Model Predictive Control notes

Philippe Weingertner

August 15, 2018

Contents

1 Motion Control

Motion Control deals with the last stage of an autonomous driving pipeline: the control module. The input to the control module will be provided by the output of the path planning module via a set of waypoints to follow as close as possible. The control module will have to provide the actuators commands (in our case steering angle and throttling; acceleration or deceleration) so that the automated driving comply with a set of rules:

- follow the planned waypoints as close as possible
- drives smoothly
- try to adjust the speed: as fast as a configurable reference when possible and driving more slowly during curves

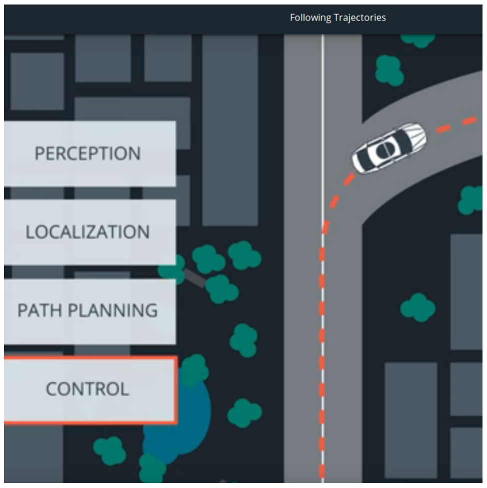


Figure 1: Autonomous Driving pipeline

2 Non linear optimization under constraints

2.1 Definition

In its most generic form we are dealing with the following problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & lower_i \leq f_i(x) \leq upper_i, \; i = 1, \ldots, m. \end{aligned}$$

Note that by setting $lower_i = upper_i$ we can define constraints as equalities as well.

2.2 Example

$$\begin{array}{ll} \text{minimize} & x_1 * x_4 * (x_1 + x_2 + x_3) + x_3 \\ \text{subject to} & x_1 * x_2 * x_3 * x_4 \geq 25 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40 \\ & 1 \leq x_1, x_2, x_3, x_4 \leq 5 \end{array}$$

2.3 Solving with ipopt

ipopt and cppad are used to solve non-linear minimization problems. ipopt requires the computation of first order (Jacobians) and 2nd order derivatives (Hessians). These derivatives will be computed automatically thanks to cppad: providing automatic differentiation services.
The previous example is solved with ipopt and CppAD here: `https://www.coin-or.org/CppAD/Doc/ipopt_solve_get_started.cpp.htm`

3 Vehicle Models

3.1 Dynamic vs Kinematic Models

3.2 Kinematic Model

3.2.1 State

The state variables are the following:

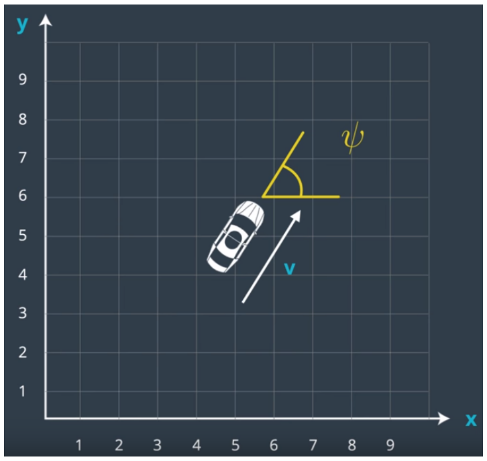


Figure 2: Model state

- x: x position
- y: y position
- ψ: angle between speed vector and x-axis
- v: speed vector

3.2.2 Deriving the kinematic model

Our state vector is
$$S_t = [x_t, y_t, \psi_t, v_t]$$
We derive an approximation model, kinematic, relating S_{t+1} and S_t. The smaller the dt the more accurate the model.

Linear movement approximation: assuming during dt that v_t and ψ_t are constant:
$$x_{t+1} = x_t + v_t * \cos(\psi_t) * dt$$
$$y_{t+1} = y_t + v_t * \sin(\psi_t) * dt$$
Rotational movement approximation: assuming during dt that v_t and steering angle δ_t are constant:
$$M_{t+1}M_t = \rho * (\psi_{t+1} - \psi_t) = v_t * dt$$
$$\tan(\delta_t) = L_f/\rho$$
So we have:
$$\psi_{t+1} = \psi_t + (v_t/\rho) * dt$$
$$\psi_{t+1} = \psi_t + (v_t/L_f) * \tan(\delta_t) * dt$$
Note that for small δ_t we have $\tan(\delta_t) \approx \delta_t$

Speed update: assuming during dt that a_t is constant:
$$v_{t+1} = v_t + a_t * dt$$

So to summarize our kinematic model is:
$$x_{t+1} = x_t + v_t * \cos(\psi_t) * dt$$

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

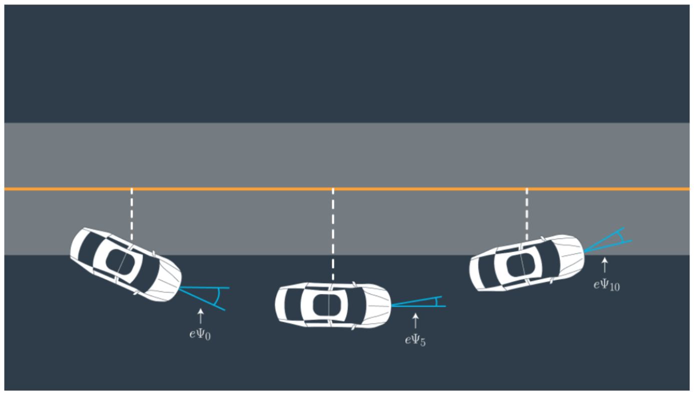
$$\psi_{t+1} = \psi_t + (v_t / L_f) * \tan(\delta_t) * dt$$

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

The state vector is $S_t = [x_t, y_t, \psi_t, v_t]$.
The actuator command $A_t = [a_t, \delta_t]$ defines a **constraint** between S_{t+1} and S_t.

3.2.3 Errors

The errors variables are the following:



The dashed white line is the cross track error.

Figure 3: Model errors

- *cte*: cross track error. It corresponds to distance of vehicle from the planned trajectory (as planned by path planning module)
- $e\Psi$: psie error is the angle difference of the vehicle trajectory with the planned trajectory (as planned by path planning module)

The new state vector is $[x_t, y_t, \psi_t, v_t, cte_t, e\Psi_t]$**.**

3.2.4 Kinematic Model

3.3 Dynamic Models

Forces, Slip Angle, Slip ratio and Tire Models

4 Model Predictive Control

MPC reframes the task of following a trajectory as an optimization problem. The solution to the optimization problem is the optimal trajectory.

MPC involves simulating different actuator inputs, predicting the resulting trajectory and minimizing a set of constraints (or cost functions).

Input: a reference trajectory we want to follow

Constraints:

- Vehicle Model
- Comfort

Output: actuator commands (steering, throttling, braking ...)

Once we found the lowest cost trajectory, we implement the very first set of actuation commands. Then we throw away the rest of the trajectory we calculated. Instead of using the old trajectory we predicted, we take our new state and use that to calculate a new optimal trajectory. In that sense, we are constantly calculating inputs over a future horizon. That's why this approach is also called Receding Horizon Control. We constantly reevaluate the trajectory because our vehicle model is not perfect and the next predicted (or planned) state may (slightly...) differ with our prediction (in the sense of a consequence of a command sent).

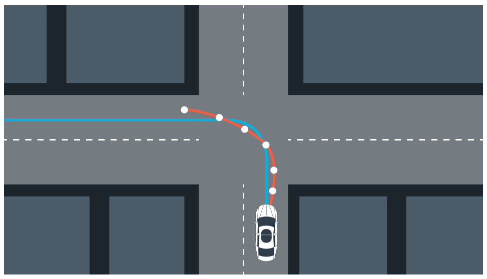
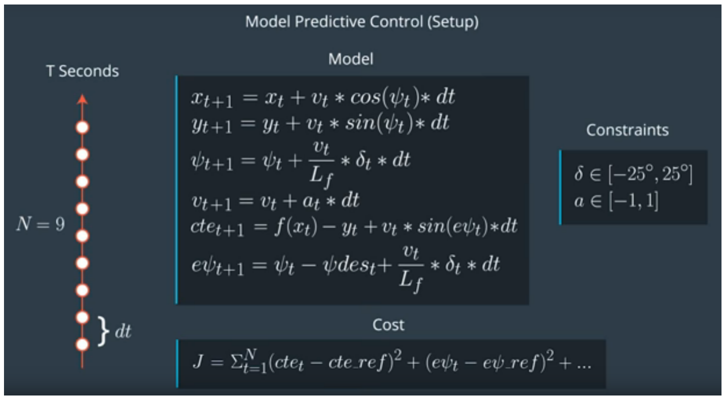


Figure 4: Minimization problem

4.1 Optimization under constraints: cost functions

4.2 Timsestep length and Elapsed duration

N=10 and dt=100 ms are used so that we are working on 1 second of data. This is a trade-off: we need enough data visibility to ensure a good prediction, but we also have to limit the amount of computation. In general, smaller dt gives better accuracy, but that will require higher N for given horizon (N*dt). However, increasing N will result in longer computational time which increases the latency. The most common choice of values is N=10 and dt=0.1 but anything between N=20, dt=0.05 should work.



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

$$
\delta \in [-25^\circ, 25^\circ] \qquad a \in [-1, 1]
$$

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

Figure 5: Solver setup with N*dt time horizon

4.3 Latency handling

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.
Thus, MPC can deal with latency much more effectively, by explicitly taking it into account, than a PID controller.

4.4 MPC Solver algorithm

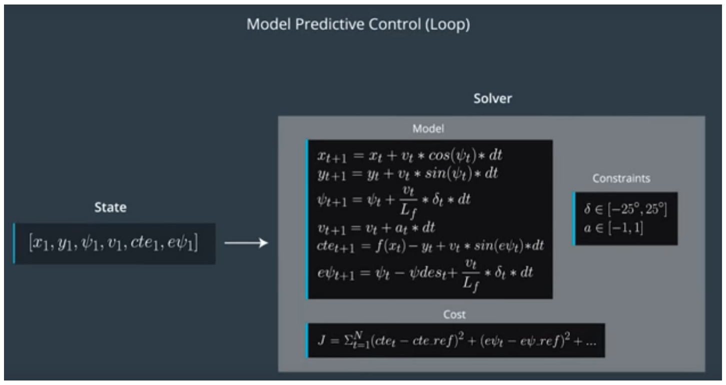

Model Predictive Control (Loop)
Solver
Model
$x_{t+1} = x_t + v_t * cos(\psi_t) * dt$
$y_{t+1} = y_t + v_t * sin(\psi_t) * dt$
$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} * \delta_t * dt$
$v_{t+1} = v_t + a_t * dt$
$cte_{t+1} = f(x_t) - y_t + v_t * sin(e\psi_t) * dt$
$e\psi_{t+1} = \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt$
Constraints
$\delta \in [-25°, 25°]$
$a \in [-1, 1]$
State
$[x_1, y_1, \psi_1, v_1, cte_1, e\psi_1]$
Cost
$J = \Sigma_{t=1}^{N}(cte_t - cte\_ref)^2 + (e\psi_t - e\psi\_ref)^2 + ...$

Figure 6: Solver input

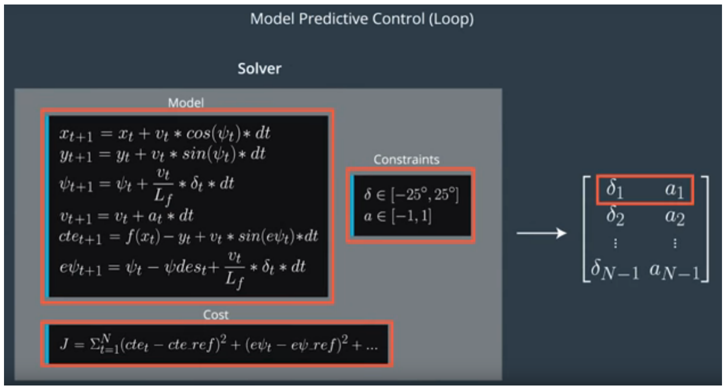

Model Predictive Control (Loop)
Solver
Model
$x_{t+1} = x_t + v_t * cos(\psi_t) * dt$
$y_{t+1} = y_t + v_t * sin(\psi_t) * dt$
$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} * \delta_t * dt$
$v_{t+1} = v_t + a_t * dt$
$cte_{t+1} = f(x_t) - y_t + v_t * sin(e\psi_t) * dt$
$e\psi_{t+1} = \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt$
Constraints
$\delta \in [-25°, 25°]$
$a \in [-1, 1]$
Cost
$J = \Sigma_{t=1}^{N}(cte_t - cte\_ref)^2 + (e\psi_t - e\psi\_ref)^2 + ...$
$\begin{bmatrix} \delta_1 & a_1 \\ \delta_2 & a_2 \\ \vdots & \vdots \\ \delta_{N-1} & a_{N-1} \end{bmatrix}$

Figure 7: Solver output

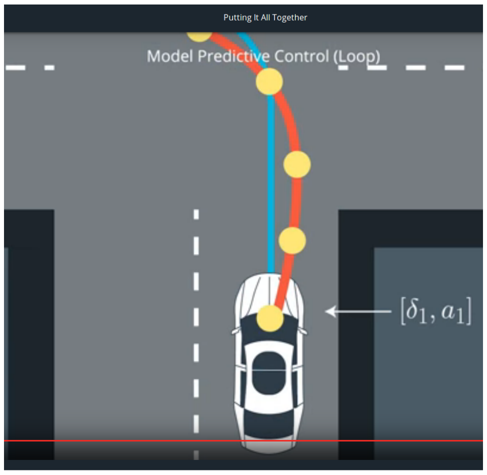


Figure 8: Solver actuator commands