Erasmus Thesis Project, 12 ECTS

HJALMAR ENDRE

Agenda

- Project overview and objectives
- Experiments & data
- State estimation
- System identification
- Validation / simulation
- Result

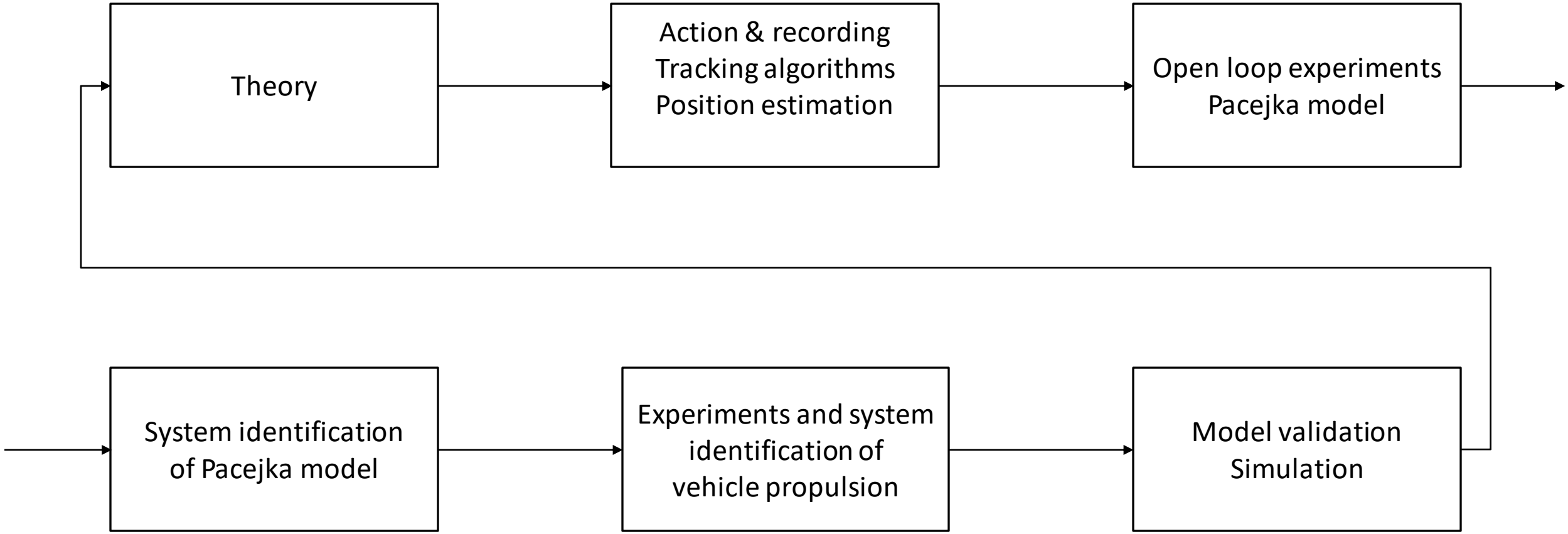
Project overview and objectives

Project status in September

- A complete physical setup
  - Batteries
  - Track
  - RC racer
- Software
  - Camera ↔ computer
  - RC racer ↔ computer
  - Image processing
  - Position detection
- Matlab simulations

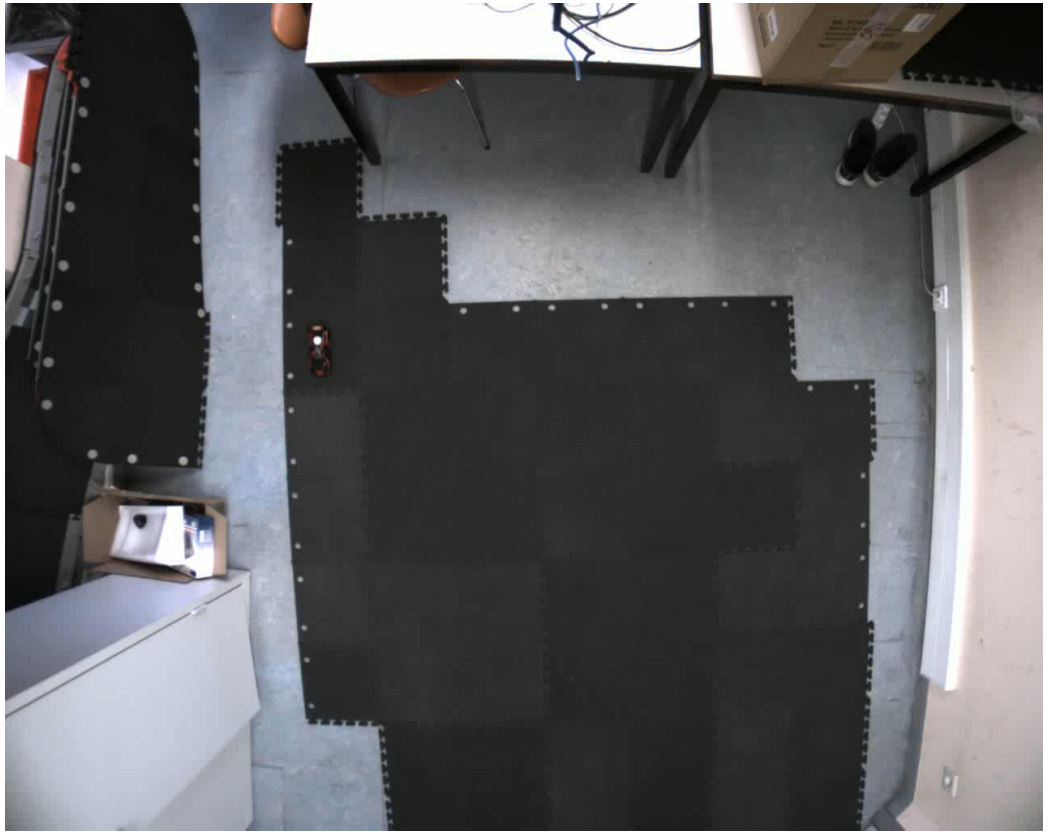
Project goals

- Primary goal: Identify model
- Secondary goal: A working MPC controller


Theory
Action & recording
Tracking algorithms
Position estimation
Open loop experiments
Pacejka model
System identification
of Pacejka model
Experiments and system
identification of
vehicle propulsion
Model validation
Simulation

Experiments

Experiments – “Rampsteer” & “Drive”





Experiments to data

- get_data_object.py
- 2 threads
- get_data(img height, img width, fps, torque applied [0,1], steering applied [-1,1], type of experiment, number of experiments with batteries)

```
d = get_data(1024, 1280, 25, 0.8, 1, 'drive', 5)

d.get_data()
```

- Saves images and timestamp.npy in directory “camera_captures”

Start Camera (Thread 1)

Action (Thread 2)

While action:

If Action == Done:

End Camera

Capture image

Future considerations

1. Too many images are saved into the directory before the RC starts moving
   a) Remove images where speed is about zero
2. Batteries run out quickly
   a) experiments vary between samples
3. Space issues for high torque input
   a) Capture revolutions per second instead of acceleration on mattress
4. Hardware decides FPS
   a) too high FPS -> images is captured too late
5. Real application would have only 1 thread

State estimation

State estimation algorithm

- Constant velocity Kalman filter
- Recursively update filter from measurements
- Measuring x and y position
- Estimating speed
- Acceleration from process noise matrix Q
- Measurement variation = 0.00317
- Process variation = 500^2
- Blocks are repeated diagonally depending on dimension

$$F = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}$$

$$x = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \sigma_r^2$$

$$Q = \begin{bmatrix} \frac{dt^4}{4} & \frac{dt^3}{3} \\ \frac{dt^3}{3} & dt^2 \end{bmatrix} \sigma_v^2$$

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\hat{x}_k = F x_{k-1}$$
$$\hat{P}_k = F F^T + Q$$

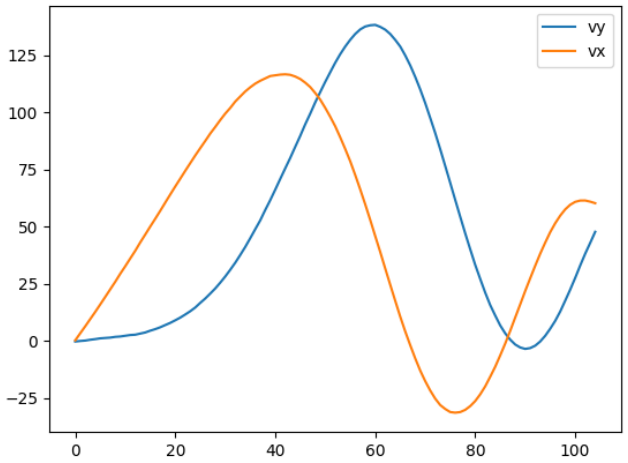
$$S = H \hat{P}_k H^T + R$$

$$K = (\hat{P}_k H^T) S^{-1}$$

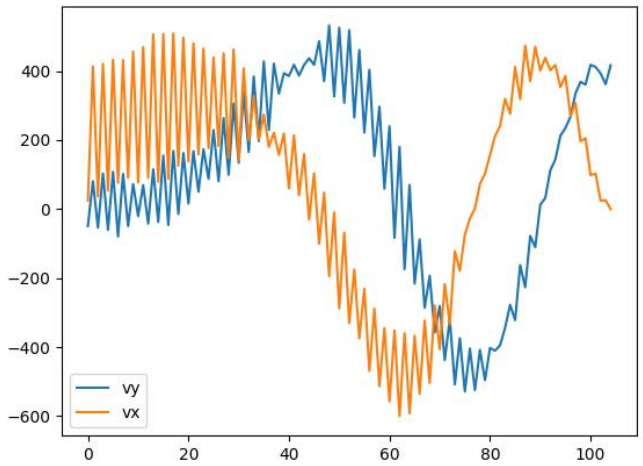
$$y = z - H \hat{x}_k$$

$$x_k = \hat{x}_k + Ky$$
$$P_k = (I - (KH))\hat{P}_k$$

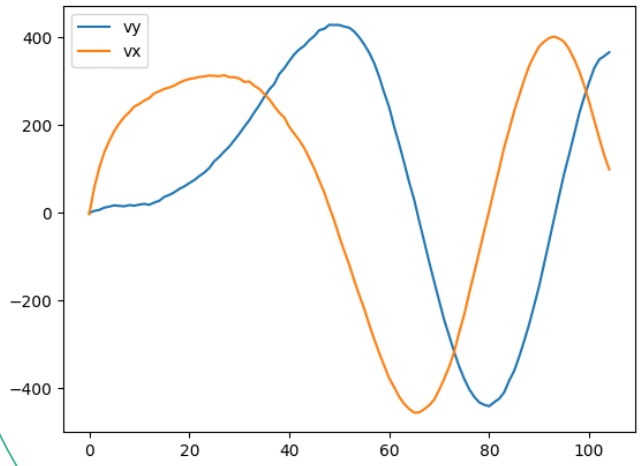
Sigma^2 = 100



Sigma^2 = 10000

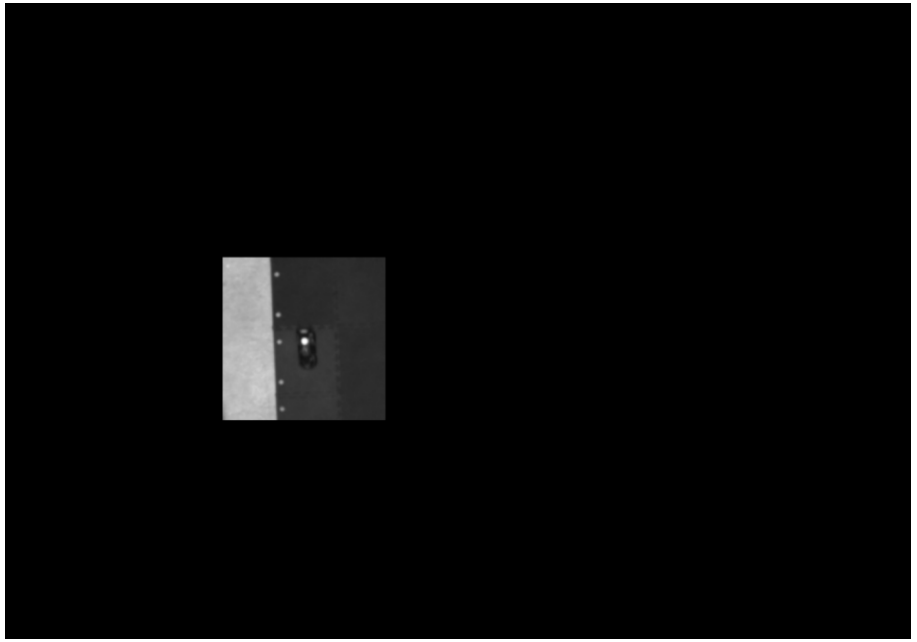


Sigma^2 = 500



Position detection

- Position of RC racer is boxed in, based on prediction
  - Greatly increases speed
  - Reduces white markers from environment
- Mean of white pixels
- Design parameters: Threshold, prediction, mask, size of box
- Starting threshold high, adapts if not enough markers



Assigning candidates to measurement update

- One white dot on car:
  - Error function using Euclidian distance from prediction vector and candidate.
- Three white dots on car:
  - Still comparing prediction vector and candidates
  - Iterate to find best combination
  - Factorial of number of candidates – most times 6 iterations



find_pos_object.py

- Main script
- Usage: position_estimation(num_dots, images directory, mask, plot)

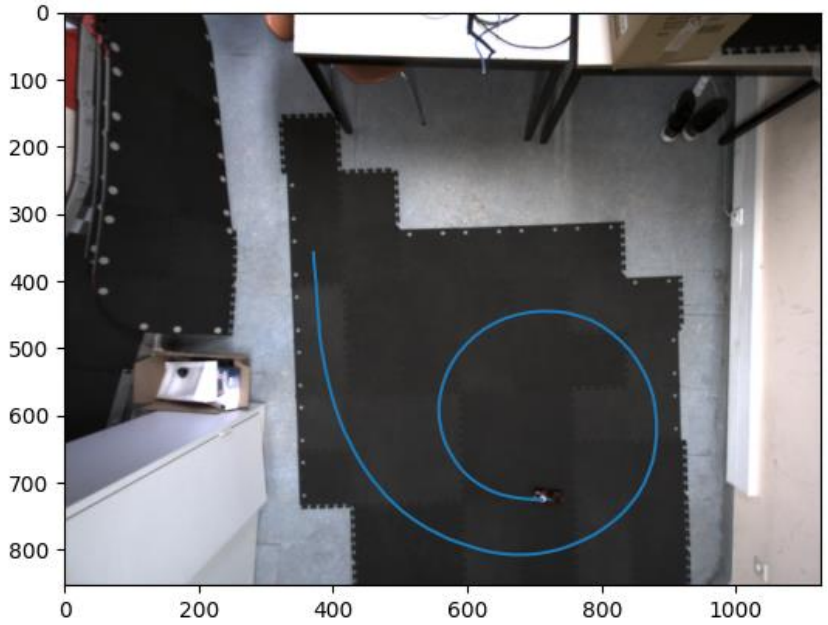
```
p = position_estimation(1,
"camera_captures/rampsteer/D07L",
'bg3_mask_sysid.png', True)

p.find_trajectory()
```

- Estimates are converted pixel to meter with factor 0.002745
- Saves state estimation in states.npy

```
params = CameraParams()
detector = PositionDetector(mask)
images = load_images(params)
for image in images
    if first_image
        pos = choose_startpos(img)
        KalmanInit(pos)
    prediction = update_filters()
    new_measurement(detector, img, prediction)
save_states()
```

Future considerations

- Make the choice of starting position automatic
  - Should be easy with the mask
- Try other state estimation algorithms
  - Constant acceleration?
- Reduce/increase size of box with the help of the uncertainty matrix
- If the setup is moved to a new room, a new calibration might be necessary



System identification

The model

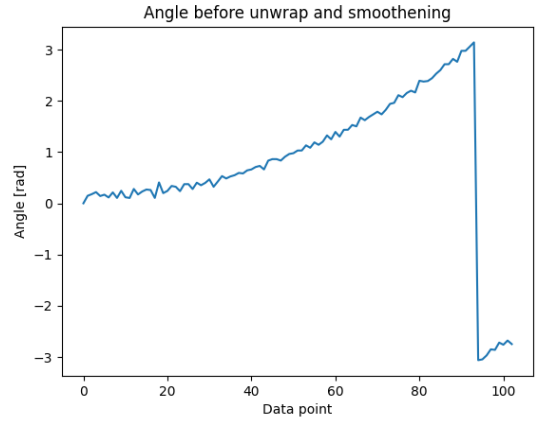
- Model composed of
  - Bicycle model
  - DC-motor equation
  - Newtons second law
  - Pacejka tire models
- Red – slip parameters
- Green – propulsion parameters
- Brown – Known constants
- Blue – System input

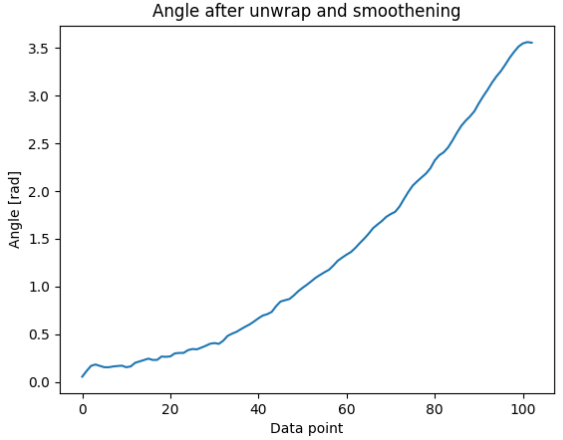
$$\alpha_f = -\arctan\left(\frac{l_f\omega + v_y}{v_x}\right) + \delta$$
$$\alpha_r = \arctan\left(\frac{(l_r\omega - v_y)}{v_x}\right)$$
$$\hat{F}_{fy} = \frac{m v_x \omega l_r}{(l_r + l_f)}\cos(\delta)$$
$$\hat{F}_{ry} = \frac{m v_x \omega l_f}{(l_r + l_f)}$$
$$F_{fy} = \beta_{f3}\sin(\beta_{f2}\arctan(\beta_{f1}\alpha_f))$$
$$F_{ry} = \beta_{r3}\sin(\beta_{r2}\arctan(\beta_{r1}\alpha_r))$$
$$F_{rx} = (C_{m1} - C_{m2}v_x)D - C_{f1} - C_{f2}v_x^2$$
$$C_{f2} = \frac{1}{2}\rho C_d A$$
$$F_{rx} = m a_x$$

$$\dot{x} = v_x\cos(\phi) - v_y\sin(\phi)$$
$$\dot{y} = v_y\sin(\phi) + v_x\cos(\phi)$$
$$\dot{\phi} = \omega$$
$$\dot{v}_x = \frac{1}{m}(F_{rx} - F_{fy}\sin(\delta)) + m\dot{v}_y\omega$$
$$\dot{v}_y = \frac{1}{m}(F_{ry} + F_{fy}\cos(\delta)) - m\dot{v}_x\omega$$
$$\dot{\omega} = \frac{1}{I_z}(F_{fy}l_f\cos(\delta) - F_{ry}l_r)$$

General

- Acceleration and angle (if not measured) is approximated with finite differences
  - Angle approximation is made by arccos or arctan -> angle must be unwrapped (see right)
- Global coordinates in states.npy must be rotated to local coordinates
  - X-axis direction of RC
  - Therefore, angle measurement is crucial for the identification
- Pre-processing
  - Smoothening
  - Peak shaving (not in final solution)
  - Outlier rejection





$$v'_x = v_x \sin(\phi) - y \sin(\phi)$$
$$v'_y = v_x \sin(\phi) + y \cos(\phi)$$

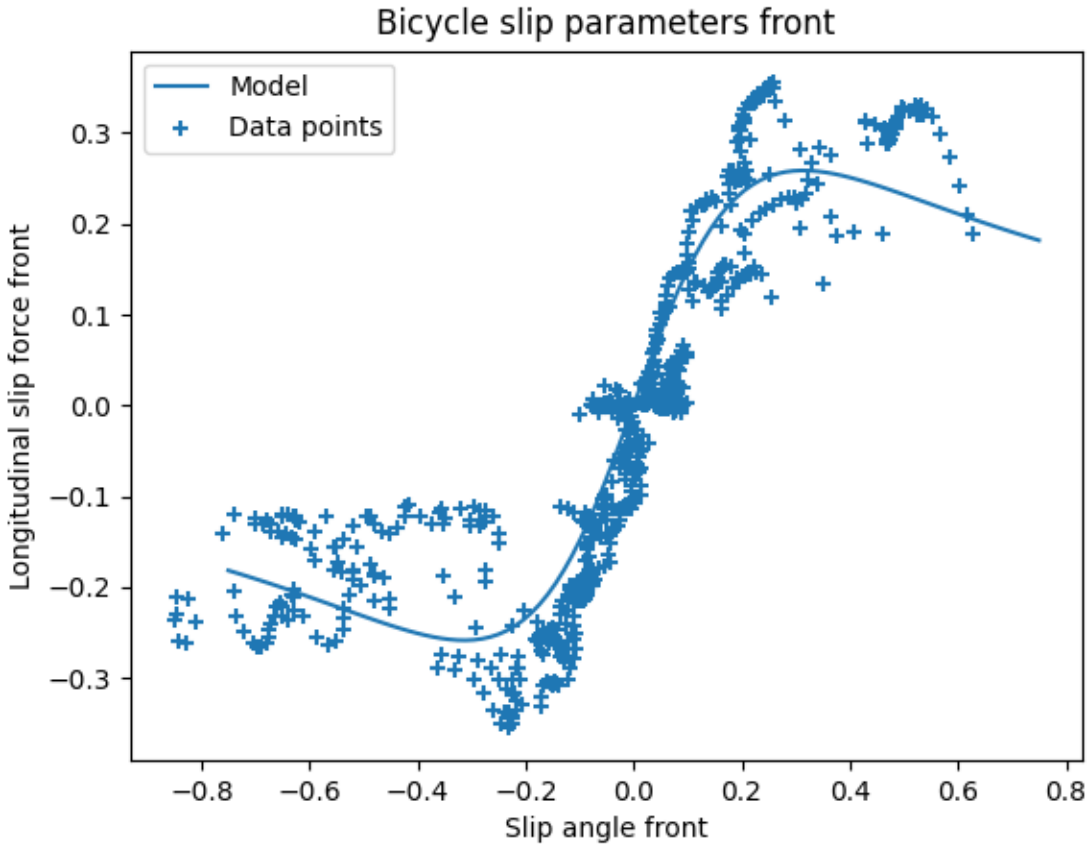
Rampsteer

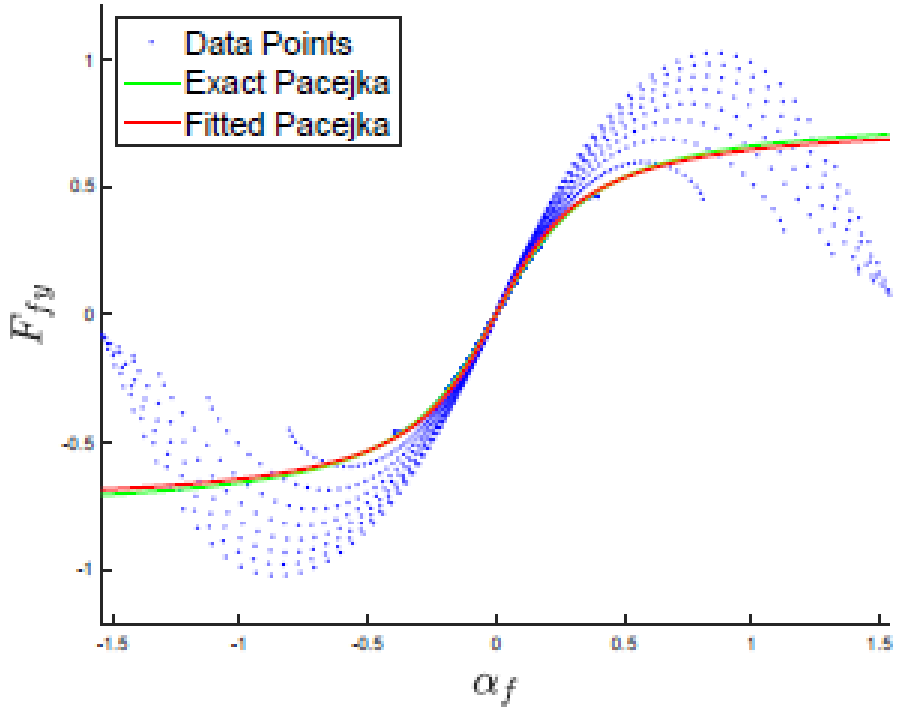
- Calculate slip angles
- Forces calculated with steady state assumption (F_hat)
  - Does the assumption hold?
- Fit Pacejka model with non-linear regression

$$\alpha_f = -\arctan(\frac{l_f\omega+v_y}{v_x}) + \delta$$
$$\alpha_r = \arctan(\frac{(l_r\omega-v_y)}{v_x})$$
$$\hat{F}_{fy} = \frac{mv_x\omega l_r}{(l_r+l_f)}\cos(\delta)$$
$$\hat{F}_{ry} = \frac{mv_x\omega l_f}{(l_r+l_f)}$$
$$F_{fy} = \beta_{f3}\sin(\beta_{f2}\arctan(\beta_{f1}\alpha_f))$$
$$F_{ry} = \beta_{r3}\sin(\beta_{r2}\arctan(\beta_{r1}\alpha_r))$$

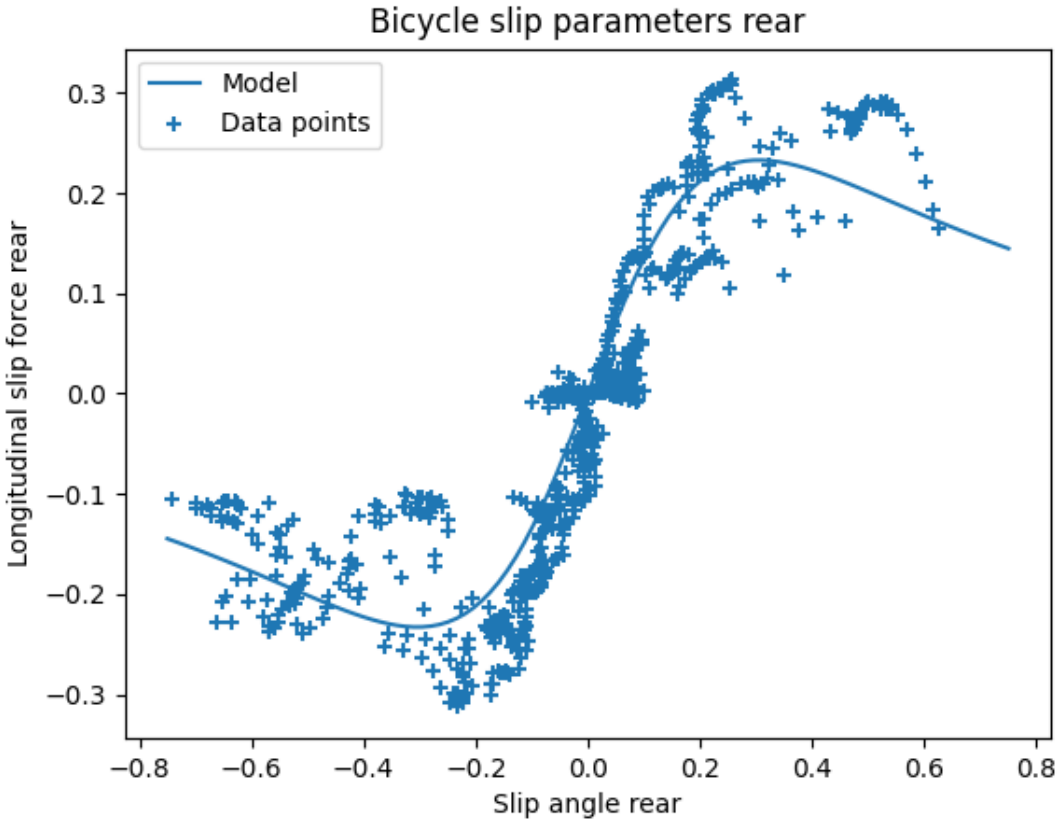
$$\dot{x} = v_x\cos(\phi) - v_y\sin(\phi)$$
$$\dot{y} = v_y\sin(\phi) + v_x\cos(\phi)$$
$$\dot{\phi} = \omega$$
$$\dot{v}_x = \frac{1}{m}(F_{rx} - F_{fy}\sin(\delta)) + m\dot{v}_y\omega$$
$$\dot{v}_y = \frac{1}{m}(F_{ry} + F_{fy}\cos(\delta)) - m\dot{v}_x\omega$$
$$\dot{\omega} = \frac{1}{I_z}(F_{fy}l_f\cos(\delta) - F_{ry}l_r)$$

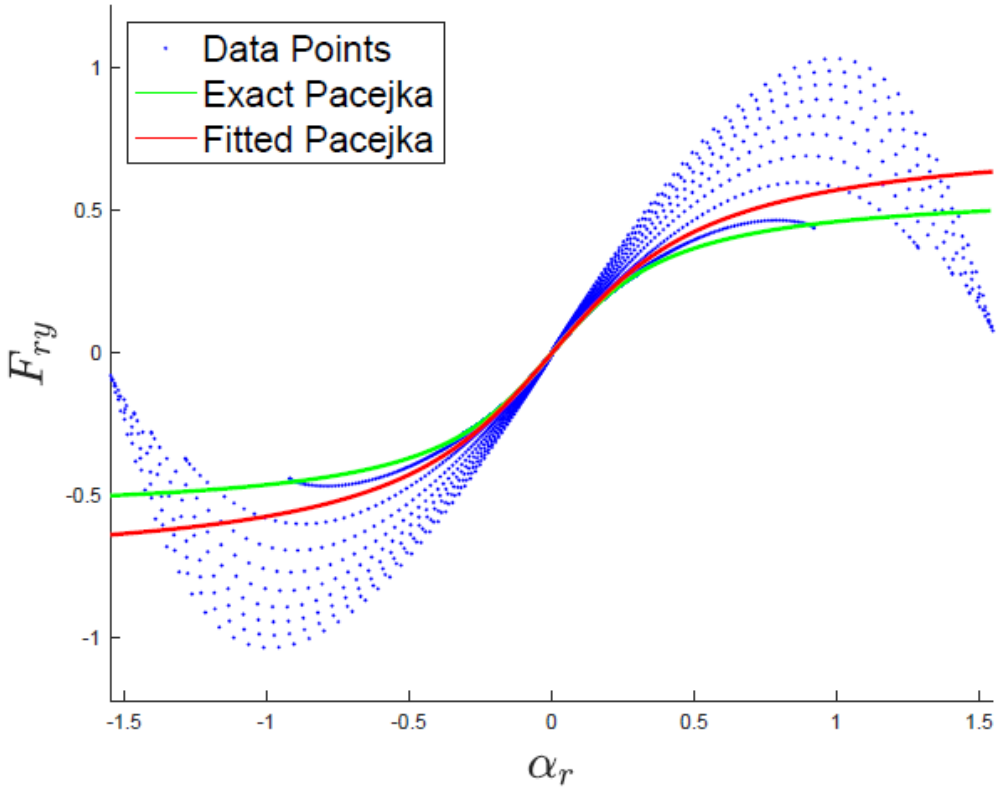
Rampsteer – comparison Matlab simulations



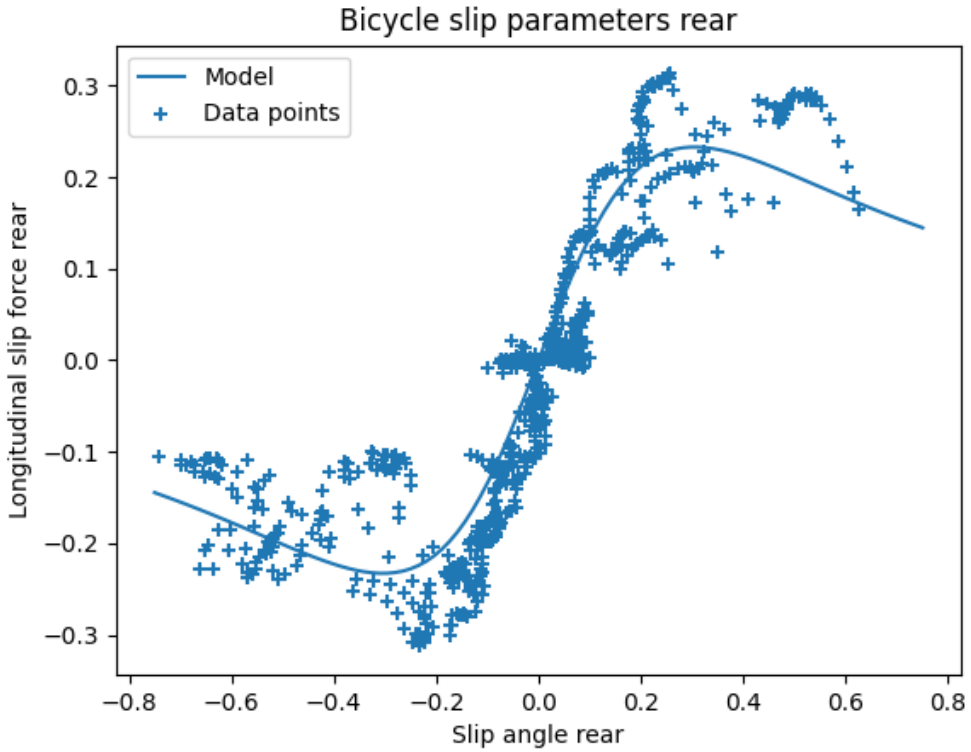


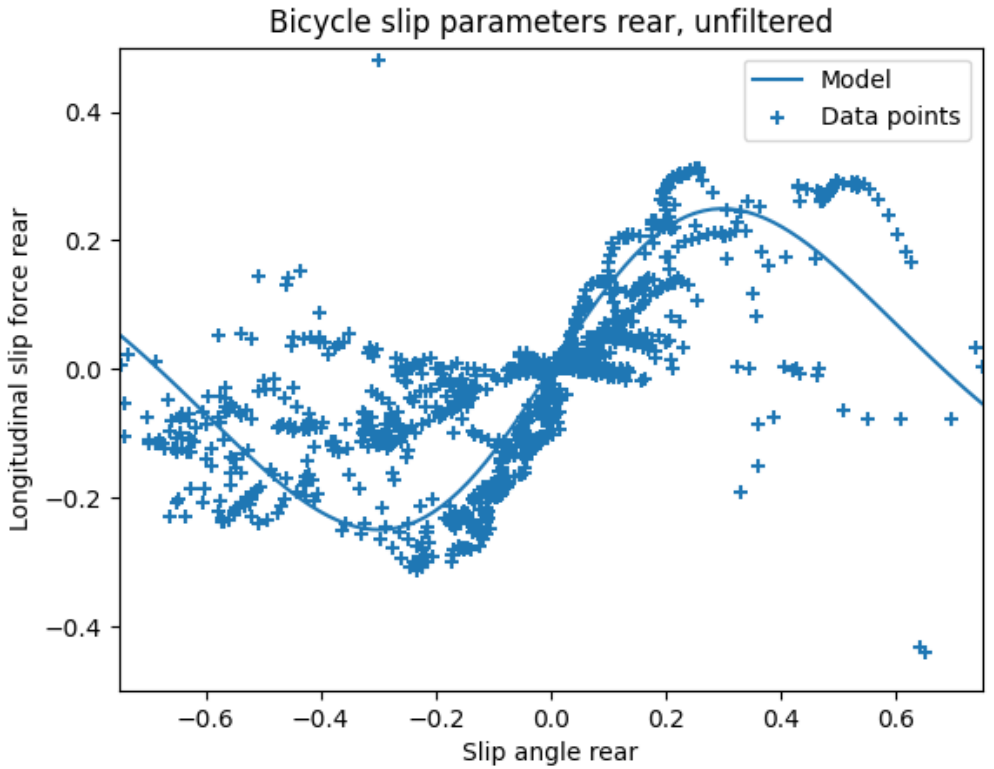
Rampsteer – comparison Matlab simulations



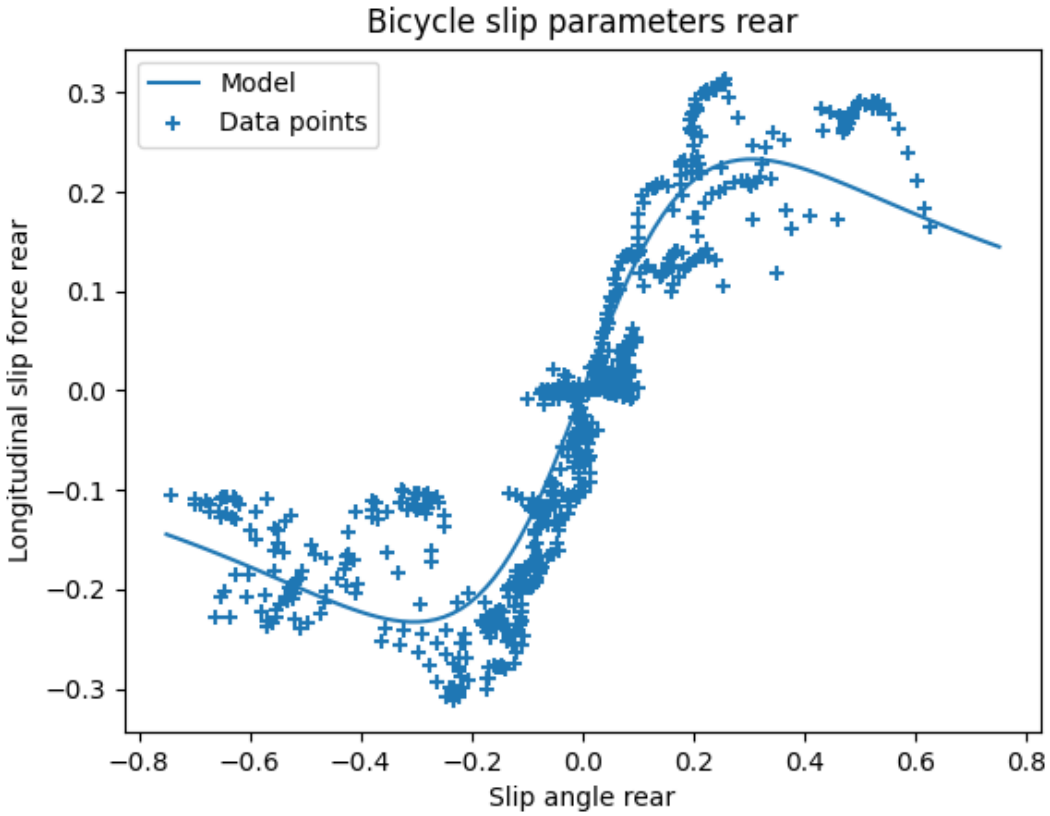


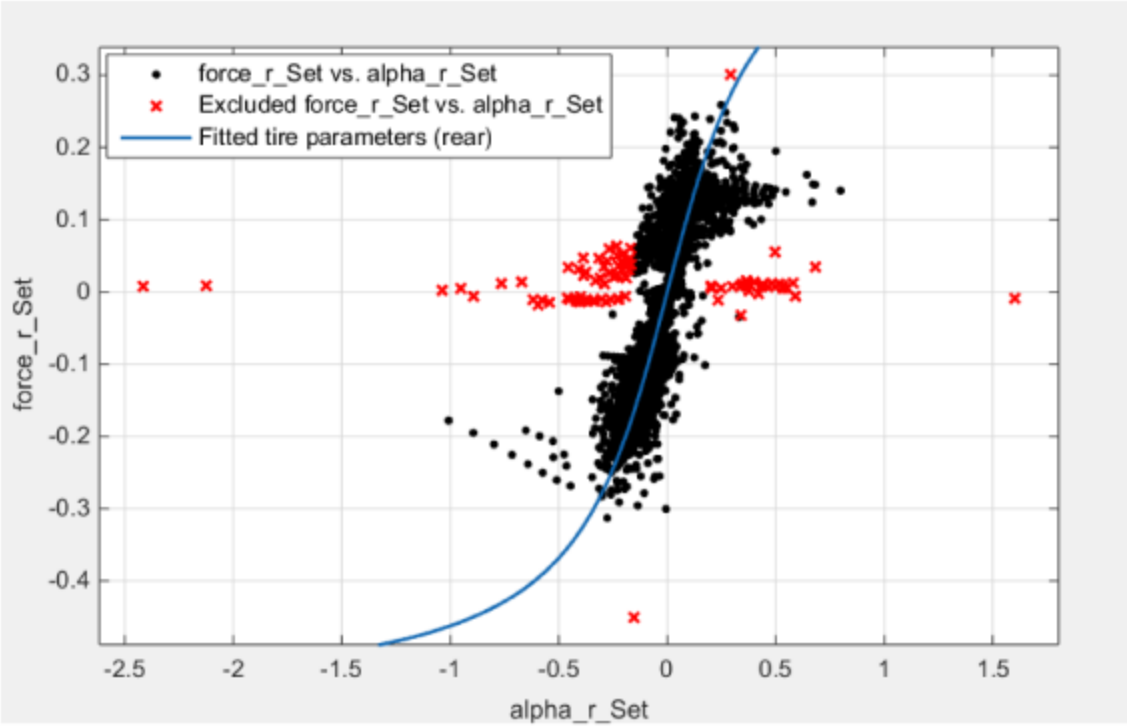
Filtering vs. no filterering





Comparison with Fontana (2013)





Drive

- Use the approximation of acceleration and Newton's second law
- Approximated in three ways
  - Least squares with all parameters and data stacked (best)

$$\begin{bmatrix} D_1 & -D_1 v_{x1} & -1 \\ D_2 & -D_2 v_{x2} & -1 \\ D_3 & -D_3 v_{x3} & -1 \end{bmatrix} \begin{bmatrix} Cm_1 \\ Cm_2 \\ C_{f1} \end{bmatrix} = \begin{bmatrix} ma_x + C_{f2} v_{x1}^2 \\ ma_x + C_{f2} v_{x2}^2 \\ ma_x + C_{f2} v_{x3}^2 \end{bmatrix}$$

  - Least squares on three weighted equations as in Engström et al. (2018)

$$\begin{bmatrix} D_0 & 0 & -1 \\ D & -Dv_x & -1 \\ D & -Dv_x & -1 \end{bmatrix} \begin{bmatrix} Cm_1 \\ Cm_2 \\ C_{f1} \end{bmatrix} = \begin{bmatrix} 0 \\ C_{f2} v_x^2 \\ ma_x + C_{f2} v_x^2 \end{bmatrix}$$

  - Scikit solver with 0.5 * C_m1 = C_f1

$$F_{rx} = (C_{m1} - C_{m2} v_x) D - C_{f1} - C_{f2} v_x^2$$
$$C_{f2} = \tfrac{1}{2} \rho C_d A$$
$$F_{rx} = m a_x$$

$$\dot{x} = v_x \cos(\phi) - v_y \sin(\phi)$$
$$\dot{y} = v_y \sin(\phi) + v_x \cos(\phi)$$
$$\dot{\phi} = \omega$$
$$\dot{v}_x = \tfrac{1}{m}(F_{rx} - F_{fy} \sin(\delta)) + m \dot{v}_y \omega$$
$$\dot{v}_y = \tfrac{1}{m}(F_{ry} + F_{fy} \cos(\delta)) - m \dot{v}_x \omega$$
$$\dot{\omega} = \tfrac{1}{I_z}(F_{fy} l_f \cos(\delta) - F_{ry} l_r)$$

Parameters

RAMPSTEER

$$F_{fy} = 3.13 \sin(2.165 \arctan(0.268\alpha_f))$$
$$F_{ry} = 2.84 \sin(2.38 \arctan(0.242\alpha_r))$$

DRIVE

$$F_{rx} = (0.266 - 0.1v_x)D - 0.1025 - 0.0011v_x^2$$

sys_id_object.py

Usage:

SystemIdentification(mask, identify, num_dots, use_previous_pos)

```
sys = SystemIdentification("bg3_mask_sysid.png", "all", 1, True)
sys.identify_system()
```

Print sys.params to see what the system is identified

```
for identify in generator(identify):
    for direct in directories
        data = load_new_data(direct)
        data = preprocess_data(data)
        stack_data(data)
    if identify == 'drive':
        weigh_data_3stage()
    params = search_params()
```

Future considerations

- Do identification for sample with three dots
  - More slip angle
- Angle and acceleration approximation not always perfect
- Use alternative models for rampsteer and drive
  - Simplified propulsion model as in Liljestrand (2018)
- Improve the peak shaving algorithm might help
- Increase number of samples
- Perform closed loop to improve parameters as in Liniger (2018)

Validation & simulation

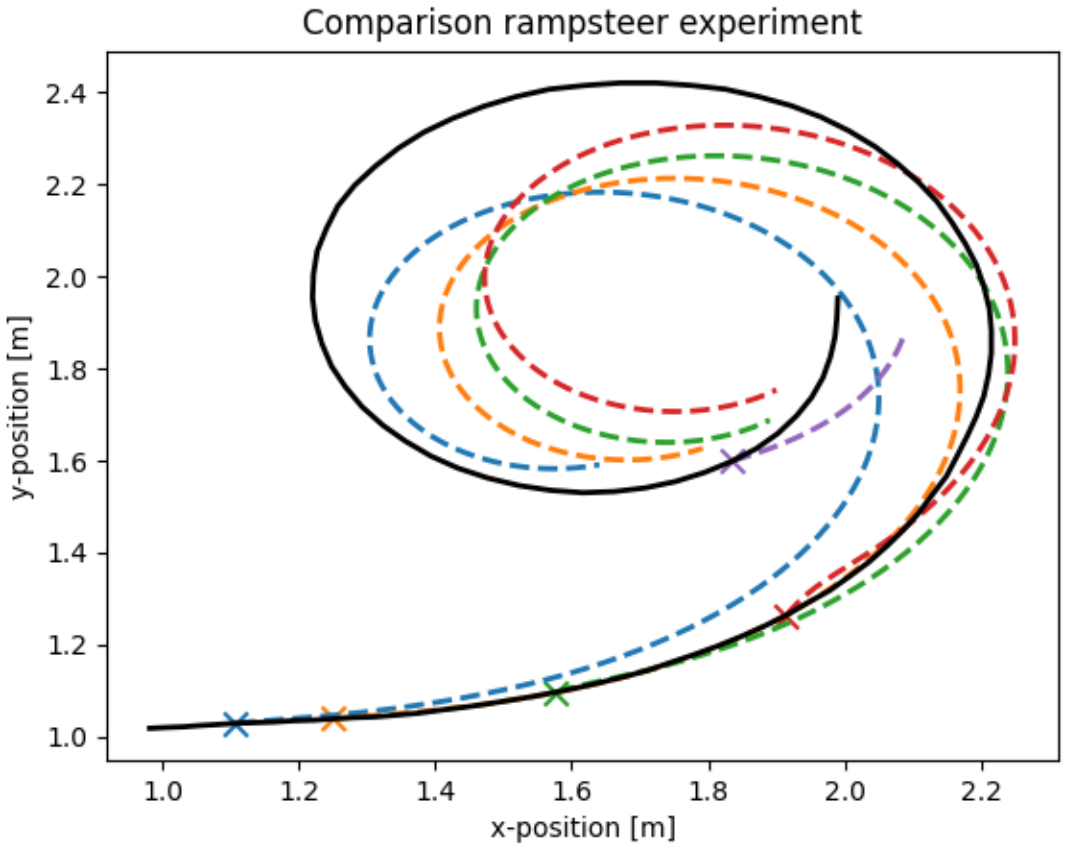
General

- 4th order Runge-Kutta
- Simulating full bicycle model

$$
\begin{aligned}
\dot{x} &= v_x \cos(\phi) - v_y \sin(\phi) \\
\dot{y} &= v_y \sin(\phi) + v_x \cos(\phi) \\
\dot{\phi} &= \omega \\
\dot{v}_x &= \tfrac{1}{m}(F_{rx} - F_{fy} \sin(\delta)) + m\dot{v}_y\omega \\
\dot{v}_y &= \tfrac{1}{m}(F_{ry} + F_{fy} \cos(\delta)) - m\dot{v}_x\omega \\
\dot{\omega} &= \tfrac{1}{I_z}(F_{fy} l_f \cos(\delta) - F_{ry} l_r)
\end{aligned}
$$

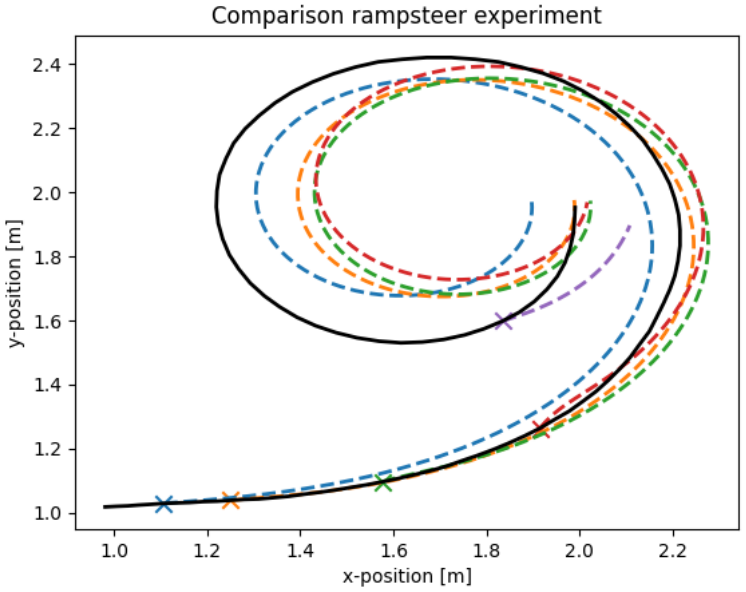
Rampsteer simulation

- Given a measurement with a given input of D = 0.7 and maximum steering angle
- Dashed lines are open loop simulation, where x-marker is the initial state
- Seems to be following trajectory for linear dynamics (low speed and slip angle)
- Does not fully capture high the spiral
  - Error of about 20 cm, which is considerable on a racetrack
  - Should be taken into consideration for the prediction horizon

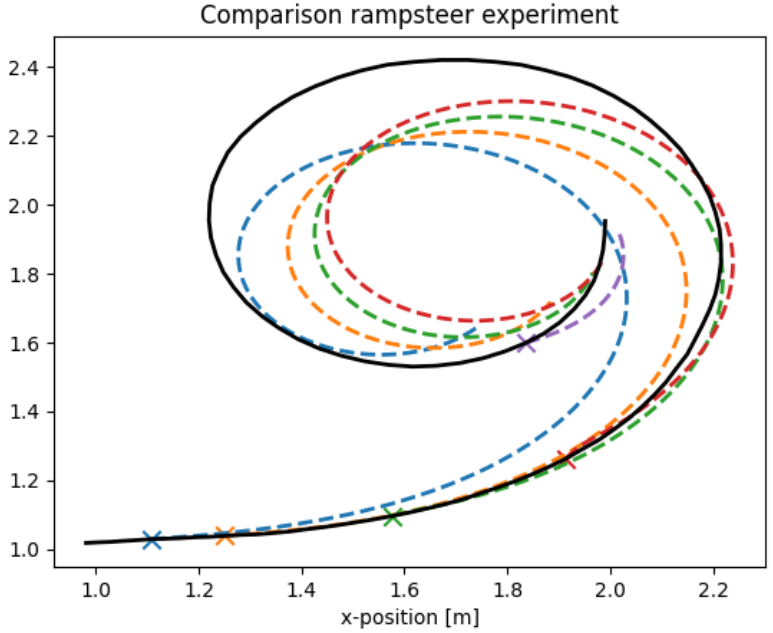


Sensitivity

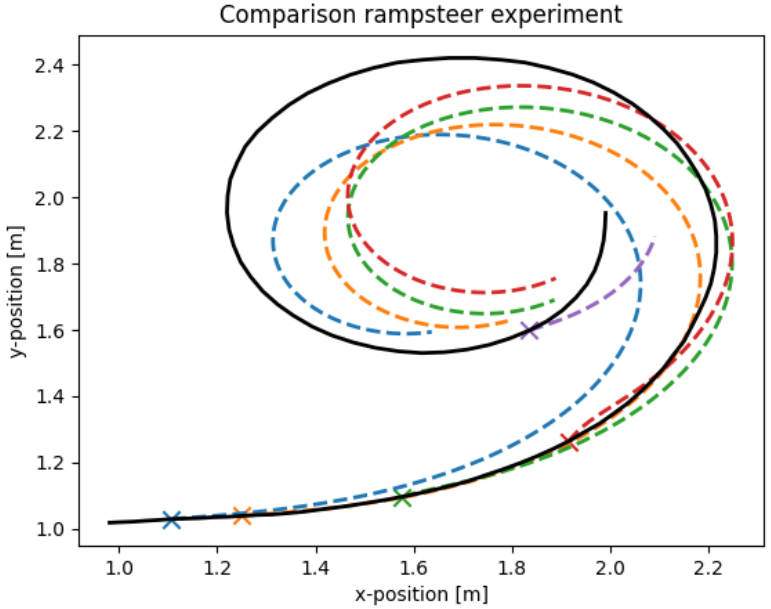
$$F_{fy} = 3.13 \sin(2.165 \arctan(0.268\alpha_f))$$
$$F_{ry} = 2.84 \sin(2.38 \arctan(0.242\alpha_r))$$
$$F_{rx} = (1.3 * 0.266 - 0.1v_x) - 0.1025 - 0.0011v_x^2$$



$$F_{fy} = 3.13 \sin(2.165 \arctan(2 * 0.268\alpha_f))$$
$$F_{ry} = 2.84 \sin(2.38 \arctan(2 * 0.242\alpha_r))$$

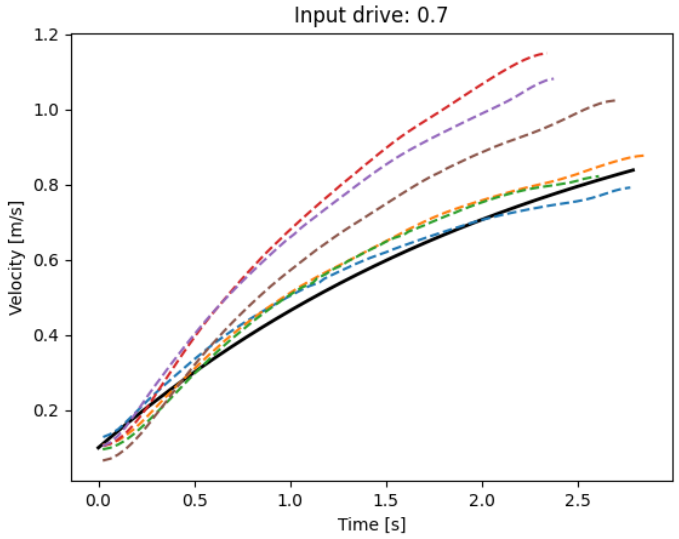


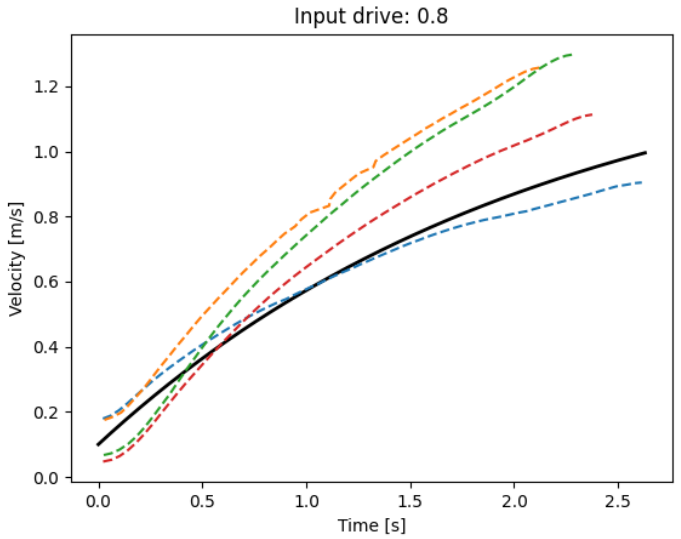
$$\dot{\omega} = \frac{1}{2 * I_z}(F_{fy} l_f \cos(\delta) - F_{ry} l_r)$$

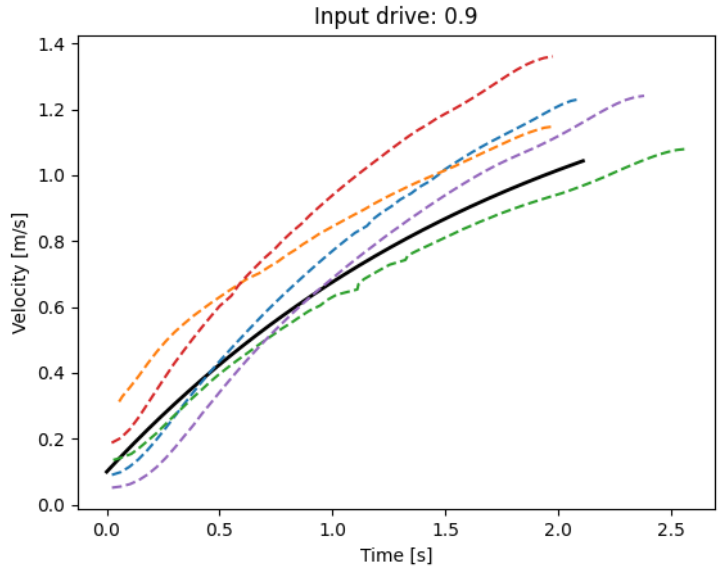


Drive simulation

- A step response with input D = 0.7, D = 0.8 and D = 0.9
  - Initial speed 0
- Seems OK, but a little bit slower than what I would expect
- Improved a lot from the first estimations
  - Removed bad samples
  - Removed outliers in acceleration data





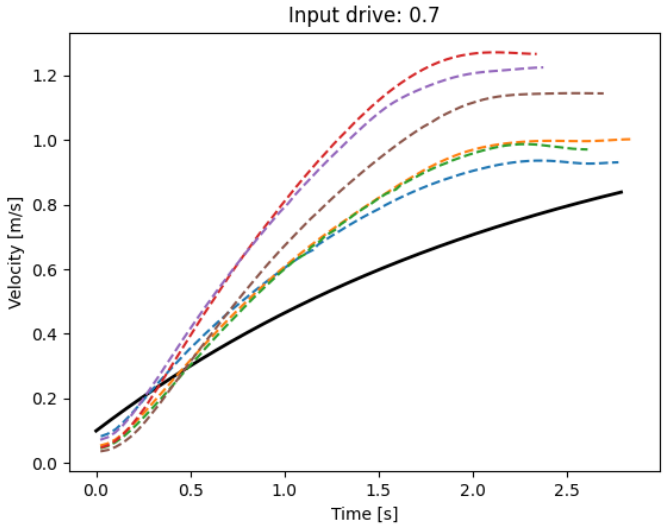


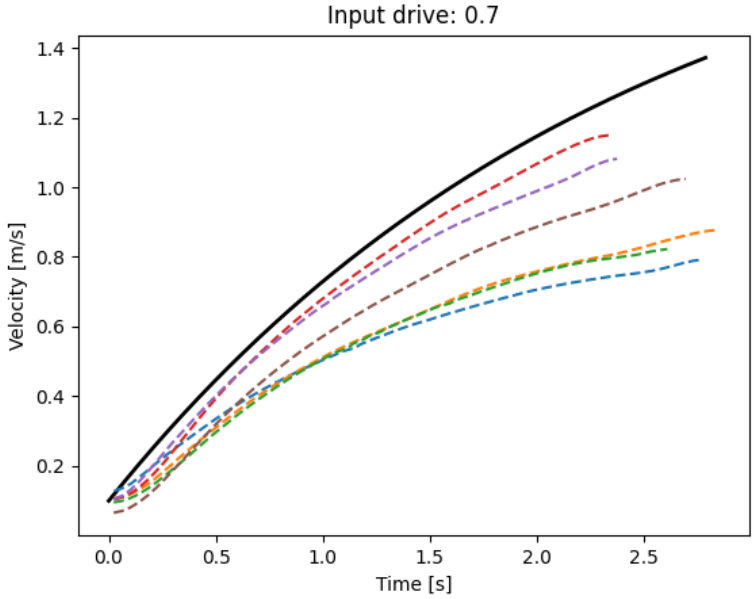
Sensitivity

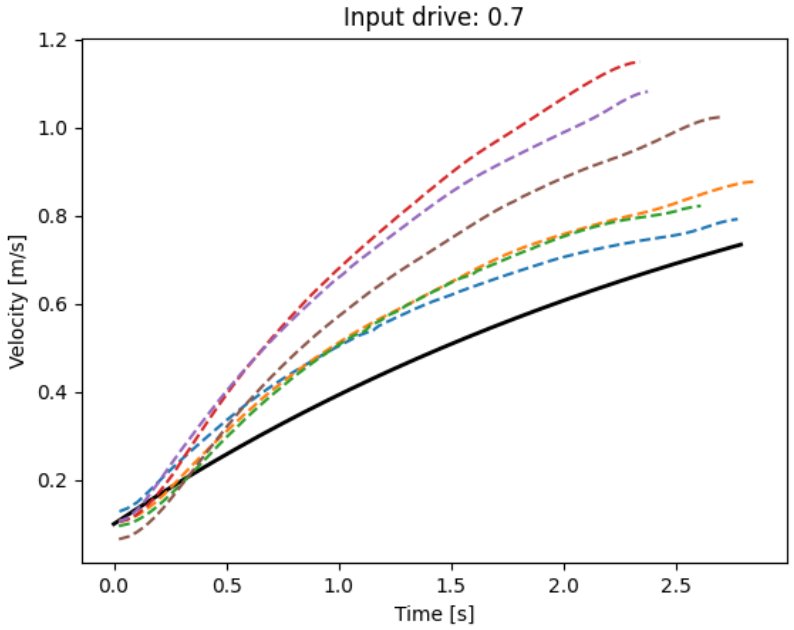
Fisheye effect

$$F_{rx} = (1.3 * 0.266 - 0.1v_x) - 0.1025 - 0.0011v_x^2$$

$$\dot{v}_x = \frac{1}{1.3*m}(F_{rx} - F_{fy}\sin(\delta)) + m\dot{v}_y\omega$$
$$\dot{v}_y = \frac{1}{1.3*m}(F_{ry} + F_{fy}\cos(\delta)) - m\dot{v}_x\omega$$







Future considerations

- The slip parameters is highly dependent on the vehicle propulsion
  - Increasing C_m1 will also improve the trajectory of the rampsteer simulation
- Closed-loop experiments might make the identification of the propulsion parameters better
- Increasing the space could make the car reach steady state
  - Will improve both models

Result

Result

- The main goal of identifying a model was reached
  - It should be good enough to use for a controller
- The model could be improved
  - Generate more data
  - Better pre-processing of the data
  - Increase space for the experiments

Questions?

References

Patrick Engström, Daniel Fredriksson, Oscar Olesen, David Olsson, and Sanna Sandberg. High performance autonomous racing with rc cars. 2018.

Giorgio Fontana. Autonomous driving of miniature race cars. 2013.

Fredrik Gustafsson. *Statistical sensor fusion*. Studentlitteratur, 2018.

Matteo Janssen. Proximal algorithms for optimal control ofmodel race cars. 2021.

Jonatan Liljestrand. State estimation of rc cars for the purpose of drift control. 2011.

Alexander Liniger. Path planning and control for autonomous racing. 2018.

Hans B. Pacejka. *Tire and Vehicle Dynamics*. Elsevier Ltd., 2012.