Writeup

Student describes their model in detail. This includes the state, actuators and update equations.

The state is composed of px, py, psi, v, cte, epsi. This is passed into MPC's Solve() method, which returns as actuators delta, a (steering value, acceleration). In order to compute these, the MPC implements a couple of model constraint and lower/upper bounds equations. These are very similar to the ones presented in the lectures.

The only interesting difference from the lecture's quiz implementation, was in calculating the cost function. Overall, it has the same structure as presented in the lectures, but an important addition to it was adding weights for each category of costs. This was by far the most time consuming part of the project, since the task was not trivial at all. My approach was to start with a reference velocity ref_v of 40, and let the model run without any weights, and see how it behaves. It was terrible. It was running off the intended trajectory very fast after the simulation began. Therefore, my next approach was to tweak the weights (penalties) of using the steering actuator, through delta_w. Moreover, I did not want to have too abrupt steerings, so I also played around with delta_gap_w, which is the weight applied to the cost of the difference of sequential actuations. After a few iterations, I was able to find the values that worked. These were also the values that I stuck with until the end of experimentation: delta_w = 9000, delta_gap_w = 900. The penalty for using the steering actuator had to be pretty large. Moreover, I had the cte_w set to a value somewhere between 50-100, not remembering the exact value.

I thought that this would give me decent results for when I bumped the ref_v to 50 as well. But I was wrong. After the first turn (i.e. on the bridge straight line), the car started shuffling left and right. At this point I realized that as I increased the reference velocity, I needed to put less *importance* on the cte cost. Therefore, I started decreasing the cte_w (by half) for each increase of ref_v. Along with these changes, I started playing around with the a_w weight, telling the system not to accelerate too much, or too abruptly (a_gap_w).

Eventually I was actually able to run the car at 100+ mph across the track, but this was not 100% reliable, so I am leaving the submission with values that make it run at around 90 mph. I am satisfied with these results, although it feels that if I put less weight on cte and tweak the acceleration actuator weights a bit more, I could get to more reliable 110+ mph.

Student discusses the reasoning behind the chosen N (timestep length) and dt (elapsed duration between timesteps) values. Additionally the student details the previous values tried.

I started with the values presented in the lectures quiz: N = 25, dt = 0.05. I played a bit with these values, by lowering N considerably, but the car did not perform as well as with my initial value, probably due to the fact that it was getting a limited see-ahead. In terms of dt, I was happy with the initial one due to being smaller than the system latency, so I think this helped the MPC predict the positions.

If the student preprocesses waypoints, the vehicle state, and/or actuators prior to the MPC procedure it is described.

The only preprocessing that I have done was the one related to adding latency. Moreover, in order to continue adding the latency at next iterations, I needed to store the previously used actuators in order to use them for the current step in predicting the vehicle's position and orientation.

The student implements Model Predictive Control that handles a 100 millisecond latency. Student provides details on how they deal with latency.

In order to deal with the latency, I calculated where the car would be after the latency (in our case 100ms), and used that position and orientation as input to the MPC. In this way we simulate where the car would be after the latency is applied, so that the MPC can give us results for that future position instead of our current one.