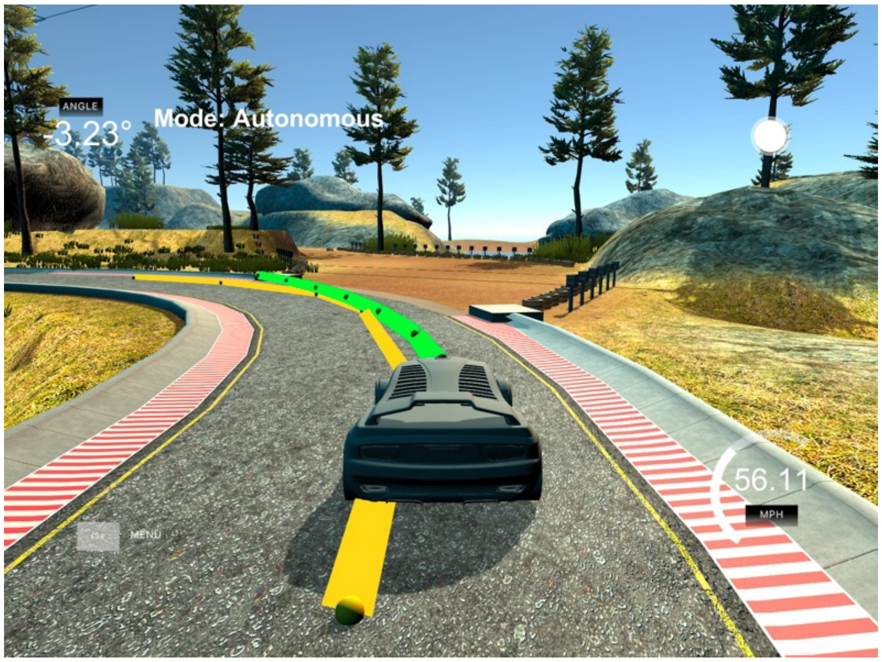
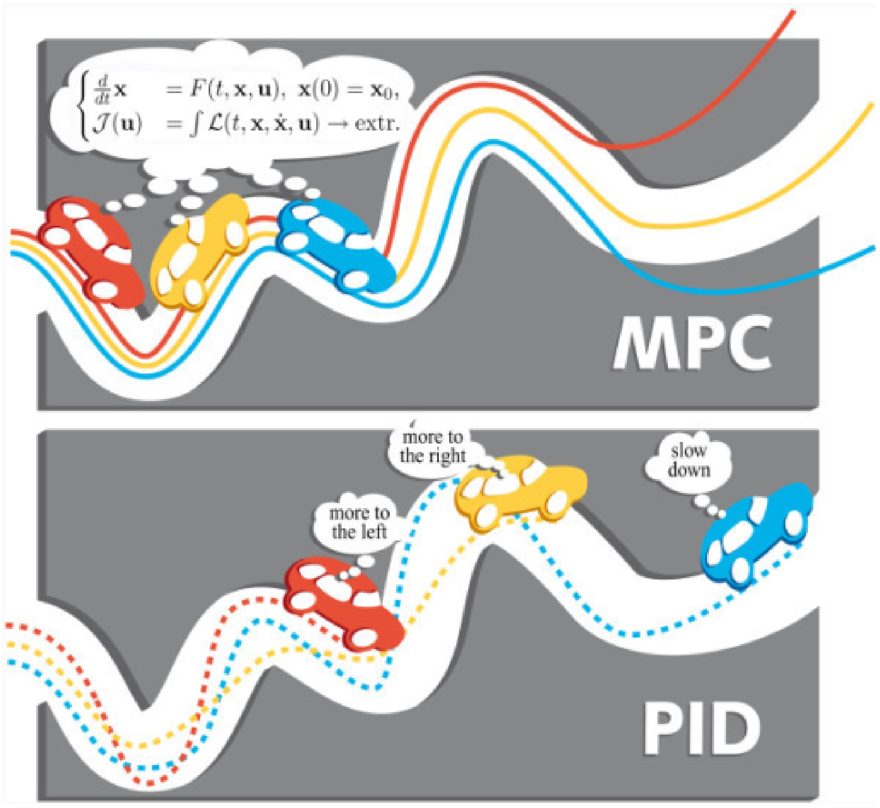
Model Predictive Control — Udacity's Self-driving Car Nanodegree

Dhanoop Karunakaran Follow
Jun 11, 2018 · 5 min read



In my previous post, I have explained about **PID controller** which is a popular algorithm used in control theory. It's a difficult challenge for a PID controller to overcome latency. But a **Model predictive control** (**MPC**) can adapt well because we can add latency in the system.

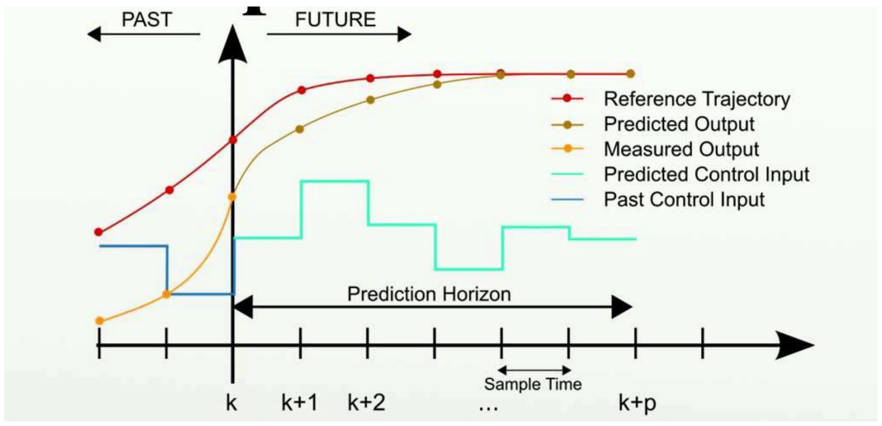
> *In a real car, an actuation command won't execute instantly—there will be a delay as the command propagates through the system. A realistic delay might be on the order of 100 milliseconds. This is a problem called* ***latency****.*


$\begin{cases} \frac{d}{dt}\mathbf{x} &= F(t,\mathbf{x},\mathbf{u}), \ \mathbf{x}(0)=\mathbf{x}_0, \\ \mathcal{J}(\mathbf{u}) &= \int \mathcal{L}(t,\mathbf{x},\dot{\mathbf{x}},\mathbf{u}) \to \text{extr.} \end{cases}$
MPC
more to the right
slow down
more to the left
PID

Source

1. Model Predictive Control(MPC)

MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints. In other words, MPC can take a vehicle's motion model into account to plan out a path that makes sense given a set of constraints, based on the limits of the vehicle's motion, and a combination of costs that define how we want the vehicle to move (such as staying close to the best fit and the desired heading, or keeping it from jerking the steering wheel too quickly).



Source: https://en.wikipedia.org/wiki/Model_predictive_control

MPC considers the following trajectory as an optimisation problem in which the solution is the car should take. MPC helps to predict a resulting trajectory with minimum cost. Then the car follows that trajectory and gets new input to calculate a new set of trajectories to optimize.

The optimization considers only a short duration's worth of way points, because that's all we need to plan for. We can tune the number of discrete points in time that the optimizer uses in its plan, as well as the time gap between them.

2. Important concepts in MPC

Model

The MPC model is based on kinematic bicycle model defined by a state of six parameters:

And the vehicle model equation is:

$$x_{t+1} = x_t + v_t cos(\psi_t) * dt$$

$$y_{t+1} = y_t + v_t sin(\psi_t) * dt$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} \delta_t * dt$$

$$v_{t+1} = v_t + a_t * dt$$

$$cte_{t+1} = f(x_t) - y_t + (v_t sin(e\psi_t) dt)$$

$$e\psi_{t+1} = \psi_t - \psi des_t + (\frac{v_t}{L_f} \delta_t dt)$$

From Udacity lecture

> x, y—position.

> psi(ψ)—orientation.

> v—velocity.

> cte—cross-track error. The difference between the trajectory defined by the waypoints and the current vehicle position y in the coordinate space of the vehicle.

> epsi (eψ)—orientation error.

Actuator constraints

There are some actuator constraints as below put in place:

· a—acceleration is in the range [-1, 1] = [full brake, full throttle]

· delta—steering angle is in the range [-25°, 25°]

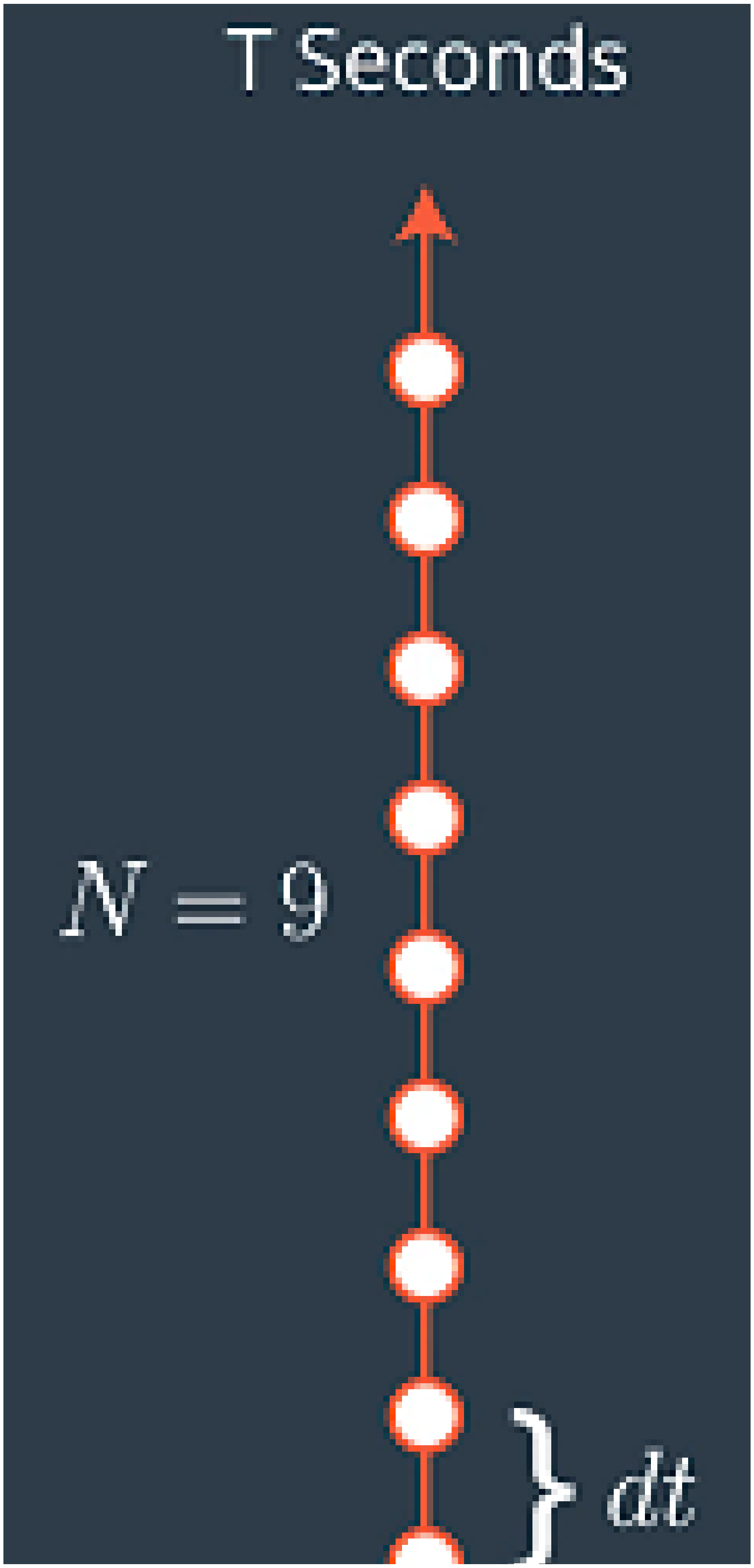
Cost function

To optimize the trajectory of the vehicle I used a cost function that would rate any trajectory on their cross track error, steering error, velocity, steering angle, acceleration, change in steering angle, and change in throttle.

Since those objectives may be contradictory to others, we add weights to the cost to reflect its priority. Here is the cost function:

$$
\begin{aligned}
J = & \sum_{t=1}^{N} w_{cte}\|cte_t\|^2 + w_{e\psi}\|e\psi_t\|^2 + w_v\|v_t - v_{target}\|^2 \\
& + \sum_{t=1}^{N-1} w_{\delta}\|\delta_t\|^2 + w_a\|a_t\|^2 \\
& + \sum_{t=2}^{N} w_{rate_{\delta}}\|\delta_t - \delta_{t-1}\|^2 + w_{rate_a}\|a_t - a_{t-1}\|^2
\end{aligned}
$$

Time-step Length(N) and Elapsed Duration(dt)


T Seconds
$N = 9$
dt

From Udacity lecture

The prediction horizon(T) is the duration over which future predictions are made. N is the number of time-steps in the horizon. dt is how much time elapses between actuations. N, dt, and T are hyper-parameters that need to be tuned for each model predictive controller we build.

Polynomial Fitting and MPC Preprocessing

The way points provided by the simulator are transformed to the car coordinate system.

```
// converting to vehicle coordinates
for(int i=0;i<ptsx.size();i++){
  double diffx = ptsx[i]-px;
  double diffy = ptsy[i]-py;
  ptsx[i] = diffx * cos(psi) + diffy * sin(psi);
  ptsy[i] = diffy * cos(psi) - diffx * sin(psi);
}

Eigen::VectorXd ptsxV = Eigen::VectorXd::Map(ptsx.data(),
ptsx.size());
Eigen::VectorXd ptsyV = Eigen::VectorXd::Map(ptsy.data(),
ptsy.size());
```

Then a 3rd-degree polynomial is fitted to the transformed waypoints.

```
auto coeffs = polyfit(ptsxV, ptsyV, 3);
```

These polynomial coefficients are used to calculate the cte and epsi later on.

```
double cte_l =  polyeval(coeffs, 0) + v * CppAD::sin(-
atan(coeffs[1])) * dt;
double epsi_l = -atan(coeffs[1])+psi_l;
```

They are used by the solver as well to create trajectory that is equivalent to the reference trajectory from path planning module.

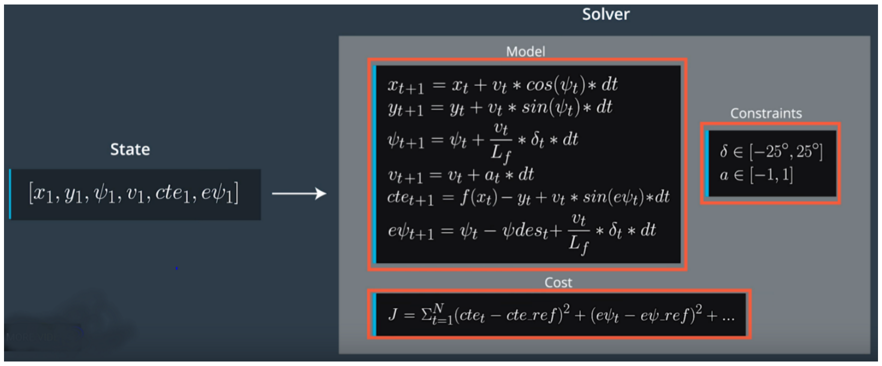
```
std::vector<double> r;
r = mpc.Solve(state, coeffs);
```

Latency

In a real car, there will be latency in a command that propagates through the system. In the simulator, it is 100 millisecond. We need to accommodate this time difference to predict waypoints for the trajectory.

3. Implementation

First step is to define the Timestep Length (N) and Elapsed Duration(dt).Then define the solver which consists of model, actuator constraints and cost function to produce the control input that minimise the cost function to follow a trajectory that we got from path planning module.



$$[x_1, y_1, \psi_1, v_1, cte_1, e\psi_1]$$

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

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

$$J = \Sigma_{t=1}^{N}(cte_t - cte\_ref)^2 + (e\psi_t - e\psi\_ref)^2 + ...$$

From Udacity lecture

Once it is defined, as mentioned earlier, solver produces the vector control input that minimize cost function by inputting current states and coefficients of the polynomial drawn on center of the road. This function gave throttle and steering value as the output. Actuator values were decided at each time step such that it minimises the cost function.

```
std::vector<double> r;
 r = mpc.Solve(state, coeffs);
```

Then we apply the first control input to the vehicle and repeat the loop.

If you would like to see full code in action, Please visit my Github repo.

If you like my write up, follow me on Github, Linkedin, and/or Medium profile.

References

1. Udacity self-driving car engineer nanodegree.
2. https://en.wikipedia.org/wiki/Model_predictive_control