Project 5 - Model Predictive Control

Vaidehi Venkatesan

March 19, 2018

Compilation

- Code compiles using cmake 3.12 and make 4.2.1. CMakeLists.txt has been modified to add new source files

Run commands

The program can be run by executing the following command:

- ./mpc

Implementation

Model

A kinematic model is used as a simplified version of actual dynamic model the vehicle used in the simulation.

The state of a vehicle is defined by a state tuple $[x, y, v, psi]$ where,

- (x, y) - defines the coordinate position of the vehicle
- ψ - defines the orientation of the vehicle
- v - defines the velocity of the vehicle in motion

In order to determine the state evolution of a vehicle, we define the following actuator inputs (a.k.a. control inputs)

- δ - defining the steering wheel angle of the vehicle
- a - defines the acceleration / deceleration of the vehicle (otherwise referenced as throttle value in code)

Give a state of vehicle X, and a set of actuator inputs [δ, a] at time 't', X_{t+1} is defined by the following update equations:

- $x_{t+1} = x_t + v_t * cos(\psi_t) * dt$
- $y_{t+1} = y_t + v_t * sin(\psi_t) * dt$
- $\psi_{t+1} = \psi_t - \frac{v_t}{L_f} * \delta_t * dt$
  - This is taking into account the simulator's steering pattern, where positive values implies right turn (clock-wise rotation) and negative value implies left turn (counter-clock wise rotation)
- $v_{t+1} = v_t + a_t * dt$

L_f measures the distance between the center of mass of the vehicle and it's front axle and defines the turn rate of the vehicle. The larger the vehicle, the slower the turn rate is. L_f for this project is defined as a constant value of 2.67.

As described in the lecture videos and project Q & A session, this value is validated after runnign a vehicle around in circle and ensuring the radius of the circle using the constant velocity and steering angle is similar to the value used in simulation. For most mid-sized cars, on an birds-eye measure, the distance between front axle and center of the vehicle measures close to 2.5 times radius of the wheel.

Timestemp Length and Elasped Duration (N & dt)

$N = 10$ and $dt = 0.1$ is chosen for the project. These values were indeed recommended by the instructors during Q&A session of the project. This combination of N and dt also helps in introducing latency while using actuator inputs. The vehicle's predicted trajectory smoothly overlaps the marked trajectory and the car drives around the lane smoothly without a lot of wavering or driving off the curbs.

Other value pairs tried for N, dt are : [20, 0.05], [24, 0.5]. These values along with stricter constraints and latency metrics does not provide good results. The vehicle wavers around the center of the lane rapidly.

These parameters are set at the start of *MPC.cpp*

Polynomial Fitting and MPC Preprocessing

Waypoint inputs used from the simulator are pre-processed to align them to $(0, 0)$ and ψ is rotated to 0 angle to align with x-axis. This simplifies the initial state for every input to the MPC solver.

polyfit is used with rotated waypoints to come up with the line coefficients which are further used with *polyeval* to compute the cross track error and psi.

main.cpp: 113-129 showcase this logic.

MPC solver is then called with the rotated state and coefficients.

Model Predictive Control with Latency

In order to handle 100 millisecond latency at each step, it is important to calculate the number of steps that the algorithm should go back to get the actuator inputs at each time step t.

100 milliseconds = 100 * 0.001 = 0.1 *sec*

Since, $N = 10$ and $dt = 0.1$, each time step t amounts for 0.1 s of time. Hence in order to introduce an additional latency of 0.1 s, at each time step t, the previous time step $t - 1$'s actuator inputs are used to compute the model constraints.

This code is introduced in *MPC.cpp: FG_eval::operator():121* line

Simulation

The vehicle successfully laps around the track.

Demo Link: Simulation Video With MPC can be found in Results folder (MPC_Output_480p.mov)