Model Predictive Control

Patrick Cleeve

Objective

The objective of this project is to code and tune an Model Predictive Controller to drive the vehicle around the simulator track with latency.

The project code is available here:
https://github.com/patrickcleeve/CarND-Term2-MPC-Project-P5

A video showing successful completion is available in the project repository.

Model Predictive Control

In this project we tune a Model Predictive Controller (MPC) to drive the vehicle around the lake track (simulator). The vehicle has a planned series of waypoints to follow in order to successfully drive around the track. Artificial latency is introduced to simulate actuator response time.

MPC allows us to optimise a controller over a series of future states, whilst only implementing the immediate state. This allows the controller to take future considerations into account when planning actions. This future control path can be visualised in the simulator as the green line.

However, optimising over future states, requires a kinematic and/or dynamic model of the vehicle's motion. Whilst more complex models can provide better control and response, due to computational complexity of the motion model, and response time required, the model complexity or time horizon is required to be limited.

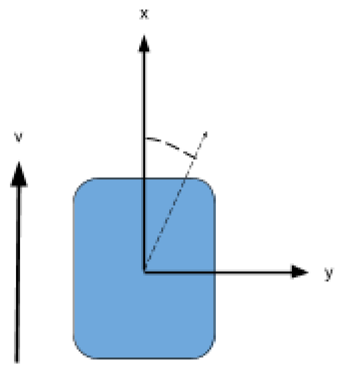
These model, controller and considerations are discussed in the following sections.

Model

Kinematic Model

We use a kinematic (bicycle) model for the vehicle's motion using four state variables. The vehicle's coordinate system is attached to the centre of the vehicle.

x_t : *vehicle* $x -$ *coordinate* (m)
y_t : *vehicle* $y -$ *coordinate* (m)
ψ_t : *vehicle orientation* (rad)
v_t : *vehicle velocity* (m/s)



Model Accuracy and Complexity

The kinematic model used is relatively simple, and not dynamic, meaning it does not represent the forces acting on the vehicle. Due to this low complexity, the model can be run in real time, and be optimised over a reasonable time horizon, compared to a higher complexity model. In addition, a dynamic model needs to be tuned for individual vehicle characteristics such as mass or wheel base. The simple models are much more transferable between different vehicles.

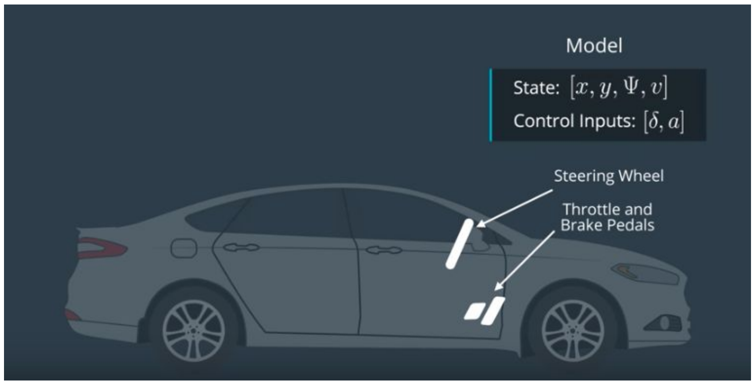
It is for these reasons we use a simplified kinematic model for the MPC. However, it may be more beneficial to use a higher complexity dynamic model in different parts of the self driving pipeline, where higher accuracy is more important than computational efficiency.

Control Inputs

We can control the vehicle's motion via two actuator variables:

δ_t : *vehicle steering angle*
a_t : *vehicle acceleration*

We simplify vehicle throttle and braking into a single acceleration control input, with negative values representing braking, and positive values for acceleration.

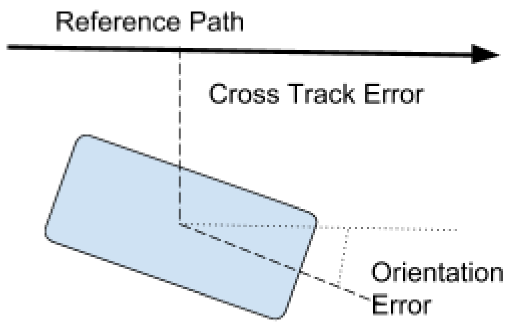


Control Inputs (Udacity Lecture)

Error Terms

We measure the vehicle's current state, in comparison to the desired state using the following error terms. These are calculated as shown, and added to the vehicle's state vector.

cte_t : *vehicle cross track error*
$e\psi_t$: *vehicle orientation error*



Kinematic Model Update Equations

We use the following kinematic equations to update the vehicle's state between subsequent timesteps.

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

L_f measures the distance between the centre of mass of the vehicle and front axle ($L_f = 2.67$).

Cost Function

In order to evaluate the performance of the controller in moving the vehicle towards the desired state, we develop a cost function. This cost function is made up of a number of individual cost functions, for different aspects of the vehicle's state and actuators. We then use a solver (ipopt) to determine the actuator values to minimise the total cost function over N timesteps, whilst satisfying variable and equation constraints (discussed below).

The individual cost functions are as follows:

Reference Cost Functions

These cost functions penalise the controller for moving away from the desired (reference) states. These include cross track error (cte) ,orientation error $(e\psi)$ and velocity (v).

$$
\begin{aligned}
Cost_{cte} &= C_1 * (cte_i - cte_{ref})^2 \\
Cost_{e\psi} &= C_2 * (e\psi_i - e\psi_{ref})^2 \\
Cost_v &= C_3 * (v_i - v_{ref})^2
\end{aligned}
$$

Actuator Cost Functions

These cost functions penalise the controller for having high values of steering angle or acceleration.

$$
\begin{aligned}
Cost_\delta &= C_4 * \delta_i^2 \\
Cost_a &= C_5 * a_i^2
\end{aligned}
$$

Change in Actuator Cost Functions

These cost functions penalise the controller for rapidly changing the values of steering angle or acceleration. These constant are influential in smoothing the turning and acceleration of the vehicle, for human comfort.

$$
\begin{aligned}
Cost_{\Delta\delta} &= C_6 * (\delta_{i+1} - \delta_i)^2 \\
Cost_{\Delta a} &= C_7 * (a_{i+1} - a_i)^2
\end{aligned}
$$

The constants for the individual cost functions are as follows:

Cost Variable	**Constant**	**Value**
Vehicle Cross Track Error (Reference)	C_1	2000
Vehicle Orientation Error (Reference)	C_2	2000
Vehicle Velocity (Reference)	C_3	1
Vehicle Steering Angle (Actuator)	C_4	5
Vehicle Acceleration (Actuator)	C_5	5
Change in Vehicle Steering Angle (Actuator)	C_6	200
Change in Vehicle Acceleration (Actuator)	C_7	10

The constant for individual cost function can be adjusted to influence the relative importance of each cost on the performance of the overall controller. As such, the solver will seek to minimise these costs (by adjusting the corresponding actuator values), over other variables. For example, as shown above vehicle cross track error and orientation error are significantly more important than vehicle velocity (2000:1). Conceptually, this can be thought of as the controller trying to stay on track much more than travel at the reference velocity (100 mph). Adjustments to the cost constants can have a significant influence on the behaviour of the vehicle/controller.

Total Cost Function

The total combined cost function is as follows:

$$Cost_{total} = \sum_{i=0}^{N}(Cost_{cte} + Cost_{e\psi} + Cost_{v}) + \sum_{i=0}^{N-1}(Cost_{\delta} + Cost_{a}) + \sum_{i=0}^{N-2}(Cost_{\Delta\delta} + Cost_{\Delta a})$$

Parameters, Variables and Constraints

Parameters

As previously discussed we can tune the a series of parameters to influence the behaviour of the vehicle and controller.

The first is the prediction time horizon. The higher we make T, the further into the future we can optimize the controller. However, we must balance this accuracy with the rapid response required by the controller. If we increase N, we will need to solve over a larger number of variables, increasing solve time, but potentially achieving a more stable controller. Whereas, by reducing dt we can increase the accuracy of each timestep prediction (less time between steps) but reduce the overall time horizon. This balance between variables is shown in the table below.

$T\ :\ Prediction\ time\ horizon = N * dt$
$N : Number\ of\ timesteps$
$dt : Timestep\ duration$

N	**dt**	**T**
5	0.1	0.5
10	0.05	0.5
10	0.1	1.0
20	0.05	1.0
25	0.05	1.25

For the project, we use N=10, dt=0.1 for time parameters as we force the solver to automatically return a result after 0.5 seconds. This parameter could also have been adjusted if we believe it would be better to wait for higher accuracy.

Initial Conditions and Variable Constraints

We initialise all variables to zero, set initial conditions and set the following variable constraints:

$-1E19 < x_i, y_i, \psi_i, v_i < 1E19$
$-25\circ < \delta_i < 25\circ$
$-1.0 < a_i < 1.0$

Equation Constraints

We constrain the model equations are follows:

$$0 = x_{t+1} - x_t + v_t * cos(\phi_t) * dt$$
$$0 = y_{t+1} - y_t + v_t * sin(\phi_t) * dt$$
$$0 = \psi_{t+1} - \psi_t - \frac{v_t}{Lf} * \delta_t * dt$$
$$0 = v_{t+1} - v_t + a_t * dt$$
$$0 = cte_{t+1} - f(x_t) - y_t + v_t * sin(e\psi_t) * dt$$
$$0 = e\psi_{t+1} - \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt$$

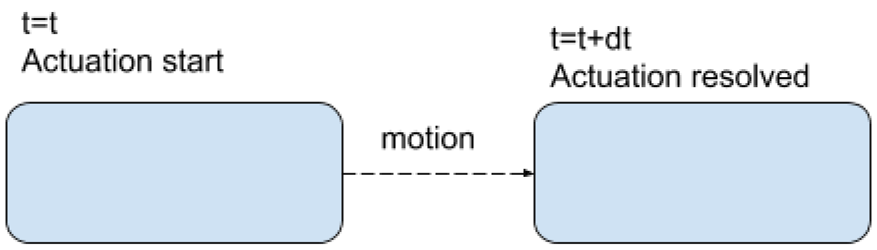
Pre-processing

Before we can solve the MPC equations, we need to be able to calculate a desired state for the controller. The simulator provides a path (waypoints) for the vehicle to follow around the track. However, first the following pre-processing steps must be accomplished:

- Read in track waypoints. These represent the desired path for the vehicle to follow.
- Convert waypoints from map coordinate system to car coordinate system, so that they are in the same coordinate system as state variables (and to simplify mathematics).
- Fit third degree polynomial to waypoints to calculate desired trajectory.
- Use polynomial coefficients to calculate cross track, and orientation error.
- Pass polynomial coefficients (along with vehicle state) to MPC::solve for use calculating desired state.

Latency

In the real world, controller latency is caused by the delay in moving actuators to the desired position. We simulate this latency by using an artificial delay of 100ms when updating the actuators in the simulator. Due to this delay, by the time our actuators update, the vehicle has travelled for a period of time. Therefore, the vehicle now requires a different set of actuator commands to respond to its new state. As a result, the vehicle is constantly trying to correct for the delayed commands, and this error increases with vehicle speed (more distance travelled for fixed latency).



There are a number of techniques to deal with the latency problem.

The simplest solution is to restrict the vehicles reference velocity to a low amount. This minimises the distance the vehicle can travel during the latency period, reducing the impact of the delay on the controller. During simulator testing, the vehicle was able to successfully drive around the lake with a reference velocity of 50 mph, without any updates to the base MPC. However, whilst this approach may successfully complete the track, it is not appropriate for use. The obvious limitations are on the vehicles velocity, and the the slow reaction time (due to delay). This will limit the turns teh vehicle is able to take, and would be a recklessly dangerous method in the real world, due to the slow reaction.

Instead, we can use model update equations to predict the vehicle's state after the latency delay, and use this predicted state to optimise the controller. This works well if the latency delay is known, and relatively small. If the delay is unpredictable, we would have to estimate an appropriate measurement. In addition, if the latency is very large, our predicted state may be very inaccurate. However, because we know our latency is constant and relatively small (0.1s), we can make relatively confident predictions for use in the controller.

In addition, because we are using the car's coordinate system for the equations of motion, we can simplify the update equations, using the following assumptions.

- As the car's coordinate system is attached to the vehicle, and oriented in the direction of motion, we can assume the current x, y and ψ state variables to be zero (0).
- We can read the current velocity, acceleration (throttle) and delta (steering angle) from the telemetry data passed from the vehicle.
- We can calculate the current cte and $e\psi$ as previously done, using the waypoint polynomial coefficients (where coeffs is the coefficients of the third-degree polynomial fitted to waypoints).

Therefore, our initial state and actuator variables at time t are:

$$\begin{aligned}
x_t &= 0 \\
y_t &= 0 \\
\psi_t &= 0 \\
v_t &= v\ (telemetry) \\
cte_t &= polyeval(coeffs,\ 0) \\
e\psi_t &= -atan(coeffs[1])
\end{aligned}$$

$$\begin{aligned}
\delta_t &= steering\ angle\ (telemetry) \\
a_t &= throttle\ (telemetry)
\end{aligned}$$

Latency State Prediction Equations

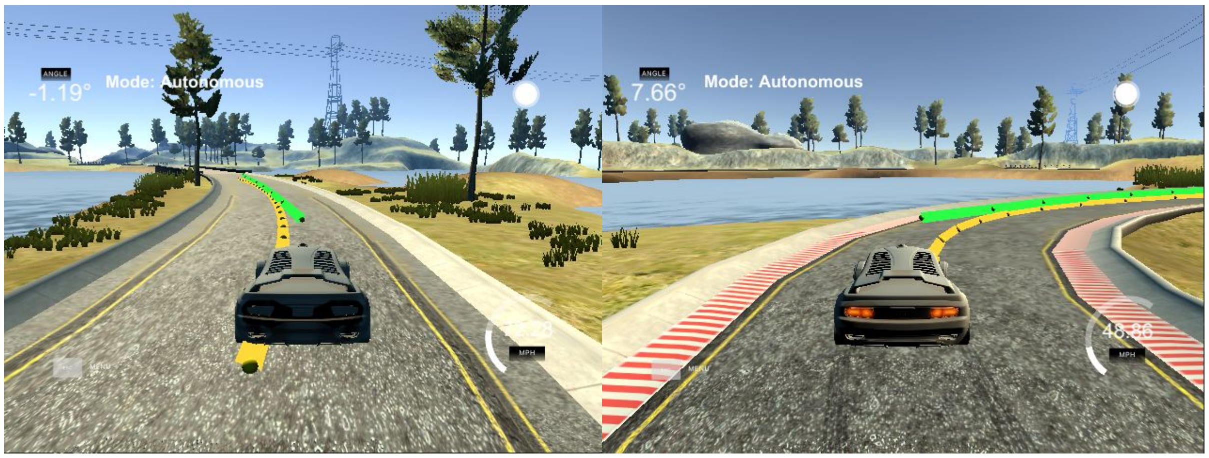
Therefore, we can simplify our update equations using the values above as ($Latency = dt$):

$$
\begin{aligned}
x_{t+dt} &= x_t + v_t * cos(\phi_t) * dt & &= v_t * dt \\
y_{t+dt} &= y_t + v_t * sin(\phi_t) * dt & &= 0 \\
\psi_{t+dt} &= \psi_t - \frac{v_t}{Lf} * \delta_t * dt & &= -\frac{v_t}{Lf} * \delta_t * dt \\
v_{t+dt} &= v_t + a_t * dt & &= v_t + a_t * dt \\
cte_{t+dt} &= f(x_t) - y_t + v_t * sin(e\psi_t) * dt & &= cte_t + v_t * sin(e\psi_t) * dt \\
e\psi_{t+dt} &= \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt & &=- e\psi_t + \frac{v_t}{L_f} * \delta_t * dt
\end{aligned}
$$

We then use the updated latency state variables as the state passed to our MPC, reducing the impact of latency on the controller.

Evaluation

We pass the update state, and waypoint polynomial coefficients to the MPC to adjust the actuator values at each timestep. For a latency of 100ms, and a reference velocity of 50 mph, the MPC is able to successfully drive the vehicle around the lake track. The vehicle performs smoothly, braking successfully around sharp corners and accelerating on straights.



Reflection

The MPC controller was able to successfully drive around the simulator track. However, a number of real world limitations were noted. Whilst a latency was added to simulate actuator motion, it was a relatively small and constant value which meant that we were able to robustly predict the future vehicle state. In the real world, this latency is likely to be more significant and variable, based on specific actuators and their relative motions and situations. THis would make it significantly more difficult to deal with, and we would likely have to slow down greatly to resolve these issues using a MPC controller.

Reference

Udacity, Project Slack (#s-t2-p-mpc)
No URL available

Udacity, Self-Driving Car Project Q&A | MPC Controller
https://www.youtube.com/watch?v=bOQuhpz3YfU&feature=youtu.be

Udacity, How to incorporate latency into the model
https://discussions.udacity.com/t/how-to-incorporate-latency-into-the-model/257391

Udacity, MPC - Car space conversion and output of Solve() intuition
https://discussions.udacity.com/t/mpc-car-space-conversion-and-output-of-solve-intuition/249469/9

Udacity, Vehicle Models
https://classroom.udacity.com/nanodegrees/nd013/parts/40f38239-66b6-46ec-ae68-03afd8a601c8/modules/f1820894-8322-4bb3-81aa-b26b3c6dcbaf

Udacity, Model Predictive Control
https://classroom.udacity.com/nanodegrees/nd013/parts/40f38239-66b6-46ec-ae68-03afd8a601c8/modules/f1820894-8322-4bb3-81aa-b26b3c6dcbaf/lessons/338b458f-7ebf-449c-9ad1-611eb933b076/concepts/00154b2e-bc08-4d00-b47e-c4209e3bbdc7

Wikipedia, Model Predictive Control
https://en.wikipedia.org/wiki/Model_predictive_control