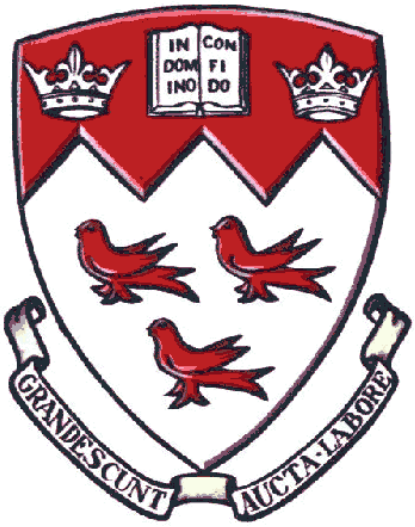
Experiment #2

Sensor Data Acquisition, Digitizing, Filtering, and Digital I/O

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



February 21st, 2015
Group 3

ECSE 426 Microprocessor Systems

Abstract

The goal of the experiment presented in this report is to implement a temperature data acquisition system using the STM32F407 Discovery board and display the acquired data through the use of the on-board Light Emitting Diodes (LEDs) in order to create a simple output display. This report will show how the built-in temperature sensor of the STM32F407 Discovery board, as well as the analog to digital converter (ADC) and LEDs were used to implement the desired system. It will also be shown how filtering of the raw data was done and how pulse width modulation (PWM) was utilized in order to realize the desired display effects.

Problem Statement

In this experiment, the internal processor temperature sensor of the STM32F407 Discovery board is used to get temperature readings of the microprocessor that will be converted into a visual LED display in order to let the user know if the temperature is increasing, decreasing or if the temperature has reached a certain threshold. The display is to be created using the LEDs that are positioned in a diamond shape on the board (i.e. LED 3 to LED 6). While in normal operation (i.e. below the temperature threshold), only one LED should be on at any one time. For each increase of 2 degrees Celsius, the display should cycle through the four LED lights in a clockwise fashion. In other words, if LED 3 is currently lit, after an increase of 8 degrees Celsius, LED 3 should be lit again. For every decrease of 2 degrees Celsius, the display should cycle through the four LEDs in a counter-clockwise manner. If the temperature of the microcontroller exceeds a certain temperature threshold, the display should enter an overheating alarm mode. The alarm mode consists of the four LEDs simultaneously flashing in a fade-in/fade-out manner. When the temperature falls back under the threshold, the alarm mode should be exited and normal mode should resume. Several challenges are associated with the LED display. While in normal operation, the transitions between LEDs should be as definitive as possible (i.e. the LED should ideally not flicker back and forth during a transition from one LED to the next). While in the alarm mode, all four LEDs must smoothly fade-in and fade-out from all the way off to fully on, in a cyclic manner. In the alarm mode, the LEDs should not be flickering on and off.

Theory and Hypothesis

According to [1], the temperature sensor of the microcontroller on the STM32F407 Discovery board is an analog sensor which outputs a voltage that varies linearly with temperature that ranges between 1.8V to 3.6V. The temperature sensor is internally connected to an ADC which allows the analog sensor readings to be converted to digital values. These digitized temperatures readings can then be used in an embedded C program to implement the LED output display. From the observation of the raw data of the digitized temperature readings outputted by the ADC shown in figure 1 and from the fact that sensor data is prone to noise [2], it can be seen that the temperature sensor output is significantly affected by noise. In particular, it can be seen in figure 1 that raw data samples can vary by more than 1℃ within only a few samples. Filtering of the raw data is therefore necessary.

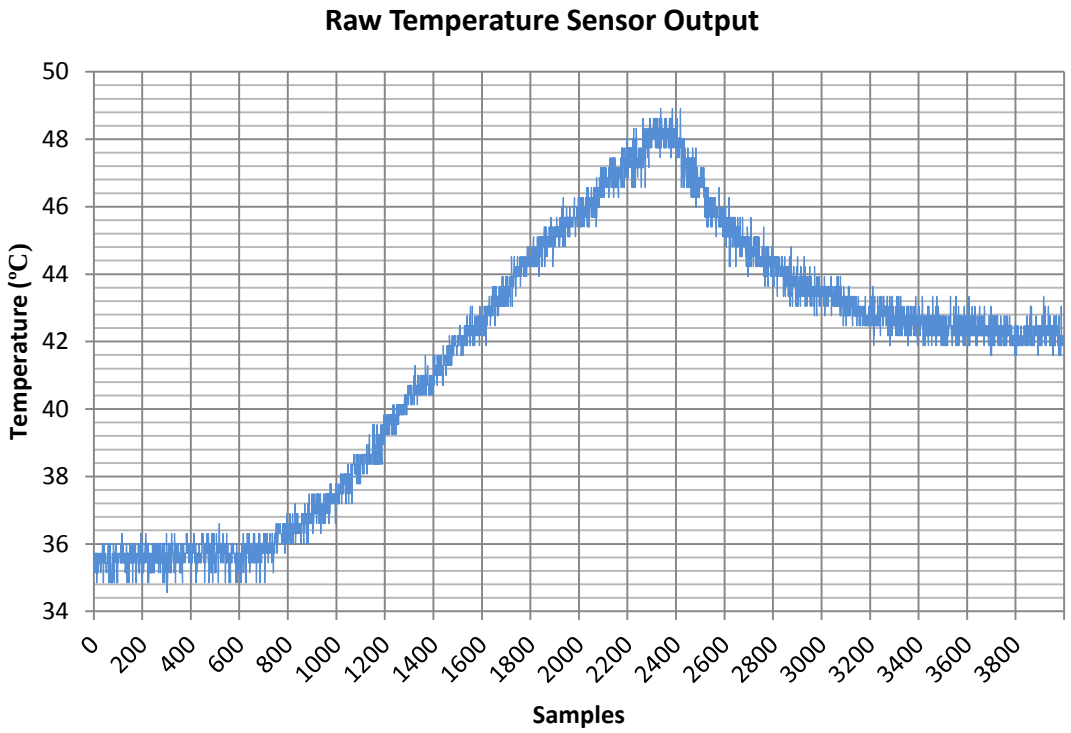


Figure 1: Digitized temperature sensor readings outputted from the ADC.

From the experiment 2 specifications [2], it is required to use a Kalman filter in order to filter the raw data. The Kalman filter is an optimal estimator for one-dimensional linear systems with Gaussian noise [3]. It is typically used to smooth noisy data and supply estimates of the filter's parameters [3]. From experiment 1 specifications [4], the parameters for the Kalman filter are the following:

$$q = process\ noise\ covariance \tag{1}$$

$$r = measurement\ noise\ covariance \tag{2}$$

$$x = value \tag{3}$$

$$p = estimation\ error\ covariance \tag{4}$$

$$k = kalman\ gain \tag{5}$$

These parameters need to be initialized to appropriate values before the filter can be used. To get a better insight into how to initialize some of these parameters it is useful to take a look at the equations governing each update of the Kalman filter which is performed every time a new measurement of the temperature is outputted from the ADC.

$$p = p + q \tag{6}$$

$$k = \frac{p}{p + r} \tag{7}$$

$$x = x + k(measurement - x) \tag{8}$$

$$p = (1 - k * p) \tag{9}$$

From a quick inspection of (6), (7), (8) and (9), it can be seen that the only 2 independent parameters (i.e. the parameters that are not changed by the updating process) are the process noise covariance (1) and the measurement noise covariance (2), therefore these parameters will have the most impact. From experimentation, the estimation error covariance (4) converges after only a few updates and its initial value is irrelevant. From equation (7), it can be seen that the Kalman gain's initial value is not important since as soon as the first update is performed, the parameter will be set to its appropriate filtering value as it only depends on the estimation error covariance (4), which, as mentioned above, converges after a few updates and the measurement noise covariance (2) which is constant. The parameter (3) is the actual filtered measurement value and its initial value can be set to the first temperature measurement produced out of the ADC. Therefore, the two parameters which have an important impact on the filtered data are the measurement noise covariance (2) and the process noise covariance (1). From the documentation of the Discovery board [5], it can be found that the precision of the temperature sensor is $\pm 1.5℃$ and consequently the measurement noise covariance will be equal to,

$$r = (\pm 1.5℃)^2 = 2.25℃^2 \tag{10}$$

The only parameter left to set is the process noise covariance (1) and since all the other degrees of freedom of the Kalman filter have been fixed or are irrelevant and finding the process noise covariance of a sensor is not trivial, the appropriate value for (1) can be set by trial and error on a set of raw data. The value of (1) can be varied until the filtered data exhibits the desired characteristics (i.e. the filtered data follows the local average values of the unfiltered data, and the filtered data's variation from one sample to the next remains small enough for the proper operation of the LED display. Otherwise the LEDs are going to flicker back and forth when the temperature approaches a transition value).

Implementation

The process of the implementation of the solution for the lab can be divided into roughly 5 parts: setup and initialization of relevant components, sensor data acquisition, conversion of acquired raw data to suitable format, data filtering and implementation of routines for the 2 required modes of operation.

Component setup and initialization

In the course of this lab, 3 components were used: the ADC, the General Purpose I/O (GPIO) and the SysTick timer. The use of these components requires their initialization and/or setup.

ADC: The ADC is used to convert the sampled analog voltage data from the processor's temperature sensor into a digital value (which is later to be converted into a temperature value). Initialization of the ADC involves the enabling of the high-speed bus (APB2) clock for the ADC1.

Afterwards, the common ADC structure, *ADC_CommonInitTypeDef*, was initialized. Since sampling was to be done only from a single channel, the ADC mode was set to independent mode (i.e. *ADC_Mode_Independent*) and the unneeded DMA access was disabled (i.e. *ADC_DMAAccessMode_Disabled*). Furthermore, the frequency of the clock the ADC was set to a division by 2 (i.e. *ADC_Prescaler_Div2*), which was deemed adequate for the purpose of this experiment.

Finally, the delay between 2 subsequent sampling phases was set to 5 clock cycles (i.e. *ADC_TwoSamplingDelay_5Cycles*), which is the smallest possible delay (however, since 12-bits resolution was used, this parameter could have also been set to a higher delay).

Afterwards, the specific ADC structure, *ADC_InitTypeDef*, was initialized. The highest available bit resolution of 12 bits (i.e. ADC_Resolution_12b) was used, to achieve the highest temperature resolution possible. Since the single mode was to be used, the scan and continuous conversion modes were disabled. As the external trigger wasn't needed for the samples' conversion, the field was set to none (i.e. *ADC_ExternalTrigConvEdge_None*). The data to set to be aligned to the right (i.e. *ADC_DataAlign_Right*). Finally, the number of conversions was set to 1, as only a single conversion of data at a time is required.

Once the common and specific ADC structures were initialized, the ADC1 module was enabled along with the temperature sensor, followed by the setting up of the temperature sensor channel 16 (also given by the macro *ADC_Channel_TempSensor*) and the ADC.

GPIO: In order to use the LEDs, it is essential to configure the relevant GPIO. Based on the board manual, GPIOD can be used to control the 4 LEDs [6]. In order to configure the GPIO, first the AHB1 peripheral clock was enabled. Then, the GPIO structure was initialized. In particular, the pins were set to the ORring of the GPIO pins 12-15, which correspond to the 4 LED pins. Naturally, the mode of operation was set to the output mode (i.e. *GPIO_Mode_OUT*), and the speed was set to 50MHz (i.e. *GPIO_Speed_50MHz*), which was deemed sufficient for the task at hand. The output type was set to push-pull (i.e. *GPIO_OType_PP*) and as there is no input, the operation mode for the pulling was disabled (i.e. *GPIO_PuPd_NOPULL*).

SysTick: The SysTick clock which was used for generating the sampling frequency was setup to the required 50Hz, by passing the ratio of the 168MHz / 50Hz to the SysTick configuration function. Internally, this ratio is used to generate an interrupt (calling the SysTick handler) every ratio clock cycles of the system clock.

Data acquisition

The temperature sensor data was acquired using the ADC1. Every time, after the SysTick handler was called, the *ADC_SoftwareStartConv()* function was called on the ADC1. Subsequently a wait loop was used until the end-of-conversion flag (i.e. *ADC_FLAG_EOC*) was set (indicating that the sample conversion has been done). The end-of-conversion flag was then reset (to allow for the next conversion to take place later) and the converted value retrieved using the *ADC_GetConversionValue()* function.

Conversion from voltage to Celsius

The acquired data from the sensor was converted from voltage to Celsius using the formula presented in the board reference manual [5]:

$$Temperature\ (℃) = \frac{V_{SENSE} - V_{25}}{AVG\_SLOPE} + 25 \tag{11}$$

where $V_{25} = 0.76\,V$ is the voltage at 25℃ and $AVG\_SLOPE = 2.5mV/℃$ is the average slope of the temperature vs. V_{SENSE} curve [1]. As the acquired data has a 12-bit resolution, it was necessary first to normalize it be dividing the obtained data by $2^{12} - 1 = 4095$. This value was then mapped into the 0 – 3

V range of the sensor by multiplying it by the maximum possible voltage value of 3 V. This value is what is referred to as V_{SENSE} in (11).

Filtering

As the acquired data is subject to noise, the raw sensor data exhibited a lot of sharp fluctuations. In order to rid the data from this noise, the data was filtered using a 1D Kalman filter. The selection of the 4 filter parameters was done part experimentally and part theoretically as described in the previous section. In order to establish, experimentally, the only unknown parameter q, Matlab simulations of the Kalman filter with raw sensor data for various values of q were performed. The results of these simulations can be found in the next section.

Temperature display and alarm routines

Each of the 2 distinct modes of operation were implemented as part of a separate routine. Namely, the normal operation routine for when the sampled temperature was below the selected threshold and the alarm operation routine for when the sampled temperature is above the threshold.

Normal operation: An array of size 4, which holds the 4 GPIO pins which correspond to the LEDs (i.e. pins 12 – 15) was defined. In order to light the correct LED, a mapping of the most recently sampled temperature into an index into this array using the following formula was performed:

$$LED\ index = \left(\frac{temperature}{2} + offset\right)\ mod\ 4 \tag{12}$$

Where offset is defined as:

$$offset = 4 * \left(1 + \left|\frac{temperature}{8}\right|\right), \tag{13}$$

with the division being integer division. The offset was used to account for the (somewhat unlikely) situation when the processor is operating in sub-zero temperatures (the temperature sensor is said to support temperatures as low -40℃ [5]). For each decrease of 8℃ another 4 is added to the offset in order to bring the term in the (12) into the 0 – 3 range. The division by 2 of the temperature in (12) ensures that the index is incremented/decremented for each 2℃ rise/fall. Finally, the modulus operation ensures the LED index is mapped into a valid index (i.e. 0 – 3).

Alarm operation: When the processor was found to be overheating, an alarm in the form of 4 LEDs fading in and out was implemented in a routine using a PWM-based technique to vary the intensities of the LEDs. One of the essential parameters of PWM is the chosen period. The system was designed to work with the SysTick timer. That is, a period X for PWM, corresponds to X clock cycles of the SysTick. It was observed that the PWM period should be very short in order to reduce the "blinking" effect. Therefore, after testing multiple values, a period of 4 was chosen. That is, the PWM period was 8ms (4 * 2ms), corresponding to a frequency of 12.5 Hz (50 Hz / 4). Originally, the plan was to increase/decrease the duty cycle after every PWM period. However, it was found that repeating a duty cycle multiple times (in our case, 3 times) before modifying the duty cycle leads to a more "smooth" fade in/out patterns, as it makes the transition from one duty cycle to another (thus from one intensity to another) slower and thereby producing a more "fluid" transition effect between the LED intensities. Therefore, after every 24ms, the duty cycle was incremented/decremented (based on the value of a variable which indicated whether the duty cycle is to be incremented or decremented). When the duty cycle reached the maximum

level (i.e. the period of the PWM), the variable was set to decrement. Similarly, when the duty cycle was the minimum (i.e. 0), the variable was set to increment.

To conclude, inside the main function a function to initialize the ADC, GPIO and SysTick components is called. Afterwards, a loop is set such that after every time the SysTick handler is called, a new temperature is sampled and converted into a digital value by the ADC. This value is then converted into a temperature reading (in Celsius) and filtered using a Kalman filter. Then, depending whether the temperature is above or below the pre-selected threshold, a routine corresponding to the normal or alarm operation is called. This process repeats until the system is powered off or stopped.

Testing and Observations

The testing phase included the testing of the selection of the Kalman filter parameters and overall system behavior.

Kalman filter testing

A Matlab function was written (see Appendix) that performs Kalman filtering on a vector of data. A total of 4096 raw sensor data samples were collected and then filtered with the aforementioned Matlab function. The results were then plotted and compared.

The following 4 plots showcase the effects of the varying the Kalman state parameter r (while keeping the other ones constant):

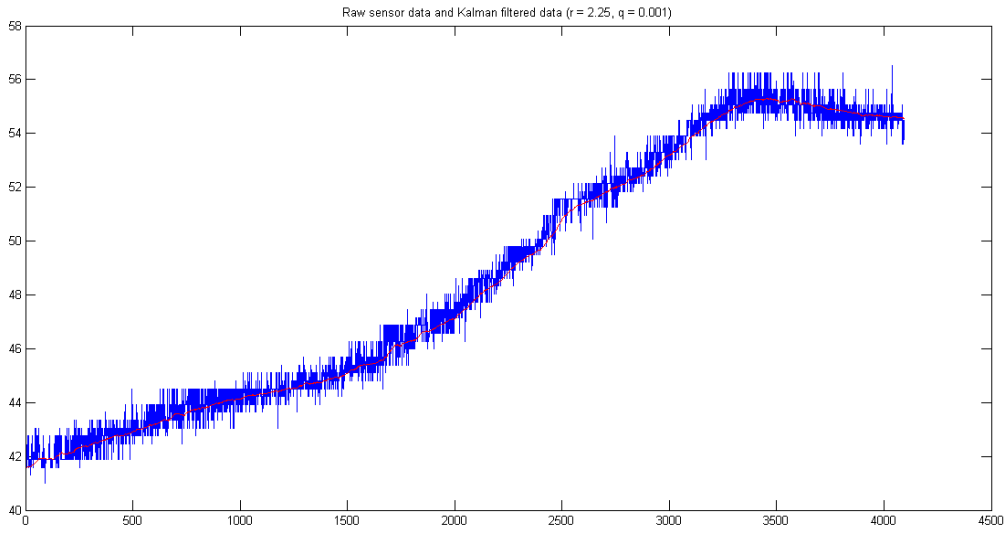


Figure 2: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 2.25 and q = 0.001.

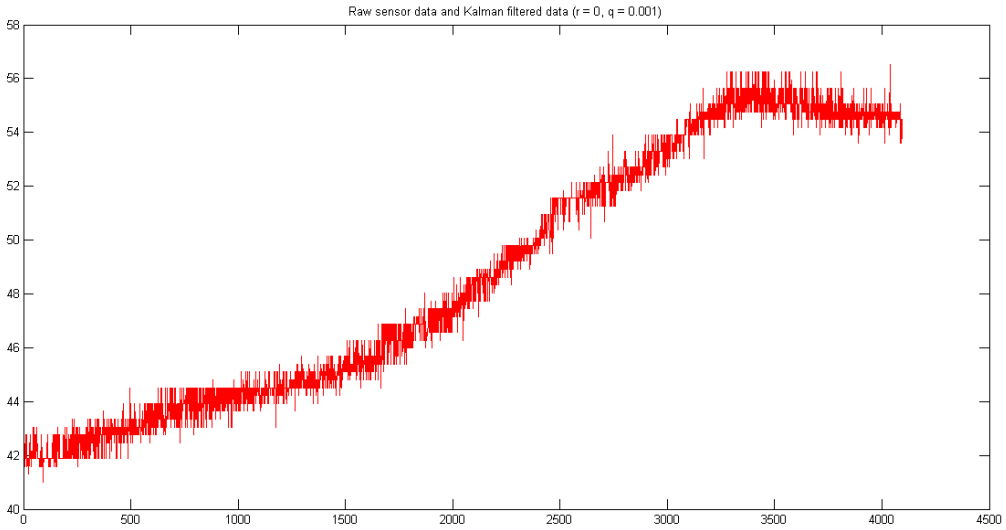


Figure 3: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 0.0 and q = 0.001.

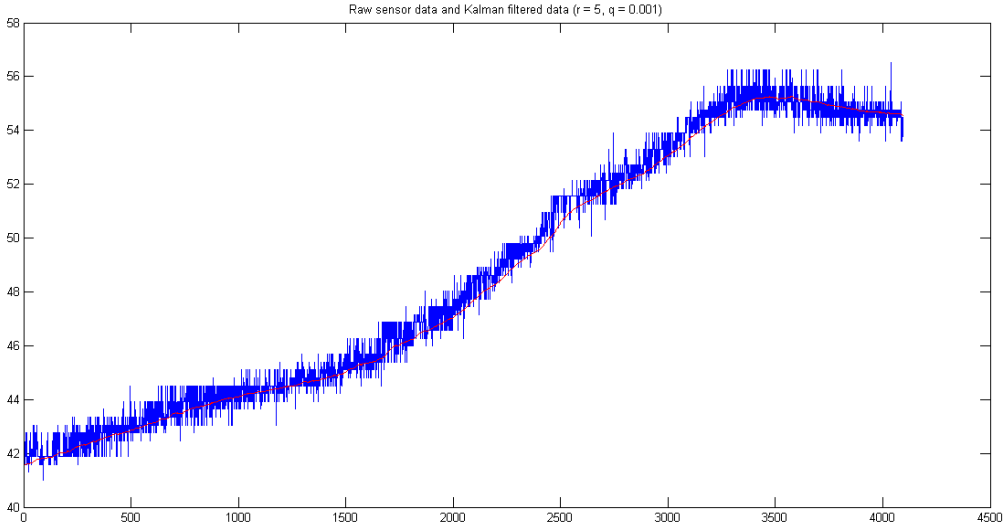


Figure 4: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 5.0 and q = 0.001.

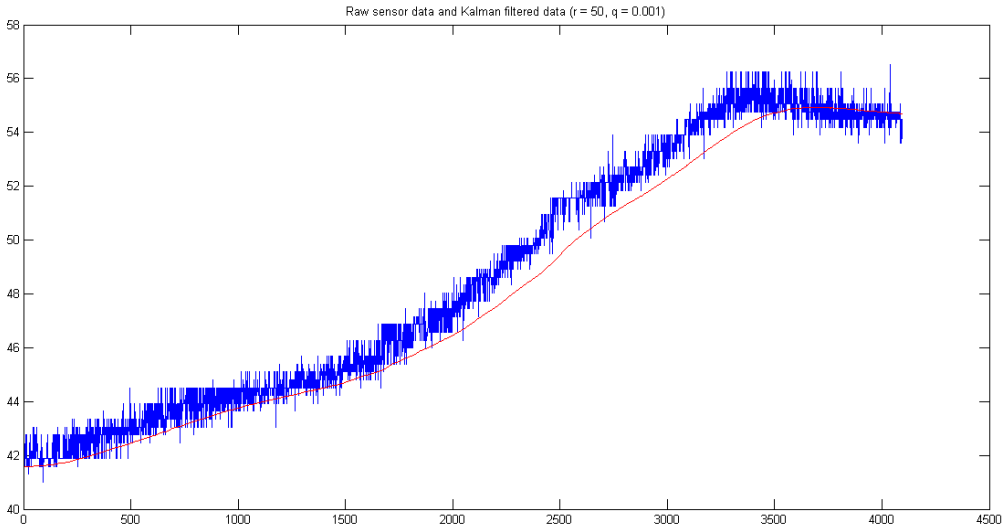


Figure 5: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 2.25 and q = 0.001.

The following 3 plots showcase the results of the Kalman filter for various initial parameter q (while keeping the other ones constant and in particular, r = 2.25):

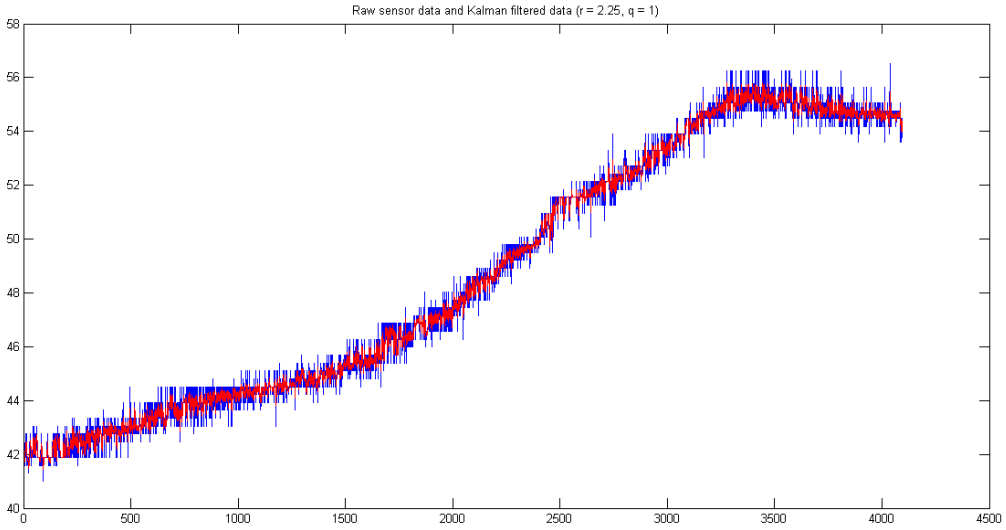


Figure 6: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 2.25 and q = 1.0.

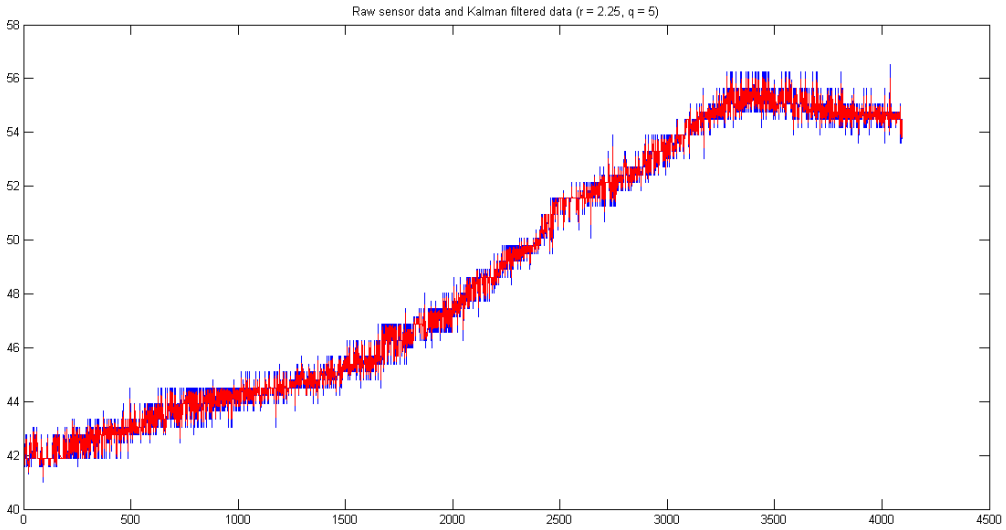


Figure 7: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 2.25 and q = 5.0.

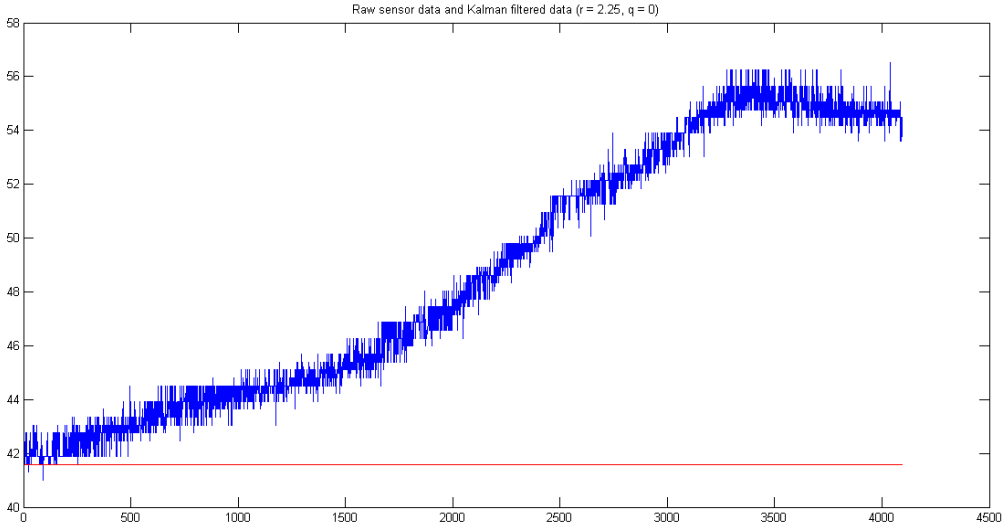


Figure 8: Kalman filtered data (red) and raw sensor data (blue) using a Kalman filter with initial parameters r = 2.25 and q = 0.0.

Using this function it was also confirmed that varying the parameter p has a very minor effect on the overall performance of the filter over many samples. In particular, the results of filtering with various parameters p were found to be quite similar.

Based on these observations and the theoretical value for r, it was concluded that using the parameters r = 2.25 and q = 0.001 (while arbitrarily setting the other ones), provided for an adequate filter setup. In particular, using these parameters, the filtered data appeared to very closely follow the general trend of the sampled data, while remaining smooth.

System testing

A print function was set in the code (relaying on the provided code for redirecting the printf() function to the debug port), in order showcase the temperature in real time as the program runs. The program was then run on the microcontroller in debug mode. The temperature of processor was varied using the hair drier. It was observed that the LEDs were turning on/off as expected (based on the printed temperature values). When reaching the threshold temperature, it was observed that, as expected, the operation was switched into the alarm mode. Once the processor has cooled down to below the threshold temperature, the operation was switched back to the normal mode. Overall, after repeating the experiment multiple times, it was observed that the operation of the system corresponded to the requirements.

Conclusion

In the course of this lab, using the STM32F407 Discovery board, a system that acquires temperature readings from the on-board temperature sensor and provides visual feedback to the user was implemented. After acquiring temperature readings and converting them to a digital format via the ADC, the results were converted into a temperature format. In order to reduce the effects of noise, the raw data was filtered using a Kalman filter, whose initial parameters were found by combining theoretical calculations and simulations. Based on the obtained, filtered temperature value, the user was provided with a visual feedback using the 4 on-board LEDs which were controlled via a GPIO. In the normal mode of operation the user saw various LED lighting up based on the increase or decrease in temperature. Once the temperature exceeded a certain threshold, the user saw an alarm in the form of 4 fading in/out LEDs, whose intensities were varied using PWM. Through a series of tests, it was concluded that the behavior of the system appeared to be in line with the requirements. The successful implementation of this experiment, provided our group with a solid introduction to the methods and techniques for interfacing with the various on-board hardware components.

References

[1] "STM32F405xx/STM32F407xx Datasheet - production data," STMicroelectronics, 2013. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00037051.pdf. [Accessed 21 February 2015].

[2] A. Suyyagh, "ECSE 426 Microprocessor Systems Lab 2: Sensor Data Acquisition, Digitizing, Filtering, and Digital I/O," 2015.

[3] R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation," *IEEE Signal Processing Magazine,* pp. 128-132, 20 August 2012.

[4] A. Suyyagh, "ECSE 426 Microprocessor Systems Lab 1: One-Dimensional Kalman Filter," 2015.

[5] "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].

[6] "UM1472 User manual Discovery kit for STM32F407/417 lines," STMicroelectronics, 2014. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf?s_searchtype=keyword. [Accessed 21 February 2015].

Appendix

Matlab code used 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
```