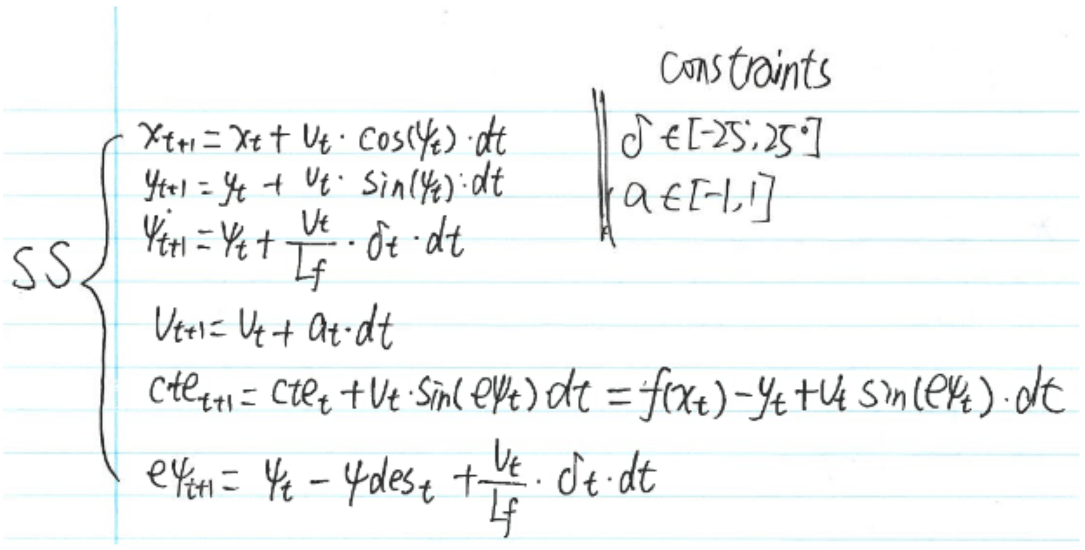
Report of Self-Driving-Car Project: Model Predictive Control

Introduction

In this project, a MPC was implemented to drive a car around the track in the car simulator. Additionally, there’s a 100 millisecond latency between actuations commands on top of the connection latency.

Model

In this project, a motion bicycle model was adoptted. The state space representation of the system is:



$$SS\begin{cases} x_{t+1} = x_t + v_t \cdot \cos(\psi_t) \cdot dt \\ y_{t+1} = y_t + v_t \cdot \sin(\psi_t) \cdot dt \\ \psi_{t+1} = \psi_t + \frac{v_t}{L_f} \cdot \delta_t \cdot dt \\ v_{t+1} = v_t + a_t \cdot dt \\ cte_{t+1} = cte_t + v_t \cdot \sin(e\psi_t) dt = f(x_t) - y_t + v_t \sin(e\psi_t) \cdot dt \\ e\psi_{t+1} = \psi_t - \psi des_t + \frac{v_t}{L_f} \cdot \delta_t \cdot dt \end{cases}$$

Constraints

$$\delta \in [-25°, 25°]$$
$$a \in [-1, 1]$$

where

- x is the longitudinal coordinate
- y is the lateral coordinate
- ψ is the yaw angle of the car body
- v is the velocity of the car
- cte is the cross track error
- and e_ψ is the error of the yaw angle according to the desired trajectory.

The control inputs of the car is δ and a. δ is the yaw angle (steer angle) of the front wheels and a is the accleration.

According to the simulator, the steer angle is within [-25, 25] degree. To get the maximum accleration, set the throtle to 1 for a while and get the difference of velocity. Since $a = \frac{dv}{dt}$, the maximum accleration is around $2m/s^2$.

Sample Time and Prediction Horizon

Ideally, the vehicle model should match real world physics as much as possible. That said, the discrtized time step (sample time) should be as small as possible so that the discretization error is negligible.

The predicted distance is roughly $v \times N \times dt$; in order to go through the bend road smoothly, the predicted distance should be around 20 metres. Assuming the car is driving at 50km/h, the total predicted time should be around $20 \times 3.6/50 = 1.44s$.

However, we are driving the simulated car instead of a real car. The car simulator updates states of the car with variable time steps. Open the file `ProjectSettings/TimeManager.asset` of the simulator repo, we find:

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!5 &1
TimeManager:
  m_ObjectHideFlags: 0
  Fixed Timestep: .0199999996
  Maximum Allowed Timestep: .333333343
  m_TimeScale: 1
```

where the maximum `fps` is 1/0.02 = 50 and the minimum `frs` is 1/0.333=3.

The sample time of MPC should match the time step that the simulator updates the control inputs of the car. I tried the sample time with `1/50`, `1/20`, `1/10`, `1/8`, `1/5`, `1/3` second and it turned out that `1/3` performed the best. And the prediction horizon is 8; as a result, the total prediction time is `8/3=2.7` second.

To make the mpc solver more stable and faster, a buffer was implemented to store previous solution. See `line 58` of [MPC.h](./src/MPC.h) and `line 213 and 301-304` of [MPC.cpp](./src/MPC.cpp):

```
// MPC.h
  std::vector<float> var_init;

// MPC.cpp
  Dvector vars(n_vars);
  for (int i = 0; i < n_vars; i++)
  {
    vars[i] = var_init[i];//0.0;
  }

  for (size_t i = 0; i < var_init.size(); i++)
  {
    var_init[i] = solution.x[i];
  }
```

The configuration of the MPC controller was set at `line 48-54` of [main.cpp](./src/main.cpp):

```
  // set mpc configuration
  mpc.Lf_ = 1.67;
  mpc.setHorizon(8);
  mpc.ref_v_ = 100 * 0.44704; // m/s
  mpc.dt_ = 1 / 3.;
  mpc.a_max = 2.0;
  mpc.a_min = -1.0;
```

Polynomial Fitting and MPC Preprocessing

To get the polynomial coefficients of the desired trajectory, first tansfer world coordinate way point positions to vehicle coordinate. Because

$$p^W = R^v p^v + T^v,$$

we get

$$p^v = (R^v)^T (p^W - T^v),$$

where p^W is world coordnate position and p^v is vehicle coordinate position, and R^v, T^v are the rotation and translation component of the vehicle.

This part was implemented in `line 113-124` of [main.cpp](./src/main.cpp):

```
  vector<WayPoint> wpts(ptsx.size());
  vector<double> vx(ptsx.size()), vy(ptsx.size()); // way points in vehicle coordinate system
  Eigen::VectorXd vc_x(ptsx.size());
  Eigen::VectorXd vc_y(ptsx.size());
  // P_v = R_c' * (P_w - P_c)
  for (size_t i = 0; i < ptsx.size(); i++)
  {
    wpts[i].x = cos(psi) * (ptsx[i] - px) + sin(psi) * (ptsy[i] - py);
    wpts[i].y = -sin(psi) * (ptsx[i] - px) + cos(psi) * (ptsy[i] - py);
    vc_x[i] = wpts[i].x;
    vc_y[i] = wpts[i].y;
  }
```

To smooth the desired trajectory, a way point buffer was implemented in `line 37-48` [helper.h](./src/helper.h):

```
struct PointsBuffer
{
  int num;
  std::deque<vector<double>> x_buf;
  std::deque<vector<double>> y_buf;

public:
  PointsBuffer() { num = 4; };
  void setBufferSize(const int s) { num = s; }
  void updateBuffer(const vector<double> &x, const vector<double> &y);
  void getPoints(vector<double> &x, vector<double> &y);
};
```

The size of point buffer was set to 5 in `line 70` of `main.cpp`:

```
ptsBuffer.setBufferSize(5);
```

Model Predictive with Latency

Processing latency is easy within MPC. Assuming the vehicle is moving constantly in the future time period of latency, solve the MPC by shifting the car’s initial states by `Latency` in time domain. This was implemented in `line 135-148` of *main.cpp*:

```
    double Latency = 0.1; //0.1;//0.1;     // 100 ms
    double Lf = mpc.Lf_;

    double x0 = 0, y0 = 0, psi0 = 0, v0 = v, cte0 = polyeval(coeffs, x0) - y0;
    double epsi0 = psi0 - atan(coeffs[1] + 2 * coeffs[2] * x0 + 3 * coeffs[3] * x0 * x0);

    double x1 = x0 + Latency * v0; // + 0.5 * Latency * Latency * acc;
    double psi1 = psi0 + v0 * delta / Lf * Latency;
    double y1 = y0; // + v * Latency * psi1 / 2.;
    double v1 = v0 + acc * Latency;
    double cte1 = cte0 + v0 * sin(epsi0) * Latency;
    double epsi1 = epsi0 + v0 * delta * Latency / Lf;

    state << x1, y1, psi1, v1, cte1, epsi1;
```

Due to the difference of vehicle coordinate system and unity coordinate system, we need to

- transfer velocity unit from mph to m/s
- convert throtle to acceleration in m/s2
- revert the sign of steer angle

Those were implemented in `line 99 and 101-104` of `main.cpp`.

Demo Simulation

Compile the program and run the program simutaneously with the simulator. The car can drive stably at max speed of 100mph.

See the recorded [video](./images/2018-05-28_22-34-53.mp4)