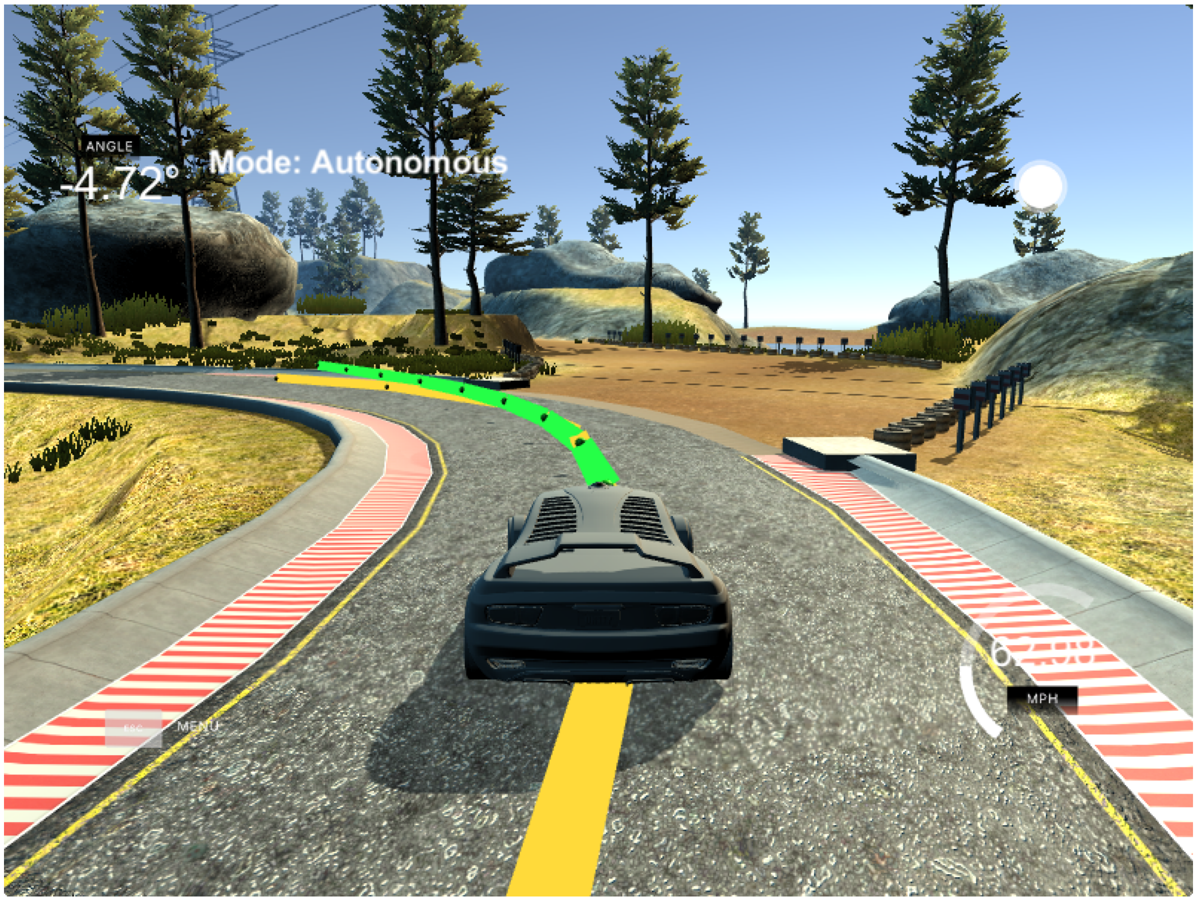
Model Predictive Control Project

Tao Li
litroncn@gmail.com



1. Introduction

In this project, we'll implement Model Predictive Control to drive the vehicle around the track even additional latency between command.

Model Predictive Control (MPC) is a control strategy that is suitable for optimizing the performance of constrained systems. Constraints are present in all control systems due to the physical and environmental limits on plant operation. Through a systematic handling of constraints, MPC can improve the performance of a system by allowing it to safely operate near constraint boundaries. This article develops a controller. Which is based on a nonlinear model of the plant and is successfully applied to a real-time 4 degrees-of-freedom(2 Cartesian coordinates, orientation and heading) autonomous vehicle system, used to simulate vehicle like motions in a simulator.

We will use the powerful tool Ipopt, CppAD to find locally optimal values while keeping the constraints set directly to the actuators and the constraints defined by the vehicle model.

2. Vehicle Model

2.1. Kinematic Bicycle Model

Kinematic models are simplifications of dynamic models that ignore tire forces, gravity, and mass. This simplification reduces the accuracy of the models, but it also makes them more tractable. At low and moderate speeds, kinematic models often approximate the actual vehicle dynamics. The vehicle model is described by

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = f(x_t, u_t) \tag{1}$$

where $f(x_t, u_t)$ is the vehicle dynamic state update equation. The vectors x_t and u_t collect the states and inputs of the vehicle at time t

$$x_t = [x_t \; y_t \; \psi_t \; v_t \; cte_t \; e_{\psi_t}] \tag{2a}$$
$$u_t = [\delta_t \; a_t] \tag{2b}$$

where cte_t represents the distance travelled along the center-line of the road, e_{ψ_t} the heading angle error between the vehicle and the path. x_t, y_t and ψ_t are the vehicle longitudinal position, lateral position and yaw angle, respectively. The inputs are the longitudinal acceleration a_t and the steering angle δ_t.

We've now defined the state, actuators and how the state changes over time based on the previous state and current actuator inputs.

Here defines update state as follows:

$$x_{t+1} = x_t + v_t * \cos(\psi_t) * dt \tag{3a}$$
$$y_{t+1} = y_t + v_t * \sin(\psi_t) * dt \tag{3b}$$
$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} * \delta_t * dt \tag{3c}$$
$$v_{t+1} = v_t + a_t * dt \tag{3d}$$
$$cte_{t+1} = \underbrace{f(x_t) - y_t}_{\text{cross track error}} + v_t * \sin(e_{\psi_t}) * dt \tag{3e}$$
$$e_{\psi_{t+1}} = \underbrace{\psi_t - \psi_{des_t}}_{\text{orientation error}} + \frac{v_t}{L_f} * \delta_t * dt \tag{3f}$$

Table 1. Notations

x, y	Position of the vehicle's CoM
ψ	Yaw angle of the car body
v	Vehicle speed
cte	Error between the center of the road and the vehicle's position
e_ψ	Heading angle error between the vehicle and the path
δ	Steering angle of the front wheels
a	Negative values signifying braking, positive values signifying accelerating
L_f	Distance between the front of the vehicle and its center of gravity

3. Model Predictive Control

In this section we explain the basis of MPC, the cost function used for optimizing, and an efficient algorithm to implement the control.

Model predictive control is an optimal control technique mainly used to solve control problems in presence of constraints. These constraints can represent physical capabilities of the controlled system, like the steering angle of a vehicle or its acceleration, or restrictions/requirements of the control problem that is being solved, like the road boundaries that the vehicle is not allowed to cross or its desired maximum orientation angle.

3.1. Mathematical Formulation

$$
\begin{aligned}
U_t^*(x(t)) = \underset{U_t}{\operatorname{argmin}} & \sum_{k=0}^{N-1} q(x_{t+k}, u_{t+k}) && \text{(4a)} \\
\text{s.t.} \quad & x_t = x(t) && \text{(4b)} \\
& x_{t+k+1} = Ax_{t+k} + Bu_{t+k},\ \forall k \geq 0 && \text{(4c)} \\
& x_{t+k} \in \mathcal{X}, u_{t+k} \in \mathcal{N},\ \forall k \geq 0 && \text{(4d)} \\
& U_t = \{u_t, u_{t+1}, \ldots, u_{t+N-1}\} && \text{(4e)}
\end{aligned}
$$

Problem is defined by
- ■ Objective that is minimized
- ■ Internal system model to predict system behavior
- ■ Constraints that have to be satisfied

3.2. Length and duration

The prediction horizon is the duration over which future predictions are made. Well refer to this as T. T is the product of two other variables, N and dt. N is the number of timesteps in the horizon. dt is how much time elapses between actuations. For example, if N were 20 and dt were 0.5, then T would be 10 seconds. N, dt, and T are hyperparameters you will need to tune for each model predictive controller you build. However, there are some general guidelines. T should be as large as possible, while dt should be as small as possible.

These guidelines create tradeoffs.

Horizon
In the case of driving a car, T should be a few seconds, at most. Beyond that horizon, the environment will change enough that it won't make sense to predict any further into the future.

Number of Timesteps
The goal of Model Predictive Control is to optimize the control inputs: [δ,a]. An optimizer will tune these inputs until a low cost vector of control inputs is found. The length of this vector is determined by N:

$$[\delta_1, a1, \delta_2, a_2, \ldots, \delta_{N1}, a_{N1}]$$

Thus N determines the number of variables optimized by the MPC. This is also the major driver of computational cost.

Timestep Duration
MPC attempts to approximate a continuous reference trajectory by means of discrete paths between actuations. Larger values of dt result in less frequent actuations, which makes it harder to accurately approximate a continuous reference trajectory. This is sometimes called "discretization error".

A good approach to setting N, dt, and T is to first determine a reasonable range for T and then tune dt and N appropriately, keeping the effect of each in mind.

3.3. Coordinate transformation

The simulator returns waypoints using the map's coordinate system, which is different than the car's coordinate system. Transforming these waypoints will make it easier to both display them and to calculate the CTE and e_ψ values for the model predictive controller.

Since the simulator speed is in mph and the latency is measured in seconds. So the speed from simulator and the reference speed should be converted to m/s before feeding the solver.

Listing 1. Car's coordinate transform.

```
h.onMessage([&](uWS::WebSocket<uWS::SERVER> ws, char *data, size_t length, uWS::OpCode ↩
    opCode) {
        ......

```

```
          vector<double> next_x_vals;
          vector<double> next_y_vals;

          Eigen::VectorXd car_x_vals(ptsx.size());
          Eigen::VectorXd car_y_vals(ptsy.size());
          for (int i = 0; i < ptsx.size(); ++i) {
            double dx = ptsx[i] - px;
            double dy = ptsy[i] - py;
            car_x_vals[i] = cos(-psi) * dx - sin(-psi) * dy;
            car_y_vals[i] = sin(-psi) * dx + cos(-psi) * dy;

            next_x_vals.push_back(car_x_vals[i]);
            next_y_vals.push_back(car_y_vals[i]);
          }

          ......
}
```

3.4. Latency

A safe control system requires sophisticated methods to handle the latency or delays brought by both the vehicles mechanical system and the communications systems.

A contributing factor to latency is actuator dynamics. For example the time elapsed between when you command a steering angle to when that angle is actually achieved. This could easily be modeled by a simple dynamic system and incorporated into the vehicle model. One approach would be running a simulation using the vehicle model starting from the current state for the duration of the latency. The resulting state from the simulation is the new initial state for MPC.

The both implementations can handle latency very well, the MPC can process latency either before or after the vehicle coordinate system transform.

Listing 2. Handle latency after car's coordinate transform.

```
h.onMessage([&](uWS::WebSocket<uWS::SERVER> ws, char *data, size_t length, uWS::OpCode ↩
    opCode) {
          ......

          double dt = 0.1;
          px = 0;
          py = 0;
          psi = 0;
          double cte  = polyeval(coeffs, 0.0);
          double epsi = -atan(coeffs[1]);

          px = px + v * cos(psi) * dt;
          py = py + v * sin(psi) * dt;
          const double Lf = 2.67;
          psi = psi + v * delta / Lf * dt;
          v = v + a * dt;
          cte = cte + v * sin(epsi) * dt;
          epsi = epsi + v * delta / Lf * dt;

          Eigen::VectorXd state(6);
          state << px, py, psi, v, cte, epsi;

          auto vars = mpc.Solve(state, coeffs, weights);
          ......
}
```

Listing 3. Handle latency before car's coordinate transform.

```
h.onMessage([&](uWS::WebSocket<uWS::SERVER> ws, char *data, size_t length, uWS::OpCode ↩
    opCode) {
        ......

        double dt = 0.1;
        px = px + v * cos(psi) * dt;
        py = py + v * sin(psi) * dt;
        const double Lf = 2.67;
        psi = psi + v * delta / Lf * dt;
        v = v + a * dt;

        Eigen::VectorXd state(6);
        state << px, py, psi, v, cte, epsi;

        auto vars = mpc.Solve(state, coeffs, weights);
        ......
}
```

3.5. Polynomial fit

We approximate X_{ref} and Y_{ref} over the next planning horizon T as third order polynomials of the curvilinear position, starting at the point closest to the vehicle's current position.

3.6. Cost function

The way to define the cost function used in this problem, once discretized and taking into account the prediction and control horizons is the one shown in equation (5)

$$\begin{aligned} J = &\sum_{t=0}^{N} (w_{cte}(cte_t - cte_{ref})^2 + w_{\psi}(e_{\psi_t} - e_{\psi_{ref}})^2 + w_v(v_t - v_{ref})^2) \\ &+ \sum_{t=0}^{N-1} (w_{\delta}\delta_t^2 + w_a a_t^2) \\ &+ \sum_{t=0}^{N-2} (w_{\dot{\delta}}(\delta_{t+1} - \delta_t)^2 + w_{\dot{a}}(a_{t+1} - a_t)^2) \end{aligned} \tag{5}$$

3.7. Constraints

One of the obvious constraint to define is the restriction of the vehicle to not pass the road boundaries. It is desired to set the middle of the lane as $cte = 0\ m$. Another constraint to define for security reasons and for the capabilities of the vehicle motor is the velocity, steering angle and acceleration. The velocity cannot be negative and it is capped at $v_{max} \approx 29\ m/s$ and regarding the acceleration it is capped at $-1\ m/s^2 \leq a \leq\ 1m/s^2$. For comfort reasons, the steering angle (δ) variations have also been constrained with values of $-25° \leq \delta \leq 25°$

3.8. Tuning parameters

The parameters that have to be tuned for the controller to work as good as possible are the prediction horizon N, the weights, as well as the sampling time. First of all the number of timesteps(N) and the timestep duration(dt) have to be chosen such that $N * dt \approx 1.5\ seconds$. In the case of driving a car, T should be a few seconds, at most. Beyond that horizon, the environment will change enough that it won't make sense to predict any further into the future. However, the choice of N will influence the number of variables, which is the major driver of computational cost.

Since the prediction horizon does not affect the other parameters, the tuning of the horizon length N is less involved. The only requirement is that the optimization problem should be feasible. Thus, it becomes clear that the horizon length will be dependent on the sampling distance, which can be achieved for multiple combinations of sampling time and speed. If the sampling distance is short, a longer horizon will be needed, and vice versa. In short, it is desirable to keep N small for computational reasons, but on the other hand it has to be suffciently large for the problem to be feasible.

This article recommends tuning the weights on the inputs as well as the rate of change of the inputs. Penalizing the rate of change yields a more robust controller but at the cost of the controller being more sluggish. Setting a small penalty or none whatsoever gives a more aggressive controller that is less robust. For instance, tuning weight of the steering value in the cost function results in a smoother steering transitions.

Here are my final hyperparameters shown in table 2, The hyperparameters are used in the nominal optimal control problem meaning that the mangnitude of lateral control (steering) are penalized more than longitudinal control (throttle and brake).

model and latency	N	dt	v	w_{cte}	w_{ψ}	w_v	w_{δ}	w_a	$w_{\dot{\delta}}$	$w_{\dot{a}}$
kinematic - 100 ms	10	0.15	29 m/s	1.0	1.0	1.0	1000.0	1.0	500.0	1.0

Table 2. Final hyperparameters.

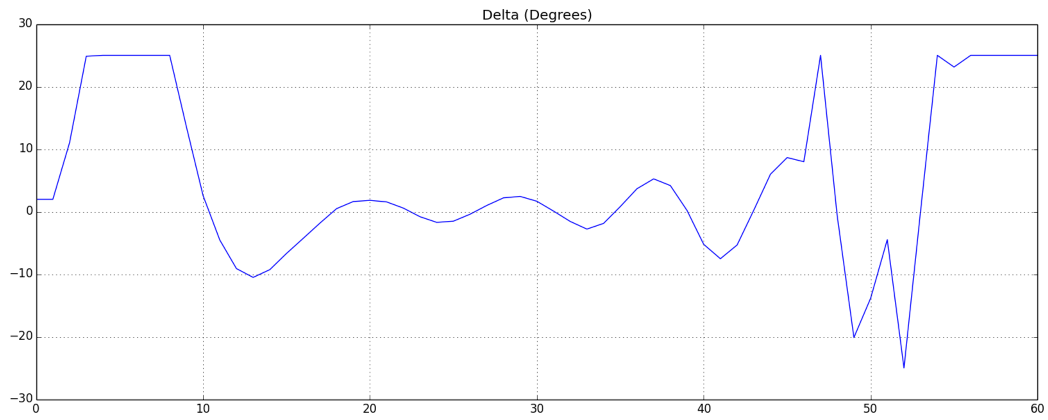


Figure 1. $w_{\delta} = 1$ $w_{\dot{\delta}} = 100$

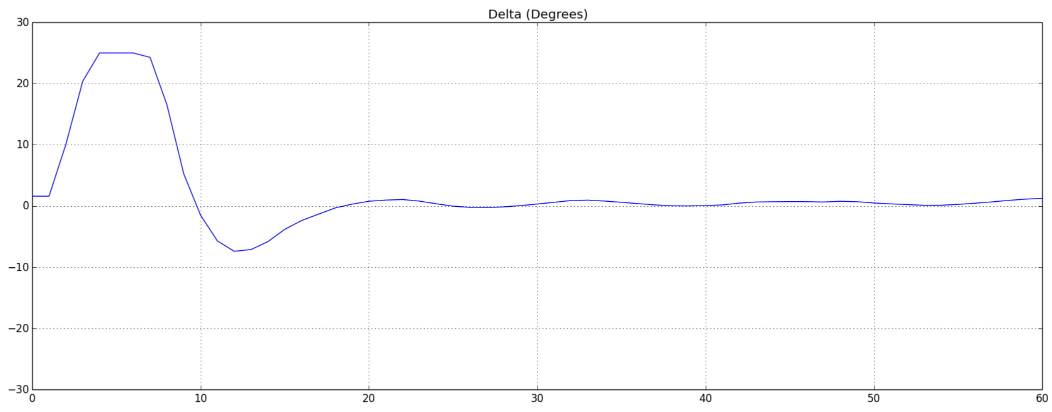


Figure 2. $w_{\delta} = 1$ $w_{\dot{\delta}} = 500$

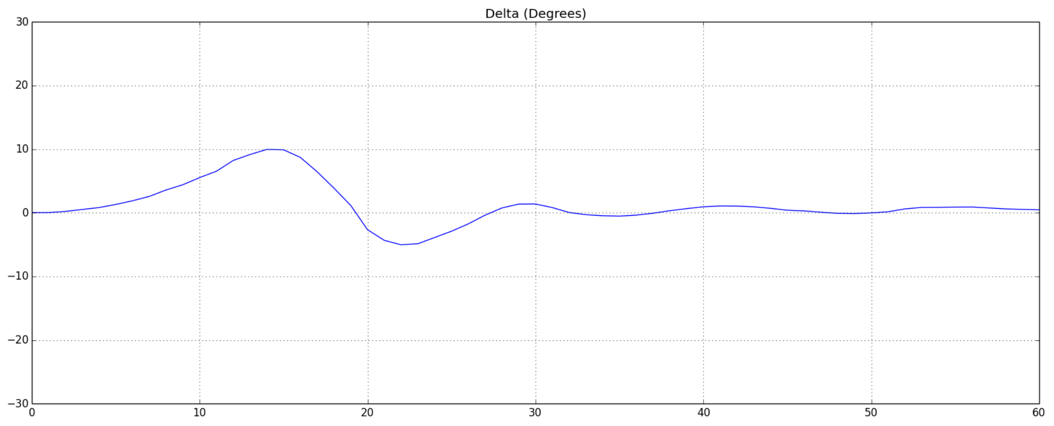


Figure 3. $w_{\delta} = 50\ w_{\dot{\delta}} = 500$

As we can see increasing weight of the steering value in the cost function results in a smoother steering transitions.

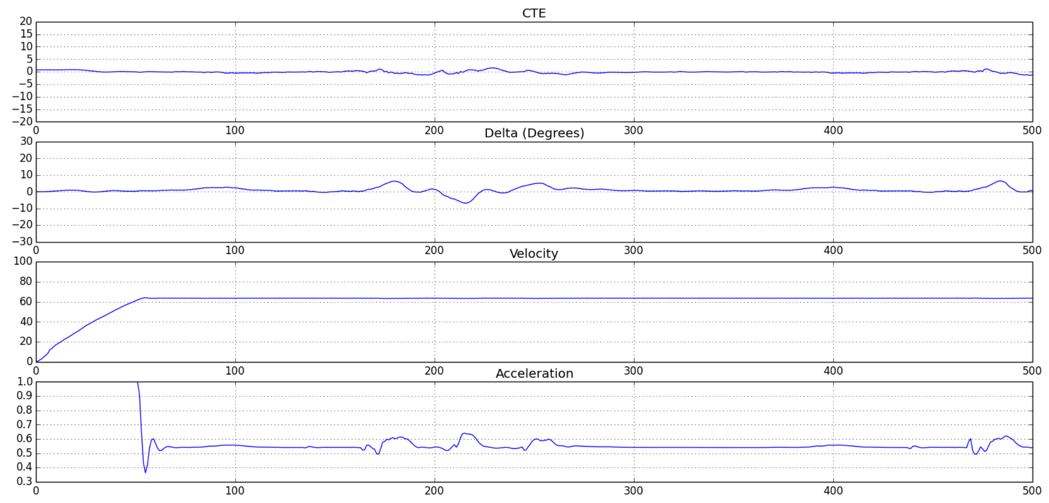


Figure 4. $N = 10\ dt = 0.15$

4. Conclusion

In this work, the MPC is able to identify the vehicle dynamics, steer the vehicle close to the reference trajectory and to operates the vehicle close to the limit of its handling capability.

Acknowledgement: We are grateful to Udacity for their valuable courseware.

5. Video

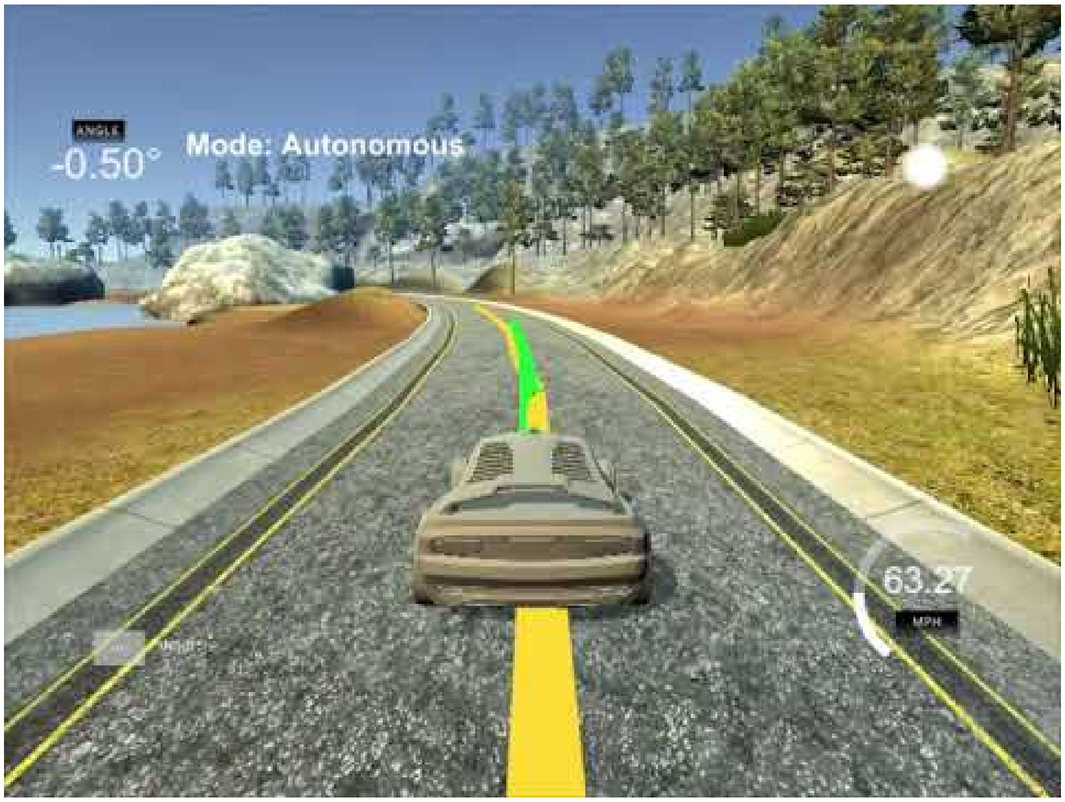


Figure 5. https://youtu.be/CK5Fq4vTkp0

References

[1] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4029–4036, 2014.

[2] J. Ji, A. Khajepour, W. Melek, and Y. Huang. Path Planning and Tracking for Vehicle Collision Avoidance based on Model Predictive Control with Multi-Constraints. In *IEEE Transactions on Vehicular Technology*, vol. 9545, no. April, pp. 1–1, 2016.

[3] E. Siampis, E. Velenis, and S. Longo. Torque Vectoring Model Predictive Control with Velocity Regulation Near the Limits of Handling. In *Vehicle System Dynamics*, pp. 2553–2558, 2015.