justification is missing in some parts, no section/subsection header numbering -0.1
Missing theory on SPI -0.2
missing theory on Timers -0.2
missing theory on NVIC, interrupts -0.2

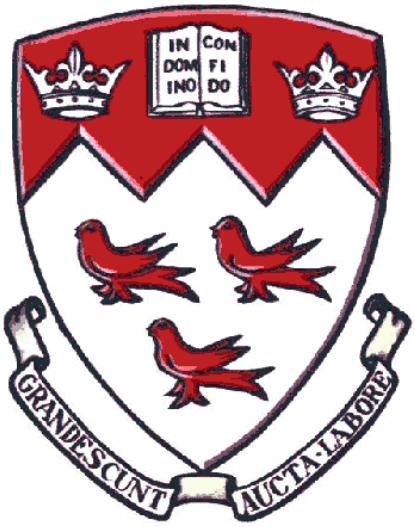
6.3/7

Experiment #3

MEMS Accelerometer, Timers and Interrupts

Your report is very well written, had it not been for the missing sections, you would have got a better mark

Maxim Goukhshtein ID: 260429739
Olivier Laforest ID: 260469066
Department of Electrical and Computer Engineering
McGill University, Montreal



March 16, 2015
Group 3

ECSE 426 Microprocessor Systems

Abstract

The goal of the experiment presented in this report was to design a tilt angle detection system using the STM32F407 Discovery board. Utilizing a 4-by-4 alphanumeric keypad, the user can interactively choose a target angle and be provided with visual feedback regarding the board's orientation via a 7-segment display. This report presents the details regarding the procedures that were used to calculate the board's tilt angle by using the LIS3DSH ST accelerometer, as well as the methods that were used to interface the board with the keypad and 7-segment display. Furthermore, the report will provide details regarding the procedures that were used to calibrate the accelerometer, as well as to filter the raw sensor data.

Problem Statement

During the course of this experiment, acceleration readings from the LIS3DSH MEMS sensor, a three-axis accelerometer, are used in order to calculate the F4 board's tilt angle. In particular, the board's accelerations along three axes are read at a rate of 100 Hz, before being used to calculate the board's pitch angle (see more details in the theory section). Prior to using the MEMS sensor, the user is prompted to input a target angle, between 0 and 180 degrees, using a 4-by-4 alphanumeric keypad connected to the board. Once a selection is made, a 7-segment display is used to provide the user with visual feedback. In particular, the display points the user to direction in which the board must be tilted (i.e. up or down) in order for it to reach the target angle. Once it is detected that the board is within 5 degrees of the chosen angle, the display switches to showing the board's current pitch angle. In order to ensure that the pitch angle is calculated accurately (i.e. within 4 degrees of the actual angle), it was necessary to perform an offline calibration of the sensor. Furthermore, to ensure the proper functioning of the keypad, it was essential to implement a robust keypad scanning algorithm and a proper button denouncing mechanism (in order to avoid false registration of button presses). Finally, given that only a single digit can be turned on at a time on the 7-segment display, it was important to design a good digit multiplexing mechanism, in order to avoid flickering and provide the user with an overall pleasant and easy to read visual feedback.

Theory and Hypothesis

General theory and conventions

In order to calculate the board's tilt angle, accurate accelerations along 3 axes must be obtained. Figure 1 presents the conventions for the various orientations and names for these axes, to be used throughout this section.

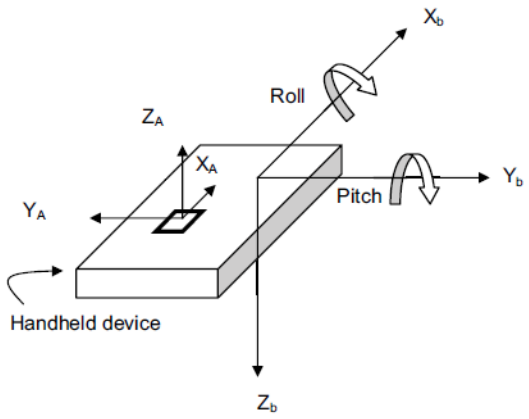


Figure 1: MEMS sensor axes and tilt angles (adapted from [1]**)**

The LIS3DSH MEMS sensor detects the acceleration due to Earth's gravity along the X_B, Y_B and Z_B body axes. The board's tilt angle can then be calculated via a projection of the detected forces onto the X_A, Y_A and Z_A sensing axes [1] [2]. When the device is oriented with one of its body axis parallel (or anti-parallel) to the gravity vector (thus the other 2 axes are orthogonal to the gravity vector), one can predict the accelerations that should be ideally reported by the sensor. For example, if the board is positioned such that the Y_B axis is parallel to the gravity vector (i.e. Y_B is pointing down), then the sensor should ideally report an acceleration of +1g along the Y-axis and 0g along the other 2 axes. Figure 2 shows the expected accelerations along the 3 axes, when the board is held stationary in each of the possible 6 positions, as described above.

Stationary position	Accelerometer (signed integer)		
	A_x	A_y	A_z
Z_b down	0	0	+1 *g*
Z_b up	0	0	-1 *g*
Y_b down	0	+1 *g*	0
Y_b up	0	-1 *g*	0
X_b down	+1 *g*	0	0
X_b up	-1 *g*	0	0

Figure 2: Expected accelerations along the 3 axes for 6 stationary positions [1]**.**

During the course of this experiment, the calculated angle was the pitch angle α. As can be seen from figure 1, the angle corresponds to the rotation about the Y_B axis, or the angle between the horizontal plane and the X_B axis. As the requirements specify that the reported angles should be between 0 and 180 degrees, when the board is placed on a flat level, the pitch angle is considered to be, by convention, 90 degrees. When rotating the board about the Y_B axis, such that the X_B axis moves upwards, the angle increases. Similarly, when the X_B axis moves downwards, the pitch angle decreases. Figure 3 illustrates this idea:

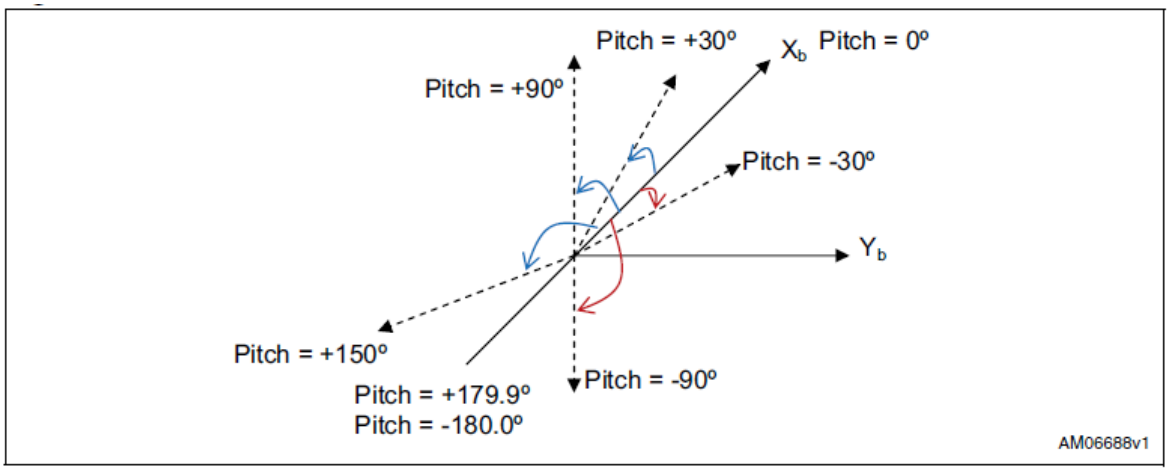


Figure 3: definition of the pitch angle α [1]

Sensor calibration

Despite having undergone factory-calibration, the MEMS sensor must undergo an offline calibration, in order to ensure a higher level of accuracy of the reported results. There are numerous ways in which an accelerometer can be calibrated. In the case of this experiment the sensor was calibrated using the procedure described in [1]. This method involves taking measurements in each of the stationary 6 positions in figure 2, and then using a least-squares approach to find the calibration matrix, which transforms the raw sensor data to the expected (normalized data). That is, if matrix N represents the normalized gravity matrix, matrix R represents the collected raw measurements, then one must calculate the matrix G such that:

$$N = MG \tag{1}$$

For example, suppose one collects n (raw) sensor measurements in each of the six positions, an $nx3$ matrix of normalized values can be constructed as follows:

$$N = \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \end{bmatrix} \; where \; N_i = [A_X \; A_Y \; A_Z]_{(n\times 3)} \; for \; i = 1,2 \ldots,6$$

Where A_X, A_Y, A_Z correspond to the expected accelerations (as defined above in figure 2) and each of the $i's$ corresponds to the one of the 6 stationary positions. For example, following the format in figure 2, for position 3 (i.e. Y_B down), one constructs $N_3 = [0\ 1\ 0]_{(n\times 3)}$.

The raw sensor data matrix R is constructed as follows:

$$R = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{bmatrix} \; where \; R_i = [A_{Xm} \; A_{Ym} \; A_{Zm} \; 1]_{(n\times 4)} \; for \; i = 1,2 \ldots,6$$

Where A_{Xm}, A_{Ym}, A_{Zm} correspond to the raw acceleration values as reported in each of the positions i by the MEMS sensor.

Having collected the sensor data for each of the positions, one can then directly calculate the matrix G, using the least-squares method:

$$G = (M^T M)^{-1} M^T N \tag{2}$$

Normalized measurements and pitch angle calculation

The aforementioned matrix G is a 4x3 matrix of calibration (or, misalignment between the sensing and body axes) coefficients that can be used to correct raw sensor data. That is, once the matrix G is identified, it is possible calculate the "correct" acceleration from the raw acceleration via:

$$A_C = G^T A_R \tag{3}$$

Where A_C is the "corrected" or "normalized" acceleration values and $A_R = [A_{Xr}\ A_{Yr}\ A_{Zr}\ 1]^T$ were $A_{Xr}\ A_{Yr}\ A_{Zr}$ are the raw acceleration measurements.

Using the normalized acceleration values one can, using trigonometry, derive the formula for calculating the pitch angle:

$$\alpha = \tan^{-1}\left(\frac{A_X}{\sqrt{(A_Y)^2 + (A_Z)^2}}\right) \tag{4}$$

Where A_X, A_Y and A_Z are the normalized accelerations.

Filtering

As is often the case, the sensor measurements are subject to noise. In order to reduce the effects of noise, the calculated pitch angles are filtered using a 1-diminsional Kalman filter. Please refer to the lab 2 report for further information regarding the theory relating to Kalman filtering.

Implementation

The implementation process can be divided into four parts: implementation of data acquisition from the MEMS accelerometer sensor, implementation of the keypad functionalities, implementation of the 7-segments display and the overall operation of the program.

MEMS Accelerometer Sensor

The model number of the Discovery board that is used is MB997C, therefore the MEMS accelerometer sensor model used is LIS3DSH. In order to acquire acceleration readings from the MEMS accelerometer sensor on the STM32F4 Discovery board, several components and interfaces need to be initialized and configured. The MEMS sensor on the Discovery board is an off-chip sensor which means that the processor does not have direct access to it. The Serial Peripheral Interface (SPI) is used to retrieve the readings from the MEMS sensor and an External Interrupt/Event Controller (EXTI) is used to let the processor know that a measurement is ready. The initialization and configuration of the SPI, GPIO pins and the enabling of their associated clocks (i.e. APB2, ABH1) is done in a private sub-function of the MEMS sensor driver's

initialization function and there is no configuration options given to the designer that uses the driver. Consequently, the configuration of these components will not be discussed here.

MEMS Sensor Configuration

For the initialization and configuration of the LIS3DSH sensor, the following components of the initialization structure, *LIS3DSH_initStruct*, had to be set up.

- *Power_Mode_Output_DataRate*
- *Axes_Enable*
- *Continous_Update*
- *AA_Filter_BW*
- *Full_Scale*
- *Self_Test*

The requirement specification of experiment 3 required the reading of the MEMS sensor to be at least a 100 times per second, therefore, output data rate was set to a $100Hz$ (i.e. *LIS3DSH_DATARATE_100*). To compute either the pitch or roll of the board, all three axes' acceleration are required, as a result, all three axes were enabled by ORing their respective enabling constant (e.g. *LIS3DSH_X_ENABLE*). The continuous update of the output registers was disabled (i.e. *LIS3DSH_ContinousUpdate_Disabled*) in order to ensure that the 16-bits output registers are not updated until both the lower and upper 8-bits registers have been read. This ensures that no values from an older sample are read [3]. The anti-aliasing filter bandwidth is usually set at half of the sampling frequency, it was therefore set to $50\ Hz$ (i.e. *LIS3DSH_AA_BW_50*). Computing the pitch and/or roll of the board requires the board to be subjected to gravity which corresponds to an acceleration of $1g$. As a result, the full scale value of the sensor was set to the smallest possible value of $2g$ (i.e. *LIS3DSH_FULLSCALE_2*) in order to account for small accelerations while retaining the maximum possible sensitivity of $0.061\ milligals/digit$. If it was known that the board may experience acceleration larger than $2g$, the full scale value should've been increased at the cost of a lowered sensitivity. Finally, the self test feature was disabled (i.e. *LIS3DSH_SELFTEST_NORMAL*) since if it is left enabled, a biasing force is constantly applied to the sensor which is not desirable for normal operation of the designed system.

After the initialization of the MEMS sensor, the interrupts signaling that the data is ready to be read needed to be configured by initializing the *LIS3DSH_DRYInterruptConfigTypeDef* parameters.

- *Dataready_Interrupt*
- *Interrupt_signal*
- *Interrupt_type*

First, the interrupt signaling that data is ready needs to be enabled (i.e. *LIS3DSH_DATA_READY_INTERRUPT_ENABLE*). Second, the interrupt is set to be active high and pulsed (i.e. *LIS3DSH_ACTIVE_HIGH_INTERRUPT_SIGNAL* and *LIS3DSH_INTERRUPT_REQUEST_PULSED*) as required by the specification of the experiment.

External Interrupt Configuration

As mentioned earlier, the MEMS sensor is an off-chip sensor and requires its interrupt to be channeled to the processor via an external interrupt. To achieve this, the following parameters of an EXTI initialization structure are configured (i.e. *EXTI_InitTypeDef*).

- *EXTI_Line*
- *EXTI_Mode*
- *EXTI_Trigger*
- *EXTI_LineCmd*

The external interrupt line 0 (i.e. *EXTI_Line0*) is selected since it is the line which is hardwired to the GPIO PE0 which is mapped to the interrupt signal of the MEMS sensor [4]. In order to use the EXT0, the high speed bus APB2 clock is enabled for SYSCFG and the EXT0 is connected to the GPIO pin PE0. The external interrupt mode was set to interrupt (i.e. *EXTI_Mode_Interrupt*) (this mode could have also been set to event mode). Rising edge triggering was selected (i.e. *EXTI_Trigger_Rising*) and the external interrupt was enabled.

Nested Vector Interrupt Controller Setup

Finally, the interrupt request handler must be enable in the Nested Vector Interrupt Controller (NVIC) by configuring the following parameters of the initialization structure *NVIC_InitTypeDef*.

- *NVIC_IRQChannel*
- *NVIC_IRQChannelPreemptionPriority*
- *NVIC_IRQChannelSubpriority*
- *NVIC_IRQChannelCmd*

The interrupt request channel is set to the one of EXTI0 (i.e. *EXTI0_IRQn*). The preemption priority and sub-priority are set to the highest priority and therefore the lowest number (i.e. 0) such that the processor handles interrupts generated by the MEMS sensor first. The channel is finally enabled to activate the handling of the MEMS interrupts. The MEMS sensor is now fully configured and ready to use.

Data Acquisition and Conversion

The specifications of experiment 3 required that either the pitch or the roll of the board be displayed in real time. The pitch was selected to be displayed. With the MEMS sensor initialized and configured, it is now generating acceleration measurements 100 times per second. Every time a measurement is ready, an interrupt is generated. The processor then switches to the corresponding handler routine which triggers the retrieval of the acceleration readings from the MEMS sensor through its driver API call. The raw acceleration readings are normalized and adjusted by multiplying them with the calibration matrix, as was explained in the theory section. These normalized X, Y and Z acceleration values are then converted into pitch angle by using equation 4.

Filtering

Given the fact that the pitch angle values are computed from noisy unfiltered raw accelerations which exhibit sharp fluctuations, the computed values also contain noise, and therefore must be filtered. In order to remove the noise from the pitch angle values, a 1-dimenstional Kalman filter is used. The selection of the initial Kalman filter parameters was done experimentally (see additional information and results in the next section).

Keypad Interface

In order to allow the user to input a desired board pitch angle, a 4X4 keypad was used. The GPIO pins PE7 to PE14 were used to connect and allow the program to interact with the keypad. These GPIOs need to be initialized and configured. The timer TIM3 was used to implement a debouncing algorithm for the keypad. However, its initialization and configuration is discussed later in this section, as TIM3 was also used in the implementation of the 7-segment display.

GPIO Configuration

To be able to use the GPIOE ports, the clock on the AHB1 bus needs to be enabled. This step had already been done inside the initialization of the MEMS sensor private driver function. The configuration of the GPIOE pins 7 to 14 is done by setting the following parameters of the *GPIO_InitTypeDef* structure.

- *GPIO_Pin*
- *GPIO_Speed*
- *GPIO_Mode*
- *GPIO_OType*
- *GPIO_PuPd*

As mentioned above, the ports PE7 to PE14 are used to connect the keypad. Ports PE7 to PE10 are connected to the rows of the keypad and are configured as outputs, while ports PE11 to PE14 are connected to the columns of the keypad and are configured as inputs. The speed of the port was set to $50MHz$ (i.e. *GPIO_Speed_50MHz*) which is adequate for the operation of the keypad. Finally, the output type of the ports were set to push/pull (i.e. *GPIO_OType_PP*) since the ports need to be driven high or low and the default push/pull configuration of the port was set to pull the ports up (i.e. *GPIO_PuPd_UP*), since the algorithm to detect a key stroke was design as active low (i.e. a pressed key represents a logical 0 and an unpressed key, a logical 1).

Keypad Operation

The keypad is composed of 4 columns and 4 rows. When a key is pressed, it electrically connects the corresponding row and column. Therefore, to identify which key is pressed by the user, an algorithm was designed such that it subsequently sets one of the rows active, and while the row is active, it scans the columns to see which one is active (if the user is currently pressing a key). By knowing the active row and column, it can be easily found which key is pressed by a simple mapping of row/column pattern to each key, since each row/column pattern can only be associated with a unique key. As long as the user has not successfully entered the desired angle and pressed the enter key, the scanning of the row/column pattern runs in a while loop. The frequency at which this while loop runs is equal to the frequency of the TIM3 timer which is $160Hz$. Given the frequency of the scanning loop and the bouncing effect of the mechanical switch of the keys of the keypad, an algorithm that ensured that a key pressed by the user is only counted once was implemented. This algorithm consists in deactivating the monitoring of the keypad for a fixed amount of TIM3 clock cycles as soon as a key is detected as pressed, which effectively reduces the frequency at which the keypad is scanned. After experimental tuning, is was found that a scanning frequency of $40Hz$ was adequate for the proper operation of the keypad. The '#' key was selected on the keypad to represent the enter key (which must be pressed to confirm an angle selection). Several safety features which ensure that the angle entered by the user is in the allowed range of 0° to 180° and that only numerical keys or the enter key are valid inputs were added. Finally, if the user chooses 3 digits, the

program automatically proceeds without requiring the user to press enter, as it is not necessary (assuming that the selected angle is less than or equal to 180°).

7-Segments Display

A 4 digit 7-segments display is used to provide visual feedback to the user. Only one digit of the 7-segments display can be displayed at once, therefore, a timer is needed to cycle and display each digit sequentially. The hardware timer TIM3 was used to provide the necessary delay. GPIO pins PD0 to PD4 and PD6 to PD12 were used to connect and operate the display.

TIM3 Configuration

Initialization of the TIM3 timer involves the activation of the APB1 bus clock. To configure the TIM3 timer, the following parameters need to be set.

- *TIM_Prescaler*
- *TIM_CounterMode*
- *TIM_Period*
- *TIM_ClockDivision*

According to [4], the base frequency of the TIM3 timer is $84MHz$. In order to achieve the desired digit cycle frequency of $160Hz$, which was found adequate by experimental tuning, the prescaler was set to its maximum possible value of $2^{16} - 1 = 65535$, which yields an intermediate frequency of $84MHz/65535 = 1282Hz$. The mode in which the counter of the TIM3 timer counts was set to counting up (i.e. *TIM_CounterMode_Up*). The period of the timer was set to 8 in order to achieve the operating frequency of $1828Hz/8 = 160Hz$. The base clock division was set to 1 since no division of the base clock was required.

NVIC Configuration

Once the TIM3 is initialized, the interrupt request corresponding to the TIM3 timer in the NVIC needs to be initialized. The TIM3_IRQ is initialized in the exact same way as the EXTI0_IRQ described in the corresponding section, except that the preemption priority of the TIM3 is set to 1. In other words, the priority of TIM3 Interrupt request is lower than the one of the MEMS sensor's interrupt. This is done because reading the MEMS sensor's measurement has a higher priority than displaying the next digit of the 7-segments display.

GPIO Configuration

The GPIO initialization parameters are the same as one presented in the keypad description in this section. The parameters of the GPIO used for the 7-segments display are initialized to the same value as the ones used to operate the keypad, except that all of the GPIOs for the display are configured as outputs and that no pull is applied to the ports. GPIO pins PD0 to PD4 and PD6 to PD7 are connected to the pins representing segment A to G of the 7-segments display, respectively. GPIO PD8 is connected to the decimal point pin and PD12 to the degree sign pin. GPIO pins PD9 to PD11 are connected to the base of three separate NPN transistors which act as switches to close the circuit to ground each digit of the 7-segments display. The port PD9 closes the circuit of the first digit on the display, PD10 the second digit and PD11 the third digit of the display.

7-Segments Display Operation

Three modes of operation are required for the 7-segments display. In the first mode, the board's pitch angle is smaller than the user's desired angle by more than 5 degrees and the display outputs the word "UP" signifying that the user should move the tip of the positive X axis of the LIS3DSH MEMS sensor as shown in figure 4. In order to display a letter or a number, the necessary segment's port which realizes the desired letter (or number) are set high before the digit of the display is lit.

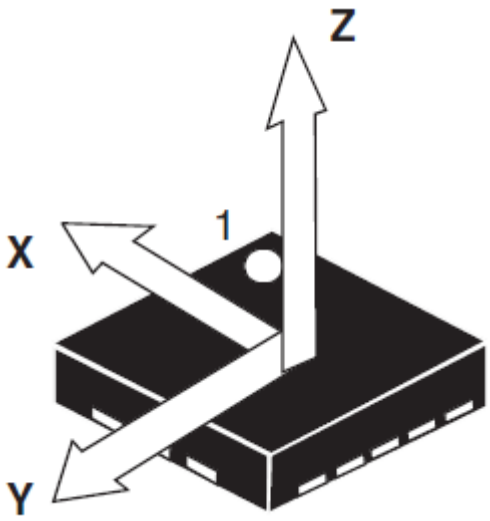


Figure 4: Positive axis orientation of the LIS3DSH MEMS Accelerometer sensor [3].

In the second mode of operation of the display, the pitch angle of the board is larger than the desired angle by more than 5 degrees and the display outputs the letters "dn", which stands for "down", signifying the user to move tilt the tip of the positive X axis down. In the final mode of operation, the board's pitch angle is within $\pm 5°$ of the desired angle and the display outputs the current board angle in real time. To correctly display the pitch angle in real time, the program identifies the magnitude of the board's pitch angle to determine if and where the decimal point should be lit, and then parses the correct digit of the pitch angle to be displayed. The three most significant digits of the pitch angle are displayed one at a time and are subsequently displayed in a cyclic manner based on the clock cycle of the TIM3 timer.

Overall Operation

The overall operation of the program is separated in two sections. First, the program waits for the user to input his desired target pitch angle. Once the desired angle is correctly entered, the program enters an infinite loop which performs two tasks in its body. When a measurement of acceleration done by the MEMS sensor is ready, the program retrieves the measurement and updates its pitch angle. Otherwise, when the TIM3 timer clock cycle expires, the program proceeds to display the next digit of the current mode of operation on the 7-segments display.

Testing and Observations

The testing of the system includes the testing of the MEMS sensor calibration, the Kalman filter parameters, the keypad and the overall system behaviour.

MEMS sensor calibration

After performing the offline calibration of the MEMS sensor (see Matlab calibration script in Appendix A1), the functioning of the sensor was tested to ensure that the reported normalized accelerations match the expected values. In order to so, the board was placed in the six stationary positions that were described in the theory section. The recorded values were found to closely match those found in the table in figure 2, indicating that the calibration was done correctly.

Kalman Filtering

The Kalman filter parameters were found largely experimentally using a Matlab script (see Appendix A2). First, around 2000 raw pitch angle data samples were collected and then, by experimentally tuning the various parameters, a good set of parameters was selected. As was the case in the previous experiment, the selection of the estimation error covariance parameter p didn't have much of an effect on the filtering, as it was found that the filtering converges to the same values relatively quickly regardless of the chosen value of p. Furthermore, based on the update equation, it's clear that the initial value of the Kalman gain k is irrelevant, as it immediately being set to some value based on the update equation for k. Therefore, it is essential to find appropriate values for the process noise covariance q and the measurement noise covariance r.

Keeping the values of r fixed at 25 and varying q (while setting $r = k = 0$), the following plots were generated:

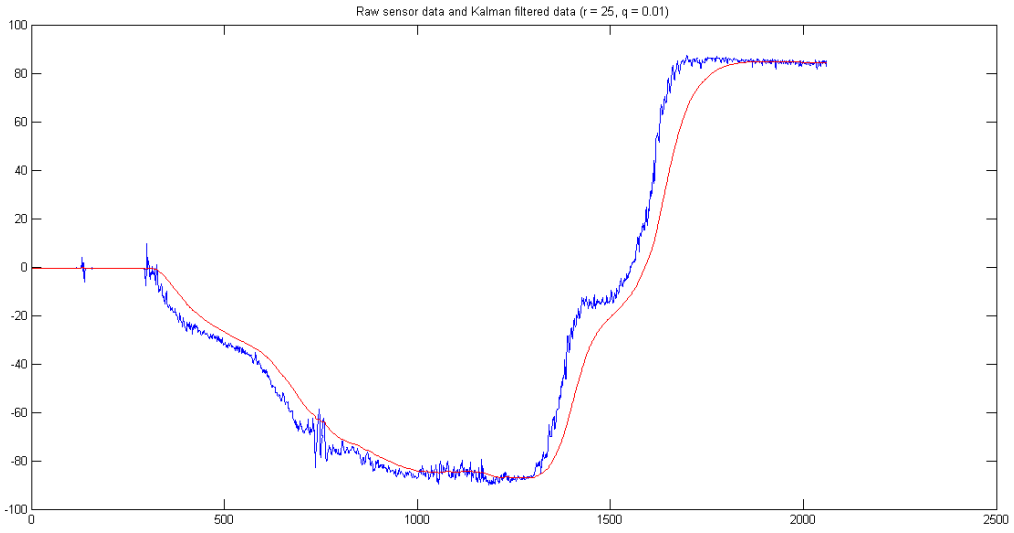


Figure 5: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 25 and q = 0.01.

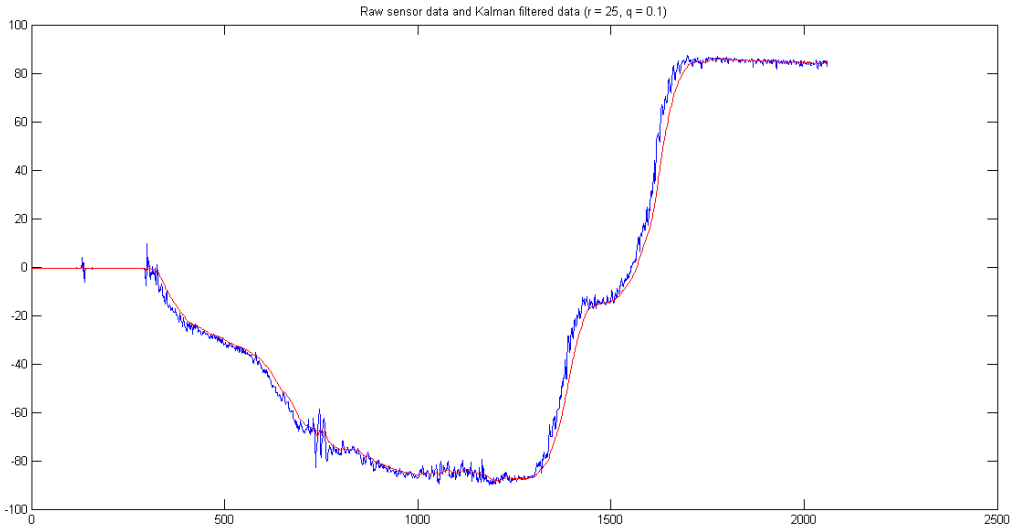


Figure 6: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 25 and q = 0.1.

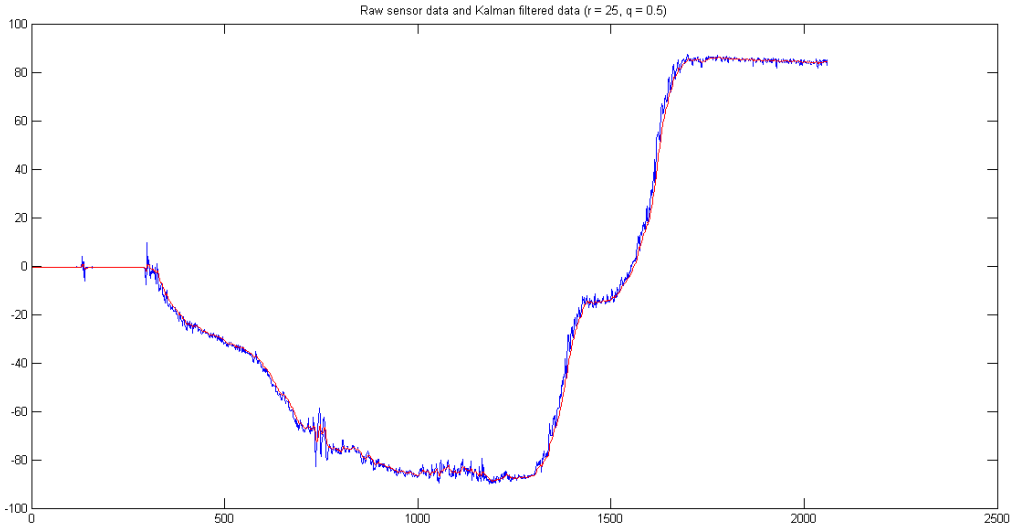


Figure 7: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 25 and q = 0.5.

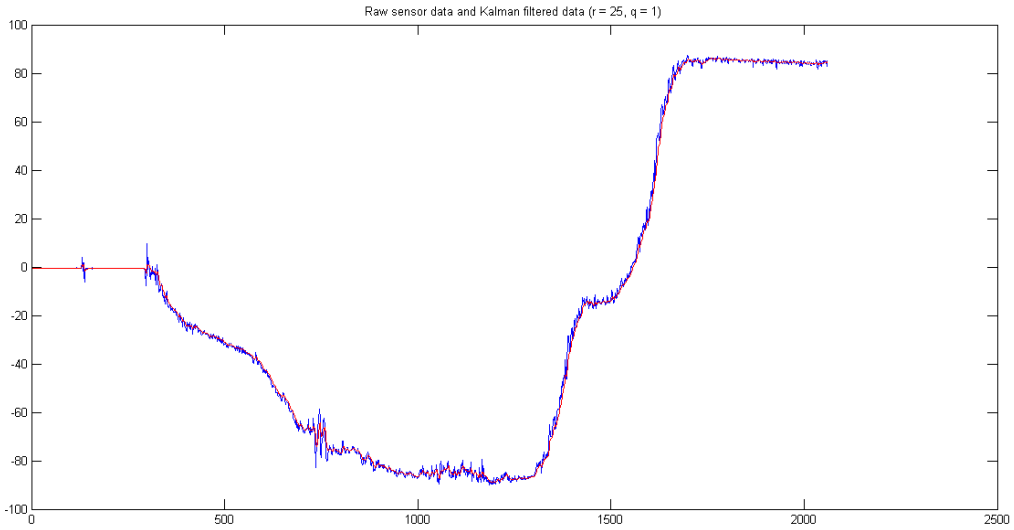


Figure 8: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 25 and q = 1.

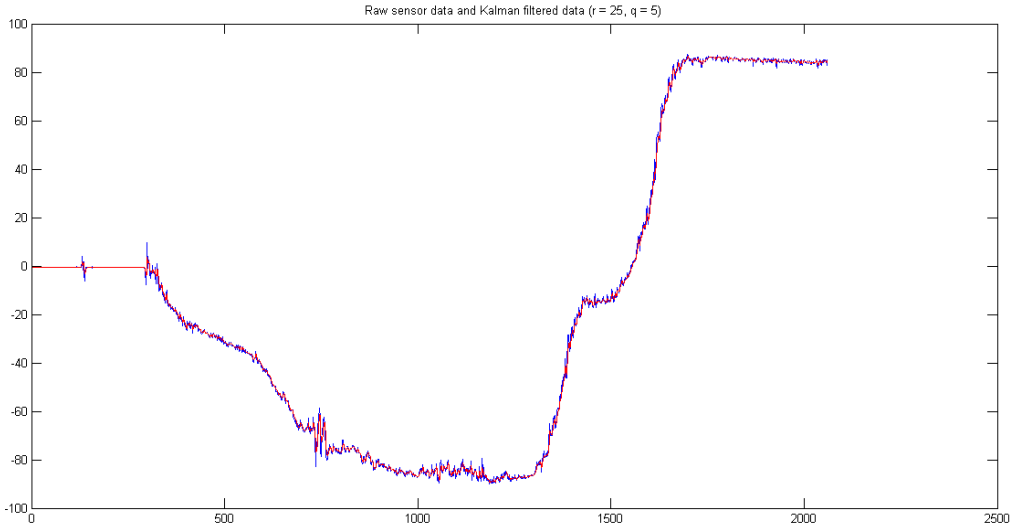


Figure 9: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 25 and q = 5.

Next, keeping the values of q fixed at 0.5 and varying r (while setting $r = k = 0$), the following plots were generated:

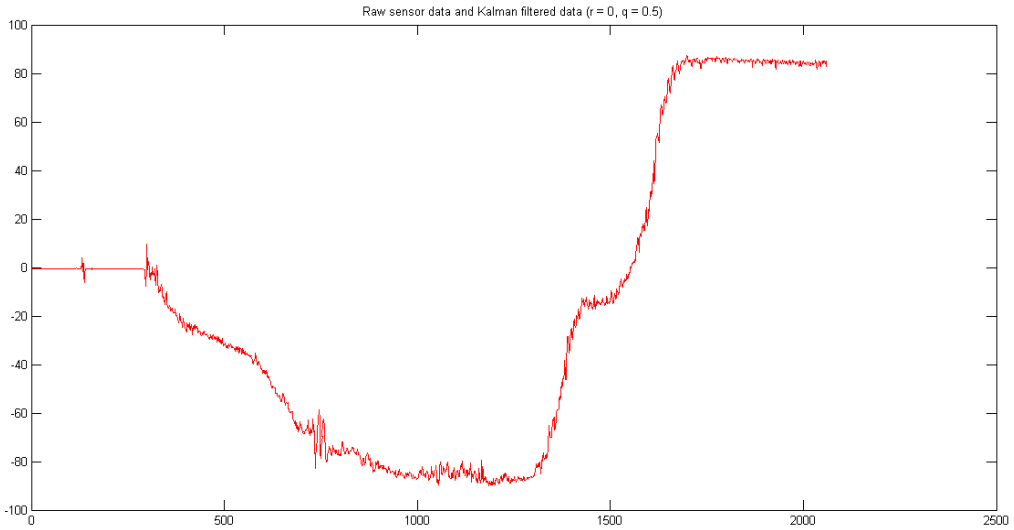


Figure 10: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 0 and q = 0.5.

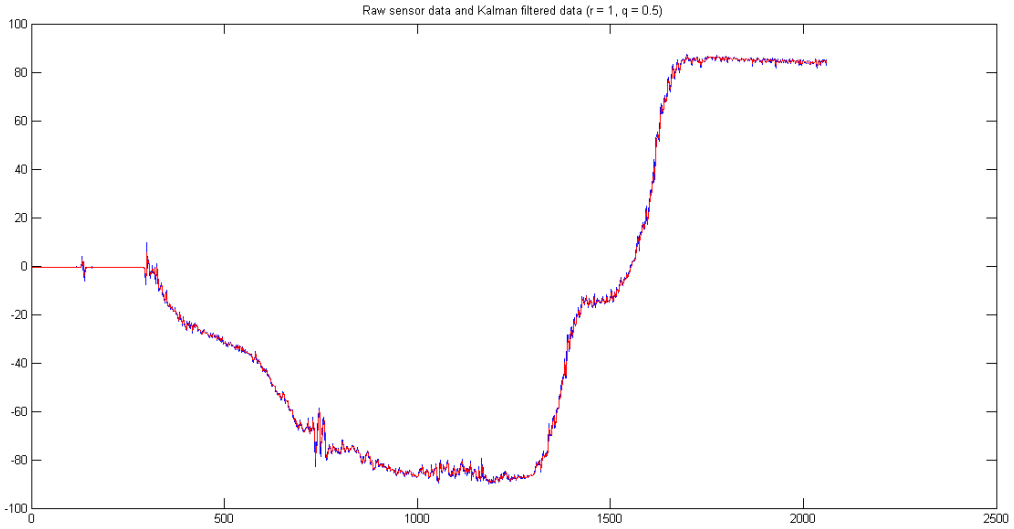


Figure 11: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 1 and q = 0.5.

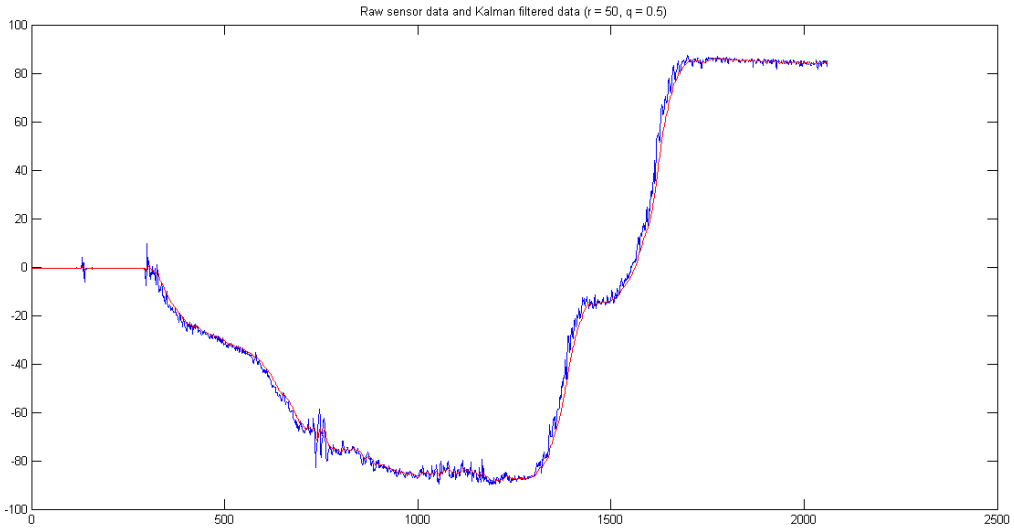


Figure 12: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 50 and q = 0.5.

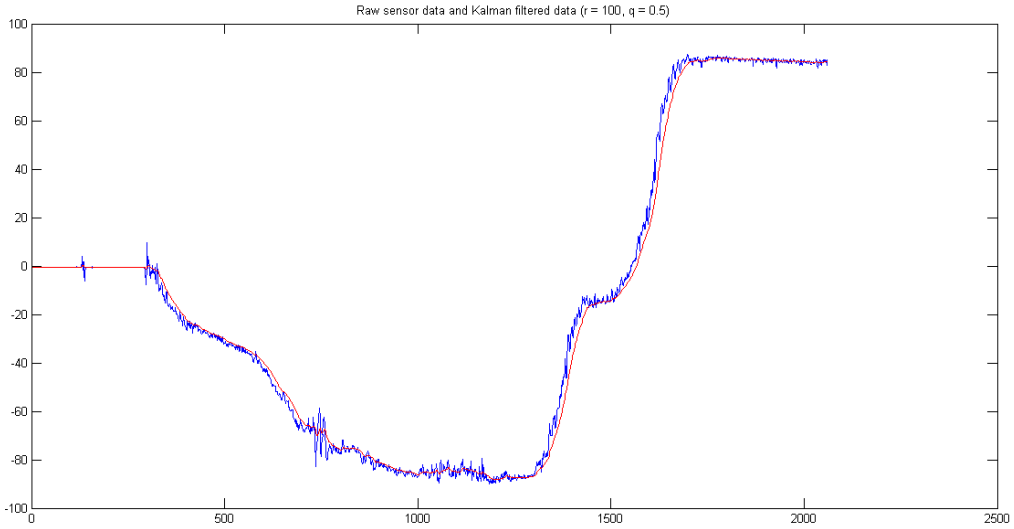


Figure 13: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 100 and q = 0.5.

Based on these experiments, it was decided to choose the Kalman filter parameters to be $r = 50$, $q = 0.5$ and $r = k = 0$, as these were deemed to provide good results. The filtered data appeared to follow the general trend very closely, while remaining rather smooth.

Keypad

In order to ensure that the button de-bouncing mechanism works properly, numerous tests, using all 16 buttons were performed. The system was run in debug mode and the user inputs (i.e. pressed buttons) were printed out. It was observed that the keypad behaviour was adequate, with button presses being

registered only once per press, indicating that the mechanism works properly. Furthermore, various special cases were tested (e.g. entering an angle greater than 180 or pressing the enter button without selecting any angle), to ensure that the system behaves as expected (i.e. prints out an error message and prompts the user to re-enter the target angle once again).

Overall system behaviour

Once all the various components were tested separately, the entire system was put under test. This was done by choosing various target angles using the keypad, and observing that the system exhibits the expected behaviour, namely that it provides the user with the expected feedback via the 7-segment display. In particular, target angles were set to correspond to known angles made by objects provided by the TAs, in order to ensure that once the board reaches the desired angle, the displayed pitch angle is within 4 degrees of the selected target angle. Using this method, it was found that the operation of the designed system was in accordance with the requirements.

Conclusion

In the course of this experiment, using the off-chip LIS3DSH MEMS accelerometer found on the STM32F4 Discovery board, a 4X4 keypad and a 4 digits 7-segment display, a system which prompts the user to input a desired pitch angle for the board and outputs a visual feedback to guide the user was implemented. Measurements of the gravitational acceleration experienced by the board combined with the use of a 1-dimentional Kalman filter to reduce the noise present in the raw data were used to compute the real-time pitch angle of the board. The Kalman filter's initial parameter values were determined experimentally. To ensure a high level of accuracy in the obtained accelerometer readings, the MEMS sensor underwent an offline calibration. The MEMS sensor communication with the microprocessor was achieved through the use External Interrupt/Event Controller and the Serial Peripheral Interface. The keypad and the 7-segments displays were connected to the board through GPIOs and their correct operation was implemented using the TIM3 hardware timer. All of these elements were combined into a system which provides the user with visual feedback to indicate the correct direction in which the board is be tilted to reach the chosen pitch angle, at which point the pitch angle is displayed in real-time.

References

[1] STMicroelectronics, *AN3182 Application note: Tilt measurement using a low-g 3-axis accelerometer,* 2010.

[2] A. Suyyagh, *ECSE426 - Microprocessor Systems Lab 3: MEMS Accelerometer, Timers and Interrupts,* Montreal, 2015.

[3] "LIS3DSH MEMS digital output motion sensor ultra low-power high performance three-axis “nano” accelerometer," October 2011. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00040962.pdf. [Accessed 16 March 2015].

[4] "RM0090 Reference manual STM32F405xx, STM32F407xx, STM32F415xx, and STM32F417xx advanced ARM-based 32-bit MCUs," STMicroelectronics, 2014. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf. [Accessed 21 February 2015].

Appendix

A1

Matlab script to calculate the calibration matrix (using the procedure described in the theory section):

```
n = 1000;
ones_col = ones(n, 1);
zeros_col = zeros(n, 1);

W1 = [csvread('position1.csv') ones_col];
W2 = [csvread('position2.csv') ones_col];
W3 = [csvread('position3.csv') ones_col];
W4 = [csvread('position4.csv') ones_col];
W5 = [csvread('position5.csv') ones_col];
W6 = [csvread('position6.csv') ones_col];
W = [W1; W2; W3; W4; W5; W6];

Y1 = [zeros_col zeros_col ones_col];
Y2 = [zeros_col zeros_col -ones_col];
Y3 = [zeros_col ones_col zeros_col];
Y4 = [zeros_col -ones_col zeros_col];
Y5 = [ones_col zeros_col zeros_col];
Y6 = [-ones_col zeros_col zeros_col];
Y = [Y1; Y2; Y3; Y4; Y5; Y6];

X = (W' * W) \ W' * Y;
```

A2

Matlab script for testing the Kalman filter parameters:

```
% 1D Kalman filtering of data using with initial Kalman state parameters
% p, r and q.
function [filtered] = KalmanFilter1D(data, p, ~, r, q)
    filtered = zeros(length(data), 1);

    % filtering
    x = data(1);
    for i = 1:length(filtered)
        p = p + q;
        k = p / (p + r);
        x = x + k * (data(i) - x);
        p = (1 - k) * p;
        filtered(i) = x;
    end

    % plotting result
    plot(1:length(filtered), data, 1:length(filtered), filtered, 'r');
    title(['Raw sensor data and Kalman filtered data (r = ', num2str(r),...
           ', q = ', num2str(q), ')']);
end
```