1 Euler's Method

It's the simplest method of numerical solution for ordinary differential equations (ODEs). Suppose we are given the following ODE with and initial condition

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \tag{1}$$

where $y'(t) = \frac{\mathrm{d}y}{\mathrm{d}t}$ denotes derivative. Derivatives with respect to time, which are common in physics, are conventionally denoted by the dot symbol. That is, $\dot{y}(t) \equiv y'(t) \equiv \frac{\mathrm{d}y}{\mathrm{d}t}$. Equation (1) is in continuous time t[1]. The solution of the ODE above is a function of time $y(t)$ for which the above equation holds. Since we are working with computers, we can't store functions of continuous variable[2] and so we need to discretize the solution. In other words, we wanna convert it to an equation in discrete time, because it's easier to implement.

The Euler method chooses some finite time step h, so that k-th discrete time step can be expressed as $t_k = t_0 + kh$. One step of the Euler method from t_k to $t_{k+1} = t_k + h$ is given by

$$y_{k+1} = y_k + hf(t_k, y_k), \tag{2}$$

where, obviously, $y_k = y(t_k)$. As you can see, the recursion can be started with y_0 at t_0, because both of these are given. We can rewrite the Euler method as

$$\frac{y_{k+1} - y_k}{h} = f(t_k, y_k) \tag{3}$$

So, for example if my equation for the change in the x-coordinate in the continuous time is given by

$$\dot{x}(t) = v(t)\cos(\psi(t)), \tag{4}$$

than, by application of the Euler method, the discretized version with step size $h = \mathrm{d}t$ will look like this

$$x(t_{k+1}) = x(t_k) + v(t_k)\cos(\psi(t_k))\mathrm{d}t. \tag{5}$$

Now, to comport with the notational conventions, we are gonna use the lower index to denote the discre time steps. Since the time steps are only distinguished by the variable k, using variable t_k is a bit redundant, so we will get rid of it too. Ultimately, we get

$$x_{k+1} = x_k + v_k\cos(\psi_k)\mathrm{d}t. \tag{6}$$

With the help of the Euler method, we have converted a *differential* equation into a *difference* equation. There are plenty of other, more accurate, methods for ODE integration, such as Runge-Kuta method (see Wikipedia for more), but they are more complicated.

[1]Since this is mathematics we're dealing with, the variable t does not have to represent time, of course.

[2]Because that would take up infinite memory.

2 Kinematic Bicycle Model

This is the kinematic bicycle model

$$\dot{x} = v\cos(\psi + \beta) \tag{7a}$$
$$\dot{y} = v\sin(\psi + \beta) \tag{7b}$$
$$\dot{\psi} = \frac{v}{l_r}\sin(\beta) \tag{7c}$$
$$\dot{v} = a \tag{7d}$$
$$\dot{\beta} = \arctan\left(\frac{l_r}{l_r + l_f}\tan(\delta_f)\right) \tag{7e}$$

Note, that the Cartesian position x, y, the speed[3] v, the inertial heading angle ψ, the angle of the current velocity β, the steering angle $\delta \triangleq \delta_f$ and acceleration a are all functions of time. In ODE speak, these would be called *dependent variables*, whereas t is the *independent variable.* Guess why! The distances from the center of gravity to the rear l_r axle and to the front l_f axle are constant parameters, which vary depending on the vehicle. The Fig. 1, stolen from Kong et al. [2015], depicts the geometric meaning of the variables involved.

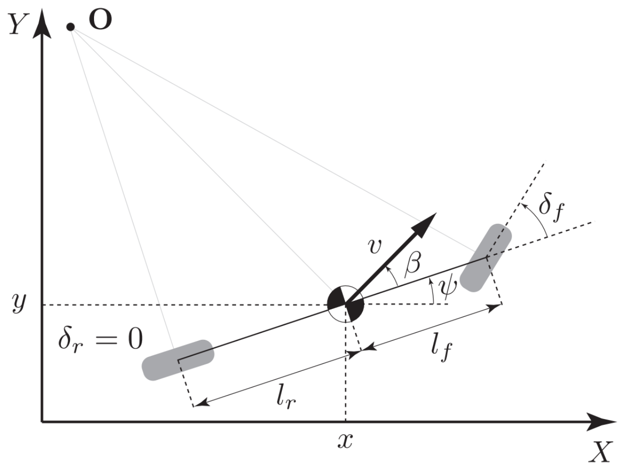

Figure 1: Kinematic bicycle model.

For some undisclosed reason, at Udacity, we are using a simplified version, given by

$$\dot{x} = v\cos(\psi) \tag{8a}$$
$$\dot{y} = v\sin(\psi) \tag{8b}$$
$$\dot{\psi} = \frac{v}{l_f}\delta \tag{8c}$$
$$\dot{v} = a \tag{8d}$$
$$\tag{8e}$$

[3]a.k.a. velocity magnitude

I guess, this model assumes, among other things, that the velocity vector always points forward (in direction of the longitudinal axis of the vehicle) and the rear axle distance $l_r = 0$, but don't quote me on that. Applying the Euler method from Section 1 yields the familiar equations

$$\begin{aligned}
x_{k+1} &= x_k + v_k \cos(\psi_k)\mathrm{d}t && \text{(9a)}\\
y_{k+1} &= y_k + v_k \sin(\psi_k)\mathrm{d}t && \text{(9b)}\\
\psi_{k+1} &= \psi_k + \frac{v_k}{l_f}\delta_k \mathrm{d}t && \text{(9c)}\\
v_{k+1} &= v_k + a_k \mathrm{d}t && \text{(9d)}\\
& && \text{(9e)}
\end{aligned}$$

where the $\mathrm{d}t$ is the discretization step size. The equations above conform to the nonlinear discrete-time state-space model in the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k), \qquad \boldsymbol{x}_0 \tag{10}$$

where the state of the system is given by $\boldsymbol{x}_k = [x_k,\ y_k,\ \psi_k,\ v_k]^\top$ and $\boldsymbol{f} : \mathbb{R}^4 \to \mathbb{R}^4$ is the system dynamics. The control vector $\boldsymbol{u}_k = [a_k,\ \delta_k]^\top$ contains the system inputs; that is, the acceleration a_k and the steering angle δ_k, that influence the system state and outputs.

3 Model Predictive Control

The goal of the controller is to determine optimal values of the control variables at each time step, such that the specified optimality criteria[4] are met. If you're familiar with classical control theory, you've surely seen the feedback control loop depicted in Fig. 2. It won't come as a surprise that the MPC follows the

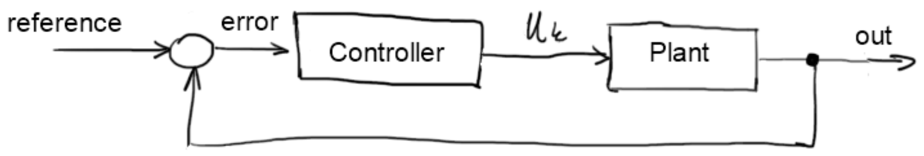

Figure 2: Classical control loop.

same general idea of the feedback loop[5].

Before applying the MPC algorithm, we will extend the state-space with a cross-track error $\tilde{y}_k$ and heading error $\tilde{\psi}_k$ variables, so that the state is now

[4]Such as, "don't jerk the steering wheel too fast", or "keep the speed limit while performing the maneuver", or "minimize the expended energy for actuating the control command".

[5]After all, feedback is the basic principle of cybernetics. Without feedback, there is no control.

$\boldsymbol{x}_k = \left[x_k,\ y_k,\ \psi_k,\ v_k,\ \tilde{y}_k,\ \tilde{\psi}_k\right]^\top$ and the model becomes

$$x_{k+1} = x_k + v_k \cos(\psi_k)\mathrm{d}t \tag{11a}$$
$$y_{k+1} = y_k + v_k \sin(\psi_k)\mathrm{d}t \tag{11b}$$
$$\psi_{k+1} = \psi_k + \frac{v_k}{l_f}\delta_k\mathrm{d}t \tag{11c}$$
$$v_{k+1} = v_k + a_k\mathrm{d}t \tag{11d}$$
$$\tilde{y}_{k+1} = \tilde{y}_k + v_k \sin(\tilde{\psi}_k)\mathrm{d}t \tag{11e}$$
$$\tilde{\psi}_{k+1} = \tilde{\psi}_k + \frac{v_k}{l_f}\delta_k\mathrm{d}t \tag{11f}$$

Let's think about error for a moment. Whenever we talk about error, we implicitly assume that there is some intended ground-truth value from which we departed - the error, therefore, expresses the difference between the desired (intended, reference) value and the actual value. These errors arise, because the world is not perfect - actuators have their limitations, the environment is not completely predictable, there is numerical noise in calculations, communication delays on the sensor network, approximative assumptions in the modeling of the vehicle motion, etc. The reference trajectory is supplied by the path planner module, which we haven't covered yet in the Term 3. For the purposes of the MPC project, the reference trajectory is created by fitting a polynomial to the supplied waypoints.

Let's define the cross-track error $\tilde{y}_k = y_k - y_k^r$ and the heading error $\tilde{\psi}_k = \psi_k - \psi_k^r$, where y_k^r and ψ_k^r are the reference values. Fig. 3 shows that the reference heading at any time step ψ_k^r is the tangential angle at the point x_k of the reference trajectory $f(x_k)$. We know that the slope of the tangential line at x_k is given by the derivative evaluated at that point, so $f'(x_k) = \tan(\psi_k^r)$ and thus the reference heading can be expressed as $\psi_k^r = \arctan(f'(x_k))$. The derivative $f'(x_k)$ depends on which model we use for fitting the reference trajectory (when f is polynomial, derivatives are trivially obtained.)

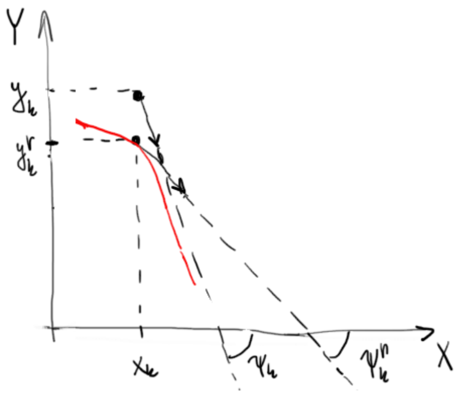

Figure 3: Cross-track error $\tilde{y}_k = y_k - y_k^r$ and heading error $\tilde{\psi}_k = \psi_k - \psi_k^r$. The reference track curve fit in red.

Recall that, the state $\boldsymbol{x}_k$, the control action $\boldsymbol{u}_k$ and the reference input $\boldsymbol{r}_k$

are defined as

$$\boldsymbol{x}_k = \left[x_k,\ y_k,\ \psi_k,\ v_k,\ \tilde{y}_k,\ \tilde{\psi}_k\right]^\top \tag{12}$$

$$\boldsymbol{u}_k = \left[a_k,\ \delta_k\right]^\top \tag{13}$$

$$\boldsymbol{r}_k = \left[y^r_k,\ \psi^r_k\right]^\top \tag{14}$$

Let the state sequence for N time steps forward be denoted as

$$\boldsymbol{x}_{k:k+N} \triangleq \left[\boldsymbol{x}_k,\ \boldsymbol{x}_{k+1},\ \ldots,\ \boldsymbol{x}_{k+N}\right] \tag{15}$$

and analogously for sequence control actions $\boldsymbol{u}_{k:k+N}$ and reference inputs $\boldsymbol{r}_{k:k+N}$. The model predictive control computes the optimal sequence of control actions for N steps ahead by minimizing the cost function[6]

$$u_{k:k+N} = \arg\min_{u_{k:k+N}} J(\boldsymbol{u}_{k:k+N};\, \boldsymbol{x}_{k:k+N},\ \boldsymbol{r}_{k:k+N},\, N), \tag{16}$$

subject to constraints[7] given by the kinematic bicycle model (plant model) in eqs. (11a) to (11f) and bounds[8] on the actuation variables

$$-1 \leq \alpha \leq 1, \qquad -25^\circ \leq \delta_f \leq 25^\circ. \tag{17}$$

The hyper-parameter N is referred to as the prediction horizon.

An example cost function used in MPC with our kinematic bicycle model might look something like this

$$J(\boldsymbol{u}_{k:k+N};\, \boldsymbol{x}_{k:k+N},\, \boldsymbol{r}_{k:k+N},\, N) = \sum_{k=1}^{N} (\tilde{y}_k)^2. \tag{18}$$

Minimizing this functional yields control actions, for which the cross-track error squared is minimal. As you can probably guess, the heading error $\tilde{\psi}_k$ is not reflected in the cost function, which means we can't expect it to be small after minimization. Besides that, there are other desirable properties that we would like the resulting control sequence to have, such as minimum energy expenditure, smooth steering etc. All of these, and more, can be incorporated by tinkering with the cost function. A more realistic cost might look something like this

$$J(\boldsymbol{u}_{k:k+N};\, \boldsymbol{x}_{k:k+N},\, \boldsymbol{r}_{k:k+N},\, N) = \sum_{k=1}^{N} (\tilde{y}_k)^2 + (\tilde{\psi}_k)^2 + (\delta_k)^2 + \sum_{k=1}^{N-1} (\delta_{k+1} - \delta_k)^2. \tag{19}$$

This accounts for the energy expended on the steering δ_k^2 as well as smooth steering transitions $(\delta_{k+1} - \delta_k)^2$. Fig. 4 outlines the model predictive control loop and role of the variables involved.

[6]Since J depends on the sequence of variables (which is a discretized function anyway), it is, strictly speaking, a *functional*.

[7]Constraints are limits on *relationships* between values.

[8]Bounds are absolute limits on the range of values.

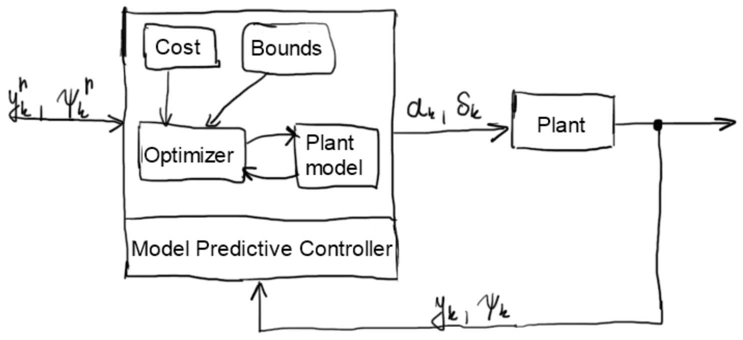

Figure 4: Model predictive control loop.

4 Implementation

In my implementation of the MPC, I'm using the cost function

$$J(\boldsymbol{u}_{k:k+N}) = \sum_{k=1}^{N} w_{cte}\tilde{y}_k^2 + w_{he}\tilde{\psi}_k^2 + w_v\tilde{v}_k^2 + w_\delta\delta_k^2 + w_a a_k^2 + \sum_{k=1}^{N-1} w_{\delta t}(\delta_{k+1} - \delta_k)^2 + w_a(a_{k+1} - a_k)^2, \tag{20}$$

where $\tilde{v}_k = (v_k - v^r)$ and $v^r = 100\,\text{mph}$ is the reference speed. The weights in the cost function achieving the desired control behavior, were determined experimentally to be

$$w_{cte} = 1500,\ w_{he} = 2000,\ w_v = 2,\ w_\delta = 50,\ \ w_a = 5,\ w_{\delta t} = 2000,\ w_{\delta a} = 10. \tag{21}$$

During the experiments, I initially stuck with prediction horizon $N = 10$, but eventually went down to $N = 7$. As that didn't give satisfactory car behavior I went up to $N = 8$, which exhibited the best control behavior at the chosen reference speed and seemed like a good trade-off between the computational load and the quality of the result. I saw no need to fiddle with the discretization period, so I stuck to $\mathrm{d}t = 0.1\,\text{s}$.

The reference car position y^r and heading ψ^r are determined from the 3rd-degree polynomial fit to the waypoints. The polynomial model and it's derivative are given by

$$f(x_k) = a_3x_k^3 + a_2x_k^2 + a_1x_k + a_0, \tag{22}$$

$$f'(x_k) = 3a_3x_k^2 + 2a_2x_k + a_1. \tag{23}$$

Before fitting the model, the current state is projected 100 ms into the future to account for the actuation delay and then it is transformed into the car's coordinate system (where the car's position coordinates and the heading are always zero). With the fitted polynomial, we can use the reasoning from the previous sections, to determine the reference position and heading as

$$y_k^r = f(0) = a_0, \tag{24}$$

$$\psi_k^r = \arctan(f'(0)) = \arctan(a_1), \tag{25}$$

and the cross-track error and heading error thus become

$$\tilde{y}_k = y_k - y_k^r = -a_0, \tag{26}$$

$$\tilde{\psi}_k = \psi_k - \psi_k^r = -\arctan(a_1). \tag{27}$$

So, at every time step, the optimization is started with the initial state (which is part of the constraints, given by the kinematic bicycle model)

$$\boldsymbol{x}_k = \left[0,\ 0,\ 0,\ v,\ \tilde{y}_k,\ \tilde{\psi}_k\right]. \tag{28}$$

References

J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, June 2015. doi: 10.1109/IVS.2015.7225830.