CarND-Controls-MPC

Self-Driving Car Engineer Nanodegree Program

Overview & Implementation:

a) Model

- The **prediction control model** was used to generate steering angle and the throttle control signals
- The **state** vector of the car has six components as was discussed in class
  - x-position of the car, x
  - y-position of the car, y
  - orientation of the car, ψ
  - velocity magnitude , v
  - cross track error, cte
  - orientation error, $e\psi$
- Prediction & trajectory:
  - Prediction length is tuned with two independent parameters:
    - number of steps, of timestep length, N
    - sample rate, or elapsed duration, dt
  - predicted horizon is $T = N \cdot dt$
  - the reference trajectory is provided by a map in map coordinates
    - need to transform from map coordinates to car coordinates using the transformation:

      $$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \begin{pmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{pmatrix} \begin{pmatrix} x_m - x \\ y_m - y \end{pmatrix}$$

      where
      - (x_m, y_m) is the position of a reference trajectory point in map coordinates
      - (x, y) is the car location in map coordinates
      - (x_c, y_c) is the position of a reference trajectory point in car coordinates
    - the set of points in the reference trajectory is modeled as a third order polynomial $f(x)$
- Control signals & interface to :
  - steering angle, δ_k, constrained to lie between $\pm 25/\pi$ radians,
    - the interface range is ± 1
    - furthermore, the model and simulator angle conventions are different resulting a minus sign when reading and writing to the simulator
  - acceleration a_k, constrained to lie between ± 1 m/s2
- Model :
  - given current state $(x_k, y_k, \psi_k, v_k, cte_k, e\psi_k)^T$ & control $(\delta_k, a_k)^k$, the next state is

$$
\begin{aligned}
x_{k+1} &= x_k + v_k \cos(\psi_k) dt \\
y_{k+1} &= y_k + v_k \sin(\psi_k) dt \\
\psi_{k+1} &= \psi_k + \frac{v_k}{L_f} \delta_k dt \\
v_{k+1} &= v_k + a_k dt \\
cte_{k+1} &= f(x_k) - y_k + v_k \sin(e\psi_k) dt \\
e\psi_{k+1} &= \psi_k - \psi_{ref,k} + \frac{v_k}{L_f} \delta_k dt
\end{aligned}
$$

where L_f is the length of the car from front to center of gravity

- Cost:
  - the cost associated between a reference and predicted trajectory $k \in \{1, 2, \cdots, N\}$, is given by

    $$J = \sum_{k=1}^{N} w_{cte}(cte_k - cte_{ref,k})^2 + w_{e\psi}(e\psi_k - e\psi_{ref,k})^2 + w_v(v_k - v_{ref})^2$$

    where
    - the reference cte error is evaluated from reference trajectory polynomial $f(x_k)$ (which will be discussed later)

      $$cte_{ref,k} = f(x_k)$$

    - the reference orientation error is evaluated as

      $$e\psi_{ref,k} = \arctan(f'(x_k))$$

    - the reference velocity v_{ref} is an input to the model
    - the weights $w_{cte}, w_{e\psi}, w_v$ are hyper parameters that assign different weight to different terms
  - additional cost factors include terms that discourage
    - large control signals

      $$\sum_{k=1}^{N-1} w_\delta \delta_k^2 + w_a a_k^2$$

    - and sudden changes in control between consecutive samples

      $$\sum_{k=1}^{N-2} w_{d\delta}(\delta_{k+1} - \delta_k)^2 + w_{da}(a_{k+1} - a_k)^2$$

      with hyper parameters $w_\delta, w_a, w_{d\delta}, w_{da}$

- List of hyper-parameters & their final values:
  - polynomial order to fit reference trajectory (chosen order 3 as used in class)
  - number of steps, of timestep length, $N = 10$
  - sample rate, or elapsed duration, $dt = 0.08$
  - the cost function weights

    $$(w_{cte}, w_{e\psi}, w_v, w_\delta, w_a, w_{d\delta}, w_{da}) = (1, 20, 0.05, 0, 0, 1000, 10)$$

  - it was important to set $w_{e\psi} \gg w_{cte}$, otherwise the car kept overshooting

- a small $w_v = 0.05$ was sufficient to keep velocity close to reference
- i did not observe other weights making much of a difference
- with these settings I could run the car with $v_{ref} = 90$

b) Time-step Length and Elapsed Duration

- (N, dt) were defined in previous section
- The car predicts up to $T = N \cdot dt$ seconds into the future
  - as the car velocity increases, time interval T will have a longer distance trajectory
  - at high speeds, I found $T = 0.8$ s to be a reasonable parameter
- For a fixed $T = N \cdot dt$
  - reducing dt increases resolution
  - but increases N, which is proportional to complexity per update
  - to limit complexity I chose $N = 10$ which results in $dt = 0.08$ seconds

c) MPC Preprocessing

- The reference trajectory (provided by a map)
  - the reference trajectory, which is in map coordinates is transformed to car coordinates using the transformation:

    $$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \begin{pmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{pmatrix} \begin{pmatrix} x_m - x \\ y_m - y \end{pmatrix}$$

    where
    - (x_m, y_m) is the position of a reference trajectory point in map coordinates
    - (x, y) is the car location in map coordinates
    - (x_c, y_c) is the position of a reference trajectory point in car coordinates
  - the resulting N points are modeled by as a third order polynomial $f(x)$, using the provided function *polyfit()*
  - when all the trajectory points are in car coordinates then the first three states also can be represented in car coordinates which simplify to zero

    $$(x_k, y_k, \psi_k) = (0, 0, 0)$$

    - in words, using car coordinates, the car is at the origin $(0, 0)$ with zero orientation (looking ahead)

d) Latency

- To incorporate the impact of $l = 0.1$ second latency in actuation, the state estimate is transformed so that each element of the state is adjusted for this latency
  - this transformation is performed after the state is in car coordinate system, i.e. with $x_k = 0$, $y_k = 0$, $\psi_k = 0$
  - the impact on the state can be determined using the kinematic model update rules discussed above in (a), with $l = dt$, and using steering angle δ_k & acceleration a_k
  - The resulting kinematic equations that generate latency compensated state components are

$$
\begin{aligned}
x_{l,k} &= v_k l \cos(\delta_k) \\
y_{l,k} &= v_k l \sin(\delta_k) \\
\psi_{l,k} &= \delta_k \left(1 + \frac{v_k l}{L_f}\right) \\
v_{l,k} &= v_k + a_k l \\
cte_{l,k} &= f(0) + v_k l \sin(e\psi_k) \\
e\psi_{l,k} &= -\arctan(f'(0)) + \frac{v_k l}{L_f}\delta_k
\end{aligned}
$$

Generic Read-me Content:

Dependencies

- cmake >= 3.5
- All OSes: click here for installation instructions
- make >= 4.1(mac, linux), 3.81(Windows)
  - 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.
- **Ipopt and CppAD:** Please refer to this document for installation 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.)
4. Tips for setting up your environment are available here
5. **VM Latency:** Some students have reported differences in behavior using VM's ostensibly a result of latency. Please let us know if issues arise as a result of a VM environment.

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./

How to write a README

A well written README file can enhance your project and portfolio. Develop your abilities to create professional README files by completing this free course.