

# Writeup for Model predictive control Project

## Part of Udacity self driving car nanodegree

Sven Eriksson

September 11, 2017

Text in headings and in bold comes from the project rubric. My answer to the question are written just underneath.

## 1 Compilation

**Your code should compile. Code must compile without errors with `cmake` and `make`. Given that we've made `CMakeLists.txt` as general as possible, it's recommend that you do not change it unless you can guarantee that your changes will still compile on any platform.**

Have not modified the `CMakeList.txt` and it compiles and run outside of CLion (the IDE I am using). So it should work.

## 2 Implementation

### 2.1 The Model

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

Used the model described in the previous sections: 'Vehicle Models' and 'Model Predictive Control'.

State consists of 2D-position ( $x$  and  $y$ ), speed ( $v$ ), heading ( $\psi$ ), crosstrack error ( $cte$ ), and orientation error ( $e\psi$ ). The actuators were acceleration ( $a$ )

and steering ( $\delta$ ).

$L_f$  is a vehicle specific parameter and a factor between the turning rate and the steering angle time speed.  $f(x)$  is that value (distance right/left) of the polynomial at distance  $x$ . The polynomial is fitted according to given waypoint in the middle of the road that has been translated into the vehicle coordinate system.  $\psi_{des}(x)$  is the direction of the tangent to the polonomial at a certain  $x$ .

The update function was:

$$\begin{aligned}x_{t+1} &= x_t + v_t \cdot \cos(\psi)dt \\y_{t+1} &= y_t + v_t \cdot \sin(\psi)dt \\ \psi_{t+1} &= \psi_t + \frac{v_t}{L_f} \delta_t dt \\ v_{t+1} &= v_t + a_t dt \\ cte_{t+1} &= f(x_t) - y_t + v_t * \sin(e\psi_t)dt \\ e\psi_{t+1} &= \psi_t - \psi_{des}(x_t) + \frac{v_t}{L_f} \delta_t dt\end{aligned}$$

## 2.2 Timestep Length and Elapsed Duration (N & dt)

**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.**

dt was chosen to be 0.1s as that was the minimum time between actuation.

The following values for N was tested: 20, 15, 12, 10, and 8.

The different values were tested with cost function tuned with N=20 and dt=0.1.

The yellow line is made up of the waypoints that are provided and the green line is made up of  $x$  and  $y$  values from different timesteps from the MPC solution.

**N=20:** Works well, the cars stays within the track and the green line mostly aligns with the yellow line.

**N=15:** Works well, the cars stays within the track and the green line mostly

aligns with the yellow line.

**N=12:** Works well, the cars stays within the track and the green line mostly aligns with the yellow line.

**N=10:** The car does occasionally touch the edges of the track and I see more oscilations.

**N=8:** The car left the track in the first sharp turn.

N=20 was picked as it worked without any problems and I did not see any advantage of using a lower N. There is probably an advantage is reducing the number of timestep to save on computations but I had no problems with computational time.

In later subsections I will explain how the first timestep ( $t=0$ ) was forced to have certain actuator values and should perhaps not be counted. In that case reduce N by one.

## 2.3 Polynomial Fitting and MPC Preprocessing

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

A cubic polynomial was fitted to provided waypoints. The waypoints where first expressed in  $x$  and  $y$  of the vehicle coordinate system before the polynomial was fitted.

As 100ms latency before actuation existed I decided to use  $t = 1$  instead of  $t = 0$  as my solution from the MPC. The actuator values for  $t = 0$  was set to be the previously choosen values / current actuator values. This made the state at  $t = 1$  fixed as the state and actuator values of  $t = 0$  were given. The benefit of doing this first update inside the MPC optimizer is that I can account for the derivative of steering and acceleration values between  $t = 0$  and  $t = 1$ . This allowed me to force a solution that were more in line with the previous one, thus creating a smoother path. When not accounting for the current actuator values in my cost function by doing the initial update outside of the MPC optimizer I got some oscilations in the steering output.

## 2.4 Model Predictive Control with Latency

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

See previous subsection, with a single state update to time  $t+1$  before using MPC. Also forcing a smoother steering by accounting for the derivative in steering between  $t = 0$  and  $t = 1$ .

Also using the solution for the actuator values at  $t+1$  (100ms in the future).

## 3 Simulation

**The vehicle must successfully drive a lap around the track. No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle). The car can't go over the curb, but, driving on the lines before the curb is ok.**

I decided to hand the project in with a target speed of 50mph. I did attempt to tune the cost function for higher speeds but the car did occasionally leave the track. As there was no requirement on speed I left it as 50mph and continued with the other projects.