Biorobotics laboratory

MOUSE TREADMILL CONTROL

Saturday 14th December, 2019

By **Didier Chérubin Negretto**

Professor: **Auke Ijspeert**

Assistant 1:**Shravan Tata Ramalingasetty**

Semester project description

Objectives and preliminary considerations The project consists of designing and manufacturing of a lightweight mechanism to improve roll authority at low angles of attack. After that the mechanism is tested and conclusions, with respect to the existing mechanism are drawn. A drone with morphing wings was designed and manufactured in the months before this project. The drone has wings with artificial overlapping feathers at the wing tip, which are used for roll control. The roll rate obtained at low angles of attack is too low to grant the desired high manoeuvrability as shown by [9] for a similar roll control strategy. In the first part of the project different methods are taken into account: ailerons, which are used in the aircraft industry and wing twisting, which is inspired by birds [**?**]. Those methods are simulated on XFLR5 (using VLM and 3D-Panels), after which wing twisting is chosen for implementation and testing. Wing twisting has many advantages compared to ailerons: higher roll moment [5] and higher lift to drag ratio [6]. On the other hand this technique is less used in industry and leads to an increased weight of the drone.

Mechanism design A weight of less than 20 $[g]$ is required as well as the interoperability with the pre-existing folding mechanism. Four different designs are taken into account (partial twisting, flexible components, cylindrical element and ball joints). In the end the design consisting of a lever actuated by a servo (ball joints) is chosen for implementation. This design is simple and can generate the required roll (roll control power over 2 $[\frac{\partial C_r}{\partial rad}]$ for $\alpha \leq 8°$) at low angles of attack, solving the initial problem. The folding angle ϕ is defined as the sum of the angles compared to normal sweep on the two sides.

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.

Testing and results The third part consists of testing the drone in the wind tunnel at different angles of attack (from 0° to 28° with steps of 4°), at three different twisting angles (0°, 5° and 10°), and with three possible wing shapes to take into account the effect of folding. Most of roll is generated with twisting at low angles of attack ($\alpha < 8°$), while at high angles folding has a greater impact (see figure 0.1).

Roll control and cost Finally a roll control algorithm for folding and twisting wing drones is presented. This is one of the most important accomplishments of this report since a study of different roll control methods on the same platform is, at the best of our knowledge, absent in literature. The cost of such a setup is discussed as well. This cost is very low, mainly due to the weight of the mechanism and to the energy consumption since the loss of aerodynamic performances during roll is small (C_d +38,98%), compared to the roll achieved ($C_r = 0.4519$ [ ]).

Contents

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Requirements . . . 3
1.3 Structure of the report . . . 4

2 System architecture and communication **4**
2.1 System architecture . . . 4
2.2 Communication . . . 5

3 Design choices **5**
3.1 Board . . . 5
3.2 Sensor . . . 5
3.3 Motor . . . 5

4 Control **5**

5 Conclusion **5**

A Code for STM32 NUCLEO 64 board **7**
A.1 Main . . . 7
A.2 Treadmill driver . . . 21
A.3 Sensor driver . . . 31
A.4 Code for unit tests . . . 36
A.5 Build script . . . 46

B Code for PC **46**
B.1 GUI . . . 46
B.2 Routine example . . . 53
B.3 Sensor data-sheet . . . 53

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

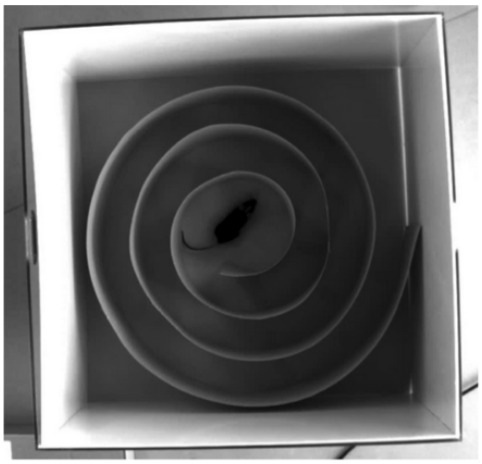


Figure 1.2 – The experimental setup used in [1].

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.

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 4 describes the control strategy and shows some preliminary responses. Finally in section 5 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 and communication

In this section the architecture and the communication are explained and detailed.

2.1 System architecture

In this section the architecture and interfaces between the different components of the machine are detailed.

2.2 Communication

In this section all the messages as well as their meaning and usage are explained.

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

3.2 Sensor

3.3 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.

4 Control

5 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] 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

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

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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

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

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

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

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

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

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

[16] 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

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. 1
1.2 The experimental setup used in [1]. 3

List of Tables

1 Summary of the requirements for the mouse treadmill platform. . . . 4

A Code for STM32 NUCLEO 64 board

A.1 Main

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

```
21
22 /* Define to prevent recursive inclusion
   -------------------------------------*/
23 #ifndef __MAIN_H
24 #define __MAIN_H
25
26 #ifdef __cplusplus
27 extern "C" {
28 #endif
29
30 /* Includes
   ------------------------------------------------------------------------------
   */
31 #include "stm32l4xx_hal.h"
32
33 /* Private includes
   ----------------------------------------------------------*/
34 /* USER CODE BEGIN Includes */
35 #include "mouseDriver.h"
36 #include "mavlink.h"
37 /* USER CODE END Includes */
38
39 /* Exported types
   ------------------------------------------------------------*/
40 /* USER CODE BEGIN ET */
41 typedef struct SENSOR{
42   GPIO_TypeDef * cs_port;
43   uint8_t cs_pin;
44   GPIO_TypeDef * pw_port;
45   uint8_t pw_pin;
46   uint8_t status;
47 } sensor_t;
48 /* USER CODE END ET */
49
50 /* Exported constants
   --------------------------------------------------------*/
51 /* USER CODE BEGIN EC */
52
53 /* USER CODE END EC */
54
55 /* Exported macro
   ------------------------------------------------------------*/
56 /* USER CODE BEGIN EM */
57
58 /* USER CODE END EM */
59
60 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
61
62 /* Exported functions prototypes
   ---------------------------------------------*/
63 void Error_Handler(void);
64
65 /* USER CODE BEGIN EFP */
66 void main_transmit_buffer(uint8_t *outBuffer, uint16_t msg_size);
67 void main_stop_motors(void);
68 void main_set_motors_speed(mavlink_motor_setpoint_t motor );
69 int main_get_huart_tx_state(void);
70 void main_write_sensor (sensor_t sensor, uint8_t adress, uint8_t data);
```

```
71 uint8_t main_read_sensor (sensor_t sensor, uint8_t adress );
72 void main_transmit_spi(uint8_t data);
73 void main_wait_160us(void);
74 void main_wait_20us(void);
75 void main_write_sensor_burst(uint8_t data);
76 void main_read_sensor_motion_burst(uint8_t *data );
77 /*
78  * PW_0 is power pin for sensor X (PB_0)
79  * PW_1 is the power pin for sensor Y (PA_4)
80  * CS_0 is the chip select for sensor X (PC_0)
81  * CS_1 is the chip select for sensor Y (PC_1)
82  */
83
84 /* USER CODE END EFP */
85
86 /* Private defines
    -----------------------------------------------------------------------*/
87 #define DT_HEART 200
88 #define PRESCALER_HEART 1000
89 #define CLOCK_FREQ 80000000
90 #define COUNTER_PERIOD_HEART ((CLOCK_FREQ/(PRESCALER_HEART))*0.001*
    DT_HEART)
91 #define PRESCALER_PWM 9
92 #define COUNTER_PERIOD_PWM 255
93 #define PULSE_PWM 10
94 #define B1_Pin GPIO_PIN_13
95 #define B1_GPIO_Port GPIOC
96 #define CS_0_Pin GPIO_PIN_0
97 #define CS_0_GPIO_Port GPIOC
98 #define CS_1_Pin GPIO_PIN_1
99 #define CS_1_GPIO_Port GPIOC
100 #define USART_TX_Pin GPIO_PIN_2
101 #define USART_TX_GPIO_Port GPIOA
102 #define USART_RX_Pin GPIO_PIN_3
103 #define USART_RX_GPIO_Port GPIOA
104 #define PW_1_Pin GPIO_PIN_4
105 #define PW_1_GPIO_Port GPIOA
106 #define LD2_Pin GPIO_PIN_5
107 #define LD2_GPIO_Port GPIOA
108 #define PW_0_Pin GPIO_PIN_0
109 #define PW_0_GPIO_Port GPIOB
110 #define TMS_Pin GPIO_PIN_13
111 #define TMS_GPIO_Port GPIOA
112 #define TCK_Pin GPIO_PIN_14
113 #define TCK_GPIO_Port GPIOA
114 #define SWO_Pin GPIO_PIN_3
115 #define SWO_GPIO_Port GPIOB
116 /* USER CODE BEGIN Private defines */
117
118 /* USER CODE END Private defines */
119
120 #ifdef __cplusplus
121 }
122 #endif
123
124 #endif /* __MAIN_H */
125
126 /************************ (C) COPYRIGHT STMicroelectronics *****END OF
```

```
  FILE****/

/* 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 */
#define TIMEOUT 2
/* USER CODE END PD */

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

/* USER CODE END PM */
```

```
43
44 /* Private variables
   ---------------------------------------------------------*/
45 SPI_HandleTypeDef hspi2;
46
47 TIM_HandleTypeDef htim1;
48 TIM_HandleTypeDef htim7;
49
50 UART_HandleTypeDef huart2;
51 DMA_HandleTypeDef hdma_usart2_tx;
52
53 /* USER CODE BEGIN PV */
54 static uint8_t inByte = 0;
55 /* USER CODE END PV */
56
57 /* Private function prototypes
   -----------------------------------------------*/
58 void SystemClock_Config(void);
59 static void MX_GPIO_Init(void);
60 static void MX_USART2_UART_Init(void);
61 static void MX_TIM7_Init(void);
62 static void MX_TIM1_Init(void);
63 static void MX_DMA_Init(void);
64 static void MX_SPI2_Init(void);
65 /* USER CODE BEGIN PFP */
66 void main_wait_160us(void){
67   int i = 0;
68   i = 0;
69   while(i<900){
70     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
71     i++;
72   }
73 }
74 void main_wait_20us(void){
75   int i = 0;
76   i = 0;
77   while(i<185){
78     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
79     i++;
80   }
81 }
82 void main_wait_1us(void){
83   int i = 0;
84   i = 0;
85   while(i<25){
86     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
87     i++;
88   }
89 }
90 int main_get_huart_tx_state(void){
91   return (HAL_DMA_GetState(&hdma_usart2_tx));
92 }
93 void main_transmit_buffer(uint8_t *outBuffer, uint16_t msg_size){
94   HAL_UART_Transmit_DMA(&huart2, outBuffer,msg_size);
95 }
96 void main_stop_motors(void)
97 {
98   HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
```

```
99    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
100 }
101 void main_set_motors_speed(mavlink_motor_setpoint_t motor )
102 {
103
104   htim1.Instance->CCR1 = motor.motor_x;
105   htim1.Instance->CCR2 = motor.motor_y;
106
107   if (motor.motor_x == 0)
108     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
109   else
110     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
111
112   if (motor.motor_y == 0)
113     HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
114   else
115     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
116
117 }
118 uint8_t main_read_sensor (const sensor_t sensor, uint8_t adress ){
119   uint8_t value = 0;
120   uint8_t adress_read = adress & 0x7F;
121
122   HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_RESET);
123   HAL_SPI_Transmit(&hspi2, &adress_read, 1, 100);
124   main_wait_160us();
125   HAL_SPI_Receive(&hspi2, &value, 1, 100);
126   main_wait_1us();
127   HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_SET);
128   main_wait_20us();
129   return (value);
130 }
131
132 void main_write_sensor (const sensor_t sensor, uint8_t adress, uint8_t
      data){
133   uint8_t value = data;
134   uint8_t adress_write = adress | 0x80;
135   uint8_t pack[2];
136   pack[0] = adress_write;
137   pack[1] = value;
138
139   HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_RESET);
140   HAL_SPI_Transmit(&hspi2, pack, 2, 10);
141   main_wait_20us();
142   HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_SET);
143   main_wait_160us();
144   main_wait_20us();
145 }
146 void main_write_sensor_burst(uint8_t data){
147   HAL_SPI_Transmit(&hspi2, &data, 1, 10);
148   main_wait_20us();
149 }
150 void main_read_sensor_motion_burst(uint8_t *data ){
151   HAL_SPI_Receive(&hspi2,data,12,100);
152   main_wait_1us();
153 }
154 void main_transmit_spi(uint8_t data){
155   uint8_t data_out = data;
```

```
156   HAL_SPI_Transmit(&hspi2, &data_out, 1, 10);
157 }
158 /* USER CODE END PFP */
159
160 /* Private user code
      ---------------------------------------------------------*/
161 /* USER CODE BEGIN 0 */
162
163 void TM7_IRQHandler(void){
164   HAL_TIM_IRQHandler(&htim7);
165
166 }
167
168 /* This callback is called by the HAL_UART_IRQHandler when the given
      number of bytes are received */
169 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
170   HAL_NVIC_DisableIRQ(USART2_IRQn);
171   mavlink_message_t inmsg;
172   mavlink_status_t msgStatus;
173   if (huart->Instance == USART2){
174     /* Receive one byte in interrupt mode */
175     HAL_UART_Receive_IT(&huart2, &inByte, 1);
176     if(mavlink_parse_char(0, inByte, &inmsg, &msgStatus)){
177
178       mouseDriver_readMsg(inmsg);
179     }
180   }
181   HAL_NVIC_EnableIRQ(USART2_IRQn);
182 }
183
184 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
185     if (htim->Instance==TIM7){
186       mouseDriver_send_status_msg();
187     }
188 }
189 /* USER CODE END 0 */
190
191 /**
192   * @brief  The application entry point.
193   * @retval int
194   */
195 int main(void)
196 {
197   /* USER CODE BEGIN 1 */
198
199   /* USER CODE END 1 */
200
201
202   /* MCU Configuration
      ---------------------------------------------------------*/
203
204   /* Reset of all peripherals, Initializes the Flash interface and the
      Systick. */
205   HAL_Init();
206
207   /* USER CODE BEGIN Init */
208
209   /* USER CODE END Init */
```

```
210
211   /* Configure the system clock */
212   SystemClock_Config();
213
214   /* USER CODE BEGIN SysInit */
215
216   /* USER CODE END SysInit */
217
218   /* Initialize all configured peripherals */
219   MX_GPIO_Init();
220   MX_USART2_UART_Init();
221   MX_TIM7_Init();
222   MX_TIM1_Init();
223   MX_DMA_Init();
224   MX_SPI2_Init();
225   /* USER CODE BEGIN 2 */
226   HAL_InitTick(0);
227   HAL_NVIC_SetPriority(USART2_IRQn,1,0);
228   HAL_NVIC_EnableIRQ(USART2_IRQn);
229   HAL_NVIC_SetPriority(TIM7_IRQn,2,0);
230   HAL_NVIC_EnableIRQ(TIM7_IRQn);
231   HAL_GPIO_WritePin(GPIOC, CS_0_Pin|CS_1_Pin, GPIO_PIN_SET);
232
233   HAL_UART_Receive_IT(&huart2, &inByte, 1);
234   HAL_TIM_Base_Start_IT(&htim7);
235   HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
236
237   mouseDriver_init();
238
239   /* USER CODE END 2 */
240
241   /* Infinite loop */
242   /* USER CODE BEGIN WHILE */
243
244   while (1)
245   {
246    mouseDriver_idle();
247     /* USER CODE END WHILE */
248
249   /* USER CODE BEGIN 3 */
250   }
251   /* USER CODE END 3 */
252 }
253
254 /**
255   * @brief System Clock Configuration
256   * @retval None
257   */
258 void SystemClock_Config(void)
259 {
260   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
261   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
262   RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
263
264   /** Initializes the CPU, AHB and APB busses clocks
265   */
266   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
267   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
```

```
268   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
269   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
270   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
271   RCC_OscInitStruct.PLL.PLLM = 1;
272   RCC_OscInitStruct.PLL.PLLN = 10;
273   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
274   RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
275   RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
276   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
277   {
278     Error_Handler();
279   }
280   /** Initializes the CPU, AHB and APB busses clocks
281   */
282   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
283                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
284   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
285   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
286   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
287   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
288
289   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) !=
      HAL_OK)
290   {
291     Error_Handler();
292   }
293   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2;
294   PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
295   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
296   {
297     Error_Handler();
298   }
299   /** Configure the main internal regulator output voltage
300   */
301   if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
      HAL_OK)
302   {
303     Error_Handler();
304   }
305 }
306
307 /**
308   * @brief SPI2 Initialization Function
309   * @param None
310   * @retval None
311   */
312 static void MX_SPI2_Init(void)
313 {
314
315   /* USER CODE BEGIN SPI2_Init 0 */
316   HAL_GPIO_DeInit(GPIOC, GPIO_PIN_3);
317
318   /*GPIO_InitTypeDef pin;
319   pin.Pin = GPIO_PIN_3;
320   pin.Mode = GPIO_MODE_OUTPUT_PP;
321   pin.Pull = GPIO_PULLDOWN;
322   pin.Speed = GPIO_SPEED_MEDIUM;
323   HAL_GPIO_Init(GPIOC, &pin);
```

```
324   HAL_GPIO_WritePin(GPIOC,GPIO_PIN_3, GPIO_PIN_RESET);*/
325
326   __HAL_RCC_SPI2_CLK_ENABLE();
327   __SPI2_CLK_ENABLE();
328   /* USER CODE END SPI2_Init 0 */
329
330   /* USER CODE BEGIN SPI2_Init 1 */
331
332   /* USER CODE END SPI2_Init 1 */
333   /* SPI2 parameter configuration*/
334   hspi2.Instance = SPI2;
335   hspi2.Init.Mode = SPI_MODE_MASTER;
336   hspi2.Init.Direction = SPI_DIRECTION_2LINES;
337   hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
338   hspi2.Init.CLKPolarity = SPI_POLARITY_HIGH;
339   hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
340   hspi2.Init.NSS = SPI_NSS_SOFT;
341   hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
342   hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
343   hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
344   hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
345   hspi2.Init.CRCPolynomial = 7;
346   hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
347   hspi2.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
348   if (HAL_SPI_Init(&hspi2) != HAL_OK)
349   {
350     Error_Handler();
351   }
352   /* USER CODE BEGIN SPI2_Init 2 */
353
354
355   /* USER CODE END SPI2_Init 2 */
356
357 }
358
359 /**
360   * @brief TIM1 Initialization Function
361   * @param None
362   * @retval None
363   */
364 static void MX_TIM1_Init(void)
365 {
366
367   /* USER CODE BEGIN TIM1_Init 0 */
368
369   /* USER CODE END TIM1_Init 0 */
370
371   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
372   TIM_MasterConfigTypeDef sMasterConfig = {0};
373   TIM_OC_InitTypeDef sConfigOC = {0};
374   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
375
376   /* USER CODE BEGIN TIM1_Init 1 */
377
378   /* USER CODE END TIM1_Init 1 */
379   htim1.Instance = TIM1;
380   htim1.Init.Prescaler = PRESCALER_PWM;
381   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
```

```
382   htim1.Init.Period = COUNTER_PERIOD_PWM;
383   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
384   htim1.Init.RepetitionCounter = 0;
385   htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
386   if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
387   {
388     Error_Handler();
389   }
390   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
391   if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
392   {
393     Error_Handler();
394   }
395   if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
396   {
397     Error_Handler();
398   }
399   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
400   sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
401   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
402   if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
       HAL_OK)
403   {
404     Error_Handler();
405   }
406   sConfigOC.OCMode = TIM_OCMODE_PWM1;
407   sConfigOC.Pulse = PULSE_PWM;
408   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
409   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
410   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
411   sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
412   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
413   if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) !=
       HAL_OK)
414   {
415     Error_Handler();
416   }
417   if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) !=
       HAL_OK)
418   {
419     Error_Handler();
420   }
421   sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
422   sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
423   sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
424   sBreakDeadTimeConfig.DeadTime = 0;
425   sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
426   sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
427   sBreakDeadTimeConfig.BreakFilter = 0;
428   sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
429   sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
430   sBreakDeadTimeConfig.Break2Filter = 0;
431   sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
432   if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) !=
       HAL_OK)
433   {
434     Error_Handler();
435   }
```

```
436   /* USER CODE BEGIN TIM1_Init 2 */
437
438   /* USER CODE END TIM1_Init 2 */
439   HAL_TIM_MspPostInit(&htim1);
440
441 }
442
443 /**
444   * @brief TIM7 Initialization Function
445   * @param None
446   * @retval None
447   */
448 static void MX_TIM7_Init(void)
449 {
450
451   /* USER CODE BEGIN TIM7_Init 0 */
452
453   /* USER CODE END TIM7_Init 0 */
454
455   TIM_MasterConfigTypeDef sMasterConfig = {0};
456
457   /* USER CODE BEGIN TIM7_Init 1 */
458
459   /* USER CODE END TIM7_Init 1 */
460   htim7.Instance = TIM7;
461   htim7.Init.Prescaler = PRESCALER_HEART;
462   htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
463   htim7.Init.Period = COUNTER_PERIOD_HEART;
464   htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
465   if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
466   {
467     Error_Handler();
468   }
469   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
470   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
471   if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) !=
      HAL_OK)
472   {
473     Error_Handler();
474   }
475   /* USER CODE BEGIN TIM7_Init 2 */
476
477   /* USER CODE END TIM7_Init 2 */
478
479 }
480
481 /**
482   * @brief USART2 Initialization Function
483   * @param None
484   * @retval None
485   */
486 static void MX_USART2_UART_Init(void)
487 {
488
489   /* USER CODE BEGIN USART2_Init 0 */
490   /* DMA controller clock enable */
491   __DMA1_CLK_ENABLE();
492
```

```
493   /* Peripheral DMA init*/
494   hdma_usart2_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
495   hdma_usart2_tx.Init.PeriphInc = DMA_PINC_DISABLE;
496   hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
497   hdma_usart2_tx.Init.PeriphDataAlignment = DMA_MDATAALIGN_BYTE;
498   hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
499   hdma_usart2_tx.Init.Mode = DMA_NORMAL;
500   hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
501   HAL_DMA_Init(&hdma_usart2_tx);
502
503   __HAL_LINKDMA(&huart2,hdmatx,hdma_usart2_tx);
504   /* USER CODE END USART2_Init 0 */
505
506   /* USER CODE BEGIN USART2_Init 1 */
507
508   /* USER CODE END USART2_Init 1 */
509   huart2.Instance = USART2;
510   huart2.Init.BaudRate = 230400;
511   huart2.Init.WordLength = UART_WORDLENGTH_8B;
512   huart2.Init.StopBits = UART_STOPBITS_1;
513   huart2.Init.Parity = UART_PARITY_NONE;
514   huart2.Init.Mode = UART_MODE_TX_RX;
515   huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
516   huart2.Init.OverSampling = UART_OVERSAMPLING_16;
517   huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
518   huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
519   if (HAL_UART_Init(&huart2) != HAL_OK)
520   {
521     Error_Handler();
522   }
523   /* USER CODE BEGIN USART2_Init 2 */
524
525   /* USER CODE END USART2_Init 2 */
526
527 }
528
529 /**
530   * Enable DMA controller clock
531   */
532 static void MX_DMA_Init(void)
533 {
534
535   /* DMA controller clock enable */
536   __HAL_RCC_DMA1_CLK_ENABLE();
537
538   /* DMA interrupt init */
539   /* DMA1_Channel7_IRQn interrupt configuration */
540   HAL_NVIC_SetPriority(DMA1_Channel7_IRQn, 0, 0);
541   HAL_NVIC_EnableIRQ(DMA1_Channel7_IRQn);
542
543 }
544
545 /**
546   * @brief GPIO Initialization Function
547   * @param None
548   * @retval None
549   */
550 static void MX_GPIO_Init(void)
```

```
551 {
552   GPIO_InitTypeDef GPIO_InitStruct = {0};
553 
554   /* GPIO Ports Clock Enable */
555   __HAL_RCC_GPIOC_CLK_ENABLE();
556   __HAL_RCC_GPIOH_CLK_ENABLE();
557   __HAL_RCC_GPIOA_CLK_ENABLE();
558   __HAL_RCC_GPIOB_CLK_ENABLE();
559 
560   /*Configure GPIO pin Output Level */
561   HAL_GPIO_WritePin(GPIOC, CS_0_Pin|CS_1_Pin, GPIO_PIN_RESET);
562 
563   /*Configure GPIO pin Output Level */
564   HAL_GPIO_WritePin(GPIOA, PW_1_Pin|LD2_Pin, GPIO_PIN_RESET);
565 
566   /*Configure GPIO pin Output Level */
567   HAL_GPIO_WritePin(PW_0_GPIO_Port, PW_0_Pin, GPIO_PIN_RESET);
568 
569   /*Configure GPIO pin : B1_Pin */
570   GPIO_InitStruct.Pin = B1_Pin;
571   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
572   GPIO_InitStruct.Pull = GPIO_NOPULL;
573   HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
574 
575   /*Configure GPIO pins : CS_0_Pin CS_1_Pin */
576   GPIO_InitStruct.Pin = CS_0_Pin|CS_1_Pin;
577   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
578   GPIO_InitStruct.Pull = GPIO_NOPULL;
579   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
580   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
581 
582   /*Configure GPIO pins : PW_1_Pin LD2_Pin */
583   GPIO_InitStruct.Pin = PW_1_Pin|LD2_Pin;
584   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
585   GPIO_InitStruct.Pull = GPIO_NOPULL;
586   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
587   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
588 
589   /*Configure GPIO pin : PW_0_Pin */
590   GPIO_InitStruct.Pin = PW_0_Pin;
591   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
592   GPIO_InitStruct.Pull = GPIO_NOPULL;
593   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
594   HAL_GPIO_Init(PW_0_GPIO_Port, &GPIO_InitStruct);
595 
596 }
597 
598 /* USER CODE BEGIN 4 */
599 
600 /* USER CODE END 4 */
601 
602 /**
603   * @brief  This function is executed in case of error occurrence.
604   * @retval None
605   */
606 void Error_Handler(void)
607 {
608   /* USER CODE BEGIN Error_Handler_Debug */
```

```
609   /* User can add his own implementation to report the HAL error return
        state */
610
611   /* USER CODE END Error_Handler_Debug */
612 }
613
614 #ifdef  USE_FULL_ASSERT
615 /**
616   * @brief  Reports the name of the source file and the source line
        number
617   *         where the assert_param error has occurred.
618   * @param  file: pointer to the source file name
619   * @param  line: assert_param error line source number
620   * @retval None
621   */
622 void assert_failed(char *file, uint32_t line)
623 {
624   /* USER CODE BEGIN 6 */
625   /* User can add his own implementation to report the file name and
        line number,
626     tex: printf("Wrong parameters value: file %s on line %d\r\n", file
        , line) */
627   /* USER CODE END 6 */
628 }
629 #endif /* USE_FULL_ASSERT */
630
631 /************************ (C) COPYRIGHT STMicroelectronics *****END OF
        FILE****/
```

A.2 Treadmill driver

```
1  /*! \file mouseDriver.c
2  \brief Implementation of the driver for the mouse treadmil project.
3
4  \author Didier Negretto
5  */
6  #ifndef MOUSEDRIVER_C_
7  #define MOUSEDRIVER_C_
8
9  #ifndef TEST
10 #include "mouseDriver.h"
11 #else
12 #include "../test/test_mouseDriver.h"
13 #endif
14 /*!
15 \def K
16 \brief Proportional coefficient for motor control.
17 */
18 #define K 10
19 /*!
20 \def K
21 \brief Proportional coefficient for motor control.
22 */
23 #define I 10
24 /*!
25 \def I
26 \brief Integral coefficient for motor control.
27 */
```

```
28 #define MAX_MOTOR_SIGNAL 100
29 /*!
30 \def MAX_MOTOR_SIGNAL
31 \brief Max value for the motor signal
32 \attention This value is used to limit the motor speed. If this is
       changed the motors might break !!
33
34 This value limits the motor speed and thus is used to vaoid spinning
       the motor too fast and break it.
35 If this value is changed the motor might spin too fast and destroy
       itself or the gear box. Extreme caution
36 needs to be taken if this value is modified.
37 */
38 #define MIN_MOTOR_SIGNAL 10
39 /*!
40 \def MIN_MOTOR_SIGNAL
41 \brief Min value for the motor signal. Any value lower than that will
       cause the motor to stop
42 */
43 #define MAX_MISSING_MEASURES 15
44 /*!
45 \def MAX_MISSING_MEASURES
46 \brief After MAX_MISSING_MEASURES non valid measures from sensors the
       motors are stopped and mode goes
47 to stop.
48 */
49 #ifndef TEST
50 /*!
51 \var actual_mode
52 \brief Global variable defining the mode of the machine
53
54 This value is updated based on the received messages. When a routine is
        running it is
55 only possible to stop the machine.
56 */
57 static uint8_t actual_mode = MOUSE_MODE_STOP;
58 /*!
59 \var actual_speed_measure
60 \brief Global variable for the measured speed
61
62 This value is updated based on sensor.
63 */
64 static mavlink_speed_info_t actual_speed_measure;
65 /*!
66 \var actual_speed_setpoint
67 \brief Global variable for the speed setpoint
68
69 This value is updated based on messages when the mode is set to SPEED.
70 */
71 static mavlink_speed_setpoint_t actual_speed_setpoint;
72 /*!
73 \var actual_motor_signal
74 \brief Global variable for the speed motor signal
75
76 This value is updated based on closed-loop control and the value
       provided in
77 \ref actual_speed_setpoint and \ref actual_speed_measure.
78 It is also possible to overwrite it by sending a
```

```
    mavlink_motor_setpoint_t message if the
79 mode is set to SPEED.
80 */
81 static mavlink_motor_setpoint_t actual_motor_signal;
82 /*!
83 \var points
84 \brief Global variable for storing the points to be followed in AUTO
    mode
85
86 The maximum amout of points is defined by \ref MAX_POINTS. This array
    is emptied after
87 every reset of the system. If not all the points are defined the
    routine is interrupted as
88 soon as a point with duration == 0 is detected.
89 */
90 static mavlink_point_t points[255];
91 /*!
92 \var actual_point
93 \brief Global variable for keeping track of the index in the \ref
    points array.
94 */
95 static uint8_t actual_point = 0;
96 /*!
97 \var actual_point_start_time
98 \brief Global variable for keeping track of the time when the last
    point in \ref points array started.
99 */
100 static uint32_t actual_point_start_time = 0;
101 /*!
102 \var actual_error
103 \brief Global variable to store and send the last error occured
104 */
105 static mavlink_error_t actual_error;
106 /*!
107 \var actual_raw_sensor
108 \brief Global variable to store and send the row sensor values from X
    and Y sensors
109 */
110 static mavlink_raw_sensor_t actual_raw_sensor[2];
111 /*!
112 \var send_msg
113 \brief Flag for sending status messages. Those messages are sent with
    lower freqency.
114 */
115 static int send_msg = 1;
116 /*!
117 \fn mouseDriver_initSetpoint
118 \brief Function that initializes the setpoint to 0
119
120 This function modifies \ref actual_speed_setpoint by setting it to 0.
121 */
122 #endif
123 /*!
124 \fn mouseDriver_sendMsg(uint32_t msgid)
125 \param msgid is the ID of the message to be sent.
126 \brief Function that sends a message given its ID.
127 \attention This function can be called in interrupts whith a priority
    lower than 0 (1,2,3,...),
```

```
128 otherwise the HAL_Delay() function stall and the STM32 crashes.
129
130 This function access global variables to send information to the
    computer.
131 Given one message ID the functions reads the information from a global
    variable and
132 sends it using the DMA as soon as the previous messages are sent.
133 */
134 void mouseDriver_sendMsg(uint32_t msgid);
135 /*!
136 \fn mouseDriver_initSetpoint
137 \brief Function that initializes the motor setpoint to 0.
138
139 This function initializes \ref actual_speed_setpoint.
140 */
141 void mouseDriver_initSetpoint(void);
142 /*!
143 \fn mouseDriver_initMode
144 \brief Function that initializes the mode to MOUSE_MODE_STOP
145
146 This function modifies \ref actual_mode by setting it to
    MOUSE_MODE_STOP.
147 */
148 void mouseDriver_initMode(void);
149 /*!
150 \fn mouseDriver_initPoints
151 \brief Function that initializes the routine points for AUTO mode to 0.
152
153 This function modifies \ref points by setting all their fields to 0.
154 */
155 void mouseDriver_initPoints(void);/*!
156 \fn mouseDriver_setMode(uint8_t mode)
157 \param mode is the mode in which the driver should be set.
158 \brief Function that sets the mode of the machine.
159
160 This functions modifies the mode of the machine. Not all transitions
    are possible,
161 this functions verifies that the transitions are lawful.
162 */
163 void mouseDriver_setMode(uint8_t mode);
164
165
166 void mouseDriver_initSetpoint(void){
167   actual_speed_setpoint.setpoint_x = 0;
168   actual_speed_setpoint.setpoint_y = 0;
169 }
170 void mouseDriver_initMode(void){
171   actual_mode = MOUSE_MODE_STOP;
172 }
173 void mouseDriver_initPoints(void){
174   for(int i=0; i<MAX_POINTS; i++){
175     points[i].duration = 0;
176     points[i].setpoint_x = 0;
177     points[i].setpoint_y = 0;
178     points[i].point_id = 0;
179   }
180   actual_point = 0;
181   actual_point_start_time = 0;
```

```
182 }
183 void mouseDriver_initMotorSignal(void){
184     actual_motor_signal.motor_x = 0;
185     actual_motor_signal.motor_y = 0;
186 }
187 void mouseDriver_init(void){
188   mouseDriver_initMode();
189   mouseDriver_initSetpoint();
190   mouseDriver_initPoints();
191   mouseDriver_initMotorSignal();
192 
193   /* Init sensor as well */
194   sensorDriver_init();
195   main_stop_motors();
196 }
197 uint32_t mouseDriver_getTime (void){
198   return (HAL_GetTick());
199 }
200 void mouseDriver_send_status_msg(void){
201   send_msg = 1;
202 }
203 void mouseDriver_control_idle(void){
204   static int count = 0;
205   if (actual_speed_measure.valid == 0){
206     count ++;
207     if(count >= MAX_MISSING_MEASURES){
208       main_stop_motors();
209       mouseDriver_setMode(MOUSE_MODE_STOP);
210     }
211     return;
212   }
213   if (actual_mode == MOUSE_MODE_SPEED || actual_mode ==
     MOUSE_MODE_AUTO_RUN){
214     actual_motor_signal.time = mouseDriver_getTime();
215     actual_motor_signal.motor_x = (float)K*(actual_speed_setpoint.
     setpoint_x-actual_speed_measure.speed_x);
216     actual_motor_signal.motor_y = (float)K*(actual_speed_setpoint.
     setpoint_y-actual_speed_measure.speed_y);
217 
218     if (actual_motor_signal.motor_x > MAX_MOTOR_SIGNAL){
219         actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL;
220     }
221     if(actual_motor_signal.motor_y  > MAX_MOTOR_SIGNAL){
222             actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL;
223         }
224 
225     main_set_motors_speed(actual_motor_signal);
226     count = 0;
227   }
228   else{
229     actual_motor_signal.motor_x = 0;
230     actual_motor_signal.motor_y = 0;
231     main_stop_motors();
232   }
233 }
234 
235 void mouseDriver_setMode(uint8_t mode){
236     if (mode == MOUSE_MODE_STOP){
```

```
237             main_stop_motors();
238             actual_point = 0;
239             actual_mode = MOUSE_MODE_STOP;
240             mouseDriver_initMotorSignal();
241         }
242         if (mode == MOUSE_MODE_AUTO_LOAD){
243             actual_mode = mode;
244             mouseDriver_sendMsg(MAVLINK_MSG_ID_HEARTBEAT);
245         }
246         if (actual_mode == MOUSE_MODE_AUTO_LOAD && mode ==
        MOUSE_MODE_AUTO_RUN ){
247             actual_point = 0;
248             actual_point_start_time = mouseDriver_getTime();
249             actual_speed_setpoint.setpoint_x = points[0].setpoint_x;
250             actual_speed_setpoint.setpoint_y = points[0].setpoint_y;
251             actual_mode = mode;
252         }
253 
254         if (actual_mode != MOUSE_MODE_AUTO_RUN)
255             actual_mode = mode;
256 }
257 void mouseDriver_sendMsg(uint32_t msgid){
258     mavlink_message_t msg;
259     static uint8_t outBuffer[MAX_BYTE_BUFFER_SIZE];
260     static uint16_t msg_size = 0;
261 
262     while (main_get_huart_tx_state() == HAL_BUSY){
263         /*Wait for other messages to be sent*/
264         HAL_Delay(1);
265     }
266 
267     switch(msgid){
268         case MAVLINK_MSG_ID_HEARTBEAT:
269             mavlink_msg_heartbeat_pack(SYS_ID,COMP_ID, &msg,
        actual_mode, mouseDriver_getTime());
270             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
271             main_transmit_buffer(outBuffer, msg_size);
272             break;
273         case MAVLINK_MSG_ID_SPEED_SETPOINT:
274             mavlink_msg_speed_setpoint_encode(SYS_ID,COMP_ID, &msg, &
        actual_speed_setpoint);
275             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
276             main_transmit_buffer(outBuffer, msg_size);
277             break;
278         case MAVLINK_MSG_ID_MOTOR_SETPOINT:
279             mavlink_msg_motor_setpoint_encode(SYS_ID,COMP_ID, &msg, &
        actual_motor_signal);
280             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
281             main_transmit_buffer(outBuffer, msg_size);
282             break;
283         case MAVLINK_MSG_ID_SPEED_INFO:
284             /* DEMO CODE INIT*/
285                 actual_speed_measure.time_x = mouseDriver_getTime();
286             /* DEMO CODE END*/
287             mavlink_msg_speed_info_encode(SYS_ID,COMP_ID, &msg, &
        actual_speed_measure);
288             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
289             main_transmit_buffer(outBuffer, msg_size);
```

```
290             break;
291         case MAVLINK_MSG_ID_ERROR:
292             mavlink_msg_error_encode(SYS_ID,COMP_ID,&msg,&actual_error)
    ;
293             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
294             main_transmit_buffer(outBuffer, msg_size);
295             break;
296         case MAVLINK_MSG_ID_POINT_LOADED:
297             mavlink_msg_point_loaded_pack(SYS_ID,COMP_ID,&msg,
    actual_point);
298             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
299             main_transmit_buffer(outBuffer, msg_size);
300             break;
301         case MAVLINK_MSG_ID_POINT:
302             mavlink_msg_point_encode(SYS_ID,COMP_ID,&msg,&points[
    actual_point]);
303             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
304             main_transmit_buffer(outBuffer, msg_size);
305             break;
306         case MAVLINK_MSG_ID_RAW_SENSOR:
307             mavlink_msg_raw_sensor_encode(SYS_ID,COMP_ID,&msg,&
    actual_raw_sensor[0]);
308             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
309             main_transmit_buffer(outBuffer, msg_size);
310             while (main_get_huart_tx_state() == HAL_BUSY){
311                     /*Wait for other messages to be sent*/
312                     HAL_Delay(1);
313             }
314             mavlink_msg_raw_sensor_encode(SYS_ID,COMP_ID,&msg,&
    actual_raw_sensor[1]);
315             msg_size = mavlink_msg_to_send_buffer(outBuffer, &msg);
316             main_transmit_buffer(outBuffer, msg_size);
317             break;
318         default:
319             break;
320     }
321 }
322 void mouseDriver_idle (void){
323     uint64_t difference = 0;
324     sensorDriver_motion_read_speed(actual_raw_sensor, &
    actual_speed_measure);
325     switch(actual_mode){
326     case MOUSE_MODE_STOP:
327         mouseDriver_initSetpoint();
328         mouseDriver_initMotorSignal();
329         actual_motor_signal.time = mouseDriver_getTime();
330         main_stop_motors();
331         mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_INFO);
332
333         break;
334     case MOUSE_MODE_SPEED:
335         mouseDriver_control_idle();
336         mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_INFO);
337         mouseDriver_sendMsg(MAVLINK_MSG_ID_MOTOR_SETPOINT);
338
339         break;
340     case MOUSE_MODE_AUTO_LOAD:
341         if (actual_point == 255){
```

```
342             actual_error.error = MOUSE_ROUTINE_TOO_LONG;
343             actual_error.time = mouseDriver_getTime();
344             mouseDriver_sendMsg(MAVLINK_MSG_ID_ERROR);
345         }
346         break;
347     case MOUSE_MODE_AUTO_RUN:
348         difference = mouseDriver_getTime()-actual_point_start_time;
349         if (difference >= points[actual_point].duration){
350             if (actual_point < MAX_POINTS-1){
351                 actual_point++;
352 
353                 if(points[actual_point].duration == 0){
354                     actual_point = 0;
355                 }
356                 actual_speed_setpoint.setpoint_x = points[actual_point
    ].setpoint_x;
357                 actual_speed_setpoint.setpoint_y = points[actual_point
    ].setpoint_y;
358                 actual_point_start_time = mouseDriver_getTime();
359             }
360         }
361         if (actual_point == MAX_POINTS){
362             mouseDriver_setMode(MOUSE_MODE_AUTO_LOAD);
363         }
364         mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_INFO);
365         mouseDriver_sendMsg(MAVLINK_MSG_ID_MOTOR_SETPOINT);
366         mouseDriver_control_idle();
367         break;
368     default:
369         break;
370     }
371     if (send_msg == 1){
372         send_msg = 0;
373         mouseDriver_sendMsg(MAVLINK_MSG_ID_HEARTBEAT);
374         if(actual_mode != MOUSE_MODE_AUTO_LOAD){
375             mouseDriver_sendMsg(MAVLINK_MSG_ID_SPEED_SETPOINT);
376             mouseDriver_sendMsg(MAVLINK_MSG_ID_RAW_SENSOR);
377             mouseDriver_sendMsg(MAVLINK_MSG_ID_MOTOR_SETPOINT);
378         }
379     }
380 
381 }
382 void mouseDriver_readMsg(const mavlink_message_t msg){
383 
384     switch(msg.msgid){
385 
386     case MAVLINK_MSG_ID_MODE_SELECTION:
387         mouseDriver_setMode( mavlink_msg_mode_selection_get_mode(&msg))
    ;
388         break;
389 
390     case MAVLINK_MSG_ID_SPEED_SETPOINT:
391         if (actual_mode == MOUSE_MODE_SPEED)
392             mavlink_msg_speed_setpoint_decode(&msg, &
    actual_speed_setpoint);
393         break;
394 
395     case MAVLINK_MSG_ID_MOTOR_SETPOINT:
```

```
396             if (actual_mode == MOUSE_MODE_SPEED)
397                 mavlink_msg_speed_setpoint_decode(&msg, &
    actual_speed_setpoint);
398             break;
399         case MAVLINK_MSG_ID_POINT:
400             if(actual_mode == MOUSE_MODE_AUTO_LOAD){
401                 mavlink_msg_point_decode(&msg, &points[actual_point]);
402                 if (actual_point == 255){
403                     actual_error.error = MOUSE_ROUTINE_TOO_LONG;
404                     actual_error.time = mouseDriver_getTime();
405                     mouseDriver_sendMsg(MAVLINK_MSG_ID_ERROR);
406                 }
407                 mouseDriver_sendMsg(MAVLINK_MSG_ID_POINT_LOADED);
408                 actual_point ++;
409 
410             }
411             break;
412         default:
413             break;
414         };
415 }
416 #endif
```

```
1  /*! \file mouseDriver.h
2  \brief Header of the driver for the mouse treadmil project.
3 
4  \author Didier Negretto
5  */
6 
7  /*
8   * Code used for driving the 3D mouse treadmill
9   * Author: Didier Negretto
10  *
11  */
12 
13 #pragma once
14 #ifndef MOUSEDRIVER_N_H
15 /*!
16 \def MOUSEDRIVER_N_H
17 \brief To avoid double includes
18 */
19 #define MOUSEDRIVER_N_H
20 
21 #ifndef TEST
22 #include "mavlink.h"
23 #include "utils.h"
24 #include "sensorDriver.h"
25 #endif
26 
27 #include <math.h>
28 /* Constants for MALINK functions*/
29 
30 /*!
31 \def SYS_ID
32 \brief System ID for MAVLink
33 */
34 #define SYS_ID 0
35 /*!
```

```
36 \def COMP_ID
37 \brief Component ID for MAVLink
38 */
39 #define COMP_ID 0
40 /* maximum size of the trasmit buffer */
41 /*!
42 \def MAX_BYTE_BUFFER_SIZE
43 \brief MAX size of transmit buffer in bytes
44 */
45 #define MAX_BYTE_BUFFER_SIZE 500
46
47 /*!
48 \def MAX_POINTS
49 \brief MAX amount of points that can be defined in AUTO mode
50 */
51 #define MAX_POINTS 255
52
53
54 /*!
55 \fn mouseDriver_init
56 \brief Function that initializes the driver of the mouse treadmill.
57
58 This functions initialites the mouse treadmill driver. It initializes
    the sensors as well.
59 */
60 void mouseDriver_init(void);
61
62
63
64 /*!
65 \fn mouseDriver_control_idle
66 \brief Function doing the control on the motors.
67 \attention This function is in charge of generating the control signals
     for the
68 motors. If it is modified, make sure to respect the specifications of
    the motor
69 to avoid damaging or destroing them !!
70
71 This function is called periodially to update the control signal for
    the motors.
72 */
73 void mouseDriver_control_idle(void);
74
75 /*!
76 \fn mouseDriver_send_status_msg
77 \brief Function generating the signal for sending messages.
78
79 This function is called periodially to set the flag for sending status
    messages.
80 */
81 void mouseDriver_send_status_msg(void);
82
83 /*!
84 \fn mouseDriver_readMsg(const mavlink_message_t msg)
85 \param msg MAVLink message to be decoded
86 \brief Function that reads one message.
87
88 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
```

A.3 Sensor driver

```
/*
 * sensorDriver.c
 *
 *  Created on: Nov 11, 2019
 *      Author: Didier
 */

# 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.
```

```
27
28  After Flashing the SROM the SROM_ID register is read to confirm that
      the
29  SROM have been flashed correctly.
30  */
31 void sensorDriver_powerup(sensor_t * sensor);
32
33 /*!
34  \fn sensorDriver_motion_read_raw(uint8_t sensor_id,
      mavlink_raw_sensor_t * sensor_data)
35  \param sensor_id 0 for sensor x, 1 for sensor y
36  \param sensor_data pointer to a structure for storing the raw sensor
      value
37  \brief This function reads raw data from the sensor given its ID and
      puts the result in the pointer.
38  */
39 void sensorDriver_motion_read_raw(uint8_t sensor_id,
      mavlink_raw_sensor_t * sensor_data);
40
41 void sensorDriver_powerup(sensor_t * sensor){
42   /* Disable the sensor */
43   HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin, GPIO_PIN_SET);
44
45   /* Make sure all sensor is switched off */
46   HAL_GPIO_WritePin(sensor->pw_port, sensor->pw_pin, GPIO_PIN_RESET);
47   main_write_sensor(*sensor, 0x00, 0x00);
48   HAL_Delay(100);
49
50   /* Gives voltage to sensors */
51   HAL_GPIO_WritePin(sensor->pw_port, sensor->pw_pin , GPIO_PIN_SET);
52   HAL_Delay(300);
53
54   /* Reset SPI port */
55   HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin,  GPIO_PIN_SET);
56   HAL_Delay(5);
57   HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin,  GPIO_PIN_RESET);
58   HAL_Delay(5);
59   HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin,  GPIO_PIN_SET);
60   HAL_Delay(5);
61
62   /* Write to Power_up_Reset register */
63   main_write_sensor(*sensor, Power_Up_Reset, 0x5A);
64
65   /* Wait at least 50 ms */
66   HAL_Delay(50);
67
68   /* Read from data registers */
69   main_read_sensor(*sensor, 0x02);
70   main_read_sensor(*sensor, 0x03);
71   main_read_sensor(*sensor, 0x04);
72   main_read_sensor(*sensor, 0x05);
73   main_read_sensor(*sensor, 0x06);
74
75   /* Start ROM Download */
76   main_write_sensor(*sensor, Config2, 0x20);
77   main_write_sensor(*sensor, SROM_Enable, 0x1d);
78   HAL_Delay(10);
79   main_write_sensor(*sensor,SROM_Enable, 0x18);
```

```
80    main_wait_160us();
81    main_wait_20us();
82
83    /* Burst start with adress */
84    HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin, GPIO_PIN_RESET);
85    main_write_sensor_burst(SROM_Load_Burst|0x80);
86    for (int i = 0; i < firmware_length; i++ ){
87      main_write_sensor_burst(firmware_data[i]);
88    }
89    HAL_GPIO_WritePin(sensor->cs_port, sensor->cs_pin, GPIO_PIN_SET);
90    main_wait_160us();
91    main_wait_20us();
92    main_wait_20us();
93
94    /* Read SROM_ID for verification */
95    sensor->status = main_read_sensor(*sensor, SROM_ID);
96
97    /* Write to Config2 for wired mouse */
98    main_write_sensor(*sensor, Config2, 0x00);
99  }
100 void sensorDriver_init(void){
101   sensorDriver_powerup(&sensor_x);
102   sensorDriver_powerup(&sensor_y);
103 }
104 void sensorDriver_motion_read_raw(uint8_t sensor_id,
      mavlink_raw_sensor_t * sensor_data){
105   uint8_t data[12];
106   int16_t temp = 0;
107   sensor_t sensor;
108
109   if (sensor_id == SENSOR_X) sensor = sensor_x;
110   else if (sensor_id == SENSOR_Y) sensor = sensor_y;
111   else return;
112   sensor_data->sensor_id = sensor_id;
113
114   /* write to motion burst adress */
115   main_write_sensor(sensor, Motion_Burst, 0xbb);
116
117   /* Prepare for burst */
118   HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_RESET);
119   sensor_data->time = mouseDriver_getTime();
120   main_write_sensor_burst(Motion_Burst);
121   /* Start burst */
122   main_read_sensor_motion_burst(data);
123   HAL_GPIO_WritePin(sensor.cs_port, sensor.cs_pin, GPIO_PIN_SET);
124   /* END of burst */
125   main_wait_20us();
126
127   /* Read other register for stopping burst mode */
128   sensor_data->product_id = main_read_sensor(sensor,  Product_ID);
129
130   /* TWO's Complement */
131   temp = (data[DELTA_X_H]<<8) | (data[DELTA_X_L]);
132   temp = ~temp + 1;
133   sensor_data->delta_x = temp;
134   temp = (data[DELTA_Y_H]<<8) | (data[DELTA_Y_L]);
135   temp = ~temp +1;
136   sensor_data->delta_y = temp;
```

```
137
138   sensor_data->squal = data[SQUAL_READ];
139   sensor_data->lift = (data[MOTION] & 0x08) >> 3;
140   sensor_data->srom_id = sensor.status;
141 }
142 void sensorDriver_motion_read_speed(mavlink_raw_sensor_t sensor_data
      [2], mavlink_speed_info_t * speed_info){
143   mavlink_raw_sensor_t raw_values[2];
144     uint32_t old_time[2];
145
146     speed_info->valid = 0;
147     old_time[0] = speed_info->time_x;
148     old_time[1] = speed_info->time_y;
149
150   sensorDriver_motion_read_raw(SENSOR_X, &raw_values[0]);
151   sensorDriver_motion_read_raw(SENSOR_Y, &raw_values[1]);
152
153   speed_info->speed_x =  (float)raw_values[0].delta_x*(float)INCH2METER
      /(float)RESOLUTION;
154   speed_info->speed_x /= (float)(raw_values[0].time-old_time[0])/(float
      )1000;
155   speed_info->time_x = raw_values[0].time;
156   speed_info->speed_y =  (float)raw_values[1].delta_x*(float)INCH2METER
      /(float)RESOLUTION;
157   speed_info->speed_y /= (float)(raw_values[1].time-old_time[1])/(float
      )1000;
158   speed_info->time_y = raw_values[1].time;
159   sensor_data[0] = raw_values[0];
160   sensor_data[1] = raw_values[1];
161
162   if((raw_values[0].lift == 0) && (raw_values[1].lift == 0) &&
163     (raw_values[0].squal >= SQUAL_THRESH) && (raw_values[0].squal >=
      SQUAL_THRESH) &&
164     (raw_values[0].product_id == 66) && (raw_values[1].product_id ==
      66)){
165     speed_info->valid = 1;
166   }
167   else{
168     speed_info->valid = 0;
169   }
170 }
```

```
1 #pragma once
2
3 #ifndef SENSORDRIVER_H_
4 #define SENSORDRIVER_H_
5
6 #ifndef TEST
7 #include "main.h"
8 #include "mavlink.h"
9 #include "sensorSROM.h"
10 #endif
11
12 /* BEGIN DEFINES FOR SENSOR INTERNAL REGISTERS */
13 #define Product_ID  0x00
14 #define Revision_ID 0x01
15 #define Motion  0x02
16 #define Delta_X_L 0x03
```

```
17 #define Delta_X_H 0x04
18 #define Delta_Y_L 0x05
19 #define Delta_Y_H 0x06
20 #define SQUAL 0x07
21 #define Raw_Data_Sum  0x08
22 #define Maximum_Raw_data  0x09
23 #define Minimum_Raw_data  0x0A
24 #define Shutter_Lower 0x0B
25 #define Shutter_Upper 0x0C
26 #define Control 0x0D
27 #define Config1 0x0F
28 #define Config2 0x10
29 #define Angle_Tune  0x11
30 #define Frame_Capture 0x12
31 #define SROM_Enable 0x13
32 #define Run_Downshift 0x14
33 #define Rest1_Rate_Lower  0x15
34 #define Rest1_Rate_Upper  0x16
35 #define Rest1_Downshift 0x17
36 #define Rest2_Rate_Lower  0x18
37 #define Rest2_Rate_Upper  0x19
38 #define Rest2_Downshift 0x1A
39 #define Rest3_Rate_Lower  0x1B
40 #define Rest3_Rate_Upper  0x1C
41 #define Observation 0x24
42 #define Data_Out_Lower  0x25
43 #define Data_Out_Upper  0x26
44 #define Raw_Data_Dump 0x29
45 #define SROM_ID 0x2A
46 #define Min_SQ_Run  0x2B
47 #define Raw_Data_Threshold  0x2C
48 #define Config5 0x2F
49 #define Power_Up_Reset  0x3A
50 #define Shutdown  0x3B
51 #define Inverse_Product_ID  0x3F
52 #define LiftCutoff_Tune3  0x41
53 #define Angle_Snap  0x42
54 #define LiftCutoff_Tune1  0x4A
55 #define Motion_Burst  0x50
56 #define LiftCutoff_Tune_Timeout 0x58
57 #define LiftCutoff_Tune_Min_Length  0x5A
58 #define SROM_Load_Burst 0x62
59 #define Lift_Config 0x63
60 #define Raw_Data_Burst  0x64
61 #define LiftCutoff_Tune2  0x65
62 /* END DEFINES FOR SENSOR INTERNAL REGISTERS */
63
64 #include <mavlink_msg_raw_sensor.h>
65 #include <stdint.h>
66
67 /* DEFINES FOR BURST READ (only usefull data) */
68 #define MOTION 0
69 #define OBSERVATION 1
70 #define DELTA_X_L 2
71 #define DELTA_X_H 3
72 #define DELTA_Y_L 4
73 #define DELTA_Y_H 5
74 #define SQUAL_READ 6
```

```
75
76 /*!
77 \def SQUAL_THRESH
78 \brief Threshold value on SQUAL to consider the measure valid.
79 */
80 #define SQUAL_THRESH 16
81
82 /*!
83 \def RESOLUTION
84 \brief Resolution of the sensor in Count per Inch (CPI)
85 \note This value needs to be updated if the resolution of the sensors
       is changed,
86
87 This value is used to convert the raw sensor value in counts to meter
       per second.
88 */
89 #define RESOLUTION 5000
90
91 /*!
92 \def INCH2METER
93 \brief Conversion factor to convert inches in meters.
94 */
95 #define INCH2METER 0.0254
96
97 /*!
98 \fn sensorDriver_init
99 \brief Initializes all sensors.
100
101 This functions powers down the sensor and does the powering up routine.
102 \note This routine takes a long time, so it is done only at start up.
103 */
104 void sensorDriver_init(void);
105
106 /*!
107 \fn sensorDriver_motion_read_speed(mavlink_raw_sensor_t sensor_data[2],
        mavlink_speed_info_t * speed_info)
108 \param sensor_data[2] array for the raw values of the 2 sensors
109 \param speed_info pointer to a mavlink_speed_info_t
110 \brief Function for reading the raw data and speed measures from the
       sensors.
111 \attention The speed_info.time_x/y is used to compute speed. This value
        should NOT BE MODIFIED by
112 the caller function
113
114 This function reads values from the sensors and puts them in the given
       pointers.
115 It also flags invalid readings, so that \ref mouseDriver_control_idle
       do not use them.
116 */
117 void sensorDriver_motion_read_speed(mavlink_raw_sensor_t sensor_data
       [2], mavlink_speed_info_t * speed_info);
118
119 #endif
```

A.4 Code for unit tests

```
1 /*
2  * display.h
```

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

#ifndef DISPLAY_H_
#define DISPLAY_H_

#define RED     "\x1b[31m"
#define GREEN   "\x1b[32m"
#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

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_ */
```

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

```
7
8  #include "test_mouseDriver.h"
9  #include "test_sensorDriver.h"
10
11 int main(void){
12
13     bool test = 1;
14
15     printf("=============================================================\n");
16     printf("**********TESTING CODE FOR MOUSE TREADMILL *********\n");
17     printf("=============================================================\n\n")
       ;
18     printf("-------------------------------------------------------------\n");
19     printf("TESTING mouseDriver.c\n");
20     printf("TESTING mouseDriver_init()\n");
21     test &= test_mouseDriver_init();
22     printf("TESTING mouseDriver_idle()\n");
23     test &= test_mouseDriver_idle();
24     printf("TESTING mouseDriver_getTime()\n");
25     test &= test_mouseDriver_getTime();
26     printf("TESTING mouseDriver_send_status_msg()\n");
27     test &= test_mouseDriver_send_status_msg();
28     printf("TESTING mouseDriver_control_idle()\n");
29     test &= test_mouseDriver_control_idle();
30     /*printf("-------------------------------------------------------------\n")
       ;
31     printf("TESTING mouseDriver.c\n");
32     if (! test_mouseDriver_init()) printf(RED"ERRORS IN
       mouseDriver_init\n"END);*/
33
34
35     if (test == 1){
36         printf("ALL TEST PASSED SUCCESSUFULLY\n");
37     }
38     else{
39         printf("=============================================================\n
       ");
40         printf("!!!!!!!!!!!! SOME TESTS NOT PASSED !!!!!!!!!!!!!!!!!!\n
       ");
41         printf("=============================================================\n
       \n");
42
43     }
44     return test;
45 }
```

```
1  /*
2   * mock_mouseDriver.h
3   *
4   *  Created on: Nov 24, 2019
5   *      Author: Didier
6   */
7
8  #ifndef MOCK_MOUSEDRIVER_H_
9  #define MOCK_MOUSEDRIVER_H_
10
11 #define HAL_BUSY  0
12 #define SYS_ID 0
```

```
13 #define COMP_ID 0
14 #define MAX_BYTE_BUFFER_SIZE 500
15 #define MAX_POINTS 255
16
17
18 static int stop_motor = 0;
19 static int sensor_init = 0;
20 static int sensor_read_x = 0;
21 static int sensor_read_y = 0;
22
23 /* Define mock variables for testing */
24 static int send_msg = 1;
25 static uint8_t actual_mode = MOUSE_MODE_STOP;
26 static mavlink_speed_setpoint_t actual_speed_setpoint;
27 static mavlink_speed_info_t actual_speed_measure;
28 static mavlink_motor_setpoint_t actual_motor_signal;
29 static mavlink_point_t points[255];
30 static uint8_t actual_point = 0;
31 static uint32_t actual_point_start_time = 0;
32 static mavlink_error_t actual_error;
33 static mavlink_raw_sensor_t actual_raw_sensor[2];
34
35 /* Define mock functions */
36 static inline void sensorDriver_init(void){sensor_init = 1; };
37 static inline uint32_t HAL_GetTick(void){
38     static uint32_t i = 0;
39     i++;
40     return i;
41 };
42 static inline void main_set_motors_speed(mavlink_motor_setpoint_t
    actual_motor_signal){stop_motor = 0;};
43 static inline void main_stop_motors(void){stop_motor = 1;};
44 static inline int main_get_huart_tx_state(void){return 1;};
45 static inline void HAL_Delay(int delay){};
46 static inline void main_transmit_buffer(uint8_t * outbuffer, int
    msg_size){};
47
48 static inline void sensorDriver_motion_read_speed(mavlink_raw_sensor_t
    actual_raw_sensor[2], mavlink_speed_info_t * actual_speed_measure){
49     sensor_read_x = 1;
50     sensor_read_y = 1;
51     actual_raw_sensor[0].delta_x = 0;
52     actual_raw_sensor[1].delta_y = 0;
53     actual_speed_measure->speed_x = 0;
54     actual_speed_measure->speed_y = 0;
55 };
56
57 #endif /* MOCK_MOUSEDRIVER_H_ */
```

```
1 /*
2  * mock_sensorDriver.h
3  *
4  *  Created on: Nov 25, 2019
5  *      Author: Didier
6  */
7
8 #ifndef MOCK_SENSORDRIVER_H_
9 #define MOCK_SENSORDRIVER_H_
```

```

typedef struct SENSOR{
    int cs_port;
    uint8_t cs_pin;
    int pw_port;
    uint8_t pw_pin;
    uint8_t status;
} 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;
```

```
29      actual_motor_signal.motor_y = 10;
30
31      sensor_init = 0;
32      stop_motor = 0;
33
34      mouseDriver_init();
35
36      test &= display(actual_mode == 0, "actual_mode initialization");
37      test &= display(actual_point == 0, "actual_point initialization");
38      test &= display(actual_point_start_time == 0, "
        actual_point_start_time initialization");
39      test &= display((actual_speed_setpoint.setpoint_y  == 0)&& (
        actual_speed_setpoint.setpoint_x  == 0), "actual_speed_setpoint
        initialization");
40      bool test_sub = 1;
41      for(int i = 0; i < MAX_POINTS; i++){
42          test_sub &= ((points[i].duration == 0) && (points[i].setpoint_x
        == 0) &&
43                       (points[i].setpoint_y == 0) && (points[i].point_id
        == 0));
44      }
45      test &= display(test_sub, "points initialized correctly");
46      test &= display(sensor_init == 1, "sensor_init initialization");
47      test &= display(stop_motor == 1, "stop_motor initialization");
48      test &= display((actual_motor_signal.motor_x  == 0)&& (
        actual_motor_signal.motor_y  == 0), "actual_motor_signal
        initialization");
49
50      return test;
51  }
52
53  bool test_mouseDriver_idle(void){
54      bool test = false;
55      actual_speed_measure.speed_x = -10;
56      actual_speed_measure.speed_y = -10;
57      actual_speed_measure.valid = 1;
58      actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
59      actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
60      actual_point_start_time = 0;
61      actual_point = 0;
62      points[0].duration = 100;
63      points[0].setpoint_x = 10;
64      points[0].setpoint_y = 10;
65      points[0].point_id = 0;
66
67      /* Test reading of sensors in SPEED mode */
68      actual_mode = MOUSE_MODE_SPEED;
69      sensor_read_x = 0;
70      sensor_read_y = 0;
71      stop_motor = 1;
72      mouseDriver_idle();
73      test = display(sensor_read_x == 1, "read sensor x in
        MOUSE_MODE_SPEED");
74      test &= display(sensor_read_y == 1, "read sensor y in
        MOUSE_MODE_SPEED");
75      test &= display(stop_motor == 0, "motor started in MOUSE_MODE_SPEED
        ");
76
```

```
77      /* Test reading of sensors in MOUSE_MODE_AUTO_RUN mode */
78      actual_mode = MOUSE_MODE_AUTO_RUN;
79      sensor_read_x = 0;
80      sensor_read_y = 0;
81      stop_motor = 1;
82      mouseDriver_idle();
83      test &= display(sensor_read_x == 1, "read sensor x in
        MOUSE_MODE_AUTO_RUN");
84      test &= display(sensor_read_y == 1, "read sensor y in
        MOUSE_MODE_AUTO_RUN");
85      test &= display(stop_motor == 0, "motor started in
        MOUSE_MODE_AUTO_RUN");
86      return test;
87  }
88  bool test_mouseDriver_getTime(void){
89      bool test = 1;
90      uint32_t start = HAL_GetTick();
91      test &= mouseDriver_getTime() == start+1;
92      test &= mouseDriver_getTime() == start+2;
93      test &= mouseDriver_getTime() == start+3;
94      test &= mouseDriver_getTime() == start+4;
95      test &= mouseDriver_getTime() == start+5;
96      display(test, "time update");
97
98      return test;
99  }
100 bool test_mouseDriver_send_status_msg(void){
101     bool test = false;
102     send_msg = 0;
103
104     mouseDriver_send_status_msg();
105
106     test = send_msg;
107     display(test, "status message send request");
108     return test;
109 }
110 bool test_mouseDriver_control_idle(void){
111     bool test = 1;
112     stop_motor = 0;
113     actual_speed_measure.speed_x = -10;
114     actual_speed_measure.speed_y = -10;
115     actual_motor_signal.motor_x = 10;
116     actual_motor_signal.motor_y = 10;
117     actual_mode = MOUSE_MODE_STOP;
118
119     /* Case actual mode == STOP */
120     printf("if (actual_mode == MOUSE_MODE_STOP)\n");
121     mouseDriver_control_idle();
122     test &= display((actual_motor_signal.motor_x == 0)&& (
        actual_motor_signal.motor_y == 0), "actual_motor_signal reset");
123     test &= display(stop_motor == 1, "motor stop");
124
125     /* Case actual mode == SPEED */
126     actual_mode = MOUSE_MODE_SPEED;
127     stop_motor = 1;
128     actual_speed_setpoint.setpoint_y = 0;
129     actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
130     actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
```

```
131     actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
132     printf("if (actual_mode == MOUSE_MODE_SPEED)\n");
133     mouseDriver_control_idle();
134     test &= display(stop_motor == 0, "motor_x speed changed");
135     for(int i = 0; i < 100; i++)
136         mouseDriver_control_idle();
137     test &= display(actual_motor_signal.motor_x <= MAX_MOTOR_SIGNAL, "
    motor_x with MAX_MOTOR_SIGNAL limit");
138 
139     stop_motor = 1;
140     actual_speed_setpoint.setpoint_x = 0;
141     actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
142     actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
143     actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
144     mouseDriver_control_idle();
145     test &= display(stop_motor == 0, "motor_y speed changed");
146     for(int i = 0; i < 100; i++)
147         mouseDriver_control_idle();
148     test &= display(actual_motor_signal.motor_y <= MAX_MOTOR_SIGNAL, "
    motor_y with MAX_MOTOR_SIGNAL limit");
149 
150     actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
151     actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
152     actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
153     actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
154     mouseDriver_control_idle();
155     test &= display(stop_motor == 0, "motor_y and motor_x speed changed
    ");
156     for(int i = 0; i < 100; i++)
157         mouseDriver_control_idle();
158     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");
159 
160     /* Reaction to invalid measures */
161     actual_speed_setpoint.setpoint_x = 0;
162     actual_speed_setpoint.setpoint_y = 0;
163     actual_speed_measure.speed_x = 1000;
164     actual_speed_measure.speed_y = 1000;
165     actual_motor_signal.motor_x = 10;
166     actual_motor_signal.motor_y = 10;
167     bool test_stop = true;
168     actual_speed_measure.valid = 0;
169     for(int i = 0;i < MAX_MISSING_MEASURES-1; i++ ){
170         test_stop &= (actual_motor_signal.motor_x == 10);
171         test_stop &= (actual_motor_signal.motor_y == 10);
172         mouseDriver_control_idle();
173     }
174     mouseDriver_control_idle();
175     test &= display(test_stop, "constant motor signal if invalid
    measure");
176     test &= display(actual_mode == MOUSE_MODE_STOP, "stop motor after
    too many invalid measures");
177 
178 
179 
180     /* Case actual mode == SPEED */
181     actual_mode = MOUSE_MODE_AUTO_RUN;
```

```
182     stop_motor = 1;
183     actual_speed_setpoint.setpoint_y = 0;
184     actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
185     actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
186     actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
187     actual_speed_measure.valid = 1;
188     printf("if (actual_mode == MOUSE_MODE_AUTO_RUN)\n");
189     mouseDriver_control_idle();
190     test &= display(stop_motor == 0, "motor_x speed changed");
191     for(int i = 0; i < 100; i++)
192         mouseDriver_control_idle();
193     test &= display(actual_motor_signal.motor_x <= MAX_MOTOR_SIGNAL, "
    motor_x with MAX_MOTOR_SIGNAL limit");
194
195     stop_motor = 1;
196     actual_speed_setpoint.setpoint_x = 0;
197     actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
198     actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
199     actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
200     mouseDriver_control_idle();
201     test &= display(stop_motor == 0, "motor_y speed changed");
202     for(int i = 0; i < 100; i++)
203         mouseDriver_control_idle();
204     test &= display(actual_motor_signal.motor_y <= MAX_MOTOR_SIGNAL, "
    motor_y with MAX_MOTOR_SIGNAL limit");
205
206     actual_speed_setpoint.setpoint_x = MAX_MOTOR_SIGNAL * 1000;
207     actual_speed_setpoint.setpoint_y = MAX_MOTOR_SIGNAL * 1000;
208     actual_motor_signal.motor_x = MAX_MOTOR_SIGNAL * 1000;
209     actual_motor_signal.motor_y = MAX_MOTOR_SIGNAL * 1000;
210     mouseDriver_control_idle();
211     test &= display(stop_motor == 0, "motor_y and motor_x speed changed
    ");
212     for(int i = 0; i < 100; i++)
213         mouseDriver_control_idle();
214     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");
215
216     test_stop = true;
217     actual_speed_measure.valid = 0;
218     actual_motor_signal.motor_x = 10;
219     actual_motor_signal.motor_y = 10;
220     for(int i = 0;i < MAX_MISSING_MEASURES-1; i++ ){
221         test_stop &= (actual_motor_signal.motor_x == 10);
222         test_stop &= (actual_motor_signal.motor_y == 10);
223         mouseDriver_control_idle();
224     }
225     mouseDriver_control_idle();
226     test &= display(test_stop, "constant motor signal if invalid
    measure");
227     test &= display(actual_mode == MOUSE_MODE_STOP, "stop motor after
    too many invalid measures");
228
229     return test;
230 }
```

```
1 /*
```

```
 * 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");
}
```

A.5 Build script

```
#!/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
```

B Code for PC

B.1 GUI

```
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},
```

```
23          "SPEED_INFO":  {"time": [], "speed_x": [], "speed_y": [], "
    start": 0},
24          "MOTOR_SETPOINT": {"time": [], "motor_x": [], "motor_y": [], "
    start": 0}
25          }
26 LOG = []
27 MAX_SAMPLES_ON_SCREEN = 200
28 print(mouseController.MAVLink_speed_info_message.fieldnames)
29 port = "/dev/cu.usbmodem14102"
30
31 class MyApplication():
32     actualMode = 0
33     actualTime = 0
34     actualSpeedSetpoint = [None, None]
35     actualMotorSetpoint = [None, None]
36     actualSpeedInfo = [None, None]
37     connection = serial.Serial(port, baudrate = 230400, timeout = 50)
38     mavlink = mouseController.MAVLink(file = connection )
39     setpointX = 0.0
40     setpointY = 0.0
41
42     def commSTM32 (self):
43        # Init variables
44        m = None
45        while(self.connection.in_waiting>0):
46            # Recive messages
47            try:
48                m = self.mavlink.parse_char(self.connection.read())
49            except:
50                pass
51            if m:
52                LOG.append(m)
53                if m.name == "HEARTBEAT":
54                    self.actualTime = m.time
55                    self.actualMode = m.mode
56                    DATA["HEARTBEAT"]["time"].append(self.actualTime)
57                    DATA["HEARTBEAT"]["mode"].append(self.actualMode)
58                elif m.name == "SPEED_SETPOINT":
59                    self.actualSpeedSetpoint[0] = m.setpoint_x
60                    self.actualSpeedSetpoint[1] = m.setpoint_y
61                    DATA["SPEED_SETPOINT"]["time"].append(self.
    actualTime)
62                    DATA["SPEED_SETPOINT"]["setpoint_x"].append(self.
    actualSpeedSetpoint[0])
63                    DATA["SPEED_SETPOINT"]["setpoint_y"].append(self.
    actualSpeedSetpoint[1])
64                    #DATA["SPEED_SETPOINT"]["setpoint_z"].append(self.
    actualSpeedSetpoint[2])
65                elif m.name == "MOTOR_SETPOINT":
66                    self.actualMotorSetpoint[0] = m.motor_x
67                    self.actualMotorSetpoint[1] = m.motor_y
68                    DATA["MOTOR_SETPOINT"]["time"].append(m.time)
69                    DATA["MOTOR_SETPOINT"]["motor_x"].append(self.
    actualMotorSetpoint[0])
70                    DATA["MOTOR_SETPOINT"]["motor_y"].append(self.
    actualMotorSetpoint[1])
71                    #DATA["SPEED_SETPOINT"]["motor_z"].append(self.
    actualMotorSetpoint[2])
```

```
72                  elif m.name == "SPEED_INFO":
73                      #print(m)
74                      DATA["SPEED_INFO"]["time"].append(m.time_x)
75                      DATA["SPEED_INFO"]["speed_x"].append(m.speed_x)
76                      #DATA["SPEED_INFO"]["speed_y"].append(m.speed_y)
77                      DATA["SPEED_INFO"]["speed_y"].append(0)
78                  elif m.name == "RAW_SENSOR":
79                      if m.sensor_id == 0:
80                          status_x = []
81                          status_x.append(m.product_id)
82                          status_x.append(m.lift)
83                          status_x.append(m.squal)
84                          status_x.append(m.srom_id)
85                      elif m.sensor_id == 1:
86                          status_y = []
87                          status_y.append(m.product_id)
88                          status_y.append(m.lift)
89                          status_y.append(m.squal)
90                          status_y.append(m.srom_id)
91                      try:
92                          if (len(status_x) == 4) and (len(status_y) ==
    4):
93                              self.app.setLabel("sensorStatus1",
    SENSOR_STATUS_MSG[1]+str(status_x[0])+"|"+str(status_y[0]))
94                              self.app.setLabel("sensorStatus2",
    SENSOR_STATUS_MSG[2]+str(status_x[1])+"|"+str(status_y[1]))
95                              self.app.setLabel("sensorStatus3",
    SENSOR_STATUS_MSG[3]+str(status_x[2])+"|"+str(status_y[2]))
96                              self.app.setLabel("sensorStatus4",
    SENSOR_STATUS_MSG[4]+str(status_x[3])+"|"+str(status_y[3]))
97                      except:
98                          pass
99
100
101                 elif m.name == "POINT":
102                     print(m)
103                 else:
104                     pass
105         m = None
106     def refreshPlot(self):
107
108         # Clear plot
109         for i in range(3):
110             self.ax[i].clear()
111
112         # Define labels
113         """
114         self.ax[2].set_xlabel("Time")
115         self.ax[2].set_ylabel("Measured speed [m/s]")
116         self.ax[1].set_ylabel("Speed setpoint [m/s]")
117         self.ax[0].set_ylabel("Motor signal [ ]")
118         """
119
120         # Limit max amout of points on one graph
121         if len(DATA["SPEED_INFO"]["time"][DATA["SPEED_INFO"]["start"
    ]:])-1>MAX_SAMPLES_ON_SCREEN:
122             DATA["SPEED_INFO"]["start"] = -MAX_SAMPLES_ON_SCREEN
123             DATA["SPEED_SETPOINT"]["start"] = -MAX_SAMPLES_ON_SCREEN
```

```
124             DATA["MOTOR_SETPOINT"]["start"] = -MAX_SAMPLES_ON_SCREEN
125
126         # Re-plot all graphs
127         self.ax[2].plot(DATA["SPEED_INFO"]["time"][DATA["SPEED_INFO"]["
    start"]:], DATA["SPEED_INFO"]["speed_x"][DATA["SPEED_INFO"]["start"
    ]:], 'b.')
128         self.ax[2].plot(DATA["SPEED_INFO"]["time"][DATA["SPEED_INFO"]["
    start"]:], DATA["SPEED_INFO"]["speed_y"][DATA["SPEED_INFO"]["start"
    ]:], 'r.')
129         self.ax[1].plot(DATA["SPEED_SETPOINT"]["time"][DATA["
    SPEED_SETPOINT"]["start"]:], DATA["SPEED_SETPOINT"]["setpoint_x"][
    DATA["SPEED_SETPOINT"]["start"]:], 'b.')
130         self.ax[1].plot(DATA["SPEED_SETPOINT"]["time"][DATA["
    SPEED_SETPOINT"]["start"]:], DATA["SPEED_SETPOINT"]["setpoint_y"][
    DATA["SPEED_SETPOINT"]["start"]:], 'r.')
131         self.ax[0].plot(DATA["MOTOR_SETPOINT"]["time"][DATA["
    MOTOR_SETPOINT"]["start"]:], DATA["MOTOR_SETPOINT"]["motor_x"][DATA
    ["MOTOR_SETPOINT"]["start"]:], 'b.')
132         self.ax[0].plot(DATA["MOTOR_SETPOINT"]["time"][DATA["
    MOTOR_SETPOINT"]["start"]:], DATA["MOTOR_SETPOINT"]["motor_y"][DATA
    ["MOTOR_SETPOINT"]["start"]:], 'r.')
133         self.ax[0].set_adjustable('box',True)
134         self.app.refreshPlot("plot")
135
136     def resetPlot(self):
137         DATA["SPEED_INFO"]["start"] = len(DATA["SPEED_INFO"]["time"])-3
138         DATA["SPEED_SETPOINT"]["start"] = len(DATA["SPEED_SETPOINT"]["
    time"])-3
139         DATA["MOTOR_SETPOINT"]["start"] = len(DATA["MOTOR_SETPOINT"]["
    time"])-3
140
141     def refreshGUI(self):
142         self.commSTM32()
143
144         # Refresh status bar
145         self.app.setStatusbar("Time: "+str(self.actualTime)+" [ms]", 0)
146         self.app.setStatusbar("Modes: "+str(MODES[self.actualMode]), 1)
147         self.refreshPlot()
148         """
149         self.app.setLabel("speedSetpointX", str(self.
    actualSpeedSetpoint[0]))
150         self.app.setLabel("speedSetpointY", str(self.
    actualSpeedSetpoint[1]))
151         self.app.setLabel("motorSetpointX", str(self.
    actualMotorSetpoint[0]))
152         self.app.setLabel("motorSetpointY", str(self.
    actualMotorSetpoint[1]))
153         """
154
155     def setMode(self):
156         self.mavlink.mode_selection_send(MODES_NUM[self.app.
    getRadioButton("optionMode")])
157         while(self.connection.out_waiting > 0):
158             time.sleep(0.001)
159         time.sleep(0.001)
160         if self.actualMode == mouseController.MOUSE_MODE_STOP:
161             self.setpointX = 0
162             self.setpointY = 0
```

```
163
164
165     def setSpeedX(self):
166         if self.actualMode == mouseController.MOUSE_MODE_SPEED:
167             self.setpointX = self.app.getEntry("speedX")
168             if self.setpointX is None or  self.setpointY is None :
169                 pass
170             else:
171                 self.mavlink.speed_setpoint_send(float(self.setpointX),
    float(self.setpointY))
172                 while(self.connection.out_waiting > 0):
173                     time.sleep(0.001)
174                 time.sleep(0.001)
175
176     def setSpeedY(self):
177         if self.actualMode == mouseController.MOUSE_MODE_SPEED:
178             self.setpointY = self.app.getEntry("speedY")
179             if self.setpointX is None or  self.setpointY is None :
180                 pass
181             else:
182                 self.mavlink.speed_setpoint_send(float(self.setpointX),
    float(self.setpointY))
183             while(self.connection.out_waiting > 0):
184                 time.sleep(0.001)
185             time.sleep(0.001)
186
187     def loadRoutine(self):
188         if self.actualMode == mouseController.MOUSE_MODE_AUTO_LOAD:
189             if (len(mouseRoutine.ROUTINE["duration"])>254 or len(
    mouseRoutine.ROUTINE["setpoint_x"])>254 or len(mouseRoutine.ROUTINE
    ["setpoint_y"])>254):
190                 raise ValueError("mouseRoutine too long")
191             if not (len(mouseRoutine.ROUTINE["duration"]) == len(
    mouseRoutine.ROUTINE["setpoint_x"]) == len(mouseRoutine.ROUTINE["
    setpoint_y"])):
192                 raise ValueError("not all components of mouseRoutine
    have the same lenght")
193
194
195             # TODO add verification on max speed and min speed
196
197
198             for i in tqdm(range(len(mouseRoutine.ROUTINE["duration"])))
    :
199                 self.mavlink.point_send(mouseRoutine.ROUTINE["duration"
    ][i],i,mouseRoutine.ROUTINE["setpoint_x"][i], mouseRoutine.ROUTINE[
    "setpoint_y"][i])
200                 stop = True
201                 while(self.connection.in_waiting>0 or stop):
202                     # Recive messages
203                     try:
204                         m = self.mavlink.parse_char(self.connection.
    read())
205                     except:
206                         pass
207                     if m:
208                         #print(m)
209                         if m.name == "POINT_LOADED":
```

```
210                             if m.point_id == i:
211                                 stop = False
212                             else:
213                                 raise Exception("ERROR LOADING DATA,
    wrong msg_id received")
214     def saveLog(self):
215         with open('log/log.txt', 'w+') as f:
216             for item in LOG:
217                 f.write("%s\n" % item)
218 
219     def runRoutine(self):
220         if self.actualMode == mouseController.MOUSE_MODE_AUTO_LOAD:
221             self.mavlink.mode_selection_send(mouseController.
    MOUSE_MODE_AUTO_RUN)
222             while(self.connection.out_waiting > 0):
223                 time.sleep(0.001)
224             time.sleep(0.001)
225 
226     def Prepare(self, app):
227         self.ax = []
228 
229         app.setTitle("Mouse treadmill GUI")
230         app.setFont(12)
231         row = 0
232         column = 0
233 
234         # Mode Selection
235         app.startFrame("modeSelection",row = row, column = column,
    colspan=4, rowspan = 1)
236         app.addLabel("optionModeLabel","Mode",0,0,1,1)
237         app.addRadioButton("optionMode",MODES[0],0,1,1,1)
238         app.addRadioButton("optionMode",MODES[1],0,2,1,1)
239         app.addRadioButton("optionMode",MODES[2],0,3,1,1)
240         app.setRadioButtonChangeFunction("optionMode",self.setMode)
241         app.stopFrame()
242         row = row+1
243 
244         # Speed entry
245         app.startFrame("speedEntry",row = row, column = column, colspan
    =4, rowspan=2)
246         app.addLabel("speedXLabel", "Speed X", 0,0,2,1)
247         app.addNumericEntry("speedX",1,0,2,2)
248         app.setEntry("speedX", 0.0)
249         app.setEntryChangeFunction("speedX", self.setSpeedX)
250         app.addLabel("speedYLabel", "Speed Y",0,2,2,1)
251         app.addNumericEntry("speedY",1,2,2,2)
252         app.setEntry("speedY", 0.0)
253         app.setEntryChangeFunction("speedY", self.setSpeedY)
254         app.stopFrame()
255         row = row+2
256 
257         # Reset plot button
258         app.startFrame("GUIButtons", row = row, column = column,
    colspan=2, rowspan=2)
259         self.app.addButton("RESET PLOTS", self.resetPlot, 0,0,1,1)
260         self.app.addButton("LOAD POINTS", self.loadRoutine, 1,0,1,1)
261         self.app.addButton("RUN ROUTINE", self.runRoutine,1,1,1,1 )
262         self.app.addButton("SAVE LOG",self.saveLog,0,1,1,1)
```

```
263             row = row+1
264 
265             # Sensor Status
266             app.startFrame("sensorStatus", row = row, column = 0)
267             self.app.addLabel("sensorStatus0",SENSOR_STATUS_MSG[0],
    0,0,1,1)
268             self.app.addLabel("sensorStatus1",SENSOR_STATUS_MSG[1],
    1,0,3,1)
269             self.app.addLabel("sensorStatus2",SENSOR_STATUS_MSG[2],
    2,0,3,1)
270             self.app.addLabel("sensorStatus3",SENSOR_STATUS_MSG[3],
    3,0,3,1)
271             self.app.addLabel("sensorStatus4",SENSOR_STATUS_MSG[4],
    4,0,3,1)
272             row = row+4
273 
274             # Real-time data plotting
275             app.startFrame("realTimePlot", row = row, column = column,
    colspan = 4, rowspan = 4)
276             self.fig = app.addPlotFig("plot",0,0,4,4, showNav = True )
277             self.ax.append(self.fig.add_subplot(311))
278             self.ax.append(self.fig.add_subplot(312))
279             self.ax.append(self.fig.add_subplot(313))
280             app.stopFrame()
281             row = row+4
282 
283 
284             # Add status bar
285             app.addStatusbar(fields = 2, side=None)
286             app.setStatusbar("Time: 0", 0)
287             app.setStatusbar("Mode: "+MODES[0], 1)
288 
289             # refresh funciton
290             app.setPollTime(100)
291             app.registerEvent(self.refreshGUI)
292 
293             # Window for sensor status
294             app.startSubWindow("sensorStatus")
295             app.addLabel("status", "SENSOR_X")
296             app.stopSubWindow()
297             app.openSubWindow("sensorStatus")
298 
299             return app
300         # Build and Start your application
301         def Start(self):
302             app = gui()
303 
304             self.app = app
305 
306             # Run the prebuild method that adds items to the UI
307             self.app = self.Prepare(self.app)
308             self.app.showAllSubWindow()
309             # Start appJar
310             self.app.go()
311 
312 if __name__ == '__main__':
313     print("=============================================================
    ")
```

```
314     print("Running GUI for mouse treadmill")
315     print("==========================================================
        ")
316
317     # Create an instance of your application
318     App = MyApplication()
319
320     # Start your app !
321     App.Start()
```

B.2 Routine example

```
1 ROUTINE = {
2     "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],
3     "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],
4     "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],
5 }
```

B.3 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 3
1.1 Pin Configuration 3
1.2 Package Outline Drawing 4
1.3 Assembly Drawings 5
1.4 PCB Assembly Recommendation 11
1.5 Reference Schematics 12
2.0 Electrical Specifications 14
2.1 Absolute Maximum Ratings 14
2.2 Recommended Operating Conditions 14
2.3 AC Electrical Specifications 15
2.4 DC Electrical Specifications 16
3.0 Serial Peripheral Interface (SPI) 18
4.0 Burst mode operation 22
5.0 SROM Download 23
6.0 Frame Capture 24
7.0 Power Up 26
8.0 Shutdown 27
9.0 Lift cut off calibration 28
10.0 Registers Table 29
11.0 Registers Description 30
12.0 Document Revision History 57

1.0 System Level Description

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

1.1 Pin Configuration

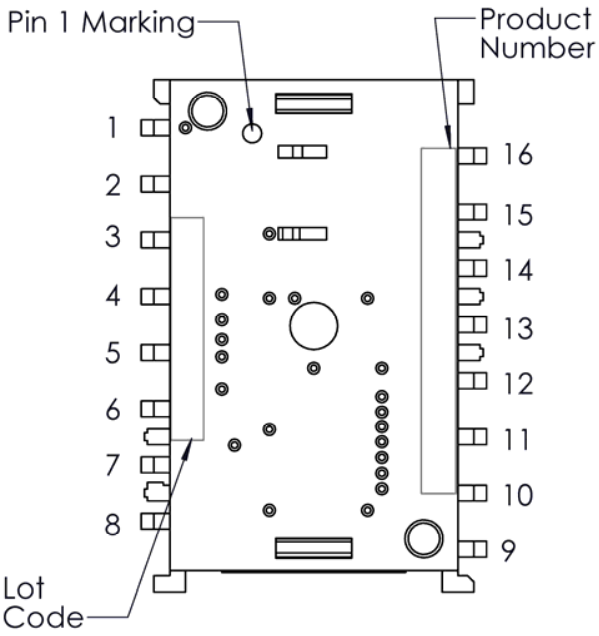


Figure 1. Device output pins

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)

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

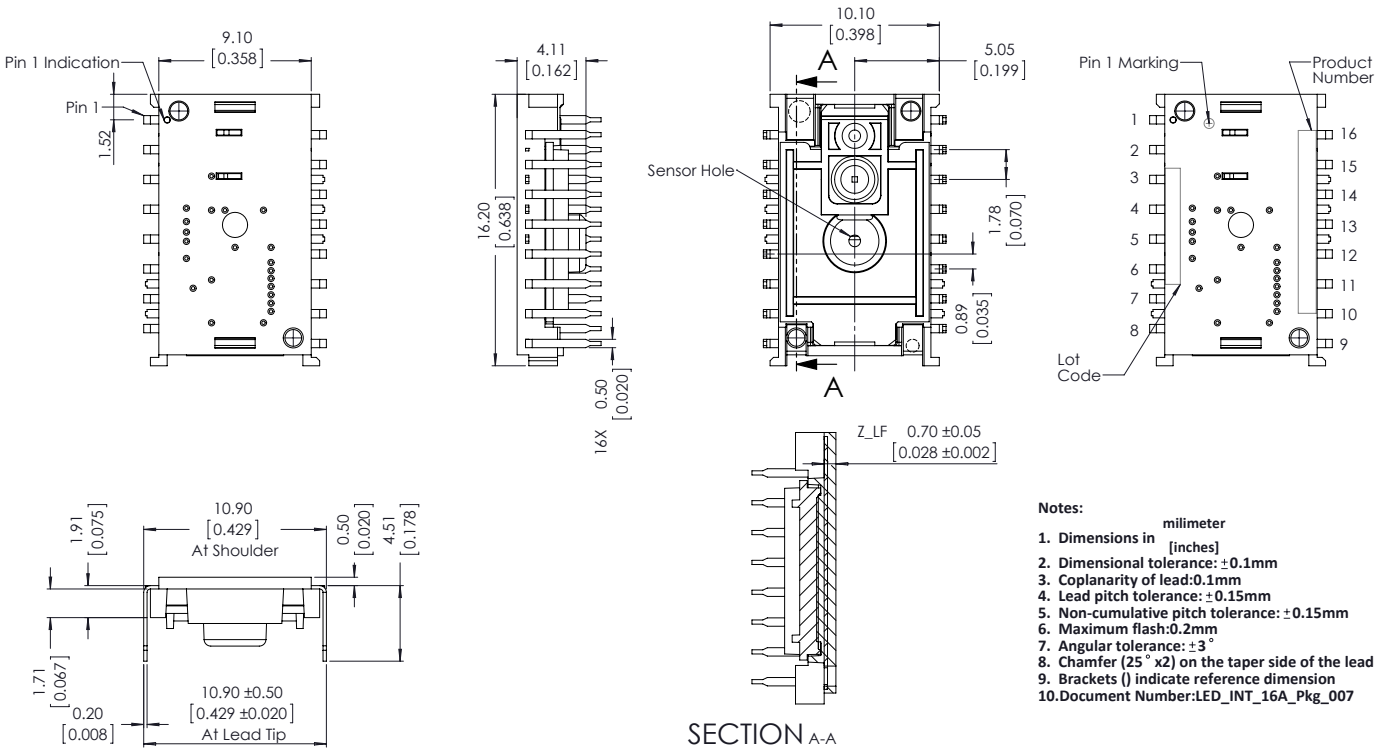


Figure 2. Package Outline Drawing

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

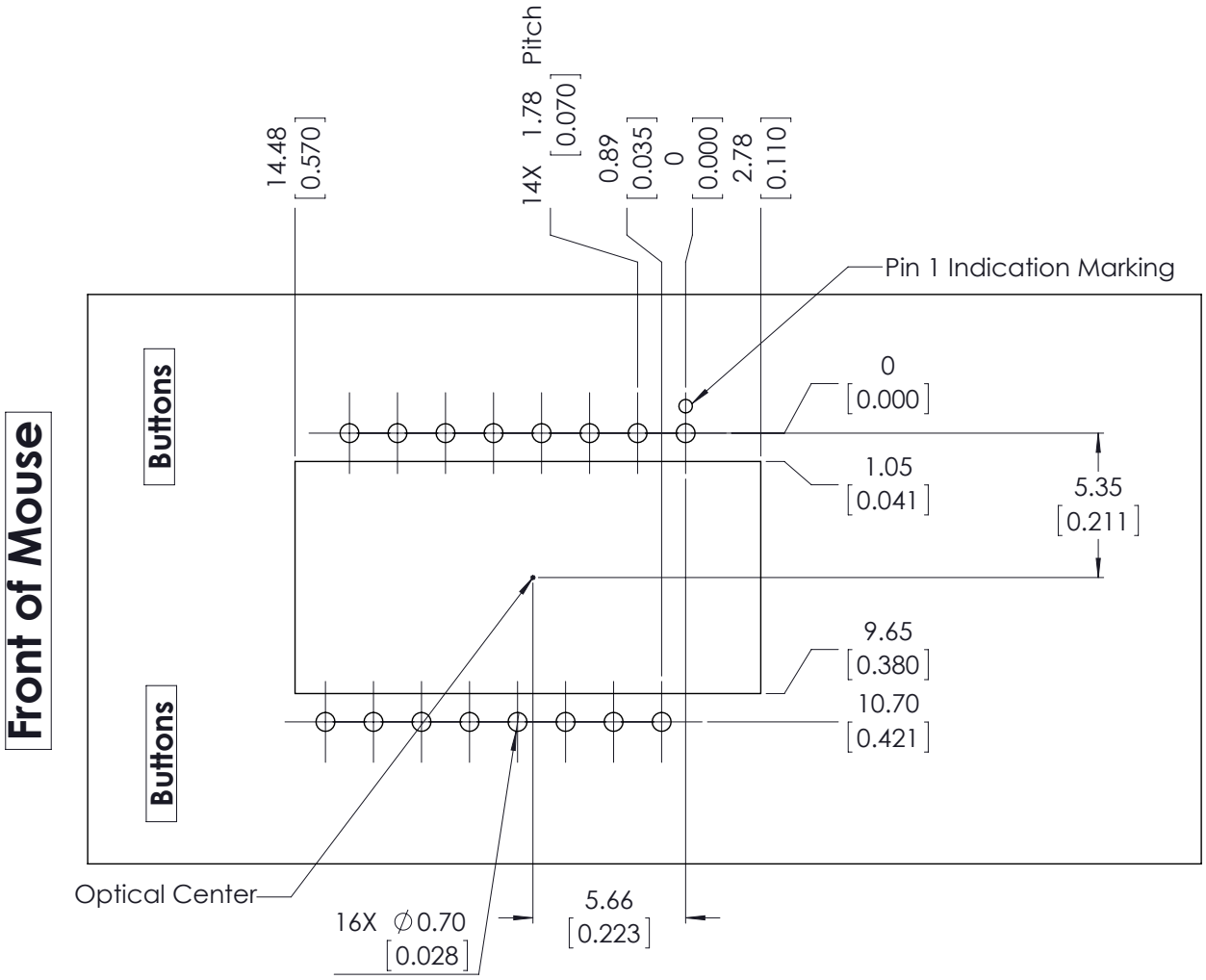


Figure 3. Recommended chip orientation, mechanical cutouts and spacing (Top View)

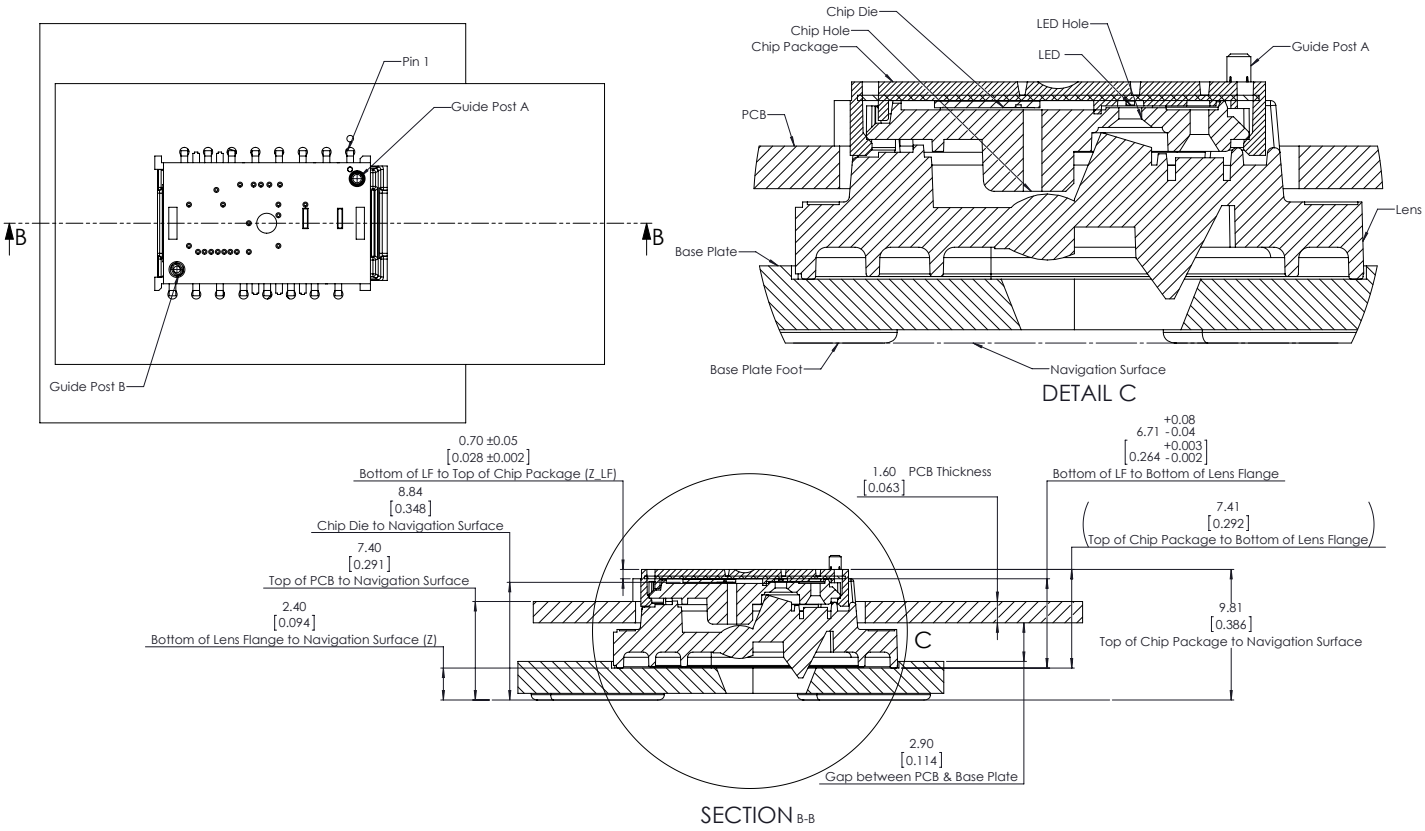


Figure 4. Assembly drawing of PMW3360DM-T2QU and distance from lens reference plane to tracking surface (Z)

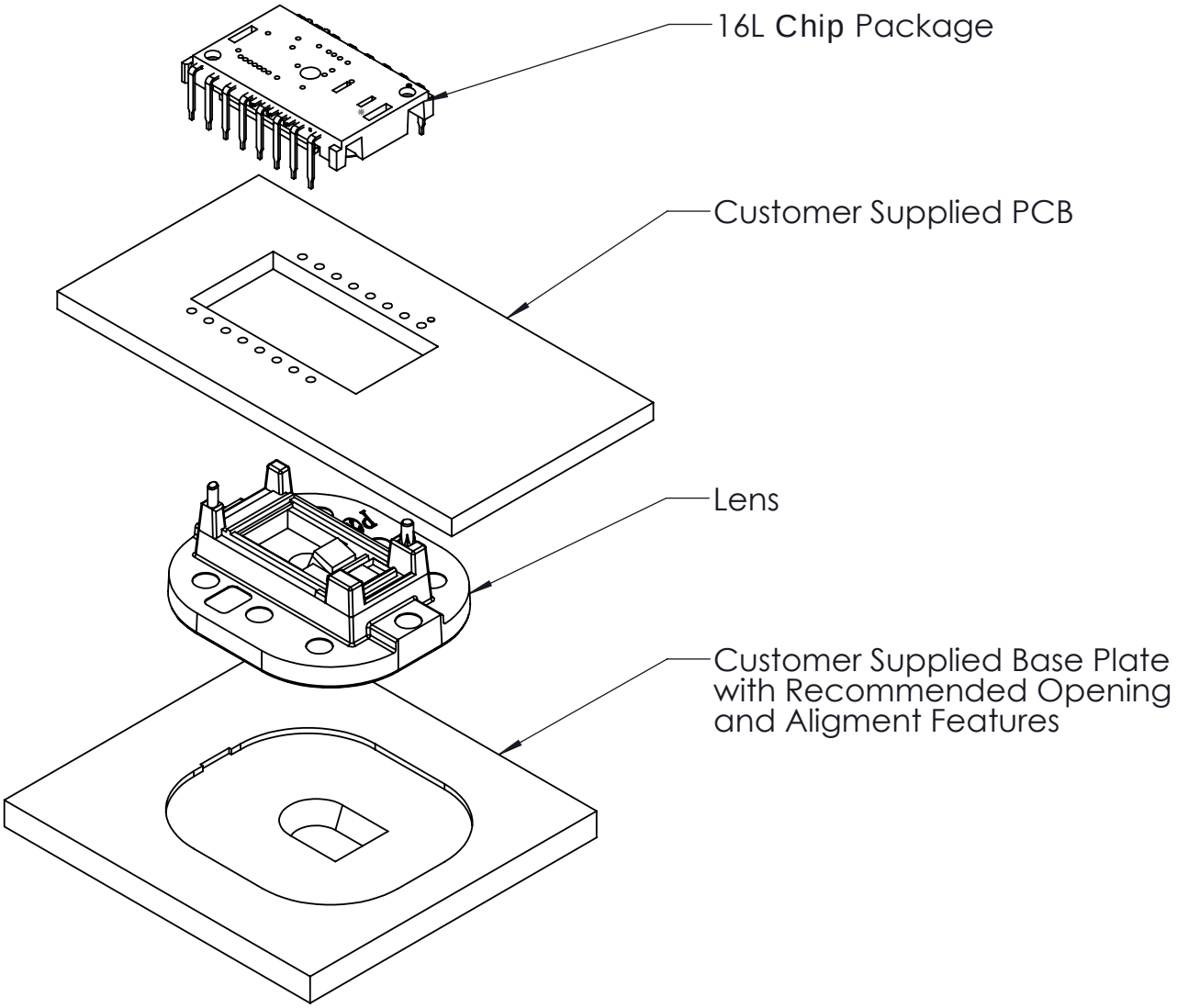


Figure 5. Exploded Assembly View

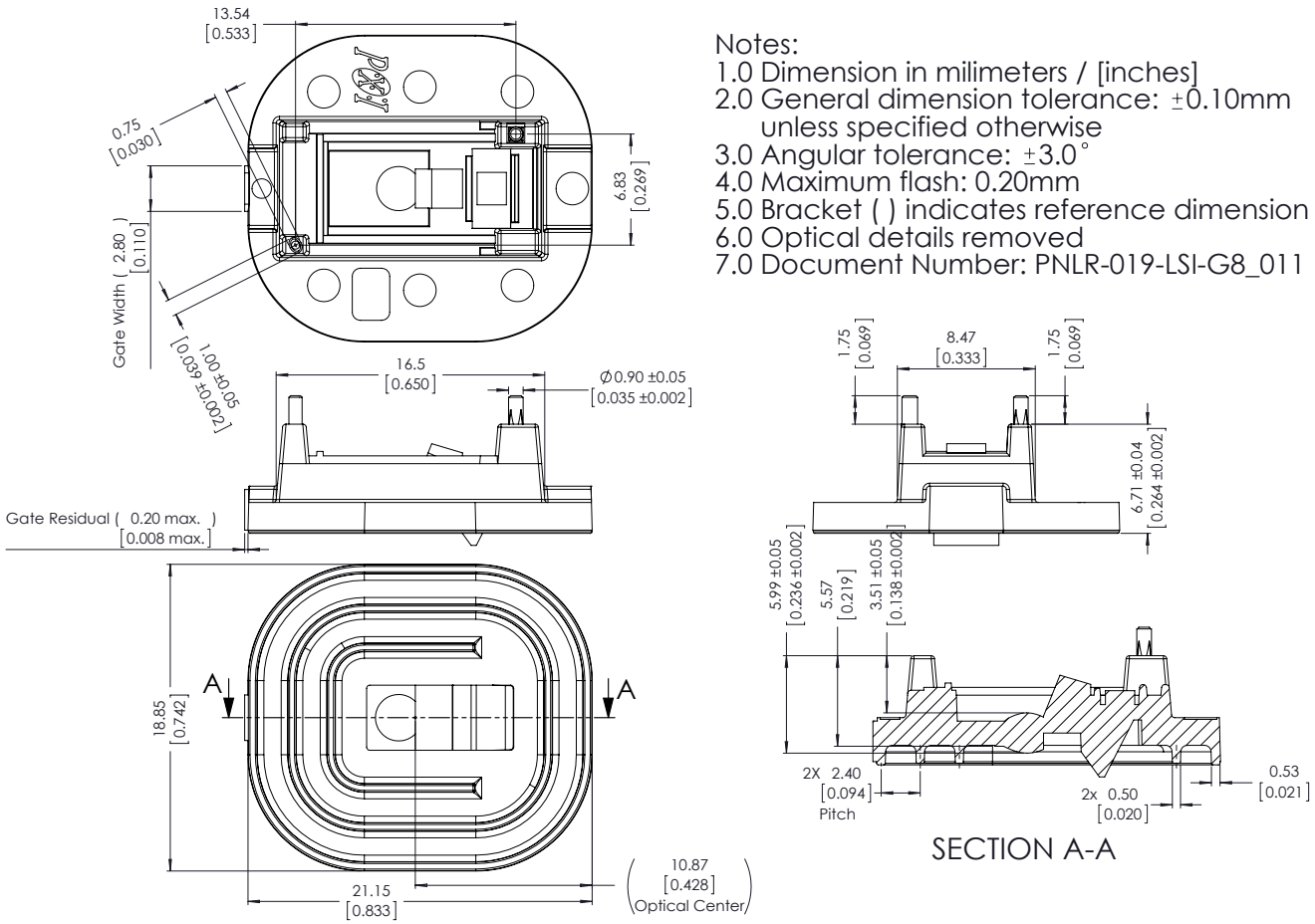


Figure 6. Lens Outline Drawing

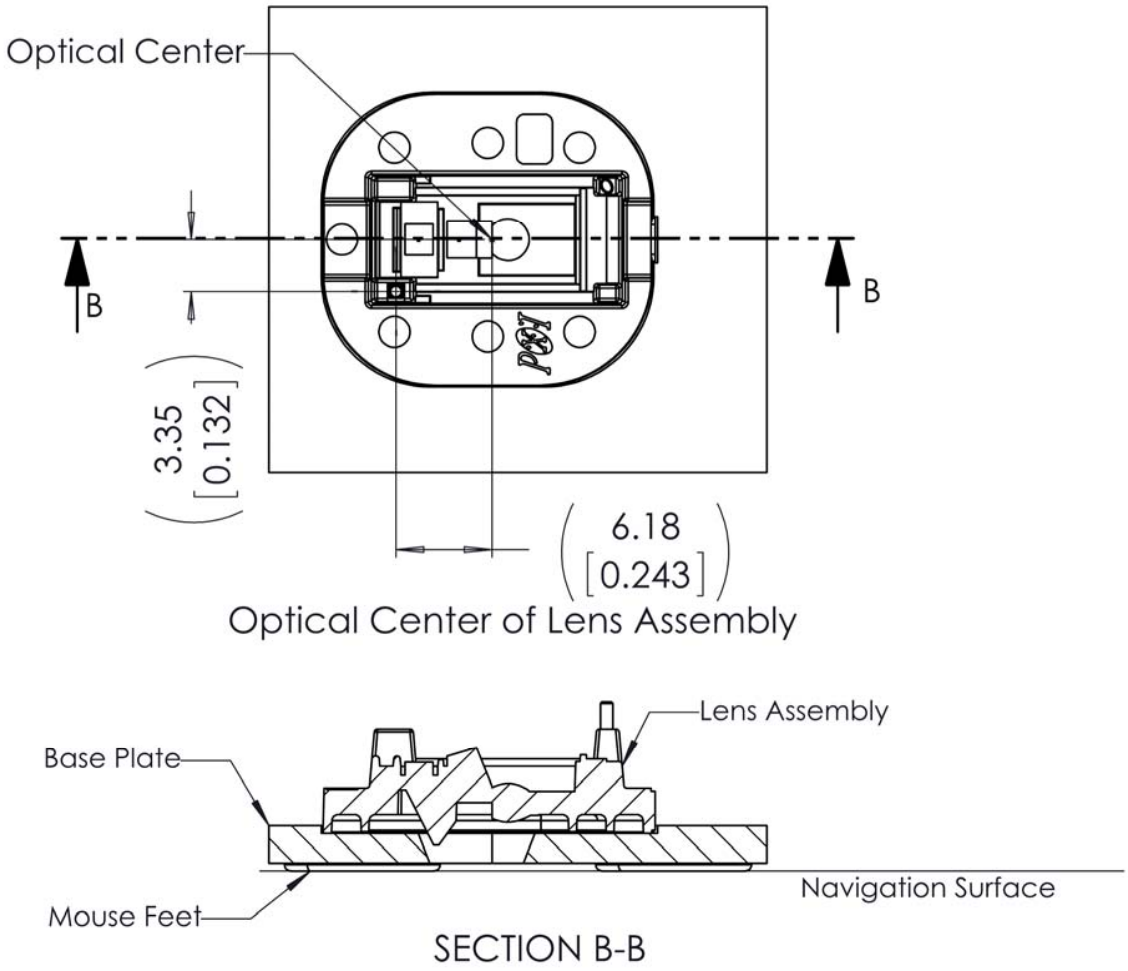


Figure 7. Cross section view of lens assembly

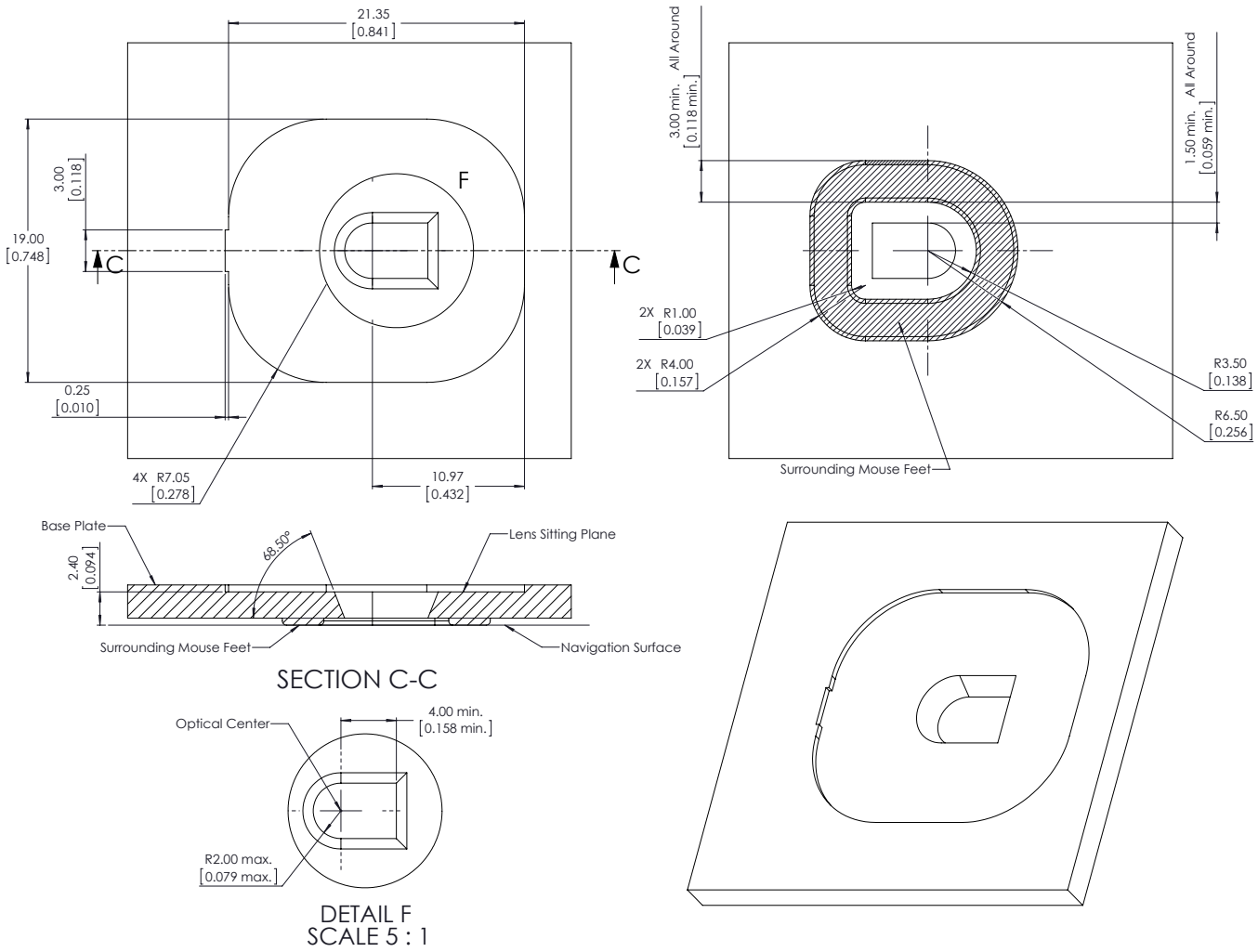


Figure 8. Recommended Base Plate Opening

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

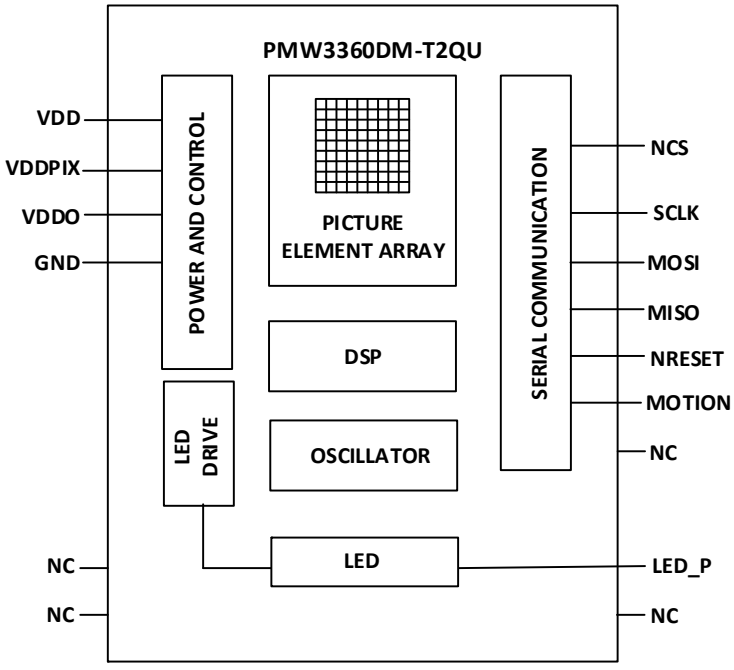


Figure 9. Block diagram of PMW3360DM-T2QU

1.5 Reference Schematics

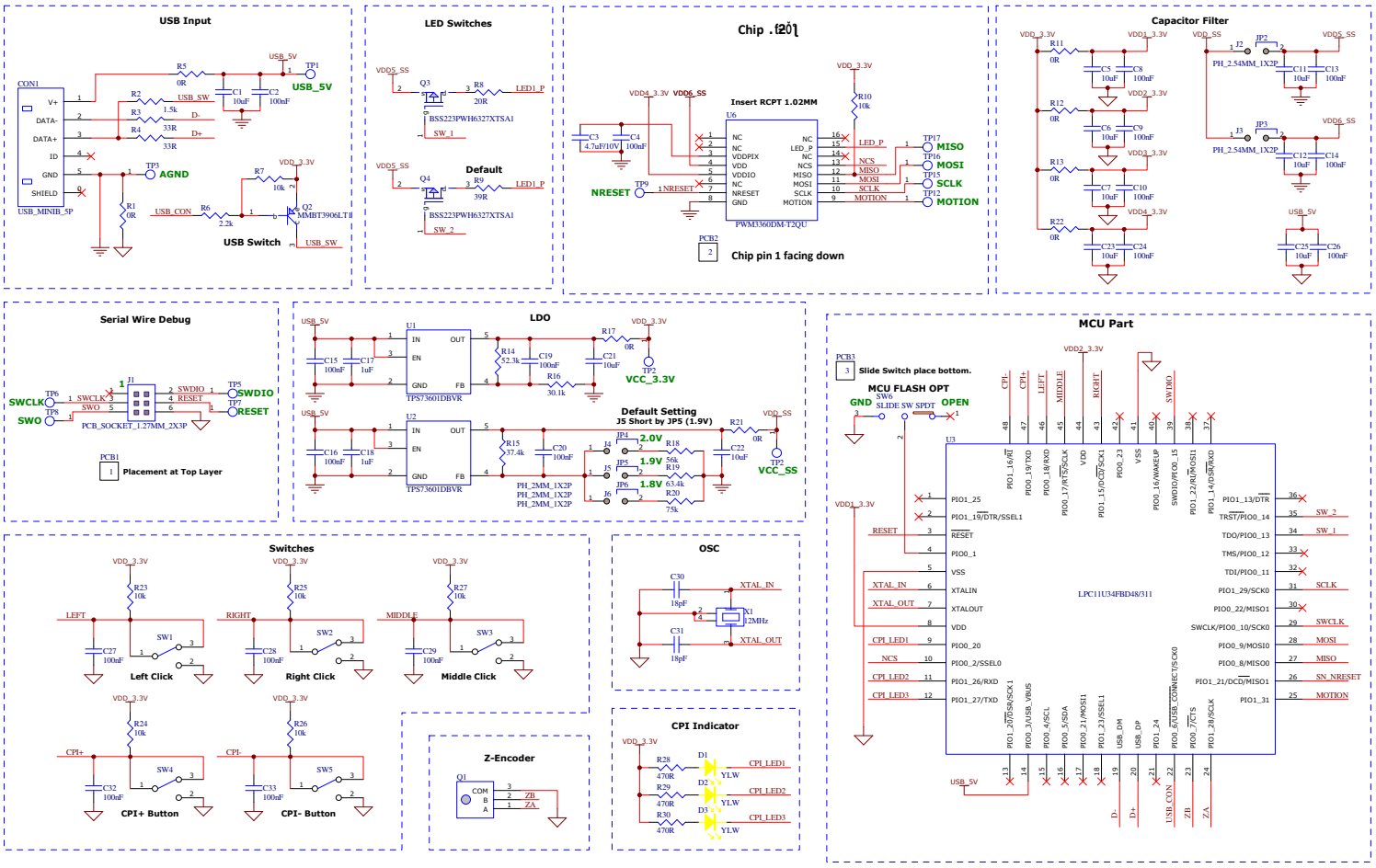


Figure 10. Schematic diagram for interface between PMW3360DM-T2QU and microcontroller on a wired solution

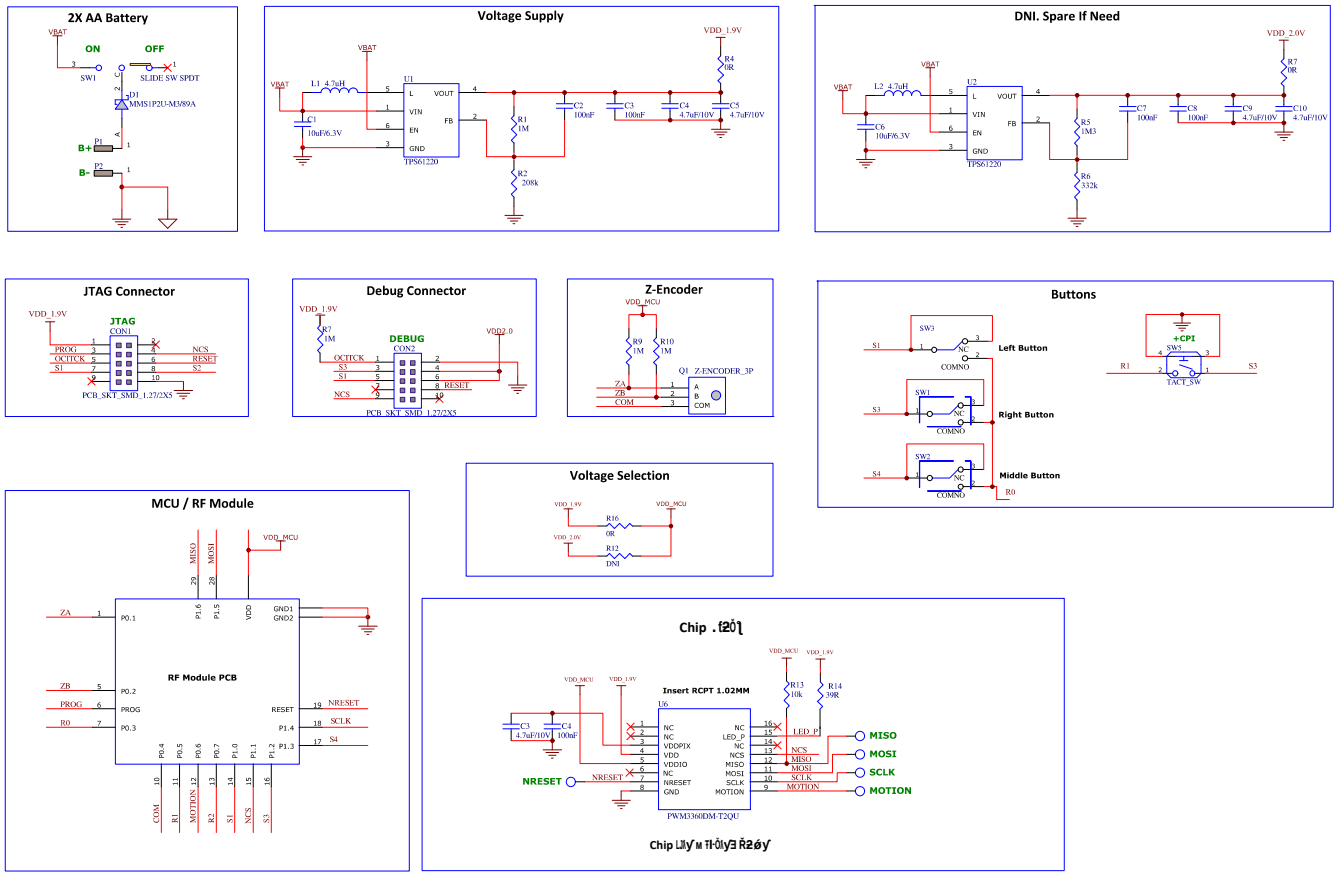


Figure 11. Schematic diagram for interface between PMW3360DM-T2QU and microcontroller on a wireless solution

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	$t_{MOT-RST}$	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 $t_{MOT-RST}$
MISO Rise Time	t_{r-MISO}		50		ns	C_L = 100pF
MISO Fall Time	t_{f-MISO}		50		ns	C_L = 100pF
MISO Delay After SCLK	$t_{DLY-MISO}$			90	ns	From SCLK falling edge to MISO data valid, no load conditions
MISO Hold Time	$t_{hold-MISO}$	200			ns	Data held until next falling SCLK edge
MOSI Hold Time	$t_{hold-MOSI}$	200			ns	Amount of time data is valid after SCLK rising edge
MOSI Setup Time	$t_{setup-MOSI}$	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	$t_{NCS-SCLK}$	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)	$t_{SCLK-NCS}$	120			ns	From last SCLK rising edge to NCS rising edge, for valid MISO data transfer
SCLK To NCS Inactive (For Write Operation)	$t_{SCLK-NCS}$	35			µs	From last SCLK rising edge to NCS rising edge, for valid MOSI data transfer
NCS To MISO High-Z	$t_{NCS-MISO}$			500	ns	From NCS rising edge to MISO high-Z state
MOTION Rise Time	$t_{r-MOTION}$		50		ns	C_L = 100pF
MOTION Fall Time	$t_{f-MOTION}$		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

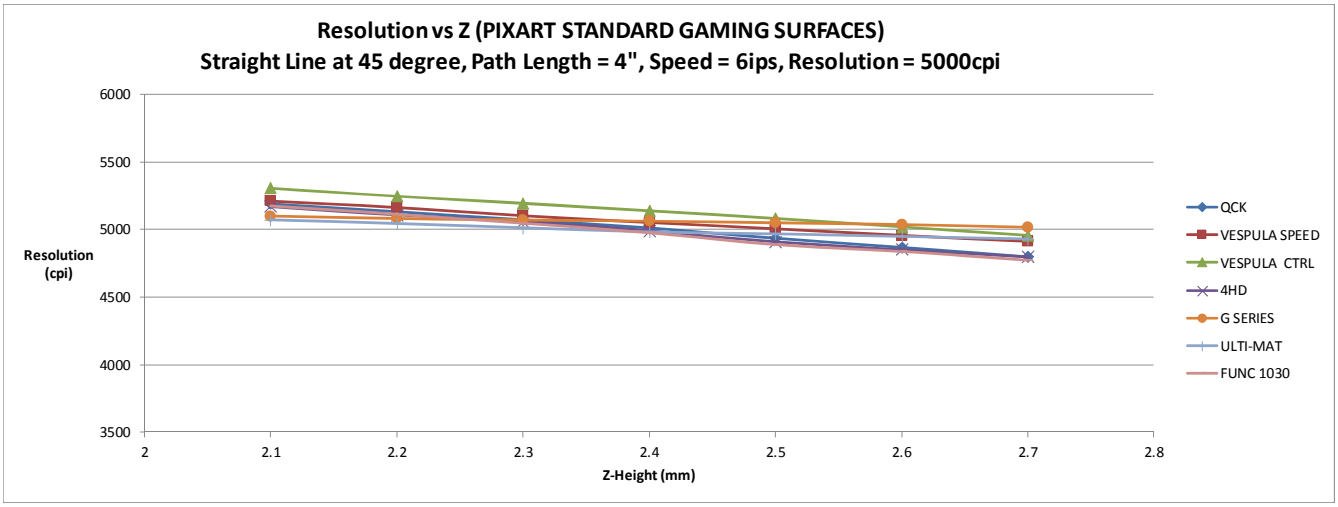


Figure 12 Mean Resolution vs. Z at default resolution at 5000cpi

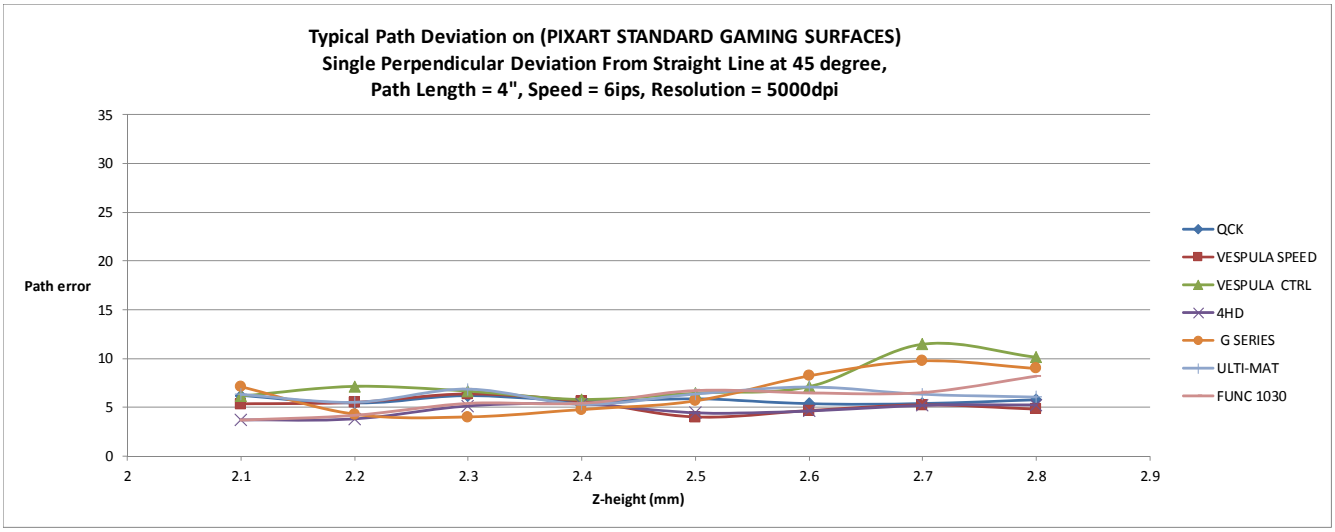


Figure 13 Path error vs. Z-height at default resolution at 5000cpi (mm)

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.

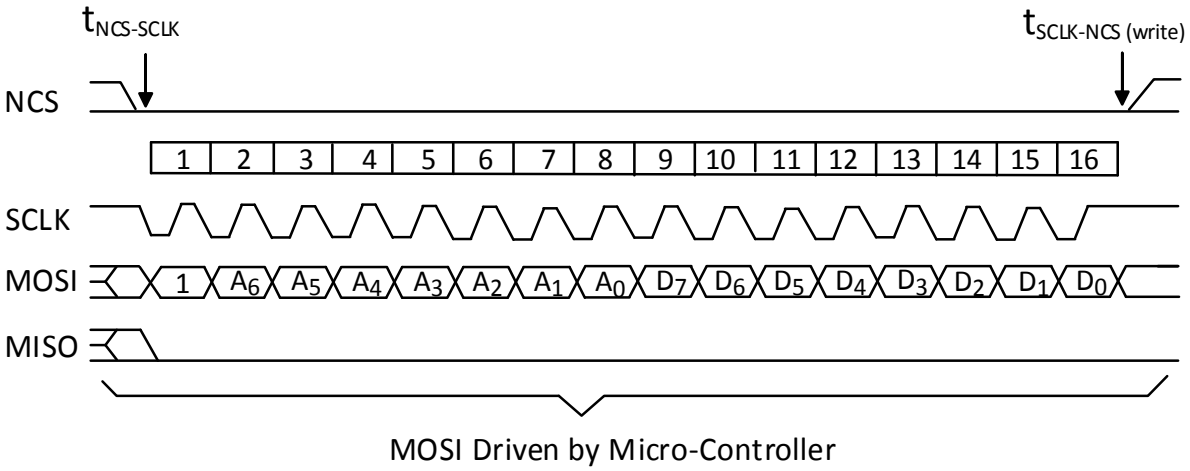


Figure 14. Write operation

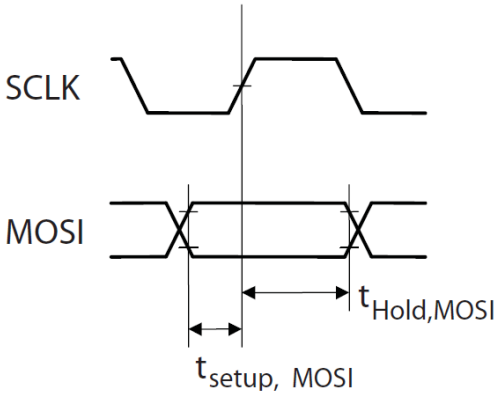


Figure 15. MOSI setup and hold time

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.

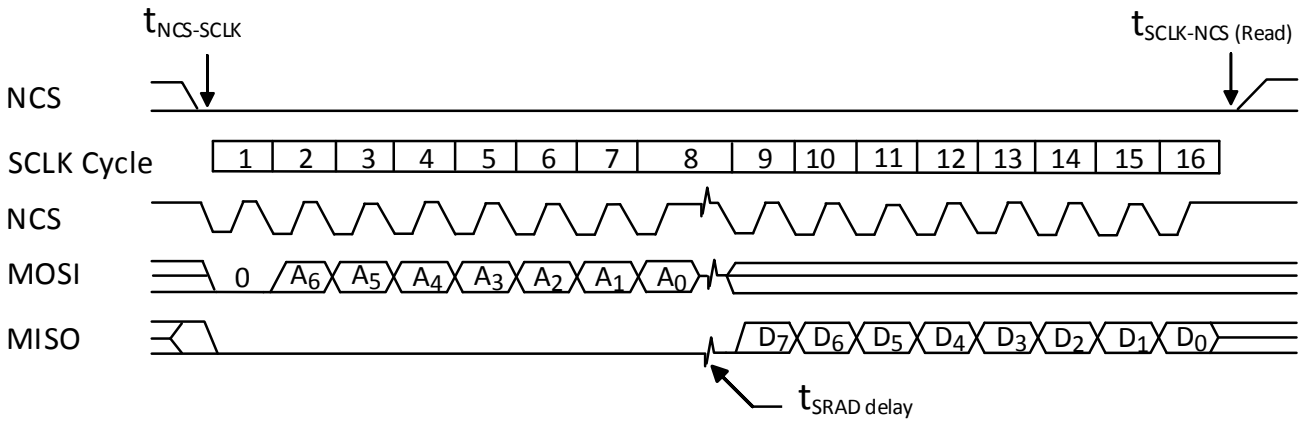


Figure 16. Read operation

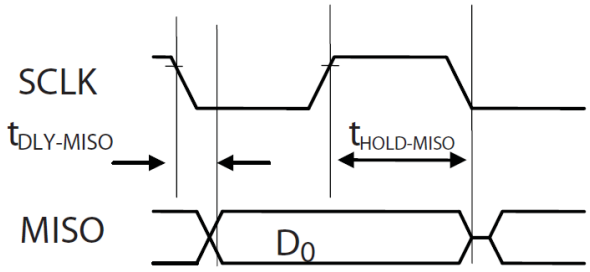


Figure 17. MISO Delay and hold time

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.

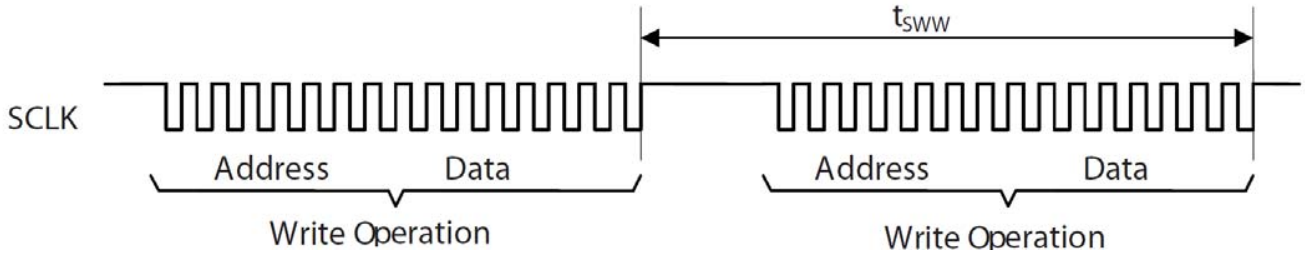


Figure 18. Timing between two write commands

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.

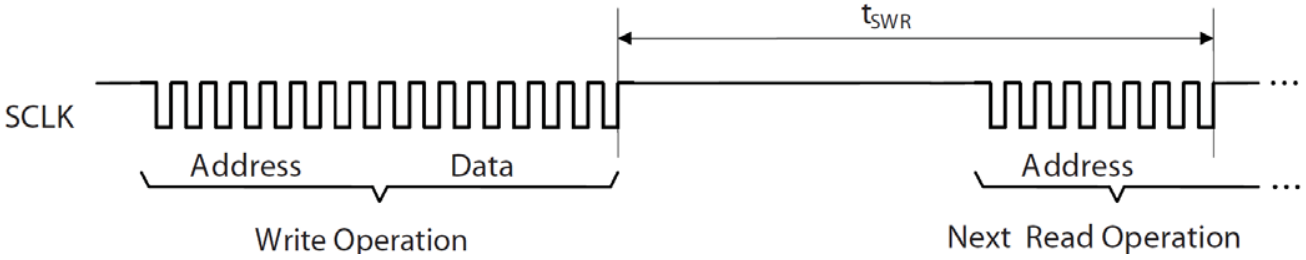


Figure 19. Timing between write and either write or subsequent read commands

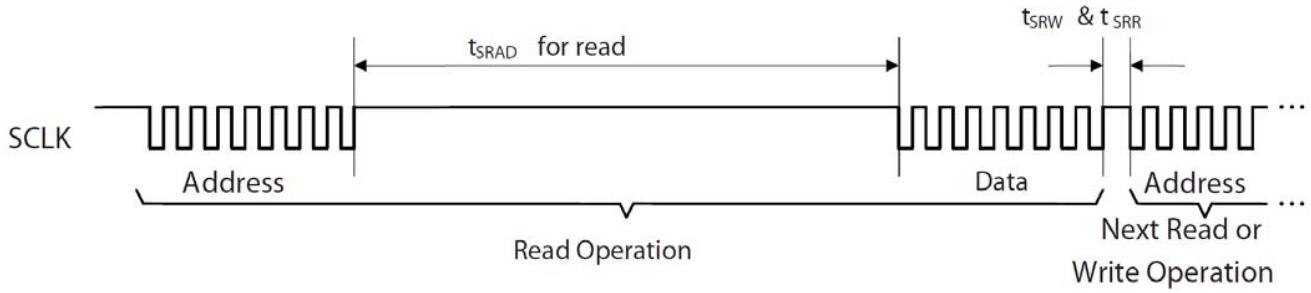


Figure 20. Timing between read and either write or subsequent read commands

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.

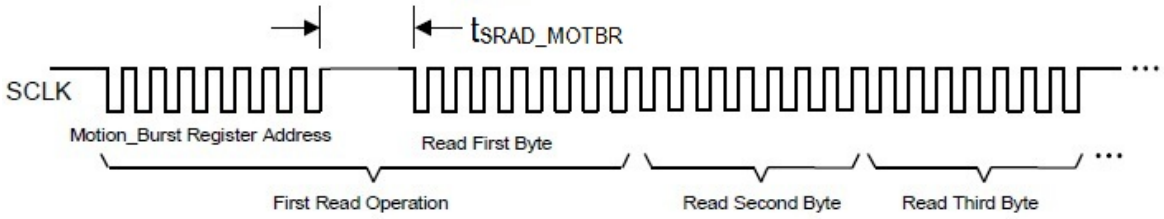


Figure 21. Motion Read sequence for step 3 to 5

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.

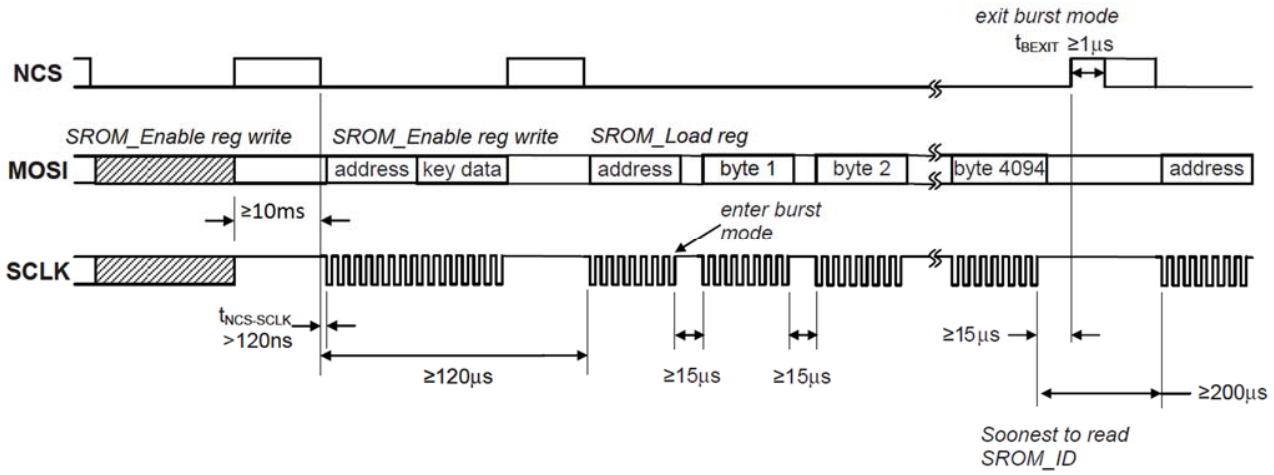


Figure 22. SROM Download Burst Mode

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.

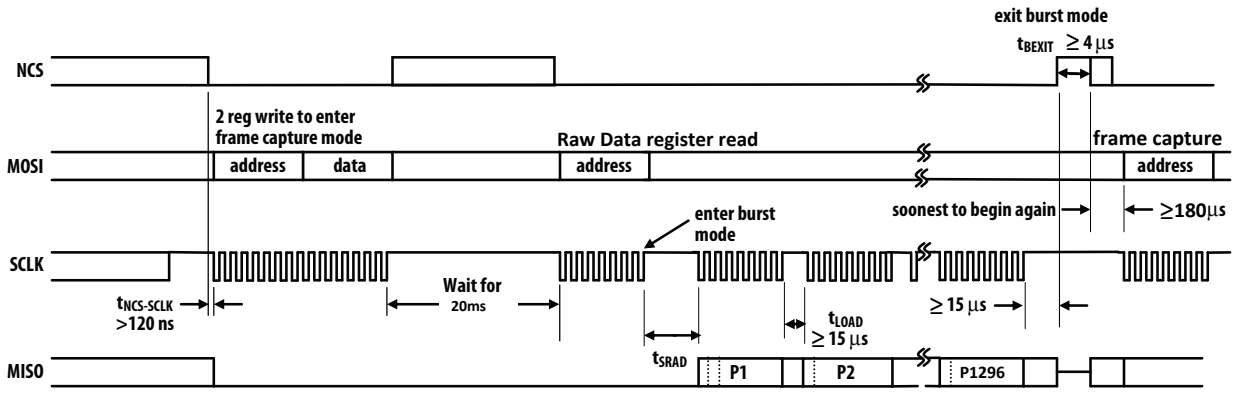
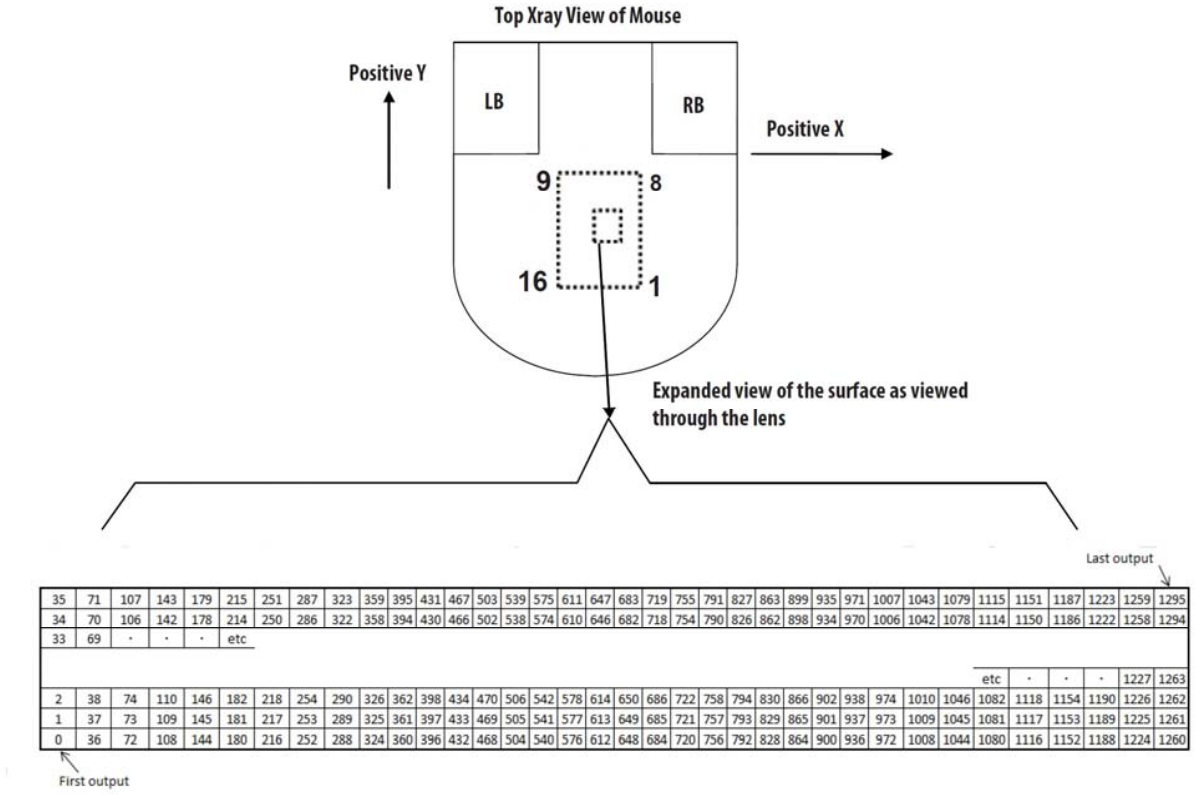


Figure 23. Frame Capture Burst Mode



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

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.							

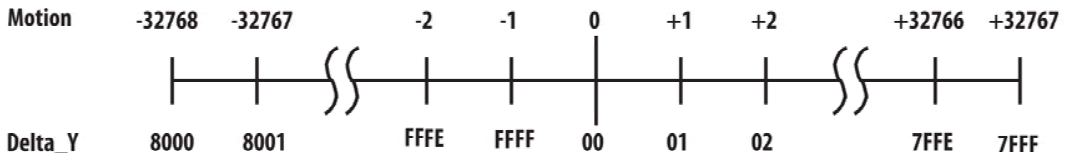
Register: 0x01								
Name: Revision_ID								
Bit	7	6	5	4	3	2	1	0
Field	RID_7	RID_6	RID_5	RID_4	RID_3	RID_2	RID_1	RID_0
	Reset Value: 0x01							
Access: R/W	Read Only							
Data Type:	8-bit unsigned integer							
Usage	This register contains the current IC revision, the revision of the permanent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

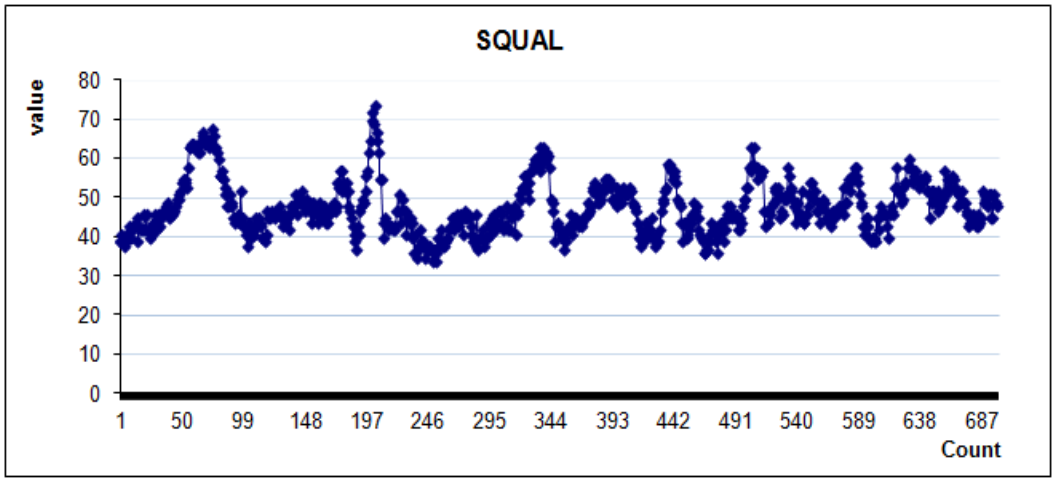


Figure 25. Average SQUAL on white paper

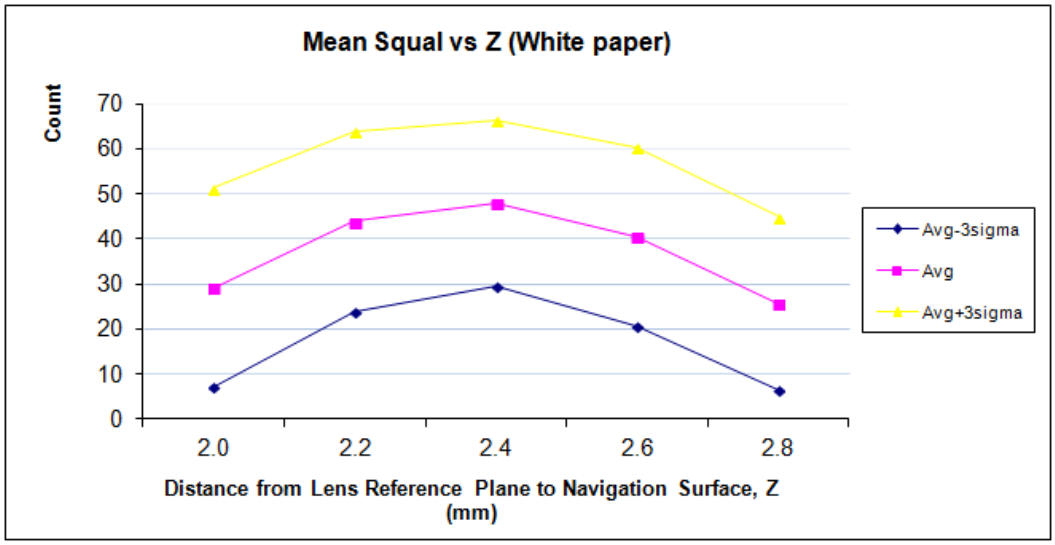


Figure 26. Mean SQUAL vs Z

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.							

<table>
<tr><th colspan="9">Register: 0x0D</th></tr>
<tr><td colspan="9">Name: Control</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>CTRL1$_7$</td><td>CTRL1$_6$</td><td>CTRL1$_5$</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td colspan="8">Reset Value: 0x02</td></tr>
<tr><td>Access: R/W</td><td colspan="8">Read Write</td></tr>
<tr><td>Data Type:</td><td colspan="8">8-bit unsigned integer</td></tr>
<tr><td>Usage</td><td colspan="8">This register defines programmable invert able of XY register scheme.
<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>CTRL1$_{[7:5]}$</td><td>000 - 0 degree
110 - 90 degree
011 – 180 degree
101 – 270 degree</td></tr>
<tr><td>Reserved$_{[4:0]}$</td><td>Reserved</td></tr>
</table>
Note: For CTRL1$_{[7:5]}$ please use 0 degree for best performance</td></tr>
</table>

<table>
<tr><th colspan="9">Register: 0x0F</th></tr>
<tr><td colspan="9">Name: Config1</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>RES_7</td><td>RES 1_6</td><td>RES_5</td><td>RES_4</td><td>RES_3</td><td>RES_2</td><td>RES_1</td><td>RES_0</td></tr>
<tr><td colspan="8">Reset Value: 0x31</td></tr>
<tr><td>Access: R/W</td><td colspan="8">Read/ Write</td></tr>
<tr><td>Data Type:</td><td colspan="8">Bit Field</td></tr>
<tr><td>Usage</td><td colspan="8">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).
<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>RES[7:0]</td><td>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)</td></tr>
</table></td></tr>
</table>

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.							

Register: 0x65								
Name: LiftCuttoff_Tune2								
Bit	7	6	5	4	3	2	1	0
Field	Reserved	$RPTH_6$	$RPTH_5$	$RPTH_4$	$RPTH_3$	$RPTH_3$	$RPTH_1$	$RPTH_0$
	Reset Value:0x00							
Access: R/W	Read Only							
Data Type:	Bit Field							
Usage	This register provides Lift cut off calibration related readout registers. See the Lift cut off calibration section for more details. **Field Name** — **Description** RPTH[6:0] — 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.							

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