Model Predictive Control in Vehicle Sim

Henry Yau

March 21, 2018

MPC Project

The goals / steps of this project are the following: * Fit polynomial to waypoints to generate desired vehicle path * Implement MPC controller to control throttle and steering of vehicle in simulator * Tune parameters to allow vehicle to travel safely around track

Controller description and implementation

1. Interpolating Polynomial Path

The desired path used by the controller is represented by an interpolating polynomial. The interpolating polynomial is generated by solving a least squares fitting problem using the center line of the road whose points are provided in a JSON object. There is an assumption that we are primarily concerned with the immediate turn and maintaining a center position on the track. That is, we are not concerned with sequential left right turns like a chicane. With this assumption, a quadratic interpolating polynomial is sufficient to generate the desired path which evaluated up to a prediction horizon.

One thing to note regarding the interpolating polynomial is that we assume the origin always begins at the vehicle's center and is oriented with the x-axis pointing in the vehicle's x-axis. This requires subtracting the vehicle's center from the center line points used to generate the polynomial. The points are then rotated by $-\phi$ where ϕ is the vehicle's current heading. This also means that the initial states (x, y, ϕ) of the vehicle provided to the solver are zero.

2. Model Predictive Control

Model Predictive Control (MPC), also known as Receding Horizon Control, computes the control at each time step by solving an open loop optimization problem for the prediction horizon. This generates a vector of future control inputs and a vector of predicted outputs using a system model. The first value of the future control input vector is applied to the actual plant. At the next time step, the systems states from the plant are used in the system model to compute the future input and future predicted output.

The open source libraries CppAD (https://projects.coin-or.org/CppAD) and Ipopt(https://projects.coin-or.org/Ipopt) are used to solve this quadratic programming problem.

The class `FG_eval` implements an object that evaluates the cost functional $f(\bar{x})$ and the constraints $g(\bar{x})$. The evaluations are written with CppAD types and operations and stored in the vector `fg[]`.

The cost functional $f(\bar{x})$ is stored in `fg[0]`. For each time step in the prediction horizon, we wish to minimize the cross tracking error (cte), the orientation error (ϕ_e), and the difference between the reference velocity (v_{ref}) and velocity states (v). This discrete cost functional can be written as follows where the t is the current time step.

$$\sum_{i=0}^{N} cte[t+i]^2 + \phi_e[t+i])^2 + (v_{ref} - v[t+i])^2$$

Additional terms may be added to the cost functional. For example minimizing the squared values of the actuators and the difference between two sequential actions can prevent the system from potentially returning a solution with a bang bang input which is not well handled by the vehicle. These two can be written as follows where δ is the steering input and a is the throttle input.

$$\sum_{i=0}^{N-1} \delta[t+i]^2 + a[t+i]^2$$

$$\sum_{i=1}^{N-1} (\delta[t+i] - \delta[t+i-1])^2 + (a[t+i] - a[t+i-1])^2$$

Constraints $g(\bar{x})$ are stored in the remaining elements of the vector `fg[]`. The vehicle model is implemented here by moving the equations to the LHS and using an equality constraint set to zero. These equations can be written as follows where f_{eval} and ϕ_{eval} are the y-position and orientation angle of the interpolating polynomial respectively and L_f is the distance between the front wheels and the vehicle's COM.

$$x[t+1] - (x[t] + v[t]\cos(\phi[t])\Delta t) = 0$$
$$y[t+1] - (y[t] + v[t]\sin(\phi[t])\Delta t) = 0$$
$$v[t+1] - (v[t] + a[t])\Delta t) = 0$$
$$\phi[t+1] - (\phi[t] - v[t]\delta[t])\Delta t/L_f) = 0$$
$$cte[t+1] - (f_{eval}[t] - y[t] + v[t]\sin(\phi_e[t])\Delta t) = 0$$
$$\phi_e[t+1] - (\phi[t] - \phi_{eval}[t] - v[t]\delta[t]\Delta t)/L_f) = 0$$

The RHS of the equality constraints are passed to the solver as the arguments `constraints_lowerbound[]` and `constraints_upperbound[]`, which are set to zero.

Inequality constraints are placed on the states (for all time steps in the prediction horizon) using the arguments `vars_lowerbound[]` and `vars_upperbound[]`. Only the actuators have finite bounds placed on them, the steering input (δ) is limited to +/- 25 degrees and the throttle (a) is limited to +/-1.

This constrained system is passed to the Ipopt solver which returns a solution vector that satisfies the prescribed tolerances or when the solver runs out of the allocated time.

3. Choosing horizon parameters and accounting for time delay

For plants with a time delay, it is good practice to have prediction horizons and control horizons satisfying:

$$N - M >> t_d/\Delta t$$

where N is the prediction horizon, M is the control horizon, t_d is the maximum time delay and Δt is the controller sample time. In this particular case, the time delay is 100ms which is introduced by simply waiting to apply the computed input to the system.

A design speed of 50mph equates to around 22m/s. 20 meters appears to give enough of the path to return to the center as well as describe the curvature of an upcoming turn in the road. Therefore at a minimum, the horizon needs to be at least one second long when traveling at full speed. Too long of a horizon would mean that the quadratic polynomial would not fit well and too short of a horizon does not provide enough foresight to account for turns.

The controller sample time Δt should be less than the time delay $t_d = 100$ms, but should not be too small otherwise the inequality above will not be satisfied. $\Delta t = 100$ms seems reasonable which requires $N = 10$ to get the prediction horizon length of 1s.

One method to account for the time delay is to use dead reckoning. Using the kinematic equations, we propagate the system states 100ms in the future and use those states as the starting point for the MPC solver. The solution to the MPC problem then gives the appropriate controls that can be applied 100ms in the future.