Model Preditive Control

Vilas Chitrakaran

July 16, 2017

Abstract

This is a description of my implementation of Project 10, Model Predictive Control in the Self Driving Car Engineer Nanodegree.

Introduction

This was a challenging, but personally speaking the most enjoyable assignment in Term 2 of this course (the Particle Filter comes a close second). Udacity provided the a basic template in which to implement the coding assignment. The following sections describe the bits we had to implement, as required by the rubric.

Model

Vehicle Motion Model

The vehicle state vector is $\begin{bmatrix} x & y & \psi & v & e^{cte} & e^{\psi} \end{bmatrix}^T \in \mathbb{R}^6$, constituted by planar position, orientation, linear velocity, cross track error and orientation error, respectively. For every time step ΔT, the evolution of state is governed by the following kinematic model

$$\begin{aligned} x_{t+1} &= x_t + v_t \cos(\psi_t)\Delta T \\ y_{t+1} &= y_t + v_t \sin(\psi_t)\Delta T \\ \psi_{t+1} &= \psi_t + \frac{v_t}{L_f}\delta_t \Delta T \\ v_{t+1} &= v_t + a_t \Delta T \\ e^{cte}_{t+1} &= \left(y_t - y^{des}_t\right) + v_t \sin(e^{\psi}_t)\Delta T \\ e^{\psi}_{t+1} &= \left(\psi_t - \psi^{des}_t\right) + \frac{v_t}{L_f}\delta_t \Delta T \end{aligned}$$

where L_f is a constant design parameter representing the distance from the front of the vehicle to its CoG, $y^{des}_t(t)$ is the desired lateral position of the vehicle such that $\left(y_t - y^{des}_t\right)$ represents the cross track error at time t, $\psi^{des}_t(t)$ is the

desired orientation of the vehicle, $\delta_t(t)$ represents the steering angle and $a_t(t)$ is the throttle. Both $y_t^{des}(t)$ and ψ_t^{des} come from the reference trajectory that the vehicle is expected to follow. How this trajectory is generated is described in a following section.

Actuator Constraints

The control inputs to the vehicle are steering angle $\delta \in [-|\delta_{max}|, |\delta_{max}|]$ and a trottle value $a \in [-1, 1]$ where I chose $\delta_{max} = 25$ degrees. The controller generates new steering and trottle commands subject to the limiting contraints of the range of actuation defined above and the minimisation of a cost function described in a later section.

MPC Prediction Horizon

I experimented with a few prediction horizons and eventually settled on a value of 1 second into the future. Any shorther and the controller wasn't able to look sufficiently far ahead to generate optimal control inputs that would pull the vehicle back on the desired trajectory at high speed, given the curviness of the simulated race track. On the other hand, longer horizons generated highly erroneous predictions perhaps due to the approximate nature of the vehicle model that ignored complex vehicle and actuator dynamics.

Once the prediction horizon was selected, the next step was to decide on the number of timesteps N and length of each timestep ΔT. Here, I chose the smallest N or conversely the largest ΔT I could get away with, as the computational cost increased in proportion to N. We were asked to simulate a propagation delay of 100ms in control signals reaching the vehicle actuators. I first tried $N = 10$ ($\Delta T = 0.1$). This resulted in quite a lot of zigzagging, especially at start. The vehicle able to reach smooth motion after tweaking the scale factors in the cost function but I couldn't really make the vehicle exceed an average speed of about 50 mph safely. Setting $N = 20$ ($\Delta T = 0.05$) smoothed out the predictions considerably, and the vehicle could now go much faster at about 65 mph given the better resolution of control. So thats what I chose - the lowest possible control resolution at which I could go sufficiently fast around the bends to be considered suicidal in the real world.

Polynomial Fitting and MPC Preprocessing

As mentioned previously, the desired trajectory was provided. This was in the form of waypoints, and a third order polynomial fit provided a sufficiently smooth trajectory for the vehicle to follow. The curve-fit function $f(x)$ provided a rather nice analytical way to compute the desired cross track position and orientation terms in the model equations as $y^{des} = f(x)$ and $\psi^{des} = \tan^{-1}(\frac{df(x)}{dt})$

as follows

$$\begin{aligned} y^{des} &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 \\ \psi^{des} &= \tan^{-1}(a_1 + 2a_2 x + 3a_3 x^2). \end{aligned}$$

The waypoints defining the desired trajectory was first transformed to vehicle coordinates at every timestep as this simplified the computations significantly (the 'current' x,y,ψ would reduce to 0 at the start of the optimisation timestep).

Model Predictive Control with Latency

The optimiser was setup with the vehicle model and actuator constraints as described previously, and programmed to minimise a cost function that penalised the cross track error, the orientation error, deviation from a reference velocity set to 80 mph, magnitude of actuation signals and magnitude of change in actuation signals. The relative 'penalties' were as follows

```
const double errorPenalty = 0.01;
const double actuationPenalty = 25;
const double steeringChangePenality = 1000.0;
const double accelerationChangePenalty = 10.0;
```

To handle the communication latency of 100 ms that delayed control commands reaching the actuators, the vehicle model utilised control inputs from 100ms in the past. Since $\Delta T = 0.05$, this meant using control inputs from 2 timesteps ago rather than the current control inputs to predict where the vehicle would be in the next time step; i.e. the δ_t and a_t terms in the vehicle motion modeling equations were replaced with $\delta_{t-2}$and a_{t-2}, respectively.

Video

A video of the final output is available at `https://youtu.be/gweVAqsuLU0`.