Local Obstacle Avoiding Trajectory Generation using Dijkstra Algorithm

Project for

MEEN 689: Decision Making for Autonomous Vehicles

Shivam Vashi (334003552)

Motivation

Self-driving cars, like the one shown in Fig 1, need to navigate their way along a global trajectory which could be something as simple as a lane centerline. In order to follow a global trajectory, local trajectories are needed that have indispensable capabilities like obstacle-avoidance while also taking vehicle motion constraints into consideration.

Fig 1. A Self-driving Car

Literature Review

The paper by Werling et al. [1] provides an algorithm to generate jerk-minimizing local trajectories that avoid obstacles while obeying lane-width constraints.

The requirement is to minimize a cost functional- C(p(t)) which takes a local trajectory function (p(t)) as an input and returns a scalar representing the associated cost. The cost functional is written in the form shown below where T is the total time for which the local trajectory is designed and p_1 is the end state of the local trajectory i.e. $p_1 = p(T)$

$$C\big(p(t)\big) \;=\; k_j J_t\big(p(t)\big) \;+\; k_t g(T) \;+\; k_p h(p_1)$$

Note that T and p_1 are features of a trajectory p(t) and thus, the cost functional is a function of p(t); and all the coefficients (k) are weightages for corresponding terms.

Also note that a two-dimensional trajectory (p(t)) can be represented as (x(t),y(t)) with x(t) and y(t) being the x and y coordinates of the trajectory.

$$J_t\big(p(t)\big) \;= J_t\Big(\big(x(t),\, y(t)\big)\Big) \;=\; \int_{t_0}^{t_1}\left(\left(\frac{d^3}{dt^3}x(\tau)\right)^2 + \left(\frac{d^3}{dt^3}y(\tau)\right)^2\right)d\tau$$

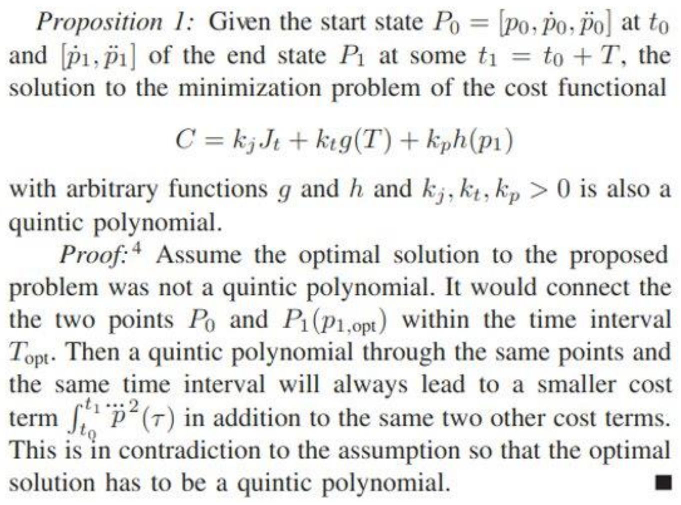
The goal is to find the structure of the local path function (p(t)) that will minimize the cost functional i.e.-

$$p^{\#}(t) \;=\; \big(x^{\#}(t),\, y^{\#}(t)\big) \;= \arg\min_{p(t)}\Big(k_j J_t\big(p(t)\big) \;+\; k_t g(T) \;+\; k_p h(p_1)\Big)$$

[2] proves that if we want to minimize just the jerk term (J_t), the x(t) and y(t) need to be quintic (5th degree) polynomials in t-

$$\big(x^{\#}(t),\, y^{\#}(t)\big) \;=\; \left(\sum_{i=0}^{5} a_i \cdot t^i\,, \sum_{j=0}^{5} b_j \cdot t^j\right)$$

Werling et al. [1] use this result as well as the following proposition (Fig 2) to prove that quintic polynomials will also minimize the cost functional defined earlier having additional terms along with the Jerk term.



Proposition 1: Given the start state $P_0 = [p_0, \dot{p}_0, \ddot{p}_0]$ at t_0 and $[\dot{p}_1, \ddot{p}_1]$ of the end state P_1 at some $t_1 = t_0 + T$, the solution to the minimization problem of the cost functional

$$C = k_j J_t + k_t g(T) + k_p h(p_1)$$

with arbitrary functions g and h and $k_j, k_t, k_p > 0$ is also a quintic polynomial.

Proof:[4] Assume the optimal solution to the proposed problem was not a quintic polynomial. It would connect the the two points P_0 and $P_1(p_{1,\text{opt}})$ within the time interval T_{opt}. Then a quintic polynomial through the same points and the same time interval will always lead to a smaller cost term $\int_{t_0}^{t_1} \dddot{p}^2(\tau)$ in addition to the same two other cost terms. This is in contradiction to the assumption so that the optimal solution has to be a quintic polynomial. ■

Fig 2. Proposition regarding the use of quintic polynomials for minimizing the cost functional [1]

Thus, the optimal local trajectory structure minimizing the desired cost functional is given by-

$$p^{\#}(t) = \left(x^{\#}(t),\, y^{\#}(t)\right) = \left(\sum_{i=0}^{5} a_i \cdot t^i \,, \sum_{j=0}^{5} b_j \cdot t^j \right)$$

Werling et al. [1] further suggest that in order to find an optimal local trajectory from the current vehicle state, a set of quintic polynomials with varying values of T and p1 is generated and then the trajectories that violate vehicle constraints (maximum velocity, acceleration, and curvature) and obstacle collisions are filtered out. Out of the remaining trajectories, the one with the minimum cost functional is chosen (shown in Fig 3)

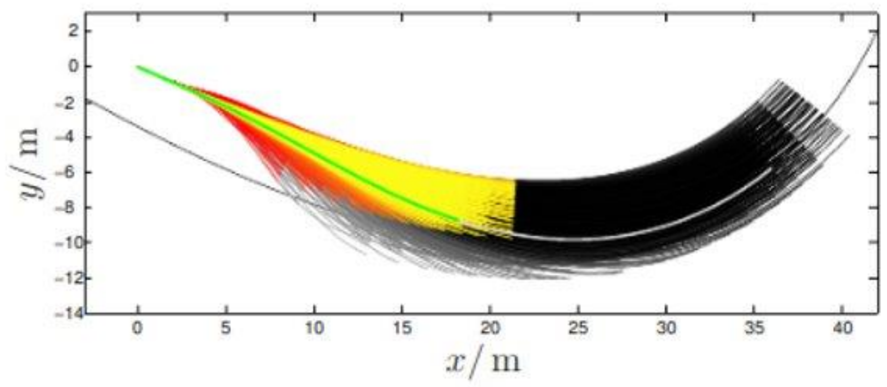


Fig 3. Colormap of the cost functional for the set of
local trajectories. Green is cheapest and Red is costliest.

Note that the end of each local trajectory has a lateral velocity of zero. This can be changed but the computation time would drastically increase because of an additional variable.

The algorithm has a few shortcomings as listed below-

1. Since the end of each local trajectory in the set has a lateral velocity of zero, the local trajectories generated before avoiding an obstacle will never converge to the global trajectory until the replanning occurs after passing the obstacle (Fig 4).

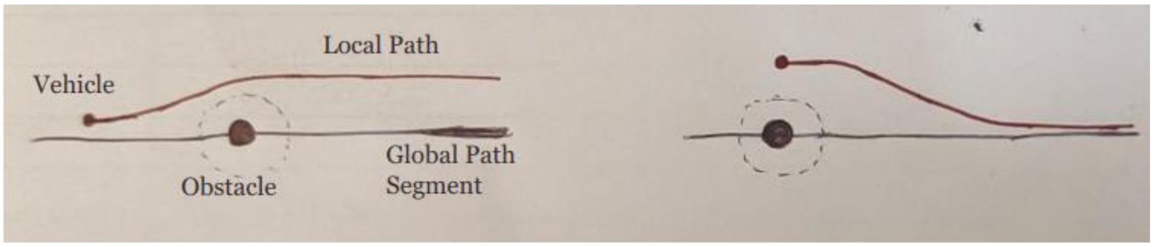


Fig 4. The Optimal Local Trajectory converges to the Global Trajectory
only after passing the obstacle

This implies that we must replan as frequently as possible to ensure optimality.

2. Since the algorithm generates all possible local trajectories from the current position, the computation time would explode if a local trajectory until the end of the global trajectory is to be designed.

An alternative graph-based approach to finding the shortest path from the current state to the end of global trajectory can be used. [3] provides a State Lattice approach to generate a graph of viable paths that the vehicle can take. Fig 5 illustrates the process of generating nodes and edges of the graph. Layers of nodes are first created along the global trajectory. Then the edges are developed using cubic splines fitted between the nodes of consecutive node layers as shown in the bottom of Fig 5.

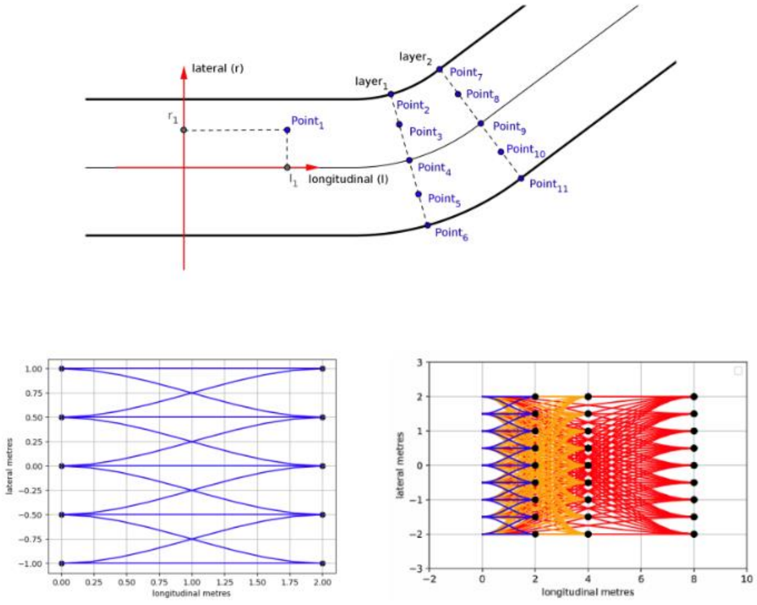


Fig 5. Graph Generation Scheme from [3]

The cost/weight (W) associated to ith edge (E$_i$) is defined as-

$$W(E_i) = k_{safe} \cdot w_{safe}(E_i) + k_{dist} \cdot w_{dist}(E_i) + k_{man} \cdot w_{man}(E_i)$$

where $w_{safe}(E_i)$, $w_{dist}(E_i)$, $w_{man}(E_i)$ are cost functions associated respectively to-

1. safety cost (closer an edge is to an obstacle, higher will be the safety cost),

2. lateral distance cost (higher the lateral distance of an edge from the global trajectory, higher will be the lateral distance cost) and
3. maneuver cost (a straight-line maneuver is cheaper than a curved one)

k_{safe} , k_{dist} , k_{man} are corresponding weightages.

The nodes and edges in the generated graph that collide with the obstacles will be eliminated and thus, we get a graph that can never return a path that would collide with the obstacles. Dijkstra's Algorithm will be used to find the cheapest path from the first node to the last node. Fig 6 shows the result of using Dijkstra Algorithm on a generated graph.

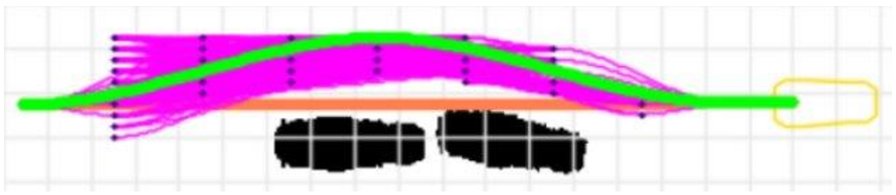


Fig 6. Example Implementation of the Algorithm provided in [3]

A shortcoming of this algorithm is that there is a lack of discussion on optimality between the node layers (edges) as there is no particular reason for the use of cubic polynomials.

Alternative Solution

A viable solution taking into consideration both the papers' strengths and weaknesses would be to use the Quintic Polynomial Local Path Generation scheme from [1] to generate the graph and obtain the optimal path using the Dijkstra's method as described in [3].

The algorithm developed to achieve this is discussed hereafter.

1. From the initial vehicle state, a set of quintic polynomial local trajectories are generated as described in [1]. The end points of each local trajectory become the start state for the next iteration. We can thus generate a desired number of 'node layers' as shown in Fig 7.

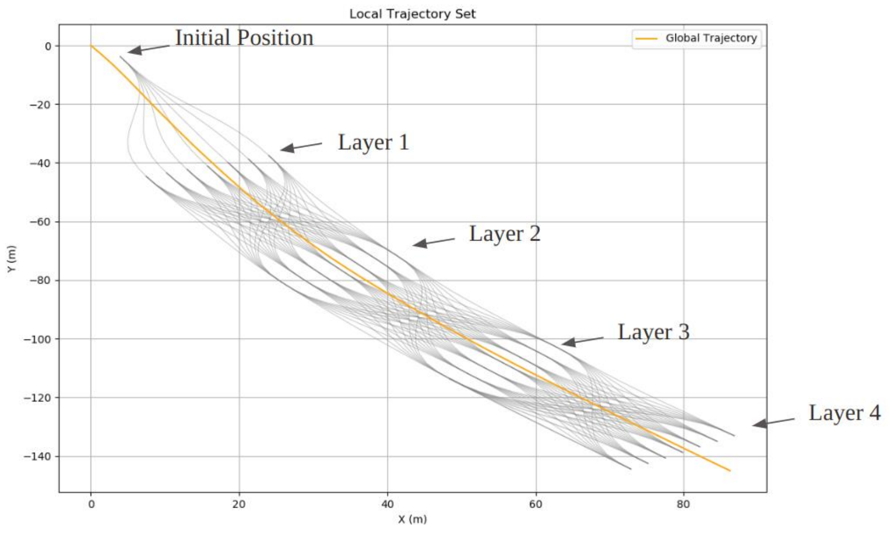


Fig 7. Generating a Set of Candidate Local Trajectories
Agnostic to the Presence of Obstacles

2. The paths that violate the vehicle constraints and obstacle collisions are removed from the set. From the remaining trajectories, each local trajectory is treated as an edge with its end points as nodes, thus creating a graph. The weight/cost of the edges is defined using the cost functional given in [1]. Fig 8 shows the final graph generated.

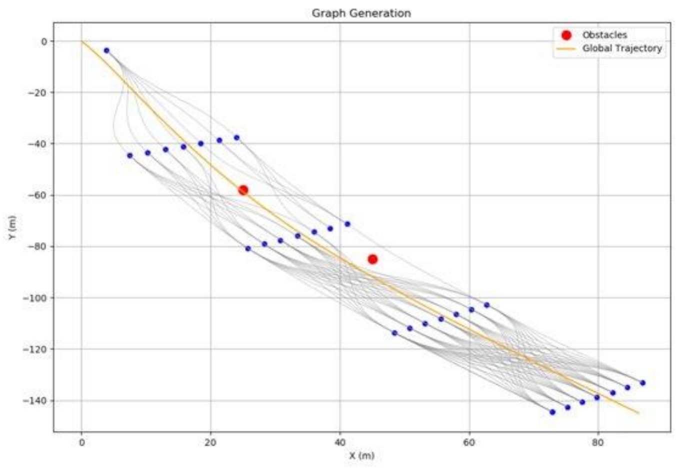


Fig 8. Filtering Invalid Trajectories and Generating the Final Graph

3. Finally, the Optimal Path is obtained using Dijkstra’s Shortest Path Algorithm. Fig 9 shows the local obstacle avoiding trajectory thus obtained in green.

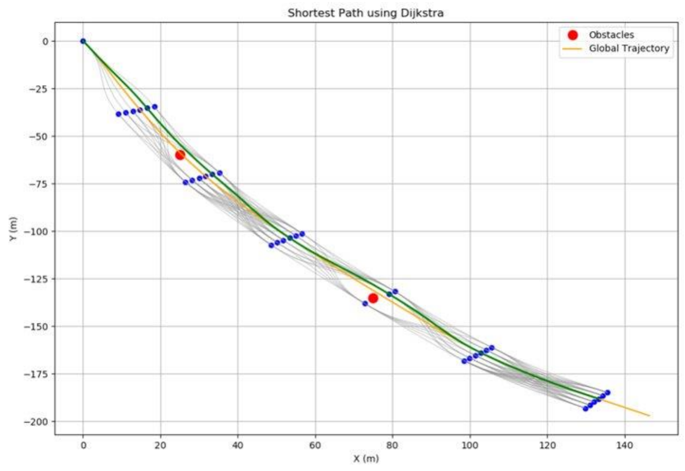


Fig 9. Local Obstacle Avoiding Trajectory Obtained by using Dijkstra’s Algorithm on the Graph

The same algorithm was deployed on a self-driving SUV to obtain the local trajectory that avoids obstacles (cones) along a global trajectory (lane centerline) as shown in Fig 10. An LQR controller was used to track the local trajectory demonstrating practical application of the Local Planner developed in this project.

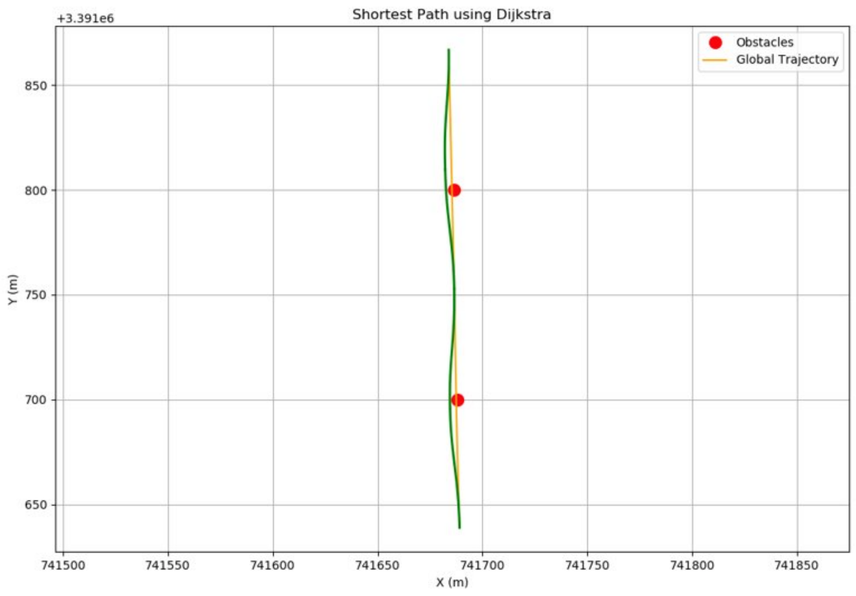


Fig 10. Deploying the Algorithm on a Self-Driving SUV to obtain the UTM Coordinates [4] of an Optimal Local Obstacle Avoiding Trajectory (shown in green)

Conclusions

The graph search method from [3] is upgraded by incorporating the path generation technique outlined in [1]. This enhancement introduces jerk-minimizing edges in the graph, contrasting with the previous method's reliance on cubic polynomials without considering any optimality features. The Optimal Local Trajectory obtained is deployed on a self-driving SUV to demonstrate practical application of the algorithm developed.

References

[1] M. Werling, J. Ziegler, S. Kammel and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame," 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 2010, pp. 987-993, doi: 10.1109/ROBOT.2010.5509799.

[2] A.Takahashi, T.Hongo, Y.Ninomiya, and G.Sugimoto. Local path planning and motion control for AGV in positioning. In IEEE/RSJ International Workshop on Intelligent Robots and Systems ’89. The Autonomous Mobile Robots and Its Applications. IROS’89.

Proceedings., pages 392–397,1989.

[3] Kornev, Ivan & Kibalov, Vladislav & Shipitko, Oleg. (2020). Local Path Planning Algorithm for Autonomous Vehicle Based on Multi-objective Trajectory Optimization in State Lattice.

[4] Wikipedia contributors. (2024, February 29). Universal Transverse Mercator coordinate system. In Wikipedia, The Free Encyclopedia. Retrieved 20:03, May 2, 2024, from https://en.wikipedia.org/w/index.php?title=Universal_Transverse_Mercator_coordinate_system&oldid=1211115793