UDACITY

Back to Self-Driving Car Engineer

Model Predictive Control (MPC)

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Enthusiastic Learner,

This work is an outstanding one. I enjoyed reviewing this work 💯 %. It is straight, concise, well documented and a well articulated material. A lot of knowledge in this area has been show cased in this project. Keep it up 😄. This is important since these algorithms are for critical systems that involve lives, etc. 👏 Bravo, best wishes in your career ahead, stay excellent and good luck.

Further Improvement Suggestions

- To expand ones knowledge in this area, it might be good in ones extra time to look at the following materials:
  - MPC basics with code examples https://yalmip.github.io/example/standardmpc/
  - Further understand MPC from this resource.
  - See http://www.cc.ntut.edu.tw/~jcjeng/Model%20Predictive%20Control.pdf
  - A Gentle Introduction to Model Predictive Control (MPC) Formulations based on Discrete Linear State Space Models
  - Good tutorial on MPC here too https://www.ee.columbia.edu/~dpwe/papers/Rawl00-mpc.pdf and https://minds.wisconsin.edu/handle/1793/10886
  - Field work on MPC in Multivariable, Model-Predictive Advanced Process Controller
  - Linear Model Predictive Control, Stability and Robustness
  - Linear design using MPC
  - Postface to “Model Predictive Control: Theory and Design”
  - A Tutorial on Model Predictive Control for Spacecraft Rendezvous

- Explicit Model Predictive Control
- A gentle introduction to MPC based on discrete linear space
- See some principles of optimal control using MPC from this material https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-323-principles-of-optimal-control-spring-2008/lecture-notes/lec16.pdf
- Some lectures notes on MPC here http://cse.lab.imtlucca.it/~bemporad/teaching/ac/pdf/AC2-10-MPC.pdf and http://cepac.cheme.cmu.edu/pasilectures/lee/LecturenoteonMPC-JHL.pdf
- MPC concepts https://web.stanford.edu/class/archive/ee/ee392m/ee392m.1056/Lecture14_MPC.pdf
- This is an application of MPC A tutorial review of economic model predictive control methods
- MPC elaborately broken down with explanation of formulae http://www.control.lth.se/media/Education/DoctorateProgram/2016/Control%20System%20Synthesis/MPC.pdf
- ISSN 0280-5316 ISRN LUTFD2/TFRT--5847--SE Model Predictive Control for Stock Postfolio Selection

Tools

- MatLab toolbox guide
- If interested in LabView, here is a good material

Compilation

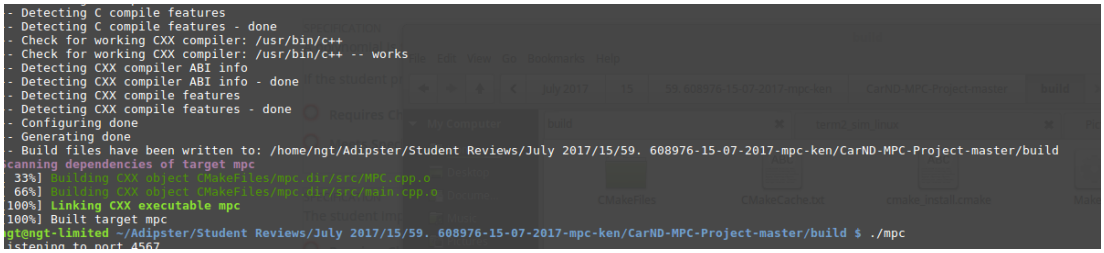
Code must compile without errors with `cmake` and `make`.

Given that we've made CMakeLists.txt as general as possible, it's recommend that you do not change it unless you can guarantee that your changes will still compile on any platform.

Impressive, keep it up!  The code compiled without errors with cmake and make when the command `cmake .. && make` is run from the build folder and with `CMakeLists.txt` submitted with the project.

Further Improvement Suggestion

- Some C++ debugging tips to consider as a developer, see
  - setting CMAKE_BUILD_TYPE to release
  - setting CMAKE_BUILD_TYPE to debug
  - In case of any failure or difficulty, most commonly Understanding why CMAKE_BUILD_TYPE cannot be set
- When cmake includes symbol tables in the generated file, GDB now can come in to assist in debugging critical faults like the famous segmentation fault, why some variables are behaving weird, etc, see
  - How to Debug Using GDB
  - GNU GDB Debugger Command Cheat Sheet
  - Debugging with GDB By Alexandra Hoffer



Implementation

Student describes their model in detail. This includes the state, actuators and update equations.

Excellent description of the model. The writeup is well elaborate and covers all section required. Great job and well done! 🎉

Further improvement suggestion

- MPC algorithm
  Screenshot from 2017-07-15 08-53-43.png

Student discusses the reasoning behind the chosen *N* (timestep length) and *dt* (elapsed duration between timesteps) values. Additionally the student details the previous values tried.

Excellently explained 👏. Student has provided a discussion of the reasoning behind the chosen N and dt values here.

Further improvement suggestions

- Choose Sample Time and Horizons and Adaptive Time Horizon Optimization in Model Predictive Control
- Selecting Building Predictive Control Based on Model Uncertainty
- Time Series Prediction Stochastic Model Predictive Control

```
2
3 2) Student discusses the reasoning behind the chosen N (timestep length) and dt (elapsed duration between timesteps) values. Additionally the student details the previous values tried.
4
5 If dt (elapsed duration between timesteps) decreases  the  predictions accuracy increases but performance may be affected. Timestep length (N) determines the number of timesteps that model predicts ahead , also N*dt is the time range
  for  solver to adjust  the optimal trajectory  I use a starting time horizont  of 10 seconds obtained through trial and error, this is N=10 and dt=0.1. Other values of dt =0.01 with N=10 was proved but the tracking error increases fast
  and vehicle shows an oscillation around the reference until he leaves the road , on the other hand if dt=1 with N=10 the error slowly increases  until he leaves the road with a damped behavior. with N=15 and dt=0.1 the velocity
  increases but the cross track error increases in curves , the final values were selected as N=7 and dt=0.1 in this case the error decreases and the average speed was 49.4  mph , also the settling time decreases fast.
6
```

```
[51]   4  #include "Eigen-3.3/Eigen/Core"
49]    5
]      6  using CppAD::AD;
       7
) [5   8  // TODO: Set the timestep length and duration
       9  size_t N = 7;
      10  double dt = 0.1;
28]   11  // Reference cross-track error
      12  double ref_v = 60;
```

A polynomial is fitted to waypoints.

If the student preprocesses waypoints, the vehicle state, and/or actuators prior to the MPC procedure it is described.

Impressive narrative on the work covered in this section. The student has adequately discussed this section.

Further improvement suggestion

- Model Predictive Control for Trajectory Following with Actuator Degradation
- Model predictive controller with life extending control

```
3) A polynomial is fitted to waypoints.

Optimal trajectory is related with a third-degree polynomial fit to the given waypoints reference. The waypoints was converted to vehicle reference frame as preprocess step  this is  x - y coordinates and orientation angle are maped
  respect frame of vehicle in main.cpp lines 108 to 122.
```

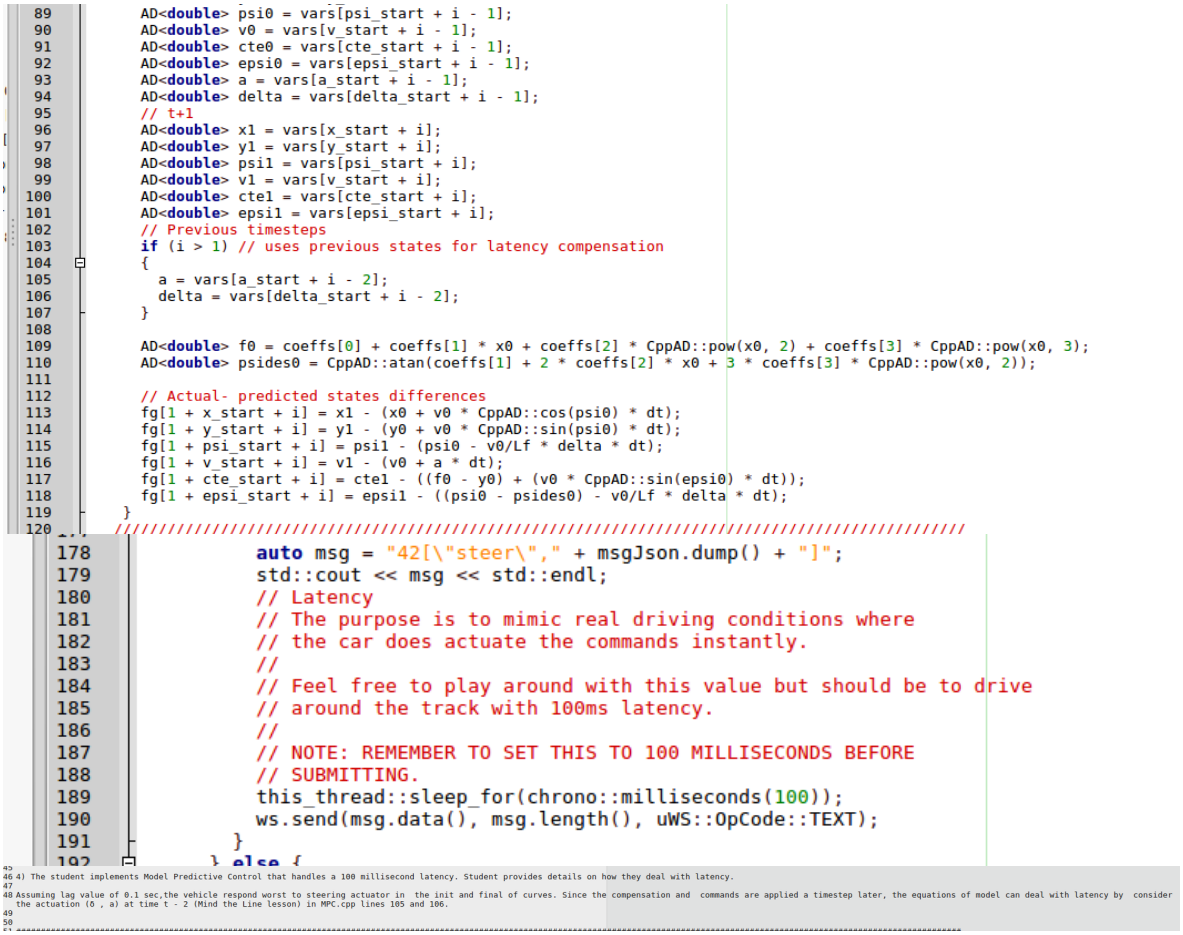
The student implements Model Predictive Control that handles a 100 millisecond latency. Student provides details on how they deal with latency.

Nicely discussed in this section, 100 millisecond latency is considered and well handled `this_thread::sleep_for(chrono::milliseconds(100));`. A detailed discussion has been given in the section. Excellently done here.

Suggestion and comments

- Fast Model Predictive Control Using Online Optimization and https://stanford.edu/~boyd/papers/pdf/fast_mpc.pdf
- MPC application to a pilot and time factor criticality https://www.ethz.ch/content/dam/ethz/special-interest/itet/institute-eeh/power-systems-dam/documents/SAMA/2016/Pan-Tianshu-MA-2016.pdf
- Predicting Time-Delays under Real-Time Scheduling for Linear Model Predictive Control
- Model Predictive Control for Transparent Teleoperation Under Communication Time Delay

```
84    ////////////////////////////////////////////////////STATES/////////////////////////////////////////////////
85    for (unsigned int i = 1; i < N; i++) {
86      // t
87      AD<double> x0 = vars[x_start + i - 1];
88      AD<double> y0 = vars[y_start + i - 1];
```



```
        AD<double> psi0 = vars[psi_start + i - 1];
        AD<double> v0 = vars[v_start + i - 1];
        AD<double> cte0 = vars[cte_start + i - 1];
        AD<double> epsi0 = vars[epsi_start + i - 1];
        AD<double> a = vars[a_start + i - 1];
        AD<double> delta = vars[delta_start + i - 1];
        // t+1
        AD<double> x1 = vars[x_start + i];
        AD<double> y1 = vars[y_start + i];
        AD<double> psi1 = vars[psi_start + i];
        AD<double> v1 = vars[v_start + i];
        AD<double> cte1 = vars[cte_start + i];
        AD<double> epsi1 = vars[epsi_start + i];
        // Previous timesteps
        if (i > 1) // uses previous states for latency compensation
        {
          a = vars[a_start + i - 2];
          delta = vars[delta_start + i - 2];
        }

        AD<double> f0 = coeffs[0] + coeffs[1] * x0 + coeffs[2] * CppAD::pow(x0, 2) + coeffs[3] * CppAD::pow(x0, 3);
        AD<double> psides0 = CppAD::atan(coeffs[1] + 2 * coeffs[2] * x0 + 3 * coeffs[3] * CppAD::pow(x0, 2));

        // Actual- predicted states differences
        fg[1 + x_start + i] = x1 - (x0 + v0 * CppAD::cos(psi0) * dt);
        fg[1 + y_start + i] = y1 - (y0 + v0 * CppAD::sin(psi0) * dt);
        fg[1 + psi_start + i] = psi1 - (psi0 - v0/Lf * delta * dt);
        fg[1 + v_start + i] = v1 - (v0 + a * dt);
        fg[1 + cte_start + i] = cte1 - ((f0 - y0) + (v0 * CppAD::sin(epsi0) * dt));
        fg[1 + epsi_start + i] = epsi1 - ((psi0 - psides0) - v0/Lf * delta * dt);
    }
    ////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
          auto msg = "42[\"steer\"," + msgJson.dump() + "]";
          std::cout << msg << std::endl;
          // Latency
          // The purpose is to mimic real driving conditions where
          // the car does actuate the commands instantly.
          //
          // Feel free to play around with this value but should be to drive
          // around the track with 100ms latency.
          //
          // NOTE: REMEMBER TO SET THIS TO 100 MILLISECONDS BEFORE
          // SUBMITTING.
          this_thread::sleep_for(chrono::milliseconds(100));
          ws.send(msg.data(), msg.length(), uWS::OpCode::TEXT);
        }
      } else {
```

```
4) The student implements Model Predictive Control that handles a 100 millisecond latency. Student provides details on how they deal with latency.

Assuming lag value of 0.1 sec,the vehicle respond worst to steering actuator in  the init and final of curves. Since the compensation and  commands are applied a timestep later, the equations of model can deal with latency by  consider
  the actuation (δ , a) at time t - 2 (Mind the Line lesson) in MPC.cpp lines 105 and 106.
```

Simulation

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

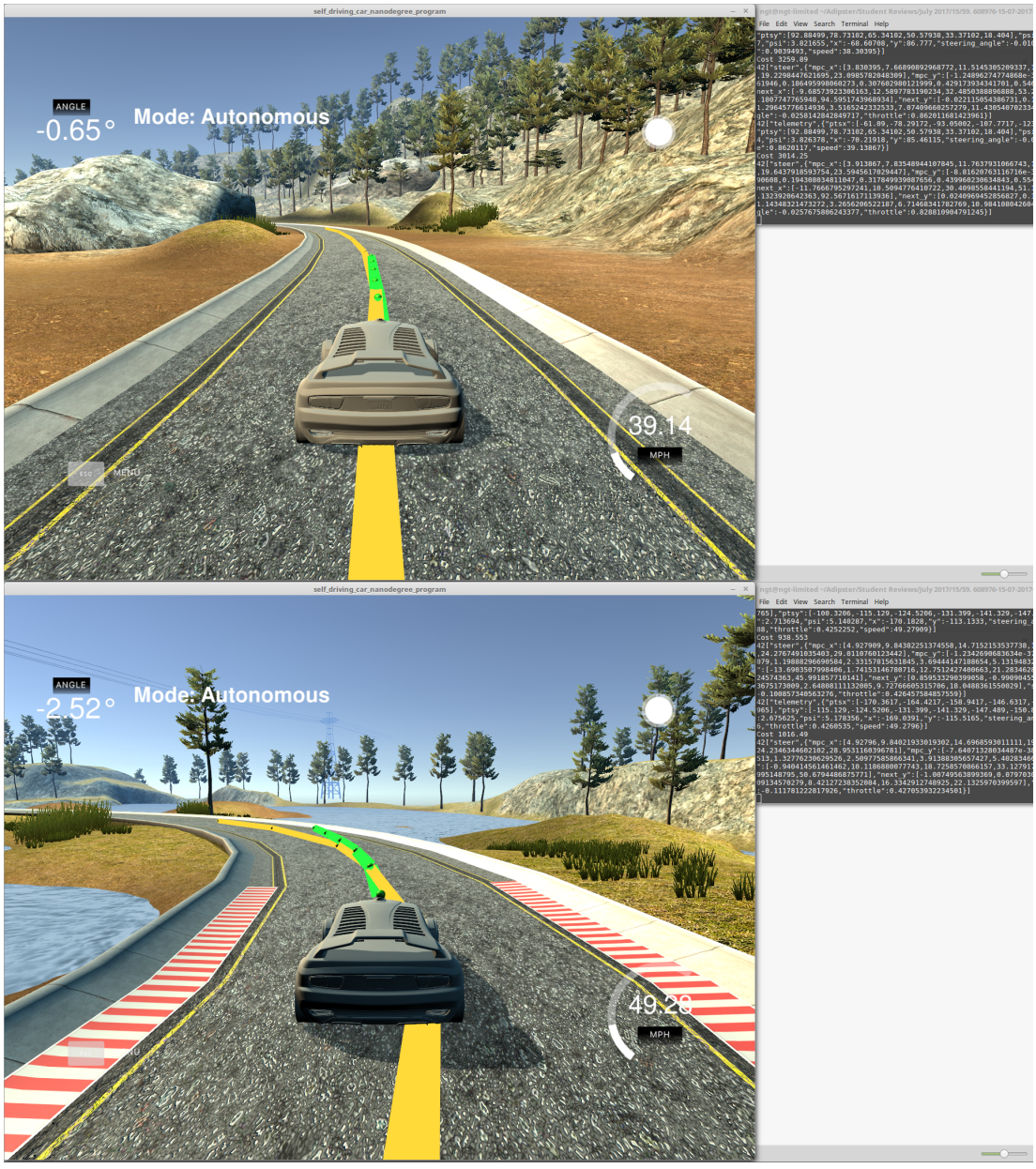
The car can't go over the curb, but, driving on the lines before the curb is ok.

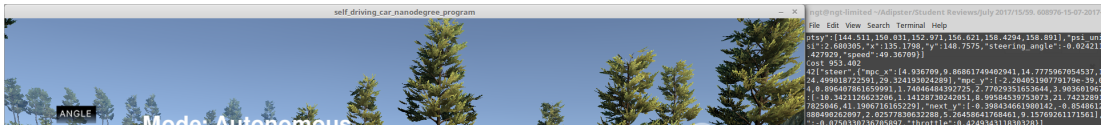
Outstanding code developed for this section. This was a remarkable output for the simulation. The simulation worked like charm. The student is going to be an outstanding self-driving car engineer. Most certain of this 😄

Further Improvement Suggestions

- This work is not too bad in the light of this rubric. There is always room for improvement. Here are some suggested materials to further improve the code and performance or behavior required for the simulation for C++ programs that are related to this area. Follow them in fixing this issue if need be
  - C++ Optimization Strategies and Techniques
  - A discussion on C++ Optimization Techniques
  - A full document with details on Tips for Optimizing C/C++ Code
  - 10 Tips for C and C++ Performance Improvement
  - Optimizing C++ A book about improving program performance
  - Optimizing C++/Writing efficient code/Performance improving features

- Another C++ Code Optimization: 10 Tips And Tricks
- Optimizing C and C++ Code
- Programming Optimization
- Programming Optimization
- Efficient C++ Performance Programming Techniques





DOWNLOAD PROJECT

RETURN TO PATH