Assignment 3 - Vehicle Motion Planning

Platforms and Algorithms for Autonomous Systems

Simone Bondi

February 20, 2024

1 Quick overview of the solution

1.1 The optimization problem

I formulated the optimization problem as a least-squares problem, where the input vector u is the optimization variable. I used the time-based linearized dynamic bicycle state-space model in Frenet space we obtained in class instead of the Kinematic model as suggested. I tried a NLP (Non-Linear Programming) problem first using the nonlinear Kinematic model and `fmincon`, but solutions took upwards of 4 seconds and often failed, too. With this QP (Quadratic Programming) problem, which i solve using `lsqlin`, each solution takes about 30ms for an horizon of 100 steps.

Assuming that the trajectory length is not that different from the reference path, i used a time-based model but parametrized the trajectory over the reference path length, given $ds = v_x dt$, where v_x is the longitudinal velocity.

1.2 Obstacle avoidance

Obstacles are strongly nonlinear (the solution space becomes concave). In the NLP problem i tried adding a quadratic APF of the form $-(l - l_{obs})^2$ in the cost function [WX23], or also a nonlinear inequality constraints based on an SDF (Signed Distance Function) - however, i did not manage to solve the problems i noted in §1.1.

Instead, i was inspired again by the work of [WX23] which employed an iterative solution strategy, in which i run multiple optimizations.

I employed a range based algorithm that supports a variable number of obstacles and of arbitrary convex shape - at each step, i cast normal rays to the right and left of the reference path, computing the lateral deviation ranges which coincide with the obstacles. Then, i negate the ranges to find the union of all feasible ranges at each step.

At this point, the basic idea would be to run the optimization for all combinations - however, there is an absurd amount of combinations, most of which are unfeasible. I therefore employ a basic graph-based algorithm to find all

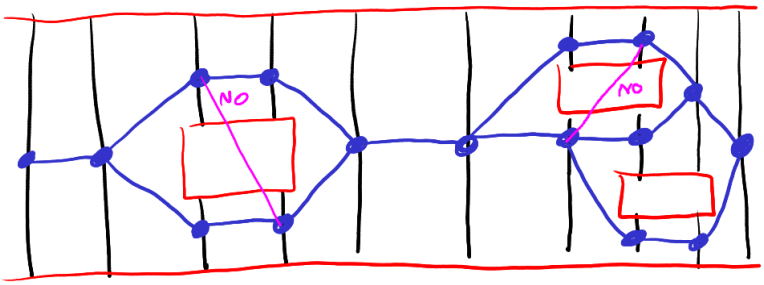

Figure 1: Obstacle avoidance graph. Violet combinations are rejected.

non-redundant paths, and then run the optimizer on each of these. Figure 1 shows the idea - with 3 obstacles laid as in the figure, counting the possible paths shows that the algorithm will run the optimizer $2 * 3 = 6$ times. Assuming each optimization takes 30ms (which is the case for an horizon of 100), the algorithm will take 180ms. However, using arbitrary shapes does not impact the optimization time, and instead only impacts the raycasting phase execution time.

1.3 Output path

My path planner returns the spline interpolation of the trajectory in Frenet space w.r.t to the reference path instead of a self-contained reference path - this allows me to avoid having to move between two frame of references when setting the initial Frenet state-space vector in the optimization (recall that my optimizer is based on the linearized dynamic model in Frenet space). This means i write the lateral error written as $e(s) = x - x_{traj}(s)$ and the road angular rate input component as $\dot{\phi}_{des} = kv_x + \omega$, where k is the reference path curvature and ω is the heading deviation rate component of the Frenet space state-space vector.

2 Comments on task results

You can run the specific task initialization script such as `task1_init`, then plot the results with the common `taskX_plot`. There's also a `live_plot` script which plots the trajectory in real time.

2.1 Path following

An effect of real-time path planning can be seen here. Given the fact that the planner should plan the motion of the vehicle, it makes sense that the optimization starting point should be the vehicle position itself. However, this results in a jagged steering input every time we update the planned path, as the control error basically returns to zero. I mitigated it by reusing the old path on

the first iteration after a path update, however i did not have enough time to find a solution to this problem. Prolonging the planning interval shows great improvements to the jagged trajectory - instead, varying the speed as specified by the assignment handout shows no difference.

2.2 Turning at an intersection

I set the optimizer lateral deviation bounds to $[-1.5, 1.5]$ to the reference path. The optimizer correctly uses the most of the lane, even extending to 1m of lateral deviation before entering the turn, to minimize both the actuator usage and deviation. The resulting actual path lateral deviation is well within the assignment specification, being in the $[-1, 0.6]$ range.

2.3 Static obstacle avoidance

The planner correctly attempts to stay on the right lane, as specified - it successfully returns to that position when the maneuver has been completed. Moreover, given the ray casting nature of the algorithm, it correctly identifies both obstacles even though the second is not aligned with the road (for simplicity, i use AABBs, or Axis-Aligned Bounding Boxes, aligned with the global frame).

At 90km/h the trajectories the vehicle takes around the obstacles are very smooth, taking upwards of 100 meters to return to perform the whole avoidance. At 50km/h, the maneuvers are completed successfully in about 50m.

References

[WX23] Yi Wei and Haiqin Xu. "Path Planning of Autonomous Driving Based on Quadratic Optimization". In: *2023 9th International Conference on Control, Automation and Robotics (ICCAR)*. 2023, pp. 308–312. DOI: 10.1109/ICCAR57134.2023.10151702.