Project Report MEAM 523, Control In Robotics

Siddharth Singh [1] **Adarsh Modh** [2]

Abstract

Through this report, the authors propose a robust framework for autonomous highway driving using a hybrid control technique which takes care of two scenarios of overtaking and safe distance following. Given the rapid growth in Autonomous Vehicles in the domain of Robotics, it becomes extremely important that novel methods are targeted to solve problem statements which are most tractable to solve and slowly build up towards more complicated scenarios. In the proposed framework, one of the more tractable scenarios of highway driving is explored. The main pieces of the entire pipeline are the global path generation, local-path planning by RRT* and Trajectory tracking by Model Predictive Control. The tests are performed over Gazebo which is a robust simulator with an integrated physics engine capturing the dynamics of the vehicle.

1. Introduction

The World of Robotics is seeing a paradigm change with breakthroughs happening at a rapid rate. Both Academia and the Industry are in a race to build more robust and dependable frameworks to get Autonomous Vehicles on the road.

Highway driving is considered to be one of the more tractable problem statements that can be solved to lead towards Autonomous Driving. The organized driving scenarios have lesser congestions as compared to urban driving traffic conditions and smooth flow of traffic makes it an ideal scenario to implement hybrid controllers. However, the highway driving also comes with higher risks owing to high speed scenarios and thus safety concerns become an important aspect to be careful about.

In this report we propose a hybrid control methodology to perform maneuvers of overtaking and safe distance keeping in a highway driving scenario. The platform is built on top of a single Model Predictive Controller which changes it's behaviour depending on the environment. In the environment the vehicle monitors the dynamic and static obstacles in real time and takes decisions to overtake the vehicle in the front or maintain safe distance.

2. Problem Formulation

The proposed methodology targets two of the most commonly seen scenarios in highway driving:

1. Safe Distance Keeping
2. Overtaking

The scenarios both depend entirely on the obstacle vehicle infront of the ego vehicle and the behaviour which it follows. Given a posted speed for the highway, the ego vehicle can choose to either follow the obstacle vehicle or overtake it. The choice of these two behaviours needs to be robust so that the ego vehicle performs the maneuvers without any lag and risk of failure. A good way to achieve this hybrid behavior is to build a hybrid controller which switches its mode depending on the behavior of the obstacle vehicle. The following figure essentially depicts the decision making process that needs to be taken by the ego vehicle in order to perform the requisite maneuvers.

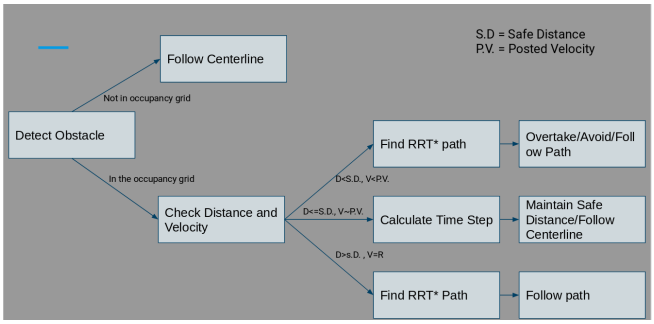

Figure 1. Decision Making Process for the AV

3. Pipeline

3.1. Center Line Formulation (Global Path)

Since the method that is proposed works in a local sense throughout, having a global reference path is essential to this method. A global path is used to get waypoints which become local goal points for local path planning framework

(in this case RRT*). Then the Path Planning Algorithm RRT* works in a local sense (in the ego-vehicle frame) to generate collision free paths. The global path basically has to mimic the road's(lane's) shape and curvature. So the easiest way to get global paths is to interpolate the points on the road that you want to be in. The road shape and curvature can be taken from any kind of map database which will provide you the path from the start to the goal in a global sense.

3.2. Detection and Occupancy grid Mapping

The detection is carried out from the LiDAR data. The LiDAR data reports obstacle position in the body frame of the car. The raw LiDAR data consists of distance of the obstacle from where the ray is deflected. These rays are uniformly distributed over the entire field of view of the LiDAR. This information is used to filter out the positions of the obstacles and their approximate size also. Once the positions of the obstacles are found in the body frame of the ego-vehicle, A binary occupancy grid is created. This basically means that the free space is marked as zeros and occupied grids are marked as ones. This occupancy grid has the car as its origin and is always created in a local sense. So in a way the occupancy grid moves with the ego-vehicle as it goes on along the path. Once we get the goal from the Center Line Formulation and the information (location) of the obstacle in the vehicle's frame we create an occupancy grid with the following properties:
1. Grid span : 20 X 20
2. Grid size : 1 X 1
3. Radius of obstacle (Minkowski Sum) : 2
4. Node Location : bottom left corner

3.3. Local Path Planning : RRT*

Once the occupancy grid has been created, the node must find an obstacle-free path from the cars position to the waypoint. We have opted for probabilistically complete methods of searching.
While A* is resolution complete for a given grid, its time complexity is O(N), while probabilistically complete methods such as RRT are O(N log N), allowing us to save significant time in each query. We are comfortable with a probabilistically complete approach because these are applicable to motion planning problems with robust solutions.
We ensure robustness of any path that we query because before the planner is even invoked, the waypoint is identified on the inflated occupancy grid, and if it is occupied space, an alternate point is searched for laterally. If an alternate point cannot be located laterally, a search downward towards the car is conducted. If even this point cannot be located, then the search is not invoked. Because our occupancy grid is created by raycasting, there is no chance for there to be a free island which could not be accessed. Further, when there is good visibility to the end point, the probability of a path being found goes to 1 exponentially fast.

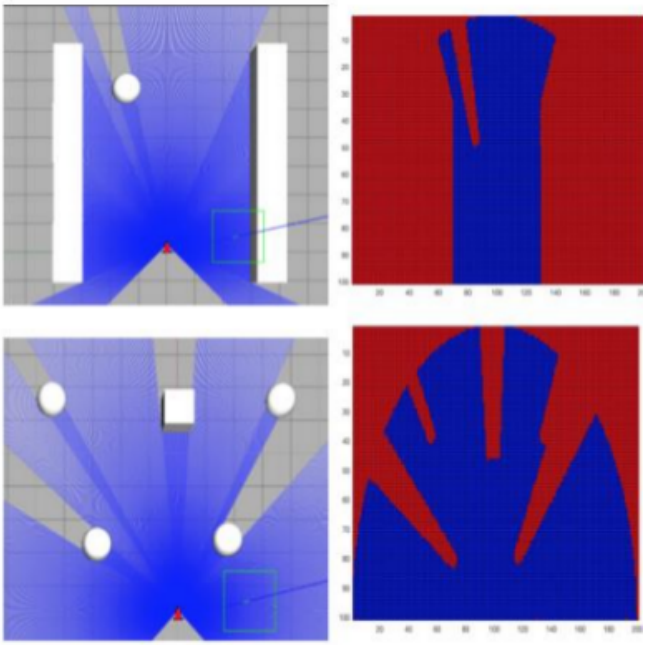

Figure 2. Laser Scan of AV in RViz

Further, we experimented with the maximum node distance between two subsequent nodes on the tree and were able to generate generally smooth and reasonably paths. Limiting this parameter prevents highly suboptimal paths from being returned.

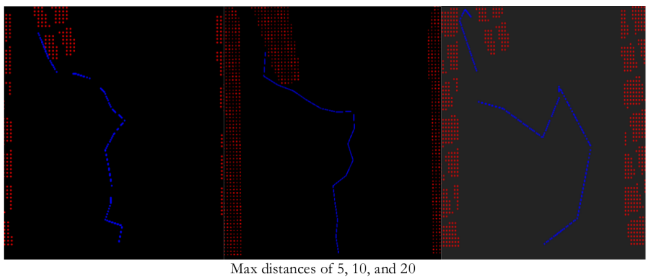

Figure 3. Paths created by RRT in Rviz

3.4. Trajectory tracking : Model Predictive Control

We have chosen a non-linear Model Predictive Controller and tried to compensate the non-linearity in the formulation of the optimiztion problem by confining the search space (feasible region) of the optimization problem and

using a linear programming optimization framework which is search based on the interior region.
The MPC cost function takes the following generic form:

$J = \sum_{i=1}^{N} w_{x_i} (r_i - x_i)^2 + \sum_{i=1}^{N} w_{u_i} \Delta u_i^2$
Where:
x_i : is the controlled variable
r_i : is the reference variable
u_i is the manipulated variable
w_{x_i} is the weight for the controlled variable
w_{u_i} is the weight for the manipulated variable

This cost functions then is optimized over a fixed horizon discretized into time steps.

3.4.1. The Vehicle Model

Model Predictive Control depends heavily on the choice of the model of the system that is being formulated and the choice of optimization that is being used. Model Predictive Control does provide the flexibility of choosing a relatively dumb model but it still would be beneficial to have a better model to provide better results.
In our formulation we have chosen to use a non-linear bicycle model which is one of the most popularly used models for vehicle kinematic analysis. Figure 4 depicts the model's view and the following equations encompass the kinematics involved.

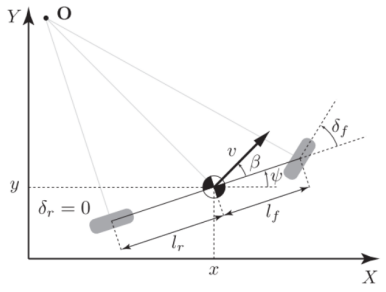

Figure 4. The Kinematic bicycle model

$x_{t+1} = x_t + v_t \cos(\psi_t)\, dt$
$y_{t+1} = y_t + v_t \sin(\psi_t)\, dt$
$\psi_{t+1} = \psi_t + \frac{v_t}{l_f} \delta_t dt$
$v_{t+1} = v_t + a_t dt$
where :
x_t is the x position at time t
y_t is the y position at time t
ψ_t is the vehicle orientation at time t
v_t is the velocity at time t

It is important to note that the cost function itself is designed in a way that we only estimate the velocity and the orientation of the vehicle from a point mass model and feed it into the bicycle model of the car to extract the desired steering angle.

3.4.2. The optimization Problem

Once we have decided the vehicle model, we then formulate the optimization problem to be solved.

Objective (cost) function :
$X = x_1, x_2, x_3, x_4, x_5$: orientation angles to be optimized
$x_0 = velocity$: velocity to be optimized ts

$$\begin{aligned}
\min Z = {} & (x_{goal} - Sum(x_0 * \cos(X) * ts)) && (1)\\
& + (y_{goal} - Sum(x_0 * \sin(X) * ts)) && (2)\\
& + (w_1 * (x_5 - x_4) && (3)\\
& + w_2 * (x_4 - x_3) && (4)\\
& + w_3 * (x_3 - x_2) && (5)\\
& + w_4 * (x_2 - x_1) && (6)\\
& + w_5 * x_1)) && (7)
\end{aligned}$$

Subject to Constraints:
$-0.418 < x_5 - x_4 < 0.418$.....1
$-0.418 < x_4 - x_3 < 0.418$.....2
$-0.418 < x_3 - x_2 < 0.418$.....3
$-0.418 < x_2 - x_1 < 0.418$.....4
$-0.418 < x_1 < 0.418$.....5
$\omega_{goal} - \omega_{current} < x_5 < \omega_{goal} - \omega_{current}$.....6
$0 < x_{goal} - Sum(x_0 * Cos(X) * ts) < 10$.....7
$0 < y_{goal} - Sum(y_0 * Sin(X) * ts) < 10$.....8

Constraints 1 through 5 are physical constraints on the vehicle's steering and constraint 6 acts as the terminal cost for the Optimization problem.
Constraints 7 and 8 are goal reaching constraints which ideally should be terminal costs but in our optimization problem we have noticed that making the constraint as a terminal cost leads to uncertain behaviour and infeasible solutions in the optimization problem.

In order for the Algorithm to solve the optimization problem gradients of the cost function and the jacobian of the constraints is also needed which are as follows.

$$
\begin{aligned}
Gradient(Z) = [&-Sum(x_0 * Cos(X) * ts \\
&- Sum(x_0 * Sin(X) * ts, \\
&- (ts * x_0 * cos(x_1)) + (ts * x_0 * sin(x_1)), \\
&- (ts * x_0 * cos(x_2)) + (ts * x_0 * sin(x_2)), \\
&- (ts * x_0 * cos(x_3)) + (ts * x_0 * sin(x_3)), \\
&- (ts * x_0 * cos(x_4)) + (ts * x_0 * sin(x_4)), \\
&- (ts * x_0 * cos(x_5)) + (ts * x_0 * sin(x_5))]
\end{aligned} \tag{8}
$$

And the Jacobian of the constraints is as follows:

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & -1 & 1 \\
0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 \\
vt * sum(cos(X)) & t * v * sin(x_1) & vt * sin(x_2) & vt * sin(x_3) & vt * sin(x_4) & vt * sin(x_5) \\
-sum(sin(X) * tv) & -vt * cos(x_1) & -vt * np.cos(x_2) & -vt * cos(x_3) & -vt * cos(x_4) & -vt * cos(x_5)
\end{bmatrix}
$$

Given the aforementioned Optimization problem, the chosen optimization algorithm is able to robustly solve the optimization problem and achieve an optimal solutions.

There are certain salient points in this problem formulation which makes it highly robust:

1. The search space is brought in the vehicles frame to make sure the orientation angles are always in a fixed interval of (-pi , pi)
2. Having the orientation angle in the feasible region also allows us to constraint the differences of the orientation angle to be within hard constraints.
3. By having a weighted formulation of the cost allows the increment of the orientation angle and hence the steering of the vehicle to be smooth.
4. The time step of the discretization of the MPC horizon is not absolute and changes its value depending on the environment. If the Ego Vehicle has to keep safe distance, the time step is optimized to maintain the necessary velocity to keep safe distance from the obstacle.

$$t_{step} = t_{step}\,(max_{vel}) + \frac{dist_{safe} - dist_{obat}}{vel_{post}}$$

3.4.3. Optimization Algorithm : Interior Point Optimization

Interior Point methods are algorithms that have the flexibility of solving both linear and non-linear convex optimization problems having equality and inequality constraints. The algorithm uses a linear programming algorithm known as Karmarkar's algorithm developed in 1984. The algorithm proves to run in polynomial time. The simple idea behind the algorithm is to traverse the interior of the feasible region based on a self-concordant barrier function which can encode a convex set. The optimization problem is a simple one as follows:

$minf(x)...x\epsilon R^n$
$S.T.$
$g_L <= g(x) <= g_u$
$x_L <= x <= x_u$
The following equations encode the barrier function:

$$B(x, \mu) = f(x) - \mu \sum_{i=1}^{m} \log\left(c_i(x)\right)$$

Where μ is the barrier parameter which converges to zero at the optimal solution.

4. Simulation Framework

Most of the code is developed in C++ and Python compatible with the ROS framework. The entire project is based out of the ROS framework and the simulations shown were developed in Gazebo. The ego-vehicle model is a package called the CatVehicle [6]. Rest of the environment was custom-designed by the authors.

5. Results

To test our framework we have conducted simulation tests for the scenarios using the aforementioned simulator. The following tests were conducted:

1. Safe distance following

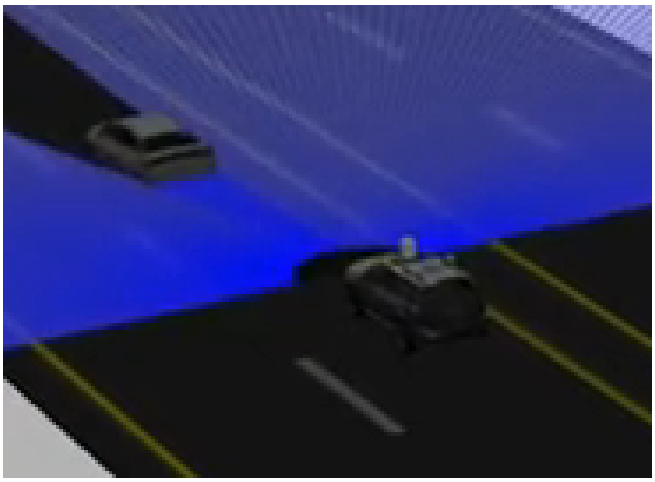

Figure 5. Following and safe distance keeping

2. Overtaking

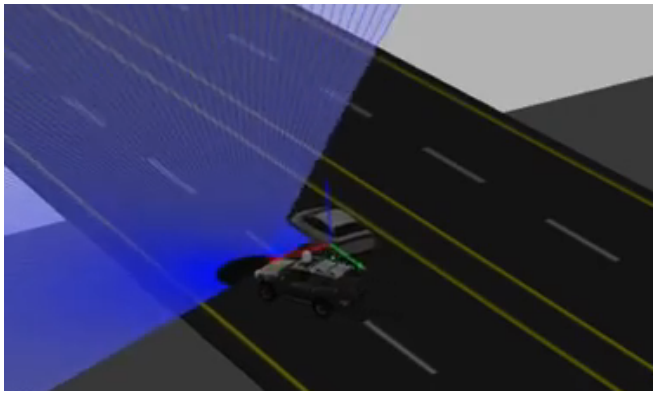

Figure 6. Overtaking

3. Following and Overtaking

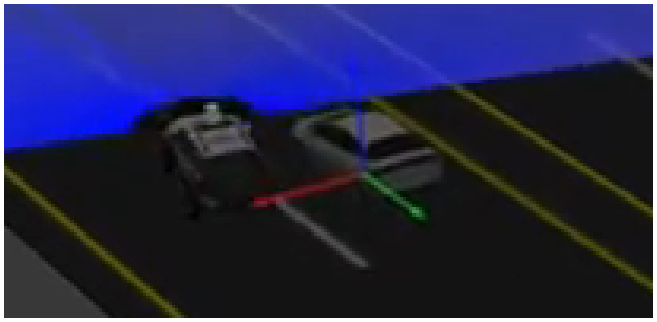

Figure 7. Overtaking

In the initial stages some target trajectories were also developed which are worst case scenario trajectories that the vehicle might go into.

1. From (0,0) to (2,2) at 90 degree vehicle orientation

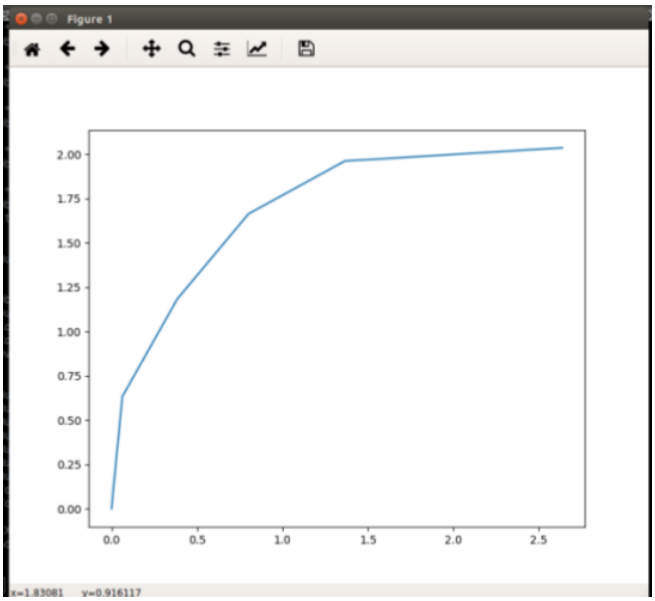

Figure 8. 90 degree turn without weighting the orientations

2. From (0,0) to (2,2) at 0 degree vehicle orientation

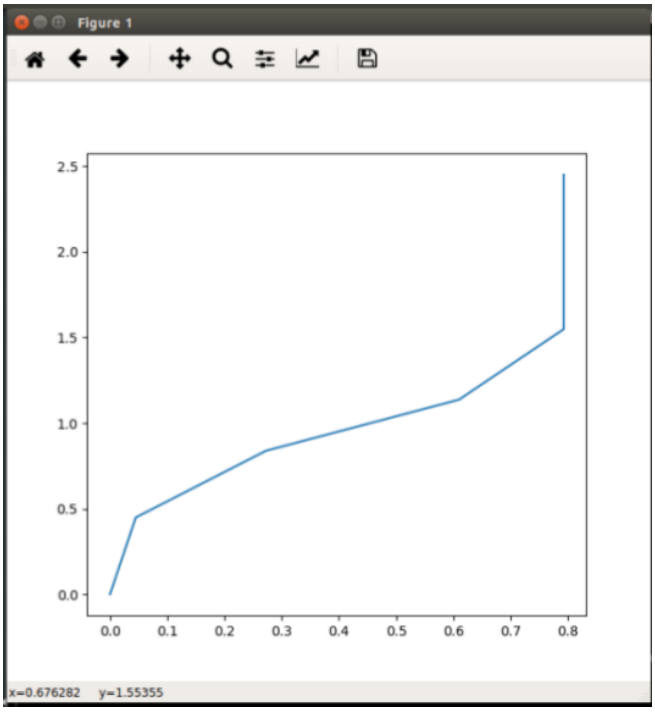

Figure 9. Parallel motion to 0 degree orientation

3. From (0,0) to (2,2) at 90 degree vehicle orientation With weights on orientation

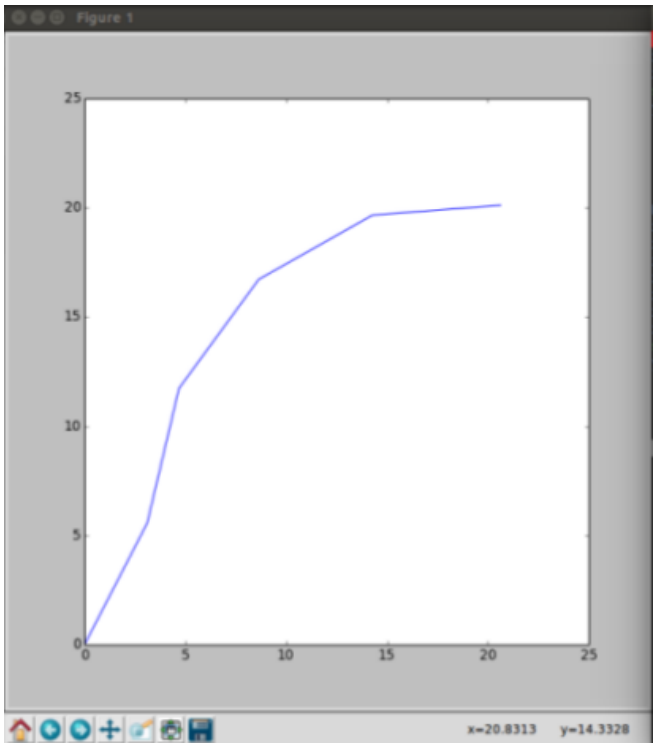

Figure 10. 90 degree turn with weighting the orientations

The results show that we are able to perform the maneuvers with sufficient confidence.

We can see that the framework tracks the predicted poses of the orientation and position fairly well. The end goal deviation is however not acceptable and more refinement in the cost function can lead to a better performance.

6. Including Dynamics

It is evident that all the formulations here are done for the kinematic model of the vehicle. It is a well known fact that the kinematic model cannot be used for high speed maneuvers and hence we would like to switch to the dynamical models of the vehicle. In the simulator however we are only able to command the velocity and orientation and not wheel torque and steering torque. In order to do so we need to have dynamical longitudinal and lateral vehicle models.
A good place to start will be to use a longitudinal vehicle model:
The time horizon discritization will be as follows:

$$t = \left[T\left(1 - \frac{i}{n}\right)\right]$$

And the model can be represented as follows:

$$\frac{V_T - U}{t} = \frac{\frac{T_w}{R_w} - \frac{T_b}{R_b} - F_{\text{gradient}} - F_{\text{rolling}}}{M}$$

$$V_T - U = \left[T\left(1 - \frac{i}{n}\right)\right]\left[\frac{\frac{T_w}{R_w} - \frac{T_b}{R_b} - F_{\text{gradient}} - \text{Frolling}}{M}\right]$$

$$\frac{T}{R_w} - \frac{T_b}{R_b} = M\left(\frac{V_T - U}{T\left(1 - \frac{i}{n}\right)}\right) + F_{gradient} + F_{\text{rolling}}$$

The aforementioned equations can then be impinged on the receding horizon of MPC.

$$f(U, i) = \left[M\left(\frac{V_T - U}{T\left(1 - \frac{i}{n}\right)}\right) + F_{\text{gradient}} + F_{\text{rolling}}\right]_{i=0}^{i=n}$$

Furthermore, on the basis of the hindering forces and the required inertial forces a system can be derived to switch between braking and acceleration mode depending on the vehicle's parameters and the current and target velocity.

$$\frac{V_T - U}{t} = \frac{\frac{T_w}{R_W} - F_{\text{gradient}} - F_{\text{rolling}}}{M}$$
$$\text{if } m\left(\frac{V_T - U}{t}\right) + F_{\text{gradient}} + F_{\text{rolling}} > 0$$

$$\frac{V_T - U}{t} = \frac{-\frac{T_b}{R_b} - F_{\text{gradient}} - F_{\text{rolling}}}{M}$$
$$\text{if } m\left(\frac{V_T - U}{t}\right) + F_{\text{gradient}} + F_{\text{rolling}} < 0$$

$$T_w = 0, T_b = 0$$
$$\text{if } m\left(\frac{V_T - U}{t}\right) + F_{\text{gradient}} + F_{\text{rolling}} = 0$$

7. Conclusions

To Conclude, we can see that the framework is able to provide robust control and tracking of the desired trajectory and even more so, the behaviour needed for the effective maneuvers. There are still some shortcomings, most important of which would be the effective goal reaching and tracking. The best way way to do so would be to have the goal reaching as a terminal cost. However, in our current optimization problem, doing so led to some unexpected behaviours of the optimizer leading to infeasible solutions which we cannot work with. Changing the cost function and also exploring other optimization algorithms can also help in solving these problems.

8. PseudoCodes

8.1. RRT

```
RRT*_plan (start, end, empty vector) {
    -Add a node containing the start
     point to the trajectory vector.
     Set its parent to be a nullptr.
    -Check whether there is a collision
     free path from the start point to
```

```
    the end point. If so, add a node
    containing the end point and set
    its parent to be starting node.
    Break.
    -If there was no direct path,
    confirm that the endpoint is in an
    open cell. If not, search right
    and left. If an open cell is still
    not found, search downward to the car
    until a point is found.
    -Once an end goal is decided, sample
    randomly for a cell and check if it
    is in free space. If it is, ensure it
    is within the max distance parameter
    from any given node on the tree.
    If not steer it toward the nearest
    node on the tree and stop at the max
    distance.
    -If there is no collision between
    this steered node and its nearest node,
    add it to the trees vector of node
    structs, set its parent to be its
    nearest neighbor on the tree, and
    check if it is within the max
    distance of the end node. If it is,
    and if there is no collision between
    the end node and this new node, add
    the end node to the tree and set
    its parent to be this new node.
    -Once the end node has been added
    to the tree, start from the end node,
    and add its parents, grandparents
    to the trajectory vector until the
    nullptr is encountered. This means
    we are back at the starting node.
    -Return the trajectory to the pose
    fitting function.
}
```

8.2. MPC

```
MPC_Tracker (current_odom, goal_odom) {
While (True)
{
    - Set goal point coordinates to
    variables x_goal y_goal and w_goal
    - Set time step in variable ts
    - Set the current odometry coordinates
        variables x_curr, y_curr and w_curr
    - Set the cost constant for
        x_coordinate reaching to be x_goal,
        y_coordinate reaching to be y_goal.
    - Set the constraint to reach the final
        goal orientation to be w_goal
    - Set the constraint to reach the
        goal point to be w_goal
    - Set the constraint to the first
        optimized angle to be 0.3 less
        or more than w_curr
    - Formulate the cost function and
        the constraints
    - Solve the optimization problem
    - Get the first orientation angle
        and velocity from the optimization
    - Convert the orientation angle to
        the steering angle as per
        the bicycle model
    - Publish the first steering angle and
        velocity to the drive parameters
}
```

9. References

1. Park, J-M Kim, D-W Yoon, Yongsoon J Kim, H Yi, K-S. (2009). Obstacle avoidance of autonomous vehicles based on model predictive control. Proceedings of The Institution of Mechanical Engineers Part D-journal of Automobile Engineering - PROC INST MECH ENG D-J AUTO. 223. 1499-1516. 10.1243/09544070JAUTO1149.

2. K. Yang and S. Sukkarieh, "An Analytical Continuous-Curvature Path-Smoothing Algorithm," in IEEE Transactions on Robotics, vol. 26, no. 3, pp. 561-568, June 2010. doi: 10.1109/TRO.2010.2042990

3. LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998).

4. Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30.7 (2011): 846-894.

5. Karaman, Sertac, and Emilio Frazzoli. "Incremental sampling-based algorithms for optimal motion planning." arXiv preprint arXiv:1005.0416 (2010).

6. Rahul Bhadani, Jonathan Sprinkle, Matthew Bunting. "The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications". Proceedings 2nd International Workshop on Safe Control of Autonomous Vehicles (SCAV 2018), Porto, Portugal, 10th April 2018, Electronic Proceedings in Theoretical Computer Science 269, pp. 3247.

7. Andreas Wchter and Lorenz T. Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. 106, 1 (March 2006), 25-57. DOI: https://doi.org/10.1007/s10107-004-0559-y

8. Babu, Mithun Oza, Yash Singh, Arun Krishna, Madhava Medasani, Shanti. (2018). Model Predictive Control for Autonomous Driving Based on Time Scaled Collision Cone. 641-648. 10.23919/ECC.2018.8550510.

9. G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber and L. B. Becker, "A Predictive Controller for Autonomous Vehicle Path Tracking," in IEEE Transactions on Intelligent Transportation Systems, vol. 10, no. 1, pp. 92-102, March 2009. doi: 10.1109/TITS.2008.2011697