

Lab 3: MEMS Accelerometer, Timer and Interrupts

ECSE 426 – Microprocessor Systems



McGill University

Winter 2015

Group 4 : Sidney Ng 260507001

Meng Yin Tao 260480207

Due March 16th 2015

Contents

[List of Figures 2](#_Toc414290405)

[List of Tables 2](#_Toc414290406)

[1. Abstract 3](#_Toc414290407)

[2. Problem Statement 3](#_Toc414290408)

[3. Theory and Hypothesis 3](#_Toc414290409)

[3.1. Accelerometer 3](#_Toc414290410)

[3.1.1. Orientation 3](#_Toc414290411)

[3.1.2. Acceleration 4](#_Toc414290412)

[3.2. Data Calibration and Filtering 4](#_Toc414290413)

[3.2.1. Data Normalization and Calibration 4](#_Toc414290414)

[3.2.2. Data Filtering 5](#_Toc414290415)

[3.2.3. Data Interpretation 5](#_Toc414290416)

[3.3. External Keypad 6](#_Toc414290417)

[3.3.1. Circuit Layout 6](#_Toc414290418)

[3.3.2. Data Acquisition 6](#_Toc414290419)

[3.3.3. Handling Key Bouncing 7](#_Toc414290420)

[3.4. External 7-Segment Display 7](#_Toc414290421)

[3.4.1. Circuit Layout 7](#_Toc414290422)

[3.4.2. Data Display 7](#_Toc414290423)

[3.5. Timing 8](#_Toc414290424)

[3.5.1. Timing Based on Sample Rate 8](#_Toc414290425)

[3.5.2. Timing Based on Hardware Timer 8](#_Toc414290426)

[4. Implementation 8](#_Toc414290427)

[4.1. Component Configuration and Initialization 8](#_Toc414290428)

[4.1.1. GPIO Configuration 8](#_Toc414290429)

[4.1.2. Accelerometer Configuration, Accelerometer Interrupt and Interrupt Handler 9](#_Toc414290430)

[4.1.3. Timer Configuration, Timer Interrupt and Interrupt Handler 9](#_Toc414290431)

[4.2. Collect User Input 9](#_Toc414290432)

[4.2.1. Initialize Keypad 9](#_Toc414290433)

[4.2.2. Return User Input 9](#_Toc414290434)

[4.2.3. Improvement on Keypad 9](#_Toc414290435)

[4.3. Tilt Angle Calculation 10](#_Toc414290436)

[4.3.1 Data Sampling 10](#_Toc414290437)

[4.3.2. Data Processing 10](#_Toc414290438)

[4.3.3. Data Comparison 11](#_Toc414290439)

[4.4. Result Display 11](#_Toc414290440)

[4.4.1. Visual Feedback 11](#_Toc414290441)

[4.4.2. Display Current Angle 11](#_Toc414290442)

[4.5 Continuous Process 11](#_Toc414290443)

[5. Testing and Observation 11](#_Toc414290444)

[5.1. Accelerometer Calibration 11](#_Toc414290445)

[5.2. Kalman Filter 11](#_Toc414290446)

[6. Conclusion 12](#_Toc414290447)

[Reference 12](#_Toc414290448)

[Appendix A – Calibration Data 13](#_Toc414290449)

# List of Figures

[Figure 1: Board Axes and Tilt 4](#_Toc414292265)

[Figure 2: Keypad Layout 6](file:///\\campus.mcgill.ca\emf\CPE\sng20\Desktop\Lab3%20Report%20save%20version.docx#_Toc414292266)

[Figure 3: 7-Segment Display Layout 7](#_Toc414292267)

[Figure 4: Timer Waveform 8](#_Toc414292268)

[Figure 5: Kalman Filter q = 0.0025, r = 5 13](file:///\\campus.mcgill.ca\emf\CPE\sng20\Desktop\Lab3%20Report%20save%20version.docx#_Toc414292269)

[Figure 6: Kalman Filter q = 0.025, r = 5 13](#_Toc414292270)

[Figure 7: Kalman Filter q = 0.25, r = 5 14](#_Toc414292271)

[Figure 8: Kalman Filter q = 2.5 r = 5 14](#_Toc414292272)

[Figure 9: Kalman Filter q = 25, r = 5 15](#_Toc414292273)

[Figure 10: Kalman Filter q = 250, r = 5 15](#_Toc414292274)

[Figure 11: Kalman Filter q = 2500 r = 5 16](#_Toc414292275)

# List of Tables

[Table 1: GPIO Configuration for 7-Segment Display 10](#_Toc414299291)

[Table 2: GPIO Configuration for LEDs 10](#_Toc414299292)

[Table 3: LIS302DL MEMs Sensor Configuration 10](#_Toc414299293)

[Table 4: GPIO Configuration for Mems sensor 11](#_Toc414299294)

[Table 5: EXTI Configuration 11](#_Toc414299295)

[Table 6: NVIC Configuration 11](#_Toc414299296)

[Table 7: TIM3 Configuration 12](#_Toc414299297)

[Table 8: Normalization Matrix 13](#_Toc414299298)

# 1. Abstract

This experiment implements a system which can determine the orientation of the STM32F4 Discovery board using its built-in accelerometer. The data a calibrated, filtered and converted. Once the position is found in degrees, it is compared to the user’s target angle. The user will use a 4x4 keypad to input the desired angles. A 7-segment display is used to show how the board should be oriented in order to achieve the desired angle. Finally, when the board is within five degrees of the target angle, the 7-segment display will show the current angle with four degrees of accuracy.

# 2. Problem Statement

The goal of this experiment is to create a system that detects the orientation of the STM32F4 Discovery board. The rotation of the board around its axes is detected using the built-in accelerometer. However, the accelerometer cannot determine the yaw (rotation around the board’s z-axis) since it does not result in a change in acceleration. Only the roll (rotation around the board’s x-axis) and pitch (rotation around the board’s y-axis) can be found using the accelerometer. When the accelerometer’s raw data becomes available, the processor will calibrate it using a pre-built calibration matrix and filter it using a Kalman filter whose parameters are experimentally defined. In order to make sense of the data, it needs to be converted into an angle in degrees. This conversion is done using the arctan function available in the math library. Once the angle is available, it can be compared against the user input. The user is prompt to enter a target angle for both the roll and the pitch using a 4x4 external keypad. As the user tilts the board, an animated indication on the 7-segment display will help the user directing the board toward the target angle. Once the user is within five degree of the target roll angle, the 7-segment display will show the current angle captured by the accelerometer. This measured angle is expected to be within four degrees of accuracy compared to the actual angle.

# 3. Theory and Hypothesis

## 3.1. Accelerometer

### 3.1.1. Acceleration

The accelerometer measures acceleration along the X, Y, and Z axes. For the use of tilt measurement applications, the board is kept relatively still. As a result, the force of acceleration acting on the sensor with the most force is gravity. As a result, the three acceleration values along the X, Y, and Z axes are components of a 1g force.

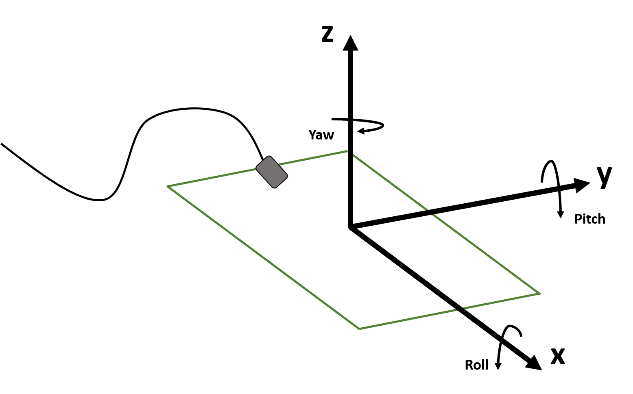


Figure 1: Board Axes and Tilt

### 3.1.2. Orientation

The tilt of the board can be measure for two angles, the pitch and the roll. The pitch is the angle of the board that occurs when there is rotation around the Y-axis, as seen in figure 1. It is measured by the angle the X-axis of the board makes with the horizontal plane. As shown in figure 1, the pitch is 0°. If the X-axis were in the position of the Z-axis in figure 1, then the pitch is 90°. The equation for pitch is given by:

(1)

The roll is the angle of the board that occurs when there is rotation around the X-axis, as seen in figure 1. It is measured by the angle the Y-axis of the board makes with the horizontal plane. As shown in figure 1, the roll is 0°. If the Y-axis were in the position of the Z-axis in figure 1, then the roll is 90°. The equation for roll is given by:

(2)

The third tilt is the yaw, the rotation around the Z-axis. It cannot be measured due to the nature of the application. For the raw there is no variation from the horizontal plane. Since the sensor depends on measuring different forces of gravity, the yaw cannot be calculated without the aid of other sensors.

## 3.2. Data Calibration and Filtering

The raw data acquired from the accelerometer cannot be used direction in equations (1) and (2). It is noisy and also most likely misaligned with the actual axes of the board. As a result, normalization and filtering must be performed on the data.

### 3.2.1. Data Normalization and Calibration

During the process of manufacturing the board, the MEMS accelerometer is not perfectly soldered to the board with its axes aligned. As a result, any acquired data must be normalized before using the data. The equation converting the raw data to the normalized acceleration is expressed as:

(1)

The [Am] matrix represents the misalignment between the accelerometer axes and the device’s axes, ASCirepresents the sensitivity (scaling) and AOSi is the zero-g level offset. Equation (1) is equivalent to:

(2)

(3)

The matrix X in equation (3) is equivalent to matrix [ACCij] in equation (2). The matrix X is the normalization vector multiplied to the raw data vector w in equation (2) to obtain the normalized data in matrix Y in equation (3).

### 3.2.2 Calibration

Since the soldering of the sensor is different on every board, the normalization values (ACCij) must be calibrated. To acquire these values, the least square method is used to solve equation (3) for the matrix X:

(4)

For the values of matrices w and Y, the raw data values and corresponding normalized values are used from the positions Xup, Xdown, Y­up, Ydown, Z­up, Zdown. Z­up is when the board is facing up in the horizontal plane as in Figure 1. Y­up is when the Y axis of the board is where the Z axis is in Figure 1. Xup is when the X axis of the board is where the Z axis is in Figure 1. The down counterparts are when the axes are in the opposite direction of the Z axis in Figure 1. The Y matrix is equal to:

(5)

The first row of the matrix corresponds to Zup, the second to Zdown, the third to Yup, the fourth to Ydown, the fifth to Xup, and the sixth to Xdown. The first column holds X values, the second Y values, and the third holds Z values. The w matrix follows the same format of the Y matrix. However, it has an additional fourth column of 1’s.

### 3.2.3. Data Filtering

--- Kalman filter (what’s the purpose of filtering?)

### 3.2.4. Data Interpretation

--- Normalize acceleration

--- Convert from acceleration to angle in radian (need equation)

--- Alternative way of getting the angle (lookup table)

## 3.3. External Keypad

### 3.3.1. Circuit Layout

The keypad used for this experiment is composed of four columns and four rows. The pin associated to each of the columns and rows where experimentally found using a multimeter. Each pin on the keypad is connected to a column or a row. When a key is pressed, a connection is made between a column and a row. Therefore, we expected each pin to be disconnected when no key is pressed, and two pins to be shorted together when a key is pressed. Using a multimeter, we found the pair of pins associated to the pressing of each key. The results are illustrated in Figure 2. The first four pins were connected to the columns from left to right respectively. The last four pins were connected to the rows from up to down respectively.

### 3.3.2. Data Acquisition

Figure 2: Keypad Layout

In order to know which pin is pressed, a scanning mechanism can be used. The first step is to find the row on which a pressed key located by setting the column pins high and watching the row pins. When no key is pressed, all row pins will remain unaffected. To achieve better result and avoid undefined states, the row pins can be pulled down. Once a key is pressed, a connection will be made between its corresponding row and column. By watching the change in voltage on the row pins, one can find the row corresponding to the pressed key. Figure 2 shows this step, where the column pins (pin 1 to 4) are set high (green) and when any key in the first row is pressed, pin 5 is set high (turn green). For pin 6, if its value toggled from low to high, a button is pressed on the second row. Similarly, the other rows can be determined in the same manner. The second step is to determine the column on which the key is located by setting the row pins high and watching the column similarly to the first step. Since the coordinates and the keys are mapped one to one, by knowing the row and column, the pressed key can be resolved. This methodology is only valid under the assumption that a single key is pressed at any time. If more than one key is pressed, the result will depend on the order in which rows and columns are scanned.

### 3.3.3. Handling Key Bouncing

The frequency at which the pins are scanned will affect the values read. If the scanning rate is much higher than the speed a user can input, the same input will be captured multiple times. To handle such situation, we can simply ignore consecutive identical inputs. However, an issue arises when user willingly inputs two identical values consecutively. To resolve that issue, a NO\_INPUT signal can be introduced to catch the time when the key is released. The duration of the NO\_INPUT must be defined to handle key bouncing, where the connection opens up without the user releasing the key. A NO\_INPUT signal is considered as valid only after is has been observed for long enough.

## 3.4. External 7-Segment Display

### 3.4.1. Circuit Layout

The 7-segment display used for this experiment is composed of four digits, colon symbol after the second digit and degree symbol after the third digit. Each digit is composed of seven segments and a decimal point. The signal lines for the segments and the decimal point are shared between the digits. In order to control which digit should be updated, a select line is used for each of them. Similarly, there is a select line for the colon and the degree symbol. The colon symbol was not used in this experiment, therefore will be omitted from now on. As shown in Figure 3, the pins from the display cannot be directly connected to the board. For the segment pins, a resistor must be inserted to reduce the voltage at the input. The select pins must be controlled through the mean of a transistor. The transistor acts as a switch, when turned on, the select pin is connected to ground. To keep the Figure 3 concise, only one resistor and one transistor are shown.

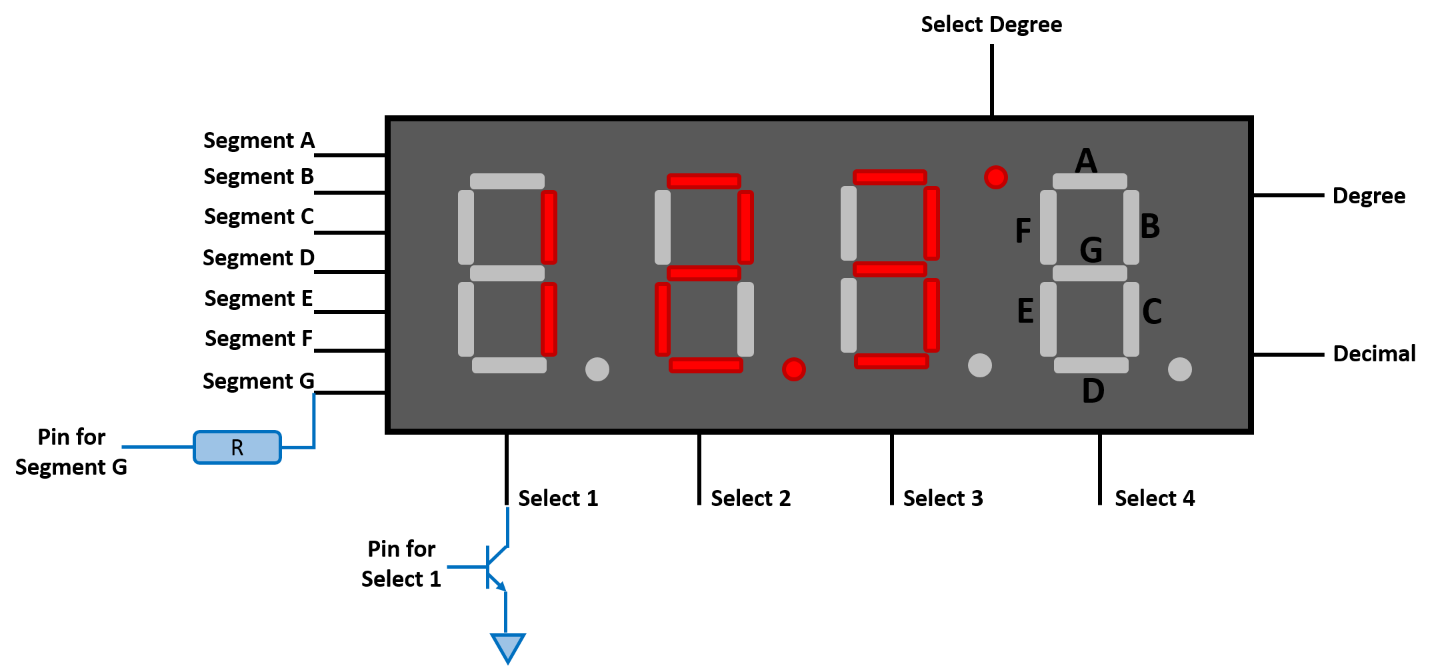


Figure 3: 7-Segment Display Layout

### 3.4.2. Data Display

As mentioned earlier, only one digit can be displayed at a time. To achieve the effect of all digits being turned on, the digits need to be refreshed quickly so the user will not see the flickering effect. To display values on a given digit, the corresponding select pin must be grounded by turning the transistor on. Once the digit is selected, the segments can be turned on to display the desired pattern. The segments must be held on for a given time period before releasing it and moving on to the next digit. If this time period is too short, the light will be faint. On the other hand, if this period is too long, there might not be enough time to refresh the digits before the flickering effect becomes noticeable. The timing of the 7-segment display can be determined experimentally with various frequencies.

## 3.5. Timing

### 3.5.1. Timing Based on Sample Rate

--- Interrupt when sample is ready

### 3.5.2. Timing Based on Hardware Timer

As mentioned in the External 7-Segment Display section, timing is important when displaying the digits. The segments must be kept on long enough to be visible without flickering. A hardware timer can be used for this purpose. The STM32F4 Discovery board is equipped with several timers operating at different maximum frequencies. Timer 3 (TIM3) used in this experiment has a bus frequency of 42 MHz. A prescaler is used to divide the frequency in to a counter frequency according to equation (6).

(6)

When the timer uses a count up counter, the counter value is incremented at each prescaler period up to a maximum. When the counter has reached the maximum value, it is reset to zero and an interrupt flag is raised. The frequency of the flag is found using equation (7).

(7)

The waveforms of the different signals are shown in Figure 4.

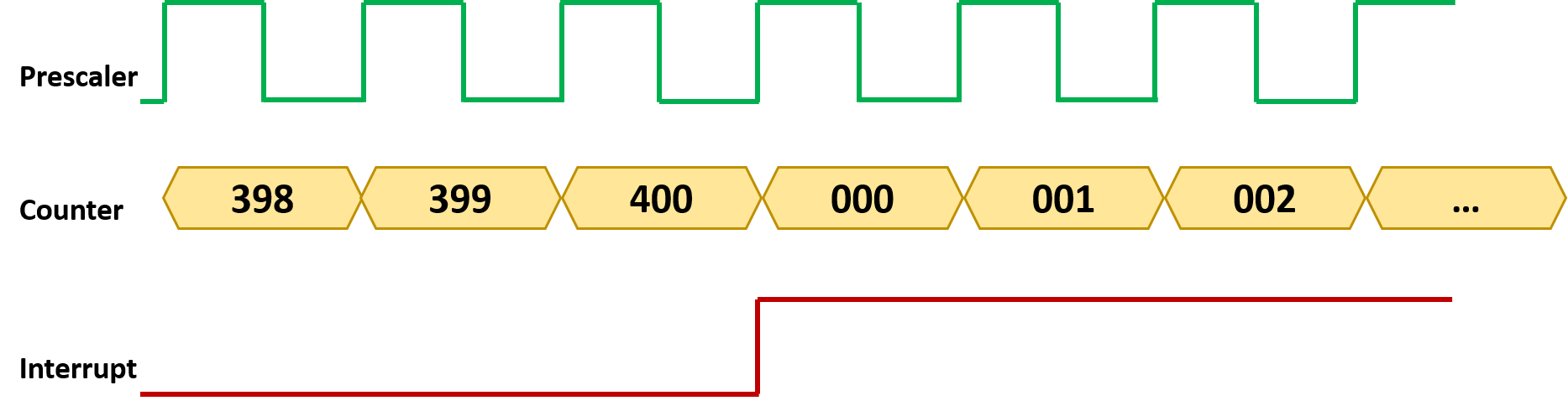


Figure 4: Timer Waveform

# 4. Implementation

The implementation of the system used the STM32F4 Discovery board, an external keypad and an external 7-segment display. The sections are presented in the order of which functions are called.

## 4.1. Component Configuration and Initialization

### 4.1.1. GPIO Configuration

The 7-segment display is an external device, it must use free ports. These ports are selected and set as output ports. An example of the configuration is shown in the following table.

Table 1: GPIO Configuration for 7-Segment Display

|  |  |
| --- | --- |
| **Configuration parameter** | **Value** |
| **GPIOx** | GPIOD |
| **periph\_GPIOx** | RCC\_AHB1Periph\_GPIOD |
| **GPIO\_Pin** | GPIO\_Pin\_8 | GPIO\_Pin\_9 | GPIO\_Pin\_10, |
| **GPIO\_Mode** | GPIO\_Mode\_OUT |
| **GPIO\_Speed** | GPIO\_Speed\_100MHz |
| **GPIO\_OType** | GPIO\_Otype\_PP |
| **GPIO\_PuPd** | GPIO\_PuPd\_DOWN |

The on board LEDs communicate through ports PD12, PD13, PD14 and PD15. These ports are also set to be output ports. An example of the configuration is shown in the following table.

Table 2: GPIO Configuration for LEDs

|  |  |
| --- | --- |
| **Configuration parameter** | **Value** |
| **GPIOx** | GPIOD |
| **periph\_GPIOx** | RCC\_AHB1Periph\_GPIOD |
| **GPIO\_Pin** | GPIO\_Pin\_12 | GPIO\_Pin\_13 | GPIO\_Pin\_14 | GPIO\_Pin\_15 |
| **GPIO\_Mode** | GPIO\_Mode\_OUT |
| **GPIO\_Speed** | GPIO\_Speed\_100MHz |
| **GPIO\_OType** | GPIO\_Otype\_PP |
| **GPIO\_PuPd** | GPIO\_PuPd\_NOPULL |

The keypad used to take user input also uses the free GPIO pins, however they cannot be configured and initialized here, because they change back and forth between an input and an output port. They will be configured based on their use.

### 4.1.2. Accelerometer Configuration, Accelerometer Interrupt and Interrupt Handler

To enable the interrupt from the MEMs sensor, we need to enable external interrupts and the NVIC controller in addition to GPIO, since the sensor is not hardwired to the board.

The Accelerometer configurations are as follow.

Table : LIS302DL MEMs Sensor Configuration

|  |  |
| --- | --- |
| **Configuration** | **Value** |
| **Power\_Mode** | LIS302DL\_LOWPOWERMODE\_ACTIVE |
| **Output\_DataRate** | LIS302DL\_DATARATE\_100 |
| **Axes\_Enable** | LIS302DL\_X\_ENABLE | LIS302DL\_Y\_ENABLE | LIS302DL\_Z\_ENABLE |
| **Full\_Scale** | LIS302DL\_SENSITIVITY\_2\_3G |
| **Self\_Test** | LIS302DL\_SELFTEST\_NORMAL |

We set power mode to active to turn on the sensor. The data rate is 100 as per the specification. All three axes are enabled since the data from all three is required to plug into equation (). The full scale value is set to sensitivity 2\_3G since the range of our data values will not exceed 2g of acceleration. Self test normal indicates that the sensor will not conduct an automatic test to test that it works.

The GPIO configurations are as follow.

Table : GPIO Configuration for Mems sensor

|  |  |
| --- | --- |
| **Configuration** | **Value** |
| **GPIO\_Pin** | LIS302DL\_SPI\_INT1\_PIN |
| **GPIO\_Mode** | GPIO\_Mode\_IN |
| **GPIO\_OType** | GPIO\_OType\_PP |
| **GPIO\_Speed** | GPIO\_Speed\_100MHz |
| **GPIO\_PuPd** | GPIO\_PuPd\_NOPULL |

The GPIO pin is the SPI interrupt pin on the MEMs sensor; this pin corresponds to GPIO pin 0. The mode is in since the processor is detecting whether there is an interrupt, not writing to the device. The output type is configured to pull up pull down. The speed of the device is 100 MHz which is arbitrary as long as it is faster than the sensor’s frequency of 100 Hz. Pull up pull down is left floating. The port for the GPIO is port E.

The external interrupt controller settings are as follow

Table : EXTI Configuration

|  |  |
| --- | --- |
| **Configuration** | **Value** |
| **EXTI\_Line** | EXTI\_Line0 |
| **EXTI\_Mode** | EXTI\_Mode\_Interrupt |
| **EXTI\_Trigger** | EXTI\_Trigger\_Rising |
| **EXTI\_LineCmd** | ENABLE |

The EXTI line is 0 to correspond to GPIO pin 0 from table (). The mode is interrupt since we are configuring an interrupt. The trigger edge is rising due to the specifications. Then we enabled the EXTI.

The NVIC controller configurations are as follow

Table : NVIC Configuration

|  |  |
| --- | --- |
| **Configuration** | **Value** |
| **NVIC\_IRQCHANNEL** | EXTI0\_IRQn |
| **NVIC\_IRQChannelPreemptionPriority** | 0x01 |
| **NVIC\_IRQChannelSubPriority** | 0x01 |
| **NVIC\_IRQChannelCmd** | ENABLE |

The interrupt request channel is EXTI 0 to correspond to table (). The priority is 0x01 for both the group and within the group. We set it higher than the display interrupt so that data acquisition will not get blocked. Then NVIC IRQ channel is activated.

The interrupt handler only checks that the status of the EXTI line. If the line is set, the handler sets the value of the interrupt flag to 1 to inform the main loop that an interrupt has occurred. Then it resets the line it checked.

Additionally, after initializing the accelerometer, it is necessary to configure the sensor to generate an interrupt when it has new data ready. To do so, we write to the control register LIS302DL\_CTRL\_REG3\_ADDR the value 0x04. This sets up the sensor to write to the SPI INT 1 pin when the data ready status goes high.

### 4.1.3. Timer Configuration, Timer Interrupt and Interrupt Handler

A hardware timer is used to manage the timing at which the 7-segment display is updated. TIM3 is configured to use a prescaler of 1000 and count up to 400. According to equation (7), this results in an interrupt frequency of 105Hz. These values were experimentally determined to avoid flickering effect while having reasonable intensity coming from the segments. The interrupt handle was designed to update to variables, the interrupt flag and the interrupt count. The interrupt flag determines when a digit on the 7-segment display should be updated. The interrupt count defines which digit to update. At every interrupt only one digit is updated; therefore, it takes four interrupts (three for the digits and one for degree symbol) to fully update the display.

Table 7: TIM3 Configuration

|  |  |
| --- | --- |
| **Configuration and Initialization parameter** | **Value** |
| **Prescaler** | 1000 |
| **CounterMode** | TIM\_CounterMode\_Up |
| **Period** | 400 |
| **ClockDivision** | TIM\_CKD\_DIV1 |
| **RepetitionCounter** | 0 |

## 4.2. Collect User Input

### 4.2.1. Initialize Keypad

As explained in the theory section, the keypad pins are used as both input and output. For this reason, they are configured on the go. After initializing the pins, the column pins are configured to be output pins and are set high whereas the row pins are configured to be input pins and are pulled down. The configuration is reversed for every read. Digits were constantly read and repeated values were omitted. The key bouncing problem is handled by ignoring sudden changes. For example, if NO\_INPUT signal was detect for the first time, it might be due to key bouncing. Only after NO\_INPUT signal was detected five consecutive times, the program can safely assume that the user has released the key and the missing signal was not due to a hardware bouncing.

### 4.2.2. Return User Input

For this experiment, users are expected to enter a value followed by the ENTER key. Since there were no key dedicated for this purpose, we used the pound key. When the ENTER key is pressed the result is scaled and returned to the processor. The scaling is necessary because the user have the liberty of entering one, two or three digits. The first digit entered is considered to be the most significant digit (the hundreds), while the third digit entered is considered to be the least significant digit (the ones). When user only enters one digit, it must be scaled to the ones and fill the other to digits with zero. Similarly, when user only enters two digits, they must be scaled to the tens and the ones.

### 4.2.3. Improvement on Keypad

Although the current implementation is functional and meet the requirements, some improvements could be added for the future. First, when the keypad function is called it only exits when ENTER is pressed. It is fine for the current implementation because the processor does not have other tasks to perform in the meantime and can wait after the keypad. To make the program more efficient, the execution should return to the main process and check the keypad status periodically. Second, the current implementation does not allow user to erase their inputs. The program must start over is the user has entered an erroneous value. A key could be dedicated to act like a BACKSPACE key.

## 4.3. Tilt Angle Calculation

The process of calculating the tilt angle involved sampling the accelerometer data, normalizing the data, filtering the normalized values, then performing the calculations.

### 4.3.1 Data Sampling

As per the requirements, the accelerometer sampled values at a frequency of 100Hz. To avoid continually polling the accelerometer for data, we only acquired data when an interrupt was generated when the accelerometer indicated it had data ready. Since the accelerometer is not hardwired to the processor, we used EXTI (External Interrupt) to generate the interrupt and NVIC (Nested Vector Interrupt Controller) to channel the interrupt.

To detect the interrupt, we used a global variable as a flag. In the interrupt handler, the flag is set. In our main loop, if the flag is set, then we proceed to reset the flag and read the new data values. Then the data is processed.

--- Signal processor when data is ready

### 4.3.2. Data Processing

Processing the data requires three steps. The first step is to normalize the data using the normalization matrix. The matrix values are stored in the tilt\_detection.h header file as an array of 12 elements. In order to perform matrix operations on the data, the array is stored within an arm\_matrix\_instance from the CMSIS DSP API. When normalizing the data is stored as a 1x4 matrix and multiplied with the normalization vector as in equation (x). The normalization parameters we used are given in Table 1. After normalizing, the output is passed to the Kalman filter.

Table 8: Normalization Matrix

|  |  |  |
| --- | --- | --- |
| -0.001007704542329 | -0.000023576795574 | -0.000020500331472 |
| -0.000005549672267 | -0.001001988618073 | -0.000000555682904 |
| 0.000001981760320 | -0.000006760475090 | 0.000964570821189 |
| -0.520517014244453 | 0.477262260417352 | -0.440876330953729 |

--- Show calibration data in **Appendix A**

--- Kalman parameters, show Matlab simulation results in **Appendix B**

The second step is to filter the data. We used the Kalman filter taken directly from lab 2. We determined the parameters experimentally as described in section 5.2. For this application, we required the use of three different instances of a Kalman filter state, one for each of the accelerometer values. Using only one Kalman filter would have resulted in incorrect final calculations. Each individual input would have advanced the iteration when all three accelerometer values should be in the same iteration for different states. The filtered values are then passed to tilt angle calculation.

The final step is to perform the calculation. The pitch angle we determined by using equation (x) and the roll angle we determined by using equation (y). However, the result of these values only range from -90 to 90. As a result, we included calculations to determine the quadrant of the angle given the sign of the z acceleration and the sign of the calculated degree.

### 4.3.3. Data Comparison

---

## 4.4. Result Display

### 4.4.1. Visual Feedback

After comparing the current orientation against the targeted angle, an animation is used to help user orient the board toward the desired position. This animation uses the first digit of the 7-segement display. When the user should increase the inclination of the board, the segments will light up from bottom up. And when the user should decrease the inclination of the board, the segments will light up from top down.

### 4.4.2. Display Current Angle

Once the user is within five degrees of the target roll angle, the 7-segment display will show the current angle. The value of the current angle is passed on to display as a floating point. The floating point is decomposed into integer digits and a signal for the decimal point. Digits are displayed one by one according to their value. The multiplexing of the segment was done before hand and the segments to turn on for all integer values (between 0 and 9) and the decimal point is pre-defined. The display is refreshed at a frequency of 25.125Hz (=105Hz/4). This frequency is controlled by the hardware timer (TIM3). At every interrupt, one digit is updated. It takes four interrupts to refresh all three digits and the degree symbol.

## 4.5 Continuous Process

--- Continuous process until user presses reset

# 5. Testing and Observation

## 5.1. Accelerometer Calibration

To calibrate our sensor, we performed offline calibration as described in section 3.2. First we collected samples of raw accelerometer values at the six positions as described in section 3.2. Then we averaged the acceleration values and formed the w matrix in accordance with its format as described in section 3.2. Then in Matlab, we solved for equation (4). Taking the output values, we then hardcoded them to an array. The normalization measured the tilt angles within 4 degrees of accuracy.

--- Within 4 degree accuracy

--- Low accuracy, therefore could use lookup table instead

--- Using complex math functions on floating points --- more power hungry

--- Using lookup table --- less accurate --- more memory needed --- less power

## 5.2. Kalman Filter

The Kalman filter initial state parameters we chose were q = 0.025, r = 5, x = k = p = 0. x is the estimated value, and since each acceleration value will be in the range of 0 to 1, we set x to 0. To get the other values of the parameters, we plotted a comparison of filtered values with the raw data with varying parameter values. We held r constant at 5 and increased q from 0.0025 to 2500 as seen in Figures 4 to 11. Just by comparing visually, as the value of q rose, the signal became noisier. The optimal value we found at q = 0.025 and r = 5 (Figure 5). The figures show the comparison for only the X values, but the filters for the Y and Z values gave the same result.

# 6. Conclusion

In this experiment, we implemented a functional system which detects the orientation of the board. The process prompts the user for an input using an external keypad. The accelerometer samples at a rate of 100Hz. Once the data is ready, it signals the processor using an interrupt. The processor will calibrate and filter the data. After converting the acceleration into an angle in degrees, it compares it against the user input. If the current angle is more than five degrees from the target angle, the 7-segment display will be used to show an animation helping the user orienting the board. If the roll angle is with five degrees of the target angle, the 7-segment display will show the current roll angle.

Some improvements could be done to the current implementation. As mentioned, the keypad could be used differently and the conversion from acceleration to angle could be more power efficient.

The system can be used for game consoles that requires movement or can be implemented as a movement tracker.

# Reference

# Appendix A – Calibration Data

Figure 5: Kalman Filter q = 0.0025, r = 5

Figure 6: Kalman Filter q = 0.025, r = 5

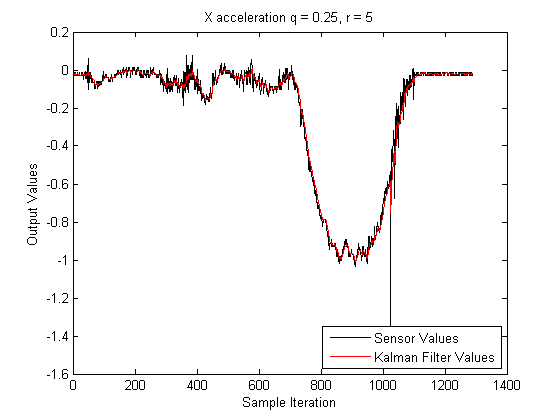


Figure 7: Kalman Filter q = 0.25, r = 5

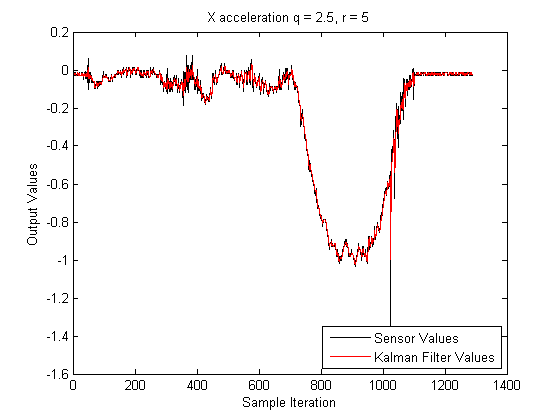


Figure 8: Kalman Filter q = 2.5 r = 5

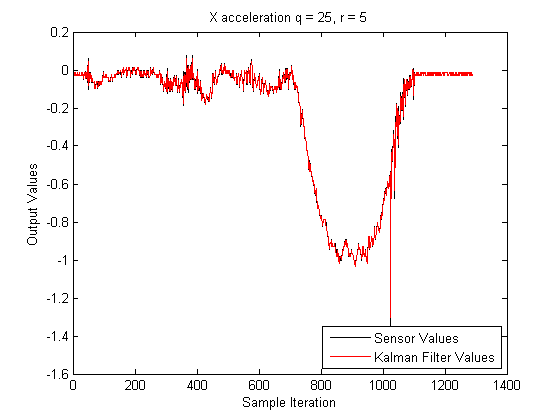


Figure 9: Kalman Filter q = 25, r = 5

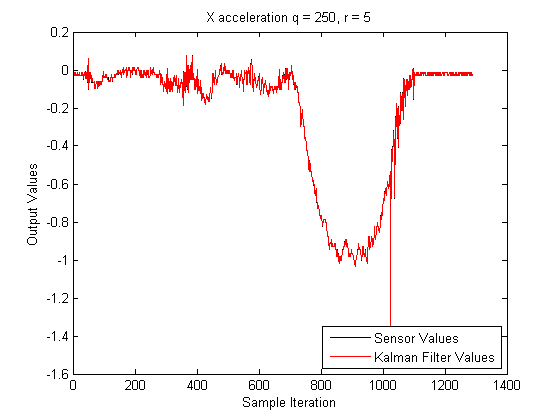


Figure 10: Kalman Filter q = 250, r = 5

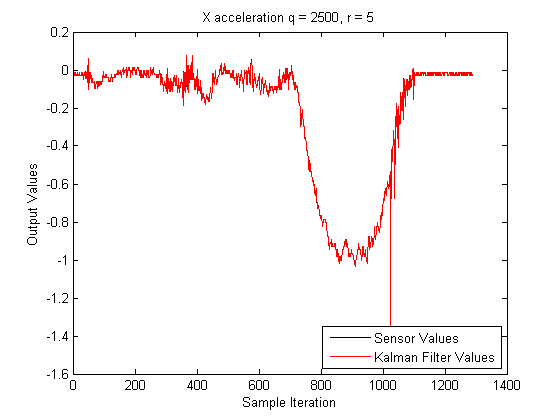


Figure 11: Kalman Filter q = 2500 r = 5