceline generated for the Final Race with a conservative estimate for track width. Because of this, when applied to the actual track, there is much more space for overtaking from both sides so that when tuning we can control and adjust the algorithm to take a turn from the ideal side of the opponents car.

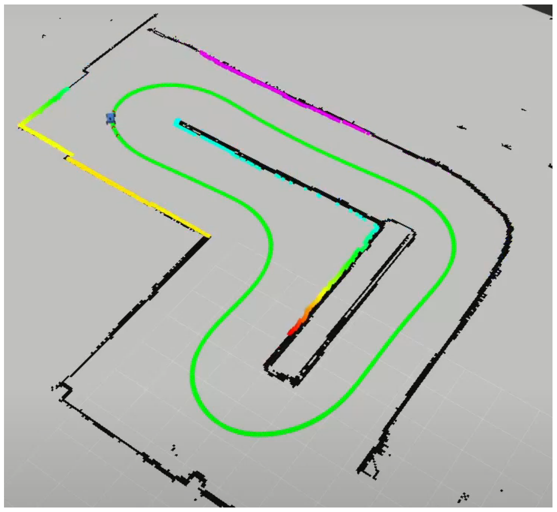

Fig. 2. Baseline Trajectory from TUMN Algorithm

The MPC is tuned to follow this trajectory without any deviations in normal circumstances without any obstacles and can complete the track quickly and consistently. If the conservative 0.6 m total track with is increased to its actual track width values, we would then get an optimized global trajectory which is feasible and traversal though the MPC. This way we would be able to get an ideal trajectory for racing while still accounting for obstacles using the local trajectory generation when needed.

V. Local Trajectory Generation

Given a set of trajectories in the car's coordinate frame, generated before hand, we are able to find which of these local trajectories are valid and the best trajectory that maximizes progress along the track. The full algorithm to get the best local trajectory is given below in Algorithm 1.
The algorithm itself consists of 3 main steps: Offline trajectory generation to map the potential trajectories, Occupancy check to remove trajectories that are blocked by obstacles and finally the best trajectory selection based on maximum progress along the track. All the 3 steps are explained in the subsequent sub-sections. At the end we are given one best possible trajectory for the car with its x, y, yaw and velocity data given to the MPC as a reference to follow.

Algorithm 1 Local Trajectory Algorithm

```
$trajectories \leftarrow trajectory, points, [x, y, v, yaw, r, \theta]$
$best\_progress \leftarrow 0$
$best\_trajectory \leftarrow None$
**for** traj in $trajectories$ **do**
    $wrt\_car \leftarrow traj \leftarrow points, [x, y, v, yaw, r, \theta]$
    $r, \theta \leftarrow wrt\_car[:, -2:]$
    $theta\_idx = \frac{max\_angle}{step\_size} + \frac{\theta}{step\_size}$
    $is\_occupied = lidar\_ranges[theta\_idx] < r$
    **if** any is_occupied is True **then**
        continue
    **else**
        $wrt\_world \leftarrow to\_world\_coord(wrt\_car[:, : 2])$
        $last\_pt \leftarrow wrt\_world[-1]$
        $dist\_arr \leftarrow norm(ref\_trajectory - last\_pt)$
        $progress = min\_idx(dist\_arr)$
        **if** progress greater than best_progress **then**
            $best\_progress \leftarrow progress$
            $best\_trajectory \leftarrow traj$
        **end if**
    **end if**
**end for**
```

A. *Offline Spline*

Offline spline is generated by rolling out vehicle dynamics for 20 steps (chosen here) where each step is for 0.02 seconds. The model chosen is the kinematic bicycle model. We have sampled velocities at 1m/s and 2m/5 for an angle ranging from -15°to +15°with respect to the longitudinal frame of the vehicle

The below Figure 3. represents one such trajectory that is generated by the algorithm with angle θ and a step size of n with each step being 0.02 seconds. By using the vehicle dynamics we can ensure that all trajectories generated are feasible and compatible with the inputs for the MPC and

Algorithm 2 Offline Spline Generation Algorithm

```
$angle\_array \leftarrow [-15, -10, 0, 10, 15]$
$v\_array \leftarrow [1, 1.5, 2]$
$t_s teps \leftarrow 20$
**for** v in $v\_array$ **do**
    **for** phi in $angle\_array$ **do**
        **for** $t\_steps$ **do**
            $x'(t) \leftarrow v \times cos(phi)$
            $y'(t) \leftarrow v \times sin(phi)$
            $v'(t) \leftarrow a$
            $phi'(t) \leftarrow v \times tan(delta)/L$
            $traj[i] \leftarrow [x'(t), y'(t), v'(t), phi'(t)]$
        **end for**
    **end for**
**end for**
```

follow a similar format to the original reference trajectory for ease of integration.

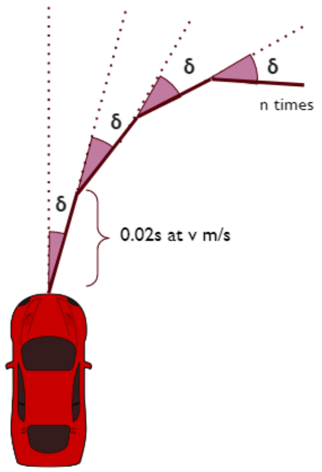

Fig. 3. Offline Spline Generation

B. Occupancy Check

Occupancy check works with data taken from the LIDAR scan data to check if all the trajectories lie within the cars valid travel space. We first take each point in a given local trajectory and convert it to its polar coordinate form. This will be in terms a series of given radius corresponding to a given θ with the car at the origin. This can be matched with the current LIDAR scan ranges which also follow a similar notation.

It is to be noted that the car's LIDAR scan image follow the an array of 0 to n range values which correspond to angles between -130 and 130 degrees, while the local trajectory θ represents the deviation of the car's yaw from the original 0 degrees, and can be positive or negative. To find the corresponding max range value in the LIDAR Scan from a given θ, the following equation is applied to all points:

$$theta\_idx = \frac{max\_angle}{step\_size} + \frac{\theta}{step\_size}$$

Where max_angle is the maximum angle of the LIDAR Scan array and $step\_size$ is the angle increments between each value in the scan, both of which can be obtained in the LaserScan message alongside the range values.

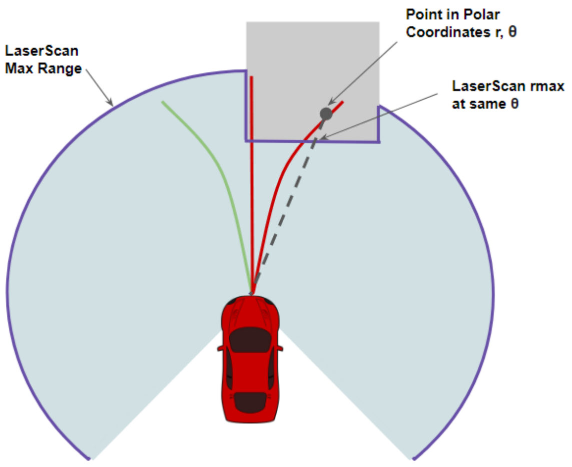

Fig. 4. LaserScan Occupancy Check

Thus by indexing $theta\_idx$ in the LaserScan array we obtain the max range value obtained by the LIDAR for that particular θ. If the max range value is less than the r of the local trajectory point then we know that that point is occupied and thus the trajectory is invalid. This is applied to all the points of all potential trajectories and the invalid trajectories are removed.

Figure 4. Shows us the process by which each trajectory point is mapped and compared to the max theta and how when a the max range from the LIDAR is smaller than the given r, that means the point is occupied due to an obstacle.

C. Best Trajectory Selection

Once the invalid trajectories have been eliminated, we need to find the best trajectory to follow out of all the feasible and valid trajectories. From a set of feasible local trajectories, the best trajectory is the one the results in the maximum progress along the raceline. Progress is measured by how far the car has moved along the raceline.

For each local trajectory, we see how far the car would travel if it followed that particular trajectory. So, we select the last point of all feasible trajectories. Then for each of these last points, we find the closest point(euclidean distance) on the raceline trajectory. These closest points define the progress on the raceline trajectory which is just the index of where they lie on the trajectory. The local trajectory that has the maximum progress is then selected. This can be seen in Figure 5. where the green path is decided as the best trajectory since it is a valid path which has maximum progress along the baseline trajectory.

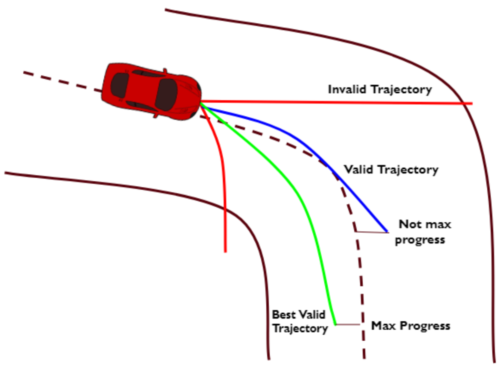

Fig. 5. Maximum Progress Along Baseline

VI. Integration

A. Trajectory Switching

As long as there is no obstacle, the most optimal trajectory for the car to follow is the raceline. Once an obstacle is detected, we switch from the raceline to the best local trajectory to follow. The local trajectories have x,y,yaw,velocity in the car's local frame. We sample equidistant points on these trajectories based on the planning horizon. These points are converted into the world's coordinate frame by a rigid body transformation. The transformaion matrices can be found out by the car's current position and yaw(given from Odometry in simulation and Particle Filter on the car). For the MPC, the trajectory to track is now these local transformed points. A control action is calculated and executed based on this. If the opponent/obstacle is no longer in the field of view of the ego-car the next reference points are created from the raceline for the MPC to follow

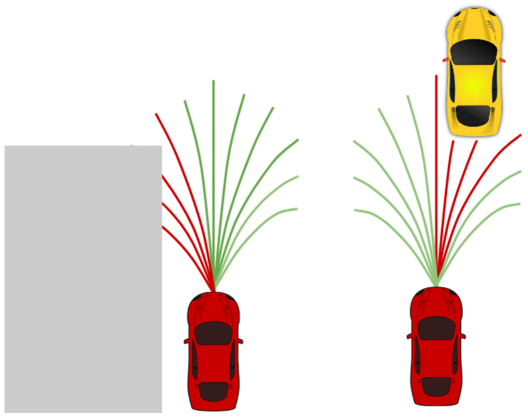

Fig. 6. Obstacle Detection

B. Obstacle Detection

An obstacle is only detected when it's in front of the car. We set a hyper-parameter theta which considers the FOV in front of the car. For our implementation, we check for +20 and -20 degrees in front of the car. If any of the LIDAR ranges in this FOV returns a distance that is less than the threshold distance(2m for our implementation), an obstacle is detected. Following this, we drop following the optimal race line and switch to following the optimal local trajectory.

Figure 6 shows that when an obstacle is detected in the vicinity, the planner switches to the local trajectory and eliminates those trajectories that are in the way of the obstacle.

C. Disparity Extender

Once an obstacle is extended, the infeasible local trajectories colliding with the obstacle need to be eliminated. We use the disparity extender approach to consider the width of the car to make sure the trajectories that are half-width away from the obstacle are infeasible.
By setting this safety width to be higher, we can, to a slight extent, account for the potential dynamic behavior of the obstacle.

VII. Results

A. Simulation

Our car was able to perform efficiently and consistently in the simulations and made no deviations from the actual path when following just the baseline trajectory. Video 1 in Section VIII Demonstration shows the car traversing the Final Race map with no obstacles using MPC. In this case since no obstacles are detected, the car stays on the baseline trajectory and is able to complete the track efficiently.
On the other hand when the car's obstacle detection parameter is set to be too sensitive or the car is set to local trajectory only, it is still able to complete the track, but the motion and path is not optimal. Video 2 in Section VIII Demonstration shows the car following just the local trajectory. We can see here that it is still able to able to complete the lap based on maximum progress local trajectory as shown in Figure 7. However, the path traversed by the car is not optimal

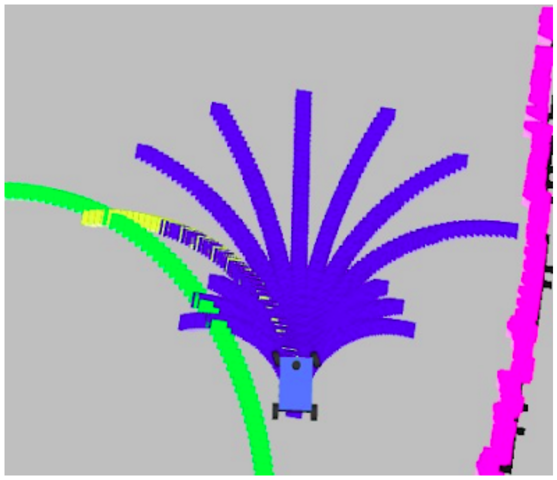

Fig. 7. Simulation: Best Trajectory Selection

The obstacle avoidance also works well for both dynamic and static obstacles, due to the various parameters set for switching and safety bubbles. Figure 8 shows how an opponent being in front of the car triggers the local trajectory planner to select a deviating path to avoid the obstacle while still trying to maximize the progress.

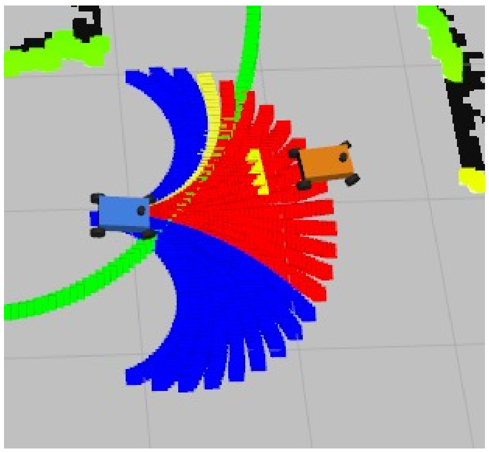

Fig. 8. Simulation: Obstacle Avoidance

In a similar case, in Figure 9, where the opponent is in front of the car, but does not obstruct the baseline trajectory, the car is correctly able to not deviate and take the best possible trajectory for the given case. It is still able to detect the opponent and remove the occupied or unsafe trajectories, but maintains its path since the opponent is not in the way.

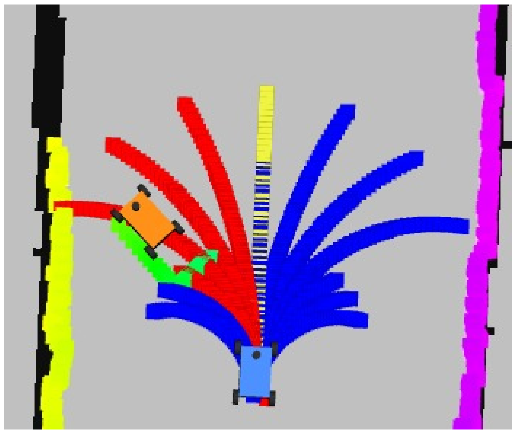

Fig. 9. Simulation: Obstacle Detection but No Avoidance Needed

Combining all these factors Video 3 in Section VIII Demonstration shows how the car efficiently switches between local and baseline trajectories with a slow moving dynamic opponent, and Video 4 shows how the system can be tuned to perform well for higher speeds with more complex and faster opponents.

B. Real Car Implementation

The implementation on the real car was able to still perform dynamic obstacle avoidance and traverse the Levine map successfully, but the speed of the car had to be significantly dropped to obtain satisfactory performance.
For Obstacle detection, Figure 10 shows how when a person/obstacle comes in front of the car, the occupied trajectories instantly turn red and selects the best local trajectory which avoids the obstacle in its way. One thing to note here is that due to the restricted space available at Levine map, even trajectories on the right with no obstacles and only closely packed walls are considered obstacles due to the track width of less than 4m. Video 5 in Section VIII Demonstration shows this detection in real-time.

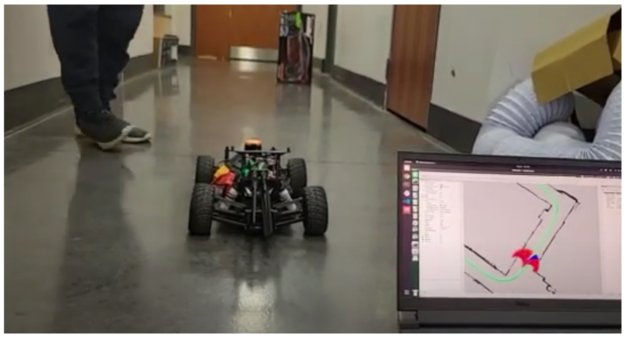

Fig. 10. Actual Car: Obstacle Detection

Video 6 in Section VIII Demonstration shows one lap of the car in Levine map, where it is successfully able to avoid static obstacles while still traversing the map. To test the performance of the car on dynamic obstacle we moved boxes in real time to actively bock the car, and the car was successfully able to avoid and overtake the dynamic obstacle without crashing. This is shown in Video 7 in Section VIII Demonstration.

C. Sim to Real Gap

For every time step we need to solve an optimization problem. This reduces the update frequency for the control action which makes it harder to implement the algorithm on the car. On the simulator, the algorithm brings down the pose update rate from 250 Hz to 20 Hz. Still this works pretty well on the simulator and the car is able to avoid obstacles as well follow the optimal trajectory when there are no obstacles. We were able to follow exact velocities given by TUM raceline on the simulator.
On the real car, the pose update rate is around 20 Hz. The algorithm brings down the control update rate to 2-3 Hz. This results in considerable lag in the control action and thus we were only able to run the car at 1m/s.
The MPC is also not able to track the trajectory well at higher speeds due to the noise from the particle filter, which can cause crashes and non-optimal movement.

VIII. Demonstration

Below are the videos of the actual performance of the car in simulation and on the actual f1/10 car.

1) Video 1: Simulation: Pure Baseline with MPC
2) Video 2: Simulation: Pure Local with MPC
3) Video 3: Simulation: Adaptive Planning with MPC, Slow
4) Video 4: Simulation: Adaptive Planning with MPC, Tuned
5) Video 5: Actual Car: Local Trajectory Selection
6) Video 6: Actual Car: Levine Full Lap
7) Video 7: Actual Car: Dynamic Obstacle Avoidance

IX. Conclusion

Our given project was efficiently able to detect and avoid dynamic and static obstacles while still completing the lap in simulation and the results on the actual car were promising as well. Given better processing speeds, we should be able to develop a competitive algorithm which can adaptively plan for obstacles and the race track. Our approach sought to combine the benefits of following a precomputed trajectory while still anticipating and planning for obstacles introduced onto the map. Using MPC as a control algorithm helped with following either the global or local trajectories efficiently as it considers minimizing cost over multiple time steps. When developing such switching algorithms, it is important to tune and set robust hyper parameters for the planner as it may lead to unnecessary switching or switching when it is too late to avoid an obstacle. Our project was able to correctly tune such parameters for both slower and faster speeds and for opponents with much more complex movements.

X. Future Scope

We strongly believe that different parts of our approach can be extended and improved in multiple ways. The local planning can be extended to generate smoother trajectories based on clothoids. We have not added the half-space constraints in the MPC formulation. Adding the half-space constraints as soft constraints can further make sure that the car doesn't collide with the walls. This means that we restrict the search space of the control problem to only those actions that could reach a feasible state (under this half space). The half spaces are defined by two affine inequality constraints per time step (one for the right and one for the left border), resulting in a convex feasible set. This could be extended to add obstacles so that the affine equation limits all points of the state in the horizon to lie within a bound that does not collide. For opponent overtaking, we can consider this opponent as an obstacle

$$F_k z_k <= f_k + s_k$$

This is an example of how half space constraint can be added for timestep k. s_k here is a slack variable which can be added to the cost in an inf norm of the cost, this is a decision variable that always help ensure feasibility.

Another direction to follow is to completely remove the local trajectories and let the MPC take care of the local trajectory generation while maximising progress along the raceline. In this approach the half constraints are defined dynamically based on the obstacles in the field of view. One of the ways to do that is through Model Predictive Contouring Control[2]. Right now, we consider the obstacle to be static at a particular time step. This is not optimal and the project can be extended to include opponent's trajectory prediction. This can be added to the MPC cost formulation to make sure the egocar's planned trajectory doesn't collide with the opponent's predicted trajectory.

XI. References

1) Jain, Achin Chaudhari, Pratik Morari, Manfred. (2020). BayesRace: Learning to race autonomously using prior experience.
2) Liniger, A., Domahidi, A., Morari, M. (2017). Optimization-based autonomous racing of 1:43 scale RC cars. Optimal Control Applications and Methods, 36, 628 - 647.
3) Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp Boris Lohmann (2020) Minimum curvature trajectory planning and control for an autonomous race car, Vehicle System Dynamics, 58:10, 1497-1527, DOI: 10.1080/00423114.2019.1631455