CarND-Controls-MPC

Self-Driving Car Engineer Nanodegree Program

Discussion

The Model Predictive Control let the car drive around the track with latency of 100ms and allows following the trajectory along a path. The MPC controller approximates the trajectory with a polynomial. The waypoints are transformed into the local vehicle coordinate system and for the polynomial fitting an approximation with a 3rd order polynomial is used.

The model state is applied with a vector of the following elements:

- x: position (x-coordinate)
- y: position (y-coordinate)
- psi: car orientation angle
- epsilon: car orientation error angle
- v: velocity
- cte: cross track error

The actuators are applied with a vector of the elements:

- delta: car steering angle
- a: car acceleartion

The Model Predictive Control handles a 100 millisecond latency which simulates latency between sensors and processing. The timestep length and frequency were chosen empirically, with dt = 0.15 and N = 11. So, dt was chosen slightly higher than the latency with a corresponding number of timesteps in the horizon.

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.
- 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.

Editor Settings

We've purposefully kept editor configuration files out of this repo in order to keep it as simple and environment agnostic as possible. However, we recommend using the following settings:

- indent using spaces
- set tab width to 2 spaces (keeps the matrices in source code aligned)

Code Style

Please (do your best to) stick to Google's C++ style guide.

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.

Hints!

- You don't have to follow this directory structure, but if you do, your work will span all of the .cpp files here. Keep an eye out for TODOs.

Call for IDE Profiles Pull Requests

Help your fellow students!

We decided to create Makefiles with cmake to keep this project as platform agnostic as possible. Similarly, we omitted IDE profiles in order to we ensure that students don't feel pressured to use one IDE or another.

However! I'd love to help people get up and running with their IDEs of choice. If you've created a profile for an IDE that you think other students would appreciate, we'd love to have you add the requisite profile files and instructions to ide_profiles/. For example if you wanted to add a VS Code profile, you'd add:

- /ide_profiles/vscode/.vscode
- /ide_profiles/vscode/README.md

The README should explain what the profile does, how to take advantage of it, and how to install it.

Frankly, I've never been involved in a project with multiple IDE profiles before. I believe the best way to handle this would be to keep them out of the repo root to avoid clutter. My expectation is that most profiles will include instructions to copy files to a new location to get picked up by the IDE, but that's just a guess.

One last note here: regardless of the IDE used, every submitted project must still be compilable with cmake and make./