Lab 2:

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

ECSE 426 Microprocessor Systems

Group 18

Chuan Qin 260562917

Wei Wang 260580783

February 22, 2016

Table of Contents

1 Abstract

The purpose of this experiment is to read data from temperature sensor, display it on a four digit 7-segment and LCD displays, and building an alarming mechanism using four different LEDs of the STM32F4-Discovery. The sampling frequency of readings is set to be 100Hz, which can be accomplished by using SysTick timer. Since that the signals obtained from Analog-to-digital converter (ADC) are too noisy to be used directly, the Kalman filter with an appropriate set of parameters is in use for data fusion.

2 Problem Statement

Temperature sensor is one of the common component in modern processors for detecting the operating temperature and cool the processor down if it is overheating. However, this sensor is not being used for that purpose in the experiment. This project aimed at retrieving the data readings at a certain rate, filtering the measurements, converting the results before displaying them using 7-segment display and LCD, and setting an alarm indicating when the temperature is too high.

First, the sensor sends the temperature values in analog voltage. The use of the analog to digital converter can digitize the data, and some different forms of noise such as electromagnetic noise, thermal noise, and quantization noise can be introduced during the conversion. Kalman filter is implemented to produce the estimate for a linear system by filtering noisy signals. By setting an appropriate frequency and filtering the noise, it increases the data accuracy.

Second, the data readings are be displayed using a four digit 7-segment display and LCD. The temperature of operating process is smoothly increasing, and when the LEDs on the board are going to turn on and off one by one in a circle, it signifying an alarm indicating the data crosses a threshold. Then, the lights are off once it drops below a given temperature.

There are few challenges encountered while dealing with the specifications listed above. They are converting the readings derived from sensor into degree Celsius for display, determining the initial state for Kaman filter, finding the speed of which the flickering problem is invisible, and the necessity of avoiding the false alarm.

3 Theory and Hypothesis

Temperature meter is expected to have evolved in term of steady, accuracy, and reliability as a consequence of need for enhanced technology. The temperature sensing system should able to deal with the following four concerns to be sensitive and responsive.

- Data in voltage to Celsius conversion
- Calculation of initial Kalman filter state

- Refresh rate of 7-segment and LCD displays
- Minimum duration of consecutive overheating values

4 Implementation

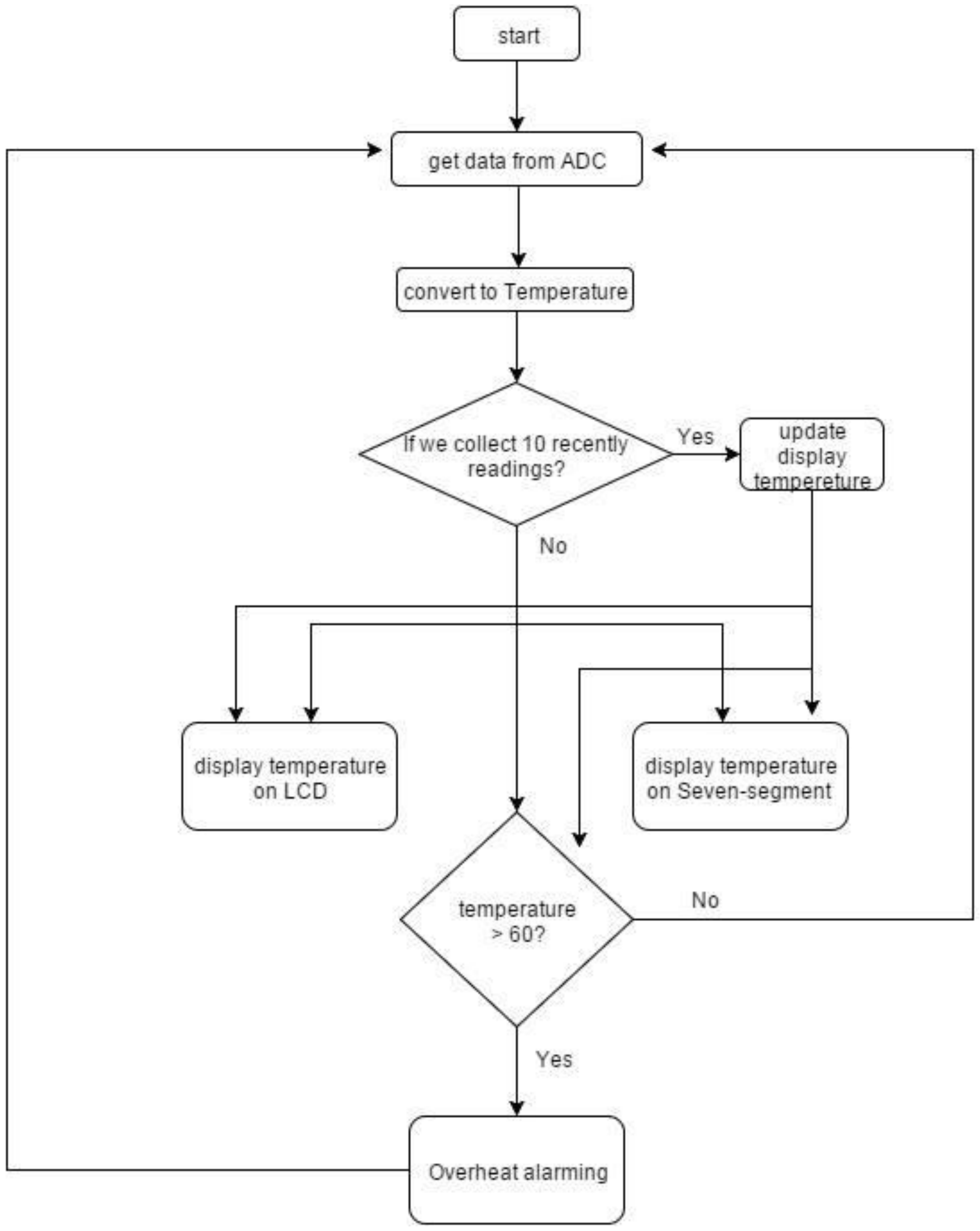
start
get data from ADC
convert to Temperature
If we collect 10 recently readings?
Yes
update display tempereture
No
display temperature on LCD
display temperature on Seven-segment
temperature > 60?
No
Yes
Overheat alarming

Figure 1 – The System Architecture Diagram

4.1 Analog to Digital Converter (ADC)

The system generates many analog signals including temperature, which need to be converted into digital data using ADC peripheral. It is easier to use standard peripheral functions. First, we enable ADC clock and choose prescaler to allow ADC to work at the board clock frequency (APB2CLK) divided by 6 (84MHz / 6), which is equal to 14MHz, because the maximum frequency of the ADC analog clock is 36 MHz [1]. Also, the input voltage is converted into a 12-bit number given a maximum value of 4095, so the resolution should be 12-bit.

```
__HAL_RCC_ADC1_CLK_ENABLE();
ADC1_Handle.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV6;
ADC1_Handle.Init.Resolution = ADC_RESOLUTION_12B; [3]
```

Next thing is to select single conversion mode for ADC [3]. It means that ADC does only one conversion and then stop in this case. The readings given by ADC Conversion is stored into a 16-bit register. Once it is done, the End of Conversion flag (EOC) is set. So, ADC will no longer do any conversion until we ask for. We right justified that data which indicates that LSB is located at the rightmost bit [2].

```
ADC1_Handle.Init.ExternalTrigConv = ADC_SOFTWARE_START;
ADC1_Handle.Init.NbrOfConversion = 1;
ADC1_Handle.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

Since the single conversion mode is in use, we can disable other modes such as scan mode, continuous conversion mode, and discontinuous conversion mode. Then, we need to select ADC Channel 16 because it is where temperature sensor is internally wired to [2], [3].

```
ADC_Channel.Channel = ADC_CHANNEL_16;
ADC_Channel.SamplingTime = ADC_SAMPLETIME_15CYCLES;
```

4.2 Sampling Frequency

The entire system is set at 1MHz; it means one tick is one microsecond. The reason of setting such large number of frequency for SysTick configuration is because of the default unit of time for LCD implementation is microsecond which will be discussed later in section 4.4.2. On the other hand, the sampling frequency is required to be 100Hz that indicates we obtain a reading from sensor every 10 milliseconds. To accomplish this, we declare one extern integer variable called TimingDelay, expecting to decrement by one per one tick. We consider this variable as a timer; once it reaches zero, the system will ask ADC to do conversion. At the beginning of the algorithm, TimingDelay is initialized to be 0, so we can get a data from ADC right after the program is executed. For every iteration, TimingDelay is equal to 10000 ticks from the start, and then decreases. Therefore, the system can get a reading from ADC every 0.01 second.

4.3 Kalman Filter

Kalman filter is an algorithm applied after retrieving readings from ADC for the purpose of minimizing the effects of measurement error. The implementation of this type of filter is based on the immediate last state and the observed physical result.

Table 1 – Kalman Filter state structure parameters

Kalman Filter State Parameters	Variables
Process Noise Covariance	q
Measurement Noise Covariance	r
Value	x
Estimation Error Covariance	p
Kalman Gain	k

The algorithm can separated into two types: prediction equations and update equations. The first type predicts the current state for the input according to the previous state values as Equation 1 and 2 in [4].

$$p_x = p_{x-1} + q_{x-1} \tag{1}$$

$$k_x = \frac{p_x}{p_x + r_{x-1}} \tag{2}$$

$$where\ x\ presents\ the\ state\ number$$

The second part is to calculate the filtered result using the estimate parameters and the input measurement given by the system. Then, this update process is repeated continuously.

$$x_x = x_{x-1} + k_x(measurement - x_{x-1}) \tag{3}$$

$$p_x = (1 - k_x)p_x \tag{4}$$

Later in the experiment, we need to define the parameters which give the best results. To test different values of parameters, we use MATLAB to generate graphs of inputs and outputs for observations. Before the test, we use MATLAB to produce a set of filtered data, and compare to the ones calculated using C implementation. In Table A-1, the difference between two types of operations is small enough that can be ignored.

4.4 Temperature Displays

There are three main steps required before the display: converting the temperature to degree Celsius, configuring the GPIO for different type of display, and connecting the display pins to its corresponding GPIO pins. Both display take in and show the average of every ten recent data.

The temperature values converted by ADC are in voltage format. Note that the data returned from ADC is not equivalent to V_{sense}. We need to determine the maximum 12-bit value would occur at the supply voltage of 3V, which can be defined by following equation:

$$V_{sense} = \frac{3000mV}{4095} \cdot value \qquad (5)$$

$$where\ value = the\ temperature\ reading\ obtained\ from\ ADC$$

We can convert readings into Celsius format using the Equation 6 in [1].

$$Temperature\ (℃) = \frac{V_{sense} - V_{25}}{Average\ Slope} + 25 \qquad (6)$$

$$where\ the\ reference\ voltage\ at\ 25℃\ (V_{25}) = 0.76\ V$$
$$and\ Average\ Slope = 0.0025\ V/℃$$

There are multiple GPIO pins on the board. These pins can be programmed as input; they can also be used to perform outputs. In order to be able to configure pins of a specific port for different types of display, we must first enable its pin clock. Then, we define the pins' mode to be push-pull output since there is no external voltage applied, activate pull up that configures inputs to be high, select clock speed between range 25MHz to 100MHz, and initialize the pins required for displays.

4.4.1 Four Digit Seven-segment Display

In the common cathode display, all 7-segment data lines are connected together to ground. Each segment, labelled from A through to G in Figure A-1 [6], can be selected individually by a high signal. More specifically, the segment will be dark when it is set to be '0', and it will be light when it is set to be '1'.

There are 12 resistors of 390 Ohm added while connecting the wire from GPIO pins to the 12 pins (7 segments, one decimal point, and 4 digit) of display. The purpose of the resistor is to protect the display by reducing current flowing to a segment. In Table 3, we set individual data lines to '1' in order to produce various numbers from 0 through 9. As shown in the Figure A-1, we notice that each digit contains a pin for decimal point (DP). We connect DP pin to GPIO Port E Pin_14. According to the values of temperature, there is only one decimal point required for the performance. The DP pin will be light with ones digit when it is on. Furthermore, we connect a transistor to each common cathode terminals as a switch to specify which digit should be illuminated. Each digit must perform sequentially from left to right because all four digits share the same segment pins. We need to extract each digit from a given output and display it accordingly (Table 2). As the frequency is 100Hz, the program spends approximately 2 milliseconds to execute all the instructions beside segment display through debugging. We assume that it will take about 8 milliseconds to display. The time delay between each digit performance should be at most 2 milliseconds. So, we choose 1.9 milliseconds as delay time for each digit display.

Table 2 – The Common Cathode Pins Configuration using GPIO Port C

Terminal	Digit 1	Digit 2	Digit 3	Digit 4
Digit	Tens	Units	Tenths	Degree
Pin Number	Pin_0	Pin_1	Pin_2	Pin_3

Table 3 – Seven-segment Display Code and Pins Configuration using GPIO Port E

Number	A	B	C	D	E	F	G
	Pin_7	Pin_8	Pin_9	Pin_10	Pin_11	Pin_12	Pin_13
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
Degree Symbol	1	1	0	0	0	1	1

4.4.2 LCD Display

For a 2x16 LCD display, the first two pins are connected to ground and +5V, respectively. To be able to adjust the contrast of LCD display, we build a voltage divider by using a potentiometer and connect the left end to ground and right end to VDD. And the middle end is connected to the pin of Vee in LCD display. The connections between LCD pins and GPIO pins are presented in Table 4. To provide backlight, we also connect last two pins of LCD display to +5V and ground accordingly [4].

We write several methods to issue different command or pass data to LCD display. In each of them, we firstly send a value to the LCD by setting the values of D7-D0 (ASCII value for data or a hexadecimal value for command). The numeric digit code is listed in Table 5. Then, we set the value of R/W to ‘0’ and the value of RS to ‘1’ when sending data, or ‘0’ when sending a command. Also, we generate an “Enable” pulse for 10 ticks, equivalent to 10 microseconds. Finally, we create a delay of 37 microseconds (1.52 milliseconds for command “clear display”) to give enough time for display to carry out the command or display the character [4]. The unit of delay time for LCD is the reason that the system’s frequency must set to be at least 1MHz.

After LCD configuration, we execute these 3 commands: function set to 8-bit mode, display on and cursor on, and character displayed from left to right and no shift occurs when character is ‘0’. Every time before display the temperature on LCD, we need to clear the display and set the starting display address to 5, so the number is performed in the middle of the first line on LCD.

Table 4 – LCD Control Pins Configuration using GPIO Port C

LCD	RS	Read or Write	Enable
Pin Number	Pin_4	Pin_5	Pin_6

Table 5 – LCD Character Code and Pins Configuration using GPIO Port C []

Number	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
	Pin_14	Pin_13	Pin_12	Pin_11	Pin_10	Pin_9	Pin_8	Pin_7
0	0	0	1	1	0	0	0	0
1	0	0	1	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	0	1	1	0	1	0	0
5	0	0	1	1	0	1	0	1
6	0	0	1	1	0	1	1	0
7	0	0	1	1	0	1	1	1
8	0	0	1	1	1	0	0	0
9	0	0	1	1	1	0	0	1
Decimal Point	0	0	1	0	1	1	1	0
Degree Symbol	1	1	0	1	1	1	1	1
Celsius	0	1	0	0	0	0	1	1

4.5 Overheating Alarm

The four different colors of LED are controlled by some pins listed in the table below. They are expected to be turned on in circular motion and in succession when the temperature is above 60℃. In the beginning, we initialize a counter which is increased by one for one tick. To remove the effect of some bad value or error produced by noise, the LEDs begin to shine only when the temperature has been greater than the threshold for at least 50000 ticks, which is 0.05 second. So the alarm will not react for a single high temperature result.

The duration for each LED to be switch on is 5000 ticks, also known as 5 milliseconds. The green LED will be set to high first. Then, it will be set to low, and the orange LED will become high for the next 5000 ticks. The red LED is the third one that will be set to be on, and the blue one later. This turn on and off process is continuously repeated until the temperature drops below 60℃. Once the temperature of the process is back within the safe range, all the four LEDs will be turned off.

Table 6 – LEDs Pin Configuration using GPIO Port D [5]

LED colors	Green	Orange	Red	Blue
Pin Number	Pin_12	Pin_13	Pin_14	Pin_15

5 Testing and Observations

According to the initial values of parameters given in lab 1, we can see that the filtered outputs still have large noises in Figure 2. We should determine the better values for each filter parameters to minimize the error caused by noise. To achieve this goal, we can first find out the relationship between the each parameter and filtered data by controlling and testing different variables. Instead of testing within a small range of temperature, we use hairdryer to heat up the processor for better reading observations, which increases the range up to around 65℃.

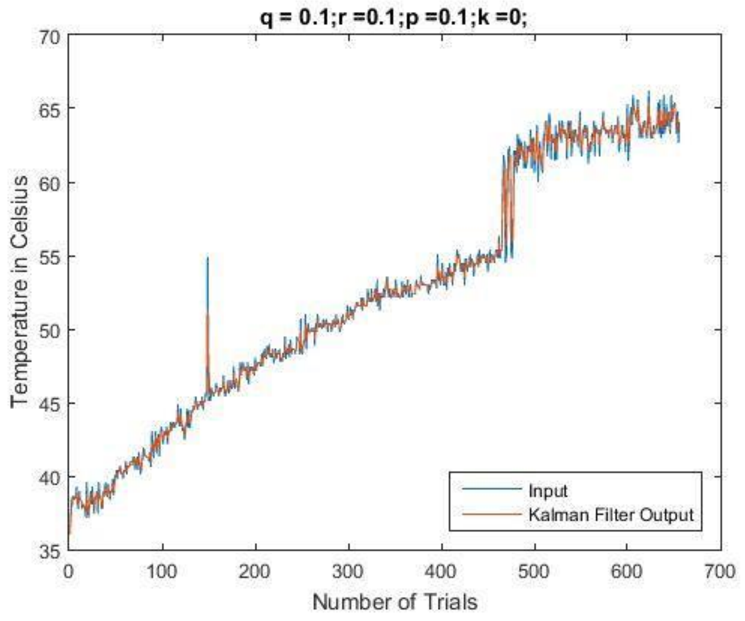

Figure 2 – Performance of Kalman filter using original parameters

We start by adjusting the values of p (estimation error covariance) and k (adaptive tuning factor), there is no obvious change indicating that the noise has been filter out even if p and k becomes a large number. As shown in Figure A-2, we multiply p value by 10. Then, we assume that the initial Kalman gain is 10 in Figure A-3. Comparing both results to Figure 2, we notice that the filter data plots are similar.

Furthermore, we try to make q become 0.2, which is little higher than the original value. In Figure 3, there are few less noise being reduced related to the all three figures above. However, the filtered data are more stable when the process noise is half of the initial one. If q becomes an extremely small, then the filtered output no longer follow the inputs, which means it leads to its insensitivity (Figure A-4, A-5).

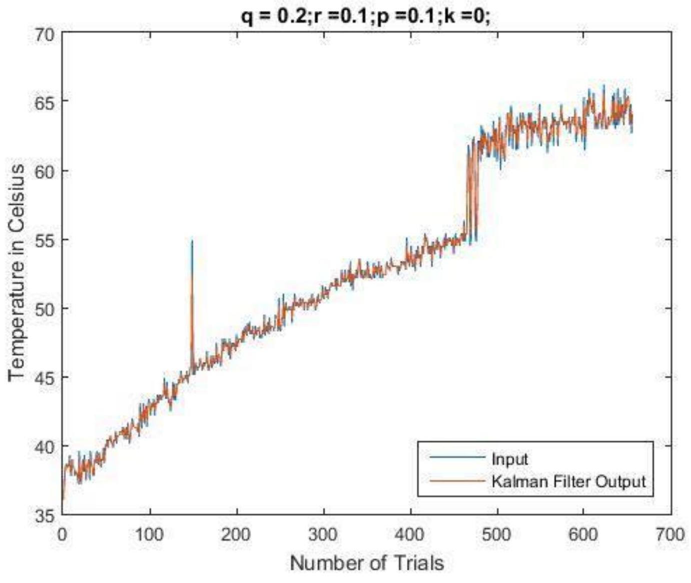

Figure 3 – Performance of Kalman Filter with process noise covariance of 0.2 (q)

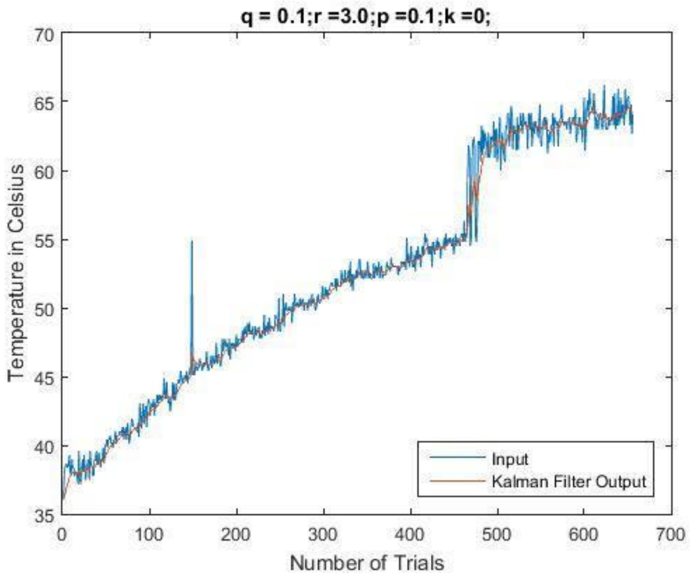

Figure 4 – Performance of Kalman Filter with measurement noise covariance of 3.0 (r)

Based on the graphs above and those in Appendix (Figure A-6, 7), we observe that if the measurement noise covariance gets higher, the plot of filtered results is similar to the plot of decrementing the process noise. For example, r is now equal to 10 times of the original value. Although, there is a limit for r. If r is as large as 50, the output is no more accurate. Therefore, the output is following the track of inputs and also cancelling error caused by noise when q = 0.05 and r = 3.0 without changing any p and k in Figure 5.

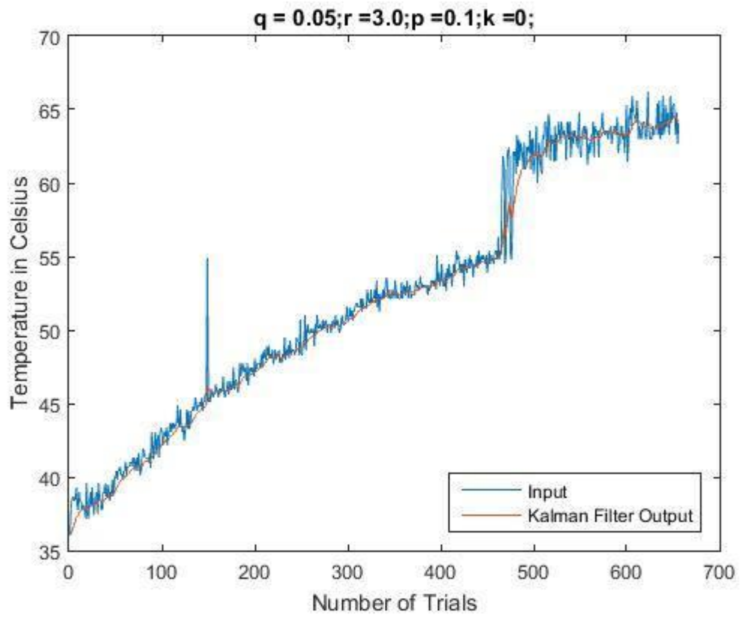

Figure 5 – Performance of Kalman Filter with appropriate parameters

6 Conclusion

In this experiment, we manage to design a monitor for temperature gauge. The first thing is to get one reading from built-in temperature sensor at 100Hz using ADC peripheral on the STM32 processor. The ADC is set to perform a single conversion on a single channel. When the conversion is complete, we implement software filter as Kalman filter to reduce noise because every sample from an ADC is a combination of signal and noise. The initial filter parameters are q, r, x, p, and k which is equal to 0.05, 3.00, 0.00, 0.10, and 0.00, respectively. Once we start the system, the first temperature reading is stored in the variable x. Due to floating point computation, the results calculated using MATLAB and C implementations are slightly different.

The second part is to show the temperature in Celsius format on four digit 7-segment display and on LCD. Four digit 7-segment display perform data in a sequential fashion, but LCD conversely can display the entire value at once. Both update rates are lower than the sampling rate, so an average of every 10 readings is sent to both displays. We alter the transition rate of about 2 milliseconds for 7-segment display to make the flickering fast enough to fool human's eyes.

The last step is to design an overheating alarm. When the temperature is higher than the threshold, 60℃, for at least 0.05 seconds, four different colors of LED will switch on one by one in a circular fashion. The lights keep rotating until the processor cool down and the temperature goes below the threshold. Overall, all the four issues are being handled and the temperature meter is sensitive, stable, and accurate.

7 Appendix

Table A-1 – Comparison of First 50 Results of Temperature in Celsius and its Difference

No.	Measurements	Filtered Outputs		Differences
		C	MATLAB	
1	36.604404	24.402936	24.402935	1.00E-06
2	37.190483	32.39515288	32.395153	-1.25E-07
3	37.190483	35.36369057	35.363689	1.57E-06
4	37.776566	36.85528629	36.855286	2.91E-07
5	37.190483	37.06245648	37.062458	-1.52E-06
6	37.776566	37.50380268	37.503803	-3.18E-07
7	37.190483	37.31016033	37.310162	-1.67E-06
8	36.018326	36.51176272	36.511765	-2.28E-06
9	37.776566	37.29345415	37.293453	1.15E-06
10	37.190483	37.22981448	37.229813	1.48E-06
11	36.897446	37.02439946	37.024399	4.62E-07
12	37.483524	37.30815403	37.308155	-9.68E-07
13	37.483524	37.41653863	37.416538	6.33E-07
14	37.776566	37.63904778	37.639046	1.78E-06
15	37.190483	37.3618195	37.36182	-4.99E-07
16	37.190483	37.25592772	37.255928	-2.80E-07
17	36.897446	37.03437383	37.034374	-1.67E-07
18	37.776566	37.49307382	37.493073	8.18E-07
19	37.776566	37.66828162	37.668282	-3.78E-07
20	37.776566	37.73520505	37.735207	-1.95E-06
21	37.483524	37.57965761	37.579659	-1.39E-06
22	36.897446	37.15802765	37.158028	-3.54E-07
23	37.483524	37.35919546	37.359196	-5.44E-07
24	38.069607	37.79825394	37.798252	1.94E-06
25	37.776566	37.78485005	37.784847	3.05E-06

26	37.483524	37.59862031	37.598618	2.31E-06
27	37.483524	37.52748688	37.527485	1.88E-06
28	37.776566	37.68142624	37.681427	-7.58E-07
29	37.190483	37.37800663	37.378006	6.32E-07
30	37.483524	37.44321995	37.443218	1.95E-06
31	36.018326	36.56258706	36.562588	-9.41E-07
32	37.776566	37.31286731	37.312866	1.31E-06
33	37.190483	37.23722965	37.237228	1.65E-06
34	37.483524	37.38944793	37.389446	1.93E-06
35	37.776566	37.62870005	37.6287	5.41E-08
36	37.190483	37.35786702	37.357868	-9.80E-07
37	37.190483	37.25441801	37.254417	1.01E-06
38	37.483524	37.3960133	37.396015	-1.70E-06
39	36.604404	36.90677185	36.906773	-1.15E-06
40	37.190483	37.08211498	37.082115	-1.79E-08
41	36.311363	36.60576406	36.605762	2.06E-06
42	37.483524	37.14824954	37.148251	-1.46E-06
43	36.311363	36.63102521	36.63102729	-2.08E-06
44	37.776566	37.33900835	37.589787	-0.250778645
45	38.069607	37.79054315	37.886333	-0.09578985
46	37.483524	37.60079488	37.637383	-0.03658812
47	36.018326	36.62277533	36.636749	-0.013973674
48	38.069607	37.51696648	37.522305	-0.005338523
49	36.018326	36.59075573	36.592796	-0.002040275
50	36.311363	36.41808152	36.418861	-0.000779475

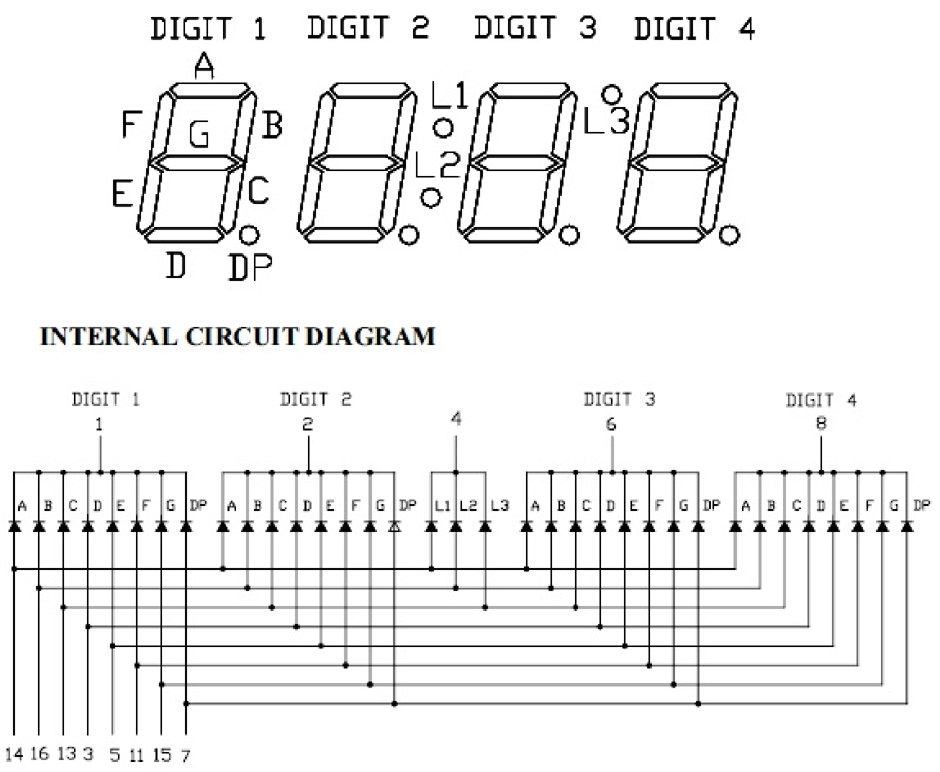

Figure A- 1 – Four digit segments arrangement [6]

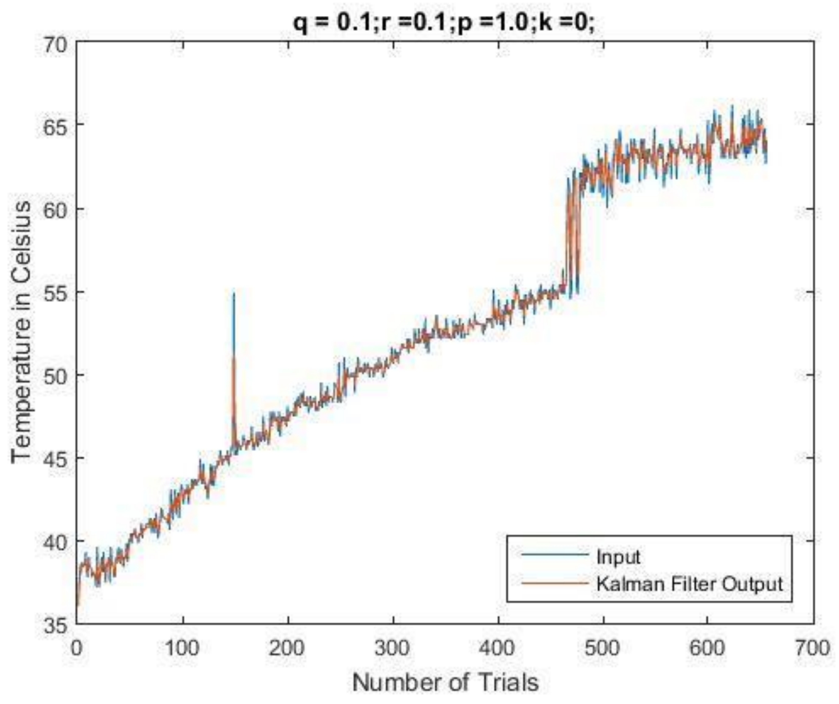

Figure A-2 – Performance of Kalman Filter with new estimation error covariance (p)

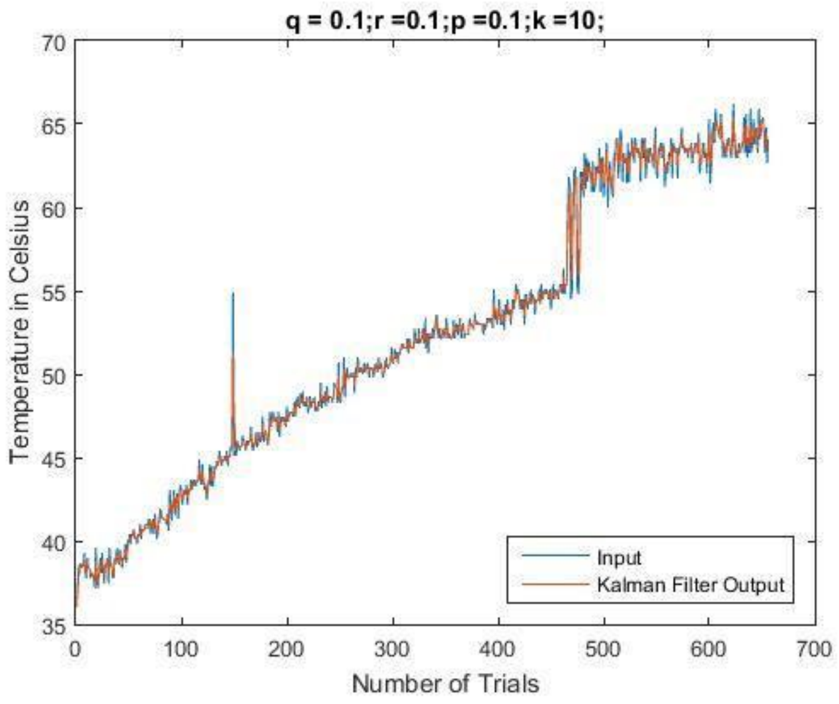

Figure A-3 – Performance of Kalman Filter with new gain of 10 (k)

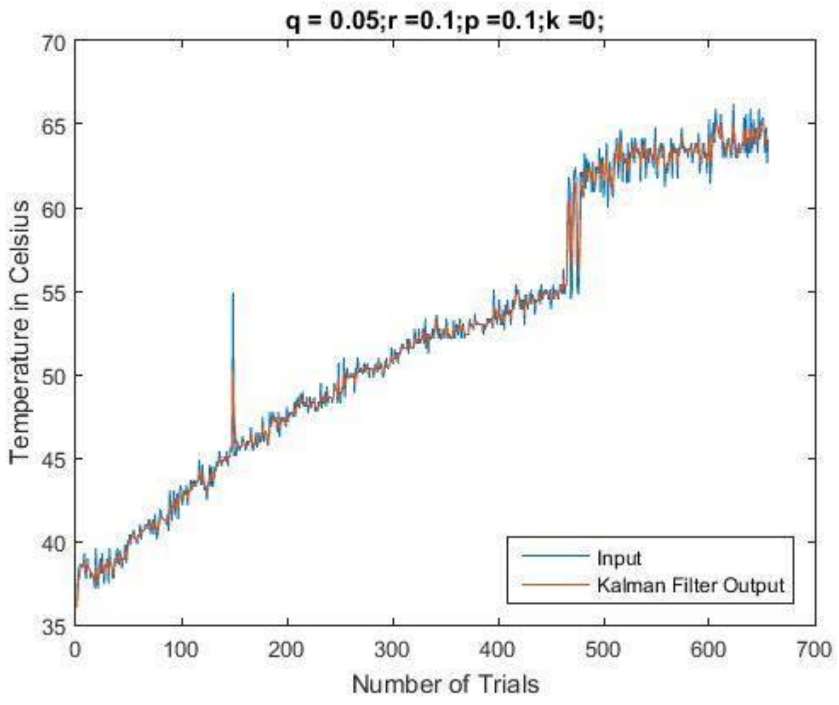

Figure A-4 – Performance of Kalman Filter with process noise covariance of 0.05 (q)

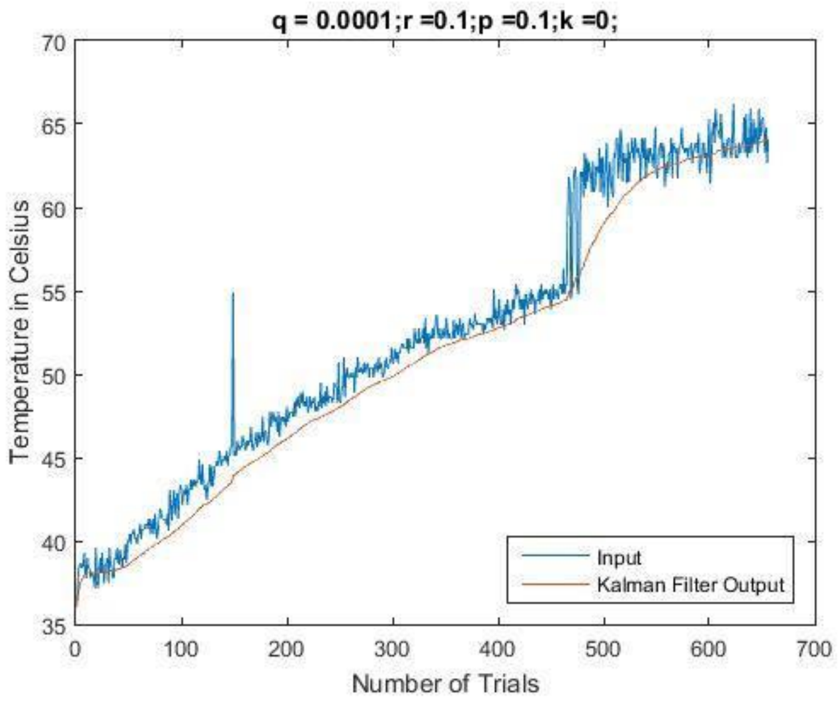

Figure A-5 – Performance of Kalman Filter with small process noise covariance (q)

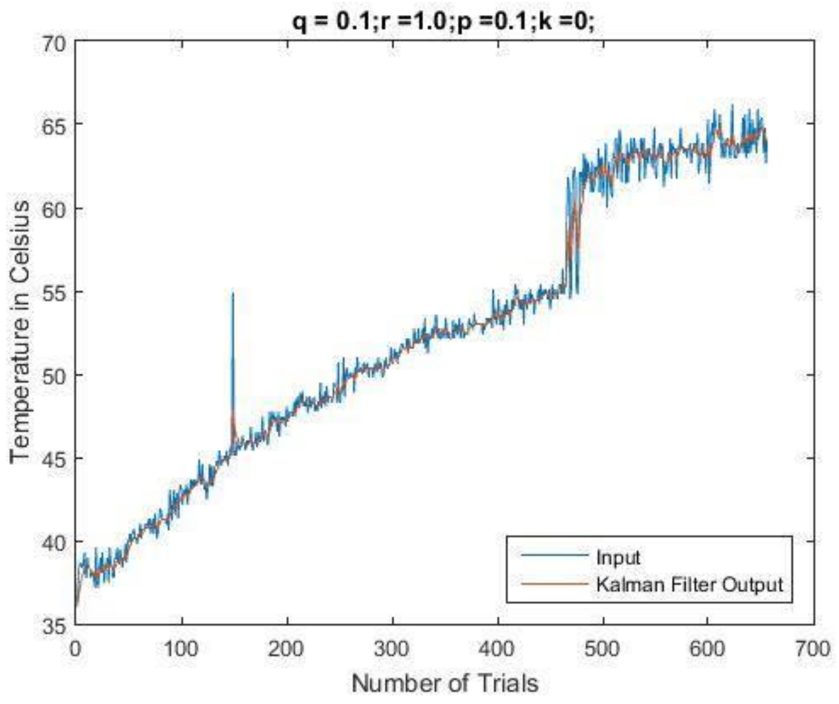

Figure A-6 – Performance of Kalman Filter with measurement noise covariance of 1.0 (r)

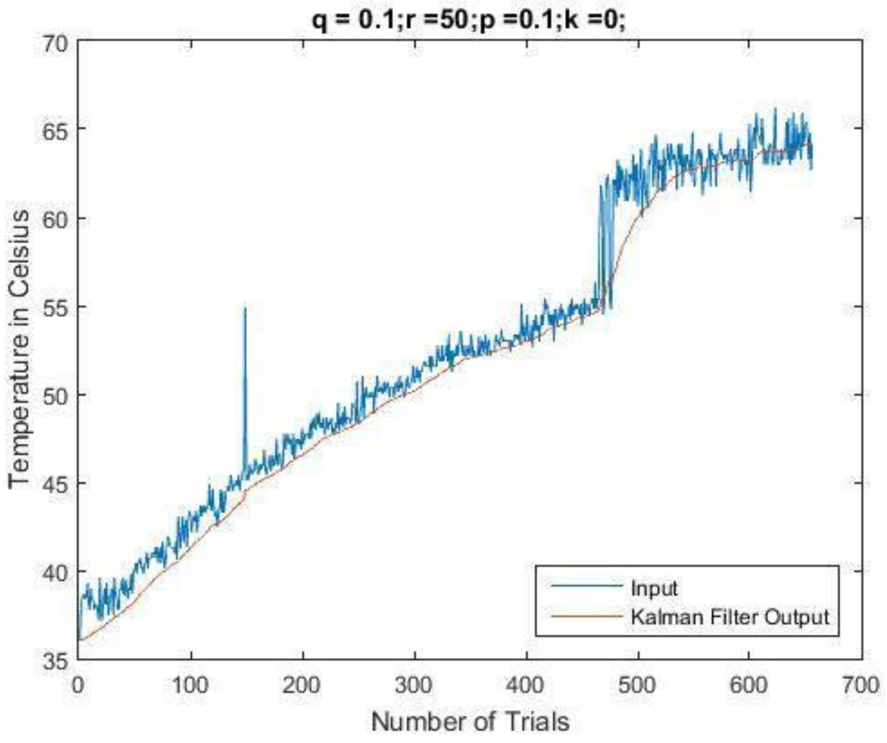

Figure A-7 – Performance of Kalman Filter with large measurement noise covariance (r)

8 References

[1] STMicroelectronics, "Doc_10 – STM32F407xx," datasheet, October 2015.

[2] *Doc_05 – STM32F4xxx Reference Manual*, Revision 1, STMicroelectronics, September 2011.

[3] *Doc_19 – Documentation of STM32F4xx Cube HAL drivers*, Revision 3, STMicroelectronics, September 2015.

[4] A. Suyyagh, Doc_32 – Generic Keypad and Hitachi HD44780 tutorial, 2016.

[5] *Doc_10 – Discovery Kit F4 Rev.C*, Revision 4, STMicroelectronics, January 2014.

[6] R. Bhatt. Serial 4-digit Seven Segment LED Display [Online]. Available: http://www.electronics-lab.com/project/serial-4-digit-seven-segment-led-display/ [Accessed: February 18, 2016]