Model Predictive Control Project

Report

Author: Aneeq Mahmood

Email: aneeq.sdc@gmail.com

Steering a car along the track using a model predictive controller (MPC)

The following goals have been achieved in this project

- Designing of a MPC to steer a car along the track in the simulator using C++
- The car does not leave the road during a lap around the track
- A video output.mp4 showing the above
- The project compiles with make and cmake
- A Write up, which reflects upon the implementation

Implementation of the MPC

1. The Model:

For the MPC to work it requires a set of states, which represent the system dynamics. Moreover, it requires the control inputs for the plant. In this case, the plant is a car which requires throttle/accelaration (a) and direction (steering, δ) to stay on the road. The goal here is to keep the car on the road and hence localize it on the middle of this road. Thus, the states of the system are :

States = { x,y,ϕ, v,cte, eϕ}

Where

x: x coordinate of the car's location

y: y coordinate of the car's location

ϕ: orientation of the car

v: velocity of the car

cte: Cross track error (cte), which is effectively how far the car is from the middle of the lane)

eϕ: residual error in orientation of the car

The update equations which predict the values of the above states at time *t+1*, provide we know the state at time *t* are based on the global kinematic model and are:

$x^{t+1} = x^t + v^t * \cos(\psi^t) * dt$

$y^{t+1} = y^t + v^t * \sin(\psi^t) * dt$

$\psi^{t+1} = \psi^t + v^t * (1/Lf) * \delta * dt$

$v^{t+1} = v^t + a^t * dt$

$cte^{t+1} = cte^t + vt * \sin(e\psi^t) * dt$

$e\psi^{t+1} = e\psi^t + v^t * (1/Lf) * \delta * dt$

where dt is the time difference between two consecutive measurements in seconds, and Lf is a constant.

Based on the above state vector and state equations along with the (δ,a) input at time *t*, the desired (δ,a) value at time *t+1* will be calculated inside the MPC.

2. Timestep Length and Elapsed Duration (N & Δt)

A MPC works over a time horizon which has N discrete steps, and within each step, inputs arrive at a certain time period which is Δt (denoted as dt in the above state equations). Hence, if N has a value of 5 and Δt is 0.5, then the MPC will predict outputs for a time duration of 5 times 0.5 ,which is 2.5 s. This value is also called the time horizon of the MPC. In MPC, the goal is to find the optimized (δ,a) values at each time step Δt, so that the optimization cost function is minimized over the whole horizon.

If one wants to predict over a large horizon for a fixed Δt, then it will require a large N. For a system which is not dynamically changing, a large N should not be a problem. Although, it will require large computation capability and possibly long calculating time, but as the system does not change drastically over time, predictions will be valid for a longer time. In the case of a car, the terrain and conditions can change over a very short time frame (for example, in a few seconds). Thus, longer predictions with a large N will not be useful.

On the other hand, a too small value of N will lead to faster optimization, but this will not enable the MPC to adapt quickly, and might lead to decreased performance such as very quick deceleration, or sudden turning of the steering, to adapt to changing road conditions. For this project, a small N (e.g. 5) has led to car leaving the road when a sharp turn comes up. A larger N (e.g. N = 14) works fine but higher values lead to longer processing time, which slows down the simulator. A final value of N=10 was eventually chosen.

The value of Δt can also affect the performance; for a fixed N, a small Δt will involve more have a small horizon, but will integrate more information confirming form the sensors. This would lead to minor adjustments in the car throttle and steering every time the controller sends its ouputs. This will be useful if the motion model of the car uses a dynamic model which incorporates effects of different forces on the tyres and on the car. However, for the simpler global kinetic model used in this project, a smaller value will not be of much use. A too large value of Δt will lead to a larger horizon, which again will not adapt too well to changing environment. For this work, Δt = 0.2 was

chosen. Together with N = 10, this gives a horzion time of 5 seconds. Hence for a car moving at 30 miles per hour, this horizon will look ahead for more than 50 m, which is sufficient for the simulator. Another reason of choosing this values was to avoid the impact of propagating latency of information coming from sensors, which will be discussed further in the last section of this document.

3. Polynomial Fitting and MPC Preprocessing

The simulator provides the future trajectory of the car in terms of x-y coordinates. The car can follow this trajectory by obtaining a curve defined by a polynomial. This then becomes a typical curve fitting problem where the x – coordinates are the input and the polynomial, fitting to the (x,y) points of the trajectory, will led to future direction of the car. However, the (x,y) coordinates of the future trajectory are points in a global map, and will need to be first converted to coordinates with respect to the car. As the direction of motion of the car is taken to be x-axis in 'car coordinates', and the car itself is considered to be the origin (0,0), the output of the curve fitting problem will give us the desired coordinates to drive upon. The conversion from global map coordinates to car coordinates is done between line 121-and line 126 in the mapin.cpp file

The code to obtain a polynomial for a curve fitting problem was given in the lectures and was subsequently used in the project.

4. Model Predictive Control with Latency

In a real-life situation, the information coming from sensors, GPS, cameras, etc is not instantaneous and requires a certain propagation latency. If this latency is fixed, its impact can be handled using the state equations and the next state can be determined over this fixed latency. The simulator offers a latency of 100 ms, which is tackled by determining the next state of the state variables over this latency, and giving this value as the current state to the MPC solver.

Hence, if { x,y,ϕ, v,cte, eϕ} is the current state, with (δ,a) being steering and throttle values, then their values over a fixed latency are updated as follows.

x += v * cos(ϕ) * latency;

y += v * sin(ϕ * latency;

ϕ += v * δ* latency / Lf;

cte += v * sin(eϕ) * latency;

eϕ += v * δ * latency/Lf;

Here it should be noted that value of Δt (which defines the time period of measurements going into the MPC) should be larger than the latency. This will guarantee that the impact of latency is well preserved into the initial state, and will be accounted for in future state estimations. Hence, this was another reason to use Δt =0.2, although Δt =0.1 worked also fine in the case where the latency was not present.

One key observation made during this project was that without latency, the car remained easily on the track without putting extra weights on the parameters in the cost function. However, with

latency being introduced, the car followed a zig zag path and would soon derail. To avoid this situation, certain weights were added to the cost function. In particular, to avoid, the zig-zag behaviour, very large weights were introduced in the cost function to ensure that steering movements are kept at minimum, and if there are steering changes, their successive differences are kept small. Weights on CTE and other parameters of the cost function were also introduced and were adjusted mostly via hit and trial.

For a certain set of weights, MPC works fine for 30 mph, and car will stay on the road even if the speed if chosen to be 40 mph.For higher values, the car will move again in zig-zag pattern, which will require manual tuning of the weights.