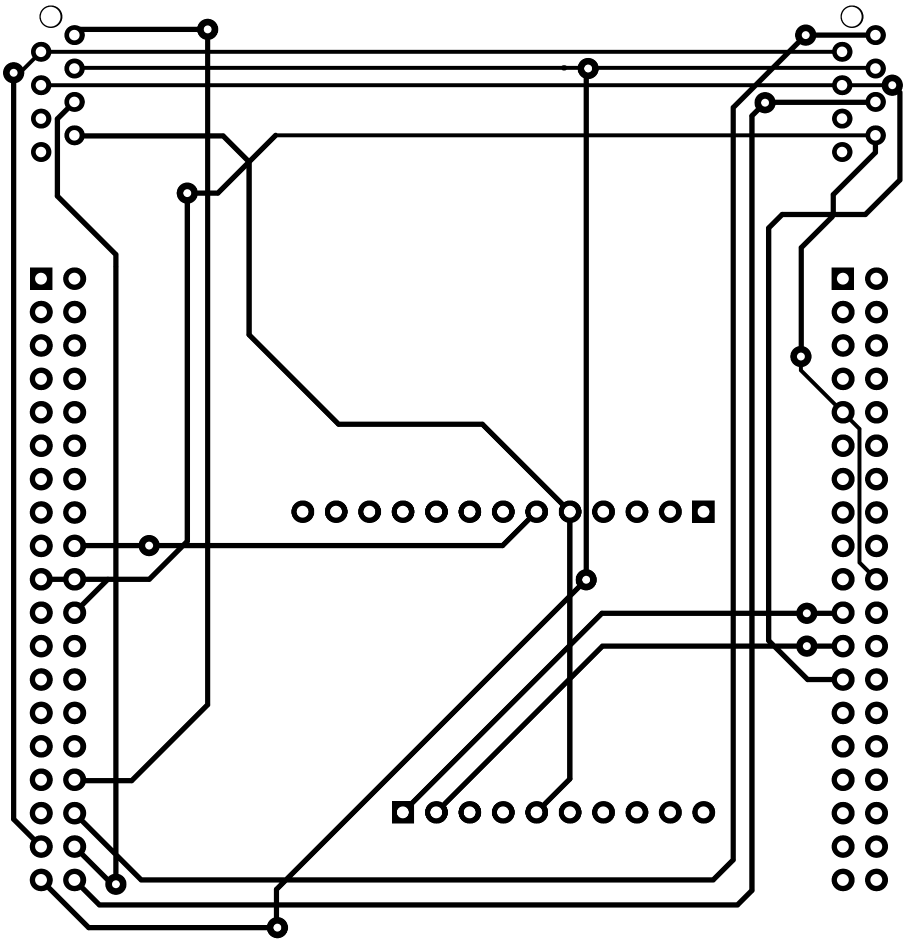
EPFL

Biorobotics laboratory



MOUSE TREADMILL CONTROL

Wednesday 1st 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 is characterized and tested. 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.

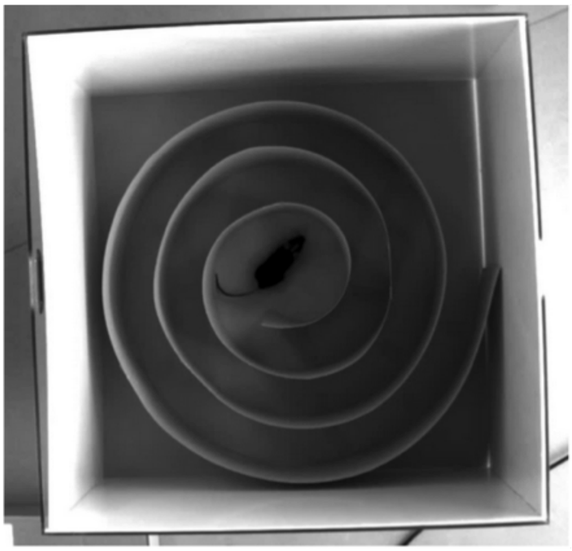


Figure 0.1 – Roll coefficient as a function of the angle of attack for different values of twisting and folding angles. Schematics of folding configurations are added.

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 some tests are carried out to characterize the system performances. First the noise at 0 $[\frac{m}{s}]$ in the sensors is measured, then the noise at constant speed. This is important to determine what type of noise is present in the system and whatever it useful to filter it and how. The quality of the surface with a static and moving treadmill is also compared. The noise is characterize with the following parameters: Then one test on the step response is performed, which gave these results: Finally the treadmill was tested using a sinusoidal routine and the tracking performances (the difference between the desired path and the real routine) are shown in figure **??**.

Cost As a prof of concept the treadmill seems to be a proper solution providing solutions to many of the drawbacks of the previous cardboard maze setup. On the other hand not all the requirements are fulfilled yet, 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 increased 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 issue of the fixed-head mouse 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

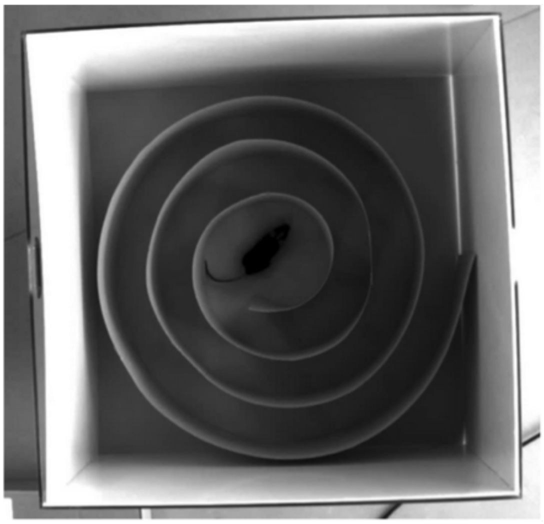


Figure 1.2 – The experimental setup used in [1].

The studies on mammal locomotion have driven more and more attention over the years, and especially experiments on mice, such as [1], have enhanced our understanding of the neuronal circuits that enable locomotion. The experimental setup in [1], on the other hand, 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

To asses these issues 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** Once a 2D speed setpoint is chosen the speed of the surface needs to be measured and the motor control signal need to be adjusted automatically 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 consist in 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, the system architecture and communication are explained in 2. Section 3, describes the design decisions and the components choices made .Section 5 describes the control strategy and shows some preliminary responses. Finally in section 6 the conclusion of the project is given. The code, code documentation as well as the data-sheets of the components are annexed.

2 System architecture

In this section the architecture of the system is explained and detailed. One first overview of the system is given in figure 2.3.

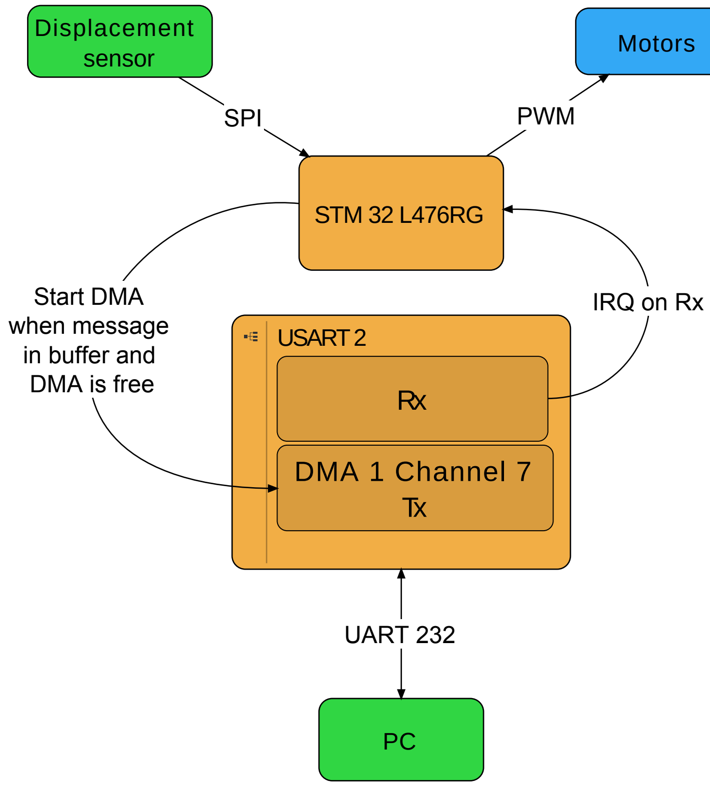


Figure 2.3 – Architecture for mouse treadmill project

The core of the system is the STM32L476RG, which can read from the sensors using the SPI interface and control the motors using the PWM. Moreover it can communicate with the computer and the GUI for data logging and to receive the inputs form 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.

3 Design choices

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

3.1 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 on the board.

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 controllers involved.

3.2 Communication

For the communication with the computer the UART 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.

Figure 3.4 – MAVLink logo

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"[2], 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 3. Thanks to the description file it is possible to generate libraries in different programming languages (C, Python, Java, ...) and if in the future a new message is required a 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.

Name	**Description**	**Sender**	**Receiver**
HEARTBEAT	Verifies communication	STM32	PC
SPEED_INFO	Measured speed	STM32	PC
SPEED_SETPOINT	Speed setpoint	PC	STM32
MODE_SELECTION	Changes mode	PC	STM32
MOTOR_SETPOINT	Up time of PWM duty cycle	STM32	PC
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

Table 3 – List and description of the MAVLink messages.

AD
if r
sag
sta
or

3.3 Sensor

For sensing the speed of the wheel a contactless solution is chosen. To achieve this goal a optical gaming mouse sensor is taken. Nevertheless the sensor need 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 implentation. 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 feature 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 performances of the sensor are summarized on table 4[1]. For more details refer to D.1.

Description	**Value**	**Unit**
High speed detection	6.3	$[\frac{m}{s}]$
High acceleration detection	490	$[\frac{m}{s^2}]$
Default resolution	5000	$[cpi]$
Resolution error of	1	[%]
4 wires SPI interface	1	–
Cost	29.99	[$]

Table 4 – PMW3660 main features.

3.4 Motor

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

The data given are

- m_s mass of the sphere
- r_s radius of the sphere
- m_w mass of the wheel
- r_w radius of the wheel
- M_{max} maximum torque provided by the motor-gearbox
- ω_{max} maximum angular speed of the motor-gearbox
- J_m inertia of the rotor

It is therefore possible to estimate the maximum continuous acceleration and speed of the sphere.

[1] $[cpi]$ stands for counts per inch.

4 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.

4.1 Installation of the PC software

First python 3 needs to be installed, for that see [3]. 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 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
```

4.2 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 4.5. Figure 4.5 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

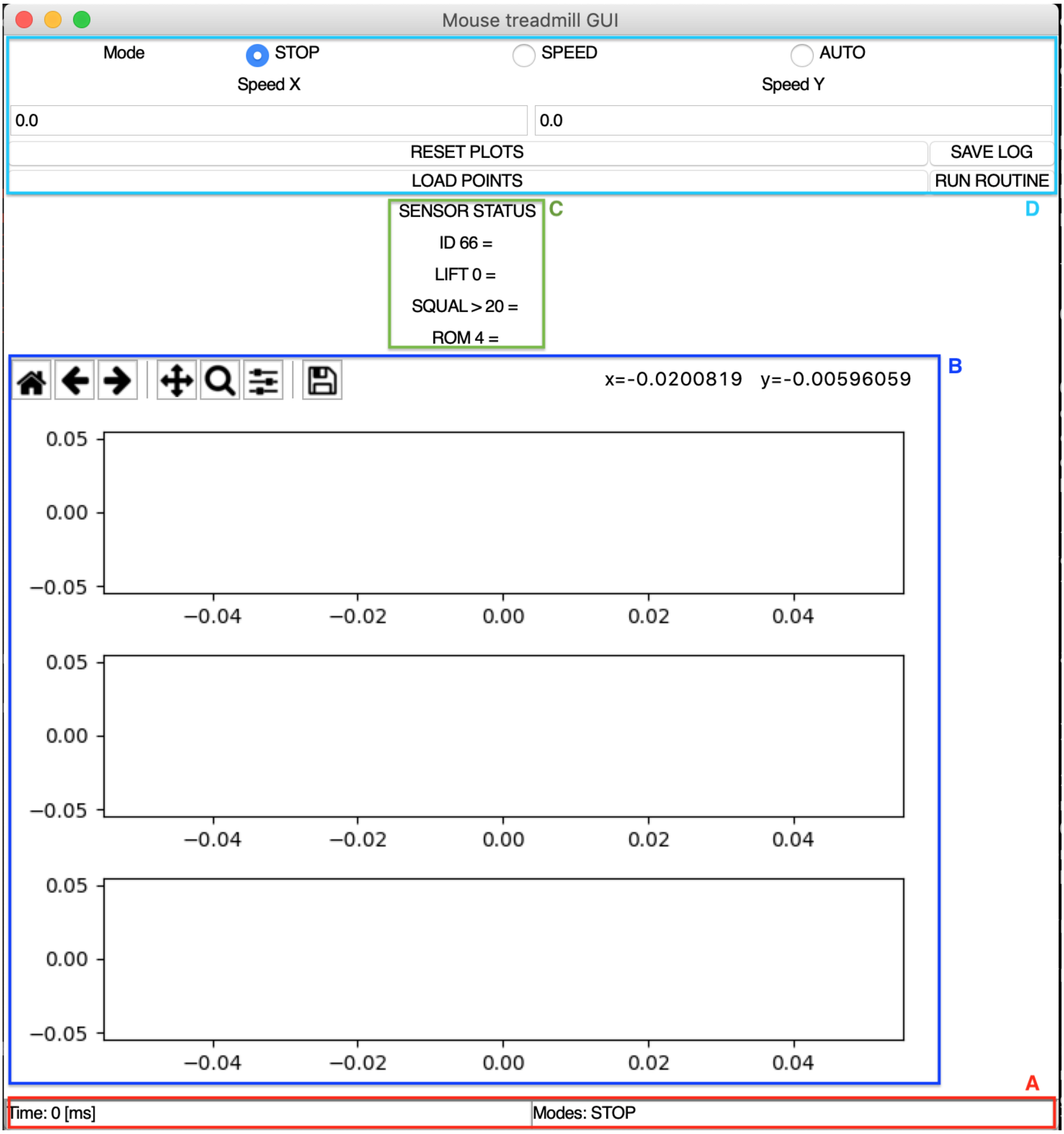


Figure 4.5 – GUI screenshot on a MAC.

SROM is flashed correctly during initialization. If everything is working correctly you should see something like table 5

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

Table 5 – 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).

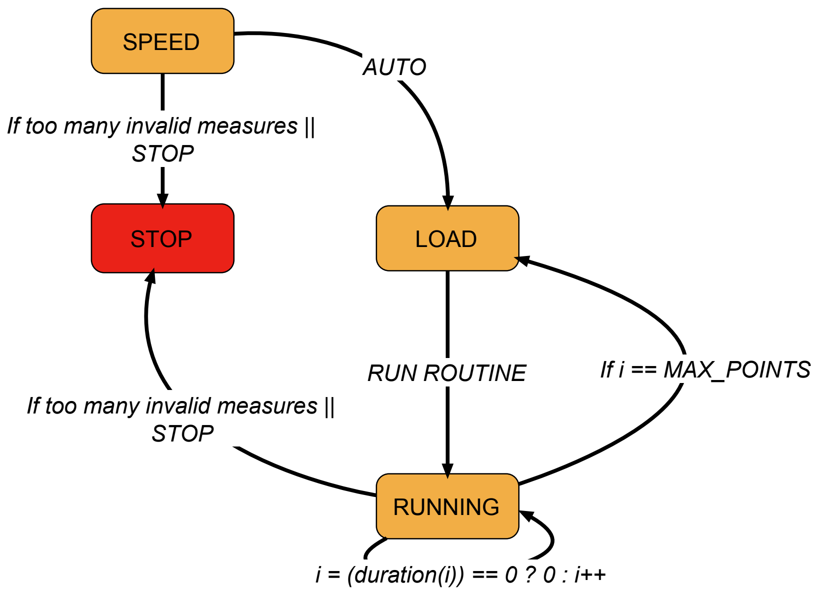


Figure 4.6 – Finite state machine of the mouse treadmill.

The finite state machine of the machine is shown in figure 4.6. 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.

4.3 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.

4.4 How to extend the system

Say wh to era all the bra !!!

5 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 5.2). 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).

5.1 Inputs/Outputs

In this section the signal definitions are described. The control diagram is shown in figure 5.7. The signals are defined as follow:

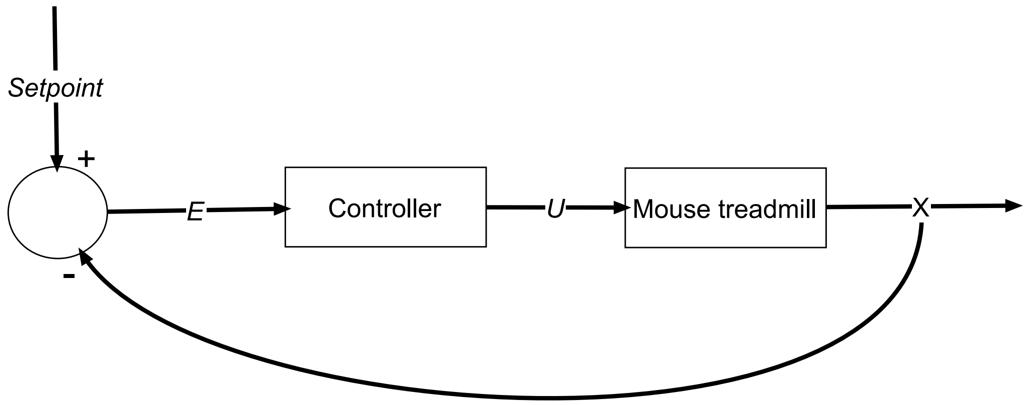


Figure 5.7 – Control diagram

- $X = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$: 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 D.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.

5.2 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 1.

$$U = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = K * \begin{bmatrix} e_x \\ e_y \end{bmatrix} + I * \begin{bmatrix} i_x \\ i_y \end{bmatrix} \tag{1}$$

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.

5.3 Results

In this section the results of the control are shown as well as the achieved performances in terms measures 5.3.1 and control 5.3.2.

5.3.1 Measure test

In this section a simple experiment is described and the results are shown. The experiment consist in letting the ball spin in at constant speed in one direction and measure the speed using the sensors. By doing this it is possible to analyse the noise present in the sensor and characterize it.

5.3.2 Control test

In this section different step responses are measured using different controller parameters and compared to see which values should be used for the I and K parameter in equation 1.

6 Conclusion

References

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

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

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

[4] Robin R. Murphy, Eric Steimle et al. *Cooperative Use of Unmanned Sea Surface and Micro Aerial Vehicles at Hurricane Wilma*,Journal of Field Robotics,2008

[5] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, Daisuke Nakazawa,*Autonomous Flying Robots*, Springer, 2010.

[6] G. Sachs, *What Can Be Learned from Unique Lateral-Directional Dynamics Properties of Birds for Mini-Aircraft*, Atmospheric Flight Mechanics Conference and Exhibit, 2007

[7] R. Pecora, F. Amoroso, and L. Lecce, *Effectiveness of Wing Twist Morphing in Roll Control*, Journal of aircraft Vol. 49, No. 6, November–December 2012

[8] Osgar John Ohanian III, Christopher Hickling, Brandon Stiltner, Etan D. Karni, *Piezoelectric Morphing versus Servo-Actuated MAV Control Surfaces*, 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, 2012

[9] Helen M. Garcia , Mujahid Abdulrahim and Rick Lind, *Roll control for a micro air vehicle using active wing morphing*, AIAA Guidance, Navigation, and Control Conference and Exhibit, 2003

[10] Mujahid Abdulrahim, Helen Garcia, and Rick Lind, *Flight Characteristics of Shaping the Membrane Wing of a Micro Air Vehicle*, Journal of aircraft Vol. 42, No. 1, January–February 2005

[11] M. Di Luca, S. Mintchev, G. Heitz, F. Noca and D. Floreano, *Bioinspired morphing wings for extended flight envelope and roll control of small drones*, Interface Focus 7, 2017

[12] William E. Green and Paul Y. Oh, *A Hybrid MAV for Ingress and Egress of Urban Environments*, IEEE transactions on robotics, 2009

[13] Bret Stanford, Mujahid Abdulrahim, Rick Lind, and Peter Ifju, *Actuation for Roll Control of a Micro Air Vehicle*, Journal of aircraft, 2007

[14] Juan Carlos Gomez and Ephrahim Garcia, *Morphing unmanned aerial vehicles*, Smart materials and structures, 2011.

[15] E.L. Houghton, P.W. Carpenter, Steven H. Collicott and Daniel T. Valentine, *Aerodynamics for engineering students*, Seventh edition.

[16] Pero Skorput, Sadko Mandzuka, Hrvoje Vojvodic, *The Use of Unmanned Aerial Vehicles for Forest Fire Monitoring* , 58th International Symposium ELMAR, 2016

[17] David Gallacher, *Drone Applications for Environmental Management in Urban Spaces: A Review*, International Journal of Sustainable Land Use and Urban Planning, 2016

[18] Ludovic Apvrille, Tullio Tanzi, and Jean-Luc Dugelay *Autonomous Drones for Assisting Rescue Services within the context of Natural Disasters*, XXXIth URSI General Assembly and Scientific Symposium, 2014

List of Figures

List of Tables

A MAVLink dialect description file

```xml
<?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>
            </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">
```

```xml
                <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">
            <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 Code for STM32 NUCLEO 64 board

B.1 Main

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

```c
  * @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. */
  /*@}*/
} 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.
\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 9
#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();
}
```

```c
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
  */
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)
  {
    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 */
}

#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****/
```

B.2 Treadmill driver

```
/*! \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 100
/*!
\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;
/*!
\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:
            /* DEMO CODE INIT*/
                actual_speed_measure.time_x = mouseDriver_getTime();
            /* DEMO CODE END*/
            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_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_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
```

B.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_x*(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_x*(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
```

B.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"

```

```c
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;
}
```

```c
/*
 * 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};

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_ */
```

```
/*
 * 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_ */
```

```
/*
 * 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);

#endif /* TEST_SENSORDRIVER_H_ */
```

```
/*
 * 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();
}
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;
```

```c
    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");
}
```

B.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
```

C Code for PC

C.1 GUI

```python
import serial
import os
import sys
import numpy as np
#import matplotlib as plt
from appJar import gui
import time
import json
from tqdm import tqdm
import routine as mouseRoutine
from pymavlink.dialects.v20 import mouse as mouseController

"""
PATH

/Users/Didier/Desktop/EPFL/Secondo_master/SemesterProject2019/GITRepository/3DMouseTreadmill/MouseTreadmillPC/python
"""
SENSOR_STATUS_MSG = ["SENSOR STATUS", "ID 66 = ", "LIFT 0 = ", "SQUAL > 20 = ", "ROM 4 = "]
MODES = ["STOP", "SPEED","AUTO","RUNNING"]
MODES_NUM = {"STOP": int(0),"SPEED": int(1),"AUTO": int(2),"RUNNING": int(3) }
DATA = { "HEARTBEAT": {"time": [], "mode": []},
        "SPEED_SETPOINT": {"time": [], "setpoint_x": [], "setpoint_y": [], "start": 0},
        "SPEED_INFO":  {"time": [], "speed_x": [], "speed_y": [], "start": 0},
        "MOTOR_SETPOINT": {"time": [], "motor_x": [], "motor_y": [], "start": 0}
        }
LOG = []
MAX_SAMPLES_ON_SCREEN = 200
print(mouseController.MAVLink_speed_info_message.fieldnames)
#port = "/dev/cu.usbmodem14102"
port  = "/dev/stdout"

class MyApplication():
    actualMode = 0
    actualTime = 0
    actualSpeedSetpoint = [None, None]
    actualMotorSetpoint = [None, None]
    actualSpeedInfo = [None, None]
    connection = serial.Serial(port, baudrate = 230400, timeout = 50)
    mavlink = mouseController.MAVLink(file = connection )
    setpointX = 0.0
    setpointY = 0.0
```

```python

    def commSTM32 (self):
        # Init variables
        m = None
        while(self.connection.in_waiting>0):
            # Recive messages
            try:
                m = self.mavlink.parse_char(self.connection.read())
            except:
                pass
            if m:
                LOG.append(m)
                if m.name == "HEARTBEAT":
                    self.actualTime = m.time
                    self.actualMode = m.mode
                    DATA["HEARTBEAT"]["time"].append(self.actualTime)
                    DATA["HEARTBEAT"]["mode"].append(self.actualMode)
                elif m.name == "SPEED_SETPOINT":
                    self.actualSpeedSetpoint[0] = m.setpoint_x
                    self.actualSpeedSetpoint[1] = m.setpoint_y
                    DATA["SPEED_SETPOINT"]["time"].append(self.actualTime)
                    DATA["SPEED_SETPOINT"]["setpoint_x"].append(self.actualSpeedSetpoint[0])
                    DATA["SPEED_SETPOINT"]["setpoint_y"].append(self.actualSpeedSetpoint[1])
                    #DATA["SPEED_SETPOINT"]["setpoint_z"].append(self.actualSpeedSetpoint[2])
                elif m.name == "MOTOR_SETPOINT":
                    self.actualMotorSetpoint[0] = m.motor_x
                    self.actualMotorSetpoint[1] = m.motor_y
                    DATA["MOTOR_SETPOINT"]["time"].append(m.time)
                    DATA["MOTOR_SETPOINT"]["motor_x"].append(self.actualMotorSetpoint[0])
                    DATA["MOTOR_SETPOINT"]["motor_y"].append(self.actualMotorSetpoint[1])
                    #DATA["SPEED_SETPOINT"]["motor_z"].append(self.actualMotorSetpoint[2])
                elif m.name == "SPEED_INFO":
                    #print(m)
                    DATA["SPEED_INFO"]["time"].append(m.time_x)
                    DATA["SPEED_INFO"]["speed_x"].append(m.speed_x)
                    DATA["SPEED_INFO"]["speed_y"].append(m.speed_y)
                elif m.name == "RAW_SENSOR":
                    if m.sensor_id == 0:
                        status_x = []
                        status_x.append(m.product_id)
                        status_x.append(m.lift)
                        status_x.append(m.squal)
                        status_x.append(m.srom_id)
                    elif m.sensor_id == 1:
                        status_y = []
                        status_y.append(m.product_id)
                        status_y.append(m.lift)
                        status_y.append(m.squal)
                        status_y.append(m.srom_id)
                    try:
                        if (len(status_x) == 4) and (len(status_y) == 4):
                            self.app.setLabel("sensorStatus1",SENSOR_STATUS_MSG[1]+str(status_x[0])+"|"+str(status_y[0]))
                            self.app.setLabel("sensorStatus2",SENSOR_STATUS_MSG[2]+str(status_x[1])+"|"+str(status_y[1]))
                            self.app.setLabel("sensorStatus3",SENSOR_STATUS_MSG[3]+str(status_x[2])+"|"+str(status_y[2]))
                            self.app.setLabel("sensorStatus4",SENSOR_STATUS_MSG[4]+str(status_x[3])+"|"+
```

```
str(status_y[3]))
                    except:
                        pass


                elif m.name == "POINT":
                    print(m)
                else:
                    pass
            m = None
    def refreshPlot(self):

        # Clear plot
        for i in range(3):
            self.ax[i].clear()

        # Define labels
        """
        self.ax[2].set_xlabel("Time")
        self.ax[2].set_ylabel("Measured speed [m/s]")
        self.ax[1].set_ylabel("Speed setpoint [m/s]")
        self.ax[0].set_ylabel("Motor signal [ ]")
        """

        # Limit max amout of points on one graph
        if len(DATA["SPEED_INFO"]["time"][DATA["SPEED_INFO"]["start"]:])-1>MAX_SAMPLES_ON_SCREEN:
            DATA["SPEED_INFO"]["start"] = -MAX_SAMPLES_ON_SCREEN
            DATA["SPEED_SETPOINT"]["start"] = -MAX_SAMPLES_ON_SCREEN
            DATA["MOTOR_SETPOINT"]["start"] = -MAX_SAMPLES_ON_SCREEN

        # Re-plot all graphs
        self.ax[2].plot(DATA["SPEED_INFO"]["time"][DATA["SPEED_INFO"]["start"]:], DATA["SPEED_INFO"]["speed_x"][DATA["SPEED_INFO"]["start"]:], 'b.')
        self.ax[2].plot(DATA["SPEED_INFO"]["time"][DATA["SPEED_INFO"]["start"]:], DATA["SPEED_INFO"]["speed_y"][DATA["SPEED_INFO"]["start"]:], 'r.')
        self.ax[1].plot(DATA["SPEED_SETPOINT"]["time"][DATA["SPEED_SETPOINT"]["start"]:], DATA["SPEED_SETPOINT"]["setpoint_x"][DATA["SPEED_SETPOINT"]["start"]:],'b.')
        self.ax[1].plot(DATA["SPEED_SETPOINT"]["time"][DATA["SPEED_SETPOINT"]["start"]:], DATA["SPEED_SETPOINT"]["setpoint_y"][DATA["SPEED_SETPOINT"]["start"]:],'r.')
        self.ax[0].plot(DATA["MOTOR_SETPOINT"]["time"][DATA["MOTOR_SETPOINT"]["start"]:], DATA["MOTOR_SETPOINT"]["motor_x"][DATA["MOTOR_SETPOINT"]["start"]:],'b.')
        self.ax[0].plot(DATA["MOTOR_SETPOINT"]["time"][DATA["MOTOR_SETPOINT"]["start"]:], DATA["MOTOR_SETPOINT"]["motor_y"][DATA["MOTOR_SETPOINT"]["start"]:],'r.')
        self.ax[0].set_adjustable('box',True)
        self.app.refreshPlot("plot")

    def resetPlot(self):
        DATA["SPEED_INFO"]["start"] = len(DATA["SPEED_INFO"]["time"])-3
        DATA["SPEED_SETPOINT"]["start"] = len(DATA["SPEED_SETPOINT"]["time"])-3
        DATA["MOTOR_SETPOINT"]["start"] = len(DATA["MOTOR_SETPOINT"]["time"])-3

    def refreshGUI(self):
        self.commSTM32()

        # Refresh status bar
        self.app.setStatusbar("Time: "+str(self.actualTime)+" [ms]", 0)
        self.app.setStatusbar("Modes: "+str(MODES[self.actualMode]), 1)
```

```python
        self.refreshPlot()
        """
        self.app.setLabel("speedSetpointX", str(self.actualSpeedSetpoint[0]))
        self.app.setLabel("speedSetpointY", str(self.actualSpeedSetpoint[1]))
        self.app.setLabel("motorSetpointX", str(self.actualMotorSetpoint[0]))
        self.app.setLabel("motorSetpointY", str(self.actualMotorSetpoint[1]))
        """

    def setMode(self):
        self.mavlink.mode_selection_send(MODES_NUM[self.app.getRadioButton("optionMode")])
        while(self.connection.out_waiting > 0):
            time.sleep(0.001)
        time.sleep(0.001)
        if self.actualMode == mouseController.MOUSE_MODE_STOP:
            self.setpointX = 0
            self.setpointY = 0


    def setSpeedX(self):
        if self.actualMode == mouseController.MOUSE_MODE_SPEED:
            self.setpointX = self.app.getEntry("speedX")
            if self.setpointX is None or  self.setpointY is None :
                pass
            else:
                self.mavlink.speed_setpoint_send(float(self.setpointX), float(self.setpointY))
                while(self.connection.out_waiting > 0):
                    time.sleep(0.001)
                time.sleep(0.001)

    def setSpeedY(self):
        if self.actualMode == mouseController.MOUSE_MODE_SPEED:
            self.setpointY = self.app.getEntry("speedY")
            if self.setpointX is None or  self.setpointY is None :
                pass
            else:
                self.mavlink.speed_setpoint_send(float(self.setpointX), float(self.setpointY))
            while(self.connection.out_waiting > 0):
                time.sleep(0.001)
            time.sleep(0.001)

    def loadRoutine(self):
        if (len(mouseRoutine.ROUTINE["duration"])>254 or len(mouseRoutine.ROUTINE["setpoint_x"])>254 or len(mouseRoutine.ROUTINE["setpoint_y"])>254):
            raise ValueError("mouseRoutine too long")
        if not (len(mouseRoutine.ROUTINE["duration"]) == len(mouseRoutine.ROUTINE["setpoint_x"]) == len(mouseRoutine.ROUTINE["setpoint_y"])):
            raise ValueError("not all components of mouseRoutine have the same lenght")


        # TODO add verification on max speed and min speed


        for i in tqdm(range(len(mouseRoutine.ROUTINE["duration"]))):
            self.mavlink.point_send(mouseRoutine.ROUTINE["duration"][i],i,mouseRoutine.ROUTINE["setpoint_x"][i], mouseRoutine.ROUTINE["setpoint_y"][i])
            stop = True
            while(self.connection.in_waiting>0 or stop):
                # Recive messages
```

```
            try:
                m = self.mavlink.parse_char(self.connection.read())
            except:
                pass
            if m:
                #print(m)
                if m.name == "POINT_LOADED":
                    if m.point_id == i:
                        stop = False
                    else:
                        print ("ERROR LOADING DATA, wrong msg_id received, STOP MODE instead.")
                        self.mavlink.mode_selection_send(MODES_NUM[0])
                        while(self.connection.out_waiting > 0):
                            time.sleep(0.001)
                        time.sleep(0.001)
                        stop = False
    def saveLog(self):
        with open('log/log.txt', 'w+') as f:
            for item in LOG:
                f.write("%s\n" % item)

    def runRoutine(self):
        if self.actualMode == mouseController.MOUSE_MODE_AUTO_LOAD:
            self.mavlink.mode_selection_send(mouseController.MOUSE_MODE_AUTO_RUN)
            while(self.connection.out_waiting > 0):
                time.sleep(0.001)
            time.sleep(0.001)

    def Prepare(self, app):
        self.ax = []

        app.setTitle("Mouse treadmill GUI")
        app.setFont(12)
        row = 0
        column = 0

        # Mode Selection
        app.startFrame("modeSelection",row = row, column = column, colspan=4, rowspan = 1)
        app.addLabel("optionModeLabel","Mode",0,0,1,1)
        app.addRadioButton("optionMode",MODES[0],0,1,1,1)
        app.addRadioButton("optionMode",MODES[1],0,2,1,1)
        app.addRadioButton("optionMode",MODES[2],0,3,1,1)
        app.setRadioButtonChangeFunction("optionMode",self.setMode)
        app.stopFrame()
        row = row+1

        # Speed entry
        app.startFrame("speedEntry",row = row, column = column, colspan=4, rowspan=2)
        app.addLabel("speedXLabel", "Speed X", 0,0,2,1)
        app.addNumericEntry("speedX",1,0,2,2)
        app.setEntry("speedX", 0.0)
        app.setEntryChangeFunction("speedX", self.setSpeedX)
        app.addLabel("speedYLabel", "Speed Y",0,2,2,1)
        app.addNumericEntry("speedY",1,2,2,2)
        app.setEntry("speedY", 0.0)
        app.setEntryChangeFunction("speedY", self.setSpeedY)
        app.stopFrame()
        row = row+2
```

```
        # Reset plot button
        app.startFrame("GUIButtons", row = row, column = column, colspan=2, rowspan=2)
        self.app.addButton("RESET PLOTS", self.resetPlot, 0,0,1,1)
        self.app.addButton("LOAD POINTS", self.loadRoutine, 1,0,1,1)
        self.app.addButton("RUN ROUTINE", self.runRoutine,1,1,1,1 )
        self.app.addButton("SAVE LOG",self.saveLog,0,1,1,1)
        row = row+1

        # Sensor Status
        app.startFrame("sensorStatus", row = row, column = 0)
        self.app.addLabel("sensorStatus0",SENSOR_STATUS_MSG[0], 0,0,1,1)
        self.app.addLabel("sensorStatus1",SENSOR_STATUS_MSG[1], 1,0,3,1)
        self.app.addLabel("sensorStatus2",SENSOR_STATUS_MSG[2], 2,0,3,1)
        self.app.addLabel("sensorStatus3",SENSOR_STATUS_MSG[3], 3,0,3,1)
        self.app.addLabel("sensorStatus4",SENSOR_STATUS_MSG[4], 4,0,3,1)
        row = row+4

        # Real-time data plotting
        app.startFrame("realTimePlot", row = row, column = column, colspan = 4, rowspan = 4)
        self.fig = app.addPlotFig("plot",0,0,4,4, showNav = True )
        self.ax.append(self.fig.add_subplot(311))
        self.ax.append(self.fig.add_subplot(312))
        self.ax.append(self.fig.add_subplot(313))
        app.stopFrame()
        row = row+4


        # Add status bar
        app.addStatusbar(fields = 2, side=None)
        app.setStatusbar("Time: 0", 0)
        app.setStatusbar("Mode: "+MODES[0], 1)

        # refresh funciton
        app.setPollTime(100)
        app.registerEvent(self.refreshGUI)

        # Window for sensor status
        app.startSubWindow("sensorStatus")
        app.addLabel("status", "SENSOR_X")
        app.stopSubWindow()
        app.openSubWindow("sensorStatus")

        return app
    # Build and Start your application
    def Start(self):
        app = gui()

        self.app = app

        # Run the prebuild method that adds items to the UI
        self.app = self.Prepare(self.app)
        self.app.showAllSubWindow()
        # Start appJar
        self.app.go()

if __name__ == '__main__':
    print("
```

```
        ================================================================
        ")
    print("Running GUI for mouse treadmill")
    print("
        ================================================================
        ")

    # Create an instance of your application
    App = MyApplication()

    # Start your app !
    App.Start()
```

C.2 Routine example

```
ROUTINE = {
    "duration":[500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500,
      500, 100,500, 500, 500, 100, 500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500,
      500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100, 500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100],
    "setpoint_x": [500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500,
      500, 100,500, 500, 500, 100, 500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500,
      500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100, 500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100],
    "setpoint_y": [500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500,
      500, 100,500, 500, 500, 100, 500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500,
      500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500,
      100,500, 500, 500, 100, 500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100,500, 500, 500, 100],
}
```

D Data-sheets

D.1 Sensor Data-sheet

PMW3360DM-T2QU: Optical Gaming Navigation Chip

General Description:

PMW3360DM-T2QU is PixArt Imaging's high end gaming integrated chip which comprises of navigation chip and IR LED integrated in a 16pin molded lead-frame DIP package. It provides best in class gaming experience with the enhanced features of high speed, high resolution, high accuracy and selectable lift detection height to fulfill professional gamers' need. The chip comes with self-adjusting variable frame rate algorithm to enable wireless gaming application. It is designed to be used with LM19-LSI lens to achieve optimum performance.

Key Features:

- Integrated 16 pin molded lead-frame DIP package with IR LED
- Operating Voltage: 1.8V - 2.1V
- Lift detection options
  - Manual lift cut off calibration
  - 2mm
  - 3mm
- High speed motion detection 250ips (typical) and acceleration 50g (max).
- Selectable resolutions up to 12000cpi with 100cpi step size
- Resolution error of 1% (typical)
- Four wire serial port interface (SPI)
- External interrupt output for motion detection
- Internal oscillator — no clock input needed
- Self-adjusting variable frame rate for optimum power performance in wireless application
- Customizable response time and downshift time for rest modes
- Enhanced programmability
  - Angle snapping
  - Angle tunability

Applications:

- Wired and Wireless Optical gaming mice
- Integrated input devices
- Battery-powered input devices

Key Chip Parameters:

Parameter	Value
Power supply Range	1.8V - 2.1V
Optical Lens	1:1
Interface	4 wire Serial Port Interface (SPI)
System Clock	70MHz
Frame Rate	Up to 12000 fps
Speed	250ips (typical)
Resolution	12000 cpi
Package Type	16 pin molded lead-frame DIP package with integrated IR LED

Ordering Information:

Part Number	Package Type
PMW3360DM-T2QU	16pin-DIP
LM19-LSI	Lens

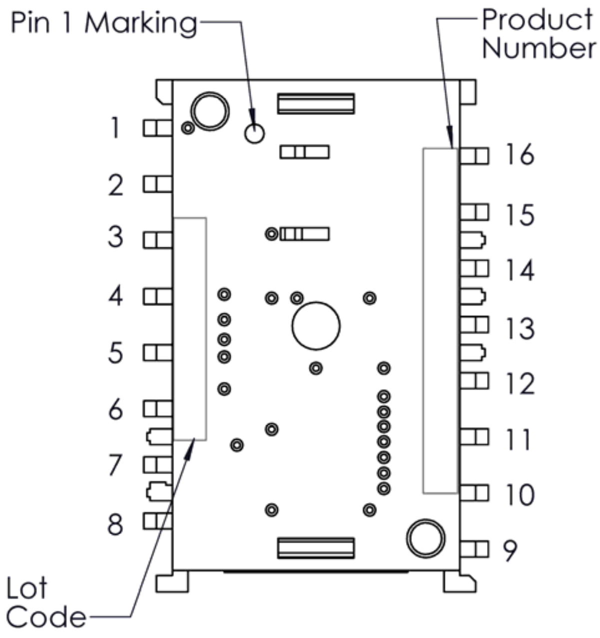
RoHS compliant

Contents

1.0 System Level Description

This section covers PMW3360's guidelines and recommendations in term of chip, lens & PCB assemblies.

1.1 Pin Configuration



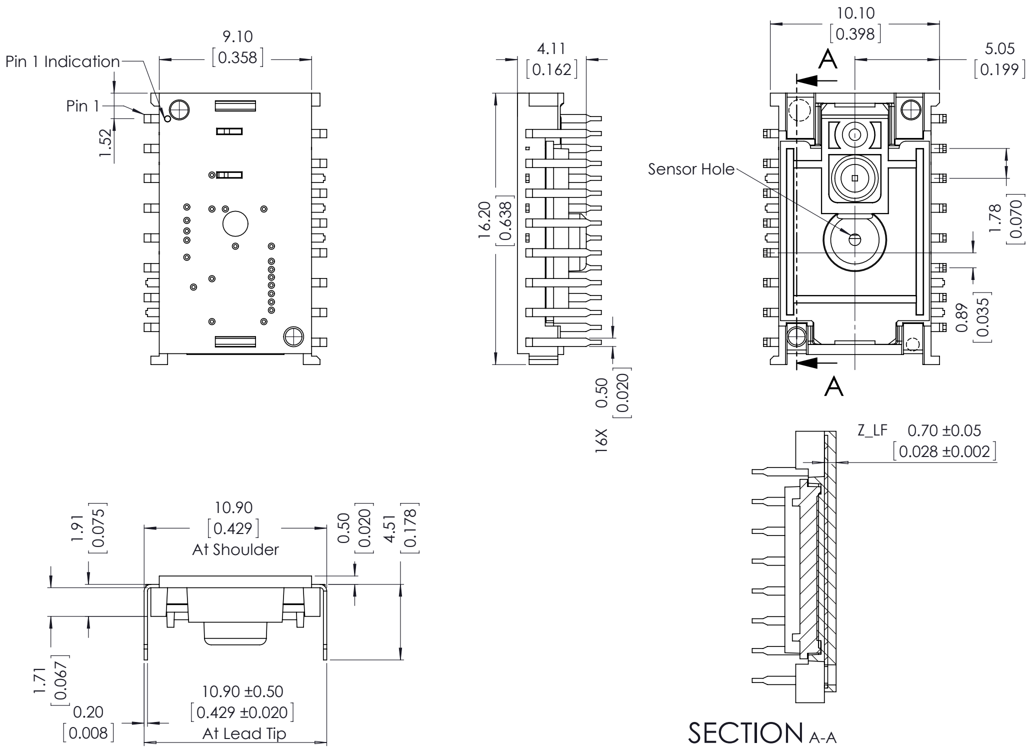
Pin No.	Function	Symbol	Type	Description
1	NA	NC	NC	(Float)
2	NA	NC	NC	(Float)
3	Supply Voltage and I/O Voltage	VDDPIX	Power	LDO output for selective analog circuit
4		VDD	Power	Input power supply
5		VDDIO	Power	I/O reference voltage
6	NA	NC	NC	(Float)
7	Reset control	NRESET	Input	Chip reset(active low)
8	Ground	GND	GND	Ground
9	Motion Output	MOTION	Output	Motion detect
10	4-wire spi communication	SCLK	Input	Serial data clock
11		MOSI	Input	Serial data input
12		MISO	Output	Serial data output
13		NCS	Input	Chip select(active low)
14	NA	NC	NC	(Float)
15	LED	LED_P	Input	LED Anode
16	NA	NC	NC	(Float)

Figure 1. Device output pins

Table 1. PMW3360DM-T2QU Pin Description

Items	Marking	Remark
Product Number	PMW3360DM-T2QU	
Lot Code	AYWWXXXXX	A : Assembly house Y: Year WW: Week XXXXX: PixArt reference

1.2 Package Outline Drawing



Notes:

1. Dimensions in $\frac{\text{milimeter}}{\text{[inches]}}$
2. Dimensional tolerance: ±0.1mm
3. Coplanarity of lead:0.1mm
4. Lead pitch tolerance: ±0.15mm
5. Non-cumulative pitch tolerance: ±0.15mm
6. Maximum flash:0.2mm
7. Angular tolerance: ±3°
8. Chamfer (25° x2) on the taper side of the lead
9. Brackets () indicate reference dimension
10. Document Number:LED_INT_16A_Pkg_007

Figure 2. Package Outline Drawing

CAUTION: It is advised that normal static discharge precautions be taken in handling and assembling of this component to prevent damage and/or degradation which may be induced by ESD.

1.3 Assembly Drawings

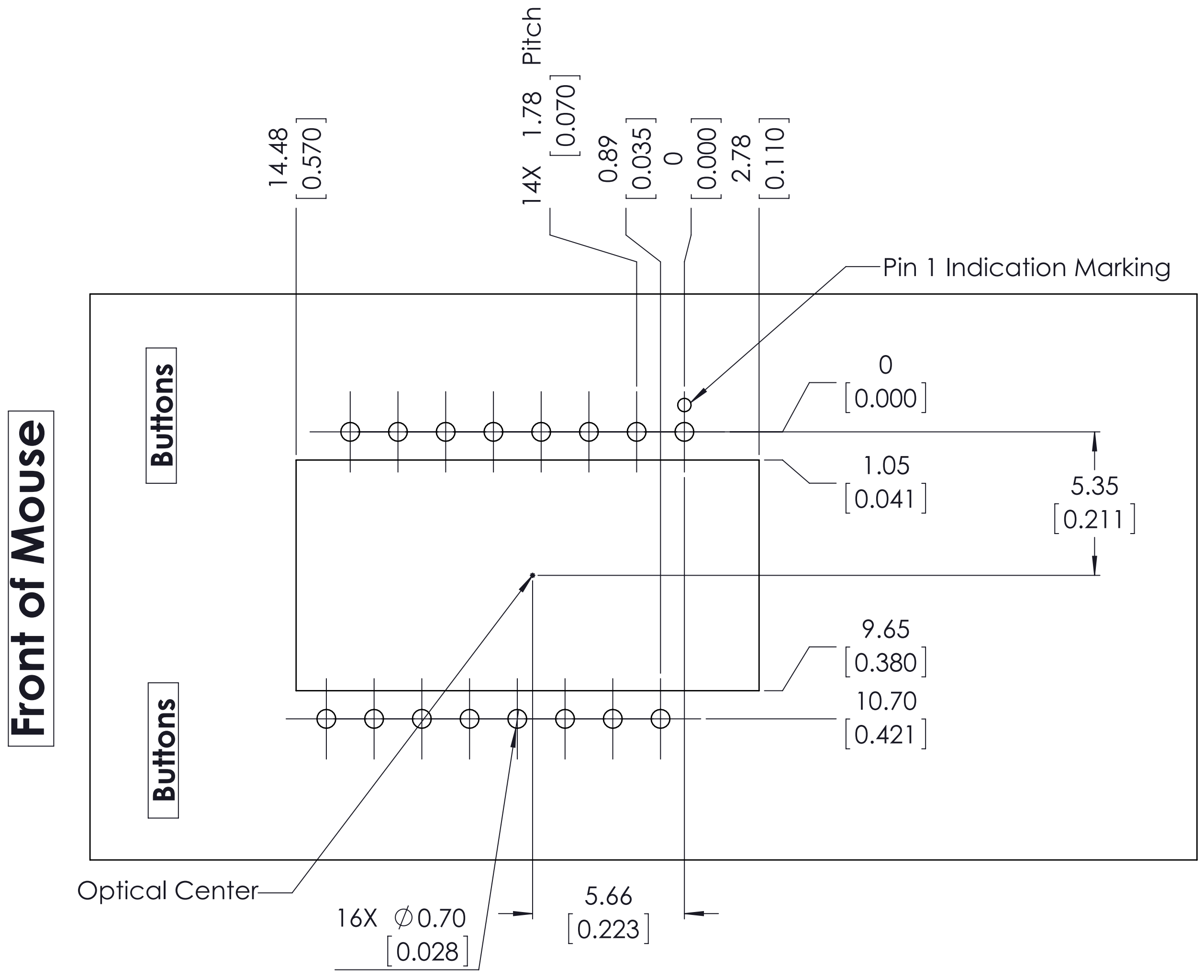


Figure 3. Recommended chip orientation, mechanical cutouts and spacing (Top View)

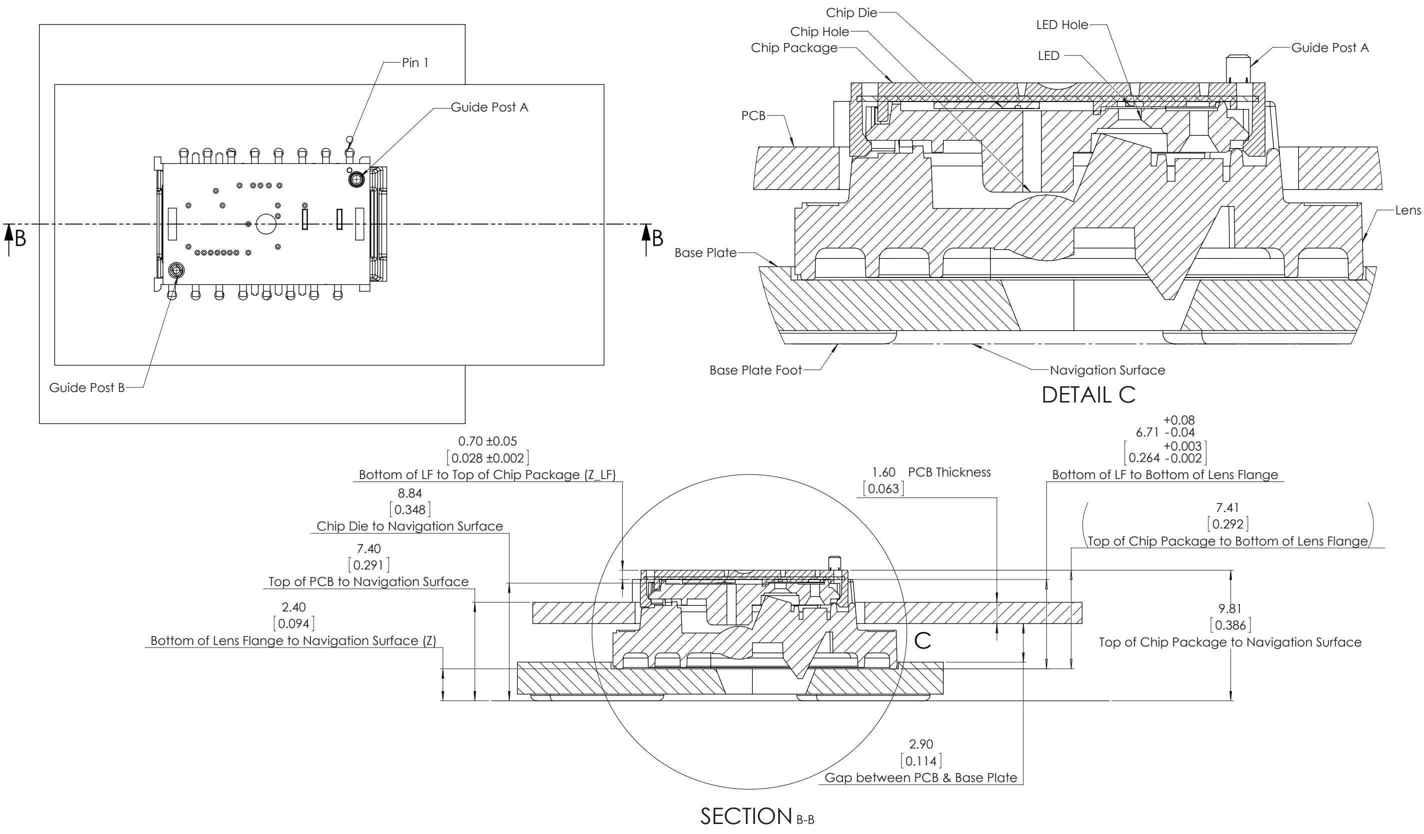


Figure 4. Assembly drawing of PMW3360DM-T2QU and distance from lens reference plane to tracking surface (Z)

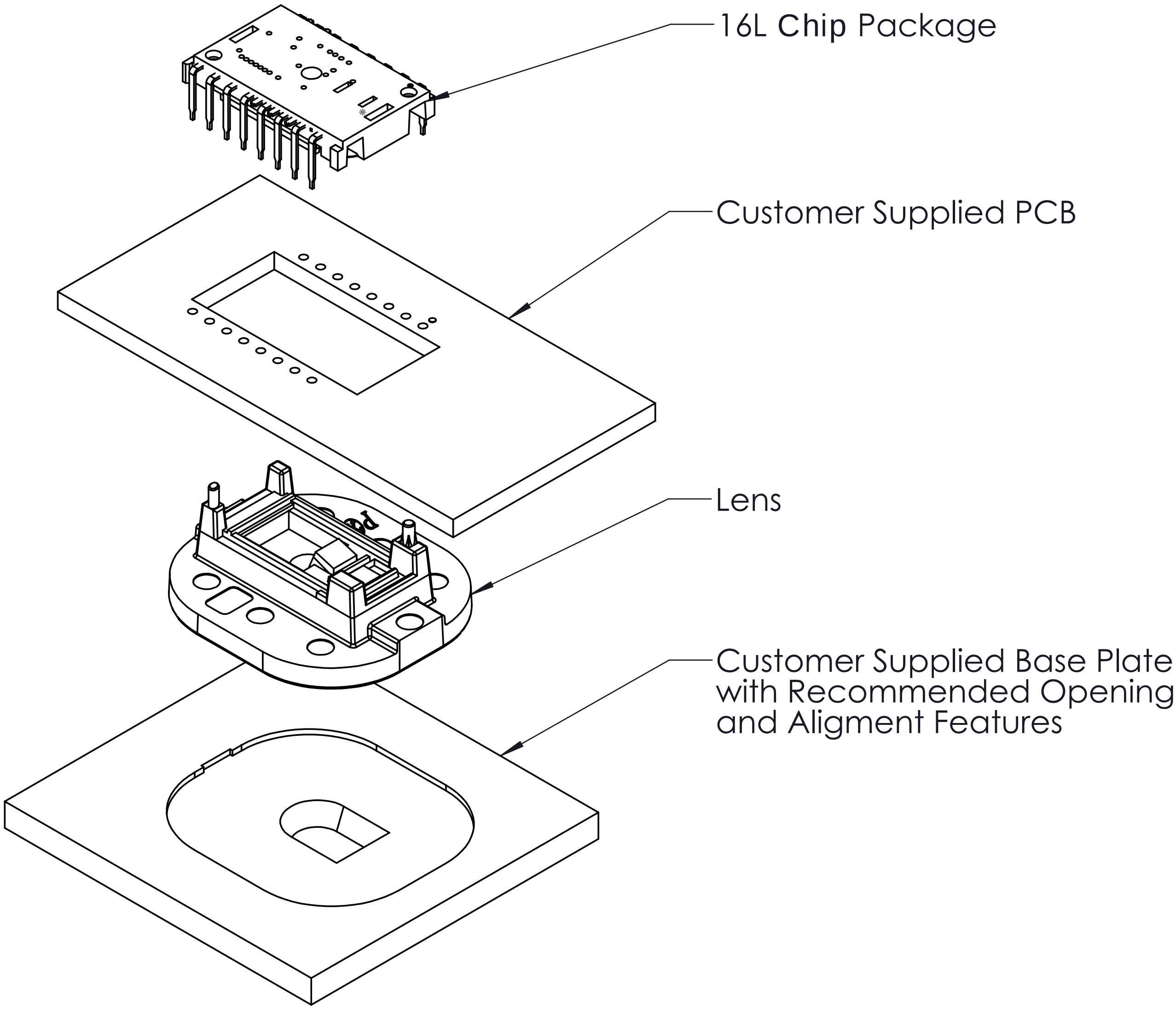


Figure 5. Exploded Assembly View

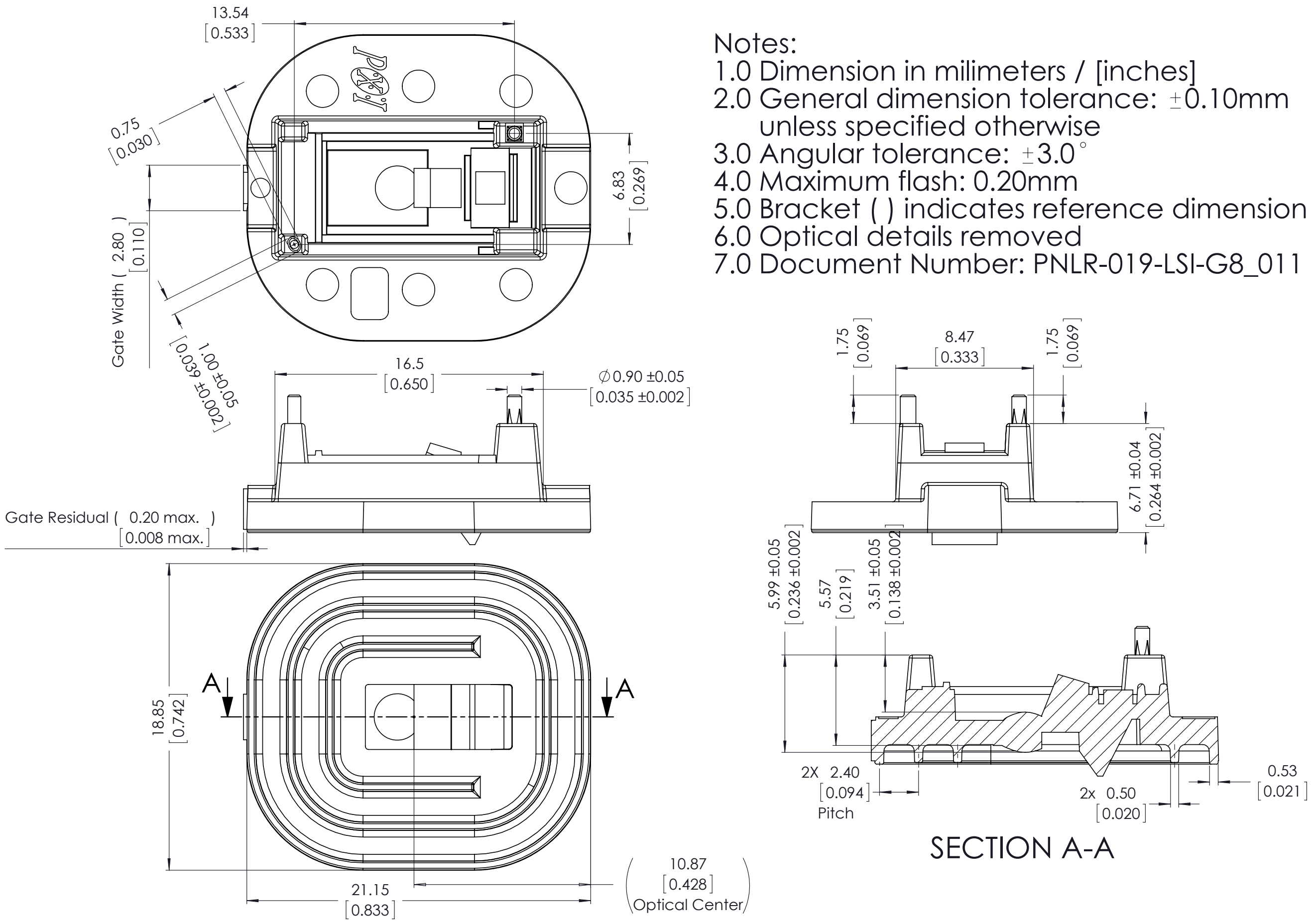


Figure 6. Lens Outline Drawing

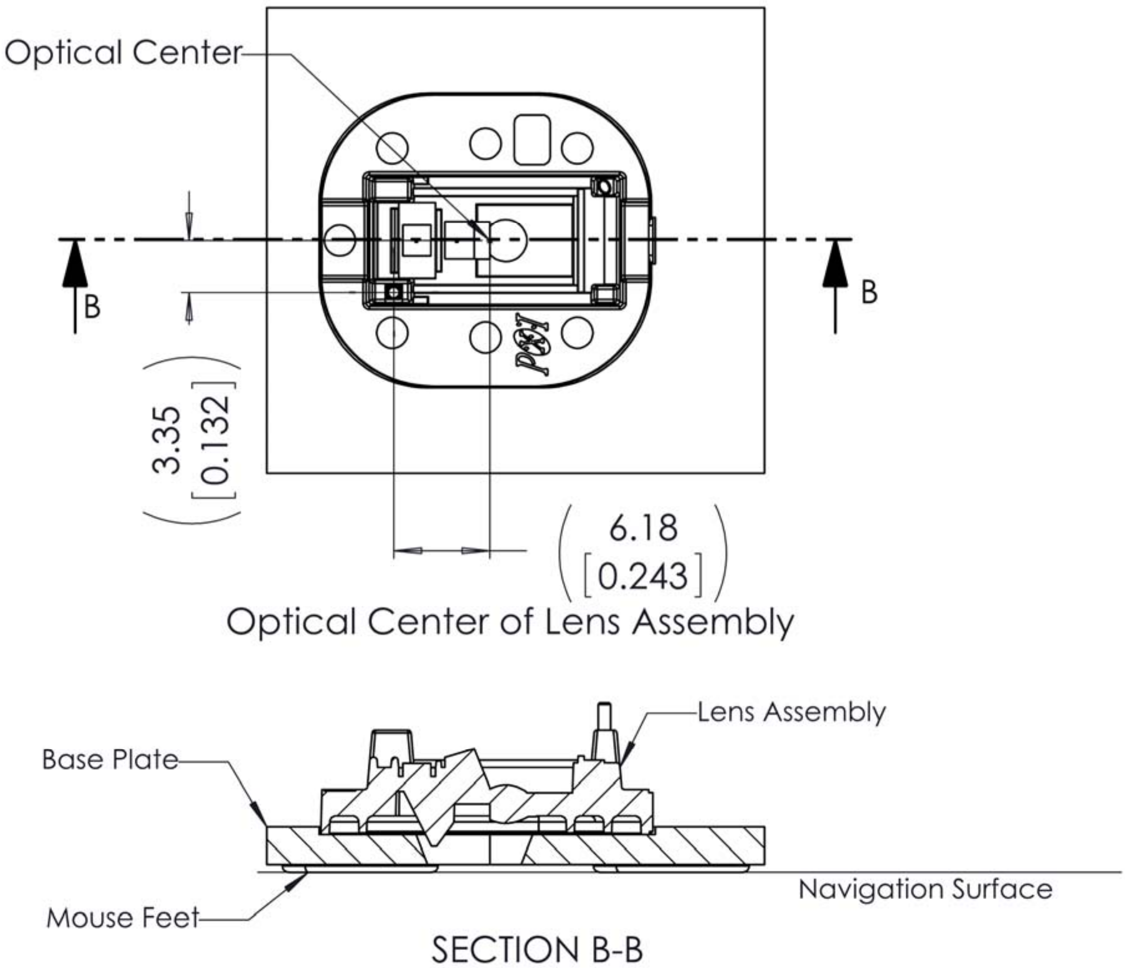


Figure 7. Cross section view of lens assembly

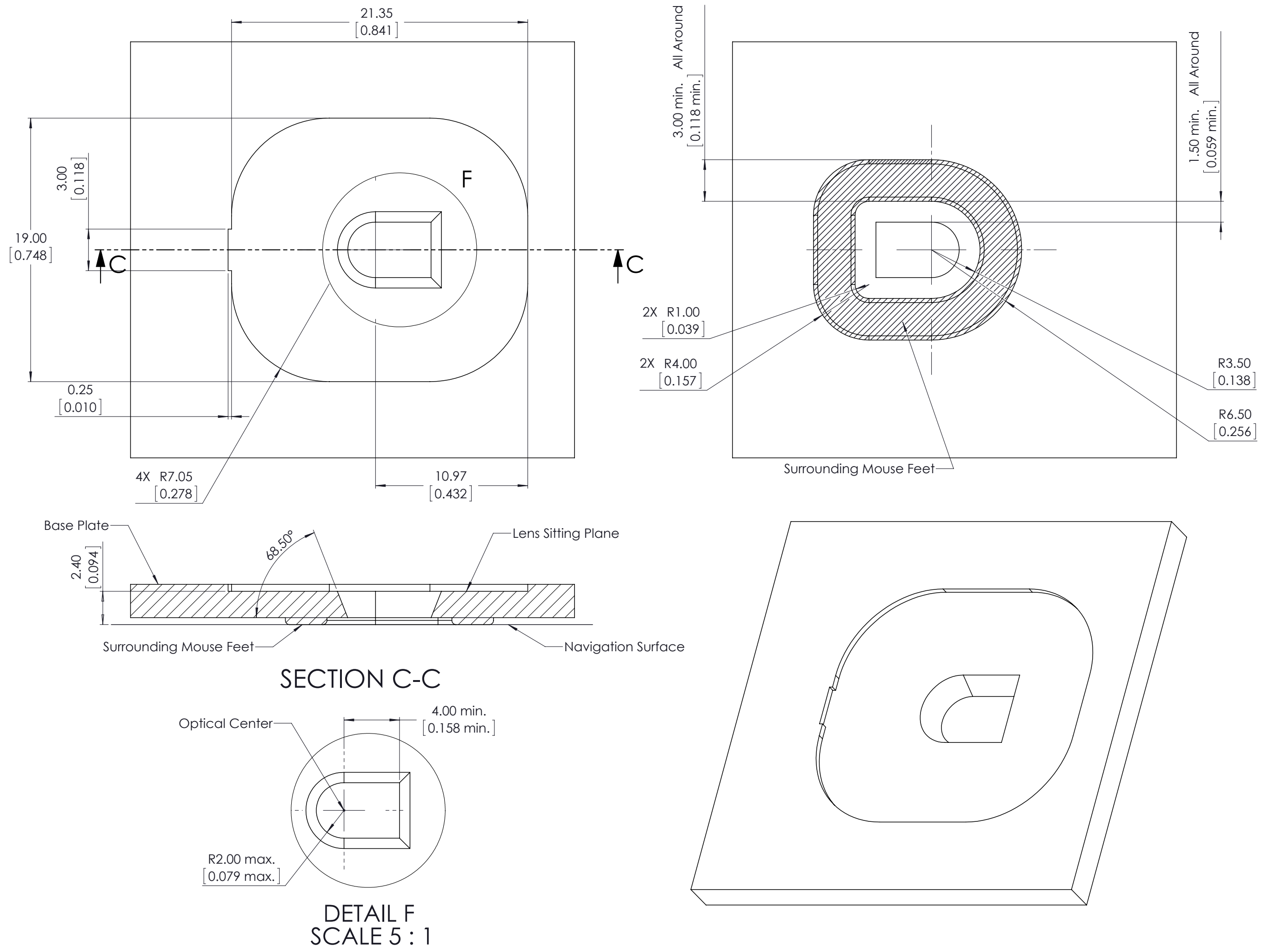


Figure 8. Recommended Base Plate Opening

Note: *Mouse feet should be placed close to the opening to stabilize the surface within the FOV of the chip.*

1.4 PCB Assembly Recommendation

1) Insert the integrated chip and all other electrical components into PCB.
2) Wave-solder the entire assembly in a no-wash solder process utilizing solder-fixture. A solder-fixture is required to protect the chip from flux spray and wave solder.
3) Avoid getting any solder flux onto the chip body as there is potential for flux to seep into the chip package, the solder fixture should be designed to expose only the chip leads to flux spray & molten solder while shielding the chip body and optical apertures. The fixture should also set the chip at the correct position and height on the PCB.
4) Place the lens onto the base plate. Care must be taken to avoid contamination on the optical surfaces.
5) Remove the protective kapton tapes from optical apertures of the chip. Care must be taken to prevent Contaminants from entering the apertures. Do not place the PCB with the chip facing up during the entire mouse assembly process. Hold the PCB vertically when removing kapton tape.
6) Insert PCB assembly over the lens onto the base plate aligning post to retain PCB assembly. The chip package will self-align to the lens via the guide posts. The optical position reference for the PCB is set by the base plate and lens. Note that the PCB motion due to button presses must be minimized to maintain optical alignment.
7) Recommendation: The lens can be permanently secured to the chip package by melting the lens' guide posts over the chip with heat staking process. Please refer to the application note PMS0122-LM19-LSI-AN for more details.
8) Install mouse top case. There must be a feature in the top case to press down onto the PCB assembly to ensure all components are stacked or interlocked to the correct vertical height.

Chip Block Diagram

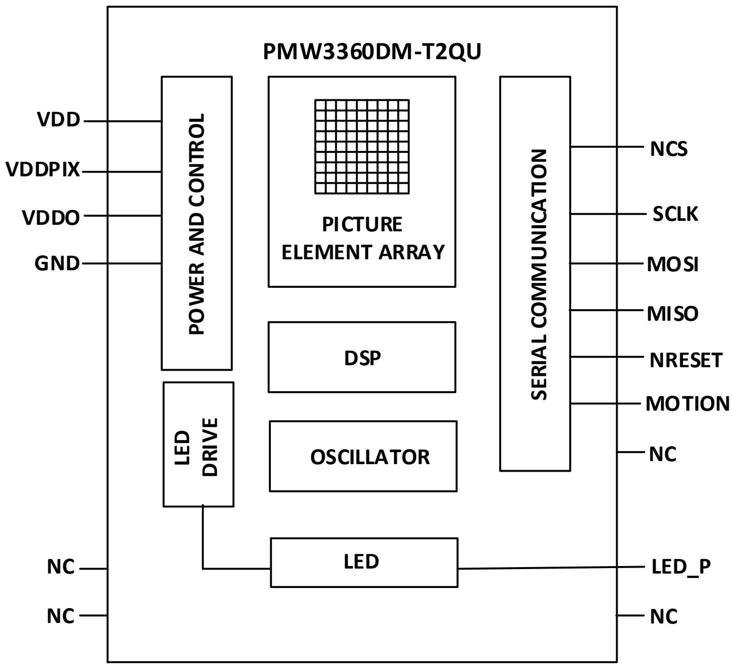


Figure 9. Block diagram of PMW3360DM-T2QU

1.5 Reference Schematics

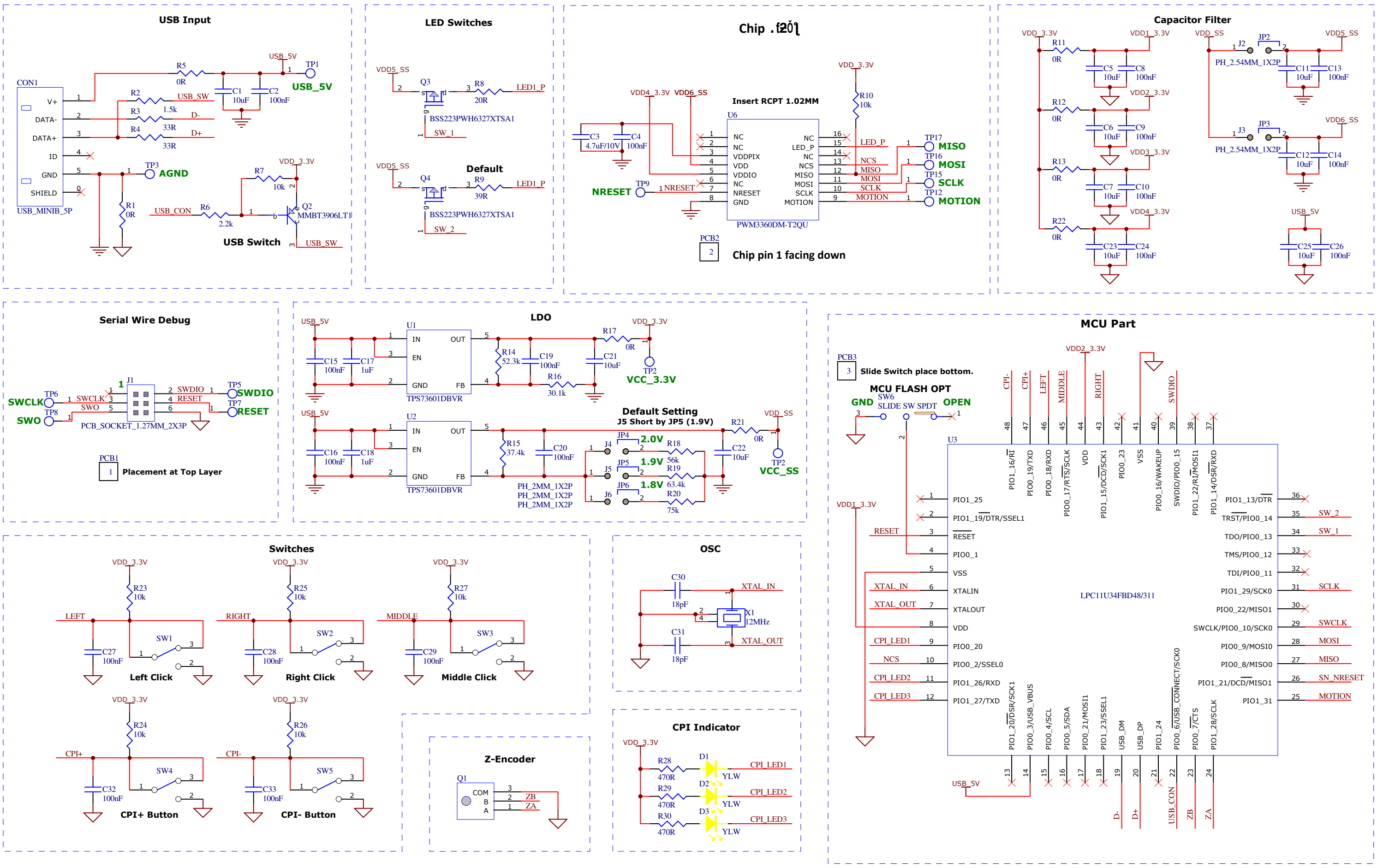


Figure 10. Schematic diagram for interface between PMW3360DM-T2QU and microcontroller on a wired solution

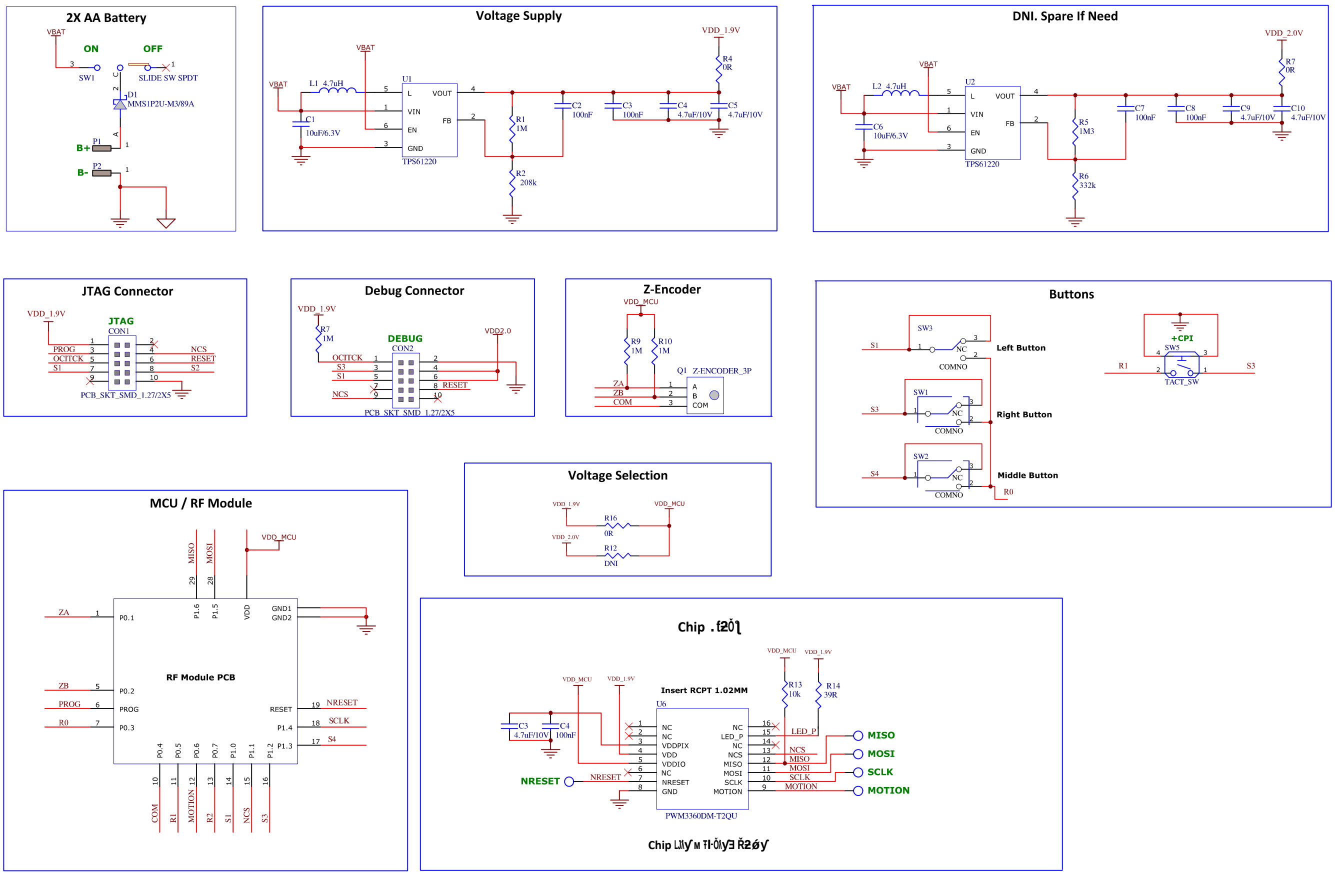


Figure 11. Schematic diagram for interface between PMW3360DM-T2QU and microcontroller on a wireless solution

2.0 Electrical Specifications

Regulatory Requirements

- Passes FCC "Part15, Subpart B, Class B", "CISPR 22 1997 Class B" and worldwide analogous emission limits when assembled into a mouse with shielded cable and following PixArt Imaging's recommendations.
- Passes IEC 62471: 2006 Photo biological safety of lamps and lamp systems

2.1 Absolute Maximum Ratings

Table 2: Absolute Maximum Ratings

Parameter	Symbol	Minimum	Maximum	Units	Notes
Storage Temperature	T_S	-40	85	°C	
Lead Solder Temperature	T_{SOLDER}		260	°C	For 7 seconds, 1.6mm below seating plane.
Supply Voltage	V_{DD}	-0.5	2.10	V	
	V_{DDIO}	-0.5	3.60	V	
ESD (Human Body Model)			2	kV	All pins
Input Voltage	V_{IN}	-0.5	3.6	V	All I/O pins.

2.2 Recommended Operating Conditions

Table 3: Recommended Operating Condition

Parameter	Symbol	Min	Typ.	Max	Units	Notes
Operating Temperature	T_A	0		40	°C	
Power Supply Voltage	V_{DD}	1.80	1.90	2.10	V	excluding supply noise
	V_{DDIO}	1.80	1.90	3.60	V	excluding supply noise. (VDDIO must be same or greater than VDD)
Power Supply Rise Time	t_{RT}	0.15		20	ms	0 to VDD min
Supply Noise (Sinusoidal)	V_{NA}			100	mVp-p	10 kHz —75 MHz
Serial Port Clock Frequency	f_{SCLK}			2.0	MHz	50% duty cycle
Distance from Lens Reference Plane to Tracking Surface	Z	2.2	2.4	2.6	mm	
Speed	S		250		ips	300ips on QCK, Vespula Speed, Vespula Control and FUNC 1030 surfaces
Resolution error	R_{esErr}		1		%	Up to 200ips on QCK with 5000 cpi
Acceleration	A			50	g	In run mode

2.3 AC Electrical Specifications

Table 4. AC Electrical Specifications

Electrical characteristics over recommended operating conditions. Typical values at 25 °C, V_{DD} = 1.9 V, V_{DDIO} = 1.9V.

Parameter	Symbol	Minimum	Typical	Maximum	Units	Notes
Motion Delay After Reset	$t_{MOT\text{-}RST}$	50			ms	From reset to valid motion, assuming motion is present
Shutdown	t_{STDWN}			500	µs	From Shutdown mode active to low current
Wake From Shutdown	t_{WAKEUP}	50			ms	From Shutdown mode inactive to valid motion. Notes: A RESET must be asserted after a shutdown. Refer to section "Notes on Shutdown", also note $t_{MOT\text{-}RST}$
MISO Rise Time	$t_{r\text{-}MISO}$		50		ns	C_L = 100pF
MISO Fall Time	$t_{f\text{-}MISO}$		50		ns	C_L = 100pF
MISO Delay After SCLK	$t_{DLY\text{-}MISO}$			90	ns	From SCLK falling edge to MISO data valid, no load conditions
MISO Hold Time	$t_{hold\text{-}MISO}$	200			ns	Data held until next falling SCLK edge
MOSI Hold Time	$t_{hold\text{-}MOSI}$	200			ns	Amount of time data is valid after SCLK rising edge
MOSI Setup Time	$t_{setup\text{-}MOSI}$	120			ns	From data valid to SCLK rising edge
SPI Time Between Write Commands	t_{SWW}	180			µs	From rising SCLK for last bit of the first data byte, to rising SCLK for last bit of the second data byte.
SPI Time Between Write And Read Commands	t_{SWR}	180			µs	From rising SCLK for last bit of the first data byte, to rising SCLK for last bit of the second address byte.
SPI Time Between Read And Subsequent Commands	t_{SRW} t_{SRR}	20			µs	From rising SCLK for last bit of the first data byte, to falling SCLK for the first bit of the address byte of the next command.
SPI Read Address-Data Delay	t_{SRAD}	160			µs	From rising SCLK for last bit of the address byte, to falling SCLK for first bit of data being read.
SPI Read Address-Data Delay for Burst Mode Motion Read	t_{SRAD_MOTBR}	35			µs	From rising SCLK for last bit of the address byte, to falling SCLK for first bit of data being read. Applicable for Burst Mode Motion Read only.
NCS Inactive After Motion Burst	t_{BEXIT}	500			ns	Minimum NCS inactive time after motion burst before next SPI usage
NCS To SCLK Active	$t_{NCS\text{-}SCLK}$	120			ns	From last NCS falling edge to first SCLK rising edge

Parameter	Symbol	Minimum	Typical	Maximum	Units	Notes
SCLK To NCS Inactive (For Read Operation)	$t_{SCLK\text{-}NCS}$	120			ns	From last SCLK rising edge to NCS rising edge, for valid MISO data transfer
SCLK To NCS Inactive (For Write Operation)	$t_{SCLK\text{-}NCS}$	35			µs	From last SCLK rising edge to NCS rising edge, for valid MOSI data transfer
NCS To MISO High-Z	$t_{NCS\text{-}MISO}$			500	ns	From NCS rising edge to MISO high-Z state
MOTION Rise Time	$t_{r\text{-}MOTION}$		50		ns	C_L = 100pF
MOTION Fall Time	$t_{f\text{-}MOTION}$		50		ns	C_L = 100pF
Input Capacitance	C_{in}		50		pF	SCLK, MOSI, NCS
Load Capacitance	C_L			100	pF	MISO, MOTION
Transient Supply Current	I_{DDT}			70	mA	Max supply current during the supply ramp from 0V to V_{DD} with min 150 us and max 20ms rise time. (Does not include charging currents for bypass capacitors)
	I_{DDTIO}			60	mA	Max supply current during the supply ramp from 0V to V_{DDIO} with min 150 us and max 20ms rise time. (Does not include charging currents for bypass capacitors)

2.4 DC Electrical Specifications

Table 5. DC Electrical Specifications

Electrical characteristics, over recommended operating conditions. Typical values at 25 °C, V_{DD} = 1.9 V, V_{DDIO} = 1.9V, LED current at 12mA, 70MHz (internal), and 1.1kHz (slow clock).

Parameter	Symbol	Min	Typ.	Max	Units	Notes
DC Supply Current	I_{DD_RUN1} I_{DD_RUN2} I_{DD_RUN3} I_{DD_RUN4} I_{DD_REST1} I_{DD_REST2} I_{DD_REST3}		16.3 18.6 21.6 37.0 2.8 61.0 32.0		mA mA mA mA mA uA uA	Average current consumption, including LED current with 1ms polling.
Power Down Current	I_{PD}		10		µA	
Input Low Voltage	V_{IL}			0.3 x V_{DDIO}	V	SCLK, MOSI, NCS
Input High Voltage	V_{IH}	0.7 x V_{DDIO}			V	SCLK, MOSI, NCS
Input Hysteresis	V_{I_HYS}		100		mV	SCLK, MOSI, NCS
Input Leakage Current	I_{leak}		±1	±10	µA	Vin=V_{DDIO} or 0V, SCLK, MOSI, NCS
Output Low Voltage	V_{OL}			0.45	V	Iout=1mA, MISO, MOTION
Output High Voltage	V_{OH}	V_{DDIO} - 0.45			V	Iout=-1mA, MISO, MOTION

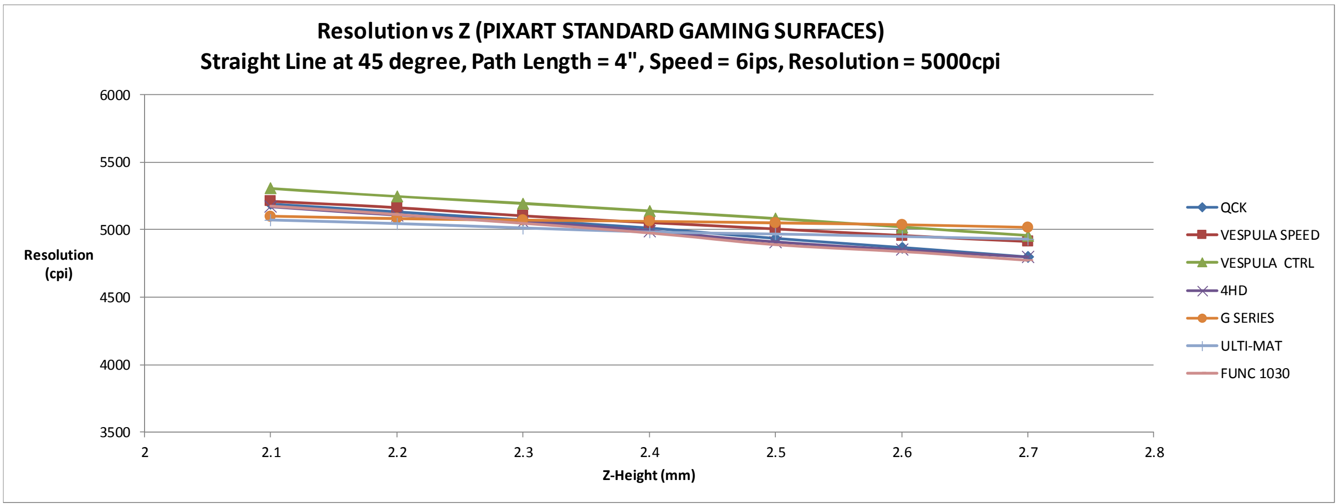


Figure 12 Mean Resolution vs. Z at default resolution at 5000cpi

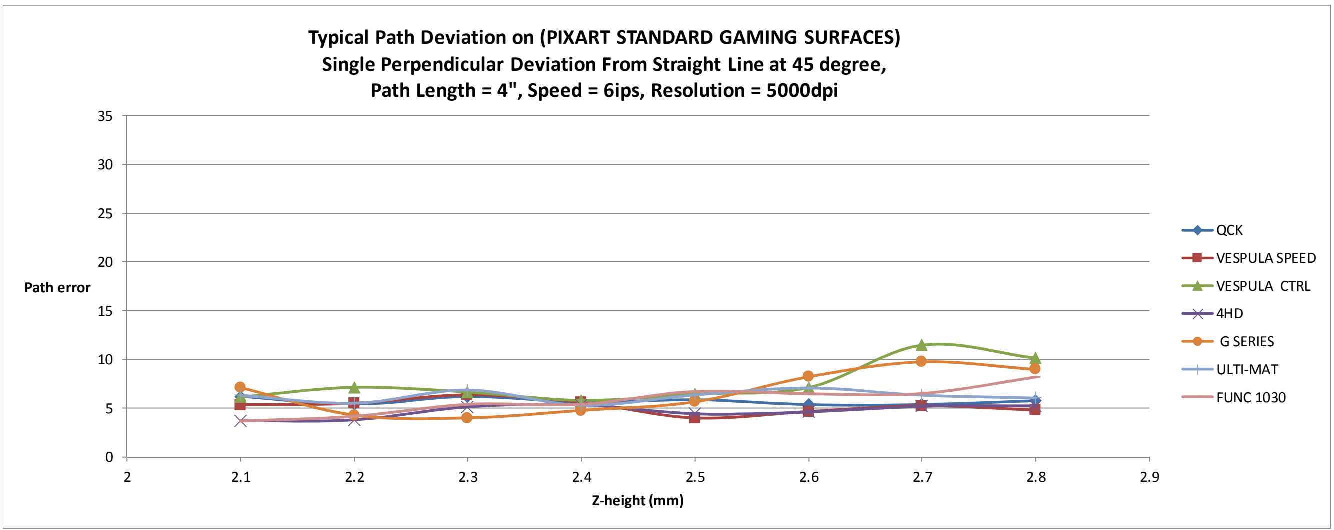


Figure 13 Path error vs. Z-height at default resolution at 5000cpi (mm)

3.0 Serial Peripheral Interface (SPI)

The synchronous serial port is used to set and read parameters in PMW3360DM-T2QU chip, and to read out the motion information. The serial port is also used to load SROM data into PMW3360DM-T2QU chip.

The port is a four wire port. The host microcontroller always initiates communication; PMW3360DM-T2QU chip never initiates data transfers. SCLK, MOSI, and NCS may be driven directly by a microcontroller. The port pins may be shared with other SPI slave devices. When the NCS pin is high, the inputs are ignored and the output is tri-stated.

The lines that comprise the SPI port are:

SCLK	Clock input, generated by the master (microcontroller).
MOSI	Input data. (Master Out/Slave In)
MISO	Output data. (Master In/Slave Out)
NCS	Chip select input (active low). NCS needs to be low to activate the serial port; otherwise, MISO will be high Z, and MOSI & SCLK will be ignored. NCS can also be used to reset the serial port in case of an error.

Motion Pin Timing

The motion pin is an active low output that signals the micro-controller when motion has occurred. The motion pin is lowered whenever the motion bit is set; in other words, whenever there is non-zero data in the Delta_X_L, Delta_X_H, Delta_Y_L or Delta_Y_H registers. Clearing the motion bit (by reading Delta_X_L, Delta_X_H, Delta_Y_L or Delta_Y_H registers) will put the motion pin high.

Chip Select Operation

The serial port is activated after NCS goes low. If NCS is raised during a transaction, the entire transaction is aborted and the serial port will be reset. This is true for all transactions including SROM download. After a transaction is aborted, the normal address-to-data or transaction-to-transaction delay is still required before beginning the next transaction. To improve communication reliability, all serial transactions should be framed by NCS. In other words, the port should not remain enabled during periods of non-use because ESD and EFT/B events could be interpreted as serial communication and put the chip into an unknown state. In addition, NCS must be raised after each burst-mode transaction is complete to terminate burst-mode. The port is not available for further use until burst-mode is terminated.

Write Operation

Write operation, defined as data going from the micro-controller to PMW3360DM-T2QU chip, is always initiated by the micro-controller and consists of two bytes. The first byte contains the address (seven bits) and has a "1" as its MSB to indicate data direction. The second byte contains the data. PMW3360DM-T2QU chip reads MOSI on rising edges of SCLK.

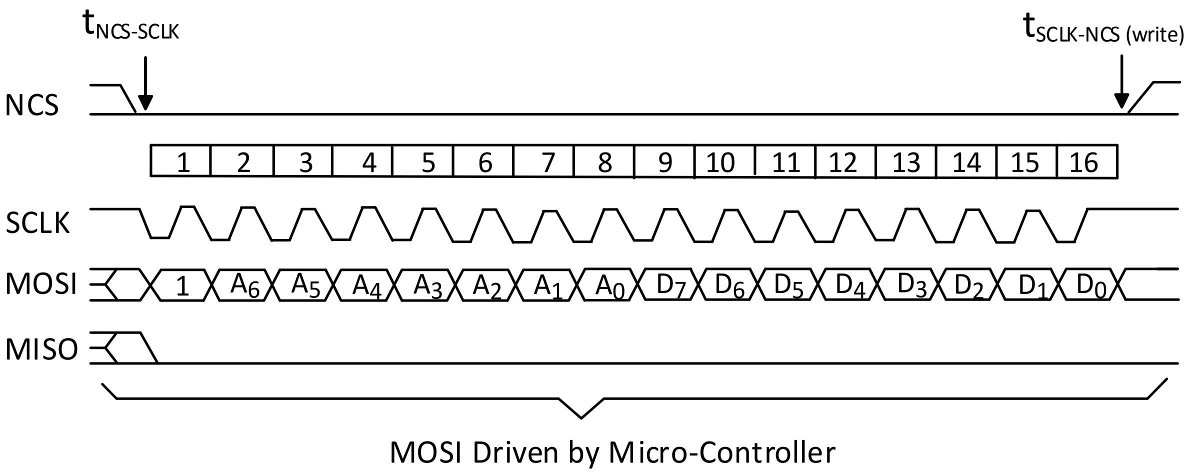


Figure 14. Write operation

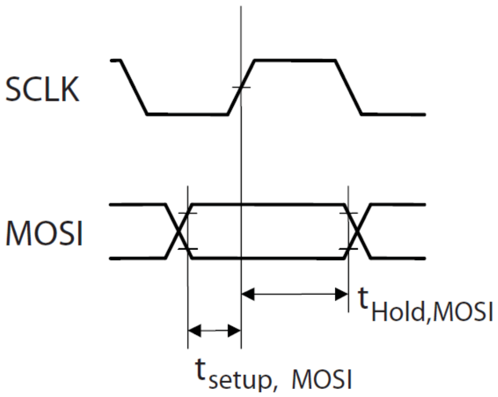


Figure 15. MOSI setup and hold time

Read Operation

A read operation, defined as data going from PMW3360DM-T2QU chip to the micro-controller, is always initiated by the micro-controller and consists of two bytes. The first byte contains the address, is sent by the micro-controller over MOSI, and has a "0" as its MSB to indicate data direction. The second byte contains the data and is driven by PMW3360DM-T2QU chip over MISO. The chip outputs MISO bits on falling edges of SCLK and samples MOSI bits on every rising edge of SCLK.

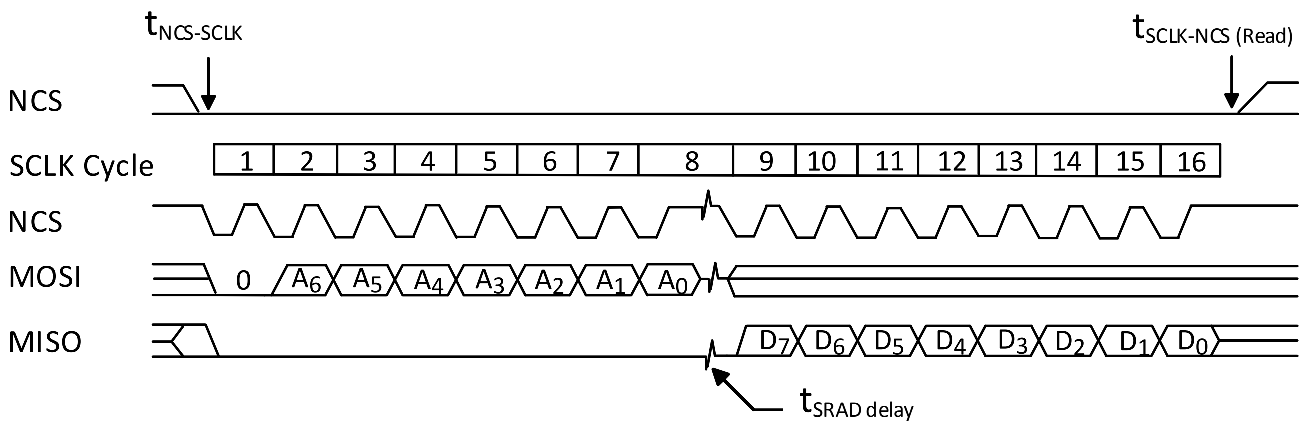


Figure 16. Read operation

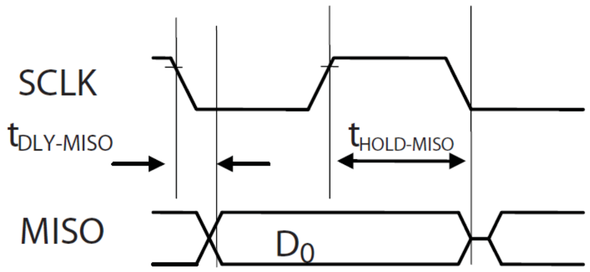


Figure 17. MISO Delay and hold time

Note: The minimum high state of SCLK is also the minimum MISO data hold time of PMW3360DM-T2QU chip. Since the falling edge of SCLK is actually the start of the next read or write command, PMW3360DM-T2QU chip will hold the state of data on MISO until the falling edge of SCLK.

Required timing between Read and Write Commands (tsxx)

There are minimum timing requirements between read and write commands on the serial port.

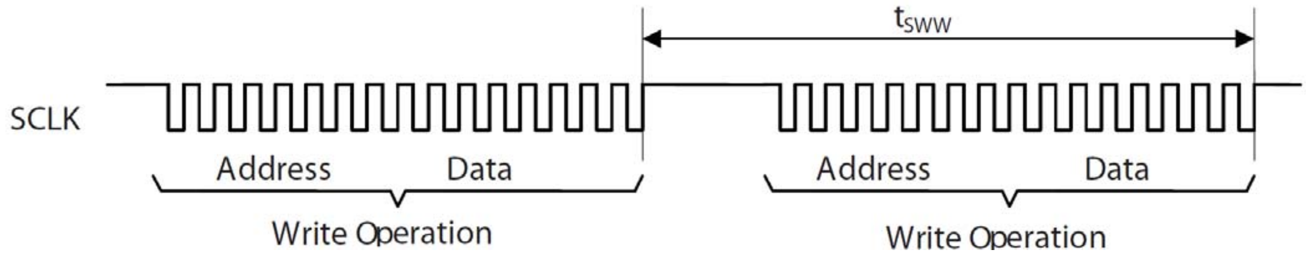


Figure 18. Timing between two write commands

If the rising edge of the SCLK for the last data bit of the second write command occurs before the t_{SWW} delay, then the first write command may not complete correctly.

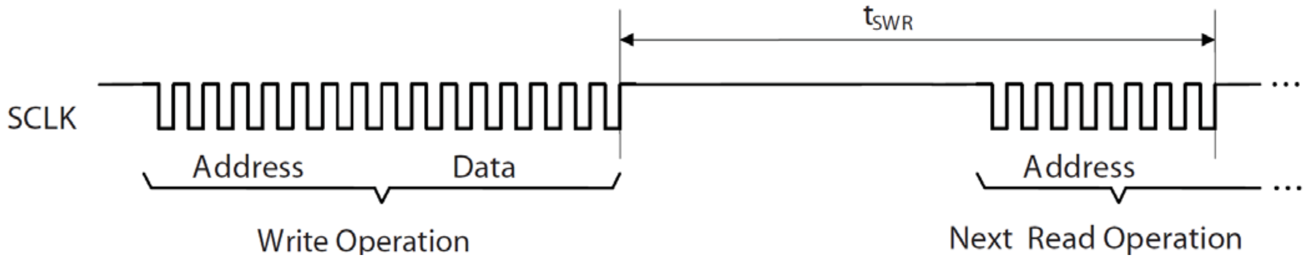


Figure 19. Timing between write and either write or subsequent read commands

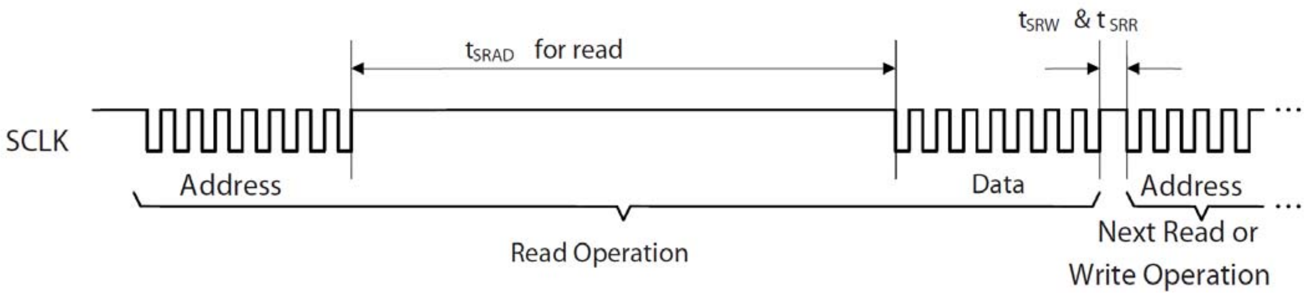


Figure 20. Timing between read and either write or subsequent read commands

If the rising edge of SCLK for the last address bit of the read command occurs before the t_{SWR} required delay, the write command may not complete correctly. During a read operation SCLK should be delayed at least t_{SRAD} after the last address data bit to ensure that the Chip has time to prepare the requested data.

The falling edge of SCLK for the first address bit of either the read or write command must be at least t_{SRR} or t_{SRW} after the last SCLK rising edge of the last data bit of the previous read operation. In addition, during a read operation SCLK should be delayed after the last address data bit to ensure that PMW3360DM-T2QU chip has time to prepare the requested data.

4.0 Burst mode operation

Burst Mode Operation

Burst mode is a special serial port operation mode which may be used to reduce the serial transaction time for three predefined operations: motion read and SROM download and frame capture. The speed improvement is achieved by continuous data clocking to or from multiple registers without the need to specify the register address, and by not requiring the normal delay period between data bytes.

Motion Read

Reading the Motion_Burst register activates this mode. PMW3360DM-T2QU chip will respond with the following motion burst report in order. Motion burst report:

BYTE[00] = Motion
BYTE[01] = Observation
BYTE[02] = Delta_X_L
BYTE[03] = Delta_X_H
BYTE[04] = Delta_Y_L
BYTE[05] = Delta_Y_H
BYTE[06] = SQUAL
BYTE[07] = Raw_Data_Sum
BYTE[08] = Maximum_Raw_Data
BYTE[09] = Minimum_Raw_Data
BYTE[10] = Shutter_Upper
BYTE[11] = Shutter_Lower

After sending the register address, the microcontroller must wait for t_{SRAD_MOTBR}, and then begin reading data. All data bits can be read with no delay between bytes by driving SCLK at the normal rate. The data are latched into the output buffer after the last address bit is received. After the burst transmission is complete, the microcontroller must raise the NCS line for at least t_{BEXIT} to terminate burst mode. The serial port is not available for use until it is reset with NCS, even for a second burst transmission.

Procedure to start motion burst:

1. Write any value to Motion_Burst register.
2. Lower NCS
3. Send Motion_Burst address (0x50).
4. Wait for t_{SRAD_MOTBR}
5. Start reading SPI Data continuously up to 12 bytes. Motion burst may be terminated by pulling NCS high for at least t_{BEXIT}.
6. To read new motion burst data, repeat from step 2.
7. If a non-burst register read operation was executed; then, to read new burst data, start from step 1 instead.

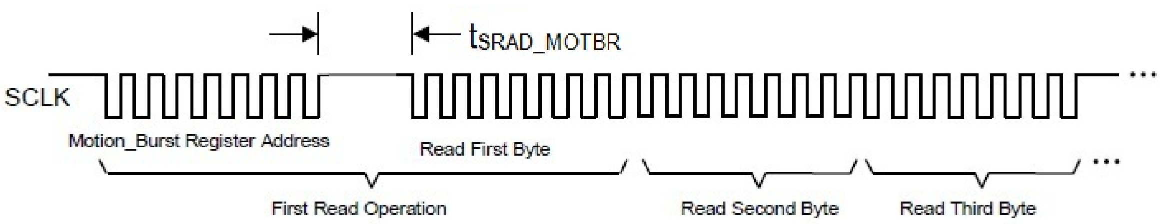


Figure 21. Motion Read sequence for step 3 to 5

Note: Motion burst data can be read from the Motion_Burst registers even in run or rest mode.

5.0 SROM Download

This function is used to load the supplied firmware file contents into PMW3360DM-T2QU after chip power up sequence. The firmware file is an ASCII text file.

SROM download procedure:
1. Perform the Power-Up sequence (steps 1 to 8)
2. Write 0 to Rest_En bit of Config2 register to disable Rest mode.
3. Write 0x1d to SROM_Enable register for initializing
4. Wait for 10 ms
5. Write 0x18 to SROM_Enable register again to start SROM Download
6. Write SROM file into SROM_Load_Burst register, 1st data must start with SROM_Load_Burst address. All the SROM data must be downloaded before SROM starts running.
7. Read the SROM_ID register to verify the ID before any other register reads or writes.
8. Write 0x00 to Config2 register for wired mouse or 0x20 for wireless mouse design.

The SROM download success may be verified in two ways. Once execution from SROM space begins, the SROM_ID register will report the firmware version. At any time, a self-test may be executed which performs a CRC on the SROM contents and reports the results in a register. Take note that the self-test does disrupt tracking performance and also reset registers to default value. The test is initiated by writing 0x15 to the SROM_Enable register and the result is placed in the Data_Out_Lower and Data_Out_Upper registers. See register description for more details.

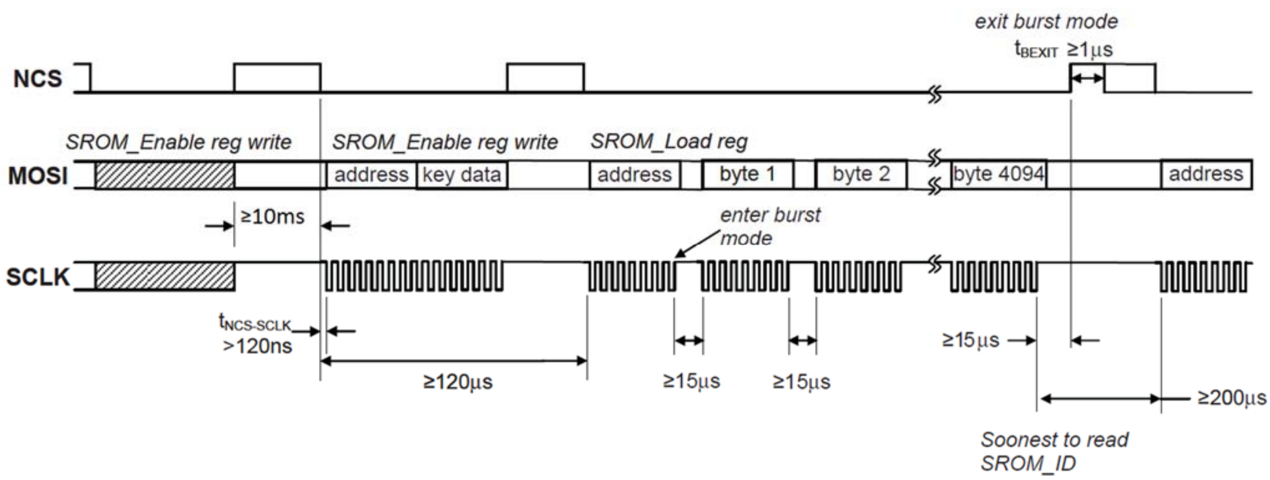


Figure 22. SROM Download Burst Mode

6.0 Frame Capture

This is a fast way to download a full array of raw data values from a single frame. This mode disables navigation and overwrites any downloaded firmware. A hardware reset is required to restore navigation, and the firmware must be reloaded.

To trigger the capture, write to the Frame_Capture register. The next available complete 1 frame image will be stored to memory. The data is retrieved by reading the Raw_Data_Burst register using burst read method per the waveform below. If the Raw_ Data_Burst register is read before the data is ready (step 6 below), it will return all zeros.

Frame Capture procedure:

1. The chip should be powered up and reset correctly (SROM download should be part of this powered up and reset sequence - refer to Power Up sequence in data sheet for more information).
2. Wait for 250ms.
3. Write 0 to Rest_En bit of Config2 register to disable Rest mode.
4. Write 0x83 to Frame_Capture register.
5. Write 0xC5 to Frame_Capture register.
6. Wait for 20ms.
7. Continue burst read from Raw_data_Burst register until all 1296 raw data are transferred.
8. Continue step 1-8 to capture another frame.

Note: Manual reset and SROM download are needed after frame capture to restore navigation.

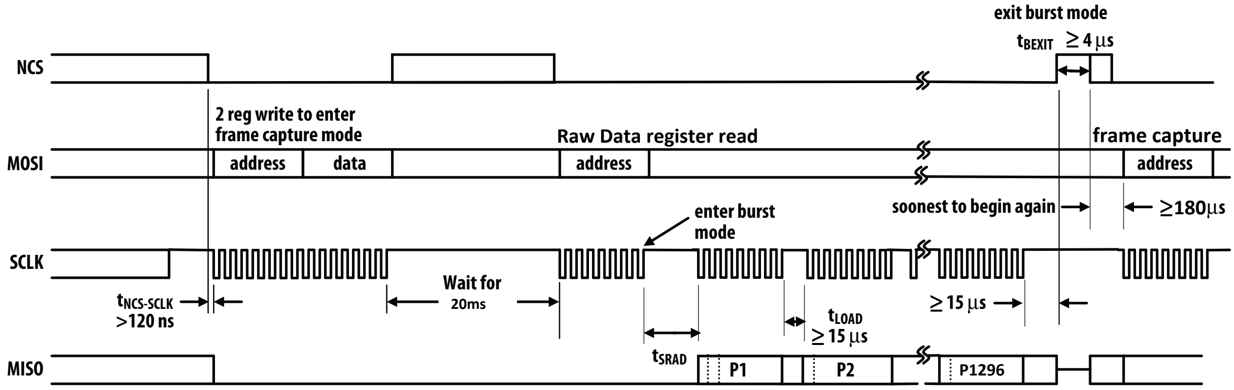
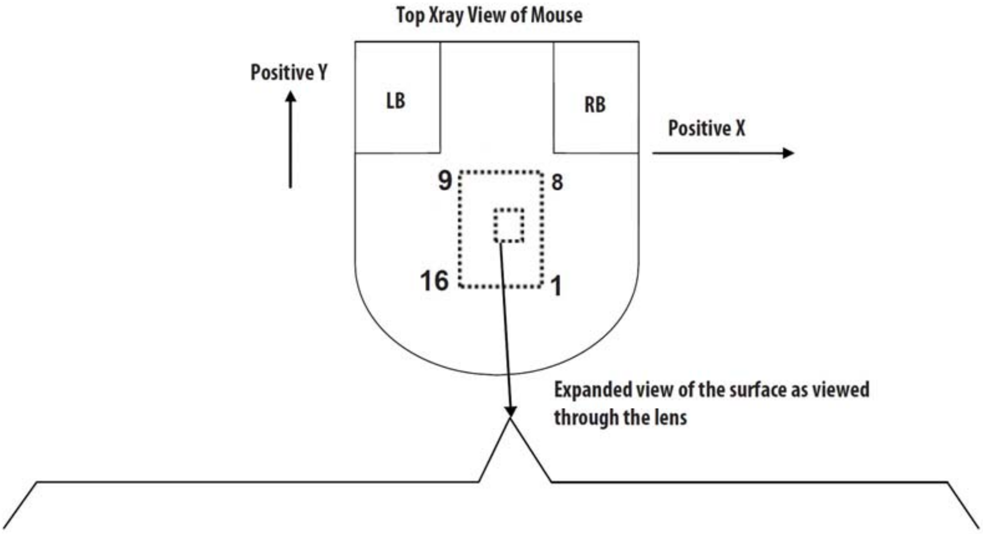


Figure 23. Frame Capture Burst Mode



Last output

35	71	107	143	179	215	251	287	323	359	395	431	467	503	539	575	611	647	683	719	755	791	827	863	899	935	971	1007	1043	1079	1115	1151	1187	1223	1259	1295
34	70	106	142	178	214	250	286	322	358	394	430	466	502	538	574	610	646	682	718	754	790	826	862	898	934	970	1006	1042	1078	1114	1150	1186	1222	1258	1294
33	69	·	·	·	etc																														
																														etc	·	·	·	1227	1263
2	38	74	110	146	182	218	254	290	326	362	398	434	470	506	542	578	614	650	686	722	758	794	830	866	902	938	974	1010	1046	1082	1118	1154	1190	1226	1262
1	37	73	109	145	181	217	253	289	325	361	397	433	469	505	541	577	613	649	685	721	757	793	829	865	901	937	973	1009	1045	1081	1117	1153	1189	1225	1261
0	36	72	108	144	180	216	252	288	324	360	396	432	468	504	540	576	612	648	684	720	756	792	828	864	900	936	972	1008	1044	1080	1116	1152	1188	1224	1260

First output

Figure 24. Raw data Map (Surface referenced)

7.0 Power Up

Although the chip performs an internal power up self reset, it is still recommend that the Power_Up_Reset register is written every time power is applied. The appropriate sequence is as follows:

1. Apply power to VDD and VDDIO in any order, with a delay of no more than 100ms in between each supply. Ensure all supplies are stable.
2. Drive NCS high, and then low to reset the SPI port.
3. Write 0x5A to Power_Up_Reset register (or, alternatively toggle the NRESET pin).
4. Wait for at least 50ms.
5. Read from registers 0x02, 0x03, 0x04, 0x05 and 0x06 one time regardless of the motion pin state.
6. Perform SROM download.
7. Load configuration for other registers.

During power-up there will be a period of time after the power supply is high but before normal operation. The table below shows the state of the various pins during power-up and reset.

State of Signal Pins After VDD is Valid		
Pin	During Reset	After Reset
NRESET	Functional	Functional
NCS	Ignored	Functional
MISO	Undefined	Depends on NCS
SCLK	Ignored	Depends on NCS
MOSI	Ignored	Depends on NCS
MOTION	Undefined	Functional

NRESET
The NRESET pin can be used to perform a full chip reset. When asserted, it performs the same reset function as the Power_Up_Reset_Register. The NRESET pin needs to be asserted (held to logic 0) for at least 100 ns.

Note:- NRESET pin has a built in weak pull up circuit. During active low reset phase, it can draw a static current of up to 600uA.

8.0 Shutdown

PMW3360DM-T2QU can be set in Shutdown mode by writing to Shutdown register. The SPI port should not be accessed when Shutdown mode is asserted, except the power-up command (writing 0x5a to register 0x3a). Other ICs on the same SPI bus can be accessed, as long as the chip's NCS pin is not asserted. The SROM download is required when wake up from Shutdown mode.

To de-assert Shutdown mode:
1. Drive NCS high, and then low to reset the SPI port.
2. Write 0x5A to Power_Up_Reset register (or, alternatively toggle the NRESET pin).
3. Wait for at least 50ms.
4. Read from registers 0x02, 0x03, 0x04, 0x05 and 0x06 one time regardless of the motion pin state.
5. Perform SROM download.
6. Load configuration for other registers.

Pin	Status when Shutdown Mode
NRESET	High
NCS	High*1
MISO	Hi-Z*2
SCLK	Ignore if NCS = 1*3
MOSI	Ignore if NCS = 1*4
MOTION	Output High

*1. NCS pin must be held to 1 (high) if SPI bus is shared with other devices. It is recommended to hold to 1 (high) during Shutdown unless powering up the chip. It must be held to 0 (low) if the chip is to be re-powered up from shutdown (writing 0x5a to register 0x3a).
*2. MISO should be either pull up or down during shutdown in order to meet the low power consumption specification in the datasheet.
*3. SCLK is ignored if NCS is 1 (high). It is functional if NCS is 0 (low).
*4. MOSI is ignored if NCS is 1 (high). If NCS is 0 (low), any command present on the MOSI pin will be ignored except power-up command (writing 0x5a to register 0x3a).

Note:- There are long wakeup times from shutdown. These features should not be used for power management during normal mouse motion.

9.0 Lift cut off calibration

This chip has the capability to optimize its lift performance by tuning internal parameters to the surface. This "Lift cut off calibration" feature involves user interaction.

Take note that the Lift cut off calibration procedure that follows references registers of seven Lift cut off calibration related registers: (i) LiftCutoff_Tune1, (ii) LiftCutoff_Tune2, (iii) LiftCutoff_Tune3, (iv) LiftCutoff_Tune_Timeout, (v) LiftCutoff_Tune_Min_Length, (vi) Raw data_Threshold and (vii) Min_SQ_Run.

1. Ensure that the chip is powered up according to the Power Up Sequence.
2. Ensure that Lift cut off calibration SROM*[1] is downloaded.
3. Delay for 30ms.
4. Prompt the user that the "Lift cut off calibration" procedure is about to begin to ensure that the mouse is placed nominally on the surface (mouse is not lifted).
5. Start the calibration procedure by setting RUN_CAL register bit to 1. The calibration procedure can be started by a SW prompt to the user or user-initiated through a mouse-click event.
6. Poll CAL_STAT[2:0] to check the status of the calibration procedure. There are three ways to successfully stop the calibration procedure: set RUN_CAL register bit to 0 if either:
    - CAL_STAT[2:0] = 0x02,
    - CAL_STAT[2:0] = 0x02 and user initiates a stop through a mouse-click event, or,
    - CAL_STAT[2:0] = 0x03.

    If CAL_STAT[2:0] = 0x04, the calibration procedure needs to be re-started.
7. Stop the calibration procedure by ensuring that the RUN_CAL register bit is 0, then wait 1msec before reading the recommended "Raw data Threshold" register value, RPTH[6:0] (lower 7 bits of LiftCutoff_Tune2 register). RPTH[6:0] recommends a raw data threshold value that replaces the default value in the tracking SROM's Raw_data_Threshold register to improve lift performance. The Raw_ data_Threshold register requires the Tracking SROM*[2] to be loaded.
8. Read the recommended "Min SQUAL Run" register value, RMSQ[7:0] (entire 8 bits of LiftCutoff_Tune3 register). RMSQ[7:0] recommends a Min SQUAL Run value that replaces the default value in the tracking SROM's Min_SQ_Run register to improve lift performance. The Min_SQ_Run register requires the Tracking SROM*[2] to be downloaded.
9. The Lift cut off calibration procedure is complete.

Note:
**[1] Lift cut off calibration SROM: SROM 0x81 or above (4KB).*
**[2] Tracking SROM: SROM 0x03 or above (4KB).*

10.0 Registers Table

PMW3360DM-T2QU registers are accessible via the serial port. The registers are used to read motion data and status as well as to set the device configuration.

Address	Register	Access (R = Read / W = Write or Read/Write= RW)	Default Value
0x00	Product_ID	R	0x42
0x01	Revision_ID	R	0x01
0x02	Motion	RW	0x20
0x03	Delta_X_L	R	0x00
0x04	Delta_X_H	R	0x00
0x05	Delta_Y_L	R	0x00
0x06	Delta_Y_H	R	0x00
0x07	SQUAL	R	0x00
0x08	Raw_Data_Sum	R	0x00
0x09	Maximum_Raw_data	R	0x00
0x0A	Minimum_Raw_data	R	0x00
0x0B	Shutter_Lower	R	0x12
0x0C	Shutter_Upper	R	0x00
0x0D	Control	RW	0x02
0x0F	Config1	RW	0x31
0x10	Config2	RW	0x20
0x11	Angle_Tune	RW	0x00
0x12	Frame_Capture	RW	0x00
0x13	SROM_Enable	W	N/A
0x14	Run_Downshift	RW	0x32
0x15	Rest1_Rate_Lower	RW	0x00
0x16	Rest1_Rate_Upper	RW	0x00
0x17	Rest1_Downshift	RW	0x1F
0x18	Rest2_Rate_Lower	RW	0x63
0x19	Rest2_Rate_Upper	RW	0x00
0x1A	Rest2_Downshift	RW	0xBC
0x1B	Rest3_Rate_Lower	RW	0xF3
0x1C	Rest3_Rate_Upper	RW	0x01
0x24	Observation	RW	0x00
0x25	Data_Out_Lower	R	0x00
0x26	Data_Out_Upper	R	0x00
0x29	Raw_Data_Dump	RW	0x00
0x2A	SROM_ID	R	0x00
0x2B	Min_SQ_Run	RW	0x10
0x2C	Raw_Data_Threshold	RW	0x0A
0x2F	Config5	RW	0x31
0x3A	Power_Up_Reset	W	N/A
0x3B	Shutdown	W	N/A
0x3F	Inverse_Product_ID	R	0xBD
0x41	LiftCutoff_Tune3	RW	0x00
0x42	Angle_Snap	RW	0x00
0x4A	LiftCutoff_Tune1	RW	0x00
0x50	Motion_Burst	RW	0x00
0x58	LiftCutoff_Tune_Timeout	RW	0x27
0x5A	LiftCutoff_Tune_Min_Length	RW	0x09
0x62	SROM_Load_Burst	W	N/A
0x63	Lift_Config	RW	0x02
0x64	Raw_Data_Burst	R	0x00
0x65	LiftCutoff_Tune2	R	0x00

11.0 Registers Description

Register: 0x00								
Name: Product_ID								
Bit	7	6	5	4	3	2	1	0
Field	PID_7	PID_6	PID_5	PID_4	PID_3	PID_2	PID_1	PID_0
	Reset Value: 0x42							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	This value is a unique identification assigned to this model only. The value in this register does not change; it can be used to verify that the serial communications link is functional.							

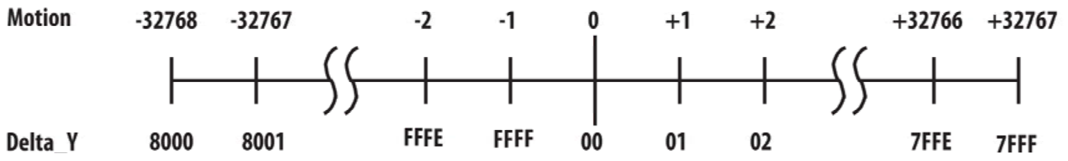
Register: 0x01								
Name: Revision_ID								
Bit	7	6	5	4	3	2	1	0
Field	RID_7	RID_6	RID_5	RID_4	RID_3	RID_2	RID_1	RID_0
	Reset Value: 0x01							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	This register contains the current IC revision, the revision of the permanent internal firmware. It is subject to change when new IC versions are released.							

Register: 0x02								
Name: Motion								
Bit	7	6	5	4	3	2	1	0
Field	MOT	Reserved	1	RData_1st	Lift_Stat	OP_MODE_1	OP_MODE_2	FRAME_RData_1st
	Reset Value: 0x20							
Access: R/W	Read/ Write							
Data Type:	8-bit Field							
Usage	This register allows the user to determine if motion has occurred since the last time it was read. The procedure to read the motion registers (Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H) is as follows: 1. Write any value to the Motion register. 2. Read the Motion register. This will freeze the Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H register values. 3. If the MOT bit is set, Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H registers should be read in the given sequence to get the accumulated motion. Note: if Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H registers are not read before the motion register is read for the second time, the data in Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H will be lost. 4. To read a new set of motion data (Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H), repeat from Step 2. 5. If any other register was read i.e. any other register besides Motion, Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H, then, to read a new set of motion data, repeat from Step 1 instead.							

Field Name	Description
MOT	Motion since last report or PD 0 = No motion 1 = Motion occurred, data ready for reading in Delta_X_L, Delta_X_H, Delta_Y_L and Delta_Y_H registers
[6]	Reserved.
[5]	1
RData_1st	This bit is set when the Raw_Data_Grab register is written to or when a complete raw data array has been read, initiating an increment to raw data 0,0. 0 = Raw_Data_Grab data not from raw data 0,0 1 = Raw_Data_Grab data is from raw data 0,0
Lift_Stat	Indicate the lift status of Chip, 0 = Chip on surface. 1 = Chip lifted.
OP_Mode[1:0]	00 – Run mode 01 – Rest 1 10 – Rest 2 11 – Rest 3
FRAME_RData_1st	This bit is set to indicate first raw data in frame capture. 0 = Frame capture data not from raw data 0,0 1 = Frame capture data is from raw data 0,0

Register: 0x03								
Name: Delta_X_L								
Bit	7	6	5	4	3	2	1	0
Field	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	16 bits 2's complement number. Lower 8 bits of Delta_X.							
Usage	X movement is counts since last report. Absolute value is determined by resolution. Reading it clears the register.							

Register: 0x04								
Name: Delta_X_H								
Bit	7	6	5	4	3	2	1	0
Field	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_8
	Reset Value: 0x04							
Access: R/W	Read Only							
Data Type:	16 bits 2's complement number. Lower 8 bits of Delta_X.							
Usage	Delta_X_H must be read after Delta_X_L to have the full motion data. Reading it clears the register.							

Register: 0x05								
Name: Delta_Y_L								
Bit	7	6	5	4	3	2	1	0
Field	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	16 bits 2's complement number. Lower 8 bits of Delta_Y.							
Usage	Y movement is counts since last report. Absolute value is determined by resolution. Reading it clears the register. 							

Register: 0x06								
Name: Delta_Y_H								
Bit	7	6	Bit	7	6	Bit	7	6
Field	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	16 bits 2's complement number. Upper 8 bits of Delta_Y							
Usage	Delta_Y_H must be read after Delta_Y_L to have the full motion data. Reading it clears the register							

Register: 0x07								
Name: SQUAL								
Bit	7	6	5	4	3	2	1	0
Field	SQ_7	SQ_6	SQ_5	SQ_4	SQ_3	SQ_2	SQ_1	SQ_0
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	The SQUAL (Surface quality) register is a measure of the number of valid features visible by the chip in the current frame. Use the following formula to find the total number of valid features. Number of Features = SQUAL Register Value * 8 The maximum SQUAL register value is 0x80. Since small changes in the current frame can result in changes in SQUAL, variations in SQUAL when looking at a surface are expected. The graph below shows 883 sequentially acquired SQUAL values, while a chip was moved slowly over white paper. SQUAL values are only valid in run mode. Disable Rest mode before measuring SQUAL.							

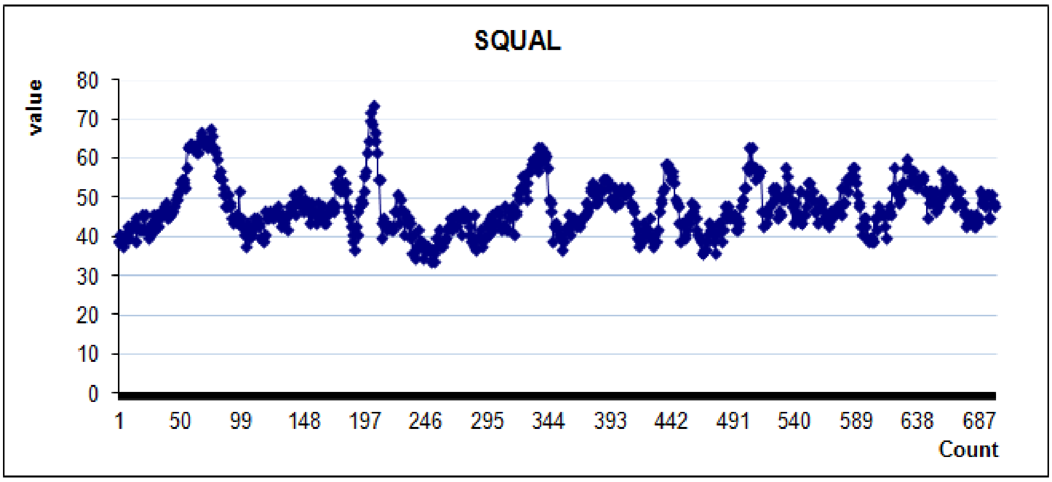


Figure 25. Average SQUAL on white paper

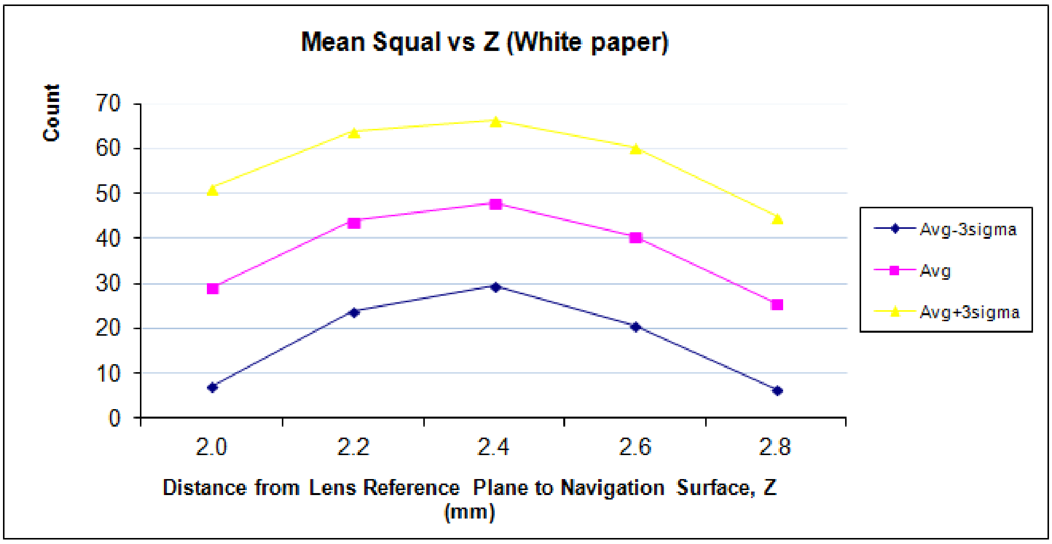


Figure 26. Mean SQUAL vs Z

Register: 0x08								
Name: Raw_Data_Sum								
Bit	7	6	5	4	3	2	1	0
Field	AP_7	AP_6	AP_5	AP_4	AP_3	AP_2	AP_1	AP_0
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	This register is used to find the average raw data value. It reports the upper byte of an 18-bit counter which sums all 1296 raw data in the current frame. To find the average raw data values follow the formula below. Average Raw Data = Register Value * 1024 / 1296 The maximum register value is 160(Dec) (0xA0) (127 * 1296 / 1024 truncated to an integer). The minimum register value is 0. The raw data sum value can change every frame							

Register: 0x09								
Name: Maximum_Raw_Data								
Bit	7	6	5	4	3	2	1	0
Field	MRD_7	MRD_6	MRD_5	MRD_4	MRD_3	MRD_2	MRD_1	MRD_0
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	Maximum Raw data value in current frame. Minimum value = 0, maximum value = 127. The maximum raw data value can change every frame							

Register: 0x0A								
Name: Minimum_Raw_Data								
Bit	7	6	5	4	3	2	1	0
Field	$MinRD_7$	$MinRD_6$	$MinRD_5$	$MinRD_4$	$MinRD_3$	$MinRD_2$	$MinRD_1$	$MinRD_0$
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	Minimum Raw data value in current frame. Minimum value = 0, maximum value = 127. The minimum raw data value can change every frame							

Register: 0x0B								
Name: Shutter_Lower								
Bit	7	6	5	4	3	2	1	0
Field	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
	Reset Value: 0x12							
Access: R/W	Read Only							
Data Type:	16-bit unsigned number							
Usage	Lower byte of the 16bit Shutter register							

Register: 0x0C								
Name: Shutter_Upper								
Bit	7	6	5	4	3	2	1	0
Field	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{10}	S_9	S_8
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	16-bit unsigned number							
Usage	Units are clock cycles of the internal oscillator. Read Shutter_Upper first, then Shutter_Lower. They should be read consecutively. The shutter is adjusted to keep the average raw data values within normal operating ranges. The shutter value is checked and automatically adjusted to a new value if needed on every frame when operating in default mode.							

Register: 0x0D								
Name: Control								
Bit	7	6	5	4	3	2	1	0
Field	$CTRL1_7$	$CTRL1_6$	$CTRL1_5$	Reserved	Reserved	Reserved	Reserved	Reserved
	Reset Value: 0x02							
Access: R/W	Read Write							
Data Type:	8-bit unsigned integer							
Usage	This register defines programmable invert able of XY register scheme.							

Field Name	Description
$CTRL1_{[7:5]}$	000 - 0 degree 110 - 90 degree 011 – 180 degree 101 – 270 degree
$Reserved_{[4:0]}$	Reserved

Note: For $CTRL1_{[7:5]}$ please use 0 degree for best performance

Register: 0x0F								
Name: Config1								
Bit	7	6	5	4	3	2	1	0
Field	RES_7	RES 1_6	RES_5	RES_4	RES_3	RES_2	RES_1	RES_0
	Reset Value: 0x31							
Access: R/W	Read/ Write							
Data Type:	Bit Field							
Usage	This register allows the user to change the X & Y or Y only resolution of the chip. Shown below are the bits, their default values, and optional values. The CPI of X & Y or Y only setting in this register depends on the Rpt_Mod register bit (refer to the description for Config2 register).							

Field Name	Description
RES[7:0]	Set resolution with CPI step of 100 cpi 0x00: 100 cpi (Minimum cpi) 0x01: 200 cpi 0x02: 300 cpi : : 0x31: 5000 cpi (default cpi) : : 0x77: 12000 cpi (maximum cpi)

Register: 0x10								
Name: Config2								
Bit	7	6	5	4	3	2	1	0
Field	Reserved	Reserved	Rest_En	Reserved	Reserved	Rpt_Mod	Reserved	O
	Reset Value: 0x20							
Access: R/W	Read/ Write							
Data Type:	Bit Field							
Usage								

Field Name	Description
[7:6]	Reserved
Rest_En	0 = Normal operation without REST mode. 1 = REST mode enable.
[4:3]	Reserved
Rpt_Mod	Select the X and Y CPI reporting mode. = 0: Normal CPI setting affects both delta X and Y. = 1: CPI setting for delta Y is defined by Config1 (address 0x0F). CPI setting for delta X is defined by Config5 (address 0x2F)
1	Reserved
Bit[0]	Must be set to 0

Register: 0x11								
Name: Angle_Tune								
Bit	7	6	5	4	3	2	1	0
Field	$Angle_7$	$Angle_6$	$Angle_5$	$Angle_4$	$Angle_3$	$Angle_2$	$Angle_1$	$Angle_0$
	Reset Value: 0x00							
Access: R/W	Read/ Write							
Data Type:	Bit Field							
Usage								

Field Name	Description
Angle[7:0]	0xE2 -30 degree 0xF6 -10 degree 0x00 0 degree (default) 0x0F +15 degree 0x1E +30 degree

Register: 0x12								
Name: Frame_Capture								
Bit	7	6	5	4	3	2	1	0
Field	FC_7	FC_6	FC_5	FC_4	FC_3	FC_2	FC_1	FC_0
	Reset Value: 0x12							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	Used to capture the next available complete 1 frame of raw data values to be stored to RAM. Writing to this register will cause any firmware loaded to be overwritten and stops navigation. A hardware reset and SROM download are required to restore normal operation for motion reading. Refer to the Frame Capture section for use details.							

Register: 0x13								
Name: SROM_Enable								
Bit	7	6	5	4	3	2	1	0
Field	SE_7	SE_6	SE_5	SE_4	SE_3	SE_2	SE_1	SE_0
	Reset Value: N/A							
Access: R/W	Write Only							
Data Type:	8-bit unsigned integer							
Usage	Write to this register to start either SROM download or SROM CRC test. See SROM Download section for details. SROM CRC test can be performed to check if SROM download was successful. Navigation is halted and the SPI port should not be used during this SROM CRC test. Registers will be reset to default value after completion of CRC test. SROM CRC read procedure is as below: 1. Write 0x15 to SROM_Enable register. 2. Wait for at least 10ms. 3. Read register Data_Out_Upper and register Data_Out_Lower .							

Register: 0x14								
Name: Run_Downshift								
Bit	7	6	5	4	3	2	1	0
Field	RD_7	RD_6	RD_5	RD_4	RD_3	RD_2	RD_1	RD_0
	Reset Value: 0x32							
Access: R/W	Read/ Write							
Data Type:	8-bit unsigned integer							
Usage	This register set the Run to Rest1 downshift time. Default value is 500ms. Use the formula below for calculation. The minimum register value is 0x01. A value of 0x00 will be internally clipped to 0x01. Run Downshift time (ms) = RD[7:0] x 10 ms Default = 50 x 10 = 500ms Max = 255x10 = 2550ms = 2.55s All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x15								
Name: Res1_Rate_Lower								
Bit	7	6	5	4	3	2	1	0
Field	$R1R_7$	$R1R_6$	$R1R_5$	$R1R_4$	$R1R_3$	$R1R_2$	$R1R_1$	$R1R_0$
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	16-bit unsigned integer							
Usage	Lower byte of the Rest1 frame rate register.							

Register: 0x16								
Name: Rest1_Rate_Upper								
Bit	7	6	5	4	3	2	1	0
Field	$R1R_{15}$	$R1R_{14}$	$R1R_{13}$	$R1R_{12}$	$R1R_{11}$	$R1R_{10}$	$R1R_9$	$R1R_8$
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	16-bit unsigned integer							
Usage	Upper byte of the Rest1 frame rate register. This register sets the Rest1 frame rate duration. Default value is 1 ms. To write to the registers, write Lower first, followed by Upper. Register read can be in any order but must be consecutive. R1R[15:0] value must not exceed 0x09B0, otherwise an internal watchdog will trigger a reset. Use the formula below for calculation. Rest1 frame rate duration = (R1R[15:0] + 1) x 1 ms Default = (0 + 1) x 1 = 1 ms All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x17								
Name: Rest1_Downshift								
Bit	7	6	5	4	3	2	1	0
Field	$R1D_7$	$R1D_6$	$R1D_5$	$R1D_4$	$R1D_3$	$R1D_2$	$R1D_1$	$R1D_0$
	Reset Value: 0x1F							
Access: R/W	Read/Write							
Data Type:	8-bit unsigned integer							
Usage	This register set the Rest1 to Rest2 downshift time. Default value is 9.92 sec. Use the formula below for calculation. The minimum register value is 0x01. A value of 0x00 will be internally clipped to 0x01. The default multiplier value is defined through SROM. Rest1 Downshift time = R1D[7:0] x 320 x Rest1_Rate. Default = Rest1_Downshift x 320 x Rest1_Rate = 9.92s (default multiplier value is 320) All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x18								
Name: Rest2_Rate_Lower								
Bit	7	6	5	4	3	2	1	0
Field	$R2R_7$	$R2R_6$	$R2R_5$	$R2R_4$	$R2R_3$	$R2R_2$	$R2R_1$	$R2R_0$
	Reset Value: 0x63							
Access: R/W	Read/Write							
Data Type:	16-bit unsigned integer							
Usage	Lower byte of the Rest2 frame rate register.							

Register: 0x19								
Name: Rest2_Rate_Upper								
Bit	7	6	5	4	3	2	1	0
Field	$R2R_{15}$	$R2R_{14}$	$R2R_{13}$	$R2R_{12}$	$R2R_{11}$	$R2R_{10}$	$R2R_9$	$R2R_8$
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	16-bit unsigned integer							
Usage	Upper byte of the Rest2 frame rate register. This register sets the Rest2 frame rate duration. Default value is 100 ms. To write to the registers, write Lower first, followed by Upper. Register read can be in any order but must be consecutive. R2R[15:0] value must not exceed 0x09B0, otherwise an internal watchdog will trigger a reset. Use the formula below for calculation. Rest2 frame rate duration = (R2R[15:0] + 1) x 1 ms Default = (99 + 1) x 1 = 100 ms All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x1A								
Name: Rest2_Downshift								
Bit	7	6	5	4	3	2	1	0
Field	$R2D_7$	$R2D_6$	$R2D_5$	$R2D_4$	$R2D_3$	$R2D_2$	$R2D_1$	$R2D_0$
	Reset Value: 0xBC							
Access: R/W	Read/Write							
Data Type:	8-bit unsigned integer							
Usage	This register set the Rest2 to Rest3 downshift time. Default value is 601.6s. Use the formula below for calculation. The minimum register value is 0x01. A value of 0x00 will be internally clipped to 0x01. Rest2 Downshift time = R2D[7:0] x 32 x Rest2_Rate. Default = 188 x 32 x 100 = 601.6s = 10mins All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x1B								
Name: Rest3_Rate_Lower								
Bit	7	6	5	4	3	2	1	0
Field	$R3R_7$	$R3R_6$	$R3R_5$	$R3R_4$	$R3R_3$	$R3R_2$	$R3R_1$	$R3R_0$
	Reset Value: 0xF3							
Access: R/W	Read/Write							
Data Type:	16-bit unsigned integer							
Usage	Lower byte of the Rest3 frame rate register.							

Register: 0x1C								
Name: Res3_Rate_Upper								
Bit	7	6	5	4	3	2	1	0
Field	$R3R_{15}$	$R3R_{14}$	$R3R_{13}$	$R3R_{12}$	$R3R_{11}$	$R3R_{10}$	$R3R_9$	$R3R_8$
	Reset Value: 0x01							
Access: R/W	Read/Write							
Data Type:	16-bit unsigned integer							
Usage	Upper byte of the Rest3 frame rate register. This register sets the Rest3 frame rate duration. Default value is 500 ms. To write to the registers, write Lower first, followed by Upper. Register read can be in any order but must be consecutive. R3R[15:0] value must not exceed 0x09B0, otherwise an internal watchdog will trigger a reset. Use the formula below for calculation. Rest3 frame rate duration = (R3R[15:0] + 1) x 1 ms Default = (499 + 1) x 1 = 500 ms All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x24								
Name: Observation								
Bit	7	6	5	4	3	2	1	0
Field	Reserved	OB_6	OB_5	OB_4	OB_3	OB_2	OB_1	OB_0
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	The user must clear the register by writing 0x00, wait for minimum T_{dly_obs} msec, and read the register. The active process will have set their corresponding bit. The register may be used as part of recovery scheme to detect a problem caused by EFT/B or ESD. T_{dly_obs} is defined as the longest frame period + 0.5msec. The longest frame period is Rest3. Clock tolerance need to be taken into account. For e.g. if the default Rest3 rate of 500msec is used, then T_{dly_obs} = (500x1.4) + 0.5 = 700.5msec.							

Field Name	Description
OB6	SROM_RUN: Indicates whether SROM is running. 0 = SROM not running 1 = SROM running
OB[5:0]	Set once per frame

Register: 0x25								
Name: Data_Out_Lower								
Bit	7	6	5	4	3	2	1	0
Field	DO_7	DO_6	DO_5	DO_4	DO_3	DO_2	DO_1	DO_0
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	16-bit unsigned integer							
Usage	Lower byte of the Data_Out register							

Register: 0x26								
Name: Data_Out_Upper								
Bit	7	6	5	4	3	2	1	0
Field	DO_{15}	DO_{14}	DO_{13}	DO_{12}	DO_{11}	DO_{10}	DO_9	DO_8
	Reset Value: 0x00							
Access: R/W	Read Only							
Data Type:	16-bit unsigned integer							
Usage	Data in these registers come from the SROM CRC test. The data can be read out in any order. The SROM CRC test is initiated by writing 0x15 to SROM_Enable register.							

CRC Result	Data_Out_Upper	Data_Out_Lower
SROM CRC test	0xBE	0xEF

Register: 0x29								
Name: Raw_Data_Grab								
Bit	7	6	5	4	3	2	1	0
Field	Valid	RD_D_6	RD_D_5	RD_D_4	RD_D_3	RD_D_2	RD_D_1	RD_D_0
	Reset Value: 0x00							
Access: R/W	Read / Write							
Data Type:	8-bit unsigned integer							
Usage	Write any value to this register to initialize the raw data output. Read motion register to check if first raw data is ready, and then read data from this register for the raw data. 1. Write 0 to Bit [5] of register 0x10 (Config2) to disable Rest mode. 2. Write any value to Raw_Data_Grab register to reset the register. 3. Read MOTION register 0x02 & check for Bit [4] for first raw data in raw data grab to be ready. 4. Then continuously reading Raw_Data_Grab register for raw data for 1296 times. Ensure Bit [7] is valid for each raw data read. 5. Write 1 to Bit [5] of register 0x10 (Config2) to enable rest mode if required.							

Register: 0x2A								
Name: SROM_ID								
Bit	7	6	5	4	3	2	1	0
Field	SR_7	SR_6	SR_5	SR_4	SR_3	SR_2	SR_1	SR_0
	0x00							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	Contains the revision of the downloaded Shadow ROM (SROM) firmware. If the firmware has been successfully downloaded and the chip is operating out of SROM, this register will contain the SROM firmware revision; otherwise it will contain 0x00.							

Register: 0x2B								
Name: Min_SQ_Run								
Bit	7	6	5	4	3	2	1	0
Field	$MSQR_7$	$MSQR_6$	$MSQR_5$	$MSQR_4$	$MSQR_3$	$MSQR_2$	$MSQR_1$	$MSQR_0$
	Reset Value: 0x10							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	This register defines the minimum Squal threshold below which the chip will produce motion delta values of zero. Typically, the default value of this register should only be modified as a result of Lift cut off calibration SROM. Min_SQ_Run is only available for Tracking SROM and above.							

Register: 0x2C								
Name: Raw_Data_Threshold								
Bit	7	6	5	4	3	2	1	0
Field	$RDTH_7$	$RDTH_6$	$RDTH_5$	$RDTH_4$	$RDTH_3$	$RDTH_2$	$RDTH_1$	$RDTH_0$
	Reset Value: 0x0A							
Access: R/W	Read/ Write							
Data Type:	Bit Field							
Usage	This register affects the SQUAL value register value. The SQUAL is a measure of the number of valid features. The raw data threshold register defines what is considered a valid feature. A low threshold value will make it easier for a feature to be considered valid. Therefore, a low raw data threshold will increase SQUAL since more features will be considered valid and vice versa. If raw data threshold is set too high, it will invalidate features that are actually trackable, thus making SQUAL too low and degrades tracking. If raw data threshold is set too low, it will validate features that are not trackable. Typically, the default value of this register should only be modified as the result of Lift cut off calibration SROM. Raw_Data_Threshold is only available with tracking SROM.							

Register: 0x2F								
Name: Config5								
Bit	7	6	5	4	3	2	1	0
Field	$RESX_7$	$RESX_6$	$RESX_5$	$RESX_4$	$RESX_3$	$RESX_2$	$RESX_1$	$RESX_0$
	Reset Value: 0x31							
Access: R/W	Read/ Write							
Data Type:	Bit Field							
Usage	This register allows the user to change the X-axis resolution when the chip is configured to have independent X-axis and Y-axis resolution reporting mode via Rpt_Mod bit = 1 in Config2 register. The setting in this register will be inactive if Rpt_Mod bit = 0.Shown below are the bits, their default values, and optional values.							

Field Name	Description
RESX [7:0]	Set resolution with CPI step of 100 cpi 0x00: 100 cpi (Minimum cpi) 0x01: 200 cpi 0x02: 300 cpi : : 0x31: 5000 cpi (default cpi) : : 0x77: 12000 cpi (maximum cpi)

Register: 0x3A								
Name: Power_Up_Reset								
Bit	7	6	5	4	3	2	1	0
Field	PUR_7	PUR_6	PUR_5	PUR_4	PUR_3	PUR_2	PUR_1	PUR_0
	Reset Value: N/A							
Access: R/W	Write Only							
Data Type:	8-bit unsigned integer							
Usage	Write 0x5a to this register to reset the chip. All settings will revert to default values. Reset is required after recovering from shutdown mode and to restore normal operation after Frame Capture							

Register: 0x3B								
Name: Shutdown								
Bit	7	6	5	4	3	2	1	0
Field	SD_7	SD_6	SD_5	SD_4	SD_3	SD_2	SD_1	SD_0
	Reset Value: N/A							
Access: R/W	Write Only							
Data Type:	8-bit unsigned integer							
Usage	Write 0xB6 to set the chip to shutdown mode. Refer to the Shutdown section for more details and on the recovery procedure.							

Register: 0x3F								
Name: Inverse_Product_ID								
Bit	7	6	5	4	3	2	1	0
Field	PID_7	PID_6	PID_5	PID_4	PID_3	PID_2	PID_1	PID_0
	Reset Value: 0xBD							
Access: R/W	Read Only							
Data Type:	Bit Field							
Usage	This value is the inverse of the Product_ID. It is used to test the SPI port hardware							

Register: 0x41								
Name: LiftCuttoff_Tune3								
Bit	7	6	5	4	3	2	1	0
Field	$RMSQ_7$	$RMSQ_6$	$RMSQ_5$	$RMSQ_4$	$RMSQ_3$	$RMSQ_3$	$RMSQ_1$	$RMSQ_0$
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	This register is valid only if the calibration procedure is stopped successfully. RMSQ[7:0] recommends a minimum Squal run value that replaces the default value in the Min_SQ_Run register to improve lift performance. LiftCutoff_Tune3 is only available if Lift cut off calibration SROM is used for Lift cut off calibration.							

Register: 0x42								
Name: Angle_Snap								
Bit	7	6	5	4	3	2	1	0
Field	AS_EN	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	The AS_EN bit in this register enables or disables the Angle Snap feature. AS_EN = 0 (Angle snap disabled. This is the default value.) AS_EN = 1 (Angle snap enabled with 5° snap setting.)							

Register: 0x4A								
Name: LiftCuttoff_Tune1								
Bit	7	6	5	4	3	2	1	0
Field	RUN_CAL	Reserved	Reserved	Reserved	Reserved	CAL_STAT2	CAL_STAT1	CAL_STAT0
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	This register is used to start either the Shutter Calibration or the SQUAL Calibration Lift cut off calibration procedure. It is also used to check the status of either procedure. Refer to the Lift cut off calibration section for more details.							

Field Name	Description
RUN_CAL	0 = Stop Shutter Calibration procedure (default) 1 = Start Shutter Calibration procedure
Bit [6:3]	Reserved
CAL_STAT[2:0]	0x00 = Reserved 0x01 = Calibration in progress. 0x02 = Calibration successfully completed (minimum length met). Surface data collection continues until timeout. Registers LiftCutoff_Tune_Min_Length and LiftCutoff_Tune_Timeout define the minimum length threshold and timeout respectively. 0x03 = Calibration successfully completed (minimum length met) and timeout has triggered. Surface data collection stops automatically. 0x04 = Calibration unsuccessful (minimum length not met) and timeout has triggered. 0x05 - 0x07 = Reserved

Register: 0x50								
Name: Motion_Burst								
Bit	7	6	5	4	3	2	1	0
Field	MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0
	Reset Value: 0x00							
Access: R/W	Read/Write							
Data Type:	8-Bit unsigned integer							
Usage	The Motion_Burst register is used for high-speed access of up to 12 register bytes. See the Burst Mode-Motion Read section for full details of operation.							

Register: 0x58								
Name: LiftCuttoff_Tune_Timeout								
Bit	7	6	5	4	3	2	1	0
Field	$RMSQ_7$	$RMSQ_6$	$RMSQ_5$	$RMSQ_4$	$RMSQ_3$	$RMSQ_3$	$RMSQ_1$	$RMSQ_0$
	Reset Value: 0x27							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	This register sets the minimum Lift cut off calibration timeout threshold. Timeout (sec) = (TIMEOUT[7:0] + 1) x 0.5 sec Default = (39 + 1) x 0.5 = 20 sec Allowed TIMEOUT[7:0] range is 0x00 (0.5 sec) to 0xF9 (125 sec). All the above values are expected to have a +40% and -20% of tolerance.							

Register: 0x5A								
Name: LiftCuttoff_Tune_Min_Length								
Bit	7	6	5	4	3	2	1	0
Field	$MINL_7$	$MINL_6$	$MINL_5$	$MINL_4$	$MINL_3$	$MINL_3$	$MINL_1$	$MINL_0$
	Reset Value: 0x09							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	This register sets the minimum Lift cut off calibration length threshold. Minimum Length (inches) = (MINL[7:0] + 1) x 2 inches Default = (9 + 1) x 2 = 20 inches Allowed MINL [7:0] range is 0x00 (2 inches) to 0xF9 (500 inches). Actual distance is expected to have a tolerance that is strongly dependent on MINL. The tolerance is approximately 40% for MINL = 0x04 (10 inches) and above. It is not recommended to set a MINL that is lower because the tolerance can potentially increase to 100%.							

Register: 0x62								
Name: SROM_Load_Burst								
Bit	7	6	5	4	3	2	1	0
Field	SL_7	SL_6	SL_5	SL_4	SL_3	SL_2	SL_1	SL_0
	Reset Value: N/A							
Access: R/W	Write Only							
Data Type:	8-Bit unsigned integer							
Usage	The SROM_Load_Burst register is used for high-speed programming SROM from an external PROM or microcontroller. See the SROM Download section for use details.							

Register: 0x63								
Name: Lift_Config								
Bit	7	6	5	4	3	2	1	0
Field	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	LIFC1	LIFC0
	Reset Value: 0X02							
Access: R/W	Read/Write							
Data Type:	Bit Field							
Usage	This register defines the lift detection height threshold. The lift status bit is asserted when the chip is above the threshold.							

LIFC[1:0]	Description
00	Reserved
10	Lift detection height = nominal height + 2 mm (default value).
11	Lift detection height = nominal height + 3 mm.

Register: 0x64								
Name: Raw_Data_Burst								
Bit	7	6	5	4	3	2	1	0
Field	RDB_7	RDB_6	RDB_5	RDB_4	RDB_3	RDB_2	RDB_1	RDB_0
	Reset Value: 0X00							
Access: R/W	Read Only							
Data Type:	8-Bit unsigned integer							
Usage	The Raw_Data_Burst register is used for high-speed access to all the raw data values for one complete frame capture, without having to write to the register address to obtain each raw data. The data pointer is automatically incremented after each read so all 1296 raw data values may be obtained by reading this register 1296 times. See the Frame Capture section for details. **Note**: Maximum raw data value is 127. PB7 is always zero.							

<table>
<tr><th colspan="9">Register: 0x65</th></tr>
<tr><td colspan="9">Name: LiftCuttoff_Tune2</td></tr>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">Field</td><td>Reserved</td><td>$RPTH_6$</td><td>$RPTH_5$</td><td>$RPTH_4$</td><td>$RPTH_3$</td><td>$RPTH_3$</td><td>$RPTH_1$</td><td>$RPTH_0$</td></tr>
<tr><td colspan="8">Reset Value:0x00</td></tr>
<tr><td>Access: R/W</td><td colspan="8">Read Only</td></tr>
<tr><td>Data Type:</td><td colspan="8">Bit Field</td></tr>
<tr><td>Usage</td><td colspan="8">This register provides Lift cut off calibration related readout registers. See the Lift cut off calibration section for more details.

<table><tr><th>Field Name</th><th>Description</th></tr><tr><td>RPTH[6:0]</td><td>These bits are valid only if calibration procedure is stopped successfully. RPTH[6:0] recommends a raw data threshold value that replaces the default value in the Raw_Data_Threshold register to improve lift performance.</td></tr></table></td></tr>
</table>

12.0 Document Revision History

Revision Number	Date	Description
1.00	19 Aug 2014	- Initial creation
1.10	26 Nov 2015	- pg8 update Fig6 Lens Outline Drawing - pg10 update Fig8 Recommended Base Plate Opening - pg28 add item #3 Delay for 30mis
1.20	25 Feb 2016	- pg23 add point #8 Write 0x00 to Config2 register for wired mouse or 0x20 for wireless mouse design
1.30	6 Apr 2016	- pg47 add Register 0x29 Pix_Grab information
1.40	3 Aug 2016	- pg55 modify Register 0x63 Lift_Config register information. Removed setting 0x00
1.50	26 Sep 2016	- Update document. Change “sensor” to “chip”& “pixel” to “raw data” - Change PixArt RoH Logo - Change Image Array to Picture Element Array