Udacity MPC– Project 3

Overview

This project implements a Model Predictive Controller (MPC) Algorithm in C++ and tested on Simulator provided by Udacity. The communication between the simulator and the application is done using WebSocket. This project utilizes the starter code given by Udacity. Following are the outputs available as outcome of this project:

1. Video files : Contains video recorded output for various speeds. The application was tested at 40mph and 70mph separately by tunning the different parameters

2. Cmake-output.log- Contains output of Cmake

3. make-output.log - contains output of make

Environment:

This project was done in Ubuntu 18 LTS and following are the components

- cmake >= 3.5
- make >= 4.1
- gcc/g++ >= 5.4
- Ipopt and CPPAD

Ouput:

Below are the output of this project:

cmake output:

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/rameshbaboov/udacity/CarND-MPC-Project/build
```

make output:

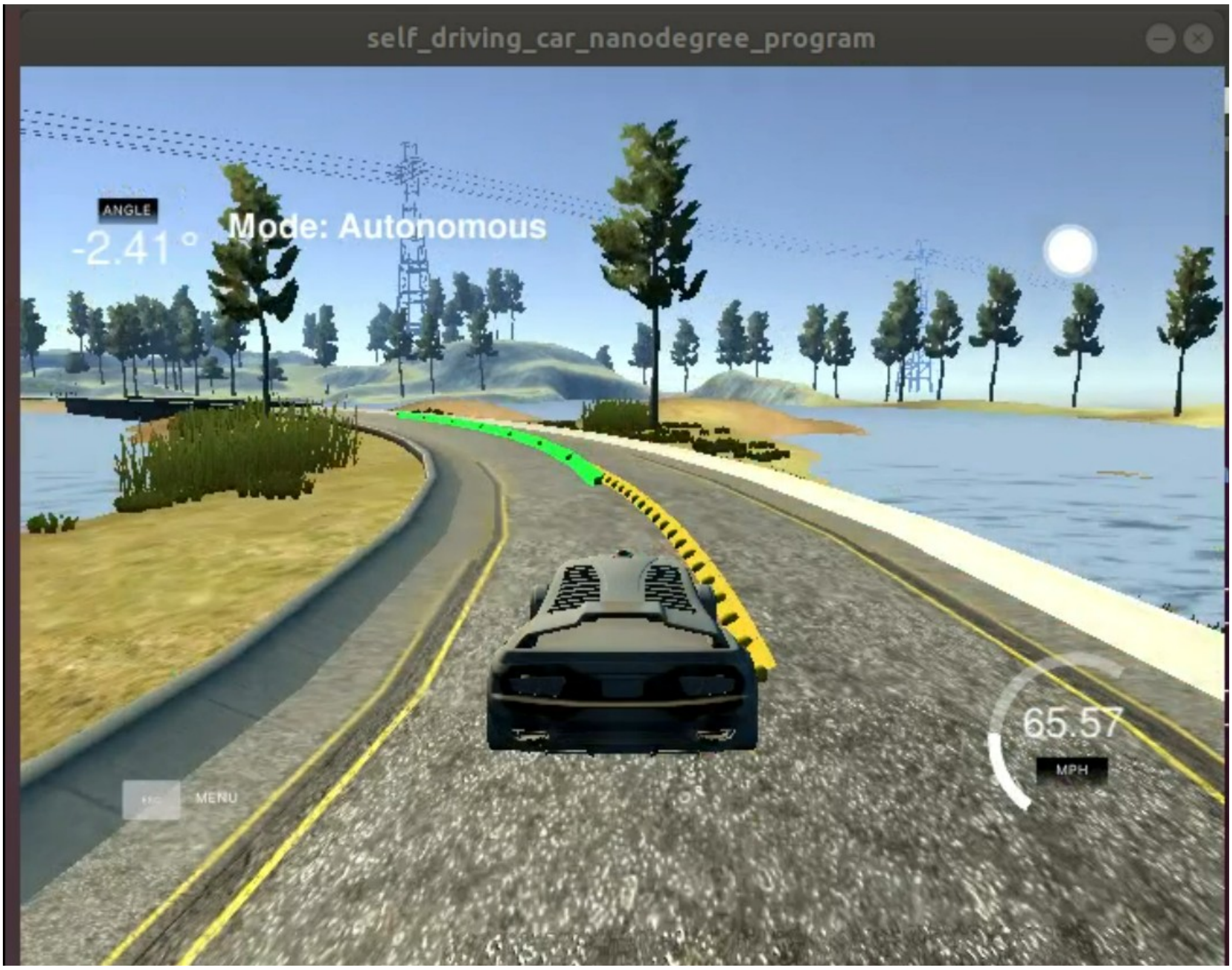
```
Scanning dependencies of target mpc
[ 33%] Building CXX object CMakeFiles/mpc.dir/src/MPC.cpp.o
[ 66%] Linking CXX executable mpc
[100%] Built target mpc
```

Simulator output:

Once the code is run following message is displayed. The application connects to the Simulator on port # 4567 , once the project is seleted and started on simulator as shown below:

Initially for few seconds, the car swings little bit and the green and yellow lines are not stable. However the car pick up speed and also the algorithm start minimizing the error.

Performance of car on curves.: The algorithm reduces the speed as the cross track error increases on curves:



By tunning the parameters ,the car is able to go 65mph even on curves.

RUBRICS:

This project satisfies all the rubric points

Rubric#1: Compilation

Criteria	Meets Specifications
Your code should compile.	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.

Code compiles without any error. The output can be seen in logs make-output.log and cmake-output.log

Rubric#2: - Implementation

Criteria	Meets Specifications
The Model	Student describes their model in detail. This includes the state, actuators and update equations.

The model used is Kinematic model (Bi cycle model). However, the model does not consider other interactions like interaction between tyres and road, cross correlation between x and y axis/

Below are the equations of the model.

```
x_[t] = x[t-1] + v[t-1] * cos(psi[t-1]) * dt
y_[t] = y[t-1] + v[t-1] * sin(psi[t-1]) * dt
psi_[t] = psi[t-1] + v[t-1] / Lf * delta[t-1] * dt
v_[t] = v[t-1] + a[t-1] * dt
cte[t] = f(x[t-1]) - y[t-1] + v[t-1] * sin(epsi[t-1]) * dt
epsi[t] = psi[t] - psides[t-1] + v[t-1] * delta[t-1] / Lf * dt
```

Where:

x, y : Car's position.
psi : Car's heading direction.
v : Car's velocity.
cte : Cross-track error.
epsi : Orientation error.
Constand Lf already given in the starter code (the distance between the car of mass and the front wheels)

Control parameters:

a : Car's acceleration (throttle).
delta : Steering angle.

Cost Function: *Below is the cost function used in the application in MPC.CPP*

```
//**************************Cost function ******************************

   // Cost function
   //initialize teh cost function to zero first
     fg[0] = 0;


   for (unsigned int t = 0; t < N; t++) {
     //penalize for cross track error
   //  fg[0] +=5000 * CppAD::pow(vars[cte_start + t], 2);      // 40mph to 80mph
     fg[0] +=10000 * CppAD::pow(vars[cte_start + t], 2);                 // 200mph
   // Penalize for orientation angle
   //  fg[0] +=5000 * CppAD::pow(vars[epsi_start + t], 2);     // 40mph to 80mph
       fg[0] +=10000 * CppAD::pow(vars[epsi_start + t], 2);   // 200mph
     // Penalize for stopping or driving too fast
        fg[0] +=  CppAD::pow(vars[v_start + t] - vel_ref, 2);
   }


   // Minimize the use of steering and brakes
   for (unsigned int t = 0; t < N - 1; t++) {
   //  fg[0] += 50*CppAD::pow(vars[delta_start + t], 2);    // 40mph to 80mph
    // fg[0] += 50*CppAD::pow(vars[a_start + t], 2);         // 40mph to 80mph
     fg[0] += 100*CppAD::pow(vars[delta_start + t], 2);  // 200mph
     fg[0] += 100*CppAD::pow(vars[a_start + t], 2);// 200mph

   // Minimize fast changing of steering and throttle / brake
   for (unsigned int t = 0; t < N - 2; t++) {
    // fg[0] +=   50000000 *CppAD::pow(vars[delta_start + t + 1] - vars[delta_start + t], 2);    //
40mph to 80mph
    //  fg[0] +=   50000 *CppAD::pow(vars[a_start + t + 1] - vars[a_start + t], 2);                // 40mph
to 80mph
     fg[0] +=   50000000 *CppAD::pow(vars[delta_start + t + 1] - vars[delta_start + t], 2);  //
200mph
     fg[0] +=   50000 *CppAD::pow(vars[a_start + t + 1] - vars[a_start + t], 2);        // 200mph
   }

  //**************************Cost function ******************************
```

The cost function tries to minimize the cross track error, Orientation angle error, stopping or driving too fast, minimize use of actuators and fast changing of actuators as shown above

Timestep Length and Elapsed Duration (N & dt)	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.

I went with N of 10 and dt of 0.1 (100 msecs) as given in the prjoect assignment. This is the optimum value that i could find. Increasing these slows down the simulator.

Polynomial Fitting and MPC Preprocessing	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.

The Polynomial fitting is defined in main.cpp using function polycit from line no 48 to 67 and called at line no 129. The waypoints provided by the simulator is transformed to the car cordinate system before the fit. Output of this function provides the co-effecient of 3rd degree polynomial

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

The actuator delay is defined in main.cpp at line no 123. i am using a delay of 0.1sec (=100msec). This delay is then used in line no 141 to 146 to calculate the state variables to handle thelatency before the solver is called.

Rubric#3: - Simulation

Criteria	Meets Specifications
The vehicle must successfully drive a lap around the track.	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.

The vehicle completes one whole lap around the track. The same is recorded in videos. The cost function is from line no 54 to 101. by accurately tuning the function, car stays on the driving lanes even during sharp turns.

There are two sets of identical codes for each of the cost function with different tuning parameters. The variable Val_ref can be set to a higher number for increased speed even at curves if line nos with comment //200mph is selected. The other set of lines can be uncommented and the current line can be commented for lower speeds val_ref.

Improvements:

1. The yellow lines are sometimes not accurate as the speed increases and they are shown behind the car.