Motion Planning Stack Self-Driving Cars with Smooth Local Planner

Algorithmic Motion Planning

Arpit Savarkar

University of Colorado Boulder
arpit.savarkar@colorado.edu

***Abstract*—To implement behavioral and local planning for self driving cars, over CARLA simulator. The objective is have a functional motion planning stack that can avoid both static and dynamic obstacles while tracking the center line of a lane, while also handling stop signs. Behavioral planning logic, as well as static collision checking, path selection, and velocity profile generation is implemented.**

I. Introduction

A. Autonomous Driving Mission

The overall objective is to navigate a vehicle from point A to point B on a map. The autonomous driving mission views this from a navigation perspective,where we need to navigate from one point on a map or current position,to another point corresponding to our final destination. In doing so, it should take the connection of different streets and their associated traffic into consideration. In order to find the most efficient path in the road network in terms of travel time or distance. Planning for an autonomous driving mission abstracts many of the important lower-level variables away from the problem in order to simplify the mission planning process. However,these lower level variables such as roads structures, obstacles, and other agents on the road, are crucial to the autonomous driving motion planning problem. These lower level variables define different driving scenarios under the operational design domain.

B. Operational Design Domain

1) Road Structure: Road structure is the lane limits and regulatory elements that are relevant to the driver. Simply driving in a lane is called lane maintenance. In the nominal case, this is when the car follows the central line of it's current lane. The final goal is to minimize deviation from the center line of the path as well as reaching reference speed which is often the speed limit to ensure efficient travel to our destination.

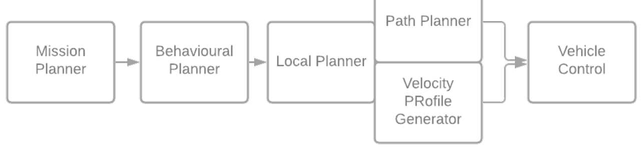
2) Lane Change: A more complex scenario is when the car has to perform a lane change maneuver. Even when there are no dynamic obstacles present in either lane, there is a need to optimize the shape of the lane change trajectory within the constraints of lateral and longitudinal acceleration under speed limit of the road, and the time horizon for execution of the maneuver.

3) Turning: Another situation is when the car needs to perform a left or right turn. This is often required when the autonomous car has to handle intersections. As with the lane change, the shape and aggressiveness of the turn will vary depending on the situation. In addition, the feasibility of actions and autonomous vehicle can take is affected by the state of the surrounding environment.

4) Static Vs Dynamic Obstacles: Static and dynamic obstacles will dramatically change both the structure of the scenario as well as the difficulty in determining the required behavior for a scenario. In the language of autonomous driving, dynamic obstacles are defined as moving agents in the planning workspace of the ego vehicle. In contrast, static obstacles or obstacles that are not moving such as parked cars, medians, and curves. Static obstacles restrict which locations the autonomous car can occupy as it plans paths to its destination. This is because the car occupies space as it travels along its path. If this overlaps with an obstacle, planned path will result in a collision. On the other hand, dynamic obstacles have a larger impact on our velocity profile and driving behavior.

5) Road Rules: Speed tracking is the nominal driving behavior. A reference speed or speed limit and the car has to maintain that speed while moving forward in a restricted lane. If there is a stop sign ahead, the car needs to smoothly slow down to a stop to maintain comfort. Staying stopped is required for certain regulatory elements. If there car is at a red light, then it needs to remain stopped until the light turns green.

Fig. 1. Motion Planning Stack



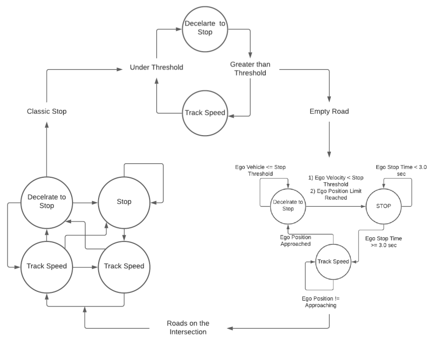
C. Motion Planning Abstraction and Hierarchical Breakdown

To solve for the optimal motion plan without any forms of selective abstraction and simplification, would simply be intractable. To remedy this, instead the task is broken up into a hierarchy of optimization problems.

1) Mission Planner: The mission planner focuses on map level navigation from the ego-vehicles current position, through a required destination. Because the scale of mission planning is often on the order of kilometers, a high level of abstraction is required to make the problem tractable. In this sense there is ignorance of aspects of the motion planning problems such as obstacles and regulatory elements, and instead focus is on the macro aspects of the problem such as traffic and road connections. If the focus is on the spatial length of the roads in the road network, a graph is constructed from the road network and then use Dijkstra's algorithm or A* search to find the shortest path. This graph can be optimized for minimize time.

2) Behavior Planner: The behavior planner is a portion of the motion planner that focuses on high level decision making required to follow the rules of the road, and recognize which maneuvers are safe to make in a given driving scenario. The behavior planner should make it clear to the path planner and velocity generator, that it is only safe to proceed when there is sufficient time gap between the ego-vehicles position, and the oncoming traffic and pedestrians in the intersection. This is taken care of by creating STATE MACHINES. The key concept behind finite state machines is that they are composed of states which represent the ego-vehicles required maneuvers, and transitions which dictate how the state should evolve depending on the inputs. These may be things like: vehicle locations, traffic light transitions, or any other element of interest.

Fig. 2. Intersection with Dynamic Obstacle



The other type of behavioral planners are rule-based systems. Essentially, these type of systems consist of a hierarchy of rules, where the hierarchy represents the relative importance of each rule. Each rule may correspond to a rule of the road such as stopping at a red light,or may correspond to a driving best practice such as maintaining a two second gap. The state of the ego-vehicle and its surroundings would then be fed into the rule-based system in the form of logical predicates.

3) Local Planner: Local planner is to generate kinematically feasible and collision-free paths as well as comfortable, and safe velocity profiles for the ego vehicle. The goal is to decompose the local planning problem into two sub-problems; path planning and velocity profile generation. The key ingenuity behind developing a good path planning algorithm is reducing the search space for optimization. There are three main categories of path planners; sampling-based planners, variational planners and lattice planners.

Lattice planners [4] constrain the search space by limiting the actions that the ego vehicle can take at any point in the workspace. This set of actions is known as the control set of the lattice planner. This control set, when paired with a discretization of the workspace, implicitly defines a graph. Obstacles can set edges that cross them to infinite cost. So the graph search allows us to perform collision checking as well.

While the lattice planner is often quite fast, the quality of paths are sensitive to the selected control set. A common variance on the lattice planner is known as the conformal lattice planner. Where goal points are selected some distance ahead of the car, laterally offset from one another with respect to the direction of the road and a path is optimized to each of these points. The path that best satisfies some objective while also remaining collision free is then chosen as the path to execute.

Fig. 3. Conformal Spatiotemporal Lattice Planner Algorithm [12]

```
**for** each station $s = 1 \cdots \#s$
    **for** each vertex $n$ at station $s$
        **if** $\hat{c}(n) \neq \infty$
            Form the vector
                $\hat{\mathbf{x}}_{\mathbf{n}} = [x(n), y(n), \theta(n), \kappa(n), t(n), v(n)]$
            **for** each acceleration $a$ and path parameter $\mathbf{p}$
                Form the trajectory $\tau_r(\mathbf{p}, t(n), v(n), a)$
                Evaluate the trajectory cost $c(\tau_r)$
                Identify the lattice vertex $n_f(\tau_r(s_f))$ —
                    —at the end of the trajectory
                **if** $\hat{c}(n) + c(\tau_r) < \hat{c}(n_f)$
                    $\hat{c}(n) \leftarrow \hat{c}(n) + c(\tau_r)$
                    $t(n_1) \leftarrow t(\tau_r(s_f))$
                    $v(n_1) \leftarrow v(\tau_r(s_f))$
                    incoming$(n_1) \leftarrow n$ // *backtrace info*
                **end if** // $\hat{c}(n)$
            **end for** // $a$
        **end if** // $\neq \infty$
    **end for** // *vertex*
**end for** // *station*
```

D. Parametric Curves

A parametric curve is a curve that can be described as a set of equations with specific parameters. These parameters often denote path traversal, whether it will be through arc length or just varying from zero to one. For autonomous driving, often but not always parametric curve is used to describe the path.

$$minf(r(u))s.t.\begin{cases} c(r(u) <= \alpha \forall u \epsilon [0,1] \\ r(0) = \beta_0 \\ r(u_f) = \beta_f \end{cases} \quad (1)$$

This is in contrast with parametric curve approach with the reactive planner, where representation of the trajectory and the path with a sequence of points in space.

1) Quintic Splines: [10] The first are quintic splines which are fifth order polynomial functions of the x and y position of the car. [10] For 'x' and 'y' defined by 5th Order splines. They help us formulate for a closed form solution.

$$\begin{aligned} x(u) &= \alpha_5 U^5 + \alpha_4 U^4 + \alpha_3 U^3 + \alpha_2 U^2 + \alpha_1 U + \alpha_0 \\ y(u) &= \beta_5 U^5 + \beta_4 U^4 + \beta_3 U^3 + \beta_2 U^2 + \beta_1 U + \beta_0 \\ & u \epsilon [0,1] \end{aligned} \quad (2)$$

quintic splines have 12 parameters, six for the x equation and six for the y equation. These parameters correspond to the polynomial coefficients that form the shape of the curve.

The downside with quintic splines is that it is often hard to constrain curvature within a certain set of bounds as is often required in autonomous driving. This has the potential to introduce cusps or even discontinuities of the curvature in the spline, which makes it difficult to approximately satisfy curvature constraints across the entire domain of the spline.

[11]The main positive of using polynomial spirals is that their structure is highly conducive to satisfying the approximate curvature constraints that are often required by the path planning problem. Since a spiral is a polynomial function of curvature, the curvature value will not change extremely quickly like it can in the case of quintic splines. This means constraining the curvature of only a few points in the spiral and the spiral will very likely satisfy the curvature constraints across the entire curve.

According to Simpsons Rule

$$\int_0^s f(s')ds' \approx \frac{s}{3n}(f(0) + 4f(\frac{s}{n}) + 2f(\frac{2s}{n}) + ... + f(s)) \quad (3)$$

The spiral ensures smooth curvature variation along the path, while the spline does not. As a brief shorthand, the spline leads to computational efficiency, while the spiral leads to easier implementation of curvature constraints.

E. Optimization for Path Parameters

The whole agenda for discussing Spirals and Quintic Splines is so that we can compute the path parameters to feed into our path planners. Boundary conditions describe the absolute minimum requirements for a path being planned between two points. Essentially, they require that for a given starting position heading and curvature, planned path ends at a specific position heading and curvature According to Equation 5. These form the first set of constraints. Using Spiral optimization, Simpsons Rule Equation 3 to evaluate the spiral. As the number of terms in the Simpson Rule increase,a more accurate solution is got to satisfy constraints. According to literature, across many domains, 8 terms in the Simpsons Formula to calculate the path.

Velocity profile generation is usually set up as a constrained optimization problem. Many velocity profile objectives are combined as goals of minimizing jerk or minimizing deviation from a desired reference. Convex optimizers are used for this purpose.

F. Trajectory Rollout Algorithm with Dynamic Windowing

[8] At a high level, the trajectory rollout planner uses the method of trajectory propagation, discussed earlier to generate a set of candidate trajectories that the robot can follow from its current point in the workspace.The obstacle information local to the robot is then determined for paths which are collision-free and which aren't.

1) Trajectory Set Generation: [8] The first step of the algorithm is to generate the set of trajectories at each time step. For trajectory rollout, each trajectory will correspond to applying a fixed input to the robot for multiple steps over a constant time horizon.Then uniformly sample these fixed inputs across the range of available input values in order to generate a variety of potential candidate trajectories. By reaching a wide variety of trajectories across the input spectrum, Improvement in the quality of the trajectory search and maneuverability is checked for as exploring a broader set of candidate paths for the robot to take.

2) Trajectory Propagation: [8] Post Creating a set of possible trajectories, we hold the velocity constant and varying the steering angle gives finer set of trajectories.

By Equation 10, swaths are generated by sweeping the body of the robot along the path, and taking the union of all the footprints at each time step of a given trajectory. The footprint of the car will correspond to a set of indices in the occupancy grid. So each rotated and translated point along the path will also correspond to different indices in the occupancy grid. The code then iterates through each point in the swath set and checking the associated index in the occupancy grid for an obstacle. If any point in the swath is occupied, then that trajectory contains a collision.

A set of collision-free kinematically feasible trajectories are then used to score using objective function. Paths that maximize the distance to the nearest obstacle are rewarded, to maximize the flexibility of feasible paths available to future time steps in the planner.

- Distance to goal

$$\begin{aligned} J = \alpha_1 ||x_n - x_{goal}|| + \alpha_2 \sum_{i=1}^{n} \kappa_i^2 + \\ \alpha_3 \sum_{i=1}^{n} ||x_i - P_{center}(x_i)|| \end{aligned} \quad (4)$$

Collision-free trajectories, and the path picked that minimizes the penalty depending on the formulation of the objective. The end time of the planning horizon shifts forward

Fig. 4. Trajectory Rollout Algorithm with Dynamic Windowing [15]

```
Input: candidates $\Pi_{\text{can}} = \{\pi_1, \dots, \pi_k\}$
Output: selection $\hat{\pi}^* \in \Pi_{\text{can}}$
 1: $A = \{1, \dots, k\}$
 2: $s_i \leftarrow 0$ for all $i \in A$
 3: $n_0 \leftarrow 0$
 4: for $l = 1, \dots, k-1$ do
 5:     $n \leftarrow n_l - n_{l-1}$
 6:     for all $i \in A$ do
 7:         for $j = 1, \dots, n$ do        ▷ Perform $n$ rollouts
 8:             $x_j \sim p(x_s)$
 9:             $\xi_j \leftarrow \text{Rollout}(x_j, \pi_i)$
10:             $s_i \leftarrow s_i + \Lambda[\xi_j]$
11:     $i_{\text{worst}} \leftarrow \arg\min_{i \in A}(s_i / n_l)$
12:     $A \leftarrow A \setminus \{i_{\text{worst}}\}$        ▷ Reject the worst $\hat{p}_\pi$
13: $\{\hat{\pi}^*\} \leftarrow A$
```

by one second for each planning cycle. Also known as receding horizon planner because at each planning cycle a fixed planning time horizon whose end times slowly recedes towards the time point at which the ego vehicle reaches the goal.

II. Problem Statement

In this project, the scripts have been developed to interact with the Carla Engine Simulator. It is highly advisable that the Appendix be read before the problem statement as the mathematical analysis is explained in detail. Additionally, the involved algorithms are described in the Section-I (Introduction).

A. Behavioral Planner

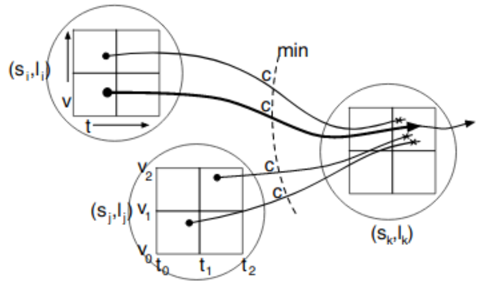
In this script of the project, a behavioural logic required to handle a stop sign is implemented based as discussed in Behavioral Planner discussed in Appendix. A state machine that transitions between lane following, deceleration to the stop sign, staying stopped, and back to lane following, when it encounters a stop sign.

B. Path Generation

For trajectory rollout, each trajectory will correspond to applying a fixed input to the robot for multiple steps over a constant time horizon.Then uniformly sample these fixed inputs across the range of available input values in order to generate a variety of potential candidate trajectories. By reaching a wide variety of trajectories across the input spectrum, Improvement in the quality of the trajectory search and maneuverability is checked for as exploring a broader set of candidate paths for the robot to take.

For the path generation section of this project, the majority of the mathematical code for generating spiral using paths is based on mathematical gradient descent and inspired from the analysis done in citation [11] and is developed from Motion Prediction Based Motion Planners according to [13] and [14].

Fig. 5. Path Generation [14]



Polynomial spiral are a polynomial curvature function with respect to arc length. [10] Closed form curvature definition allows for simple curvature constraint checking.

$$\begin{aligned}
\kappa(s) &= a_3 s^3 + a_2 s^2 + a_1 s + a_0 \\
\theta(s) = \theta_0 &+ \int_0^s a_3(s')^3 + a_2(s')^2 + a_1 s' + a_0 ds' \\
&= \theta_0 + a_3\frac{s^4}{4} + a_2\frac{s^3}{3} + a_1\frac{s^2}{2} + a_0 s \\
x(s) &= x_0 + \int_0^s cos(\theta(s'))ds' \\
y(s) &= y_0 + \int_0^s sin(\theta(s'))ds'
\end{aligned} \tag{5}$$

C. Path Selection

The path selection portion of this project involves evaluating an objective function based on curvature, velocity, and acceleration constraints, taking consideration into obstacles over the generated path for optimum path selection. The goal is to eliminate paths that are in collision with static obstacles, and to select paths that both track the centerline of the global path.

$$f_{be}(a_0, a_1, a_2, a_3, s_f) = \int_0^{s_f} (a_3 s^3 + a_2 s^2 + a_1 s + a_0)^2 ds \tag{6}$$

The fact that the objective function and its gradient have closed form solutions make it an objective function that is highly conducive to nonlinear programming.

Thus the final Optimization Problem for finding the parameters, now becomes

$$\begin{aligned}
min f_{be}&(a_0, a_1, a_2, a_3, s_f) s.t. \\
&|\kappa(\frac{s_f}{3})| <= \kappa_{max} \\
x_s(0) &= x_0,\ y_s(0) = y_0, \\
\theta(0) &= \theta_0,\ \kappa(0) = \kappa_0, \\
&|\kappa(\frac{2s_f}{3})| <= \kappa_{max} \\
x_s(s_f) &= x_f,\ y_s(s_f) = y_f, \\
\theta(s_f) &= \theta_f,\ \kappa(s_f) = \kappa_f
\end{aligned} \tag{7}$$

Softening the equality constraints, to optimize for non-linear solvers,

$$\begin{aligned} min f_{be}(a_0, a_1, a_2, a_3, s_f) + \alpha(x_s(s_f) - x_f) \\ +\beta(y_s(s_f) - y_f) + \gamma(\theta_s(s_f) - \theta_f) \\ s.t. \\ |\kappa(\frac{s_f}{3})| <= \kappa_{max} \\ |\kappa(\frac{2s_f}{3})| <= \kappa_{max} \\ \kappa(s_f) = \kappa_f \end{aligned} \tag{8}$$

Although this allows the optimizer to violate the boundary condition equality constraints, the optimizer will be strongly encouraged to converge to a solution that is as close as possible to the boundary conditions before the bending energy penalty term will be large enough to influence the optimizer.

D. Static Collision Checker

In particular, for this project circle-based collision checking is implemented on our computed path. A circle location calculation for each point on the path is calculated. To not indulge into Mapping for the environment and sensor modelling. The waypoints of the parked cars, is stored in a separate file so a circle-based collision checking can be used based only on distance.

E. Velocity Profile Generation

The velocity planner will not handle all edge cases, but will handle stop signs, lead dynamic obstacles, as well as nominal lane maintenance. Physics Functions under classic Newtonian Mechanics are used to calculate the nominal acceleration, following acceleration, ramping velocity and deceleration velocity profiles are generated.

A good initial value for our final velocity is the reference velocity given to us by the behavior planner. This reference velocity will largely be influenced by the maneuver that the behavior plan selected based on the current driving scenario. That is if the light is still red, the behavior planner will issue a stopped maneuver to the local planner which will include outputting a zero velocity reference.Else if the car is driving straight along a given road, the reference velocity may just be the speed limit of that current road.

The next input we take into consideration are the dynamic obstacle states. In particular, the lead vehicle in front of us. The lead vehicle speed regulates the ego car's speed. TTC (Time to Collision) is factored from this analysis, the time to collision is a function of the ego vehicle's relative velocity to the lead vehicle as well as the length of the path to the lead vehicle. Ensuring a safe goal, there will be a need to have reached the lead vehicle speed, if it is moving slower than the ego vehicle.

During overtake, or lane change, The lead vehicle will move ahead by the time ego vehicle reaches the current position. At that point, the ego vehicle has reached its speed preventing itself from colliding. The final input for velocity profiling is the maximum curvature along planned path. Usually, the maximum curvature across sampled points in conformal lattice planner path is taken and then the associated maximum speed for that points found. The profile must then reach this required speed by the point in the path. We therefore define a deceleration to the required speed at the minimum point and an acceleration afterward.

III. RESULTS

Upon fine tuning the hyper-parameters, the following snippets can be seen on a Server Mode of the Carla Simulator, and the code acts like a Client, updating the agents on the Simulator, based on the Script design and analysis. Metric Based Behavioral and local Planner is implemented to visualize what is fed into the controller from the planner, velocity profile generated is also visualized.

Fig. 6. Static Obstacle in Sim

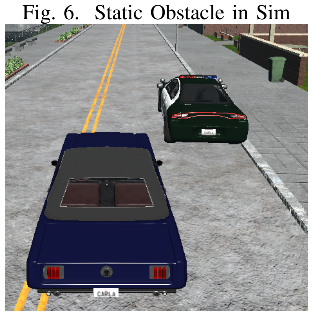


Fig. 7. Carla Simulator View of Following a Dynamic Obstacle

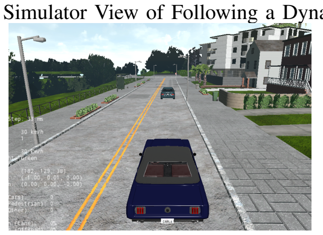
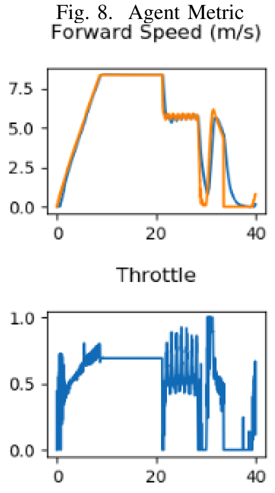
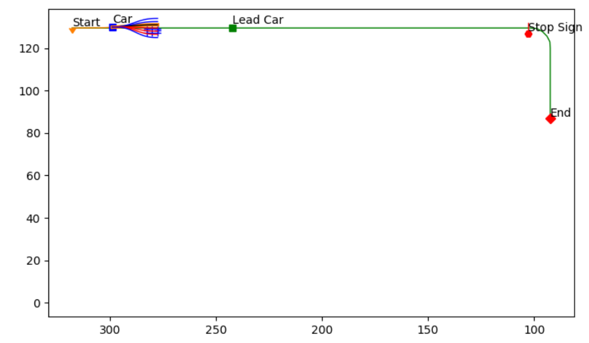


Fig. 8. Agent Metric



As it can be seen the Sub-Figure 1 of fig 8, the velocity profile generated during ramp up, tracking lead vehicle and trapezoidal ramp down within road limits during turn, the orange value is the generated value from the script and the blue is actually undertaken by the car controller. All the input controllers are Clipped within the velocity and acceleration bound.

Fig. 9. Trajectory View of Avoiding Static Obstacle



The cost function plays an important role in shaping the final behavior, and more work is necessary to develop cost functions for more complex behaviors. Investigation of more sophisticated acceleration profiles while maintaining execution efficiency is avoided here for brevity. Using a constant acceleration over the course of the path can lead to execution errors, since vehicles typically cannot change acceleration abruptly. At low speeds, constructing paths using a cubic polynomial spiral while staying strictly within curvature rate overly limits the maneuverability of the vehicle. A refinement to the types of actions considered may be necessary at low speeds.

The right lane is given a slightly lower cost to traverse than the left lane, encouraging the ego vehicle to stay in the right lane. Since the bonus awarded for driving further by the final cost coefficients exceeds the slight penalty for driving in the left lane, the planner selects a trajectory that moves into the left lane, passes the static obstacle, and merges back in front of it.

Fig. 10. Vehicle Trajectory during Turn

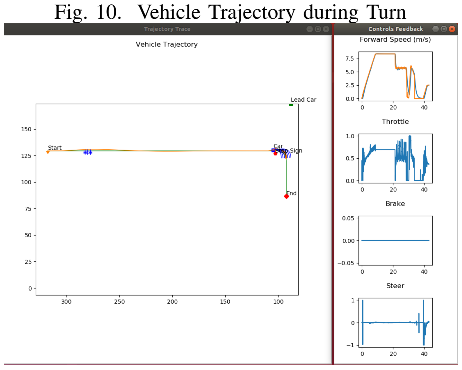
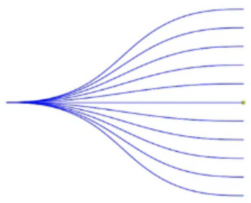


Fig. 11. Comformal Lattice Planner Possible Path Generation



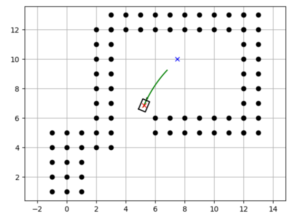


Fig. 12. In Between Run- Workspace I

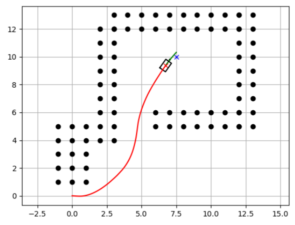


Fig. 13. Path Found - Workspace I

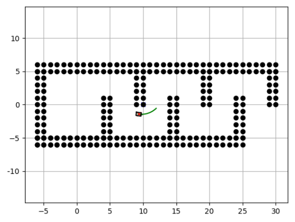


Fig. 14. In Between Run - Workspace II

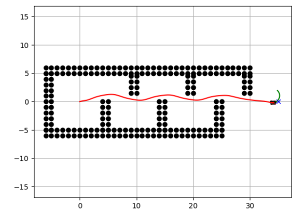


Fig. 15. Path Found - - Workspace II

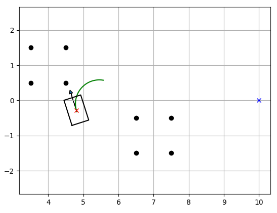


Fig. 16. In Between Run - Workspace III

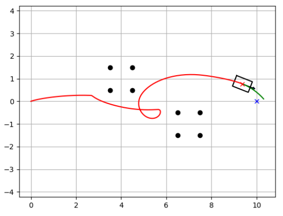


Fig. 17. Path Found - - Workspace III

IV. Links to necessary files

1) Link to the Github Code: **https://github.com/arpit6232/Autonomous_Vehicle_Lattice_planner**
These Codes act a Python Clients to the Carla Simulator which acts like a server. Additionally the Trajectory roll-out planner is a standalone code which is independent of Carla Sim.
2) Link to the Video of the functioning Code: **https://www.youtube.com/watch?v=TstrvFuLkCo&ab_channel=arpitsavarkar**
    a) Since heavy rendering is involved during the talk of the simulator with the code, Simulation runs slowly. Speed up the video if necessary
    b) A running Plot of the trajectory generated during run can be seen, Velocity Profiles, Steer and brake profiles plots have been blocked off on the actual code, due to security updates on the Carla Simulator.

V. Conclusion

In this project, a method was developed to decompose the motion planning stack into a hierarchical structure. Furthermore, the highest level task, mission planning for navigation was discussed in detail, where the shortest path over a graph is analyzed. Trajectory roll-out motion planner was developed and its implementation was tested for 3 different environment. This algorithm is usually used for for static environments, and a simple waypoint based methods for collision checking, path prediction, and time to collision calculation was undertaken. Finally the lowest level of motion planning hierarchy a local planner was designed, which allowed, for generation of smooth paths through the environment necessary for implementation by real vehicle controllers. All of the above was implemented over Carla Simulator, for kinodynamic applicability of the planner to test the results.

References

[1] P. Polack, F. Altché, B. d'Andréa-Novel and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, 2017, pp. 812-818, doi: 10.1109/IVS.2017.7995816.

[2] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, 2013, pp. 5041-5047, doi: 10.1109/ICRA.2013.6631297.

[3] N. Ratliff, M. Zucker, J. A. Bagnell and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 489-494, doi: 10.1109/ROBOT.2009.5152817.

[4] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, 2008, pp. 2611-2616, doi: 10.1109/IROS.2008.4651220.

[5] Urmson, C., Baker, C., Dolan, J., Rybski, P., Salesky, B., Whittaker, W., Ferguson, D., Darms, M. (2009). Autonomous Driving in Traffic: Boss and the Urban Challenge. AI Magazine, 30(2), 17. https://doi.org/10.1609/aimag.v30i2.2238

[6] K. Massow et al., "Deriving HD maps for highly automated driving from vehicular probe data," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, 2016, pp. 1745-1752, doi: 10.1109/ITSC.2016.7795794.

[7] J. Wei, J. M. Snider, T. Gu, J. M. Dolan and B. Litkouhi, "A behavioral planning framework for autonomous driving," 2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, 2014, pp. 458-464, doi: 10.1109/IVS.2014.6856582.

[8] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.

[9] Pivtoraiko, M., Knepper, R.A. and Kelly, A. (2009), Differentially constrained mobile robot motion planning in state lattices. J. Field Robotics, 26: 308-333. https://doi.org/10.1002/rob.20285

[10] A. Kelly and B. Nagy, "Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control," The International Journal of Robotics Research, vol. 22, no. 7, pp. 583–601, 2003. This paper discusses the math behind generating spirals to desired terminal states.

[11] A. Piazzi and C. G. L. Bianco, "Quintic G/sup 2/-splines for trajectory planning of autonomous vehicles," Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511). This paper discusses the math behind generating quintic splines to desired terminal states.

[12] M. Mcnaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," 2011 IEEE International Conference on Robotics and Automation, 2011. This paper introduces the concepts behind generating a conformal spatiotemporal lattice for on-road motion planning.

[13] https://github.com/paulyehtw/Motion-planning-based-on-Model-Predictive-control-and-Bezier-spline

[14] https://github.com/qiaoxu123/Self-Driving-Cars/blob/master/Part4-MotionPlanningforSelf-DrivingCars/Module7-Puttingitalltogether-SmoothLocalPlanning/Module7-Puttingitalltogether-SmoothLocalPlanning.md

[15] M. C. Koval, J. E. King, N. S. Pollard and S. S. Srinivasa, "Robust trajectory selection for rearrangement planning as a multi-armed bandit problem," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, 2015, pp. 2678-2685, doi: 10.1109/IROS.2015.7353743.

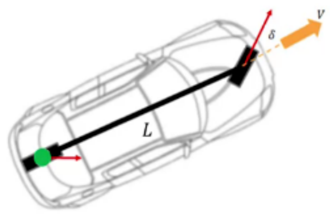
This Marks the End of the Report, as part of the deliverable for this project a detailed study of the Motion Planning Stack for self-driving cars is required which forms the Appendix for the report

Appendix

A. Models

1) Bicycle Kinematic Model: [1] The kinematics for the ego vehicle are often simplified to what is known as the kinematic bicycle model. One reason why this model is chosen is that bicycles have a range of acceptable steering angle values similar to a car. For a fixed velocity v, this range of steering angle values and δcorresponds to a range of admissible curvatures K that the vehicle can follow. This means that when performing motion planning while using the bicycle model, there is a maximum magnitude of curvature that can be executed when traversing a given path denoted by K max. Non-holonomic constraints need to be considered in the workspace.

Fig. 18. Bicycle Model



$$\begin{aligned} x' &= vcos(\theta) \\ y' &= vsin(\theta) \\ \theta' &= \frac{V \tan \delta}{L} \\ \delta_{\text{min}} &<= \delta <= \delta_{\text{max}} \\ v_{\text{min}} &<= v <= v_{\text{max}} \\ \kappa &= \kappa_m \end{aligned} \tag{9}$$

'x' and 'y' correspond to the base link position of the robot. Θcorresponds to the heading of the chassis with respect to x-axis and δis the steering angle input, v is the velocity input.

2) Kinematic Model Discretization: Discretization of differential equations allows for efficient computation of trajectories and recursive update scheme saves computation time. While this is useful for trajectory planning, it is also useful for motion prediction, where a kinematic model is known for different agent in the driving scenario and an educated guess on the control inputs can be synthesized. From this, an estimate to their future trajectory which can help plan the motion to avoid collisions. If the steering angles are varied along the path according to some steering function, more complex maneuvers can be performed, which are critical for such tasks as obstacle avoidance. This is essentially the crux of the local planning problem, calculating the required control inputs to safely navigate to a given goal point.

$$\begin{aligned} x_n &= \sum_{i=0}^{n-1} v_i cos(\theta_i) \triangle t = x_{n-1} + v_{n-1} cos(\theta_{n-1}) \triangle t \\ y_n &= \sum_{i=0}^{n-1} v_i sin(\theta_i) \triangle t = x_{n-1} + v_{n-1} sin(\theta_{n-1}) \triangle t \\ \theta_n &= \sum_{i=0}^{n-1} \frac{v_i tan(\delta_i)}{L} \triangle t = \theta_{n-1} + \frac{v_{n-1} tan(\delta_{n-1})}{L} \triangle t \end{aligned} \tag{10}$$

3) Vehicle Dynamic Model: Vehicle constraints focus on keeping the car in a stable safe state. The first of the dynamics constraints is imposed by what is called the friction ellipse of the car. Tire slip and the friction ellipse in bicycle model. The friction ellipse shows the maximum magnitude of the friction forces that can be generated between a car tire and the road. If the applied forces of the car's engine exceed the friction forces of the tire, the tires will slip on the road surface. The turning functionality of a vehicle relies on the tires gripping the road surface. So to maintain control and stability, the car must remain within the friction circle. At the end of the day, this boils down to lateral and longitudinal acceleration constraints. In general, though the acceleration constraints are often much higher than what is physically comfortable for the passengers inside the car. Therefore, in non-emergency situations we focus on the comfort rectangle of accelerations. These values constrain, the lateral and longitudinal accelerations to each lie within a range of comfort, denoted on the longitudinal acceleration along and lateral acceleration.

This results in a tighter constraint on the feasible accelerations for a motion plan than the friction ellipse requires. From the lateral acceleration constraints, as well as the curvature of the path, we indirectly constrain the velocity of the car. The velocity of the car V is a function of the lateral acceleration of the car, as well as the instantaneous turning radius of the path r.

Governing by the constraints

$$\begin{aligned} a_{\text{lat}} &= \frac{v^2}{r} \\ a_{\text{lat}} &<= a_{\text{lat max}} \\ \kappa &= \frac{1}{r} \end{aligned} \tag{11}$$

4) Road signs and conservative approach: Different classes of dynamic obstacles such as cars, bicycles, and pedestrians will all have different behaviors and motion models. Constraining motion plan based on the behavior of these other agents will often involve prediction, which is subject to uncertainty. Dynamic and static obstacles will constrain both behavior planning process, where maneuver decisions are made, as well as our local planning process, where it will affect velocity profile planning.

While the rules of the road provides some constraints to the planning problem, they also help make informed decisions about other agents behaviors in the environment. The ego vehicle clearly needs to respect red lights and stop signs to ensure safety, but also needs to be aware of speed limits in different areas and other dynamic regulatory elements, such as those presented by construction sites. In general, regulatory elements will have a large impact on which behaviors are available when performing motion planning.

B. Objective Constraints

1) Path Length: When performing path planning, the simplest and most intuitive goal is that would like to reach our destination as efficiently as possible. In the path planning context this, often translates to minimizing the arc length of the path that is under planning. Intuitively, the arc length of the path is the total accumulated distance the car will travel as it traverses the path. In many cases, the path will be parameterized by arc length. So this can be formulated as a penalty term proportional to the arc length. By minimizing arc length, the shortest path computation from current location, through required destination is undertaken. When optimizing a velocity profile, time to destination is often the objective under minimization. In terms of arc length, this is the same as integrating the inverse of the longitudinal velocity over the path as a function of arc length. Intuitively, this is equivalent to dividing the total distance by the velocity traveled along the curve at each point. Except since the velocity changes along the path, it is formulated as an integral. By minimizing this integral, intuitively time to reach the destination while following a given planned path is minimized.

$$\begin{aligned} s_f &= \int_{x_i}^{x_f} \sqrt{1 + (\frac{dy}{dx})^2} dx \\ T_f &= \int_0^{s_f} \frac{1}{v(s)} ds \\ \kappa &= \frac{1}{r} \end{aligned} \tag{12}$$

The reference path to follow will be given to pass on by the mapping module as the center line of a length, or the required path for a turn in an intersection. To ensure that the ego vehicle follows the reference path as closely as possible, consider the integral of difference terms, it penalizes deviations from the input reference path Xref. This allows for penalizing the speed limit violations more harshly while still allowing to encourage the autonomous vehicle to reach its required reference velocity.

$$\begin{aligned} &\int_0^{s_f} ||x(s) - x_{\text{ref}}(s)|| ds \\ &\int_0^{s_f} ||v(s) - v_{\text{ref}}(s)|| ds \end{aligned} \tag{13}$$

2) Smoothness: To maintain stability and comfort, constrain on maximum acceleration magnitude is needed. When optimizing comfort in the objective function of velocity profile, focus is also needed to minimize the jerk along trajectory. The jerk along the car's trajectory greatly impacts the user's comfort while in the car.

$$\int_0^{s_f} x'''(s)|| ds \tag{14}$$

3) Curvature: Paths with high curvature constrained the maximum velocity along the path to a low value in order to remain within the friction ellipse of the vehicle. To ensure to avoid points of high curvature along path, formulation of penalty for large absolute curvature values is needed, bending energy of the path is used for this. This objective distributes the curvature more evenly along the path, preventing any one along the path from reaching too high a total curvature value.

$$\int_0^{s_f} ||\kappa(s)||^2 ds \tag{15}$$

C. Motion Prediction

Motion prediction attempts to estimate the future positions, headings, and velocities of all dynamic objects in the environment over some finite horizon. The predicted paths also allow to make sure that the path which the ego vehicle plans to execute, will not collide with any future objects at a future time. In order to be able to predict the motion of moving objects,access to information about the environment around ego vehicle is needed. Especially as it relates to dynamic objects.

First is the history of the dynamic vehicle state or the vehicle track as it moves through the environment. This can be extremely useful. A high definition roadmap can also be used as an additional information source, to determine future behavior of dynamic objects. An image of the dynamic object in its current state can also be a useful source of information that can improve predictions. This is true for both vehicles and pedestrians. For vehicles, the image can provide information concerning the current indicator light or brake lights states. Similarly, images of pedestrians can serve to show the current orientation of the person, which can help predict a future direction of travel, even if the pedestrian is currently stationary.

1) Prediction Classes Assumptions: The first class of assumptions is to rely on, is that vehicles must follow a set of physical constraints governing their movement. Vehicle Kinematics and Dynamics os applied to vehicles in the environment to predict their motion. This is known as physics-based prediction.

The second class of assumptions that are used are that almost all motions by a vehicle on the road, are made up of a finite set of maneuvers in a restricted domain in the environment. In this case, the assumption is that vehicles which are on the road will stay on the road and follow the driving rules. For example, they will most likely stay in their lane unless indicating otherwise and stop at regulatory elements requiring stops. They are unlikely to drive over sidewalks or lawns or through obstacles.

Finally, the third class makes the same assumptions as the maneuver-based assumptions. However, instead of only evaluating each vehicle independently, incorporation of the assumption that the dynamic objects will react and interact with each other. An example of this type of prediction, is during a merge by a vehicle into an adjacent lane. Often, the vehicle in the destination lane will slow down to make more room for the incoming vehicle to maintain a safe following distance. These types of assumptions are referred to as interaction-aware assumptions.

D. Map-Aware Motion Prediction

Map-aware algorithms make two broad categories of assumptions to improve the motion predictions particularly for vehicles.

1) Position-based assumptions: The first assumption made to improve the position component of the prediction, is that vehicles driving down a given lane usually follow that lane. The vehicle will most likely turn along with the roadway. The second assumption that can be made is that a drive lane change or direction prediction, can be made based on the state of the indicator light of the vehicle. If such a detection has been made by the perception stack for a vehicle, it is possible to switch the prediction to account for this additional information.

2) Velocity-based assumptions: Velocity-based assumptions are used to improve the velocity prediction of a dynamic object. All vehicles on the road are affected by road geometry. Thus, it is useful to assume that as a vehicle approaches a turn with significant curvature, the vehicle is likely to slow down to avoid exceeding its lateral acceleration limits. Finally, velocity prediction can be greatly enhanced if we also consider a regulatory elements, which the dynamic object may encounter. HD road maps can also be used to improve the prediction.

For roadways with natural curvature, an assumption is made that a vehicle which is on a drive lane will likely follow that drive lane through the curve.Motion predictions can be updated then by using the center line of the mainland map as the predicted path of the vehicle, instead of the straight line path generated by constant velocity predictions. The center line of a lane lit is defined as a set of points making up a polyline that is equally spaced from both lane boundaries. While minor deviations from the exact center line can be expected, the center line can act as a good motion prediction approximation.

3) Multi-Hypothesis Prediction: It can be identified that the most likely behavior using objects state, appearance, and track information, and then construct a prediction based on the most likely behavior, that is it can be constructed to predict for the most likely behaviors and associate a probability that the agents will follow a particular path based on the state appearance and track information. The second approach is a slight generalization of the first and this is called the Multi-Hypothesis Prediction, which is generic enough for multiple traffic conditions.

Each nominal behavior of a vehicle based on the full range of possibilities available to it at its current location in the HD road map is considered. For the three-way intersection includes three possibilities: turn left, turn right, or stay stationary. Based on corroborating evidence such as indicators signals, position to the left or right of the center line, and the state of the vehicle at the intersection. It is possible to evaluate each of the three hypotheses in terms of the likelihood the agent will execute each of them within the prediction horizon. These probabilities can be hard to quantify exactly. So can either be learned from training data of many vehicles proceeding through similar intersections, or can be engineered and refined from real-world testing.

E. Time to Collision

All of the above analysis and breakdown leads into the analysis for us to calculate the time to collision for the vehicle. The time to collision provides a valuable measure of behavioral safety in a self-driving vehicle, and is heavily used in assessing potential maneuvers, based on the current driving environment. [5] By knowing if collisions are imminent and when they might occur and with which object, a self-driving system can better plan safe maneuvers through the environment, or prepare for emergency evasive actions if needed. To evaluate the time to collision between dynamic objects, we use their predicted paths to identify possible collision points. If a collision point exists, the time to collision is a measure of the amount of time until that collision occurs.

To evaluate the time to collision between dynamic objects, their predicted paths can be used to identify possible collision points. If a collision point exists, the time to collision is a measure of the amount of time until that collision occurs.

- Simulation-based approaches simulate the movement of every vehicle in this scene over a finite horizon into the future. [5] At each simulation time step, new position, heading, and occupancy extent predictions are calculated for every dynamic object. This is done by propagating forward the predicted trajectory. Once the position of all dynamic objects is known for a given time in the simulated future, a check is conducted to determine if any two or more dynamic objects have collided with each other. If they have collided, the location and time to the collision is then noted. Note that this is a different type of collision checking than in the static case, where swaths of the entire path were constructed and compared to static object locations. Because these collision checks are performed between moving objects, need is to only check collisions between the geometries of each object at each instant, as the objects will occupy new locations at the next time step.
- Estimation-based approaches function by computing the evolution of the geometry of each vehicle as it moves through its predicted path. [5] The results is a swath for each vehicle that can then be compared for potential collisions. Once the swath has been computed, their intersection can be explored for potential collision points by computing, if any two or more dynamic objects will occupy the same space at the same time.If this is true, potential collision points are tagged, and an estimation of when each vehicle will arrive at each collision point is used to estimate the time to collision.

[5] Due to their simplifying assumptions however, estimation-based approaches usually create a less accurate estimate of the collision point and the time to collision. Simulation approaches on the other hand do not need to rely on these approximations, and with small step sizes, can

perform high fidelity evaluation of the time to collision. These relative differences make each algorithm suited to different applications. For real-time applications such as onboard time to collision computation, estimation-based approaches are preferred. Whereas offline, where accuracy is vital such as dataset creation and simulation-based testing, simulation-based approaches are preferred. Based on the Estimation or Simulation techniques, time to collision for the agents on road can be calculated.

A behavior planning system, plans the set of high level driving actions, or maneuvers to safely achieve the driving mission under various driving situations. The set of maneuvers which are planned should take into account, the rules of the road, and the interactions with all static and dynamic objects in the environment. The set of high level decisions made by the planner must ensure vehicle safety and efficient motion through the environment. The over-arching goal of the behavioral planner is quantify and qualify where to stop, how long to stay stopped for, and when to proceed through the intersection and on road. The behavior planners should also be able to deal with inputs that are both inaccurate, corrupted by measurement noise, and incorrect, affected by perception errors such as false positive detections and false negative detections.

F. Driving Behavioral Maneuvers

The following is not an exhaustive list as a representation of the set of likely maneuvers or driving behaviors encountered throughout regular driving that an autonomous vehicle may need to execute.

- Track Speed: This behavior amounts to unconstrained driving on open road, meaning that the only restriction on forward progress is that the speed limit should be respected.
- Follow lead vehicle: The speed of the vehicle in front of the ego vehicle should be matched and a safe following distance should be maintained.
- Decelerate to stop: A stop point exists in the ego vehicle's lane within the planning horizon, and the vehicle should decelerate to a standstill at that stop point. Every regulatory element that requires a complete stop triggers this behavior.
- Remain Stopped: The vehicle should continue to stay stationary for a fixed amount of time. As an example, when the vehicle stops at a stop sign, it should stay stopped for at least three seconds.
- Merge Behavior: The vehicle should either merge into the left or right lane at this time.

G. Inputs to Behavioral Planner

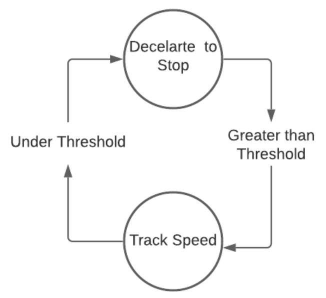
In order for the behavior planner to be able to produce the required output, it needs a large amount of information from many other software systems in the autonomy stack of the car.

- The behavior planner relies on full knowledge of the road network near the vehicle. This knowledge comes from the [6] high definition road map.
- The behavior planner must know which roadways to follow to get to a goal location. This comes in the form of a mission path over the road network graph.
- The location of the vehicle is also vital to be able to correctly position HD roadmap elements in the local environment around the vehicle. Accurate localization information is also needed from the localization system as a result.
- Behavior planner requires all relevant perception information, in order to fully understand the actions that need to be taken, to safely activate the mission. This information includes, all observed dynamic objects in the environment, such as cars, pedestrians, or bicycles. For each dynamic object, its current state, predicted paths, collision points, and time to collision, are all required. It also includes, all observed static objects in their respective states, such as parked vehicles, construction cones, and traffic lights, with an indication of their state.
- A local occupancy grid map defining the safe areas to execute maneuvers.

H. Finite State Machines

1) Handling a stop sign intersection with no traffic.: [7] For behavior planning system, the states will represent each of the possible driving maneuvers, which can be encountered.

Fig. 19. Intersection Finite-State Machine

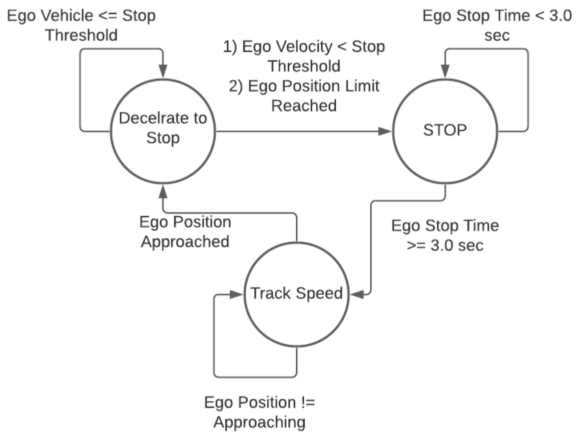


2) Four way Intersection without Dynamic Obstacles: [7] The area of the intersection where a vehicle should begin safely braking is defined as the approaching zone of the intersection. The other states are The zone of the intersection in which the vehicle must stop and wait until the appropriate time to proceed. Finally, the zone in which the vehicle is crossing the actual intersection is defined as on the intersection. The size of each of the above zones are dynamically changed based on two primary factors: the ego vehicle speed, at higher speed, it is required for more distance to safely and to comfortably stop, and the size of the intersection. The entry action defines the stop point location. ”Stop”, this maneuver tells the vehicle to stay stopped in its current location. The entry action is to start a timer to wait for a fixed amount of time before proceeding through the intersection.

The transition condition on moving from track speed to decelerate to stop is therefore entry into the approaching zone.

Then, once decelerating, the next maneuver which the autonomous vehicle must execute is to come to a full stop before the stop line, or in the at zone of the intersection. A timer is started to make sure that the vehicle stays in the stopped state for three seconds before proceeding in accordance with typical driving rules. Multiple hyper-parameters here can be fine tuned to get a more precise update actions or state.

Fig. 20. Four Way Intersection


Ego Vehicle <= Stop Threshold
1) Ego Velocity < Stop Threshold
2) Ego Position Limit Reached
Ego Stop Time < 3.0 sec
Decelrate to Stop
STOP
Ego Position Approached
Ego Stop Time >= 3.0 sec
Track Speed
Ego Position != Approaching

3) Complete Intersection: [7] In driving, there are many such scenarios which will be encountered. For example, there are three-way stop intersections, traffic light controlled intersections, and straight road scenarios, just to name a few. While many of these scenarios share some similarities largely due to the logic required to handle them, each of the scenarios can be considered to require fundamentally different driving behaviors. Hierarchical state machine in Figure 2. The transitions between the high level scenario state machine would be a rule that defines when a new scenario has been entered, based on the HD roadmap and dynamic vehicle information. A semi-critical Analysis of a car states and actions undertaken by University of Toronto is seen in Figure 2.

I. Collision

Collision checking is that it is a challenging, computationally intensive problem present in many domains, including autonomous driving and other robotic applications. Guaranteeing a safe collision-free optimal path not only requires perfect information about the environment, it also requires a prohibitive amount of computation power to compute in exact form especially for complex shapes and detailed environments. For autonomous driving which requires real-time planning, we have neither of these available to us. the information given to us by the occupancy grid is an imperfect estimate, which means that there is a need to add buffers to collision checking algorithms to make them more error tolerant.

1) Swatch Computation: In exact form, collision checking amounts to rotating and translating the footprint of a vehicle along every point of a given path. Each point in the footprint is rotated by the heading at each path point and translated by the position of that same path point.

$$S = Set_{p\epsilon P}(f(x(p), y(p), \theta(p)) \tag{16}$$

After performing Swath compute for every point along the path, the resulting swath of the car along the path is given by the union of each rotated and translated footprint. It is then checked to see in the entire set to see if there are any obstacles inside it.

In a discretized occupancy grid, computation of the algorithm , update steps for grid resolution δbecomes

$$\begin{aligned} x_i &= \frac{x(p) + \frac{X}{2}}{\delta} \\ y_i &= \frac{y(p) + \frac{Y}{2}}{\delta} \\ S &= SU(x_i, y_i) \end{aligned} \tag{17}$$

2) Lattice Planner Swaths: [9] This computation becomes quite expensive as the problem scales, which makes it difficult to use when performing real-time planning. In addition, using the exact footprint when calculating the swath can be dangerous due to our imperfect information, as there is no buffer for errors in obstacle positions. Because this swath-based method is often computationally expensive, it is often useful when exploiting a lot of repetition in motion planning algorithm as in the case in lattice planners.

A conservative approximation to collision checking is needed that is it would be one that may report a collision along a path even if there isn't one, but will never report no collision along a path if one does actually occur.

3) Circle Collision Swaths: [9] Approximation is useful for some algorithms because it is computationally cheap to check if a point lies within a circle. All that is needed is is to check if the distance between any of the points of the obstacle and the center of the circle is less than the radius of the circle.

$$||(x_i, y_i) - (x_c, y_c)|| <= r \tag{18}$$

There are three main categories of path planners; sampling-based planners, variational planners and lattice planners.

Sampling-based planners [2] randomly sample the control inputs of the car uniformly, in order to generate potential paths for the car to traverse. Sampling-based algorithms are extremely fast but at the cost of potentially generating poorer quality erratic paths when run in a short number of cycles. Variational planners [3] rely on the calculus of variations to optimize a trajectory function which maps points in time to positions in the workspace according to some cost-function that takes obstacles and robot dynamics into consideration. Variational planners are usually trajectory planners, which means they combine both path planning and velocity planning into a single-step. Variational methods are often slower, more complex and their convergence to a feasible path is sensitive to the initial conditions. An example of a variational method is the CHOMP algorithm.

The local planner is the portion of the hierarchical planner that executes the maneuver requested by the behavior planner in a collision-free, efficient, and comfortable manner. This results in either a trajectory, which is a sequence of points in space at given times or a path and velocity profile, which is a sequence of points in space with the required velocities at each point. This plan can then be given as the reference input to the controllers.

The root problem which the local path planner works on is given a starting position, heading, and curvature, find a path to an ending position heading and curvature that satisfies our kinematic constraints. In the context of an optimization, the starting and end values can be formulated as the boundary conditions of the problem, and the kinematic motion of the vehicle can be formulated as continuous time constraints on the optimization variables.

In the continuous domain, planning along the computed paths/swaths, formulating a double ended boundary value problem of start and goal and kinematic constraint restricting the maximum curvature along the path can be computationally expensive thus sampling is considered. To simplify the representation of our optimization problem to satisfy the above created samples, parametric curves are considered.

$$\begin{aligned} x_s(s) = x_0 + \frac{s}{24}[cos(\theta\frac{s}{8}) + 2cos(\theta\frac{2s}{8}) + 4cos(\theta\frac{3s}{8}) \\ +2cos(\theta\frac{4s}{8}) + 4cos(\theta\frac{5s}{8}) + 2cos(\theta\frac{6s}{8}) + 4cos(\theta\frac{7s}{8}) \\ cos(\theta(s)] \end{aligned} \tag{19}$$

$$\begin{aligned} y_s(s) = y_0 + \frac{s}{24}[sin(\theta\frac{s}{8}) + 2sin(\theta\frac{2s}{8}) + 4sin(\theta\frac{3s}{8}) \\ +2sin(\theta\frac{4s}{8}) + 4sin(\theta\frac{5s}{8}) + 2sin(\theta\frac{6s}{8}) + 4sin(\theta\frac{7s}{8}) \\ sin(\theta(s)] \end{aligned} \tag{20}$$

A useful approximation to the X and Y position of the spiral at any given arc length point is defined by arc length parameter s. Returning to boundary conditions, an approximation for the path ending location is found and the boundary conditions can be written out in terms of the known parameters of the spiral.

Thus the setup for out path constraints to optimize becomes

$$\begin{aligned} x_s(s_f) &= x_f \\ y_s(s_f) &= y_f \\ \theta(s_f) &= \theta_f \\ \kappa(s_f) &= \kappa_f \end{aligned} \tag{21}$$

Specifically for autonomous driving path planning, there are tight curvature constraints. Cars have an absolute minimum turning radius and need to stay within lateral acceleration limits to maintain wheel traction and ride comfort in the vehicle. Additional constraints can be added on the parameters of the spiral and considering the initial and final boundary condition, there are curvature constraints as a function of the parameters of the spiral, and this leads to all of the required constraints to solve the optimization problem.

$$f_{be}(a_0, a_1, a_2, a_3, s_f) = \int_0^{s_f} (a_3 s^3 + a_2 s^2 + a_1 s + a_0)^2 ds \tag{22}$$

The fact that the objective function and its gradient have closed form solutions make it an objective function that is highly conducive to nonlinear programming.

Thus the final Optimization Problem for finding the parameters, now becomes

$$\begin{aligned} min f_{be}(a_0, a_1, a_2, a_3, s_f) s.t. \\ |\kappa(\frac{s_f}{3})| <= \kappa_{max} \\ x_s(0) = x_0, \\ y_s(0) = y_0, \\ \theta(0) = \theta_0, \\ \kappa(0) = \kappa_0, \\ |\kappa(\frac{2s_f}{3})| <= \kappa_{max} \\ x_s(s_f) = x_f, \\ y_s(s_f) = y_f, \\ \theta(s_f) = \theta_f, \\ \kappa(s_f) = \kappa_f \end{aligned} \tag{23}$$

Softening the equality constraints, to optimize for non-linear solvers,

$$\begin{aligned} min f_{be}(a_0, a_1, a_2, a_3, s_f) + \alpha(x_s(s_f) - x_f) \\ +\beta(y_s(s_f) - y_f) + \gamma(\theta_s(s_f) - \theta_f) \\ s.t. \\ |\kappa(\frac{s_f}{3})| <= \kappa_{max} \\ |\kappa(\frac{2s_f}{3})| <= \kappa_{max} \\ \kappa(s_f) = \kappa_f \end{aligned} \tag{24}$$

Although this allows the optimizer to violate the boundary condition equality constraints, the optimizer will be strongly encouraged to converge to a solution that is as close as possible to the boundary conditions before the bending energy penalty term will be large enough to influence the optimizer.

J. Conformal Lattice Planner

[12] The goal is to plan a feasible collision-free path from the autonomous cars current position, to a given goal state. The conformal lattice planner exploits the structured nature of roads, to speed up the planning process while avoiding obstacles. By focusing on only those smooth path options that swerve slightly to the left or right of the goal path, the conformal lattice planner produces plans that closely resemble human driving.The conformal lattice planner chooses a central goal state as well as a sequence of alternate goal states, that are formed by laterally offsetting from the central goal state, with respect to the heading of the road.

There's a key trade off when selecting goal state for path planning. A goal state that is close to the current ego vehicle position, reduction in the computation time required to find a path to the goal point. However, reduction in the ability of the planner to avoid obstacles farther down a path, in a smooth and comfortable manner. This can be problematic at higher speeds where the car will cover more distance in between planning cycles. For the figuratively simple case the goal point as the point along the center line of the lane, that is a distance ahead of the car equal to the goal horizon.

Once these probable goals states have been found, then the calculation of the spirals required to reach each one of them is undertaken. At this point, the collision is not checked, just kinematically feasible paths to each of the goal states are calculated. Therefore, the use the optimization formulation as discussed earlier, to solve for a cubic spiral, from the current location, to each end location. Once an optimization problem is solved, the resulting is the parameter vector. Once spiral coefficients are found, the the sample points along the spiral are extracted to get a discrete representation of the entire path. Since a closed form solution of the position along the spiral is not available, numerical integration is performed.

Using the collision checking techniques discussed earlier, removal of the infeasible paths from the set is undertaken. Most generally, a binary occupancy grid that contains one if a cell is occupied, and zero otherwise. Car footprint in terms of the cells of the occupancy grid are used, and they sweep the footprint out across each point in the spiral to generate the swath of the path. If the occupancy grid at a cell in the swath contains an obstacle, the path in question will collide with the obstacle, and should be marked as having a collision. If this never occurs for any of the cells in the swath along the entire path, the path is considered collision-free.

A simple metric is for the path and goal selection is the planner to select paths from the path set, that are as close to the center of lane state as possible while avoiding obstacles. This approach is implemented using a receding horizon planner, with the goal in the center of the lane receding at the end of the horizon.