Model Predictive Control (MPC)

COMPILATION

Your code should compile

It compiles with cmake / make in my MacbookPro with XCode 9.1 installed

Implementation

The Model

The model is a simple kinematic model, with the turn equation as:

$$\frac{d\varphi}{dt} = \frac{v(t)*\delta(t)}{Lf}$$

The rest of the equations are the usual ones for speed and acceleration:

$$x(t) = \int_0^t v(t) * \cos\big(\varphi(t)\big)\, dt$$
$$y(t) = \int_0^t v(t) * \sin\big(\varphi(t)\big)\, dt$$
$$\varphi(t) = \int_0^t \frac{v(t)*\delta(t)}{Lf} dt$$
$$v(t) = \int_0^t a(t)\, dt$$

All the calculations are done in car coordinates so cross track error and psi error are computed as:

cte(x) = polynomial(x) – car_y

$$\varphi err(x) = \frac{d\big(polynomial(x)\big)}{dx} - \varphi(x)$$

The waypoints are adjusted by a third order polynomial.

Equations are represented in the MPC.cpp file lines 132 – 160. They are the same as for the quiz with some changes for the 3rd order polynomial.

Timestep Length and Elapsed Duration (N and dt)

I have tried different values with N ranging from 10 to 50 and dt from 0.05 to 0.25.

At low speeds, there are not much problems and everything converges nicely but as you increment speed there is a need to reduce N and dt. If N * dt is too big then the end part of the predicted trajectory takes too much importance.

Also, it generates strange solutions in closed bends.

Lastly, I set in a N = 15 and dt = 0.05 which gives good convergence over speeds from 10 to near 100.

Polynomial Fitting and MPC Preprocessing

First waypoints are converted to the car coordinate system with

```
std::tuple<double, double>transformToCar(double x, double y, double x_car, double y_car,
double sigma){

    double dx = x - x_car;
    double dy = y - y_car;
    double carx= cos(sigma)*dx + sin(sigma)*dy;
    double cary = -sin(sigma)*dx + cos(sigma)*dy;

    return std::make_tuple(carx, cary);
}
```

Then a third order polynomial is adjusted. I tried with a second order one but it seems to me a third order gives better results.

Model Predictive Control with Latency

First try was to update car position with the received speed and turn variables for the latency time and solve afterwards. Resulst were not good with too much oscillations.

Finally, I compute which solution index I must use by dividing latency by the timestep:

```
    double latency = 0.1;
    int step = floor(latency/dt);
    return {solution.x[x_start + 1+step],   solution.x[y_start + 1+step],
        solution.x[psi_start + 1+step], solution.x[v_start + 1+step],
        solution.x[cte_start + 1+step], solution.x[epsi_start + 1+step],
        solution.x[delta_start+step],   solution.x[a_start+step], cost};
```

Simulation

The vehicle must successfully drive a lap around the track

Once the algorithm was working it easily did a turn around the track at low 10 speed.

As I increased the speed I had to tune equations to get less oscillations in closed bends but there is no fun without speed and didn't arrived to a good solution for getting speeds over 50 with the latency. The turn after the bridge was really bad.

So, I included a speed tuning algorithm. It computes how much turns the road and adjusts the speed accordingly:

```
    // Supose we are at x = 0 because that is our frame of reference
```

```
// We compute the direction of the pol. at x = v * dt* N

double lx = v * dt * N ;
double deriv = 3 * coeffs[3] * lx*lx + 2 * coeffs[2]*lx + coeffs[1];
double psil = atan(deriv);

// We use a simple algoritm to set speed. That way in straight roads
// we go fast and in bends we slow.

ref_v = (max_v-min_v) * (1 - fabs(psil)*dec_factor/M_PI) + min_v;
```

where we have

- Max_v : Maximum speed we want to attain
- Min_v : Minimum speed
- Dec_factor : a constant to reduce speed with the turn

Results are much much better.

With max_v = 100, min_v = 45 and dec_factor=2 it runs between lines with some oscillation in closed bends and speeds that attain more than 90.

With max_v = 130 we gain a little speed but touch the white / red areas but no curb.

max_v , min_v and dec_factor are set in MPP.cpp lines 32/34.