SDC-Term 2-MPC-Project

Self-Driving Car Engineer Nanodegree Program

The purpose of this project is to implement a model predictive controller to drive a vehicle along a desired path (reference trajectory). It is tested in a simulator provided by Udacity. The simulator outputs x and y positions, speed, and orientation of the vehicle along with the reference trajectory.
In the simulator, the reference trajectory is show as a **YELLOW** line and the predicted path is in **GREEN**.

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
  - Run either `install-mac.sh` or `install-ubuntu.sh`.
  - If you install from source, checkout to commit `e94b6e1`, i.e. `git clone https://github.com/uWebSockets/uWebSockets`
    - `cd uWebSockets`
    - `git checkout e94b6e1`
    -

    Some function signatures have changed in v0.14.x. See this PR for more details.

- Fortran Compiler
    - Mac: `brew install gcc` (might not be required)
    - Linux: `sudo apt-get install gfortran`. Additionall you have also have to install gcc and g++, `sudo apt-get install gcc g++`. Look in this Dockerfile for more info.
- Ipopt
    - Mac: `brew install ipopt`
    - 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 the Ipopt releases page or the Github releases page.
        - 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.
- Not a dependency but read the DATA.md for a description of the data sent back from the simulator.

Basic Build Instructions

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

Tips

1. It's recommended to test the MPC on basic examples to see if your implementation behaves as desired. One possible example is the vehicle starting offset of a straight line (reference). If the MPC implementation is correct, after some number of timesteps (not too many) it should find and track the reference line.
2. The `lake_track_waypoints.csv` file has the waypoints of the lake track. You could use this to fit polynomials and points and see of how well your model tracks curve. NOTE: This file might be not completely in sync with the simulator so your solution should NOT depend on it.
3. For visualization this C++ matplotlib wrapper could be helpful.

Project Instructions and Rubric

Note: regardless of the changes you make, your project must be buildable using cmake and make!
More information is only accessible by people who are already enrolled in Term 2 of CarND. If you are enrolled, see the project page for instructions and the project rubric.

Model

The model used is a kinematic bicycle model. The model state includes:

- position `x`
- position `y`
- orientation `psi`
- velocity `v`
- cross-track error `cte`
- orientation error `epsi`

The control inputs are:

- steering angle `delta`
- acceleration `a`

The moving model of the vehicle is showing below:

```
// values at timestep [t+1] based on values at timestep [t] after dt seconds
// Lf is the distance between the front of the vehicle and the center of gravity

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] + 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(epsi[t]) * dt;
epsi[t+1] = psi[t] - psi_des + v[t]/Lf * delta[t] * dt;
```

Timestep Length and Elapsed Duration (`N` & `dt`)

The prediction horizon `T` is the product of the timestep length `N` and elapsed duration `dt`. Timestep length refers to the number of timesteps in the horizon and elapsed duration is how much time elapses between each actuation.
The prediction horizon I settled on was one second, with `N = 10` and `dt = .1`.

I tried different combinations of `N` and `dt`, including (`N=20, dt=0.05`), (`N=15, dt=0.05`), (`N=10, dt=0.05`), (`N=20, dt=0.1`) and so on. With higher `N` value, if the vehicle "overshot" the reference trajectory, it would begin to oscillate wildly and drive off the track. With lower value of `N`, the vehicle may drive straight off the track.
Among all the pairs of parameters, (`N = 10, dt = .1`) performs the best result.

Polynomial Fitting and MPC Preprocessing

The point are first transformed into the vehicle's coordinate system, making the first point the origin. This is done by subtracting each point from the current position of the vehicle.
Next, the orientation is also transformed to 0 so that the heading is straight forward. Each point is rotated by psi degrees.
After that the vector of points is converted to an Eigen vector so that it is ready to be an argument in the polyfit function where the points are fitted to a 3rd order polynomial. That polynomial is then evaluated using the polyeval function to calculate the cross-track error.

Model Predictive Control with Latency

A delay of 100 ms need to be taken care of after the MPC works. When this latency was first introduced, the model oscillated about the reference trajectory and, at high speeds, drove off the track.
In order to deal with the latency, I set the initial states to be the states after 100 ms. This allows the vehicle to "look ahead" and correct for where it will be in the future instead of where it is currently positioned.

Video

One lap of the car’s performance in the track is shown in the video attached with this folder named CarND-Term 2-MPC.