MPC Controller

Udacity Self Driving Car project #10

This Project is the tenth task of the Udacity Self-Driving Car Nanodegree program. The main goal of the project is to implement in C++ Model Predictive Control to drive the car around the track. The program uses a simple Global Kinematic Model. Parameters were tuned in order to reach maximal speed.

The following present the mathematical modelling and the implementation.

Mathematical Modelling

Waypoint Modelling

Objective is to find a third polynomial curve to fit a set of given waypoints to support motion modelling.

Waypoints are given as a set of (x,y) coordinates in a global coordinate system, which have to be transformed via a standard rotation matrix

$$R_{psi} = \begin{pmatrix} \cos psi & \sin psi \\ -\sin psi & \cos psi \end{pmatrix}$$

With psi being the current orientation of the vehicle delivered from the simulator.

Note: the psi delivered from the simulator is the negative of the mathematically expected psi, which is incorporated in the rotation matrix.[1]

Waypoint Objective

The objective is to

$$\min(\sum(y_{vehicle}(x_{vehicle}) - (c_0 + c_1 * x_{vehicle} + c_2 * {x_{vehicle}}^2 + c_3 * {x_{vehicle}}^3))^2)$$

subject to

$$1E^{-19} \leq (c_0, c_1, c_2, c_3) \leq 1E^{19}$$

I.e. this is an unconstraint objective function.

Motion Modelling

Objective is to find a set of steering and acceleration values (actuators) to follow the given sets of waypoints ahead of the car.

There will be M given waypoints and the objective is to model N steps ahead with a time step of a given dt. Since the waypoints could be generated arbitrarily and their time difference will most likely not match dt, there is a requirement to estimate the curve of the waypoints under the assumption that the waypoints are realistic and do follow a polynomial curve, i.e. the waypoints are not jumpy and lend themselves to driving a vehicle.

There is an objective velocity of (an arbitrary) 100 mph, i.e. a velocity that is high enough to make the car speedily drive forward.

[1] See https://en.wikipedia.org/wiki/Rotation_matrix

Variables

Available variables are

d	Delta, steering value
a	Acceleration
x	X coordinate of the vehicle
y	Y coordinate of the vehicle
v	Velocity
psi	Current orientation of the vehicle
cte	Cross track error, position of the vehicle away from its idea position
epsi	Orientation error away from its ideal orientation

There are N values of each, except for the actuators (steering and acceleration), where there are N-1 values since for the last values at timestep N it would need to estimate another time step, which is not available.

Objective

The objective is to *smoothly* and *as fast as possible* drive the vehicle *as close as possible* to the given waypoints. The following objects are given in order of their importance, for example it is much more important to drive as close as possible than it is to drive smoothly. This will be represented as relative weights given to each objective.

As close as possible	Minimize cross track error and Orientation error
As fast as possible	Minimize difference of velocity to reference velocity
Smoothly	Minimize change in actuators, i.e. no hard acceleration nor hefty steering. Minimize actuators, i.e. aim to not accelerate nor steer if not required.

For all $i \ in \ [1, N]$[2]

$$\min \sum cte_i^2 * w_{cte}$$

$$\min \sum epsi_i^2 * w_{epsi}$$

$$\min \sum (v_i - ref_v)^2 \text{ with } ref_v \text{ a given constant (target velocity)}$$

For all $i \ in \ [1, N-1]$

$$\min \sum (a_i - a_{i-1})^2 * w_{a\_delta}$$

$$\min \sum (d_i - d_{i-1})^2 * w_{d\_delta}$$

Additionally, with small weights, large throttle or steering values will be penalized by

For all $i \ in \ [1, N]$[3]

$$\min \sum d_i^2 * w_d$$

[2] Since values at i=0 besides the actuators are fixed, there is no reason to include them in the target functions
[3] Since values at i=0 besides the actuators are fixed, there is no reason to include them in the target functions

$$\min \sum a_i^2 * w_a$$

Dependencies

Variables are not all independent of each other. There is a given starting point and all other variables except the two actuators have dependencies between them.

The following table displays dependencies for all i in [1;N]. *dt* is a given time interval, *Lf* is a given turn radius of the vehicle. This matches the global kinematic model presented in the Udacity lectures.

The given waypoints are represented by a polynomial and its coefficients *coeffs* and its evaluation function *polyeval* and its first derivative evaluation *polyeval'*.

x	$x_i = x_{i-1} + \cos(psi_{i-1}) * v_{i-1} * dt$
y	$y_i = y_{i-1} + \sin(psi_{i-1}) * v_{i-1} * dt$
v	$v_i = v_{i-1} + a_{i-1} * dt$
psi	$psi_i = psi_{i-1} + v_{i-1} * d_{i-1} * \frac{dt}{Lf}$
cte	$cte_i = (\text{polyeval}(x_{i-1}) - y_{i-1}) + \sin(epsi_{i-1}) * v_{i-1} * dt$
epsi	$epsi_i = (psi_{i-1} - \text{atan polyeval}'\, x_{i-1}) + v_{i-1} * d_{i-1} * \frac{dt}{Lf}$

At the starting point (time step i = 0), the position and orientation will be vehicle centric, i.e. x0, y0, and psi0 will be 0 while current velocity v0, current cross track error cte0, and current orientation error epsi0 will be given from the simulator.

Constraints

The first set of values are given and must be met.

$$0 \le x_0, y_0, psi_0 \le 0$$

$$cte \le cte_0 \le cte$$

$$v \le v_0 \le v$$

$$epsi \le epsi_0 \le epsi$$

Where cte, v, epsi are the starting values from the simulator.

Throttle (approximately acceleration) must be greater than -1 and less than 1, and steering (delta) must be with range on +/- 25 radians.

$$-1 \le a_i \le 1$$

$$-0.436332 \le d_i \le 0.436332$$

Additional constraints are with (x,y) coordinates that are required for the simulator output for display. Here the predicted (x',y') coordinates from the motional model and the optimizer values (x,y) have to match for all timesteps after the initial one (see above)[4]:

[4] It turns out that the optimizer does not strictly require the (x,y) coordinates to address the objective yet without including them here, the values that the optimizer uses wouldn't be known afterwards.

$$0 \leq x'_i - x_i \leq 0$$

$$0 \leq y'_i - y_i \leq 0$$

Implementation

Code Structure

The code has two main parts:

1. The main loop in *main.cpp* that takes telemetry values from the simulator, executes predictions and optimizations and sends values back to the optimizer
2. Prediction and Optimization classes in *MPC.cpp*.

Classes

There are three classes in the code base:

MPC	The main class is MPC, which is carried in the main loop of step 1 and holds all values and functionality to execute prediction and optimization
FG_polynomial	Class to fit a polynomial curve with IPopt
FG_motion	Class to optimize actuators with IPopt

Configurable Constants

All constants are set at the header of *MPC.h*.

The main values to update and configure are

Constant	Description	Value	Rationale
N	Number of time steps to look ahead	10	Partly set because of comments in forums, partly because 15 timesteps made the car crash
dt	Time difference for each step in seconds	0.1	The values choosen worked well.
Ref_v	Reference Velocity	100	Should be as large as possible while safely driving the car. At 150 the car regularly crashed, while at 100 the car just makes all corners
W_*	Weights for the optimizer target function	Various	Values are taken from Udacity's FAQ and worked well without any need to update

Latency

In reality, there is a small yet noticeable latency when steering or accelerating a car. In this mode, the controller artificially delays messages back to the simulator by 0.1 seconds to model this latency.

To address this, the controller takes the initial state from the simulator and predicts where the car might be in 0.1 seconds via its kinematic model before modelling and predicting actuators via the optimizer.

Differences to proposed solution

There are multiple differences to the proposed solution by Udacity:

1. No Eigen library: the only reason to include the Eigen library is to perform a polynomial curve fitting yet this can be achieved with IPopt alone thus the Eigen library was dropped. This resulted in a 20x smaller executable file without loss of functionality or performance.
2. Only required variables are included in the optimizer variables. Udacity's proposal included all state and actuators variables in the optimizer variables yet only the actuators are strictly required while the (x,y) coordinates were included to have those part of the solution and thus part of the simulator output (the *green* line).
3. Code that remains constant is moved from the main loop to initializing code, mainly into the constructor of the MPC class to avoid repeated execution.

Issues and Observations

Performance

The main issue along the project was performance of the local setup:

Windows 10 with an Ubuntu Virtual Machine.

In all previous projects, the controller ran on the virtual machine while the simulator ran natively on Windows 10, and it always worked fine. In this project, potentially due to the memory requirements of the optimizer, the setup executed without any issues, yet the car would not keep on the track. This result was weeks of trying to debug an ultimately working code base.

In the end the controller was running on a c3.large AWS EC2 instance with enough memory and computer power to run it, and the simulator ran locally on Windows 10.