User manual

William Wärn
Karl Karlsson
Arvid Linder
Carl Sjöstedt
Emil Strömblad
Claes Thunberg
Gustav Åberg

December 12, 2022

Version 0.1

Project Identity

Orderer:	Lars Eriksson E-mail: lars.eriksson@liu.se
Customer:	Fredrik Wemmert, Aurobay
Supervisor:	Robin Holmbom E-mail: robin.holmbom@liu.se
Course Responsible:	Daniel Axehill, Gustaf Hendeby E-mail: daniel.axehill@liu.se, gustaf.hendeby@liu.se

Project Members

Name	**Responsibility**	**E-mail**
William Wärn	Project leader	Wilwa907@student.liu.se
Karl Karlsson	Documentation & Design	Kakar847@student.liu.se
Arvid Linder	Software	Arvli383@student.liu.se
Carl Sjöstedt	Test & Hardware	Carsj142@student.liu.se
Emil Strömblad	Information	Emist067@student.liu.se
Claes Thunberg	Quality	Clath750@student.liu.se
Gustav Åberg	Test & Experiment Design	Gusab106@student.liu.se

CONTENTS

DOCUMENT HISTORY

Version	Date	Changes	Done by	Reviewed
0.1	2022-10-23	First draft	Whole group	

1 INTRODUCTION

This document is the user manual for the project *Machine Learning and Adaptive Control for Improving Servo Performance* within the project course *Automatic Control, TSRT10*. The main purpose of this document is to present information on how to operate the test rig used in the project.

2 HARDWARE

A couple of different hardware components work together to be able to control the throttle, mainly: a Raspberry Pi 3b+ and a windows PC. The Raspberry Pi could be described as the interpreter between the throttle and the PC. It receives the reference angle from the PC and the actual angle from the throttle. In this project the PC was used to simulate the vehicle model to limit the computational load on the Raspberry Pi. It should be possible to run the engine simulation directly on the Raspberry Pi as well but this has not been verified during this project.

2.1 Wiring scheme

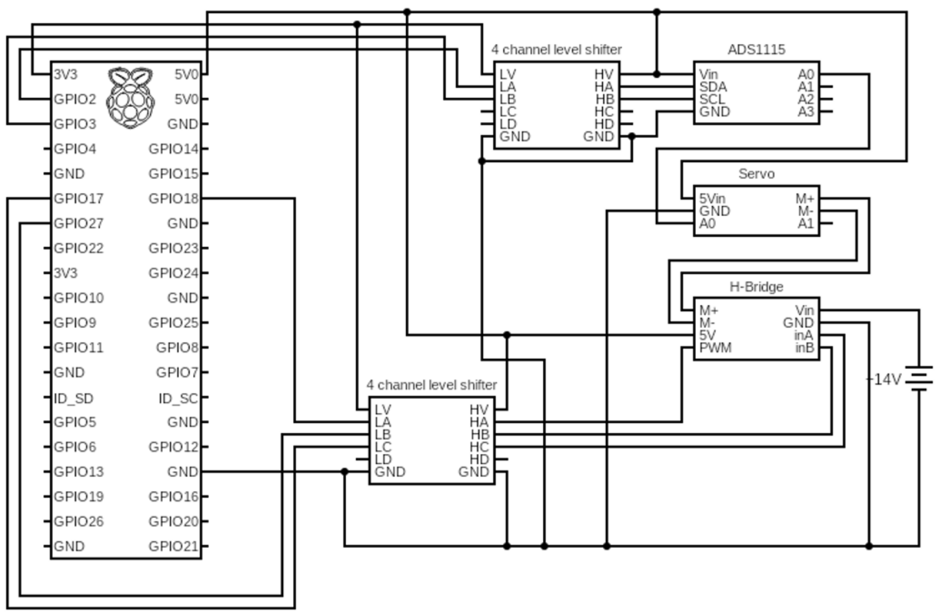


Figure 1: A picture showing the wiring scheme for the test setup.

2.2 Raspberry Pi

The Raspberry Pi utilizes the GPIO pins to communicate with the throttle, these connections can be seen in Figure 2 and Figure 3. Note that the green wire is **not** connected. The Raspberry Pi is powered by a power adapter and connected to the screen through a HDMI-cable. A keyboard and mouse can be connected through the USB-ports on the Raspberry Pi.

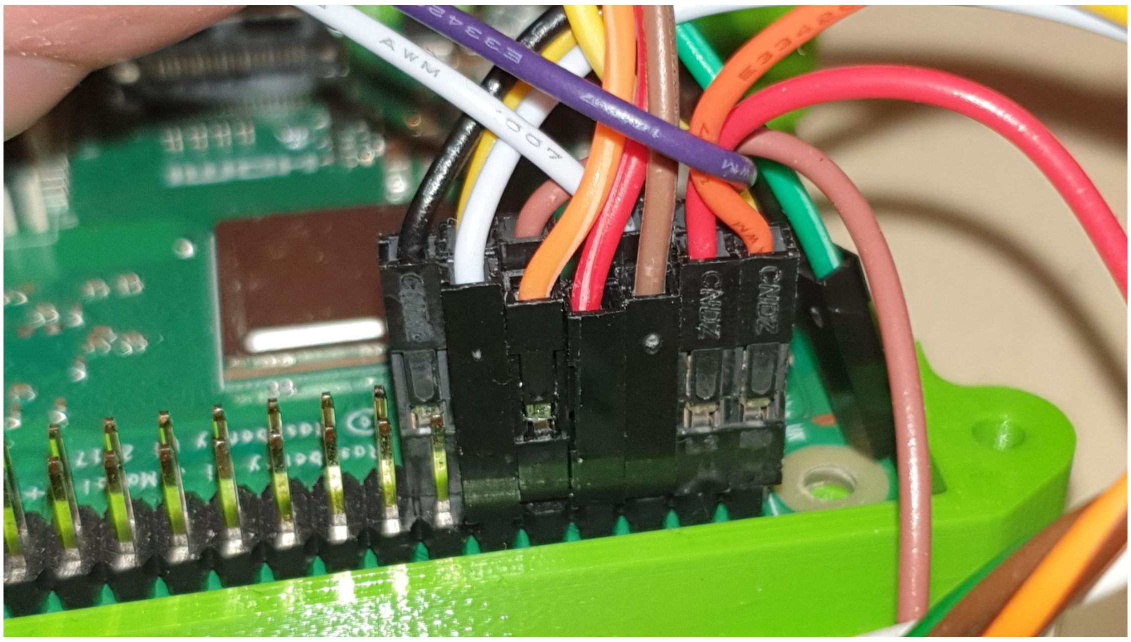


Figure 2: A picture showing how to connect the GPIO pins (1/2).

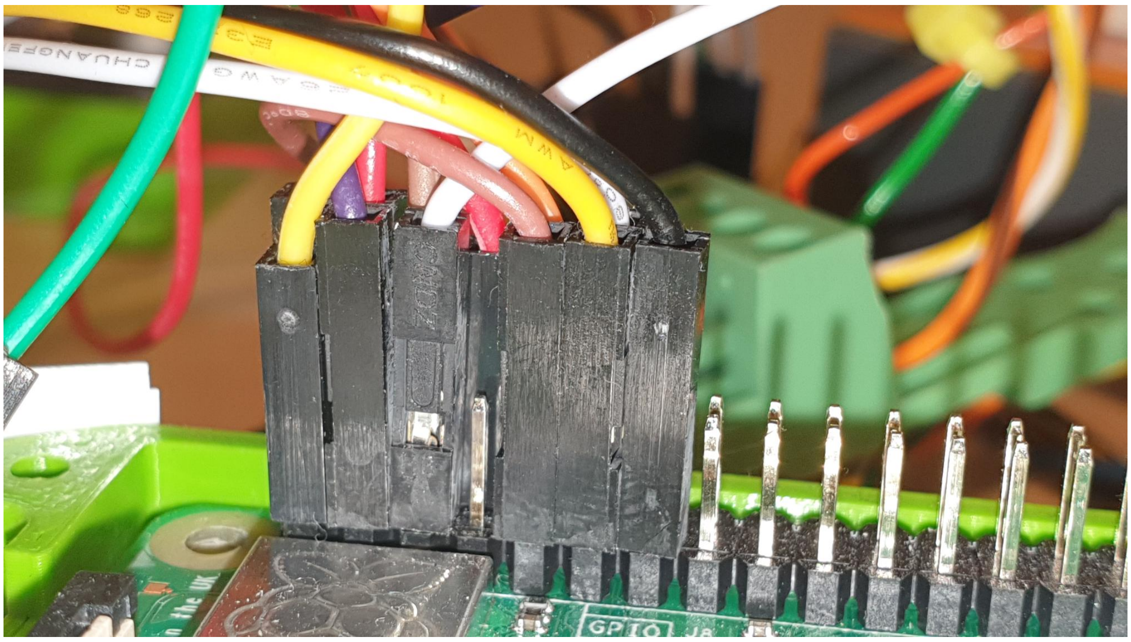


Figure 3: A picture showing how to connect the GPIO pins (2/2).

2.3 PC

The PC is connected to the Raspberry Pi via USB. See Figure 4. This allows serial communication (UART) between the computer and the Raspberry Pi.

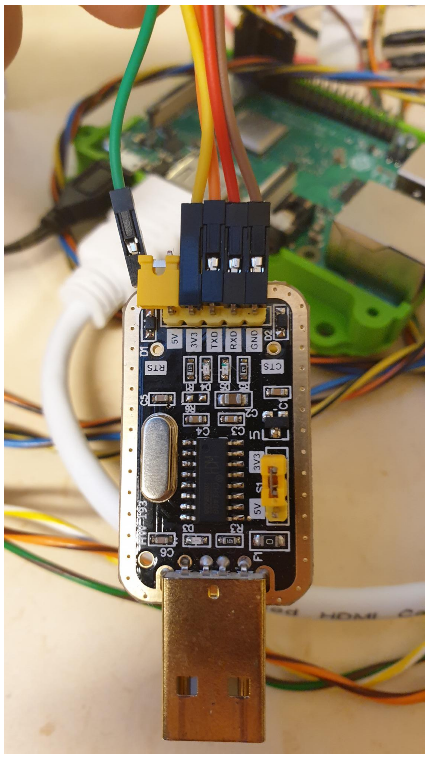


Figure 4: A picture showing how to connect the wires to the USB connection.

2.4 H-bridge

To transform the PWM signals from the Raspberry Pi to a motor current a H-bridge of model AEK-MOT-2DC70S1 is used. To work properly it will need three signals; motor clockwise, motor anticlockwise and PWM. The two first must be set opposite to each other for the motor to work.

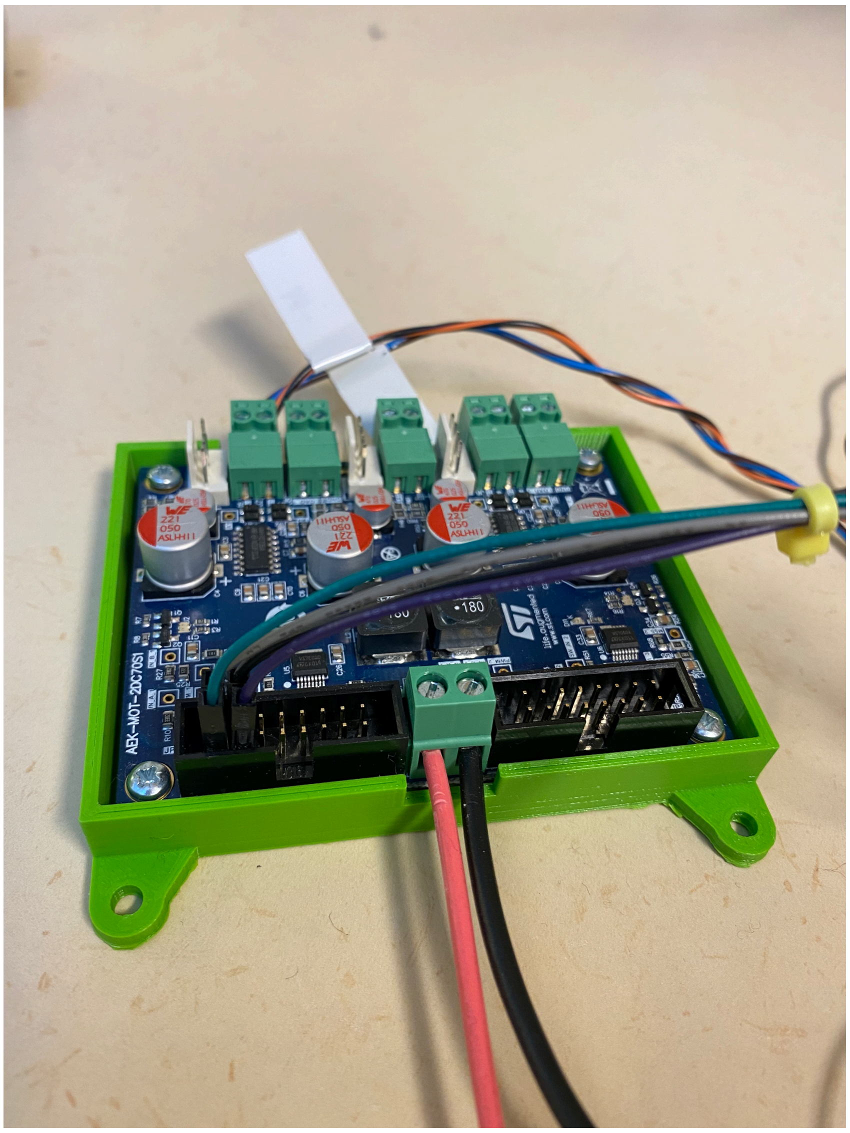


Figure 5: A picture showing how to connect the H-bridge.

2.5 Throttle

The throttle is connected with six cables, see Figure 6. The function of each pin is described in Table 1 below:

Table 1: I2C connections on Raspberry Pi.

Pin	Description
6	Position sensor 1
5	Position sensor 5v
4	Position sensor 2
3	Ground
2	Motor -
1	Motor +

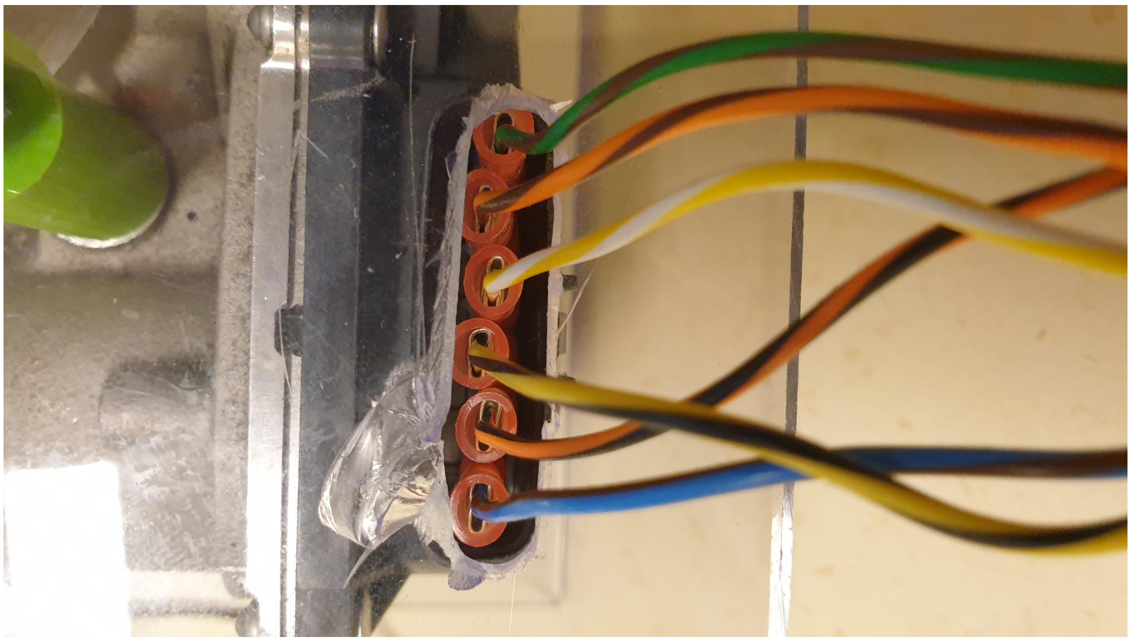


Figure 6: A picture showing how to connect the throttle.

3 SOFTWARE

A number of programs are used to actuate the throttle, both on the PC and on the Raspberry Pi. Note that some of the toolboxes in Matlab have limited compatibility with Mac OS and Linux. In other words, if possible, use a PC with windows.

3.1 Matlab / Simulink

Matlab and Simulink are used extensively on the PC. To be able to run the throttle a Matlab version of **2020b or newer** is required, also a number of different toolboxes should be installed.

Toolboxes:

- Real time desktop.
- Control system toolbox.
- Statistics and Machine learning toolbox.
- Mapping toolbox.

3.2 Python

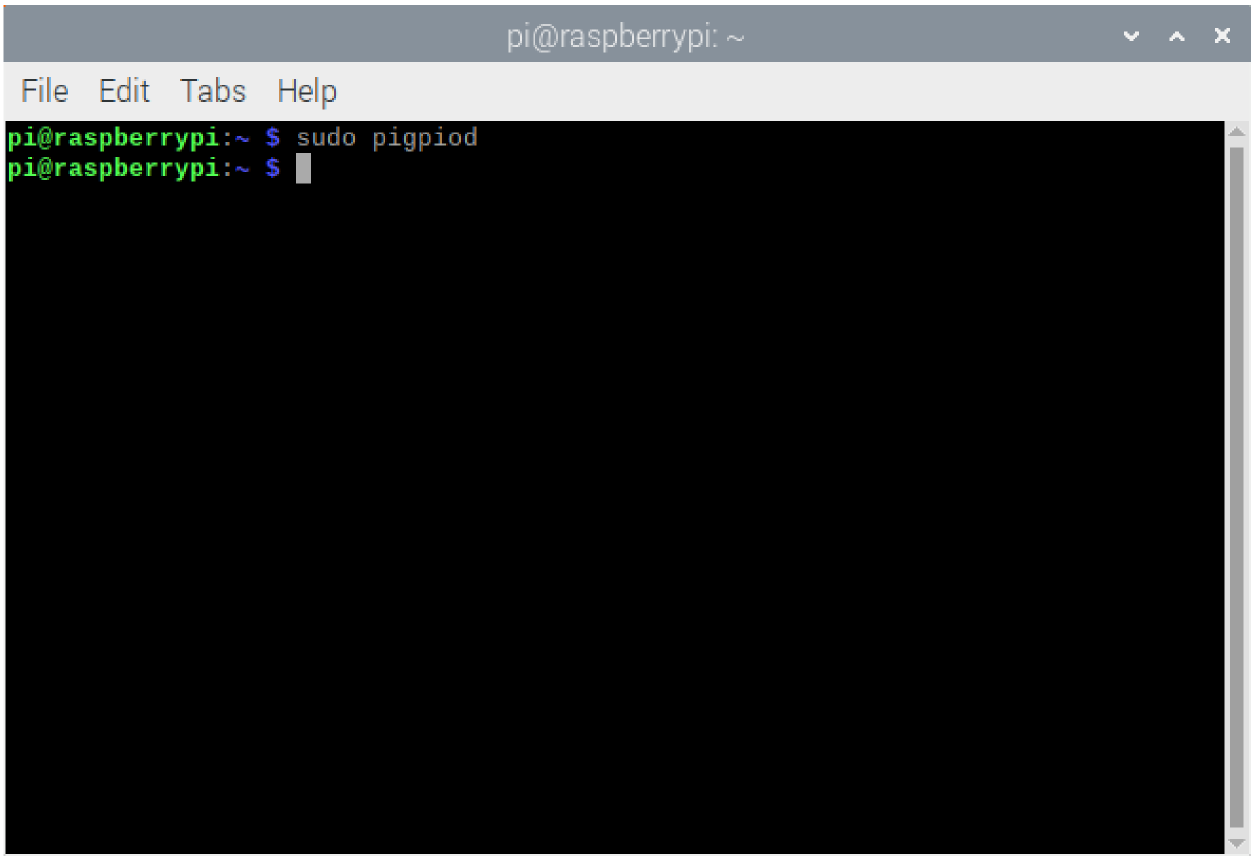
Python is used on the Raspberry Pi. The Thonny IDE is used for this and is an already installed IDE on the Raspberry Pi.

4 START-UP

To get up and running with the setup, follow the steps below.

4.1 Start the Raspberry Pi

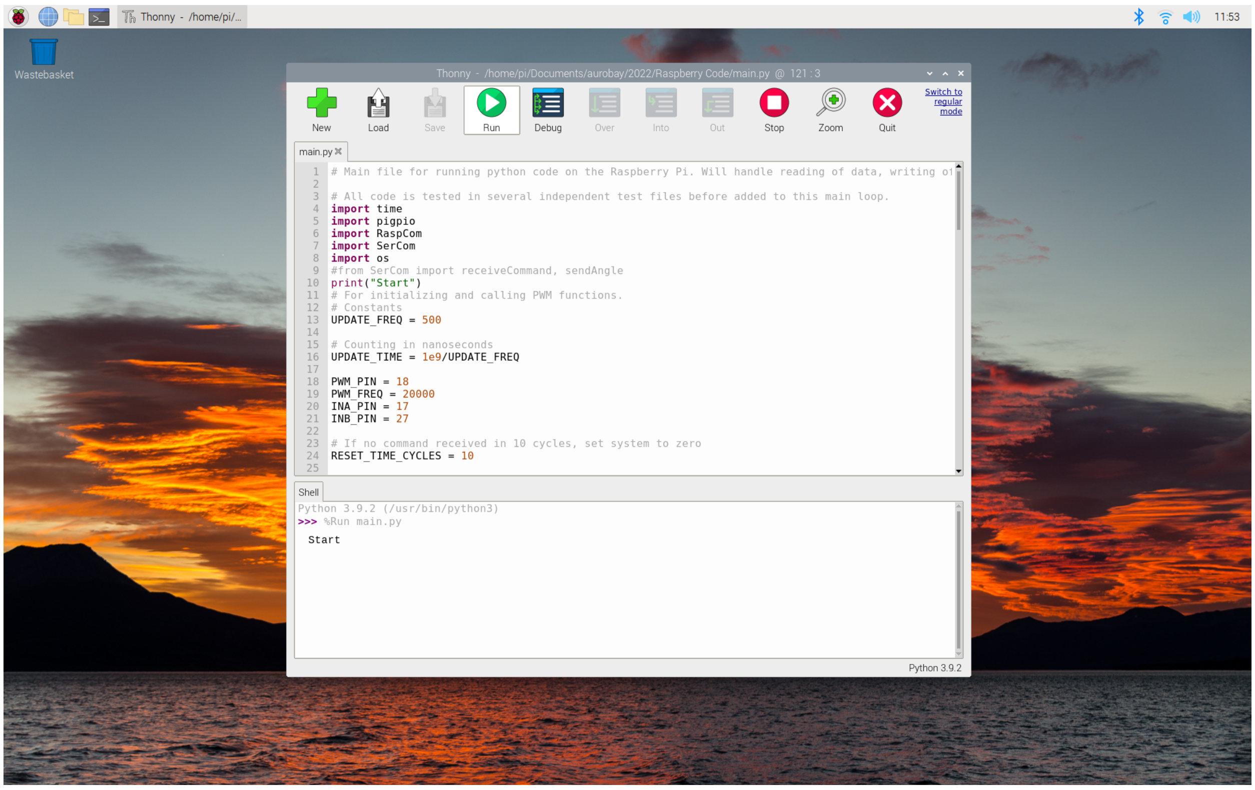
Before starting the Raspberry make sure that the USB connector is not connected to the PC. If it is, the Raspberry might not boot properly. Plug in the Raspberry Pi to a 230V outlet using the connector. It should then boot automatically. Also make sure that the power supply is turned on, otherwise the throttle can not be actuated.
Once the booting procedure has finished, open up the terminal and run *sudo pigpiod*. This launches the pigpio daemon which allows control of the GPIO pins, see Figure 7.



```
pi@raspberrypi:~ $ sudo pigpiod
pi@raspberrypi:~ $ 
```

Figure 7: Running the pigpio daemon from the terminal.

If the pigpio deamon is not launched the Raspberry Pi cannot receive or send data using the GPIO pins. The next step is to launch the Thonny IDE and run the *main.py* script, see Figure 8.

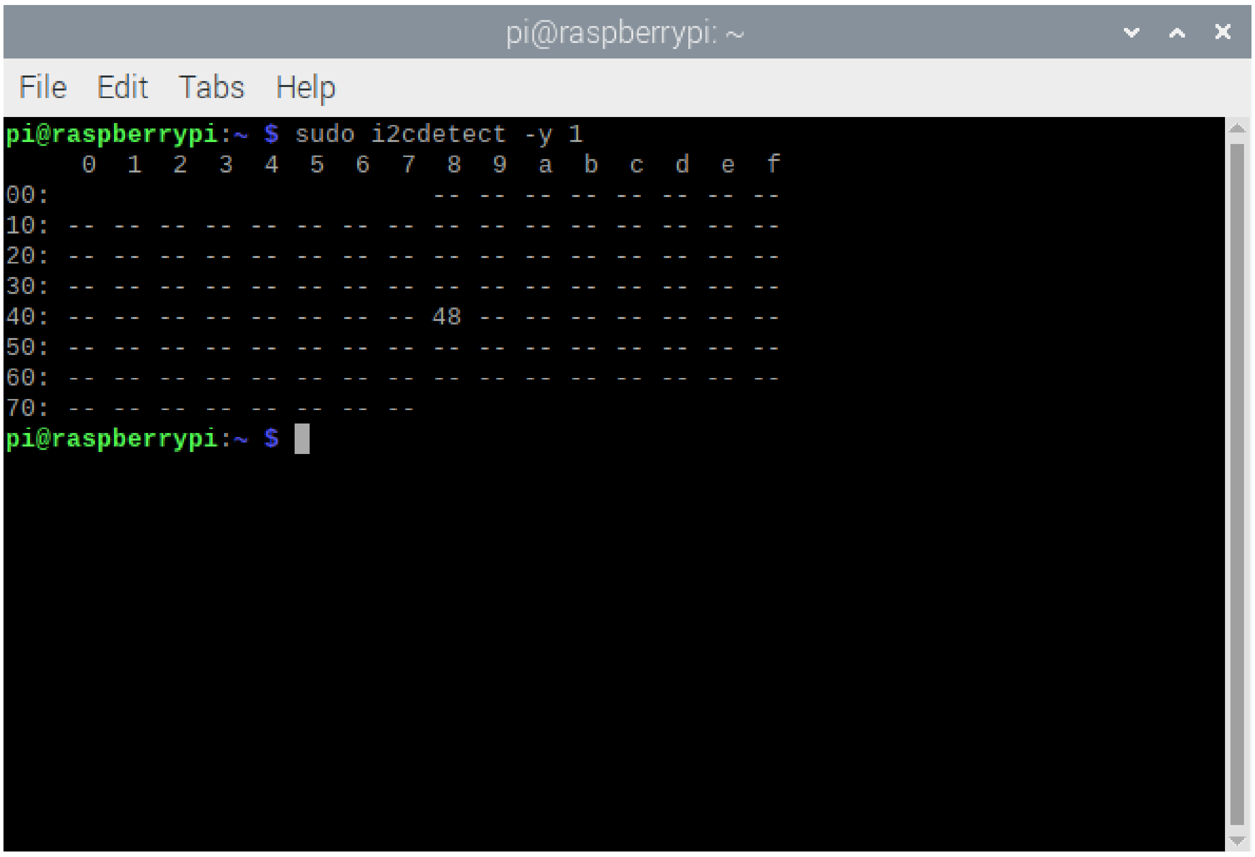


```
# Main file for running python code on the Raspberry Pi. Will handle reading of data, writing of

# All code is tested in several independent test files before added to this main loop.
import time
import pigpio
import RaspCom
import SerCom
import os
#from SerCom import receiveCommand, sendAngle
print("Start")
# For initializing and calling PWM functions.
# Constants
UPDATE_FREQ = 500

# Counting in nanoseconds
UPDATE_TIME = 1e9/UPDATE_FREQ

PWM_PIN = 18
PWM_FREQ = 20000
INA_PIN = 17
INB_PIN = 27

# If no command received in 10 cycles, set system to zero
RESET_TIME_CYCLES = 10

```

```
Python 3.9.2 (/usr/bin/python3)
>>> %Run main.py

  Start
```

Figure 8: The *main.py* program that allows control of the throttle.

Once this is done, actuation of the throttle should be possible. If there is an I2C error, a good idea is to check whether or not the Raspberry Pi can find the connected components. This can done by writing *sudo i2cdetect -y 1*, see Figure 9



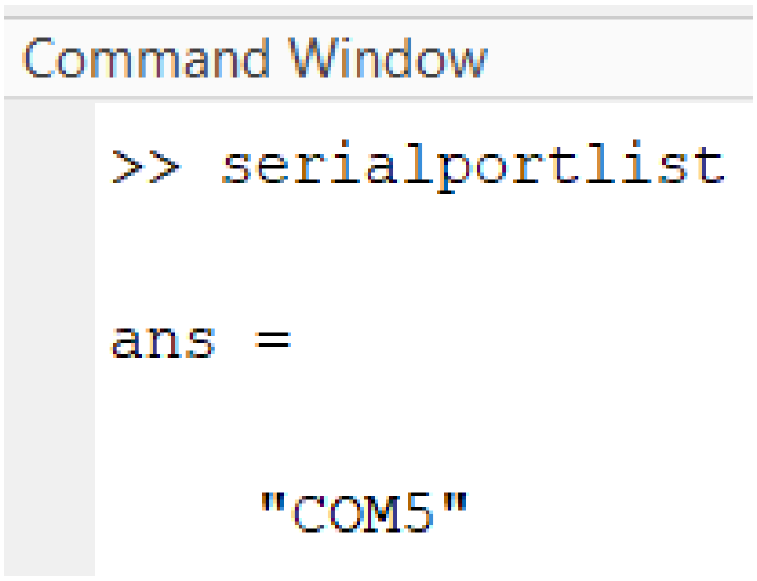
```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $ 
```

Figure 9: Running the i2cdetect command will show all connected devices.

Note that one of the pins shows 48 in Figure 9. If the entire list is empty that means that the Raspberry Pi cant connect to the I2C device.

4.2 Running Matlab

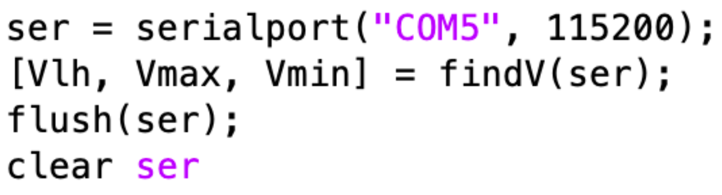
To be able to connect the Raspberry Pi to Matlab, Matlab has to know with which serial port it should communicate with. To do this run the *serialportlist* command in the Matlab command window, see Figure 10.



```
Command Window
>> serialportlist

ans = 

    "COM5"
```

Figure 10: The serialportlist command shows all the available serial ports.

This commands shows all the serial ports that Matlab can connect to. To connect to one of them, create a serial port object (in this case named "ser") and the baudrate should **always** be set to 115200. Then the *findV.m* script should be runnable. This script gets the voltages for maximum opening, closing and limp home throttle position, these parameters are essential for generating a good reference value. The procedure of this is described according to Figure 11. By running this the throttle should be actuated immediately.



```
ser = serialport("COM5", 115200);
[Vlh, Vmax, Vmin] = findV(ser);
flush(ser);
clear ser
```

Figure 11: A picture showing how to create the serial port through Matlab which allows the connection between Raspberry Pi and Matlab.

Before running the throttle the in Simulink all variables has to be accessible in the workspace. To create these variables run the *main_throttle_model.m* script. This script creates all variables and runs the calibration. This script can also simulate the model directly, however running the model directly from Simulink is advantageous since the results can be analyzed immediately.

4.3 Running Simulink

The throttle controller gets its reference value from the engine and drivetrain model from the course *Modelling and Control of Engines and Drivelines, TSFS09*. To find the throttle controller go into the ECU, Boost control and then into Hardware in the loop throttle block, see Figure 12.

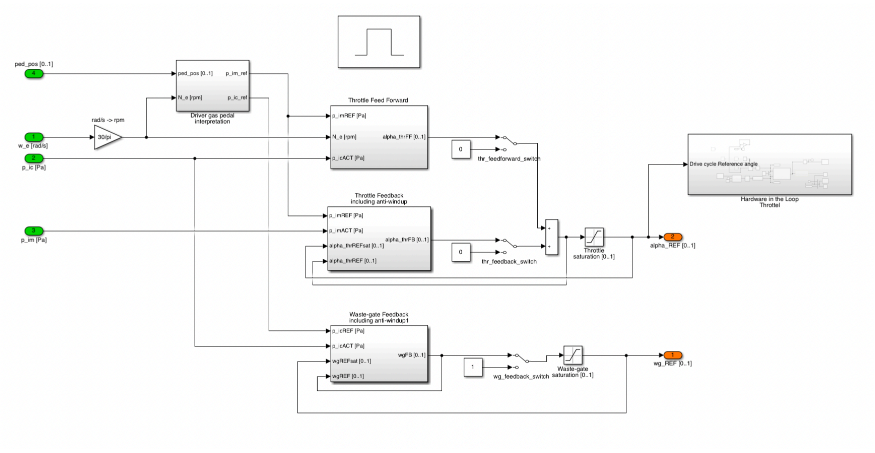

ped_pos [0..1]
Driver gas pedal interpretation
rad/s -> rpm
30/pi
w_e [rad/s]
p_ic [Pa]
p_im [Pa]
Throttle Feed Forward
thr_feedforward_switch
Throttle Feedback including anti-windup
thr_feedback_switch
Throttle saturation [0..1]
alpha_REF [0..1]
Hardware in the Loop Throttel
Drive cycle Reference angle
Waste-gate Feedback including anti-windup1
wg_feedback_switch
Waste-gate saturation [0..1]
wg_REF [0..1]

Figure 12: Shows an overview of the boost control in the vehicle model containing the throttle model.

To adjust the controller enter the *Hardware in the loop throttle block*, the view should look similar to Figure 13.

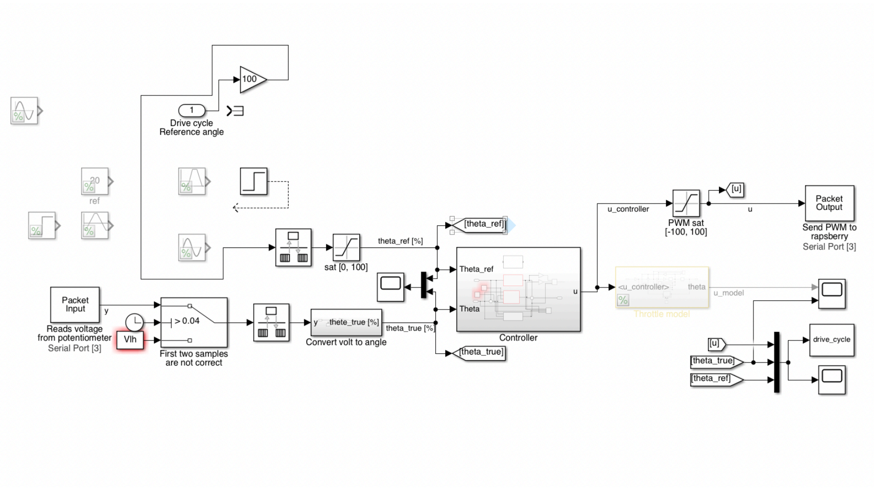
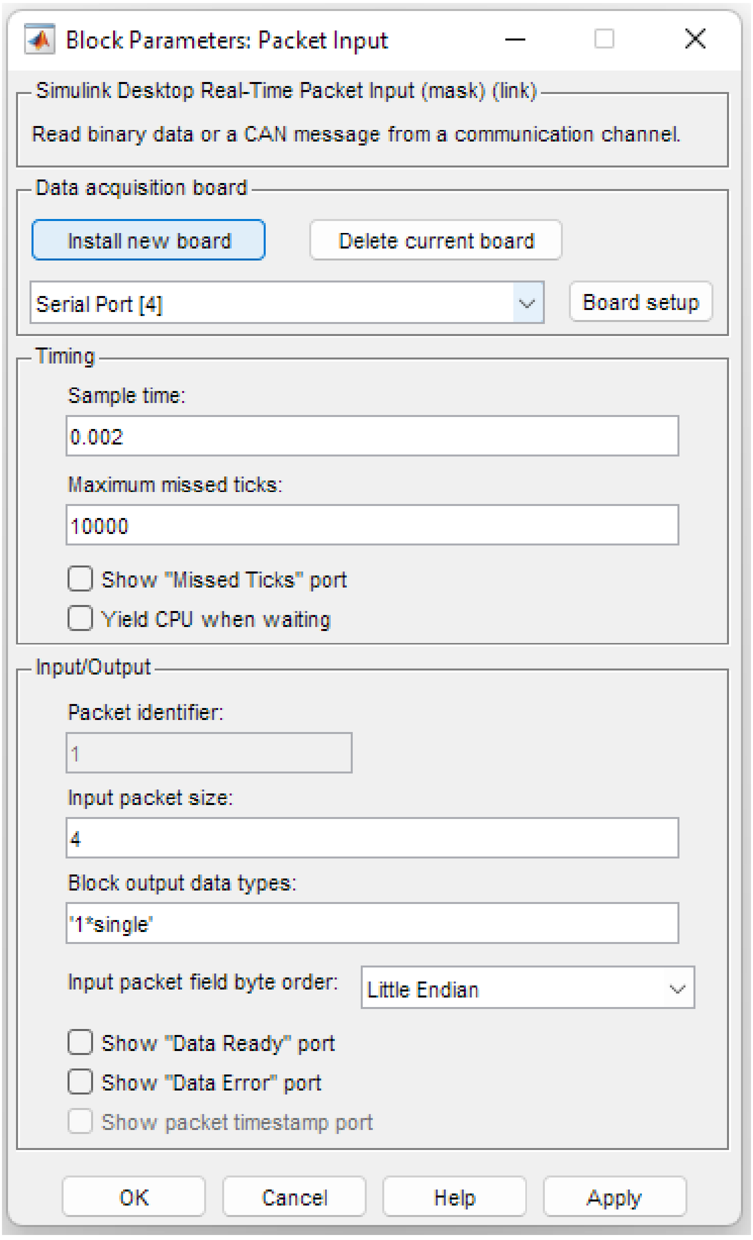


Figure 13: The overview of the throttle controller block.

To be able to send and receive signals using Simulink its first of all very important that Simulink Real Time Desktop is installed. Then the *Packet output* and *Packet input* blocks have to be calibrated for the PC in use. This will differ for pretty much all PC:s since the USB ports often have different names. As mentioned previously use the *serialportlist* command in Matlab command window to get the serial port in use. Once the serial port has been verified this specific serial port be set up in the *Packet output* and *Packet input* blocks, see Figure 14. If the serial port that should be used is not showing up a new board has to be installed. This is done in *install board* > *standard devices* > *serial port*. Here the correct name can be set manually.



Block Parameters: Packet Input

Simulink Desktop Real-Time Packet Input (mask) (link)

Read binary data or a CAN message from a communication channel.

Data acquisition board

Serial Port [4]

Timing

Sample time: 0.002

Maximum missed ticks: 10000

☐ Show "Missed Ticks" port

☐ Yield CPU when waiting

Input/Output

Packet identifier: 1

Input packet size: 4

Block output data types: '1*single'

Input packet field byte order: Little Endian

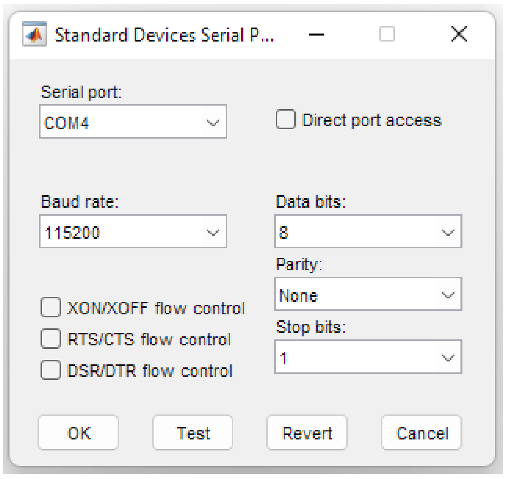
☐ Show "Data Ready" port

☐ Show "Data Error" port

☐ Show packet timestamp port

Figure 14: Shows the different options for the packet in block.

Go to *board setup* and choose the correct serial port. As mentioned earlier the baudrate should **always** be set to 115200, otherwise the communication will not work.



Standard Devices Serial P...

Serial port: COM4

Direct port access: ☐

Baud rate: 115200

Data bits: 8

Parity: None

Stop bits: 1

☐ XON/XOFF flow control

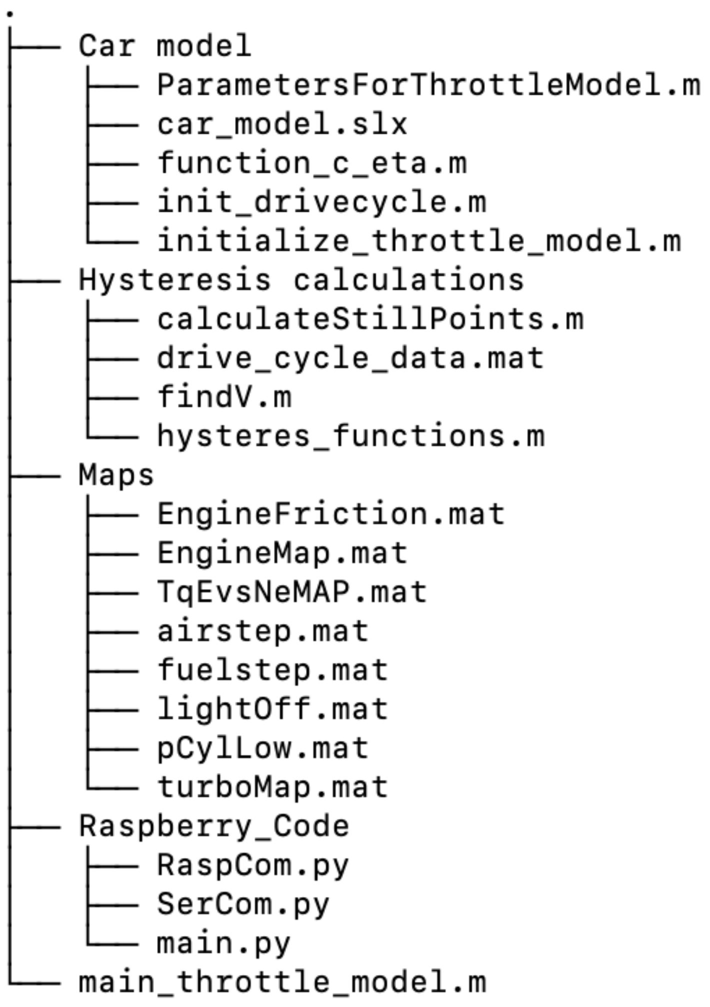
☐ RTS/CTS flow control

☐ DSR/DTR flow control

Figure 15: Shows the different options for the packet in block.

Make sure that the correct board is used for both packet in and out blocks. When these blocks have been calibrated it should be possible to actuate the throttle just using Simulink.

5 FOLDER LAYOUT



```
.
├── Car model
│   ├── ParametersForThrottleModel.m
│   ├── car_model.slx
│   ├── function_c_eta.m
│   ├── init_drivecycle.m
│   └── initialize_throttle_model.m
├── Hysteresis calculations
│   ├── calculateStillPoints.m
│   ├── drive_cycle_data.mat
│   ├── findV.m
│   └── hysteres_functions.m
├── Maps
│   ├── EngineFriction.mat
│   ├── EngineMap.mat
│   ├── TqEvsNeMAP.mat
│   ├── airstep.mat
│   ├── fuelstep.mat
│   ├── lightOff.mat
│   ├── pCylLow.mat
│   └── turboMap.mat
├── Raspberry_Code
│   ├── RaspCom.py
│   ├── SerCom.py
│   └── main.py
└── main_throttle_model.m
```

Figure 16: Shows the tree of the git repository.