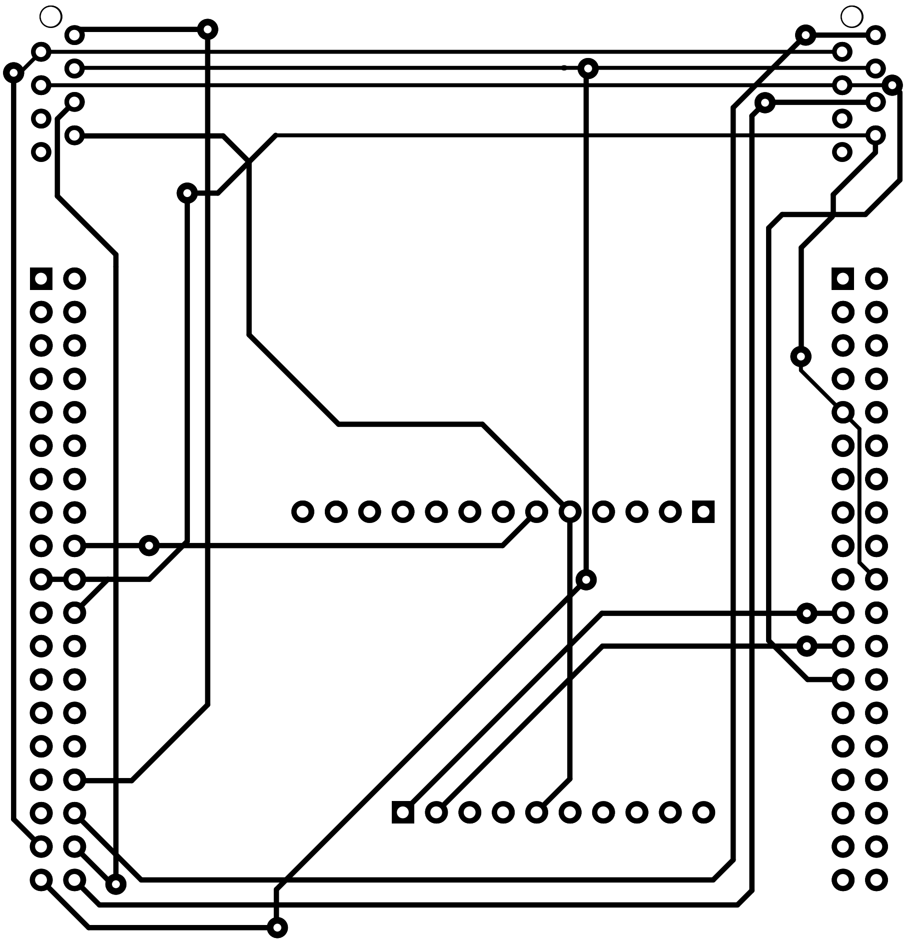
Biorobotics laboratory



MOUSE TREADMILL CONTROL

Wednesday 8th January, 2020

By **Didier Chérubin Negretto**

Professor: **Auke Ijspeert**

Assistant :**Shravan Tata Ramalingasetty**

Semester project description

Objectives and preliminary considerations The project consists of designing and manufacturing the electronics and control for a mouse treadmill. After that the system's performances are characterized. This work starts from [1]: in this paper a cardboard maze is used to test the mouse behaviour. This setup is quite simple and comes with some drawbacks (i.e. it is not possible to analyse the mouse gait or control its speed and direction), which are addressed by the treadmill design. The new design features closed-loop speed control, a user interface (with real-time plotting), data logging, moreover the system can be expanded easily thanks to the use of MAVLink.

System architecture For the system architecture one μcontroller is used for the closed-loop control. This controller sends and receives data using a USB cable and the UART 232 protocol with the MAVLink messaging protocol. On the other side the PC can get the messages, log them and plot them. To measure the speed of the treadmill two optical sensors are used. These sensors are the same that can be found in gaming mouse. The sensors provide rich information which can be used not only to measure the speed of the surface of the treadmill, but also the estimate the quality of the measure itself. By using this information the control loop is aware of possible measuring problems and can therefore discard low quality information.
Finally, to ease the user experience, a cross-platform graphical user interface as well as documentation, unit tests and a user manual are provided.

Testing and results Once the system is built the performances are verified, more precisely, the sensor noise is analysed as well as the speed of the control loop. To carry out an analysis of the total noise on the sensors (which is due to the sensor itself and the vibrations/imperfections of the machine), the machine is set to a constant speed setpoint, while data are logged. This procedure is repeated for different speeds. Figure 0.1 shows the reference speed as well as the mean, standard deviation (std), median and min/max values of the measured speed for all the setpoints tested. One can notice that it is not possible to reach the last setpoint due to saturation of the control signal. The minimum speed is always around 0 $[\frac{m}{s}]$ due to some imperfections in the moving part that causes it to get stuck periodically. On the other hand the maximum std is < 0.008 $[\frac{m}{s}]$, thus fulfilling the requirement of ≤ 0.02 $[\frac{m}{s}]$. By integrating the speed it is possible to compute the position, doing so leads to a std on the position ≤ 0.045 $[mm]$, fulfilling the requirements. Finally the control loop is fast since it logs the speed measurements with a frequency of 200 $[Hz]$ 97.3% of the time.

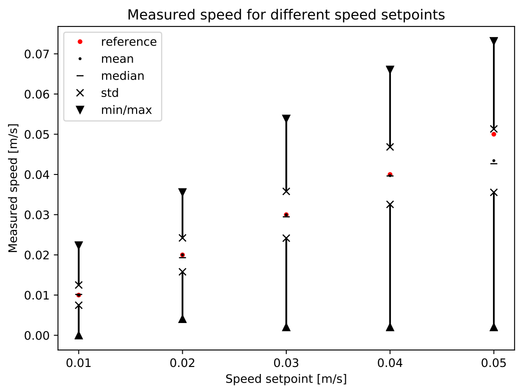


Figure 0.1 – Measured speed as a function of the reference one.

Drawbacks and future improvements As a prof of concept the treadmill seems to be a proper improvement of the previous design since it provides solutions to many of the drawbacks suffered by the cardboard maze. On the other hand not all the requirements are fulfilled yet (e.g. maximum speed), some improvements may include: new motors, which can provide the required speed and torque for better tracking and a camera to get information on the mouse position so that experiments with a free moving mouse can be performed as well. The main drawbacks of this design are the increased complexity, the higher cost and the need of a fixed-head mouse. No need to underline that the much richer information that can be retrieved using the new design as well as the possibility to solve the issues mentioned above in future iterations of this design outfaces all the drawbacks.

Contents

1 Introduction

In this section the main objectives and the state of the art for the project are presented as well as the overall structure of this report.

1.1 Motivation

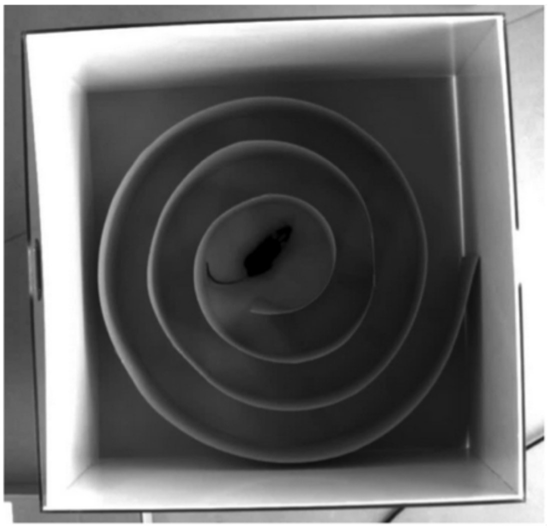


Figure 1.2 – The experimental setup used in [1].

The studies on mammal locomotion have driven more and more attention over the years, and especially experiments on mice, such as [1] or [2], have enhanced our understanding of the neuronal circuits that enable locomotion. The experimental setup in [1], is quite rudimental. As shown in 1.2 it only consist in a spiral maze made out cardboard. This setup comes with some advantages such as:

- Low price
- Simple to implement and use
- Untrained mice can be employed
- Free moving mouse

As well as some disadvantages:

- Impossibility to analyse the mouse gait
- The mouse movements can't be imposed

A more advanced platform is used in [2]. In this case a rotating headpost allows 2-photon imaging in freely locomoting and rotating mice. This means that the measuring apparatus is fixed to the mouse head, while the mouse os free to move at his will. With such a setup it is possible to analyse the mouse gait, but it is not possible to control it, thus it is not the correct approach to use in the new design.
To asses the issues mentioned above a new design is needed for conducting such experiments. The new platform needs to allow the control on the walking surface on which the mouse is standing in such a way that a specific speed profile can be imposed to the mouse. Moreover it must be possible to analyse the mouse gait using cameras.
For the new design inspiration is taken from some existing solutions on the market.

1.2 Requirements

First the mechanical requirements are discussed and stated. Table 1 summarizes them.

Description	**Value**	**Unit**
Dimensions of the moving surface	0.5	$[m^2]$
Course	∞	$[m]$
Maximum speed	3	$[\frac{m}{s}]$
Maximum acceleration	2	$[\frac{m}{s^2}]$
Position resolution	0.01	$[m]$
Speed resolution	0.02	$[\frac{m}{s}]$
Maximum weight	0.1	$[kg]$
Mounting time for 1 person	30	$[min]$
Maximum weight of the mouse	40	$[g]$
Length of common experiment (distance, time)	(20, 600)	$([m],[s])$

Table 1 – Summary of the requirements for the mouse treadmill platform.

The functional requirements are listed as well:

- **Closed-loop control** The user can specify a 2D speed setpoint, the control is then able to measure the speed of the treadmill surface and adjust the motor signals to reach the desired setpoint.
- **Speed routines** The user can define a speed routine, which needs to be executed by the treadmill. The speed routine consists of a list of 2D speed setpoints and the time interval during which the machine should execute them.
- **User interface** The user can use a graphical user interface (GUI) on a computer to be able to use the mouse treadmill. This interface informs the user if the sensors are correctly connected and initialized, and it should give a live update of the treadmill speed.
- **Data logging** The user can save the data sent by the treadmill during the experiment for future uses.
- **Expandability of the system** The user can easily expand the system with other controllers to have other features, than the ones listed above.

1.3 Structure of the report

This report is structured as follows: an introduction is given in section 1. Section 2 describes the design decisions and the components choices made. Section 3 describes the control strategy. Finally in section 4 the conclusion of the project is given. After that in section 5 the user manual for mouse treadmill is given. The code and the data-sheets of the components are annexed. All the work done on the project can be downloaded from `https://github.com/DidierNegretto/3DMouseTreadmill`.

2 Design choices

In this section the design choices are explained and justified. First an overview of the system architecture is given, then choice of the board and the sensors is analysed and finally the calculations for the motor dimensioning are shown.

2.1 System architecture overview

The overview of the system is given in figure 2.3.

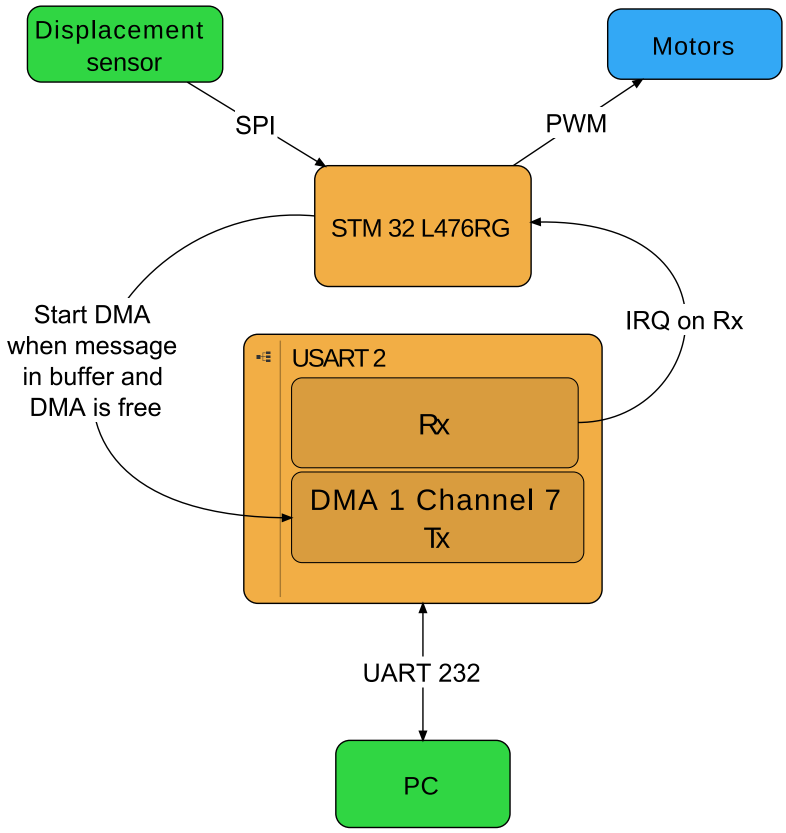


Figure 2.3 – Architecture for mouse treadmill project

The core of the system is the STM32L476RG, which can read from the sensors using the SPI interface and control the motors using PWM. Moreover it can communicate with the computer and the GUI for data logging and to receive the inputs from the user. The communication with the computer uses the DMA capabilities of the microcontroller to free the processor from waiting for the communication to end before being able to take care of other tasks. More detailed informations on how the treadmill works and can be used are given in section 5.

2.2 Board

For the board choice different types are taken into account:

- **Single board computer**: In this category the raspberry pi and the odroid are taken into consideration. These boards offer powerful computers, which can be running operating systems such as Linux or Windows, which makes them interesting. Unfortunately they can't provide any accurate timing, which is needed for the motor control and PWM generation.

- **Evaluation boards**: In this category the STM32 nucleo boards as well as the arduino boards can be found. These boards allow proper timing of the signals and accurate PWM generation, but on the other hand a computer is needed for plotting and storing the data, which can't be done locally on the board due to memory restrictions and limited resources available.

Due to the constraints in the system the second category is consider for implementation, the STM32L476RG board is taken for the system. Table 2 summarizes the features of the board.

Description	**Value**	**Unit**
Architecture	ARM-Cortex 32-bit with FPU	–
Clock frequency	80	$[MHz]$
Flash memory	1	$[MB]$
RAM memory	128	$[KB]$
I2C interfaces	3	–
USART interfaces	5	–
SPI interfaces	3	–
DMA controller	14	–
Cost	20.58	$[CHF]$

Table 2 – STM32L476RG main features.

One of the most important feature of the board is the DMA, which enhances the performances of the CPU. The DMA is used for the UART communication with the computer. This technique frees the CPU from waiting for the UART communication to be finished, so that it can spend more time on other activities. This same solution can be, in principle, adopted for the SPI communication if a standard SPI is used. Unfortunately the timing diagrams for the sensors are not standard, thus some time needs to be "wasted" by the processor so that the sensors can keep up with the communication. Other interesting features are: the big flash memory, the good RAM memory and the low cost. One drawback is that dynamic memory allocation is not possible in such an small system to prevent stack overflow and problems during run time. This is why the size of the speed routine is limited to a given number of points. Finally the multiple serial interfaces allow the possibility to expand the system to a bigger one with more μcontrollers involved.

2.3 Communication

For the communication with the computer the UART 232 protocol is chosen. This choice is almost mandatory since most boards are provided with an UART to USB interface and a mini-USB connector. The STM32L476RG is no exception to this rule. This protocol comes with the advantage that can be used to communicate with most of the PCs, but it comes with limited baud rate. The main settings for the UART protocol are reported in table 3.

Figure 2.4 – MAVLink logo

Parameter	**Value**	**Unit**
Baud rate	230400	$[\frac{Bits}{s}]$
Word length	8	$[Bits]$
Parity	None	–
Stop bits	1	$[Bits]$
MSB first	Disable	–

Table 3 – Table describing the main parameters of the UART communication protocol.

Since the system needs to be expanded for future more complex experiments some thought is put in the choice of the messaging protocol to allow this key feature. The best solution found is

MAVLink. "MAVLink is a very lightweight messaging protocol for communicating with drones"[3], one can say that the mouse treadmill is not meant to fly around, but this messaging protocol is flexible enough to be adapted to the mouse treadmill. More precisely a dialect is described in A, and summarized in table 4. Thanks to the description file (A) it is possible to generate libraries in different programming languages (C, Python, Java, ...) and if in the future a new message is required an additional definition can be added to the file and the libraries can be regenerated.

Despite the light weight MAVLink comes with some interesting features, such is high reliability (detects packets drops and corruption), high efficiency (only 14-bits of overhead), it can also allow up to 255 concurrent systems on the network. Thus it looks perfect for the expandability requirement.

Name	**Description**	**Sender**	**Receiver**	**Type**
HEARTBEAT	Verifies communication	STM32	PC	Status
SPEED_INFO	Measured speed	STM32	PC	Info
SPEED_SETPOINT	Speed setpoint	PC/STM32	STM32/PC	Status
MODE_SELECTION	Changes mode	PC	STM32	–
MOTOR_SETPOINT	Up time of PWM duty cycle	STM32	PC	Info
POINT_LOADED	Acknowledge for routine point loaded	STM32	PC	–
POINT	Information for one point of the routine	PC	STM32	–
ERROR	Error message	STM32	PC	–
RAW_SENSOR	Raw sensor values	STM32	PC	Status

Table 4 – List and description of the MAVLink messages. The Type indicates whatever the message is high frequency (Info), low frequency (Status) or none of the previous ones (–)

2.4 Sensor

For sensing the speed of the wheel a contactless solution is chosen. To achieve this goal a optical gaming mouse sensor is taken. Another criterion is that the sensor needs to come mounted on a PCB with a simple interface to reduce the time needed to design and manufacture the machine. Because of that the PMW3360 is chosen for the implementation. The working principle of the sensor is quite simple. The sensor is equipped with a LED to light a given area and a camera. The camera takes picture of the moving surface with a frequency of up to 12000 $[fps]$. Using the integrated DSP module some features are extracted form the images and, by knowing the displacement of the features, it is possible to determine how much the surface has moved on the X and Y direction. Some other useful information can be retrieved from the sensor such as :

- **Lift status** This bit in the motion register gives information about the status of the sensor and especially if the sensor detects a surface or not. This information is used to determine if the read value is valid or not.
- **Surface quality (SQUAL)** This register gives an information about how many features are detected on the surface. This value is used to verify the quality of the measurement, which is considered valid only if the number of detected features is above a given threshold.

- **SROM ID** This value is read after the power up of the sensor to verify that the SROM of the sensor is uploaded correctly using the SPI interface. If this value is not as expected it means that the sensor is not initialized correctly and thus might not work properly.

The specifications of the sensor are summarized on table 5. For more details refer to E.1.

Description	**Value**	**Unit**
High speed detection	6.3	$[\frac{m}{s}]$
High acceleration detection	490	$[\frac{m}{s^2}]$
Default resolution	0.00508	$[mm]$
Resolution error of	1	[%]
4 wires SPI interface	1	–
Cost	29.99	[$]

Table 5 – PMW3660 main features.

2.5 Motor

In this section one motor proposition B [1] is shown with all the calculations used to justify such a choice. To properly dimension the motors these assumptions are taken:

1. $\eta = 1$ No losses in wheel-sphere coupling
2. No slip of the wheel on the sphere
3. Hollow sphere
4. Flat disk
5. Negligible rotor and gearbox inertia

The data given are:

- $m_s = 2$ $[kg]$ mass of the sphere
- $r_s = 0.2$ $[m]$ radius of the sphere
- $m_w = 0.0114$ $[kg]$ mass of the wheel
- $r_w = 0.03$ $[m]$ radius of the wheel
- $M_{max} = 0.11$ $[Nm]$ maximum torque provided by the motor-gearbox
- $\omega_{max} = 1000$ $[rpm]$ maximum angular speed of the motor-gearbox

It is therefore possible to estimate the maximum continuous acceleration and speed of the sphere. The maximum continuous speed can be computed using:

$$v_{max} = \omega_{max} \frac{r_s}{60} = 3.16[\frac{m}{s}] \tag{1}$$

For the acceleration first the inertia of the wheel J_w and of the sphere J_s can be computed using:

$$J_w = \frac{1}{2} m r_w^2 \tag{2}$$

$$J_s = \frac{2}{3} m r_s^2 \tag{3}$$

3 Control

In this section the main aspect of the control are discussed as well as some results. For the closed-loop control a simple PI controller is used (see 3.2). This can be improved in future works to allow for faster and better performance control. The implementation of the controller is done in CodeSTM32/mouseDrive.c in the function void mouseDriver_control_idle(void).

[1]This motor are not used in the actual version of the treadmill, but might be used for a future iteration.

3.1 Inputs/Outputs

In this section the signal definitions are described. The control diagram is shown in figure 3.5. The signals are defined as follow:

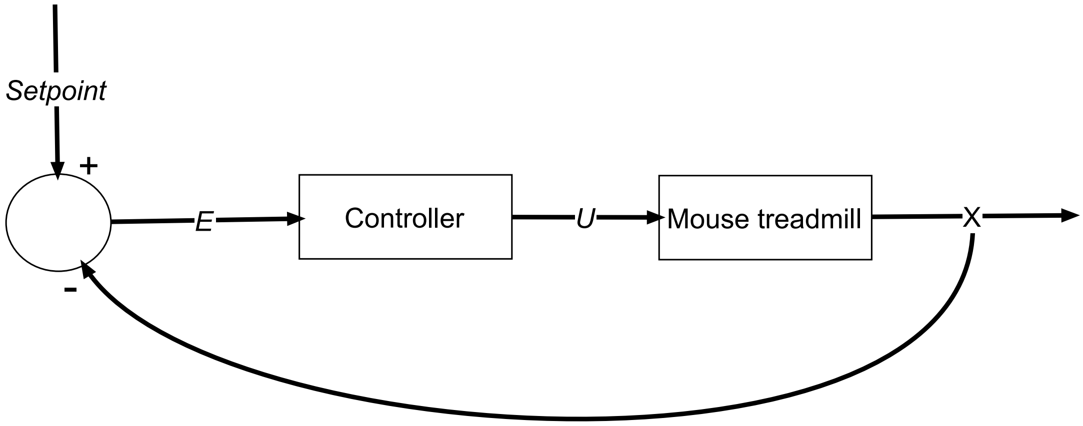


Figure 3.5 – Control diagram

- $X = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$: is the measured v_x and v_y speeds. This measure is done using the optical sensors, which means that the raw values are as defined in the datasheet of the sensors (see E.1). In short words the sensor runs a navigation program, which does the correlation between two images as fast as possible and finds out of how many counts the two images are displaced in the x and y direction of the sensor, those values are then integrated up to when the motion burst is performed. Since two sensors are used, only one axis of each sensor is meaningful for the control (since the other one is always fixed). The information that can be retrieved from one sensor is the number of counts along one axis that the ball has done since the last read. This can then be translated in to meters by knowing the resolution of the sensor (which is given in counts per inch) and the relation between inches and meters. Furthermore the speed can be computed by keeping track of when the last measure is taken, and the actual time, thus knowing the dt between two measures.
- $E = \begin{bmatrix} e_x \\ e_y \end{bmatrix}$: is the error id est the difference between the setpoint and the measured speed
- $U = \begin{bmatrix} u_x \\ u_y \end{bmatrix}$: is the control signal, which is the up time of the duty cycle of the PWM signal controlling the X and Y directions. The parameters of this signal can be modified by using the PRESCALER_PWM and COUNTER_PERIOD_PWM. This two values allow for defining the frequency and number of possible values of the PWM signal.

3.2 Controller

In this section the internal structure of the controller is described. The inputs in the controller are the errors on the speed setpoint in X and Y. The control signal is defined as in equation 4.

$$U = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = K * \begin{bmatrix} e_x \\ e_y \end{bmatrix} + I * \begin{bmatrix} i_x \\ i_y \end{bmatrix} \tag{4}$$

Where K and I are constant scalar values and $\begin{bmatrix} i_x \\ i_y \end{bmatrix}$ is a vector containing the sum of the errors over all the past measures where the motor signal U is not the maximum allowed. This condition is taken

to avoid wind up and overshoot in the controller.
Moreover the control is done only if the measures taken are valid. Which means that the SQUAL measure is bigger than SQUAL_THRESH, the sensors are not lifted, and the PRODUCT_ID is equal 66. If those conditions are met it means that the surface quality is good, the sensor "sees" correctly the surface and the communication is done correctly. If the measures are not valid for more than MAX_MISSING_MEASURES the motors are stopped and the mode goes to STOP mode to avoid damage to the machine.

3.3 Results

In this section the results of the control are shown as well as the achieved performances in terms measures 3.3.1 and control 3.3.2.

3.3.1 Measure test

In this section a simple experiment is described and the results are shown. First the zero speed noise is measured in the sensors, then the same measure is repeated while spinning the ball at constant speed. This let us characterize the noise in the sensors and decide whatever some filtering is required.

3.3.2 Control test

In this section the I and K parameters in equation 4 are tuned, then the step responses are measured. Finally a 2D routine is described and the tracking error is analysed to see whatever the machine is able to follow such a routine and with which performances.

4 Conclusion

5 User manual for mouse treadmill software

The software is well documented in the docs folder, nevertheless some important things are pointed out in this report so that the user can more easily install and start using the mouse treadmill. The installation guide for the PC software, a user manual for the GUI, a explanation on how to write a speed routine as well as a guide on how to expand the system with new messages is provided. Note that all the provided commands and instructions are tested for MAC, mavlink is available also for LINUX and WINDOWS, the user can adapt these command to be able to install and successfully use the software on his machine.

5.1 Installation of the PC software

First python 3 needs to be installed, for that see [4]. GIT needs to be install as well. Some other python packages needs to be installed, they can be obtained using PiP. The required ones are:

- pyserial
- os
- sys
- numpy
- appjar
- tqdm
- json
- matplotlib

Make sure that pymavlink is not install. This is important since the dialect used is not a standard one, but it is custom. Do not install pymavlink using PiP.

To install the software the sequent steps have to be accomplished:

1. Clone the git repository of the project using

```
$ git clone https://github.com/DidierNegretto/3DMouseTreadmill.git
```

2. Move inside the repository

```
$ cd 3DMouseTreadmill/
```

3. Make sure no previous version of pymavlink is installed

```
$ pip uninstall pymavlink
```

4. Remove the mavlink directory

```
$ rm -r -f mavlink/
```

5. Clone the mavlink repository

```
$ git clone https://github.com/mavlink/mavlink.git
```

6. Update the submodule

```
$ git submodule update --init --recursive
```

7. Copy mouse.xml file and the mouse.py files into mavlink/pymavlink/dialects/v20

8. Change directory to mavlink/

```
$ cd mavlink
```

9. Export the path to the repository so that python will find all the code it needs to run

```
$ export PYTHONPATH=‘path_to_repository/3DMouseTreadmill/‘
```

10. Change directory to pymavlink

```
$ cd pymavlink
```

11. Setup everything using the setup.py provided

```
$ python3 setup.py install --user
```

5.2 How to use the GUI

In this section the use of the GUI and its functionalities are described. First of all the GUI provided can be expanded using the functions in the mouse.py generated using mavlink, and thus can be improved for future versions of the project. One screenshot of the GUI is shown in figure 5.6. Figure 5.6 is taken on a MAC, the GUI have the same functionalities on other platforms, but it might look different. The library used to design the GUI is Appjar, which is compatible with MAC, LINUX and Windows.

- **A** In this region the content of the HEARTBEAT message are presented. Time is the time from boot of the system in milliseconds. Modes is the mode in which the stm32 is. The mode can be STOP, SPEED AUTO or RUNNING.

- **B** In this region the real-time data are plotted. The top plot shows the X and Y motor signals, the middle one shows the X and Y speed setpoints and the bottom one shows the motor signal. By using the top left buttons it is possible to save the plots and navigate them, it is advisable to store the data and plot them afterwards using another script for better analysis due to the fast update of the plots.

- **C** In this region data from the sensors are displayed. ID is the product ID of the sensor and is used to verify that the communication between the sensor and the board is working correctly. LIFT is 0 if the sensor detects correctly the surface and 1 if it does not. SQUAL is the surface quality information. This value should be greater than 20 for the measure to be good. ROM is the SROM ID of the sensor. This value is used to verify that the SROM is flashed correctly during initialization. If everything is working correctly you should see something like table 6

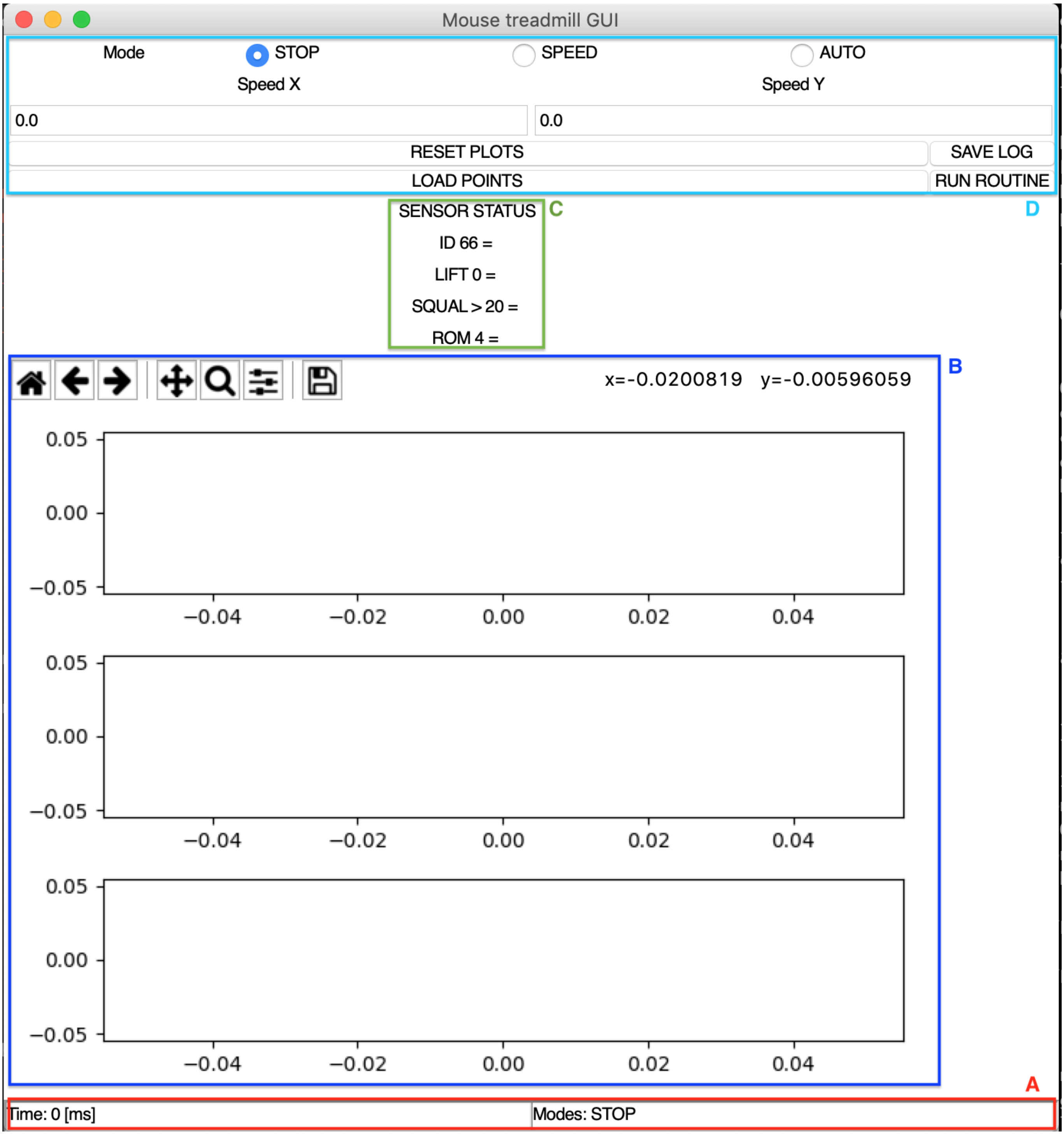


Figure 5.6 – GUI screenshot on a MAC.

SENSOR STATUS
ID 66 = 66 | 66
LIFT 0 = 0 | 0
SQUAL > 20 = 34 | 43
ROM 4 = 4 | 4

Table 6 – Example of GUI output for sensor initialized and connected correctly and detecting a good quality surface. Before = the name of the information and its correct value are shown, after the sensor x | sensor y readings are displayed.

- **D** In this region the input from the user are taken. In the first line the user can select the mode to be used:
  - **STOP**: When this mode is selected the motor are stopped.
  - **SPEED**: When this mode is selected the motor setpoints can be typed in the two entries under the Speed X and Speed Y labels.
  - **AUTO**: When this mode is selected the user can load the points of the routine on the board (Modes: LOAD) and then run the routine (Modes: RUNNING).

  Finally the RESET PLOTS button is used to reset the plot in case of reset on the board, the SAVE LOG button stores all the data received in a file in /log/log.txt. This file is overwritten every time so if you need the data please copy them in a safe place. The LOAD POINTS sends the points defined in routine.py to the board. This will work only if the mode is set to AUTO (should see LOAD in Modes in **A** and the time should not be updated). Finally the RUN ROUTINE button starts the routine on the board (the board goes in mode RUNNING).

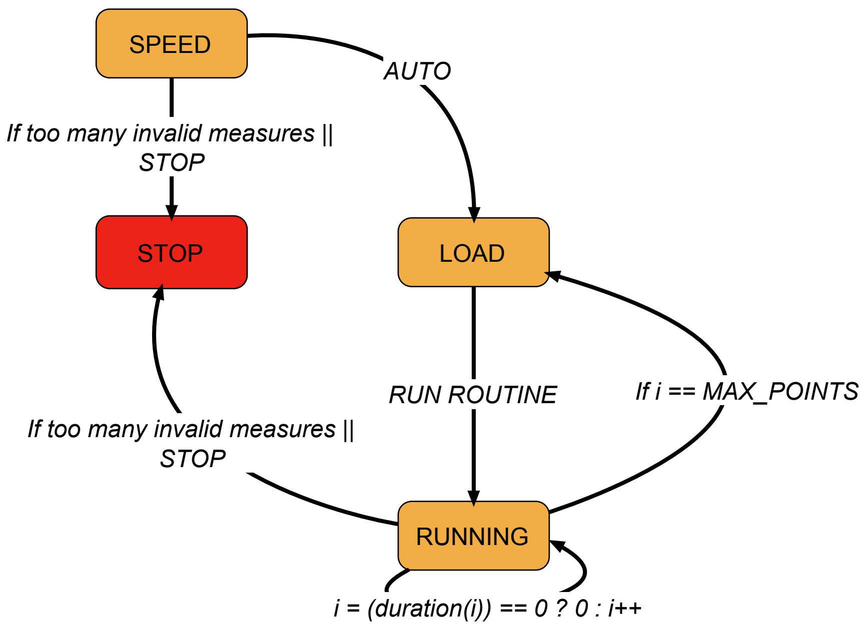


Figure 5.7 – Finite state machine of the mouse treadmill.

The finite state machine of the machine is shown in figure 5.7. All the capital letters conditions (except for MAX_POINTS) are widgets on the GUI that can be pressed by the user while using the machine.

5.3 How to write a routine

In this section the way to properly define a routine is described. An example routine is provided in MouseTreadmillPC/python/routine.py. The routine is a python dictionary containing a list of durations, setpoint_x and setpoint_y. The two setpoints define the desired speed along x and y, while the duration is the time span during which the two setpoints are applied. One should notice that the system time is discrete and increased every millisecond, moreover one should take into account the settling time for the control and the maximum acceleration provided by the motors to do a proper discretization of the desired speed profile.
A duration of 0 means that the end of the routine is reached and the routine is started again at the first point defined. A maximum of 255 points can be defined, if more points are needed the id of the point have to be changed from type uint8_t to uint16_t to allow for IDs above 255. A memory limitation is still present, but for a number of points above 1'000.

5.4 How to extend the system

To extend the system with new messages and features the main operation consist in modifying A. This files describes all the messages and constants used in the communication protocol, thus it possible to add/modify them. If you need to create a new message or constant, please have a look at the already defined ones and use them as a template. To extend the system please follow the following steps:

1. Get the basic system installed correctly, for that see 5.1.
2. Modify A (mouse.xml) as needed.
3. Generate the C libraries for the STM32, for that you need:

   ```
   $ cd 3DMouseTreadmill/mavlink
   $ python 3 mavgenerate.py
   ```

   Now a GUI asking you information appear, this must be filled as follow:

   - **XML** there you indicate the mouse.xml file that was previously modified
   - **Out** there you indicate the 3DMouseTreadmill/MAVLink Library/
   - **Language** Choose C
   - **Protocol** Choose 2.0
   - **Validate** Choose Yes
   - **Validate Units** Choose Yes

   Now you can press on generate. The GUI should be similar to figure 5.8a. If some errors are shown, correct them and try again.
4. Adapt the code in the STM32 project if needed.
5. Generate the python libraries for the PC, for that you need:

   (a) Run mavgenerate.py (if not still running)

   ```
   $ python 3 mavgenerate.py
   ```

   Now a GUI asking you information appear, this must be filled as follow:

- **XML** there you indicate the mouse.xml file that was previously modified
- **Out** there you indicate the 3DMouseTreadmill/mouse.py
- **Language** Choose Python
- **Protocol** Choose 2.0
- **Validate** Choose Yes
- **Validate Units** Choose Yes

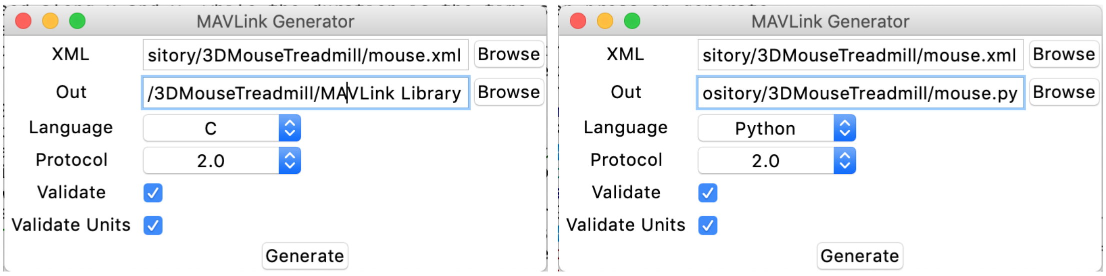
Now you can press on generate. The GUI should be similar to figure 5.8b. If some errors are shown, correct them and try again.

(b) Change directory to the parent one

```
$ cd ../
```

(c) repeat the installation guide (see 5.1) from point 3.

6. Adapt the python code if necessary.



(a) C libraries. (b) python libraries.

Figure 5.8 – mavgenerate GUI screenshots properly setup for generating python and C libraries.

References

[1] Jared M. Cregg, Roberto Leiras, Alexia Montalant, Ian R. Wickersham, and Ole Kiehn, *Brainstem Neurons that Command Left/Right Locomotor Asymmetries*

[2] Jakob Voigts, Mark T. Harnett, *Somatic and Dendritic Encoding of Spatial Variables in Retrosplenial Cortex Differs during 2D Navigation*

[3] MAVLink Developer Guide, *https://mavlink.io/en/*

[4] Python website, *https://www.python.org/downloads/*

List of Figures

List of Tables

A MAVLink dialect description file

```
<?xml version="1.0"?>
<mavlink>
    <version>3</version>
    <dialect>2</dialect>
    <enums>
        <enum name="MOUSE_MODE">
            <description>This enum defines the mode to be used</description>
            <entry value="0" name="MOUSE_MODE_STOP">
                <description>All motion of mouse treadmill is stopped</description>
            </entry>
            <entry value="1" name="MOUSE_MODE_SPEED">
                <description>Constanst speed is applied. Speed selected by PC
                    message SPEED_SETPOINT.</description>
```

```xml
            </entry>
            <entry value="2" name="MOUSE_MODE_AUTO_LOAD">
                <description>Predefined speed profile is loaded</description>
            </entry>
            <entry value="3" name="MOUSE_MODE_AUTO_RUN">
                <description>Predefined speed profile is applied</description>
            </entry>
        </enum>
        <enum name="MOUSE_ERROR">
            <description>This enum defines the possible errors</description>
            <entry value="0" name="MOTOR_ERROR">
                <description>The motor driver flaged an error, this might be due to many sources, see datasheet of motor driver.</description>
            </entry>
            <entry value="1" name="MOTOR_LOW_SPEED">
                <description>The speed setpoint chosen is too low to be achieved.</description>
            </entry>
            <entry value="2" name="MOTOR_HIGH_SPEED">
                <description>The speed setpoint chosen is too high to be achieved.</description>
            </entry>
            <entry value="3" name="MOUSE_ROUTINE_TOO_LONG">
                <description>More than 255 points have been defined in the mouse routine.</description>
            </entry>
            <entry value="4" name="SENSOR_NOT_RESPONDING">
                <description>One sensor is not responding correctly.</description>
            </entry>
        </enum>
            <enum name="SENSOR_ID">
            <description>This enum defines the sensors directions</description>
            <entry value="0" name="SENSOR_X">
                <description>Sensor ID for X direction.</description>
            </entry>
            <entry value="1" name="SENSOR_Y">
                <description>Sensor ID for Y direction.</description>
            </entry>
        </enum>
    </enums>
    <messages>
        <message id="0" name="HEARTBEAT">
            <description>The heartbeat message shows that a system or component is present and responding. Sender = STM32 Receiver = PC
            </description>
            <field type="uint8_t" name="mode" enum="MOUSE_MODE">Actual operating mode</field>
            <field type="uint32_t" name="time">Time from boot of system</field>
        </message>
        <message id="1" name="SPEED_INFO">
```

```xml
            <description>The message giving the actual speed of the motor. Sender = STM32 Receiver = PC
            </description>
            <field type="uint32_t" name="time_x">Time from boot of system for speed_x measure</field>
            <field type="uint32_t" name="time_y">Time from boot of system for speed_y measure</field>
            <field type="float" name="speed_x">Speed in x direction</field>
            <field type="float" name="speed_y">Speed in y direction</field>
            <field type="uint8_t" name="valid">0 if data are not valid, 1 if data are valid </field>
        </message>
         <message id="2" name="SPEED_SETPOINT">
            <description>The message is sent to send and validate the setpoint sent from computer. Sender = PC/STM32 Receiver = STM32/PC
            </description>
            <field type="float" name="setpoint_x">Speed setpoint in x direction</field>
            <field type="float" name="setpoint_y">Speed setpoint in y direction</field>
        </message>
        <message id="3" name="MODE_SELECTION">
            <description>This message is used to select the mode of the STM32 Sender = PC Receiver = STM32
            </description>
            <field type="uint8_t" name="mode" enum="MOUSE_MODE">Actual operating mode</field>
        </message>
        <message id="4" name="MOTOR_SETPOINT">
            <description>This message defines the raw motor input values. This values defines the Duty_Cycle up time for PWM signals. Sender = STM32 Receiver = PC
            </description>
            <field type="uint32_t" name="time">Time from boot of system</field>
            <field type="float" name="motor_x">Speed setpoint in x direction</field>
            <field type="float" name="motor_y">Speed setpoint in y direction</field>
        </message>
        <message id="5" name="POINT_LOADED">
            <description>This message is used to acknowledge the receipt of one point for auto mode Sender = STM32 Receiver = PC
            </description>
            <field type="uint16_t" name="point_id">Last ID of point loaded</field>
        </message>
        <message id="6" name="POINT">
            <description>This message is used to send one point for auto mode. Sender = PC Receiver = STM32
            </description>
            <field type="uint32_t" name="duration">Time during which the setpoint need to be kept</field>
            <field type="uint16_t" name="point_id">point ID</field>
```

```
            <field type="float" name="setpoint_x">Speed setpoint in x direction</field>
            <field type="float" name="setpoint_y">Speed setpoint in y direction</field>
        </message>
        <message id="7" name="ERROR">
            <description>This message is used to send errors Sender = STM32 Receiver = PC
            </description>
            <field type="uint32_t" name="time">Time from boot of system</field>
            <field type="uint8_t" name="error" enum="MOUSE_ERROR">error ID</field>
        </message>
        <message id="8" name="RAW_SENSOR">
                <description>This message contains raw sensor values Sender = STM32 Receiver = PC
                </description>
                <field type="uint32_t" name="time">Time from boot of system</field>
                <field type="uint8_t" name="sensor_id">0 for X, 1 for Y.</field>
                <field type="int16_t" name="delta_x">Displacement along sensor's x in counts per inch.</field>
                        <field type="int16_t" name="delta_y">Displacement along sensor's y in counts per inch.</field>
                        <field type="uint8_t" name="squal">Quality of the surface. For white paper is around 30.</field>
                        <field type="uint8_t" name="lift">1 if the sensor is lifted (not measuring). 0 otherwise</field>
                <field type="uint8_t" name="product_id">0x42 if the serial communication with the sensor works correctly.</field>
            <field type="uint8_t" name="srom_id">0x00 if initialisation is not done. Other value if done correctly.</field>
            </message>
    </messages>
</mavlink>
```

B Motor proposition

maxon

Configured drive

Motor - ECXSP16M BL KL A STD 24V
Gearhead - GPX16 SP STE 44:1

Part number: B7A31C479448 Revision number 2

Orders are processed and shipped from Switzerland within 11 working days.
General Terms and Conditions: https://www.maxongroup.ch/maxon/view/content/terms_and_conditions_page

To open the integrated CAD file, please save this document and open it in Acrobat Reader. The STEP file is available after a double-click on the pin icon.

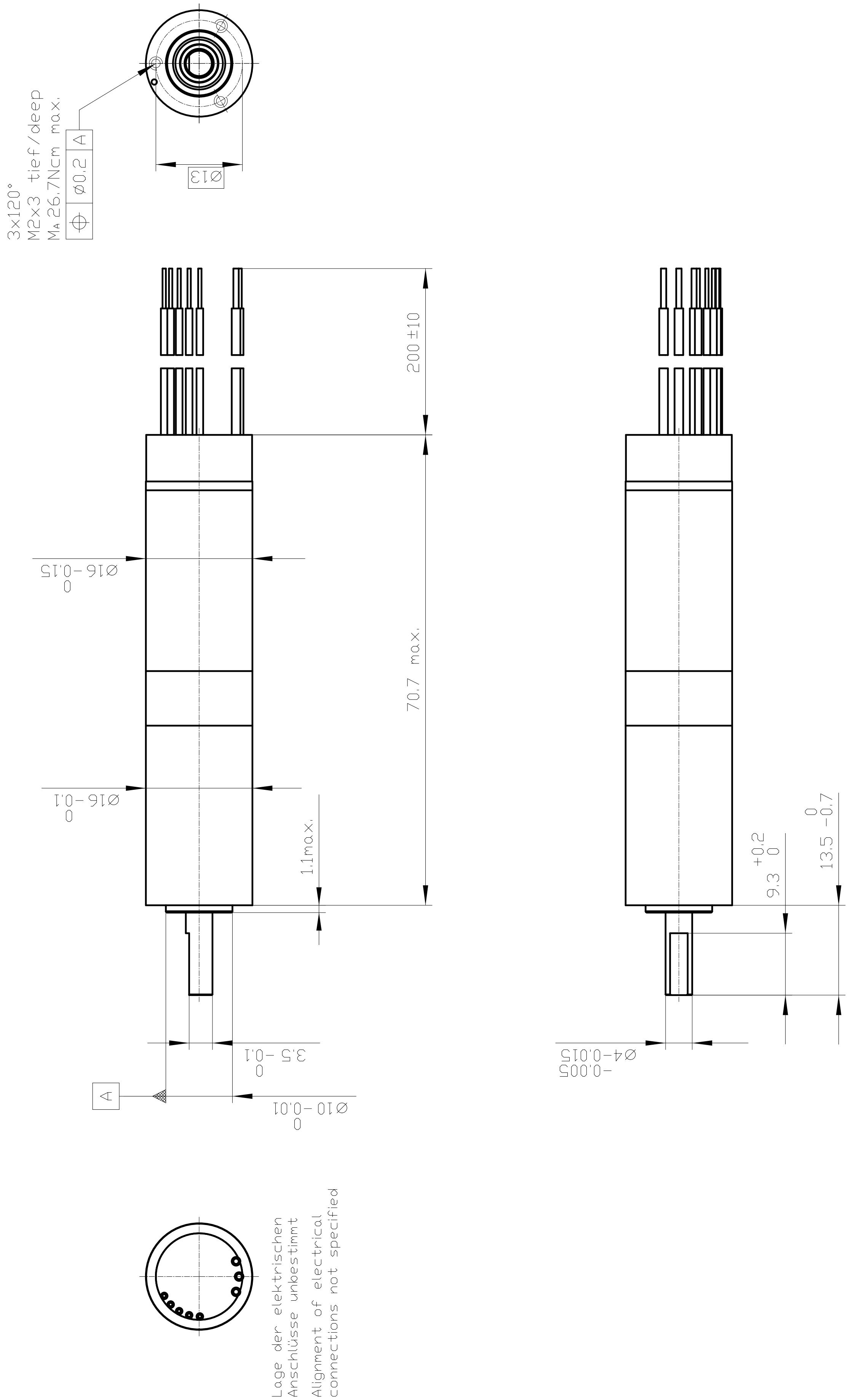
B7A31C479448.stp (STP AP 214)

Open configuration: https://www.maxongroup.com/maxon/view/configurator/?ConfigID=B7A31C479448

maxon

Axial play gearhead: 0...0.1mm

Motor - ECXSP16M BL KL A STD 24V
Gearhead - GPX16 SP STE 44:1
Drawing not to scale!



Unit of measure: mm
ISO 5456-1
ISO 1101
ISO 965-1
ISO 2768-m
ISO 8015

Motor (Cable type: AWG22)

Red	Motor 1
Black	Motor 2
White	Motor 3

Hall-Sensor (Cable type: AWG26)

Orange	V_{hall} 3…24V
Blue	GND
Yellow	Hall-Sensor 1
Brown	Hall-Sensor 2
Gray	Hall-Sensor 3

maxon

Summary of your selected configuration

Total weight of the drive: 105.6 g

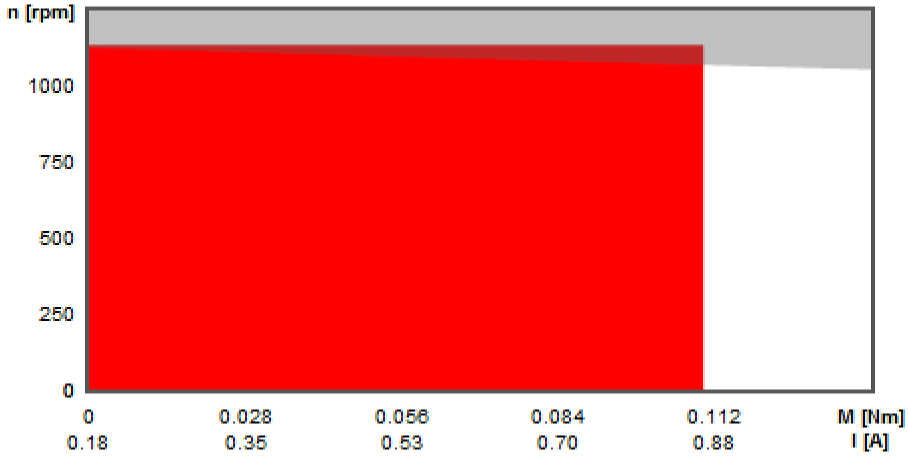
	ECXSP16M BL KL A STD 24V	
Product detail		
	Commutation	With Hall sensors
	Nominal voltage	24 V
	Motor bearings	Preloaded ball bearing
	Version	Standard

	GPX16 SP STE 44:1	
Product detail		
	Reduction	44
	Number of stages	2

Legend for part designation

EB	Precious metal brushes	GB	Graphite brushes	CLL	Spark suppression	BL	Brushless
A	Hall sensors	B	Sensorless	KL	Ball bearings	SL	Sintered bearings
GPX	Planetary gearhead	ENX	Encoder	ENC	Encoder	IMP	Pulses
ST	Number of stages	HP	High Power	S/M/L	Short/medium/long	HS	High Speed
STE	Sterilizable	INT	Integ.	STD	Standard	SP	Speed
ABS	Absolute	LN	Reduced noise level	A	Standard	LZ	Reduced backlash
C	Ceramic bearing			STEC	Sterilizable, Ceramic bearing		

Selected operating point



Continuous operation range

Continuous operation range

Selected operating point

Out of voltage range

Not recommended range

Continuous operation range at reduced thermal resistance Rth2 50%

ECXSP16M BL KL A STD 24V

Product specification

Values at nominal voltage	
Nominal voltage	24 V
No load speed	49600 min^{-1}
No load current	177 mA
Nominal speed	45300 min^{-1}
Nominal torque (max. continuous torque)	4.93 mNm
Nominal current (max. continuous current)	1.24 A
Stall torque	63.2 mNm
Stall current	13.9 A
Max. efficiency	79.1 %

Characteristics	
Max. output power continuous	23.5 W
Terminal resistance phase-to-phase	1.73 Ω
Terminal inductance phase-to-phase	0.0893 mH
Torque constant	4.55 mNm A^{-1}
Speed constant	2100 min^{-1} V^{-1}
Speed/torque gradient	797 min^{-1} mNm^{-1}
Mechanical time constant	6.73 ms
Rotor inertia	0.806 gcm^2

Thermal data	
Thermal resistance housing-ambient	20.3 KW^{-1}
Thermal resistance winding-housing	1.52 KW^{-1}
Thermal time constant of the winding	1.83 s
Thermal time constant of the motor	508 s
Ambient temperature	-20…100 °C
Max. winding temperature	125 °C

Mechanical data	
Max. permissible speed	55000 min^{-1}
Axial play	0…0.29 mm
Preload	1.5 N
Direction of force	Zug
Radial backlash	0 mm
Max. axial load (dynamic)	1.5 N

maxon

Max. force for press fits (static)	60 N
Static, supported shaft	2500 N
Max. radial load 5 mm from flange	10 N
Measurement from the flange	5 mm

Further specifications	
Number of pole pairs	1
Number of phases	3
Typical noise level	50 dBA (50000 min^{-1})
Typical noise level at reference speed	50000 min^{-1}
Number of autoclave cycles	0

Information about motor data. https://www.maxongroup.com/medias/CMS_Downloads/DIVERSES/12_137_EN.pdf

GPX16 SP STE 44:1

Product specification

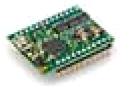
Gearhead data		
	Reduction	44:1
	Absolute reduction	4356/100
	Number of stages	2
	Max. continuous torque	0.11 Nm
	Max. intermittent torque	0.14 Nm
	Direction of rotation, drive to output	=
	Max. efficiency	80 %
	Average backlash no-load	1.6 °
	Mass inertia	0.014 gmc^2
	Max. transmittable power (continuous)	21 W
	Max. short-time transferable output	25 W

Technical data		
	Output shaft bearing	Wälzlager
	Max. radial play, 5 mm from flange	max. 0.1 mm
	Axial play	0…0.1 mm
	Max. permissible radial load, 5 mm from flange	35 N
	Max. permissible axial load	30 N
	Max. permissible force for press fits	100 N
	Max. continuous input speed	50000 min^{-1}
	Max. intermittent input speed	70000 min^{-1}
	Recommended temperature range	-10..135 °C

Information about gearhead data: https://www.maxongroup.com/medias/CMS_Downloads/DIVERSES/12_203_EN.pdf

maxon

ESCON Module 24/2



Product specification

Motor		
	DC motors up to	48 W
	EC motors up to	48 W
Sensor		
	Without sensor (DC motors)	Yes
	Sensorless (EC motors)	
	Digital incremental encoder (2-channel, single-ended)	Yes
	Digital incremental encoder (2-channel, differential)	Yes
	Digital incremental encoder (3-channel, differential)	Yes
	Digital Hall sensors (EC motors)	Yes
	SSI absolute encoder	
	Analog incremental encoder (2-channel, differential)	
Operating modes		
	Current controller	Yes
	Speed controller (open loop)	Yes
	Speed controller (closed loop)	Yes
	Positioning controller	
Electrical data		
	Operating voltage VCC	10..24 VDC
	Logic supply voltage VC optional	.. VDC
	Max. output voltage (factor * VCC)	0.98 *
	Max. output current Imax	6 A
	Max. duration of peak output current Imax	4 s
	Continuous output current Icont	2 A
	Switching frequency of the power stage	53.6 kHz
	Sampling rate, PI current controller	53.6 kHz
	Sampling rate, PI speed controller (closed loop)	5.36 kHz
	Sampling rate, PID positioning controller	kHz
	Max. efficiency	92 %
	Max. speed (DC motors)	150000 min-1
	Max. speed (1 pole pair), block commutation	150000 min-1
	Max. speed (1 pole pair), sinusoidal commutation	min-1
	Built-in motor choke per phase	uH
Inputs		
	Hall sensor signals	H1, H2, H3
	Encoder signals	A, A\, B, B\
	Max. encoder input frequency	1 MHz
	Digitale Eingänge	2

maxon

	Functionality of digital inputs	Enable, enable CW, enable CCW, enable CW+CCW, enable + direction of rotation, stop, PWM set value, RC Servo set value, fixed set value
	Analog inputs	2
	Resolution, range, circuit	12-bit, -10…+10V, differential
	Functionality of inputs	Set value, current limit, offset, speed ramp
	Potentiometers	
	Functionality of the potentiometers	
	DIP switch	
	Functionality of the DIP switch	
Outputs		
	Digital outputs	2
	Functionality of digital outputs	ready, speed comparator, current comparator, commutation frequency
	Analog outputs	2
	Resolution, range	12-bit, -4…+4V
	Functionality of analog outputs	current monitor,speed monitor, temperature, fixed value
Voltage outputs		
	Hall sensor supply voltage	+5 VDC, max. 30 mA
	Encoder supply voltage	+5 VDC, max. 70 mA
	Auxiliary output voltage	+5 VDC, max. 10 mA
	Output voltage (reference)	
Ambient conditions		
	Temperature – operation	-30.. 60 °C
	Temperature – storage	-40 .. 85 °C
	Temperature – extended range	+60…+80 °C, Derating: -0.1 A/°C
	Humidity (non-condensing)	5 .. %
Mechanical data		
	Weight	7 g
	Dimensions (L x W x H)	35.6 x 26.7 x 12.7 mm
	Mounting	mountable on socket terminal strips pitch 2.54 mm

C Code for STM32 NUCLEO 64 board

C.1 Main

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "mouseDriver.h"
#include "mavlink.h"
/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */
/**
 * A structure to represent one sensor
 */
typedef struct SENSOR{
  /*@{*/
  GPIO_TypeDef * cs_port; /**< the chip select port for the sensor */
  uint8_t cs_pin; /**< the chip select pin for the sensor */
  GPIO_TypeDef * pw_port; /**< the power port for the sensor */
  uint8_t pw_pin; /**< the power pin for the sensor */
  uint8_t status; /**< the sensor status. This is the SROM_ID after the upload of the
  firmware. This value should not be 0 otherwise the upload of the SROM is failed. */
  /*@}*/
```

```c
} sensor_t;
/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */
/*!
\fn main_transmit_buffer(uint8_t *outBuffer, uint16_t msg_size)
\param outBuffer buffer to be transmitted over UART
\param msg_size size of the buffer
\brief This function sends the buffer using UART.

\attention The transmission is done using a DMA. Before sending a message
it is important to check that the previous one has been sent. This can be done
using \ref main_get_huart_tx_state .
*/
void main_transmit_buffer(uint8_t *outBuffer, uint16_t msg_size);
/*!
\fn main_stop_motors()
\brief This function stops the motors

The PWM duty cycle is set to 0% for the two motors
\note The PWM duty cycle is represented by a uint type.
The min/max of that value are defined by how the timer is
setup in the microcontroller. The max value can be limited
by limitations in the motors or in the mechanical build of the
machine
*/
void main_stop_motors(void);
/*!
\fn main_set_motors_speed(mavlink_motor_setpoint_t motor )
\param motor PWM duty cycle for the two motors
\brief This sets the motor duty cycle to one specified in the
motor parameter

The PWM duty cycle is set to 0% for the two motors
\note The PWM duty cycle is represented by a uint type.
The min/max of that value are defined by how the timer is
setup in the microcontroller. The max value can be limited
by limitations in the motors or in the mechanical build of the
machine
*/
void main_set_motors_speed(mavlink_motor_setpoint_t motor );
/*!
\fn main_get_huart_tx_state()
```

```
\return the HAL_state of UART transmit
\brief Function used to verify if the channel for writing the buffer is available or busy.
*/
int main_get_huart_tx_state(void);
/*!
\fn main_write_sensor(sensor_t sensor, uint8_t adress, uint8_t data)
\param sensor sensor to which we want to write
\param adress adress of the register to be modified
\param data data to written in the given sensor and register
\brief This function writes a byte in a given register of a given sensor.

\note The writing is done by generating proper signals in the pins. For more details
on the sensor register and timing diagrams see resources/sensorDatasheet.pdf
*/
void main_write_sensor (sensor_t sensor, uint8_t adress, uint8_t data);
/*!
\fn main_read_sensor(sensor_t sensor, uint8_t adress)
\param sensor sensor from which we want to read
\param adress adress of the register to be read
\return the value in the given register and sensor
\brief This function reads a byte in a given register of a given sensor.

\note The reading is done by generating proper signals in the pins. For more details
on the sensor register and timing diagrams see resources/sensorDatasheet.pdf
*/
uint8_t main_read_sensor (sensor_t sensor, uint8_t adress );
/*!
\deprecated
\fn main_transmit_spi(uint8_t data)
\param data data to be transmitted on the spi2
\brief This function transmit one byte on the spi2
*/
void main_transmit_spi(uint8_t data);
/*!
\fn main_wait_160us()
\brief function used to wait around 160 [us].
\note the wait is achieved by toggling the green LED.
*/
void main_wait_160us(void);
/*!
\fn main_wait_20us()
\brief function used to wait around 20 [us].
\note the wait is achieved by toggling the green LED.
*/
void main_wait_20us(void);
/*!
\fn main_write_sensor_burst(uint8_t data)
\param data by to be written during the burst
\brief function used during a write burst
\attention Use this function only during a burst write.
*/
void main_write_sensor_burst(uint8_t data);
/*!
\fn main_read_sensor_motion_burst(uint8_t *data )
\param data pointer on a table of uint8_t used to store the
data read from a motion read burst
\brief function used to do a burst read for the motion read burst
as specified in resources/resources/sensorDatasheet.pdf

\attention Use this function only during a motion read burst.
```

```c
\note The data received from the motion read burst are raw datas and have
to be treated to obtain meaningfull values and verify that the sensor is not
lifted and the surface quality is good enough to consider the measure as valid.
*/
void main_read_sensor_motion_burst(uint8_t *data );
/*
 * PW_0 is power pin for sensor X (PB_0)
 * PW_1 is the power pin for sensor Y (PA_4)
 * CS_0 is the chip select for sensor X (PC_0)
 * CS_1 is the chip select for sensor Y (PC_1)
 */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#define DT_HEART 200
#define PRESCALER_HEART 1000
#define CLOCK_FREQ 80000000
#define COUNTER_PERIOD_HEART ((CLOCK_FREQ/(PRESCALER_HEART))*0.001*DT_HEART)
#define PRESCALER_PWM 1000
#define COUNTER_PERIOD_PWM 255
#define PULSE_PWM 10
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define CS_0_Pin GPIO_PIN_0
#define CS_0_GPIO_Port GPIOC
#define CS_1_Pin GPIO_PIN_1
#define CS_1_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define PW_1_Pin GPIO_PIN_4
#define PW_1_GPIO_Port GPIOA
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
#define PW_0_Pin GPIO_PIN_0
#define PW_0_GPIO_Port GPIOB
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
```

```
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/*!
\def TIMEOUT
\brief Constant used as timeout in ms.
\deprecated Using DMA makes the transfer free from the processor, thus the
TIMEOUT never appens.
*/
#define TIMEOUT 2
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim7;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_tx;
```

```

/* USER CODE BEGIN PV */
/*!
\var inByte
\brief Buffer for one byte.

This is the buffer used to copy data form UART. When one byte is available it is stored in
inByte and then parsed using the mavlink_parse_char function. Everytime one
byte arrives the inByte variable is overwritten.
*/
static uint8_t inByte = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
    */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM7_Init(void);
static void MX_TIM1_Init(void);
static void MX_DMA_Init(void);
static void MX_SPI2_Init(void);
/* USER CODE BEGIN PFP */
void main_wait_160us(void){
  int i = 0;
  i = 0;
  while(i<900){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    i++;
  }
}
void main_wait_20us(void){
  int i = 0;
  i = 0;
  while(i<185){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    i++;
  }
}
/*!
\fn main_wait_1us(void)
\brief Function for waiting approximately one microsecond
*/
void main_wait_1us(void){
  int i = 0;
  i = 0;
  while(i<25){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    i++;
  }
}
int main_get_huart_tx_state(void){
  return (HAL_DMA_GetState(&hdma_usart2_tx));
}
void main_transmit_buffer(uint8_t *outBuffer, uint16_t msg_size){
  HAL_UART_Transmit_DMA(&huart2, outBuffer,msg_size);
}
void main_stop_motors(void)
{
  HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
```

```
  HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
}
void main_set_motors_speed(mavlink_motor_setpoint_t motor )
{

  htim1.Instance->CCR1 = motor.motor_x;
  htim1.Instance->CCR2 = motor.motor_y;

  if (motor.motor_x == 0)
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
  else
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

  if (motor.motor_y == 0)
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
  else
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);

}
uint8_t main_read_sensor (const sensor_t sensor, uint8_t adress ){
  uint8_t value = 0;
  uint8_t adress_read = adress & 0x7F;

  HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_RESET);
  HAL_SPI_Transmit(&hspi2, &adress_read, 1, 100);
  main_wait_160us();
  HAL_SPI_Receive(&hspi2, &value, 1, 100);
  main_wait_1us();
  HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_SET);
  main_wait_20us();
  return (value);
}

void main_write_sensor (const sensor_t sensor, uint8_t adress, uint8_t data){
  uint8_t value = data;
  uint8_t adress_write = adress | 0x80;
  uint8_t pack[2];
  pack[0] = adress_write;
  pack[1] = value;

  HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_RESET);
  HAL_SPI_Transmit(&hspi2, pack, 2, 10);
  main_wait_20us();
  HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_SET);
  main_wait_160us();
  main_wait_20us();
}
void main_write_sensor_burst(uint8_t data){
  HAL_SPI_Transmit(&hspi2, &data, 1, 10);
  main_wait_20us();
}
void main_read_sensor_motion_burst(uint8_t *data ){
  HAL_SPI_Receive(&hspi2,data,12,100);
  main_wait_1us();
}
void main_transmit_spi(uint8_t data){
  uint8_t data_out = data;
  HAL_SPI_Transmit(&hspi2, &data_out, 1, 10);
}
/* USER CODE END PFP */
```

```

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/*!
\fn TM7_IRQHandler(void)
\brief Handle for IRQ of Timer 7

Timer 7 is used to generate a periodic interrupt to send status messages.
Those messages give information about the status of the system and are sent periodically.
The messages giving more important information such as the speed of the ball are sent
as fast as possible, which means faster than the status messages.
*/
void TM7_IRQHandler(void){
  HAL_TIM_IRQHandler(&htim7);
}

/*!
\fn HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
\param huart pointer on huart structure (as defined in the HAL library)
\brief Function called everytime a new byte is available from UART communication

This function is used to receive data from UART communication. Everytime one byte is
received by the STM32 it is copied in the \ref inByte and then passed to the mavlink_parse_char
function. Once enough byte are taken and one message is received the function
\ref mouseDriver_readMsg is called and a subsiquent action is taken.
*/
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
  HAL_NVIC_DisableIRQ(USART2_IRQn);
  mavlink_message_t inmsg;
  mavlink_status_t msgStatus;
  if (huart->Instance == USART2){
    /* Receive one byte in interrupt mode */
    HAL_UART_Receive_IT(&huart2, &inByte, 1);
    if(mavlink_parse_char(0, inByte, &inmsg, &msgStatus)){

      mouseDriver_readMsg(inmsg);
    }
  }
  HAL_NVIC_EnableIRQ(USART2_IRQn);
}

/*!
\fn HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
\param htim pointer on timer structure (as defined in the HAL library)
\brief Function called everytime a certain time is enlapsed

This function is used to send periodically some status information to the PC.
*/
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if (htim->Instance==TIM7){
      mouseDriver_send_status_msg();
    }
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
```

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */


  /* MCU Configuration
     --------------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM7_Init();
  MX_TIM1_Init();
  MX_DMA_Init();
  MX_SPI2_Init();
  /* USER CODE BEGIN 2 */
  HAL_InitTick(0);
  HAL_NVIC_SetPriority(USART2_IRQn,1,0);
  HAL_NVIC_EnableIRQ(USART2_IRQn);
  HAL_NVIC_SetPriority(TIM7_IRQn,2,0);
  HAL_NVIC_EnableIRQ(TIM7_IRQn);
  HAL_GPIO_WritePin(GPIOC, CS_0_Pin|CS_1_Pin, GPIO_PIN_SET);

  HAL_UART_Receive_IT(&huart2, &inByte, 1);
  HAL_TIM_Base_Start_IT(&htim7);
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);

  mouseDriver_init();

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  while (1)
  {
   mouseDriver_idle();
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

```

```
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
 * @brief SPI2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_SPI2_Init(void)
{
```

```

  /* USER CODE BEGIN SPI2_Init 0 */
  HAL_GPIO_DeInit(GPIOC, GPIO_PIN_3);

  /*GPIO_InitTypeDef pin;
  pin.Pin = GPIO_PIN_3;
  pin.Mode = GPIO_MODE_OUTPUT_PP;
  pin.Pull = GPIO_PULLDOWN;
  pin.Speed = GPIO_SPEED_MEDIUM;
  HAL_GPIO_Init(GPIOC, &pin);
  HAL_GPIO_WritePin(GPIOC,GPIO_PIN_3, GPIO_PIN_RESET);*/

  __HAL_RCC_SPI2_CLK_ENABLE();
  __SPI2_CLK_ENABLE();
  /* USER CODE END SPI2_Init 0 */

  /* USER CODE BEGIN SPI2_Init 1 */

  /* USER CODE END SPI2_Init 1 */
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_HIGH;
  hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 7;
  hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi2.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI2_Init 2 */


  /* USER CODE END SPI2_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
```

```
TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

/* USER CODE BEGIN TIM1_Init 1 */

/* USER CODE END TIM1_Init 1 */
htim1.Instance = TIM1;
htim1.Init.Prescaler = PRESCALER_PWM;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = COUNTER_PERIOD_PWM;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
  Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = PULSE_PWM;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
  Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.BreakFilter = 0;
sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
sBreakDeadTimeConfig.Break2Filter = 0;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
```

```c
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM7 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM7_Init(void)
{

  /* USER CODE BEGIN TIM7_Init 0 */

  /* USER CODE END TIM7_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM7_Init 1 */

  /* USER CODE END TIM7_Init 1 */
  htim7.Instance = TIM7;
  htim7.Init.Prescaler = PRESCALER_HEART;
  htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim7.Init.Period = COUNTER_PERIOD_HEART;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM7_Init 2 */

  /* USER CODE END TIM7_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */
  /* DMA controller clock enable */
  __DMA1_CLK_ENABLE();

  /* Peripheral DMA init*/
```

```
  hdma_usart2_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
  hdma_usart2_tx.Init.PeriphInc = DMA_PINC_DISABLE;
  hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
  hdma_usart2_tx.Init.PeriphDataAlignment = DMA_MDATAALIGN_BYTE;
  hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
  hdma_usart2_tx.Init.Mode = DMA_NORMAL;
  hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
  HAL_DMA_Init(&hdma_usart2_tx);

  __HAL_LINKDMA(&huart2,hdmatx,hdma_usart2_tx);
  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 230400;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel7_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel7_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel7_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

```

```
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, CS_0_Pin|CS_1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, PW_1_Pin|LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(PW_0_GPIO_Port, PW_0_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : CS_0_Pin CS_1_Pin */
  GPIO_InitStruct.Pin = CS_0_Pin|CS_1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PW_1_Pin LD2_Pin */
  GPIO_InitStruct.Pin = PW_1_Pin|LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PW_0_Pin */
  GPIO_InitStruct.Pin = PW_0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(PW_0_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

```

```c
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(char *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

C.2 Treadmill driver

```c
/*! \file mouseDriver.c
\brief Implementation of the driver for the mouse treadmil project.

\author Didier Negretto
*/
#ifndef MOUSEDRIVER_C_
#define MOUSEDRIVER_C_

#ifndef TEST
#include "mouseDriver.h"
#else
#include "../test/test_mouseDriver.h"
#endif
/*!
\def K
\brief Proportional coefficient for motor control.
*/
#define K 10
/*!
\def K
\brief Proportional coefficient for motor control.
*/
#define I 10
/*!
\def I
\brief Integral coefficient for motor control.
*/
#define MAX_MOTOR_SIGNAL 255
/*!
\def MAX_MOTOR_SIGNAL
\brief Max value for the motor signal
\attention This value is used to limit the motor speed. If this is changed the motors might break !!

This value limits the motor speed and thus is used to vaoid spinning the motor too fast and break it.
If this value is changed the motor might spin too fast and destroy itself or the gear box. Extreme caution
needs to be taken if this value is modified.
*/
#define MIN_MOTOR_SIGNAL 10
/*!
\def MIN_MOTOR_SIGNAL
```

```
\brief Min value for the motor signal. Any value lower than that will cause the motor to stop
*/
#define MAX_MISSING_MEASURES 15
/*!
\def MAX_MISSING_MEASURES
\brief After MAX_MISSING_MEASURES non valid measures from sensors the motors are stopped and mode goes
to stop.
*/
#ifndef TEST
/*!
\var actual_mode
\brief Global variable defining the mode of the machine

This value is updated based on the received messages. When a routine is running it is
only possible to stop the machine.
*/
static uint8_t actual_mode = MOUSE_MODE_STOP;
/*!
\var actual_speed_measure
\brief Global variable for the measured speed

This value is updated based on sensor.
*/
static mavlink_speed_info_t actual_speed_measure;
/*!
\var actual_speed_setpoint
\brief Global variable for the speed setpoint

This value is updated based on messages when the mode is set to SPEED.
*/
static mavlink_speed_setpoint_t actual_speed_setpoint;
/*!
\var actual_motor_signal
\brief Global variable for the speed motor signal

This value is updated based on closed-loop control and the value provided in
\ref actual_speed_setpoint and \ref actual_speed_measure.
It is also possible to overwrite it by sending a mavlink_motor_setpoint_t message if the
mode is set to SPEED.
*/
static mavlink_motor_setpoint_t actual_motor_signal;
/*!
\var points
\brief Global variable for storing the points to be followed in AUTO mode

The maximum amout of points is defined by \ref MAX_POINTS. This array is emptied after
every reset of the system. If not all the points are defined the routine is interrupted as
soon as a point with duration == 0 is detected.
*/
static mavlink_point_t points[255];
/*!
\var actual_point
\brief Global variable for keeping track of the index in the \ref points array.
*/
static uint8_t actual_point = 0;
/*!
\var actual_point_start_time
\brief Global variable for keeping track of the time when the last point in \ref points array started.
*/
static uint32_t actual_point_start_time = 0;
```

```c
/*!
\var actual_error
\brief Global variable to store and send the last error occured
*/
static mavlink_error_t actual_error;
/*!
\var actual_raw_sensor
\brief Global variable to store and send the row sensor values from X and Y sensors
*/
static mavlink_raw_sensor_t actual_raw_sensor[2];
/*!
\var send_msg
\brief Flag for sending status messages. Those messages are sent with lower freqency.
*/
static int send_msg = 1;
/*!
\fn mouseDriver_initSetpoint
\brief Function that initializes the setpoint to 0

This function modifies \ref actual_speed_setpoint by setting it to 0.
*/
#endif
/*!
\fn mouseDriver_sendMsg(uint32_t msgid)
\param msgid is the ID of the message to be sent.
\brief Function that sends a message given its ID.
\attention This function can be called in interrupts whith a priority lower than 0 (1,2,3,...),
otherwise the HAL_Delay() function stall and the STM32 crashes.

This function access global variables to send information to the computer.
Given one message ID the functions reads the information from a global variable and
sends it using the DMA as soon as the previous messages are sent.
*/
void mouseDriver_sendMsg(uint32_t msgid);
/*!
\fn mouseDriver_initSetpoint
\brief Function that initializes the motor setpoint to 0.

This function initializes \ref actual_speed_setpoint.
*/
void mouseDriver_initSetpoint(void);
/*!
\fn mouseDriver_initMode
\brief Function that initializes the mode to MOUSE_MODE_STOP

This function modifies \ref actual_mode by setting it to MOUSE_MODE_STOP.
*/
void mouseDriver_initMode(void);
/*!
\fn mouseDriver_initPoints
\brief Function that initializes the routine points for AUTO mode to 0.

This function modifies \ref points by setting all their fields to 0.
*/
void mouseDriver_initPoints(void);/*!
\fn mouseDriver_setMode(uint8_t mode)
\param mode is the mode in which the driver should be set.
\brief Function that sets the mode of the machine.

This functions modifies the mode of the machine. Not all transitions are possible,
```

```
this functions verifies that the transitions are lawful.
*/
void mouseDriver_setMode(uint8_t mode);

/*!
\fn mouseDriver_initMotorSignal
\brief Function that initializes the motor signals to 0.

This function modifies \ref actual_motor_signal by setting all their fields to 0.
*/
void mouseDriver_initMotorSignal(void);

void mouseDriver_initSetpoint(void){
  actual_speed_setpoint.setpoint_x = 0;
  actual_speed_setpoint.setpoint_y = 0;
}
void mouseDriver_initMode(void){
  actual_mode = MOUSE_MODE_STOP;
}
void mouseDriver_initPoints(void){
  for(int i=0; i<MAX_POINTS; i++){
    points[i].duration = 0;
    points[i].setpoint_x = 0;
    points[i].setpoint_y = 0;
    points[i].point_id = 0;
  }
  actual_point = 0;
  actual_point_start_time = 0;
}
void mouseDriver_initMotorSignal(void){
    actual_motor_signal.motor_x = 0;
    actual_motor_signal.motor_y = 0;
}
void mouseDriver_init(void){
  mouseDriver_initMode();
  mouseDriver_initSetpoint();
  mouseDriver_initPoints();
  mouseDriver_initMotorSignal();

  /* Init sensor as well */
  sensorDriver_init();
  main_stop_motors();
}
uint32_t mouseDriver_getTime (void){
  return (HAL_GetTick());
}
void mouseDriver_send_status_msg(void){
  send_msg = 1;
}
void mouseDriver_control_idle(void){
  static int count = 0;
    static float integral_x = 0;
    static float integral_y = 0;
    float error_x = 0;
    float error_y = 0;
  if (actual_speed_measure.valid == 0){
    count ++;
    if(count >= MAX_MISSING_MEASURES){
      main_stop_motors();
      mouseDriver_setMode(MOUSE_MODE_STOP);
```

```
            integral_x = 0;
            integral_y = 0;
    }
    return;
  }
  if (actual_mode == MOUSE_MODE_SPEED || actual_mode == MOUSE_MODE_AUTO_RUN){
    actual_motor_signal.time = mouseDriver_getTime();
        error_x = actual_speed_setpoint.setpoint_x-actual_speed_measure.speed_x;
        error_y = actual_speed_setpoint.setpoint_y-actual_speed_measure.speed_y;
    actual_motor_signal.motor_x = (float)K*(error_x)+(float)I*integral_x;
    actual_motor_signal.motor_y = (float)K*(error_y)+(float)I*integral_y;

    if (actual_motor_signal.motor_x > MAX_MOTOR_SIGNAL){
        actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL;
    }
    if(actual_motor_signal.motor_y  > MAX_MOTOR_SIGNAL){
            actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL;
        }

    main_set_motors_speed(actual_motor_signal);
        integral_x += (actual_motor_signal.motor_x < MAX_MOTOR_SIGNAL)? error_x : 0;
        integral_y += (actual_motor_signal.motor_y < MAX_MOTOR_SIGNAL)? error_y : 0;
    count = 0;
  }
  else{
    actual_motor_signal.motor_x = 0;
    actual_motor_signal.motor_y = 0;
    main_stop_motors();
        integral_x = 0;
        integral_y = 0;
  }
}

void mouseDriver_setMode(uint8_t mode){
    if (mode == MOUSE_MODE_STOP){
        main_stop_motors();
        actual_point = 0;
        actual_mode = MOUSE_MODE_STOP;
        mouseDriver_initMotorSignal();
    }
    if (mode == MOUSE_MODE_AUTO_LOAD){
        actual_mode = mode;
        mouseDriver_sendMsg(MAVLINK_MSG_ID_HEARTBEAT);
    }
    if (actual_mode == MOUSE_MODE_AUTO_LOAD && mode == MOUSE_MODE_AUTO_RUN ){
        actual_point = 0;
        actual_point_start_time = mouseDriver_getTime();
        actual_speed_setpoint.setpoint_x = points[0].setpoint_x;
        actual_speed_setpoint.setpoint_y = points[0].setpoint_y;
        actual_mode = mode;
    }

    if (actual_mode != MOUSE_MODE_AUTO_RUN)
        actual_mode = mode;
}
void mouseDriver_sendMsg(uint32_t msgid){
    mavlink_message_t msg;
    static uint8_t outBuffer[MAX_BYTE_BUFFER_SIZE];
    static uint16_t msg_size = 0;

```

```
    while (main_get_huart_tx_state() == HAL_BUSY){
        /*Wait for other messages to be sent*/
        HAL_Delay(1);
    }

    switch(msgid){
        case MAVLINK_MSG_ID_HEARTBEAT:
            mavlink_msg_heartbeat_pack(SYS_ID,COMP_ID, &msg, actual_mode, mouseDriver_getTime());
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_SPEED_SETPOINT:
            mavlink_msg_speed_setpoint_encode(SYS_ID,COMP_ID, &msg, &actual_speed_setpoint);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_MOTOR_SETPOINT:
            mavlink_msg_motor_setpoint_encode(SYS_ID,COMP_ID, &msg, &actual_motor_signal);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_SPEED_INFO:
            mavlink_msg_speed_info_encode(SYS_ID,COMP_ID, &msg, &actual_speed_measure);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_ERROR:
            mavlink_msg_error_encode(SYS_ID,COMP_ID,&msg,&actual_error);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_POINT_LOADED:
            mavlink_msg_point_loaded_pack(SYS_ID,COMP_ID,&msg,actual_point);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_POINT:
            mavlink_msg_point_encode(SYS_ID,COMP_ID,&msg,&points[actual_point]);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        case MAVLINK_MSG_ID_RAW_SENSOR:
            mavlink_msg_raw_sensor_encode(SYS_ID,COMP_ID,&msg,&actual_raw_sensor[0]);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            while (main_get_huart_tx_state() == HAL_BUSY){
                    /*Wait for other messages to be sent*/
                    HAL_Delay(1);
            }
            mavlink_msg_raw_sensor_encode(SYS_ID,COMP_ID,&msg,&actual_raw_sensor[1]);
            msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
            main_transmit_buffer(outBuffer, msg_size);
            break;
        default:
            break;
    }
}
void mouseDriver_idle (void){
    uint64_t difference = 0;
    sensorDriver_motion_read_speed(actual_raw_sensor, &actual_speed_measure);
```

```
    switch(actual_mode){
    case MOUSE_MODE_STOP:
        mouseDriver_initSetpoint();
        mouseDriver_initMotorSignal();
        actual_motor_signal.time = mouseDriver_getTime();
        main_stop_motors();
        mouseDriver_control_idle();
        mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_INFO);

        break;
    case MOUSE_MODE_SPEED:
        mouseDriver_control_idle();
        mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_INFO);
        mouseDriver_sendMsg(MAVLINK_MSG_ID_MOTOR_SETPOINT);

        break;
    case MOUSE_MODE_AUTO_LOAD:
        if (actual_point == 255){
            actual_error.error = MOUSE_ROUTINE_TOO_LONG;
            actual_error.time = mouseDriver_getTime();
            mouseDriver_control_idle();
            mouseDriver_sendMsg(MAVLINK_MSG_ID_ERROR);
        }
        break;
    case MOUSE_MODE_AUTO_RUN:
        difference = mouseDriver_getTime()-actual_point_start_time;
        if (difference >= points[actual_point].duration){
            if (actual_point < MAX_POINTS-1){
                actual_point++;

                if(points[actual_point].duration == 0){
                    actual_point = 0;
                }
                actual_speed_setpoint.setpoint_x = points[actual_point].setpoint_x;
                actual_speed_setpoint.setpoint_y = points[actual_point].setpoint_y;
                actual_point_start_time = mouseDriver_getTime();
            }
        }
        if (actual_point == MAX_POINTS){
            mouseDriver_setMode(MOUSE_MODE_AUTO_LOAD);
        }
        mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_INFO);
        mouseDriver_sendMsg(MAVLINK_MSG_ID_MOTOR_SETPOINT);
        mouseDriver_control_idle();
        break;
    default:
        break;
    }
    if (send_msg == 1){
        send_msg = 0;
        if(actual_mode != MOUSE_MODE_AUTO_LOAD){
          mouseDriver_sendMsg(MAVLINK_MSG_ID_HEARTBEAT);
            mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_SETPOINT);
            mouseDriver_sendMsg(MAVLINK_MSG_ID_RAW_SENSOR);
            mouseDriver_sendMsg(MAVLINK_MSG_ID_MOTOR_SETPOINT);
        }
    }

}
void mouseDriver_readMsg(const mavlink_message_t msg){
```

```
	switch(msg.msgid){

	case MAVLINK_MSG_ID_MODE_SELECTION:
		mouseDriver_setMode( mavlink_msg_mode_selection_get_mode(&msg));
		break;

	case MAVLINK_MSG_ID_SPEED_SETPOINT:
		if (actual_mode == MOUSE_MODE_SPEED)
			mavlink_msg_speed_setpoint_decode(&msg, &actual_speed_setpoint);
		break;

	case MAVLINK_MSG_ID_MOTOR_SETPOINT:
		if (actual_mode == MOUSE_MODE_SPEED)
			mavlink_msg_speed_setpoint_decode(&msg, &actual_speed_setpoint);
		break;
	case MAVLINK_MSG_ID_POINT:
		if(actual_mode == MOUSE_MODE_AUTO_LOAD){
			mavlink_msg_point_decode(&msg, &points[actual_point]);
			if (actual_point == 255){
				actual_error.error = MOUSE_ROUTINE_TOO_LONG;
				actual_error.time = mouseDriver_getTime();
				mouseDriver_sendMsg(MAVLINK_MSG_ID_ERROR);
			}
			mouseDriver_sendMsg(MAVLINK_MSG_ID_POINT_LOADED);
			actual_point ++;

		}
		break;
	default:
		break;
	};
}
#endif
```

```
/*! \file mouseDriver.h
\brief Header of the driver for the mouse treadmil project.

\author Didier Negretto
*/

/*
 * Code used for driving the 3D mouse treadmill
 * Author: Didier Negretto
 *
 */

#pragma once
#ifndef MOUSEDRIVER_N_H
/*!
\def MOUSEDRIVER_N_H
\brief To avoid double includes
*/
#define MOUSEDRIVER_N_H

#ifndef TEST
#include "mavlink.h"
#include "utils.h"
#include "sensorDriver.h"
#endif

```

```
#include <math.h>
/* Constants for MALINK functions*/

/*!
\def SYS_ID
\brief System ID for MAVLink
*/
#define SYS_ID 0

/*!
\def COMP_ID
\brief Component ID for MAVLink
*/
#define COMP_ID 0

/* maximum size of the trasmit buffer */
/*!
\def MAX_BYTE_BUFFER_SIZE
\brief MAX size of transmit buffer in bytes
*/
#define MAX_BYTE_BUFFER_SIZE 500

/*!
\def MAX_POINTS
\brief MAX amount of points that can be defined in AUTO mode
*/
#define MAX_POINTS 255

/*!
\fn mouseDriver_init
\brief Function that initializes the driver of the mouse treadmill.

This functions initialites the mouse treadmill driver. It initializes the sensors as well.
*/
void mouseDriver_init(void);

/*!
\fn mouseDriver_control_idle
\brief Function doing the control on the motors.
\attention This function is in charge of generating the control signals for the
motors. If it is modified, make sure to respect the specifications of the motor
to avoid damaging or destroing them !!

This function is called periodially to update the control signal for the motors.
*/
void mouseDriver_control_idle(void);

/*!
\fn mouseDriver_send_status_msg
\brief Function generating the signal for sending messages.

This function is called periodially to set the flag for sending status messages.
*/
void mouseDriver_send_status_msg(void);

/*!
\fn mouseDriver_readMsg(const mavlink_message_t msg)
\param msg MAVLink message to be decoded
\brief Function that reads one message.

```

```
This function is called in main.c. Depending on the received message different actions are taken.
*/
void mouseDriver_readMsg(const mavlink_message_t msg);

/*!
\fn mouseDriver_getTime
\return The actual time in ms from boot of the system.
\brief Function that gets the time of the system from boot.
*/
uint32_t mouseDriver_getTime (void);

/*!
\fn mouseDriver_idle
\brief Idle function for the mouse treadmill driver.
\note This function needs to be called periodically to ensure a correct behaviour.

This is the idle function of the mouse treadmill. It reads values from the sensors,
calls \ref mouseDriver_control_idle, and sends high frequency messages (not the status ones).
*/
void mouseDriver_idle (void);


#endif
```

C.3 Sensor driver

```
/*! \file sensorDriver.c
\brief Implementation of the sensor driver for the mouse treadmill project.

\author Didier Negretto
*/
# include "sensorDriver.h"

/*!
 \var sensor_x
 \brief variable for storing data for the x sensor.
 */
static sensor_t sensor_x = {CS_0_GPIO_Port,CS_0_Pin,PW_0_GPIO_Port,PW_0_Pin,0};

/*!
 \var sensor_y
 \brief variable for storing data for the y sensor.
 */
static sensor_t sensor_y = {CS_1_GPIO_Port,CS_1_Pin,PW_1_GPIO_Port,PW_1_Pin,0};

/*!
 \fn sensorDriver_powerup(sensor_t *sensor)
 \param sensor sensor structure of the sensor to be powered up
 \brief This function turns off and the on the sensor. It then performs the power up routine
 \note This routine is time consuming and done only at start up.

 After Flashing the SROM the SROM_ID register is read to confirm that the
 SROM have been flashed correctly.
*/
void sensorDriver_powerup(sensor_t * sensor);

/*!
 \fn sensorDriver_motion_read_raw(uint8_t sensor_id, mavlink_raw_sensor_t * sensor_data)
 \param sensor_id 0 for sensor x, 1 for sensor y
 \param sensor_data pointer to a structure for storing the raw sensor value
 \brief This function reads raw data from the sensor given its ID and puts the result in the pointer.
```

```
*/
void sensorDriver_motion_read_raw(uint8_t sensor_id, mavlink_raw_sensor_t * sensor_data);

void sensorDriver_powerup(sensor_t * sensor){
  /* Disable the sensor */
  HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin, GPIO_PIN_SET);

  /* Make sure all sensor is switched off */
  HAL_GPIO_WritePin(sensor->pw_port, sensor->pw_pin, GPIO_PIN_RESET);
  main_write_sensor(*sensor, 0x00, 0x00);
  HAL_Delay(100);

  /* Gives voltage to sensors */
  HAL_GPIO_WritePin(sensor->pw_port, sensor->pw_pin , GPIO_PIN_SET);
  HAL_Delay(300);

  /* Reset SPI port */
  HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin,  GPIO_PIN_SET);
  HAL_Delay(5);
  HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin,  GPIO_PIN_RESET);
  HAL_Delay(5);
  HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin,  GPIO_PIN_SET);
  HAL_Delay(5);

  /* Write to Power_up_Reset register */
  main_write_sensor(*sensor, Power_Up_Reset, 0x5A);

  /* Wait at least 50 ms */
  HAL_Delay(50);

  /* Read from data registers */
  main_read_sensor(*sensor, 0x02);
  main_read_sensor(*sensor, 0x03);
  main_read_sensor(*sensor, 0x04);
  main_read_sensor(*sensor, 0x05);
  main_read_sensor(*sensor, 0x06);

  /* Start ROM Download */
  main_write_sensor(*sensor, Config2, 0x20);
  main_write_sensor(*sensor, SROM_Enable, 0x1d);
  HAL_Delay(10);
  main_write_sensor(*sensor,SROM_Enable, 0x18);
  main_wait_160us();
  main_wait_20us();

  /* Burst start with adress */
  HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin, GPIO_PIN_RESET);
  main_write_sensor_burst(SROM_Load_Burst|0x80);
  for (int i = 0; i < firmware_length; i++ ){
    main_write_sensor_burst(firmware_data[i]);
  }
  HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin, GPIO_PIN_SET);
  main_wait_160us();
  main_wait_20us();
  main_wait_20us();

  /* Read SROM_ID for verification */
  sensor->status = main_read_sensor(*sensor, SROM_ID);

  /* Write to Config2 for wired mouse */
```

```
  main_write_sensor(*sensor, Config2, 0x00);
}
void sensorDriver_init(void){
  sensorDriver_powerup(&sensor_x);
  sensorDriver_powerup(&sensor_y);
}
void sensorDriver_motion_read_raw(uint8_t sensor_id, mavlink_raw_sensor_t * sensor_data){
  uint8_t data[12];
  int16_t temp = 0;
  sensor_t sensor;

  if (sensor_id == SENSOR_X) sensor = sensor_x;
  else if (sensor_id == SENSOR_Y) sensor = sensor_y;
  else return;
  sensor_data->sensor_id = sensor_id;

  /* write to motion burst adress */
  main_write_sensor(sensor, Motion_Burst, 0xbb);

  /* Prepare for burst */
  HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_RESET);
  sensor_data->time = mouseDriver_getTime();
  main_write_sensor_burst(Motion_Burst);
  /* Start burst */
  main_read_sensor_motion_burst(data);
  HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_SET);
  /* END of burst */
  main_wait_20us();

  /* Read other register for stopping burst mode */
  sensor_data->product_id = main_read_sensor(sensor,  Product_ID);

  /* TWO's Complement */
  temp = (data[DELTA_X_H]<<8) | (data[DELTA_X_L]);
  temp = ~temp + 1;
  sensor_data->delta_x = temp;
  temp = (data[DELTA_Y_H]<<8) | (data[DELTA_Y_L]);
  temp = ~temp +1;
  sensor_data->delta_y = temp;

  sensor_data->squal = data[SQUAL_READ];
  sensor_data->lift = (data[MOTION] & 0x08) >> 3;
  sensor_data->srom_id = sensor.status;
}
void sensorDriver_motion_read_speed(mavlink_raw_sensor_t sensor_data[2], mavlink_speed_info_t *
      speed_info){
  mavlink_raw_sensor_t raw_values[2];
    uint32_t old_time[2];

    speed_info->valid = 0;
    old_time[0] = speed_info->time_x;
    old_time[1] = speed_info->time_y;

  sensorDriver_motion_read_raw(SENSOR_X, &raw_values[0]);
  sensorDriver_motion_read_raw(SENSOR_Y, &raw_values[1]);

  speed_info->speed_x =  -(float)raw_values[0].delta_y*(float)INCH2METER/(float)RESOLUTION;
  speed_info->speed_x /= (float)(raw_values[0].time-old_time[0])/(float)1000;
  speed_info->time_x = raw_values[0].time;
  speed_info->speed_y =  -(float)raw_values[1].delta_y*(float)INCH2METER/(float)RESOLUTION;
```

```
  speed_info->speed_y /= (float)(raw_values[1].time-old_time[1])/(float)1000;
  speed_info->time_y = raw_values[1].time;
  sensor_data[0] = raw_values[0];
  sensor_data[1] = raw_values[1];

  if((raw_values[0].lift == 0) && (raw_values[1].lift == 0) &&
    (raw_values[0].squal >= SQUAL_THRESH) && (raw_values[0].squal >= SQUAL_THRESH) &&
    (raw_values[0].product_id == 66) && (raw_values[1].product_id == 66)){
    speed_info->valid = 1;
  }
  else{
    speed_info->valid = 0;
  }
}
```

```
/*! \file sensorDriver.h
\brief Header of the sensor driver for the mouse treadmil project.

\author Didier Negretto
*/
#pragma once

#ifndef SENSORDRIVER_H_
#define SENSORDRIVER_H_

#ifndef TEST
#include "main.h"
#include "mavlink.h"
#include "sensorSROM.h"
#endif

/* BEGIN DEFINES FOR SENSOR INTERNAL REGISTERS */
#define Product_ID  0x00
#define Revision_ID 0x01
#define Motion  0x02
#define Delta_X_L 0x03
#define Delta_X_H 0x04
#define Delta_Y_L 0x05
#define Delta_Y_H 0x06
#define SQUAL 0x07
#define Raw_Data_Sum  0x08
#define Maximum_Raw_data  0x09
#define Minimum_Raw_data  0x0A
#define Shutter_Lower 0x0B
#define Shutter_Upper 0x0C
#define Control 0x0D
#define Config1 0x0F
#define Config2 0x10
#define Angle_Tune  0x11
#define Frame_Capture 0x12
#define SROM_Enable 0x13
#define Run_Downshift 0x14
#define Rest1_Rate_Lower  0x15
#define Rest1_Rate_Upper  0x16
#define Rest1_Downshift 0x17
#define Rest2_Rate_Lower  0x18
#define Rest2_Rate_Upper  0x19
#define Rest2_Downshift 0x1A
#define Rest3_Rate_Lower  0x1B
#define Rest3_Rate_Upper  0x1C
#define Observation 0x24
```

```
#define Data_Out_Lower  0x25
#define Data_Out_Upper  0x26
#define Raw_Data_Dump 0x29
#define SROM_ID 0x2A
#define Min_SQ_Run  0x2B
#define Raw_Data_Threshold  0x2C
#define Config5 0x2F
#define Power_Up_Reset  0x3A
#define Shutdown  0x3B
#define Inverse_Product_ID  0x3F
#define LiftCutoff_Tune3  0x41
#define Angle_Snap  0x42
#define LiftCutoff_Tune1  0x4A
#define Motion_Burst  0x50
#define LiftCutoff_Tune_Timeout 0x58
#define LiftCutoff_Tune_Min_Length  0x5A
#define SROM_Load_Burst 0x62
#define Lift_Config 0x63
#define Raw_Data_Burst  0x64
#define LiftCutoff_Tune2  0x65
/* END DEFINES FOR SENSOR INTERNAL REGISTERS */

#include <mavlink_msg_raw_sensor.h>
#include <stdint.h>

/* DEFINES FOR BURST READ (only usefull data) */
#define MOTION 0
#define OBSERVATION 1
#define DELTA_X_L 2
#define DELTA_X_H 3
#define DELTA_Y_L 4
#define DELTA_Y_H 5
#define SQUAL_READ 6

/*!
\def SQUAL_THRESH
\brief Threshold value on SQUAL to consider the measure valid.
*/
#define SQUAL_THRESH 16

/*!
\def RESOLUTION
\brief Resolution of the sensor in Count per Inch (CPI)
\note This value needs to be updated if the resolution of the sensors is changed,

This value is used to convert the raw sensor value in counts to meter per second.
*/
#define RESOLUTION 5000

/*!
\def INCH2METER
\brief Conversion factor to convert inches in meters.
*/
#define INCH2METER 0.0254

/*!
\fn sensorDriver_init
\brief Initializes all sensors.

This functions powers down the sensor and does the powering up routine.
```

```
\note This routine takes a long time, so it is done only at start up.
*/
void sensorDriver_init(void);

/*!
\fn sensorDriver_motion_read_speed(mavlink_raw_sensor_t sensor_data[2], mavlink_speed_info_t * speed_info)
\param sensor_data[2] array for the raw values of the 2 sensors
\param speed_info pointer to a mavlink_speed_info_t
\brief Function for reading the raw data and speed measures from the sensors.
\attention The speed_info.time_x/y is used to compute speed. This value should NOT BE MODIFIED by
the caller function

This function reads values from the sensors and puts them in the given pointers.
It also flags invalid readings, so that \ref mouseDriver_control_idle do not use them.
*/
void sensorDriver_motion_read_speed(mavlink_raw_sensor_t sensor_data[2], mavlink_speed_info_t * speed_info);

#endif
```

C.4 Code for unit tests

```
/*! \file display.h
\brief Header and implementation of display function for unit tests

\author Didier Negretto
*/

#ifndef DISPLAY_H_
#define DISPLAY_H_

/* DEFINES COLORS FOR DISPLAY IN TERMINAL */
/*!
\def RED
\brief Prints text between RED and \ref END in red color
*/
#define RED     "\x1b[31m"
/*!
\def GREEN
\brief Prints text between GREEN and \ref END in green color
*/
#define GREEN   "\x1b[32m"
/*!
\def END
\brief stops printin using color.
*/
#define END   "\x1b[0m"

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

#ifdef COLOR
static inline bool display (bool correct, const char *name){
    if(correct == 1){
        printf("    ["GREEN "OK" END"] ");
        printf(name);
        printf(GREEN " DONE SUCCESSFULY\n" END);
        return 1;
    }
```

```
    else{
       printf("["RED "NO" END"]      ");
       printf(name);
       printf(RED " PERFORMED INCORRECTLY OR NOT AT ALL\n" END);
       return 0;
    }
    return 0;
}
#else
/*!
 \fn static inline bool display (bool correct, const char *name)
 \param correct 1 if the test is successfull 0 if it is not
 \param name pointer to string with the name of the test that is run
 \return The result of the test (1 if correct == 1, 0 if correct == 0).
 \brief This function prints on the terminal is the test is passed successfully
 or not
 */
static inline bool display (bool correct, const char *name){
    if(correct == 1){
        printf("    [OK] ");
        printf("%s", name);
        printf(" DONE SUCCESSFULY\n");
        return 1;
    }
    else{
       printf("[NO]      ");
       printf("%s", name);
       printf(" PERFORMED INCORRECTLY OR NOT AT ALL\n");
       return 0;
    }
    return 0;
}
#endif
#endif /* DISPLAY_H_ */
```

```
/*! \file main.c
\brief Main for unit testss
\author Didier Negretto

This main is compiled and run after the compilation of the stm32 project
This main runs the unit tests and prints which tests are passed and which are not
\attention  The bash script for the automatic unit testing after compilation
was written for MAC and may not work on LINUX or Windows. To solve this issue
modify CodeSTM32/src/build.sh
*/

#include "test_mouseDriver.h"
#include "test_sensorDriver.h"

int main(void){

    bool test = 1;

    printf("========================================================\n");
    printf("**********TESTING CODE FOR MOUSE TREADMILL *********\n");
    printf("========================================================\n\n");
    printf("--------------------------------------------------------\n");
    printf("TESTING mouseDriver.c\n");
    printf("TESTING mouseDriver_init()\n");
    test &= test_mouseDriver_init();
    printf("TESTING mouseDriver_idle()\n");
```

```
    test &= test_mouseDriver_idle();
    printf("TESTING mouseDriver_getTime()\n");
    test &= test_mouseDriver_getTime();
    printf("TESTING mouseDriver_send_status_msg()\n");
    test &= test_mouseDriver_send_status_msg();
    printf("TESTING mouseDriver_control_idle()\n");
    test &= test_mouseDriver_control_idle();
    /*printf("--------------------------------------------------\n");
    printf("TESTING mouseDriver.c\n");
    if (! test_mouseDriver_init()) printf(RED"ERRORS IN mouseDriver_init\n"END);*/


    if (test == 1){
        printf("ALL TEST PASSED SUCCESSUFULLY\n");
    }
    else{
        printf("==================================================\n");
        printf("!!!!!!!!!!!! SOME TESTS NOT PASSED !!!!!!!!!!!!!!!!!\n");
        printf("==================================================\n\n");

    }
    return test;
}
```

```
/*
 * mock_mouseDriver.h
 *
 *  Created on: Nov 24, 2019
 *      Author: Didier
 */

#ifndef MOCK_MOUSEDRIVER_H_
#define MOCK_MOUSEDRIVER_H_

#define HAL_BUSY  0
#define SYS_ID 0
#define COMP_ID 0
#define MAX_BYTE_BUFFER_SIZE 500
#define MAX_POINTS 255


static int stop_motor = 0;
static int sensor_init = 0;
static int sensor_read_x = 0;
static int sensor_read_y = 0;

/* Define mock variables for testing */
static int send_msg = 1;
static uint8_t actual_mode = MOUSE_MODE_STOP;
static mavlink_speed_setpoint_t actual_speed_setpoint;
static mavlink_speed_info_t actual_speed_measure;
static mavlink_motor_setpoint_t actual_motor_signal;
static mavlink_point_t points[255];
static uint8_t actual_point = 0;
static uint32_t actual_point_start_time = 0;
static mavlink_error_t actual_error;
static mavlink_raw_sensor_t actual_raw_sensor[2];

/* Define mock functions */
static inline void sensorDriver_init(void){sensor_init = 1; };
static inline uint32_t HAL_GetTick(void){
```

```
    static uint32_t i = 0;
    i++;
    return i;
};
static inline void main_set_motors_speed(mavlink_motor_setpoint_t actual_motor_signal){stop_motor = 0;};
static inline void main_stop_motors(void){stop_motor = 1;};
static inline int main_get_huart_tx_state(void){return 1;};
static inline void HAL_Delay(int delay){};
static inline void main_transmit_buffer(uint8_t * outbuffer, int msg_size){};

static inline void sensorDriver_motion_read_speed(mavlink_raw_sensor_t actual_raw_sensor[2],
     mavlink_speed_info_t * actual_speed_measure){
    sensor_read_x = 1;
    sensor_read_y = 1;
    actual_raw_sensor[0].delta_x = 0;
    actual_raw_sensor[1].delta_y = 0;
    actual_speed_measure->speed_x = 0;
    actual_speed_measure->speed_y = 0;
};

#endif /* MOCK_MOUSEDRIVER_H_ */
```

```
/*! \file mock_sensorDriver.h
\brief In this file mock functions are defined for the sensor driver unit tests

\author Didier Negretto
*/


#ifndef MOCK_SENSORDRIVER_H_
#define MOCK_SENSORDRIVER_H_

/**
 * A mock structure to represent one sensor
 */
typedef struct SENSOR{
    /*@{*/
    int cs_port; /**< the chip select port for the sensor */
    uint8_t cs_pin;/**< the chip select pin for the sensor */
    int pw_port; /**< the power port for the sensor */
    uint8_t pw_pin;/**< the power pin for the sensor */
    uint8_t status;/**< the sensor status. This is the SROM_ID after the upload of the
  firmware. This value should not be 0 otherwise the upload of the SROM is failed. */
    /*@}*/
} sensor_t;

#define CS_0_GPIO_Port 0
#define CS_0_Pin 0
#define PW_0_GPIO_Port 0
#define PW_0_Pin 0

#define CS_1_GPIO_Port 1
#define CS_1_Pin 1
#define PW_1_GPIO_Port 1
#define PW_1_Pin 1

#define GPIO_PIN_SET 1
#define GPIO_PIN_RESET 0

static int firmware_length = 3;
static int firmware_data[3] = {1,2,3};
```

```c

static inline void main_wait_160us(void){};
static inline void main_wait_20us(void){};
static inline uint8_t main_read_sensor(sensor_t sensor, uint8_t adress ){return adress;};
static inline void main_write_sensor(sensor_t sensor, uint8_t adress, uint8_t value){};
static inline void main_read_sensor_motion_burst(uint8_t* buffer){};
static inline void main_write_sensor_burst(uint8_t adress){};
static inline void HAL_Delay(int delay){};
static inline void HAL_GPIO_WritePin(int port, int pin, int state){};
static inline uint32_t mouseDriver_getTime(void){
    static uint32_t i = 0;
    i++;
    return i;
}

#endif /* MOCK_SENSORDRIVER_H_ */
```

```c
/*
 * test.h
 *
 *  Created on: Nov 24, 2019
 *      Author: Didier
 */

#ifndef TEST_MOUSEDRIVER_H_
#define TEST_MOUSEDRIVER_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include "mavlink.h"

/* Define testing functions*/
bool test_mouseDriver_init(void);
bool test_mouseDriver_idle(void);
bool test_mouseDriver_getTime(void);
bool test_mouseDriver_send_status_msg(void);
bool test_mouseDriver_control_idle(void);

#endif /* TEST_MOUSEDRIVER_H_ */
```

```c
/*
 * test_sensorDriver.h
 *
 *  Created on: Nov 25, 2019
 *      Author: Didier
 */

#ifndef TEST_SENSORDRIVER_H_
#define TEST_SENSORDRIVER_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include "mavlink.h"

/* Define test functions */
bool test_sensorDriver_init(void);

```

```c
#endif /* TEST_SENSORDRIVER_H_ */
```

```c
/*
 * test_mouseDriver.c
 *
 *  Created on: Nov 24, 2019
 *      Author: Didier
 */
#include "test_mouseDriver.h"
#include "mock_mouseDriver.h"
#include "display.h"
#include "mouseDriver.c"


bool test_mouseDriver_init(void){

    bool test = 1;

    actual_mode = 5;
    for(int i = 0; i < MAX_POINTS; i++){
        points[i].duration = i;
        points[i].setpoint_x = i;
        points[i].setpoint_y = i;
        points[i].point_id = i;
    }
    actual_point = 10;
    actual_point_start_time = 10;
    actual_speed_setpoint.setpoint_x = 10;
    actual_speed_setpoint.setpoint_y = 10;
    actual_motor_signal.motor_x = 10;
    actual_motor_signal.motor_y = 10;

    sensor_init = 0;
    stop_motor = 0;

    mouseDriver_init();

    test &= display(actual_mode == 0, "actual_mode initialization");
    test &= display(actual_point == 0, "actual_point initialization");
    test &= display(actual_point_start_time == 0, "actual_point_start_time initialization");
    test &= display((actual_speed_setpoint.setpoint_y  == 0)&& (actual_speed_setpoint.setpoint_x  == 0), "actual_speed_setpoint initialization");
    bool test_sub = 1;
    for(int i = 0; i < MAX_POINTS; i++){
        test_sub &= ((points[i].duration == 0) && (points[i].setpoint_x == 0) &&
                     (points[i].setpoint_y == 0) && (points[i].point_id == 0));
    }
    test &= display(test_sub, "points initialized correctly");
    test &= display(sensor_init == 1, "sensor_init initialization");
    test &= display(stop_motor == 1, "stop_motor initialization");
    test &= display((actual_motor_signal.motor_x  == 0)&& (actual_motor_signal.motor_y  == 0), "actual_motor_signal initialization");

    return test;
}

bool test_mouseDriver_idle(void){
    bool test = false;
    actual_speed_measure.speed_x = -10;
    actual_speed_measure.speed_y = -10;
    actual_speed_measure.valid = 1;
```

```
    actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
    actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
    actual_point_start_time = 0;
    actual_point = 0;
    points[0].duration = 100;
    points[0].setpoint_x = 10;
    points[0].setpoint_y = 10;
    points[0].point_id = 0;

    /* Test reading of sensors in SPEED mode */
    actual_mode = MOUSE_MODE_SPEED;
    sensor_read_x = 0;
    sensor_read_y = 0;
    stop_motor = 1;
    mouseDriver_idle();
    test = display(sensor_read_x == 1, "read sensor x in MOUSE_MODE_SPEED");
    test &= display(sensor_read_y == 1, "read sensor y in MOUSE_MODE_SPEED");
    test &= display(stop_motor == 0, "motor started in MOUSE_MODE_SPEED");

    /* Test reading of sensors in MOUSE_MODE_AUTO_RUN mode */
    actual_mode = MOUSE_MODE_AUTO_RUN;
    sensor_read_x = 0;
    sensor_read_y = 0;
    stop_motor = 1;
    mouseDriver_idle();
    test &= display(sensor_read_x == 1, "read sensor x in MOUSE_MODE_AUTO_RUN");
    test &= display(sensor_read_y == 1, "read sensor y in MOUSE_MODE_AUTO_RUN");
    test &= display(stop_motor == 0, "motor started in MOUSE_MODE_AUTO_RUN");
    return test;
}
bool test_mouseDriver_getTime(void){
    bool test = 1;
    uint32_t start = HAL_GetTick();
    test &= mouseDriver_getTime() == start+1;
    test &= mouseDriver_getTime() == start+2;
    test &= mouseDriver_getTime() == start+3;
    test &= mouseDriver_getTime() == start+4;
    test &= mouseDriver_getTime() == start+5;
    display(test, "time update");

    return test;
}
bool test_mouseDriver_send_status_msg(void){
    bool test = false;
    send_msg = 0;

    mouseDriver_send_status_msg();

    test = send_msg;
    display(test, "status message send request");
    return test;
}
bool test_mouseDriver_control_idle(void){
    bool test = 1;
    stop_motor = 0;
    actual_speed_measure.speed_x = -10;
    actual_speed_measure.speed_y = -10;
    actual_motor_signal.motor_x = 10;
    actual_motor_signal.motor_y = 10;
    actual_mode = MOUSE_MODE_STOP;
```

```
    /* Case actual mode == STOP */
    printf("if (actual_mode == MOUSE_MODE_STOP)\n");
    mouseDriver_control_idle();
    test &= display((actual_motor_signal.motor_x == 0)&& (actual_motor_signal.motor_y == 0), "
      actual_motor_signal reset");
    test &= display(stop_motor == 1, "motor stop");

    /* Case actual mode == SPEED */
    actual_mode = MOUSE_MODE_SPEED;
    stop_motor = 1;
    actual_speed_setpoint.setpoint_y = 0;
    actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
    printf("if (actual_mode == MOUSE_MODE_SPEED)\n");
    mouseDriver_control_idle();
    test &= display(stop_motor == 0, "motor_x speed changed");
    for(int i = 0; i < 100; i++)
        mouseDriver_control_idle();
    test &= display(actual_motor_signal.motor_x <= MAX_MOTOR_SIGNAL, "motor_x with
      MAX_MOTOR_SIGNAL limit");

    stop_motor = 1;
    actual_speed_setpoint.setpoint_x = 0;
    actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
    mouseDriver_control_idle();
    test &= display(stop_motor == 0, "motor_y speed changed");
    for(int i = 0; i < 100; i++)
        mouseDriver_control_idle();
    test &= display(actual_motor_signal.motor_y <= MAX_MOTOR_SIGNAL, "motor_y with
      MAX_MOTOR_SIGNAL limit");

    actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
    actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
    mouseDriver_control_idle();
    test &= display(stop_motor == 0, "motor_y and motor_x speed changed");
    for(int i = 0; i < 100; i++)
        mouseDriver_control_idle();
    test &= display((actual_motor_signal.motor_y <= MAX_MOTOR_SIGNAL) && (actual_motor_signal.
      motor_x <= MAX_MOTOR_SIGNAL), "motor_y and motor_x with MAX_MOTOR_SIGNAL limit");

    /* Reaction to invalid measures */
    actual_speed_setpoint.setpoint_x = 0;
    actual_speed_setpoint.setpoint_y = 0;
    actual_speed_measure.speed_x = 1000;
    actual_speed_measure.speed_y = 1000;
    actual_motor_signal.motor_x = 10;
    actual_motor_signal.motor_y = 10;
    bool test_stop = true;
    actual_speed_measure.valid = 0;
    for(int i = 0;i < MAX_MISSING_MEASURES-1; i++ ){
        test_stop &= (actual_motor_signal.motor_x == 10);
        test_stop &= (actual_motor_signal.motor_y == 10);
        mouseDriver_control_idle();
    }
```

```c
    mouseDriver_control_idle();
    test &= display(test_stop, "constant motor signal if invalid measure");
    test &= display(actual_mode == MOUSE_MODE_STOP, "stop motor after too many invalid measures");



    /* Case actual mode == SPEED */
    actual_mode = MOUSE_MODE_AUTO_RUN;
    stop_motor = 1;
    actual_speed_setpoint.setpoint_y = 0;
    actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
    actual_speed_measure.valid = 1;
    printf("if (actual_mode ==  MOUSE_MODE_AUTO_RUN)\n");
    mouseDriver_control_idle();
    test &= display(stop_motor == 0, "motor_x speed changed");
    for(int i = 0; i < 100; i++)
        mouseDriver_control_idle();
    test &= display(actual_motor_signal.motor_x <= MAX_MOTOR_SIGNAL, "motor_x with MAX_MOTOR_SIGNAL limit");

    stop_motor = 1;
    actual_speed_setpoint.setpoint_x = 0;
    actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
    mouseDriver_control_idle();
    test &= display(stop_motor == 0, "motor_y speed changed");
    for(int i = 0; i < 100; i++)
        mouseDriver_control_idle();
    test &= display(actual_motor_signal.motor_y <= MAX_MOTOR_SIGNAL, "motor_y with MAX_MOTOR_SIGNAL limit");

    actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
    actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
    actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
    mouseDriver_control_idle();
    test &= display(stop_motor == 0, "motor_y and motor_x speed changed");
    for(int i = 0; i < 100; i++)
        mouseDriver_control_idle();
    test &= display((actual_motor_signal.motor_y <= MAX_MOTOR_SIGNAL) && (actual_motor_signal.motor_x <= MAX_MOTOR_SIGNAL), "motor_y and motor_x with MAX_MOTOR_SIGNAL limit");

    test_stop = true;
    actual_speed_measure.valid = 0;
    actual_motor_signal.motor_x = 10;
    actual_motor_signal.motor_y = 10;
    for(int i = 0;i < MAX_MISSING_MEASURES-1; i++ ){
        test_stop &= (actual_motor_signal.motor_x == 10);
        test_stop &= (actual_motor_signal.motor_y == 10);
        mouseDriver_control_idle();
    }
    mouseDriver_control_idle();
    test &= display(test_stop, "constant motor signal if invalid measure");
    test &= display(actual_mode == MOUSE_MODE_STOP, "stop motor after too many invalid measures");

    return test;
}
```

```c
/*
 * test_sensorDriver.c
 *
 *  Created on: Nov 25, 2019
 *      Author: Didier
 */

#include "test_sensorDriver.h"
#include "mock_sensorDriver.h"
#include "display.h"
#include "sensorDriver.c"

bool test_sensorDriver_init(void){
    return display(0,"TEST SENSOR DRIVER");
}
```

C.5 Build script

```bash
#!/bin/bash
# Script for compiling and running test before compilation
# of the STM32 code and upload.
echo PRE-BUILD STEPS
echo CLEANING TESTS
make clean -C ../../CodeSTM32/test/Debug/
echo COMPILING TESTS
make all -C ../../CodeSTM32/test/Debug/
echo RUNNING TESTS
./../../CodeSTM32/test/Debug/test
```

D Code for PC

D.1 GUI

D.2 Routine example

E Data-sheets

E.1 Sensor Data-sheet