Craig Vargas
SDCND
MPC Write-up

MPC Implementation

State

The state of my MPC implementation is maintained through four data points:

X – Cartesian coordinate representing position
Y – Cartesian coordinate representing position
Psi – Vehicle bearing in radians
V – Vehicle speed

For each data point I maintain a temporal series of ten predictions for the data point through time. All of these data points across the different state variables are stacked into one vector.

Actuators

The actuators used to control the vehicle are the throttle and the vehicle's steering system. The throttle takes in values from -1 to +1 inclusive, which correspond to full break and full throttle respectively. The steering system also takes in values from -1 to +1, corresponding to full left turn and full right turn respectively. As with the state variables, I maintain a temporal series of actuator updates for each actuator. The number of actuator updates are one less than the number of state predictions because each actuator instruction is associated with a transition from one state to the next. The series of actuator updates are also stacked into the same vector as the state.

Update Equations

The update equations are based on a kinematic model that defines the state of the vehicle in terms of the previous state and the most recent actuator instructions. I use this relationship to define constraints for an optimization problem whose most basic goal is to keep the car as close to the desired path as possible. The solver algorithm computes a set of actuations for the next nine steps of the vehicle through time, after which the first of these actuation instructions is passed to the vehicle's control system. After that the next state of the car is passed back to the MPC and the optimization problem is reinitialized and solved again. This loop continues as long as the vehicle is in motion.

I mentioned earlier that the most basic goal of the optimization problem is to keep the car as close to the desired path as possible. There are other goals as well:

1. Keep the bearing of the car as close to the desired bearing as possible. This helps ensure that the car does not veer off path.
2. Keep the velocity of the car as close to a predefined reference velocity as possible. This helps to ensure that the car does not stop in the middle of the road as a means of maintaining the more basic optimization goals like staying on the desired path.
3. Keep the use of the actuators as low as possible. This is important to keep the ride "smooth". Without accounting for this factor the car would break and accelerate aggressively, as well as turn a lot back and forth, effectively swaying the car from left to right repeatedly. In my implementation, minimizing the use of the steering actuator was given quite a drastic weight of 500 times all other weights. This was the only way I could get the car to not overshoot turns that were meant to put the car back on course.
4. Keep the values of consecutive actuations as close together as possible. In other words, if the throttle instruction was +1 in time step t, we do not want the next throttle instruction to be -1. Instead we prefer a smooth progression into the breaking sequence. This smoothing out of the consecutive actuations was another possible place where I could have given a large weight to the steering portion in order to prevent the car from overshooting its turns. I didn't do it in my implementation however, because the importance given to the overall use of steering alone sufficed.

Time Variables

There are two variables in my implementation that pertain to time: N and dt. N is the number of steps that the model attempts to predict the state for, and dt is the duration of time between each step. The total prediction time for each iteration of the model is T = N*dt. Since we are dealing with the motion of a car, it does not make sense for T to be 10 minutes or even 30 seconds. Too much is subject to change over the course of even 10 seconds while driving a vehicle, so it does not make sense to waste time predicting its state 10 seconds in the future with only a current snapshot of time. I chose N=10 and dt=0.1. The variable dt was chosen as 0.1 largely because this is the delay assumed between sending an actuation decision and when the actuator would respond. Setting dt equal to the actuation delay was a simple way ensure that the next prediction state would actually corresponded to a time point in time just after completion of the actuation. Setting N equal to 10 seemed like the minimum logical value for this variable since anything lower would make the prediction horizon less than a second. I could have used N=20, since a two second horizon is reasonable as well, but the model worked well with N set to 10 and increasing N has a negative effect on the model in that it increases the computation time.

Polynomial Fitting

One of the inputs to the MPC is a set of waypoints for the car to follow. The location of each waypoint is given with respect to the map. I transformed the waypoint information from the map's coordinate system to the car's coordinate system. This preprocessing was done out of

convenience to display the car's path, but also made it easier to maintain the MPC's state model. A second order polynomial was fit to the waypoints on the map in order to calculate a desired path for the car to follow. The track was relatively simple so a second order polynomial sufficed for a path for the car to follow one second in the future.

Latency

There was a 100 millisecond delay between the actuation instruction and the execution of that instruction. This latency in the system was dealt with by predicting the car's state 100 milliseconds in the future and assuming that predicted state to be the initial state of the system. This allowed the actuation instruction to be better aligned with the car's state upon execution. The kinematic model was used here to predict the car's state after the latency period.