CPM Vehicle Parameter Identification and Model Predictive Control

Janis Maczijewski

July 5, 2019

1 Vehicle Dynamics Model

This is an end-to-end, grey-box model for the vehicle dynamics. The model parameters are not measured directly, but optimized to best fit the vehicle behavior.

$$\boldsymbol{x} = [p_x, p_y, \psi, v]$$
$$\boldsymbol{u} = [f, \delta, V]$$

p_x	IPS x-position
p_y	IPS y-position
ψ	IPS yaw angle
v	Odometer Speed
f	Dimensionless motor command
δ	Dimensionless steering command
V	Battery voltage

$$\dot{p}_x = p_1 \cdot v \cdot (1 + p_2 \cdot (\delta + p_9)^2) \cdot \cos(\psi + p_3 \cdot (\delta + p_9) + p_{10}) \tag{1}$$
$$\dot{p}_y = p_1 \cdot v \cdot (1 + p_2 \cdot (\delta + p_9)^2) \cdot \sin(\psi + p_3 \cdot (\delta + p_9) + p_{10}) \tag{2}$$
$$\dot{\psi} = p_4 \cdot v \cdot (\delta + p_9) \tag{3}$$
$$\dot{v} = p_5 \cdot v + (p_6 + p_7 \cdot V) \cdot \text{sign}(f) \cdot |f|^{p_8} \tag{4}$$

This is a kinematic bicycle model with some added terms to account for various errors.

- p_1: Compensate calibration error between IPS speed and odometer speed.
- $(1 + p_2 \cdot (\delta + p_9)^2)$: Compensate for speed differences due to different reference points between the IPS and odometer. The formulation is simplified with a second-order Taylor approximation.
- p_3: Side slip angle (Schwimmwinkel) due to steering.
- p_{10}: IPS Yaw calibration error.
- p_4: Unit conversion for the steering state.
- p_5: Speed low pass (PT1).
- p_6, p_7: Motor strength depends on the battery voltage.
- p_8: Compensate non-linear steady-state speed.
- p_9: Steering misalignment correction.

2 Model Discretization

The model is discretized with the explicit Euler method, as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta t \cdot f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}) \tag{5}$$

This discretization is chosen for its simplicity and computational efficiency.

TODO justification inaccuracy: small timestep, and discretization is included in identification.

3 Parameter Identification

Optimal parameter estimation problem for the vehicle dynamics. The optimization tries to find a set of model parameters, that best explain/reproduce the experiment data.

$$\begin{aligned}
\underset{\boldsymbol{x}_k^j, \boldsymbol{p}}{\text{minimize}} \quad & \sum_{j=1}^{n_{experiments}} \sum_{k=1}^{n_{timesteps}} E(\boldsymbol{x}_k^j - \hat{\boldsymbol{x}}_k^j) && (6)\\
\text{subject to} \quad & \boldsymbol{x}_{k+1}^j = \boldsymbol{x}_k^j + \Delta t \cdot f(\boldsymbol{x}_k^j, \hat{\boldsymbol{u}}_k^j, \boldsymbol{p}) && (7)\\
& k = 1..(n_{timesteps} - 1) && (8)\\
& j = 1..n_{experiments} && (9)\\
& && (10)
\end{aligned}$$

$\hat{\boldsymbol{x}}_k^j$	Measured States
$\hat{\boldsymbol{u}}_k^j$	Measured Inputs
f	Vehicle dynamics model
$\boldsymbol{p}$	Model parameters
Δt	Constant timestep $0.02s$
E	Error penalty function

Error penalty E: Weighted quadratic error with model specific extensions. The yaw error function has a period of 2π, so that a full rotation does not count as an error. This is done using $\sin(\Delta\psi/2)^2$.

Delays: This kind of optimization problem is not well suited for identifying the delay times (Totzeiten). The delays are solved in an outer loop. The delay is guessed/assumed and the measurement data is modified by appropriately shifting it in the time index k. This optimization problem is solved many times for combinations of delay times. The delays that create the lowest objective value are taken as the solution.

4 Model Predictive Control

The MPC uses the identified model to calculate control inputs in real time. The MPC must run in an environment with narrow computational constraints. It must run on a Raspberry Pi Zero W and in less than 10ms per time step. This results in a computational budget of roughly 100000 to 500000 double precision floating point operations per time step.

4.1 Optimization Problem and Optimizer

The MPC is formulated as a (mostly) unconstrained minimization problem, with a box constraint:

$$\underset{z \in \mathbb{R}^n}{\text{minimize}} \quad J(z) \tag{11}$$

$$\text{subject to} \quad -1 \leq z_i \leq 1 \tag{12}$$

The optimization problem is solved using a simple and lightweight method, the gradient descent method with momentum. The method works by iterating the following two equations:

$$m^{(j)} := \beta m^{(j-1)} - \nabla J(z^{(j-1)})$$
$$z^{(j)} := \text{clip}(z^{(j-1)} + \alpha m^{(j)})$$

where $z^{(j)}$ is the approximate solution, which is improved in each iteration, $m^{(j)}$ is the momentum term, $\alpha = 0.4$ and $\beta = 0.6$ are constants, $\text{clip}(\cdot)$ applies the following function element-wise $\min(1, \max(-1, x_i))$ and $\nabla J(z^{(j-1)})$ is the gradient of the objective.

A constant number of iterations $j = 1...N$ are executed. This guarantees a constant computation time, but not the convergence to the solution. The approximate solution $z^{(N)}$ is however sufficient in practice.

4.2 Trajectory Tracking Problem

The goal of the MPC is to make the vehicle follow a given reference trajectory. The reference trajectory is given as a Cubic Hermite spline function $r(t) = [p_{x,ref}, p_{y,ref}]$. This function is evaluated on an appropriate time grid t_k to give the discrete reference trajectory $[p_{x,ref,k}, p_{y,ref,k}]$ where $k = 1...H_p$. The MPC objective is defined as follows:

$$\begin{aligned} J = &\sum_{k=1}^{H_p} \left[(p_{x,k} - p_{x,ref,k})^2 + (p_{y,k} - p_{y,ref,k})^2 \right] \\ &+ 0.5 \sum_{k=1}^{H_u} (f_k - f_{k-1})^2 \\ &+ 0.01 \sum_{m=1}^{H_u} (\delta_m - \delta_{m-1})^2 \end{aligned}$$

The predicted states are calculated explicitly and recursively. This is known as a single shooting method.

$$\begin{bmatrix} p_{x,k+1} \\ p_{y,k+1} \\ \psi_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} p_{x,k} \\ p_{y,k} \\ \psi_k \\ v_k \end{bmatrix} + \Delta t_{MPC} \cdot f \left(\begin{bmatrix} p_{x,k} \\ p_{y,k} \\ \psi_k \\ v_k \end{bmatrix}, \begin{bmatrix} f_{\boldsymbol{m}_k} \\ \delta_{\boldsymbol{m}_k} \\ V_{\boldsymbol{m}_k} \end{bmatrix}, \boldsymbol{p} \right), \quad k = 0...(H_p - 1)$$

where f is the identified model from section (ref).

$\boldsymbol{m}$ is an index vector that allows reuse of input vectors. Current implementation $\boldsymbol{m} = [1, 1, 2, 2, 3, 3]$.

$H_p = 6$, $H_u = 3$

The MPC uses a longer discretization time step $\Delta t_{MPC} = 0.05s$.

The inputs for the trajectory tracking problem are:

- $[p_{x,0}, p_{y,0}, \psi_0, v_0]$: The measured state with delay compensation

- $[f_0, \delta_0]$: The previous command
- $[p_{x,ref,k}, p_{y,ref,k}]$: The discrete reference trajectory
- V_m: The measured battery voltage
- $\boldsymbol{p}$: The model parameters

The variables for the trajectory tracking problem are: $z = [f_1, ..., f_{H_u}, \delta_1, ..., \delta_{H_u}]$.

4.3 Delay Compensation

The processes of measuring the state, running the MPC controller and applying the new command introduces a delay in the control loop. This delay is compensated by performing a short simulation after measuring the state and before running the MPC. Thus, the MPC will optimize the inputs for a future state, which matches the time by which the new commands take effect. This simulation requires some command inputs. These are the MPC commands from the previous timesteps. The simulation runs for 3 samples, or 60ms.

4.4 CasADi and Code Generation

The trajectory tracking problem and the optimizer are implemented symbolically with Matlab and CasADi. CasADi's code generator for C is used to obtain an efficient implementation for the Raspberry Pi.

4.5 Finetuning

The controller implementation collects statistical data about the trajectory tracking errors in real time.

TODO play with all the MPC hyperparameters to minimize the tracking error.