CarND-Controls-MPC

Self-Driving Car Engineer Nanodegree Program

This README can also be found in this repo as a PDF (in case the equations don't show properly).

Dependencies

- cmake >= 3.5
- All OSes: click here for installation instructions
- make >= 4.1
  - Linux: make is installed by default on most Linux distros
  - Mac: install Xcode command line tools to get make
  - Windows: Click here for installation instructions
- gcc/g++ >= 5.4
  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - [install Xcode command line tools]((https://developer.apple.com/xcode/features/)
  - Windows: recommend using MinGW
- uWebSockets == 0.14, but the master branch will probably work just fine
  - Follow the instructions in the uWebSockets README to get setup for your platform. You can download the zip of the appropriate version from the releases page. Here's a link to the v0.14 zip.
  - If you have MacOS and have Homebrew installed you can just run the ./install-mac.sh script to install this.
- Ipopt
  - Mac: `brew install ipopt --with-openblas`
  - Linux
    - You will need a version of Ipopt 3.12.1 or higher. The version available through `apt-get` is 3.11.x. If you can get that version to work great but if not there's a script `install_ipopt.sh` that will install Ipopt. You just need to download the source from here.
    - Then call `install_ipopt.sh` with the source directory as the first argument, ex: `bash install_ipopt.sh Ipopt-3.12.1`.
  - Windows: TODO. If you can use the Linux subsystem and follow the Linux instructions.
- CppAD
  - Mac: `brew install cppad`
  - Linux `sudo apt-get install cppad` or equivalent.
  - Windows: TODO. If you can use the Linux subsystem and follow the Linux instructions.
- Eigen. This is already part of the repo so you shouldn't have to worry about it.
- Simulator. You can download these from the releases tab.

Basic Build Instructions

1. Clone this repo.
2. Make a build directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`
4. Run it: `./mpc`.

Building using Docker

1. docker build -t mpcp .
2. docker run -p 127.0.0.1:4567:4567 mpcp ./mpc

Discussion

Pre-processing of Waypoints

All the waypoints are transformed from the global reference frame to the vehicle reference frame. The following transform is used for each of the waypoints:

$$\begin{aligned} x_{trans} &= [x_{waypoint} - x_p] * cos(\psi) + [y_{waypoint} - y_p] * sin(\psi) \\ y_{trans} &= -[x_{waypoint} - x_p] * sin(\psi) + [y_{waypoint} - y_p] * cos(\psi) \end{aligned} \tag{1}$$

Where ψ is the angle of the vehicle with respect to the x-axis and x_p and y_p are the location of the vehicle in the global reference frame. $x_{waypoint}$ and $y_{waypoint}$ are the x, y coordinates of a given waypoint in the global reference frame.

Polynomial Fitting

The transformed waypoints are fit to a 3rd order polynomial $f(x_t)$.

$$f(x_t) = c_0 + c_1 x_t + c_2 x_t^2 + c_3 x_t^3 \tag{2}$$

With c_i $i \in \{0, 1, 2, 3\}$ being the coefficients.

Given that the waypoints are now in the reference frame of the vehicle, the current cross track error cte_0 is simply the value of the polynomial at point $x_0 = 0.0$, so we have $cte_0 = f(x_0) = f(0.0) = c_0$.

The current orientation error $e\psi$ is determined by the arc-tangent of the derivative of the polynomial.

$$e\psi = arctan(f'(x_t)) = arctan(c_1 + 2c_2 x_t + 3c_3 x_t^2) \tag{3}$$

The result $e\psi_0$ is also at point $x_0 = 0.0$ and hence $e\psi_0 = arctan f'(x_0) = arctan f'(0.0) = arctan(c_1)$.

The Model

The **state** is a 6 element vector with the following elements:

- x position of vehicle
- y position of vehicle
- ψ - the orientation of the vehicle
- v - speed of the vehicle

- cte - cross track error
- $e\psi$ - orientation error

The **actuators** are contained in a 2 element vector:

- δ - steering angle
- a - acceleration

The following **update equations** were used:

$$\begin{aligned} x_{t+1} &= x_t + v_t * cos(\psi_t) * dt \\ y_{t+1} &= y_t + v_t * sin(\psi_t) * dt \\ \psi_{t+1} &= \psi_t + \frac{v_t}{L_f} * \delta * dt \\ v_{t+1} &= v_t + a_t * dt \\ cte_{t+1} &= f(x_t) - y_t + v_t * sin(e\psi_t) * dt \\ e\psi_{t+1} &= \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt \end{aligned} \tag{4}$$

With the current state being $[x_0, y_0, \psi_0, v_0, cte_0, e\psi_0] = [0.0, 0.0, 0.0, v, f(0.0), arctan(f'(0.0))]$.

Timestep Length, Frequency and MPC Latency

Given that MPC has a latency of 100ms, I decided to choose a dt slightly higher than that, namely $dt = 0.15$. I then experimented with different values for N. Here the best suited value turned out to be $N = 11$. The prediction horizon T then becomes $T = (N - 1) * dt = 10 * 0.15 = 1.5$. With higher values of N, the vehicle would drive erratically and quickly drive off the track. Lower values, would make the prediction very limited.

Demo Video

A demo video of this project can be found on Youtube here.