Section 1: General Lab Info /0.5 points

Microprocessors 2

EECE4800

Lab 3: I2C Interfacing

Instructor: Yan Luo
TA: Ioannis Smanis

Group 10

Kyle Marescalchi

Turned in 11/20/2017

Due 11/20/2017

Section 2: Contributions /1 points

1. Group Member 1 – Kyle Marescalchi (Me) ……….……………………….
My contribution to the lab report focused mainly on the implementation of the I2C temperature sensor and on ensuring that the files would work. We encountered several problems with the program that I worked to resolve, ranging from issues with the temperature sensor reading negative values and otherwise. I also worked to assist in commenting the code and ensuring a proper structure to our program.

2. Group Member 2 –Hans Hoene ……….…………….
Hans created the main code that implemented the I2C temperature sensor's actual function, taking in the values from the temperature sensor and creating a threshold through them. Beyond this, he created the .bat files that allowed for us to easily add or remove files with the Galileo 2. He also implemented the code for the webcam, which was simple but effective.

3. Group Member 3 – Derek Teixera
Derek worked on the hardware implementation and debugging, working primarily on the webcam portion of the lab. He researched the CV functions for the webcam, and developed the process in which it would work and release data so that it would not cause a memory leak. Beyond this, he found and provided the majority of the software specifications that would allow for our code to function properly.

Section 3: Purpose /0.5 points

The purpose of Lab 3 was to learn how to implement USB or I2C into the Galileo 2 board. The primary purpose of the function was to interface both the temperature sensor and the webcam to the board, and implement a code that would take the temperature and record a photo when the temperature reached a certain threshold. The overarching purpose behind this lab is learning how to access preexisting libraries for peripheral devices, then using those for our specific needs and purposes.

Section 4: Introduction /0.5 points

Lab 3 required the implementation of a temperature sensor and a webcam into our Galileo board. This required the use preexisting libraries for the computer vision (CV) functions that were used in our code. The overall code that we used was implemented in C, whereas there was the option to implement the code in C++ for ease of use with the CV tools. This was done as we could resolve the compatibility errors between C and C++, and made the rest of the coding easier for us. The development of the CV functions allowed for the webcam to work, but the I2C temperature sensor required custom addressing to access its information. The overall goal of the lab was to be able to perform the conversion of the I2C device (with its own custom resolution) and to have both the webcam and the temperature sensor interact – the sensor being used to activate the webcam.

Section 5: Materials, Devices and Instruments /0.5 points

- Intel Galileo Gen 2, SN: FZGL50400CW, 5V
- I2C Temperature Sensor
- Personal Computer
- Wires
- USB 2.0 Webcam
- Ethernet Cable
- USB-to-UART Cable

Section 6: Schematics /0.5 points

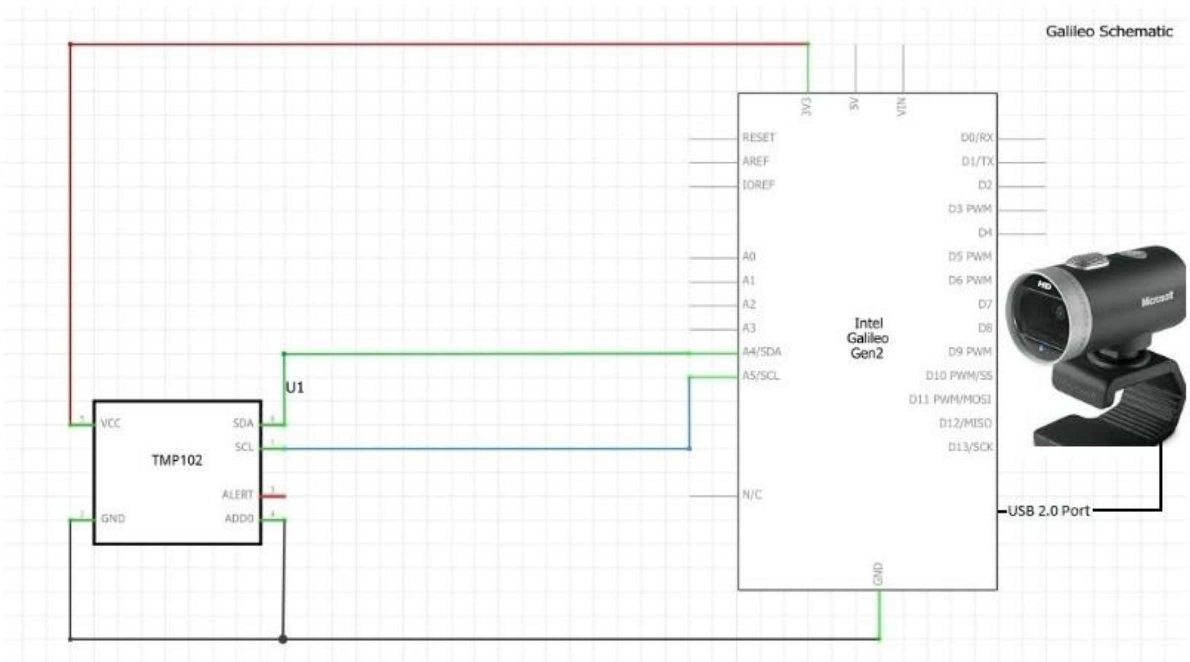


Figure 1: Galileo schematic.

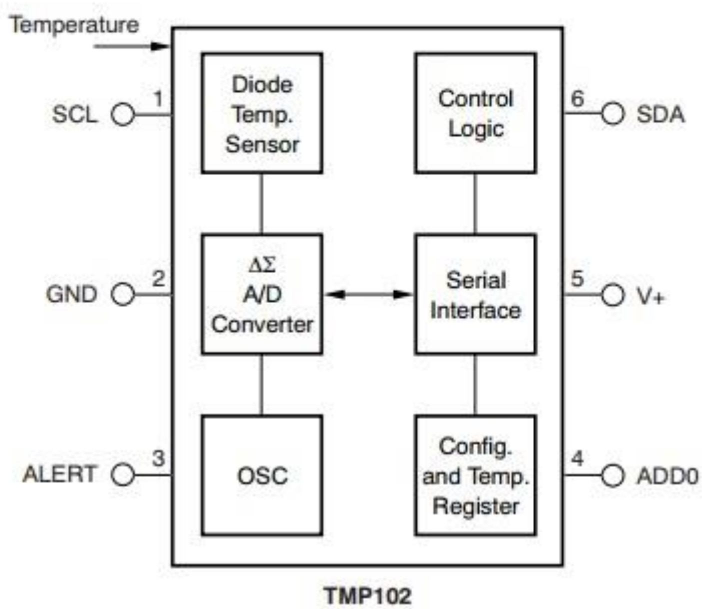


Figure 2: TMP102 schematic.

Section 7: Lab Methods and Procedure /2 points

Hardware design:

The circuit was extremely simple to implement, as there were only two peripheral devices that were both hardwired almost directly to the Intel Galileo. The I2C temperature sensor was connected to the 3.3V output of the Galileo, A0 was connected to ground, and the SDA/SCL pins were connected to their respective pins on the schematic seen above. A0 was connected to ground so that its voltage could be properly referenced to its 0.0625% resolution, which was acquired from the TMP102 datasheet. The I2C configured to address 48, which can be seen below in figure three.

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- UU UU UU -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- UU 48 -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
root@galileo:/sys/class# i2cdetect -l
i2c-0   i2c             intel_qrk_gip_i2c                       I2C adapter
root@galileo:/sys/class# i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- UU UU UU -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- UU 48 -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
root@galileo:/sys/class#
```

Figure 3: I2C Address.

The USB webcam was directly connected to the USB port on the Intel Galileo. The address for the USB was found in port zero, which was later used in our programming. Finally, the connection for the USB webcam was confirmed, and can be seen as such in Figure 4 below.

```
root@galileo:/dev# cd /dev
root@galileo:/dev# ls
autofs           mmcblk0p2           ptyq3      tty12  tty39  tty8     ttyq8
block            mqueue              ptyq4      tty13  tty4   tty9     ttyq9
bus              net                 ptyq5      tty14  tty40  ttyGS0   ttyqa
char             network_latency     ptyq6      tty15  tty41  ttyS0    ttyqb
console          network_throughput  ptyq7      tty16  tty42  ttyS1    ttyqc
core             null                ptyq8      tty17  tty43  ttyp0    ttyqd
cpu              port                ptyq9      tty18  tty44  ttyp1    ttyqe
cpu_dma_latency  ppp                 ptyqa      tty19  tty45  ttyp2    ttyqf
disk             ptmx                ptyqb      tty2   tty46  ttyp3    uio0
esramtest0       pts                 ptyqc      tty20  tty47  ttyp4    uio1
fd               ptyp0               ptyqd      tty21  tty48  ttyp5    urandom
full             ptyp1               ptyqe      tty22  tty49  ttyp6    vcs
fuse             ptyp2               ptyqf      tty23  tty5   ttyp7    vcs1
hpet             ptyp3               ram0       tty24  tty50  ttyp8    vcs2
hugepages        ptyp4               random     tty25  tty51  ttyp9    vcs3
i2c-0            ptyp5               rfkill     tty26  tty52  ttypa    vcs4
iio:device0      ptyp6               rtc        tty27  tty53  ttypb    vcs5
imrtest0         ptyp7               rtc0       tty28  tty54  ttypc    vcs6
initctl          ptyp8               shm        tty29  tty55  ttypd    vcsa
input            ptyp9               snd        tty3   tty56  ttype    vcsa1
kmem             ptypa               spidev1.0  tty30  tty57  ttypf    vcsa2
kmsg             ptypb               stderr     tty31  tty58  ttyq0    vcsa3
log              ptypc               stdin      tty32  tty59  ttyq1    vcsa4
loop-control     ptypd               stdout     tty33  tty6   ttyq2    vcsa5
loop0            ptype               tty        tty34  tty60  ttyq3    vcsa6
loop1            ptypf               tty0       tty35  tty61  ttyq4    zero
mem              ptyq0               tty1       tty36  tty62  ttyq5
mmcblk0          ptyq1               tty10      tty37  tty63  ttyq6
mmcblk0p1        ptyq2               tty11      tty38  tty7   ttyq7
root@galileo:/dev# [  237.240172] usb 1-1: new high-speed USB device number 3 using ehci-pci
[  237.971906] uvcvideo: Found UVC 1.00 device USB2.0 Camera (1e4e:0110)
[  237.988966] input: USB2.0 Camera as /devices/pci0000:00/0000:00:14.3/usb1/1-1/1-1:1.0/inpu
t/input3
```

Figure 4: USB Webcam connection.

Software design:

In designing our software, we focused on the implementation of the I2C temperature sensor and the webcam. Using the address of the I2C sensor, which was shown above in Figure 3, the IO control (IOCTL) function wrote the temperature sensor as a slave device to the Galileo, which was then written as a read-only device. In order to reach the temperature, a buffer stored a total of two bytes from the address of the I2C device – which would actually account for a total of twelve bits. In order to fulfill the proper reading of the data, as seen in Figure 5 below, the buffer's two bytes had to be shifted properly when stored into a temporary variable. Please refer to Appendix 1 for the code mentioned here.

Table 3. Byte 1 of Temperature Register[1]

D7	D6	D5	D4	D3	D2	D1	D0
T11 (T12)	T10 (T11)	T9 (T10)	T8 (T9)	T7 (T8)	T6 (T7)	T5 (T6)	T4 (T5)

(1) Extended mode 13-bit configuration shown in parenthesis.

Table 4. Byte 2 of Temperature Register[1]

D7	D6	D5	D4	D3	D2	D1	D0
T3 (T4)	T2 (T3)	T1 (T2)	T0 (T1)	0 (T0)	0 (0)	0 (0)	0 (1)

(1) Extended mode 13-bit configuration shown in parenthesis.

Figure 5: Temperature reading.

The returned value was multiplied by 0.0625 in accordance with the resolution of the I2C device. When the actual program was ran, we set a threshold for the temperature based off the average readings from five iterations, then would direct the webcam to take a picture following. Refer to Appendix 2 for this code.

For our webcam, our code was relatively simple. It took four sets of functions, seen below, and also referenced in Appendix 3.

```
CvCapture *capture;
IplImage *image;

/*
        1) Establish file name
        2) capture frame
        3) retrieve data from frame
        4) save image to file
        5) release capture
        6) release image
*/

sprintf(filename, "%s/%u.jpg", DEST_FOLDER, id);       // filename is [DEST_FOLDER]/[id].jpg
capture = cvCaptureFromCAM(CV_CAP_ANY);                // capture frame from the camera; stop webcam
                                                       // argument can be zero since there is only one device connected
image = cvQueryFrame(capture);                         // grabs and retrieves data from captured frame
cvSaveImage(filename, image, 0);                       // save image to file as JPG
cvReleaseCapture(&capture);                            // release capture
cvReleaseImage(&image);                                // release image
```

Figure 6: CV code.

This code seen would first write the file where our image will be saved, capture it using the cvCaptureFromCAM function, then use the query function to implement it better. This was then saved, and the capture and image files were then released to avoid a memory leak. Our CV code was written entirely in C, as opposed to the given C++, which required the opencv and opencv2 files, both on the Galileo and in our code. These header files can be seen in Appendix 4, and are referenced in our PIC.C file.

Section 8: Trouble Shooting /1 points

Issue 1: **A primary issue that we had was with our webcam. The webcam would take incorrect photographs or off-color results.**

In order to resolve this issue, we first tested another webcam to see if we received the same problem. Such was not the case, however, for the new webcam retrieved perfect results. We exchanged our faulty webcam for a properly functioning one, and the hardware worked as intended following. Refer to the results section for the related screen captures.

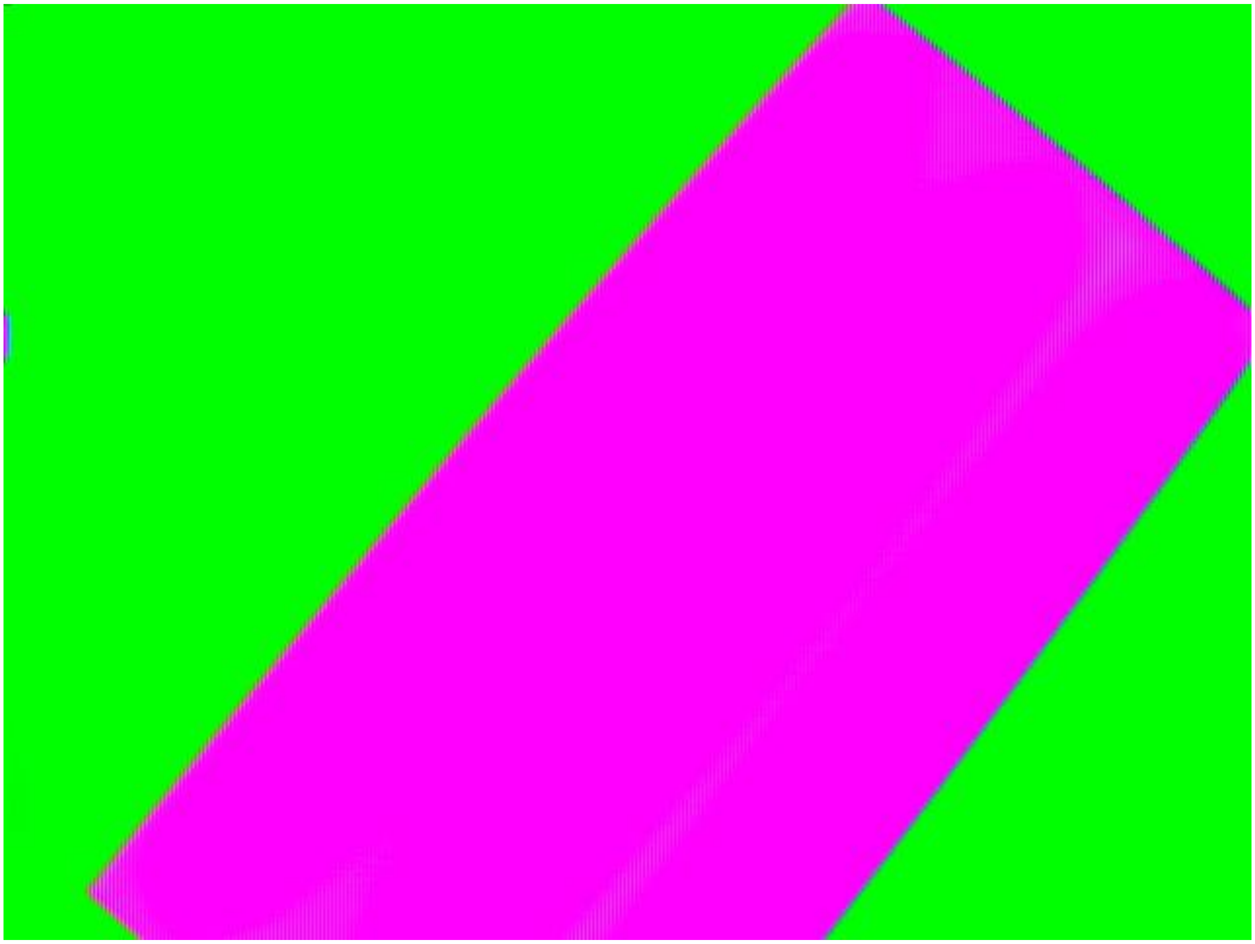
Issue 2: **When recording the temperature values, we would occasionally receive correct results, sometimes our results would fluctuate in value, and even occasionally receive negative results.**

The cause of this issue was found in our programming itself, and not with the hardware. This was by way of two separate issues in our code, referenced in Appendix 1. We had the function `temp = (buffer[0] << 4) + (buffer[1] >> 4);` with an or function instead of the addition. The or function was not yielding consistent results. The addition function resolved the heavy fluctuation, but not the negative. In order to resolve the negative values, we made the buffer and the temperature files unsigned integers. When the top bit was accessed, it was reading it as a negative value instead.

Issue 3: **Not knowing if we our I2C device was connecting.**

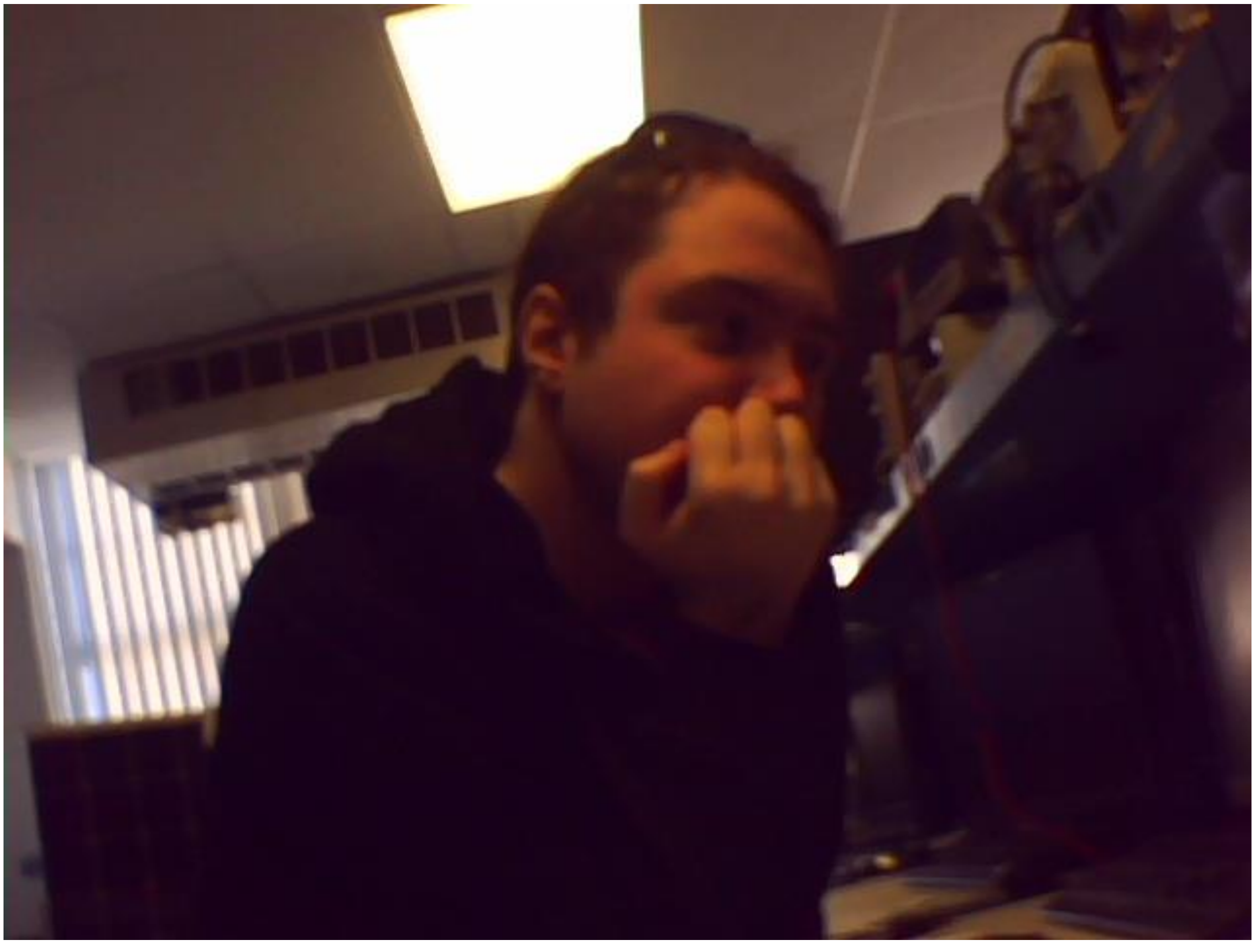
For the initial stages of our project, we struggled with implementing the I2C temperature sensor – if only because we could not properly detect it and address it. We did eventually find out that its address was found in the i2cdetect function for the Galileo, which yielded its address being 48.

Webcam Screenshots:



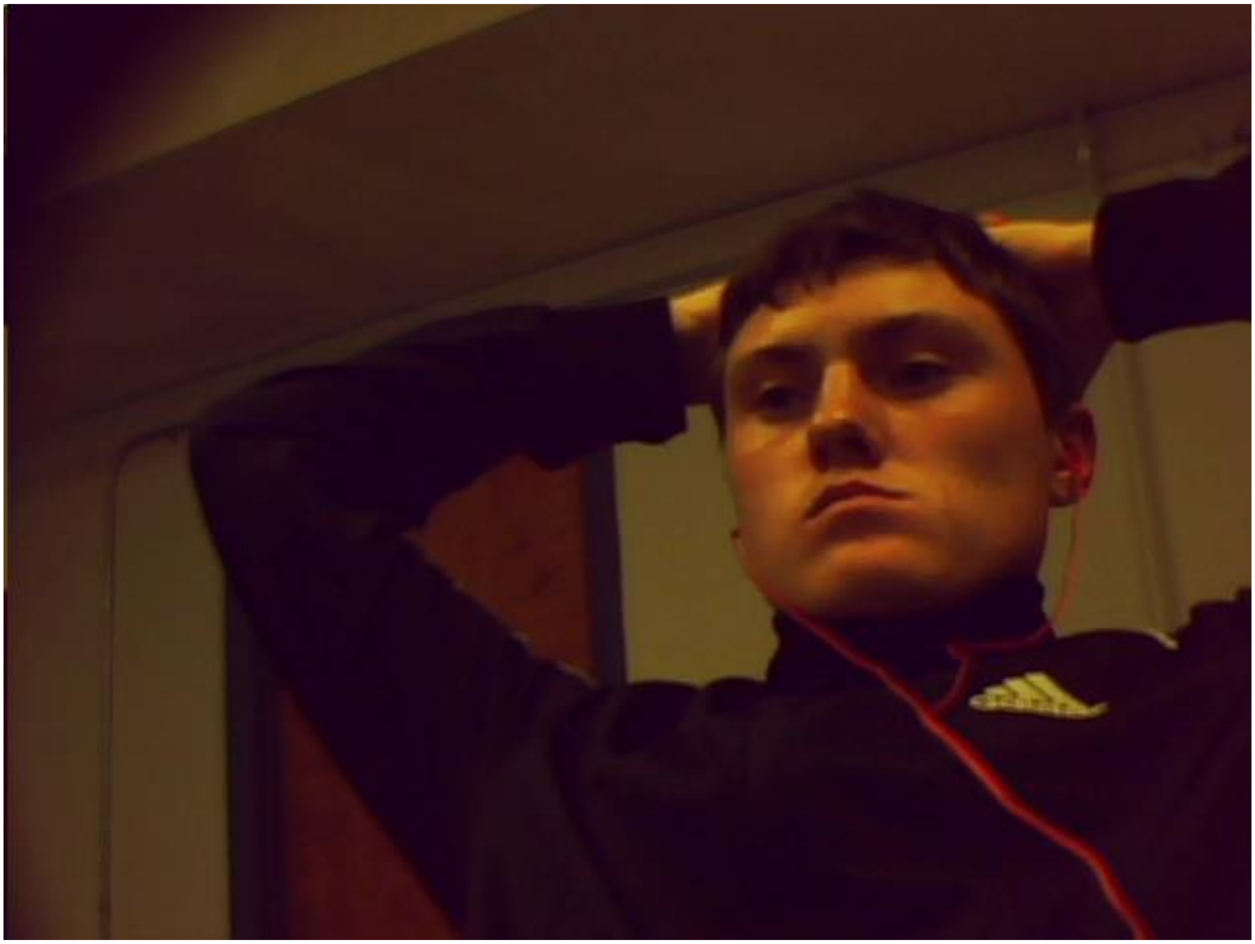
Result 1: Webcam error.

This webcam image was the glitch referenced throughout the lab report, where the webcam would return an entirely incorrect image.



Result 2: Webcam image.

This is one of our proper images from the webcam, where the picture taken was proper and the quality was alright.

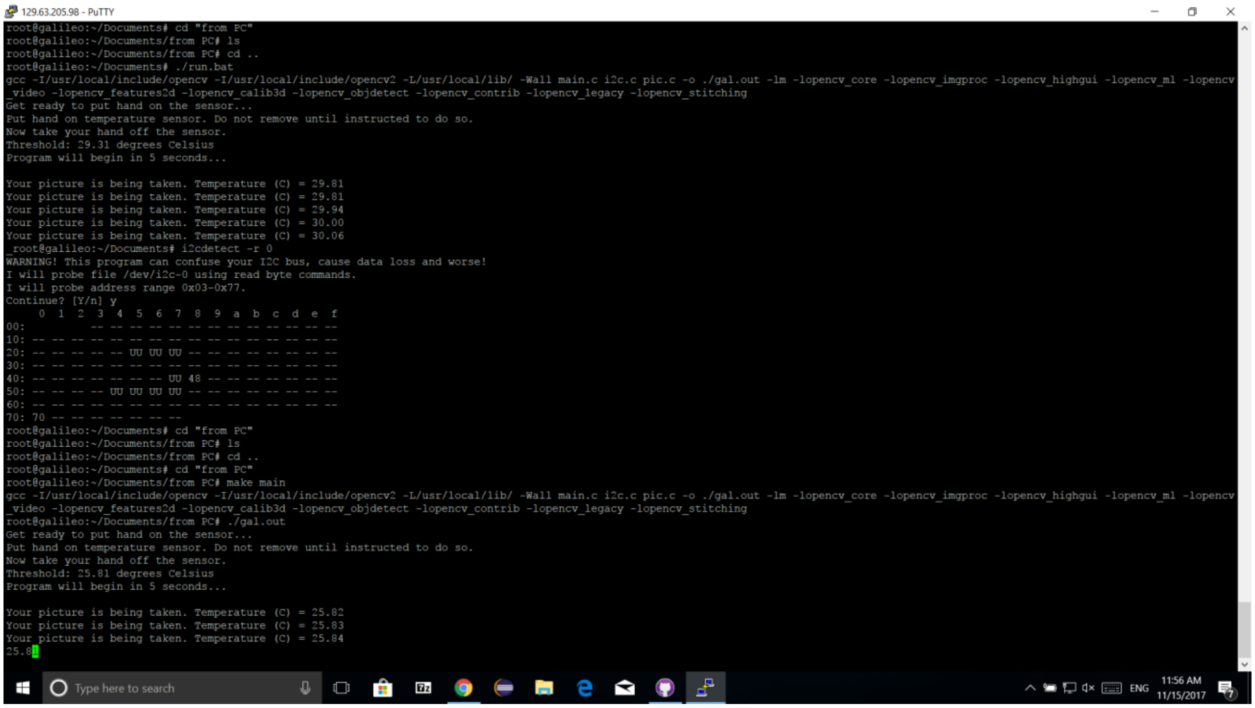


Result 3: Another image.

This is another resulting image from the webcam from our tests.

Result 4: Alternative webcam.

This image is taken from Derek's personal webcam, which was our way of testing for the initial webcam being broken. As can be seen, the picture quality is much higher and clearer than the provided webcam, and this operated under the same code as the broken webcam.




```
root@galileo:~/Documents# cd "from PC"
root@galileo:~/Documents/from PC# ls
root@galileo:~/Documents/from PC# cd ..
root@galileo:~/Documents# ./run.bat
gcc -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/usr/local/lib/ -Wall main.c i2c.c pic.c -o ./gal.out -lm -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv
_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_stitching
Get ready to put hand on the sensor...
Put hand on temperature sensor. Do not remove until instructed to do so.
Now take your hand off the sensor.
Threshold: 29.31 degrees Celsius
Program will begin in 5 seconds...

Your picture is being taken. Temperature (C) = 29.81
Your picture is being taken. Temperature (C) = 29.81
Your picture is being taken. Temperature (C) = 29.94
Your picture is being taken. Temperature (C) = 30.00
Your picture is being taken. Temperature (C) = 30.06
_root@galileo:~/Documents# i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- UU UU UU -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- UU 48 -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
root@galileo:~/Documents# cd "from PC"
root@galileo:~/Documents/from PC# ls
root@galileo:~/Documents/from PC# cd ..
root@galileo:~/Documents# cd "from PC"
root@galileo:~/Documents/from PC# make main
gcc -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/usr/local/lib/ -Wall main.c i2c.c pic.c -o ./gal.out -lm -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv
_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_stitching
root@galileo:~/Documents/from PC# ./gal.out
Get ready to put hand on the sensor...
Put hand on temperature sensor. Do not remove until instructed to do so.
Now take your hand off the sensor.
Threshold: 25.81 degrees Celsius
Program will begin in 5 seconds...

Your picture is being taken. Temperature (C) = 25.82
Your picture is being taken. Temperature (C) = 25.83
Your picture is being taken. Temperature (C) = 25.84
25.81
```

Results 5: Terminal execution and temperature.

This is a recording of our terminal results from our final program. As can be seen, it would tell you to put your hand atop the temperature sensor, wait until the threshold was calculated, then remove your hand from such. Whenever the temperature read above the threshold, it would take a picture up to five results.

A1.

```
double readTemp(int handle) {
	// return temperature in celsius

	unsigned char buffer[2];					// bytes that shall be read will go here
	unsigned int temp;

	read(handle, buffer, 2);					// read 2 bytes from temperature register
	temp = (buffer[0] << 4) + (buffer[1] >> 4);			// shift bytes by appropriate amounts to get 12-bit value
	return (double)temp * 0.0625;					// multiply by 0.0625 (2^-4) [the resolution] to get temperature in celsius
}
```

A2.

```
double sampleTemp(int handle) {

	unsigned int i;
	double sum;

	sum = 0;
	for (i = 0; i < NUM_SAMPLES; i++) {
		sum += readTemp(handle);
	}

	return sum / NUM_SAMPLES;
}
```

A3.

```
CvCapture *capture;
IplImage *image;

/*
	1) Establish file name
	2) capture frame
	3) retrieve data from frame
	4) save image to file
	5) release capture
	6) release image
*/

sprintf(filename, "%s/%u.jpg", DEST_FOLDER, id);	// filename is [DEST_FOLDER]/[id].jpg
capture = cvCaptureFromCAM(CV_CAP_ANY);			// capture frame from the camera; stop webcam
							// argument can be zero since there is only one device connected

image = cvQueryFrame(capture);				// grabs and retrieves data from captured frame
cvSaveImage(filename, image, 0);			// save image to file as JPG
cvReleaseCapture(&capture);				// release capture
cvReleaseImage(&image);					// release image
```

A4.

```
// Open CV Header Files
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
```