Documentation Erasmus Thesis

Hjalmar Endre

January 2022

Contents

1 Introduction **3**

2 Experiment and data collection **3**

3 State estimation **4**

4 System identification **6**

5 Other **8**
5.1 simulation.py 8
5.2 mapper.py 8
5.3 imgtovid.py 8
5.4 leastsquare.py 8
5.5 TorqueSysId.py 9
5.6 position_detection.py 9
5.7 kalman.py 9

1 Introduction

This thesis project aimed to develop a model for a RC car by doing system identification. The main scripts was developed in Python. Part 2, 3 and 4 is the most important parts in order to reproduce the results of the thesis project.

2 Experiment and data collection

The main script to perform the experiments is called *get_data_object.py* and can be found in the folder *apps*. The script will create two threads upon creation, one thread will run the camera and one thread will communicate the action to the RC car. The reason why this design decision was made was because the script was too slow to run everything in one thread and still capture images with similar time between every picture. No image processing is done while images are captured, since this takes too much time. A suitable framrate was decided to be 25 frames per second. The size of the pictures that was captured in this project was in size 1024x1280. The script will create an object called *get_data.py*, which can be created by using the following:
args:
height: 1024 was standard during my experiments
width: 1280 standard during my experiments
framerate: 25 standard during my experiments. Depends on computer speed
drive: state what input drive should be applied to the racer. Number between 0-1.
turn: how fast and to what extent should the racer turn. For a speed test, set turn=0.
For a left turn, set speed to 1 for it to ramp up to max turn (-1 for right)
experiment: "rampsteer" or "drive" or "test"
battery_run: how many experiments is done with the batteries?

The result will be saved to a folder that gets the name depending on input variables. Example:

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

will output a folder with name test/5D08L and
d = get_data(1024, 1280, 25, 0.7, -1, 'rampsteer', 2) outputs

rampsteer/2D07R
In order to start the experiment, one can apply the following two lines:
d = get_data(1024, 1280, 25, 0.8, 1, 'drive', 5)
d.get_data()
The setup that is required to do this will be to have the Camera and a large area as in figure 1. In the output directory, the script will save images numbered as in image (1),...,image(100) for example. There will also be a numpy array with

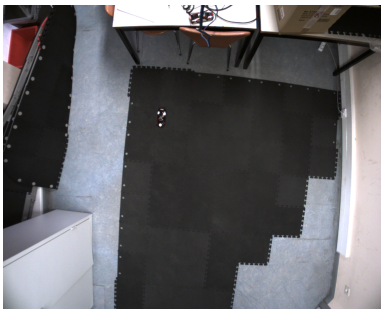

Figure 1: The setup

the timestamps of every picture (*timestamp.npy*). At the moment, too many pictures are saved to the folder, so it might be an idea to clean out the first pictures for performance (until the RC car starts moving). It is also important to have a good amount of batteries and a battery charger when doing experiments, since they run out quickly.

3 State estimation

In part 2, we obtained output folders with images. In the next script, *find_pos_object.py* the aim is to translate a list of pictures into speed, velocity and angle of the RC car. The code is found in the folder *apps*. Example of how to use the script:

p = position_estimation(num_dots: int, path: str, mask: str, draw_graphs: bool)
p = position_estimation(3, "camera/captures/rampsteer/3D09L", 'bg3_mask_sysid.png', True)
p.find_trajectory()
p.draw_velocity()

draw_graphs = true will print out a trajectory after the script is finished. If the user do not want a graphic output, set it to false.

num_dots: the number of white dots that is tracked in the images. path: will be the path to a folder where a set of images is (as well as timestamp vector stored as npy file)
mask: name of the mask that will be used
The starting position is chosen by the user by clicking on the white markers in a pop-up. An important part of the script is to decide whether the script should look for one or more white markers on the RC car. When the RC car only has one marker, the state vector as in equation 1 will be saved to the output directory.

$$x = [x, v_x, y, v_y] \tag{1}$$

If the user states that the car has three markers, the state vector that will be

saved is equation 2.

$$x = [x, v_x, y, v_y, \phi, \omega] \tag{2}$$

The difference between the two is that the angle can be measured from the position of the white markers, which is not the case with one marker. The angle is calculated from the y-vector to the vector of direction of the RC car. It is done by arctan2, which can create some jumps in the measurement of the angle. To perform the state estimation, a constant velocity Kalman filter can be used [Gustafsson, 2018]. For every state matrix pair ($[x, v_x]$), a block is repeated as in equation 3.

$$\begin{aligned} F &= \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \\ P &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ R &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \sigma_r^2 \\ Q &= \begin{bmatrix} \frac{dt^4}{4} & \frac{dt^3}{3} \\ \frac{dt^3}{3} & dt^2 \end{bmatrix} \sigma_v^2 \\ H &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{3}$$

The variance for R and Q is set to be 0.00317 and 500^2 respectively. The variance for the measurements was decided by an experiment with a RC car standing still and the variance for the process noise by trial and error. The matrices are sent to the script *kalman.py*, found in *apps*. In *kalman.py*, the recursive Kalman filter is performed, seen in equation 4.

$$\begin{aligned} \hat{x}_k &= F x_{k-1} \\ \hat{P}_k &= F F^T + Q \\ S &= H \hat{P}_k H^T + R \\ K &= (\hat{P}_k H^T) S^{-1} \\ y &= z - H \hat{x}_k \\ x_k &= \hat{x}_k + K y \\ P_k &= (I - (KH)) \hat{P}_k \end{aligned} \tag{4}$$

The Kalman filter is not only used to smooth measurements depending on previous measurements, but the predictions are also important for the script ($\hat{x}_k$). When processing the images, the location of the RC car is boxed in a rectangle. To see how the script interacts with the image processing, see the class *PositionsDetector* in *image_processing*. Another design parameter that is used in the script is the binary threshold. The threshold will start off very high, and will then adapt to lower threshold if not enough candidates for markers is detected. After detecting candidates, a most likely candidate will be chosen.

The distance to the candidates will be compared to the predictions made by the Kalman filter. If there are more than one dot on the RC vehicle, every position and candidate will be combined. The lowest error will be chosen as the best candidates. Another important aspect of this script is the calibration of the fisheye parameters. Every image is calibrated using the parameters that is unique for every room. If the setup is moved and the height of the camera is different, then the parameters should probably be recalibrated for best performance. Another important constant is the conversion factor from pixel to meter (0.002745). The constant can be derived using the script *mapper.py* in folder *test*. The states for every image will be exported as a .npy in the output folder.

4 System identification

The main script for the system identification is called *sys_id_object.py* and can be found in *rcracer* directory. To start using this script one should preferably have an array of directories from experiments. The main object can be used in the following way:
Input:
SystemIdentification(Mask, "drive" or "rampsteer" or "all", number of dots in samples, Use old positions or not)

Example on how to use the class:
sys = SystemIdentification("bg3_mask_sysid.png", "all", 1, True)
sys.identify_system()
print(sys.params)
The script will loop through all of the experiments in the folders named rampsteer and drive. If the angle was not measured, the script will approximate angle based on the how is changed in x and y direction. All of the states will be smoothened by a gaussian filter. This will all be in the preprocession method of the object. Depending on the number of images and the name of the directories, inputs are also created in the preprocessing step. As mentioned in part 3, the angles are estimated using arctan2. This means that it is necessary to use an unwrapping function to reduce big jumps. This is important for the conversion from global to local coordinates, as seen in equation 5.

$$\begin{aligned} v'_x &= v_x \sin(\phi) - y \sin(\phi) \\ v'_y &= v_x \sin(\phi) + y \cos(\phi) \end{aligned} \tag{5}$$

If the angle is estimated wrong, the velocity in y-direction will be too big. There are two models that can be identified using the object. The first one is the Pajecka tyre model, as seen in equation 6 [Pacejka, 2012]. In this model, the blue color is input from the experiment, brown is a measured constant and red is the parameters that should be fitted to the data. The force in y-direction

is approximated using equation 7.

$$\begin{aligned} \alpha_f &= -\arctan(\frac{l_f\omega + v_y}{v_x}) + \delta \\ \alpha_r &= \arctan(\frac{(l_r\omega - v_y)}{v_x}) \\ F_{fy} &= \beta_{f3}\sin(\beta_{f2}\arctan(\beta_{f1}\alpha_f)) \\ F_{ry} &= \beta_{r3}\sin(\beta_{r2}\arctan(\beta_{r1}\alpha_r)) \end{aligned} \tag{6}$$

$$\begin{aligned} \hat{F}_{fy} &= \frac{mv_x\omega l_r}{(l_r + l_f)}\cos(\delta) \\ \hat{F}_{ry} &= \frac{mv_x\omega l_f}{(l_r + l_f)} \end{aligned} \tag{7}$$

An important assumption for equation 7 is that the RC car is in steady state. When calculating the slip angles and slip forces, some outliers are deleted. One of the outliers is if we have a positive slip angle but a negative slip force and vice versa. In a subsequent step, the parameters are identified via a non linear solver. The other equation meant for identification is the propulsion parameters as seen in equation 8 [Liniger, 2018]. The green parameters in equation 8 are to be identified.

$$\begin{aligned} F_{rx} &= (C_{m1} - C_{m2}v_x)D - C_{f1} - C_{f2}v_x^2 \\ C_{f2} &= \frac{1}{2}\rho C_d A \\ F_{rx} &= ma_x \end{aligned} \tag{8}$$

An alternative model was presented in Liljestrand [2011] and Engström et al. [2018] as seen in equation 9.

$$F_{rx} = C_{m1}D - C_{m2}v_x - C_{f1} - C_{f2}v_x^2 \tag{9}$$

These parameters can be estimated via least squares. In the script, I have included three ways to identify propulsion parameters. One uses the scikit solver, one using a weighted equation system with three equations and one simple least square with data stacked. The weighted equation system can be seen in equation 10 [Engström et al., 2018] and the least square in equation 11.

$$\begin{bmatrix} D_0 & 0 & -1 \\ D & -Dv_x & -1 \\ D & -Dv_x & -1 \end{bmatrix} \begin{bmatrix} Cm_1 \\ Cm_2 \\ C_{f1} \end{bmatrix} = \begin{bmatrix} 0 \\ C_{f2}v_x^2 \\ ma_x + C_{f2}v_x^2 \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} D_1 & -D_1v_{x1} & -1 \\ D_2 & -D_2v_{x2} & -1 \\ D_3 & -D_3v_{x3} & -1 \end{bmatrix} \begin{bmatrix} Cm_1 \\ Cm_2 \\ C_{f1} \end{bmatrix} = \begin{bmatrix} ma_x + C_{f2}v_{x1}^2 \\ ma_x + C_{f2}v_{x2}^2 \\ ma_x + C_{f2}v_{x3}^2 \end{bmatrix} \tag{11}$$

The identified parameters are saved in a dictionary, that can be printed by including *print(object.params)*. With the data that was gathered in the thesis,

the identified parameters can be seen in equation 12.

$$\begin{aligned} F_{fy} &= 3.13 \sin(2.165 \arctan(0.268\alpha_f)) \\ F_{ry} &= 2.84 \sin(2.38 \arctan(0.242\alpha_r)) \\ F_{rx} &= (0.266 - 0.1v_x)D - 0.1025 - 0.0011v_x^2 \end{aligned} \tag{12}$$

5 Other

5.1 simulation.py

The actual trajectories was compared to the model with a Runge-Kutta 4th order integrator. This can be found in *simulation.py*. The model that is being simulated can be seen in equation 13.

$$\begin{aligned} \dot{x} &= v_x \cos(\phi) - v_y \sin(\phi) \\ \dot{y} &= v_y \sin(\phi) + v_x \cos(\phi) \\ \dot{\phi} &= \omega \\ \dot{v}_x &= \frac{1}{m}(F_{rx} - F_{fy} \sin(\delta)) + m\dot{v}_y\omega \\ \dot{v}_y &= \frac{1}{m}(F_{ry} + F_{fy} \cos(\delta)) - m\dot{v}_x\omega \\ \dot{\omega} &= \frac{1}{I_z}(F_{fy} l_f \cos(\delta) - F_{ry} l_r) \end{aligned} \tag{13}$$

The script can be run without any additional inputs. The parameters from the system identification has to be inserted manually. The script is useful for validation of parameters.

5.2 mapper.py

This script is very simple. Thea idea is that the user should point out the edges of the mattress. Since we know how many tiles there are (8 in the script) and the length of every tile is a certain length.

5.3 imgtovid.py

A script partly taken from GitHub. The only change is that the sorting of the images is matched to how the images are named in the folders. Arguments that has to be sent into the script is input folder, input images type, output type and output folder.

5.4 leastsquare.py

Most of this script is now moved into the main script sys_id_object.py. Contains a few ways to do least squares on drive parameters.

5.5 TorqueSysId.py

Nothing new in this file. Included some ways to calculate the inertia that an RC vehicle would have if kept off the ground. My idea was that it could be used to identify the DC-motor parameters in the propulsion equation.

5.6 position_detection.py

Some changes were made to position_detection.py. At the moment, I do not use the mask that is being sent into the the *find_pos_object.py*. Instead, the script masks everything but a box around the RC car. This speeds up the script greatly and reduces the risk of wrong markers. I have also added threshold as an input to the object. This enables *sys_id_object.py* to adapt the threshold depending on the amount of markers that is identified.

5.7 kalman.py

The script takes the matrices as input. It contains the recursive Kalman algorithm and is used for the state estimation.

References

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

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

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.