Section 1: General Lab Info /0.5 points

Course name: Microprocessor Systems II and Embedded Systems

Course Number: EECE.5520

Lab title: Controlling an I2C Device

Instructors names: Yan Luo
Ioannis Smanis

Group number: 7

Student name: Poornima Manjunath

Hand in Date: 10/21/2017

Lab Due Date: 11/20/2017

Section 2: Contributions /1 points

1. Group Member 1 – Poornima Manjunath

   I have taken part in connecting wifi to the Galileo board. Our wifi network was always getting connected to the UMass Lowell Open network. Then I read through the document given in the GitHub and used the connmanctl>agent on and connmanctl> connect commands and made the wifi network to get connected to eduroam.

   I also helped the project by finding how to transfer pictures from the SD card to the Windows machine using WinSCP.

   I have handled the coding part of the project. The compilation step of the OpenCv program was tricky. We tried many ways to compile it. But we struggled a lot. I had to do a lot of research for compilation method.

   I also read the document https://www.kernel.org/doc/Documentation/i2c/dev-interface provided in the GitHub and followed the methods accordingly to program the Gesture sensor APDS9960.I also read the datasheet of the sensor and found the required registers needed for the sensor configuration.

2. Group Member 2 – Rinkal Shah
   She handled the hardware design part of the project. She also read through the datasheet of the Gesture sensor and found out the Pin configuration. She also helped for the OpenCv code compilation. She also helped in coding the sensor program by telling which registers have to be configured in order to get precise results. She also helped by determining a suitable threshold value for the gesture sensor value to trigger the web cam.

3. Group Member 3 – Akhila Nair
   She helped in wifi connection to the Galileo board. She also helped in interfacing the Gesture sensor. She also helped in debugging the program.

Section 3: Purpose /0.5 points

This lab helps us to gain knowledge about interfacing new devices to the system so that our embedded system has richer functions. Through this lab, we will be able to add more functionality to our Galileo board. This lab helps us to understand the usage of OpenCV library functions used for capturing and processing the images. Also, introduces us to the Gesture sensor which has multiple capabilities like ambient light and color (as clear, red, green, and blue) measuring, proximity detection, and gesture sensing especially when touchless gestures are the new frontier in the world of human-machine interfaces. The lab also us helps to

• Understand I2C bus protocol.

- ➢ We will get to know how to use Linux I2C libraries and APIs.

• Be able to control an I2C device using Linux on a Galileo board

- ➢ We have to use gesture sensor to trigger the capture of images from webcam. We need to define a threshold and check if the sensor data exceed the threshold. If so, capture images and save them to the file system.

• Be able to capture, store and process camera images on Linux

- ➢ programming on Linux to access and handle the provided webcam and capture images. Store the images on the SD card.

Section 4: Introduction /0.5 points

In this lab, we are using the gesture sensor IC SparkFun RGB and Gesture Sensor - APDS-9960 to trigger the USB web cam to take the pictures and save them on the SD card.

This is the SparkFun RGB and Gesture Sensor, a small breakout board with a built in APDS-9960 sensor that offers ambient light and color measuring, proximity detection, and touchless gesture sensing. With this RGB and Gesture Sensor you will be able to control a computer, microcontroller, robot, and more with a simple swipe of your hand! This is, in fact, the same sensor that the Samsung Galaxy S5 uses and is probably one of the best gesture sensors on the market for the price.

The APDS-9960 is a serious little piece of hardware with built in UV and IR blocking filters, four separate diodes sensitive to different directions, and an I^2C compatible interface.

We are then triggering the USB 2.0 webcam to take the picture with the camera using the OpenCv library.

This can be put to use in applications like

- Motion-sensing security camera
- Intelligent and sensitive selfi apps in mobiles.
- The gesture sensor triggered web cam can be put to use in automotives for blind-spot recognition, and parking assist.
- Touchless user interface is an emerging type of technology in relation to gesture control. It can be combined with the image processing technology for an effective use.

Section 5: Materials, Devices and Instruments /0.5 points

Materials:

1. Breadboard

2. Jumper cables

Devices:

➢ Intel Galileo Gen2 board

- Galileo is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to 5V at the I/O pins. This provides support for 5V Uno shields and is the default behavior. By switching the jumper position, the voltage translation can be disabled to provide 3.3V operation at the I/O pins. Digital pins 0 to 13 (and the adjacent

AREF and GND pins), Analog inputs 0 to 5, the power header, ICSP header, and the UART port pins (0 and 1)

- Micro-SD Card

This 8GB card contains the tailored Yocto Linux* image with many libraries and tools to integrate with the IoTDK.

- USB to 6-pin FTDI serial cable

The FTDI cable is a USB to Serial (TTL level) converter which allows for a simple way to connect TTL interface devices to USB. The I/O pins of this FTDI cable are configured to operate at 5V.

- Integrated Wi-Fi* plus Bluetooth* card, including a half to full height PCIe Extender

- Wi-Fi Antennas
- Gesture Sensor: SparkFun RGB and Gesture Sensor APDS-9960

Power Supply: 3.3V MAX (Please do not exceed this limit) Interface: I2C (3.3V tolerance) Device Address: 0x39

PIN OUT: SDA: I2C Data Line (3.3V tolerance) SCL: I2C Clock Line (3.3V tolerance) VCC: 3.3V GND: Ground INT: Interrupt VL : Optional Power to IR LED (3.3V tolerance)

Hooking up Gesture Sensor on Galileo Board: there is no need for pull-up resistors on the I2C lines.Gesture detection range of 4 to 8 inches (10 to 20 cm).

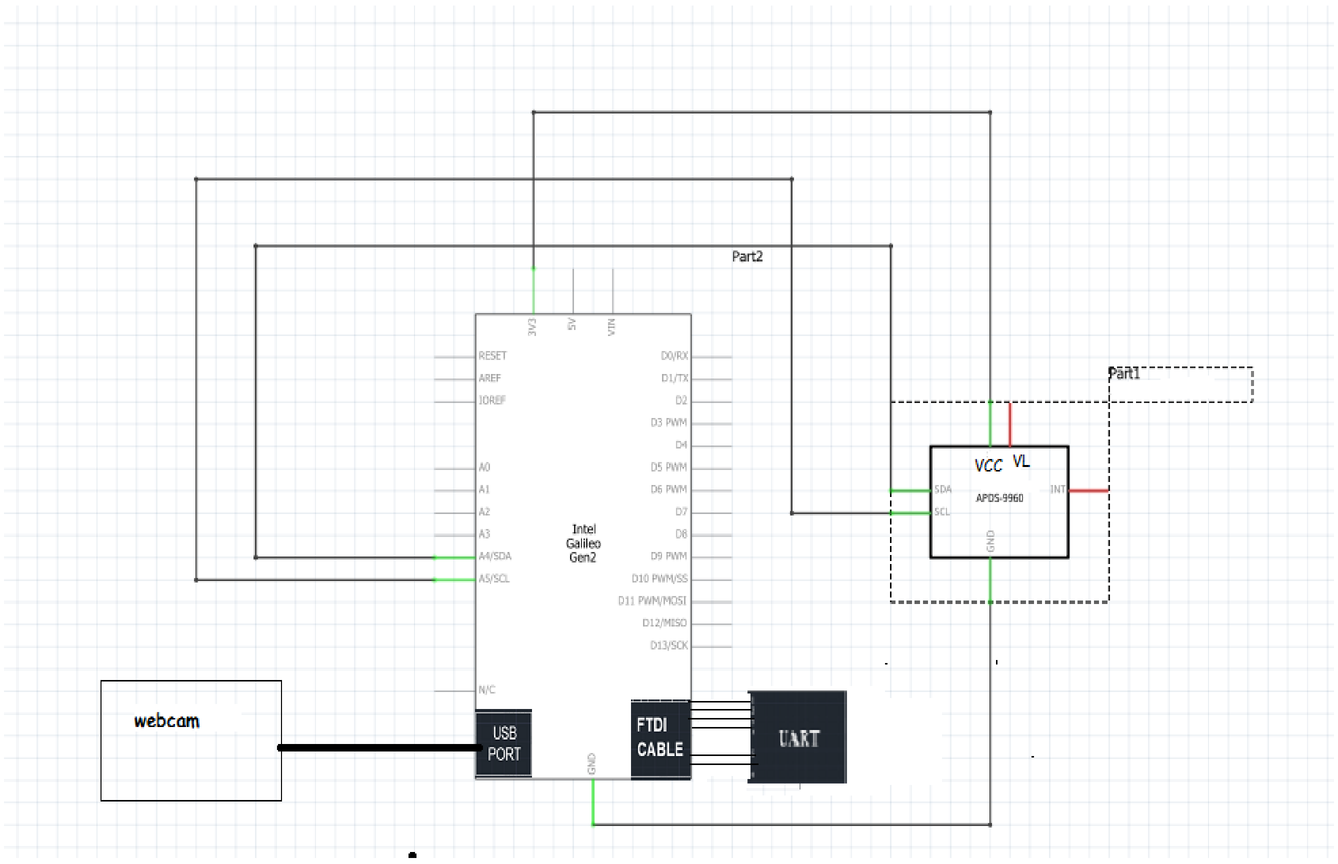
- USB 2.0 Web-Camera - 5.0 MegaPixel

Instruments and Software:

1. 12V regulated DC security power adapter

2. Laptop

Section 6: Schematics /0.5 points



Section 7: Lab Methods and Procedure /2 points

Hardware design:

The basic principle of this project is to interface gesture sensor to our embedded system i.e Intel Galileo using I2C bus protocol. Also, we are interfacing a USB webcam to the Intel Galileo and hence adding more functionalities to our embedded system.

We use the Gesture sensor to detect the proximity and use it to trigger the webcam. We define a threshold and check if the sensor data exceed the threshold. If so, capture images and save them to the file system.

Connecting Gesture sensor to the Intel Galileo:

Gesture sensor APDS9960 has the Pin configuration as below:

SDA: I2C Data Line (3.3V tolerance)

SCL: I2C Clock Line (3.3V tolerance)

VCC: 3.3V

GND: Ground

INT: Interrupt

VL : Optional Power to IR LED (3.3V tolerance)

VCC pin is connected to 3.3 v pin of Intel Galileo board and GND is connected to the Ground pin of the Intel Galileo.

SDA and SCL pins of the sensor should be connected to A4 (SDA) and A5 (SCL) of Galileo’s expansion I/O ports. There is no need to wire the pull-up resistors or enable pull-up resistors on Galileo Board for the I2C bus since the sensor breakout boards already have them.

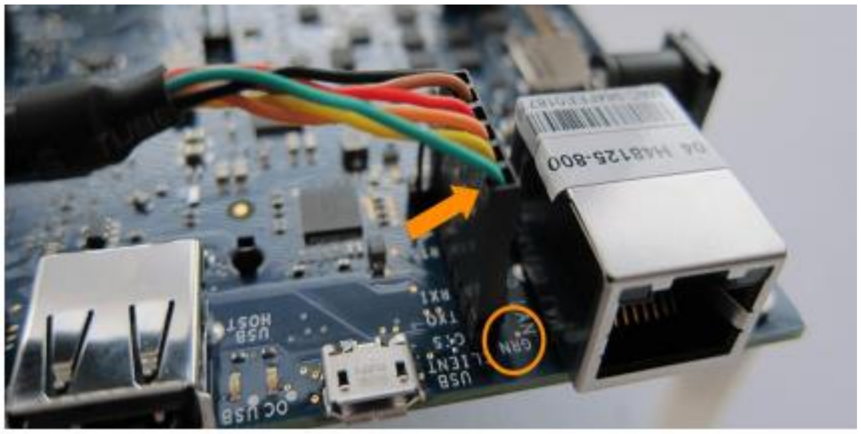
Connecting webcam to the Intel Galileo:

Connect the web cam to the USB port of the Intel Galileo board.

Connecting the FTDI cable to the Intel Galileo:

FTDI cable has the below pinout:

Green Wire: RTS (3.3V tolerance)
Yellow Wire: RX Data Line (3.3V tolerance)
Orange Wire: TX Data Line (3.3V tolerance)
Red Wire: 5V
Brown Wire: CTS (3.3V tolerance)
Black Wire: Ground



Connecting wifi to the Intel Galileo:

Step 1. Install the wi-fi hardware on the PCI-express socket on the back of the Intel Galileo Gen2 board (Very gently connect the white and the black antenna-wires on the WiFi card)

Step2: Power the wifi using the command “connmanctl enable wifi”

Step3: To connect to a network, first scan the available networks using the command “connmanctl scan wifi”.

Step4: List the available networks using the command “connmanctl services”.

Step5: To connect to the “eduroam” network edit the edit “eduroam.config” file applying your credentials (school email and school password) and save it , then store this file in “/var/lib/connman/“ Galileo’s directory

Step6: Save the fille and restart connman to connect to the new network.

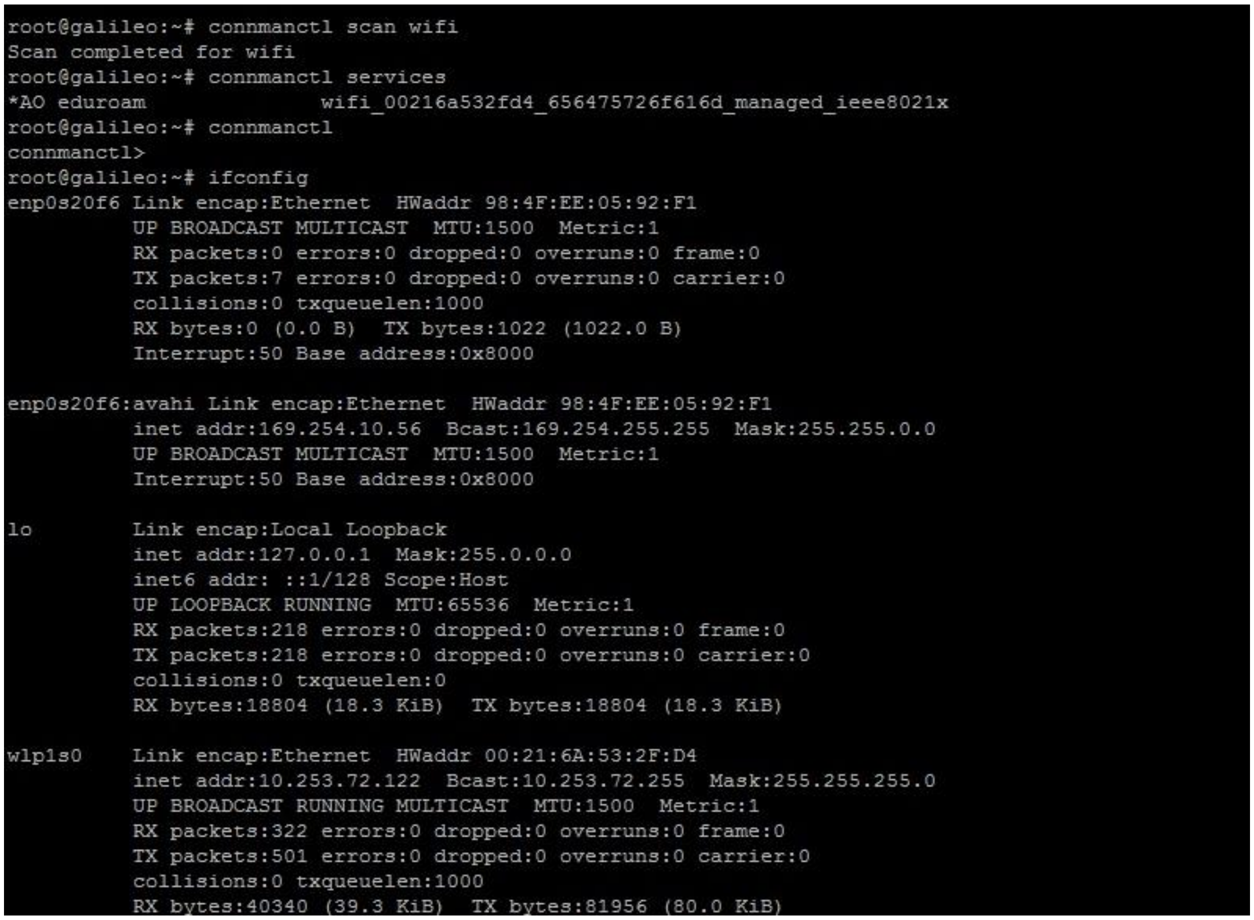
```
:~# systemctl restart connman
```

Step7: Galileo will connect to the eduroam network. In our case, sometimes Galileo used to get connected to the open network UMass Lowell. In that case, we ran connmanctl in interactive mode.To start interactive mode simply type: $ connmanctl

Step8: Register the agent to handle user requests. The command is: connmanctl> agent on.

Step9: We can now connect to the eduroam network using the command

connmanctl> connect wifi_XXXX(eduroam network)



```
root@galileo:~# connmanctl scan wifi
Scan completed for wifi
root@galileo:~# connmanctl services
*AO eduroam              wifi_00216a532fd4_656475726f616d_managed_ieee8021x
root@galileo:~# connmanctl
connmanctl>
root@galileo:~# ifconfig
enp0s20f6 Link encap:Ethernet  HWaddr 98:4F:EE:05:92:F1
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:1022 (1022.0 B)
          Interrupt:50 Base address:0x8000

enp0s20f6:avahi Link encap:Ethernet  HWaddr 98:4F:EE:05:92:F1
          inet addr:169.254.10.56  Bcast:169.254.255.255  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Interrupt:50 Base address:0x8000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:218 errors:0 dropped:0 overruns:0 frame:0
          TX packets:218 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18804 (18.3 KiB)  TX bytes:18804 (18.3 KiB)

wlp1s0    Link encap:Ethernet  HWaddr 00:21:6A:53:2F:D4
          inet addr:10.253.72.122  Bcast:10.253.72.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:322 errors:0 dropped:0 overruns:0 frame:0
          TX packets:501 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:40340 (39.3 KiB)  TX bytes:81956 (80.0 KiB)
```

Software design:

Configuration of Gesture sensor APDS-9960:

We are using the Gesture sensor to detect the proximity. The Proximity detection feature provides distance measurement by photodiode detection of reflected IR energy sourced by the integrated LED.

We need to set up the Enable register with the address 0x80.We have to set the ENABLE<PON> (0x80<0>) and ENABLE<PEN> (0x80<2>) bits. We then have to read the proximity data from the PDATA register with the address 0x9C. It will have a higher value if an object is near to the sensor. The value decreases as the object is taken far from the sensor.

In our case, we have set 30 as threshold for proximity data. If it is more than 30, web cam will be triggered to take a picture.

<u>Programming I2C Devices from Linux:</u>

Gesture sensor APDS9960 is an i2c device.

- Usually, i2c devices are controlled by a kernel driver. But it is also possible to access all devices on an adapter from user space, through the /dev interface. We need to load module i2c-dev for this. Each registered i2c adapter gets a number, counting from 0. We can examine /sys/class/i2c-dev/ to see what number corresponds to which adapter.We found that our adapter has the number 0.
  If we want to access an i2c adapter from the program we have to include "#include <linux/i2c-dev.h>".

- Open the device file, as follows

```
int file;
int adapter_nr = 0; /* probably dynamically determined */
char filename[20];

snprintf(filename, 19, "/dev/i2c-%d", adapter_nr);
file = open(filename, O_RDWR);
if (file < 0) {
  /* ERROR HANDLING; you can check errno to see what went wrong */
  exit(1);
}
```

- When we have opened the device, we must specify with what device
address we want to communicate:

```
int addr = 0x39; /* The I2C address */

if (ioctl(file, I2C_SLAVE, addr) < 0) {
  /* ERROR HANDLING; you can check errno to see what went wrong */
  exit(1);
}
```

- We have to write 0x05 to the Enable register with the address 0x80 to make the Power ON and Proximity Enable ON. We are using the below piece of code:

```
sprintf(buffer,"i2cset -y -ff 0 0x39 0x80 0x05");
system(buffer);
```

- We have to read the proximity data from the PDATA(Proximity Data)register 0x9C. So, we need to write the Device register to be accessed to the i2c bus.

```
     Char buf[10];
     buf[0] = 0x9C;

 if (write(file, buf, 1) != 1) {
  /* ERROR HANDLING: i2c transaction failed */
 }
```

- We can read the PDATA register content using the command

```
/* Using I2C Read, equivalent of i2c_smbus_read_byte(file) */
  if (read(file, buf, 1) != 1) {
    /* ERROR HANDLING: i2c transaction failed */
  } else {
    /* buf[0] contains the read byte */
  }
```

<u>Programming the web cam using OpenCV library functions:</u>

- Check if the camera is connected to your Galileo board: browse to the "/dev" directory and type "ls" to see the device list of your board. You should see a device called "video0"

- We need to Create pointer and initialize detected camera "0".

  ```
  CvCapture *capture = cvCaptureFromCAM(CV_CAP_ANY);
  ```

  CV_CAP_ANY is the constant 0 in the enumerator list defined in OpenCV library file.

- We create pointer to IplImage structure for storing image data in c style interface.
  IplImage *iplImg;

- Capture single frame and store in iplImg using camera address.
  iplImg = cvQueryFrame(capture);

- Save image data (iplImg) to buffer location on SD drive.

cvSaveImage(buffer, iplImg);

- De-initialize camera before exiting.
  cvReleaseCapture(&capture);

Algorithm:

1. Open the i2c device file.
2. Specify with what device address we want to communicate.
3. Write 0x05 to the Enable register with the address 0x80 to make the Power ON and Proximity Enable ON.
4. Write the Device register PDATA address(0x9c) to be accessed to the i2c bus.
5. Read the PDATA register content.
6. If it is more than threshold (in our case 30) then take the picture using OpenCV built-in APIs.
7. Else print "Nobody is here".
8. Keep doing this inside a while(1) loop. Also increment the index number used to store the image.

Section 8: Trouble Shooting /1 points

ISSUE 1:

At first, we were unable to connect to the wifi , because we had made a slight mistake in editing the eduroam.config file. We had used lower case letter 't' for the feature Type. And also we were not using the right credentials for the password. When we corrected this, we were able to connect to wifi.

ISSUE 2:

We faced problem in compiling the code for web cam that uses OpenCV library. Later we got to know one solution after doing a lot of research that we have to run the compilation commands using the online OpenCV library enabling our wifi connection.

This is how we finally compiled our code for webcam.

ISSUE 3:

After we ran our code for web cam it had executed properly. But the next time we tried running it it started giving error saying “undefined reference to main” “ld returned exit status 1”

Then we removed our .viminfo file from the directory and the code compiled and executed fine.

ISSUE 4:

While configuring the Gesture sensor, we were not setting the Enable register (0x80)in a right way. We were writing wrong values to the 0x80. We were writing 0x25 instead of 0x05. So were not able to get proper results for proximity detection. i.e the PDATA(0x9C) value was always constant. When we put the right value to Enable(0x80) we were able to get proper results.

ISSUE 5:

On the day of demo, our Gesture sensor had got damaged. Hence, we were getting the runtime error

Error: Write failed.

When we replaced the Gesture sensor, the code worked fine.

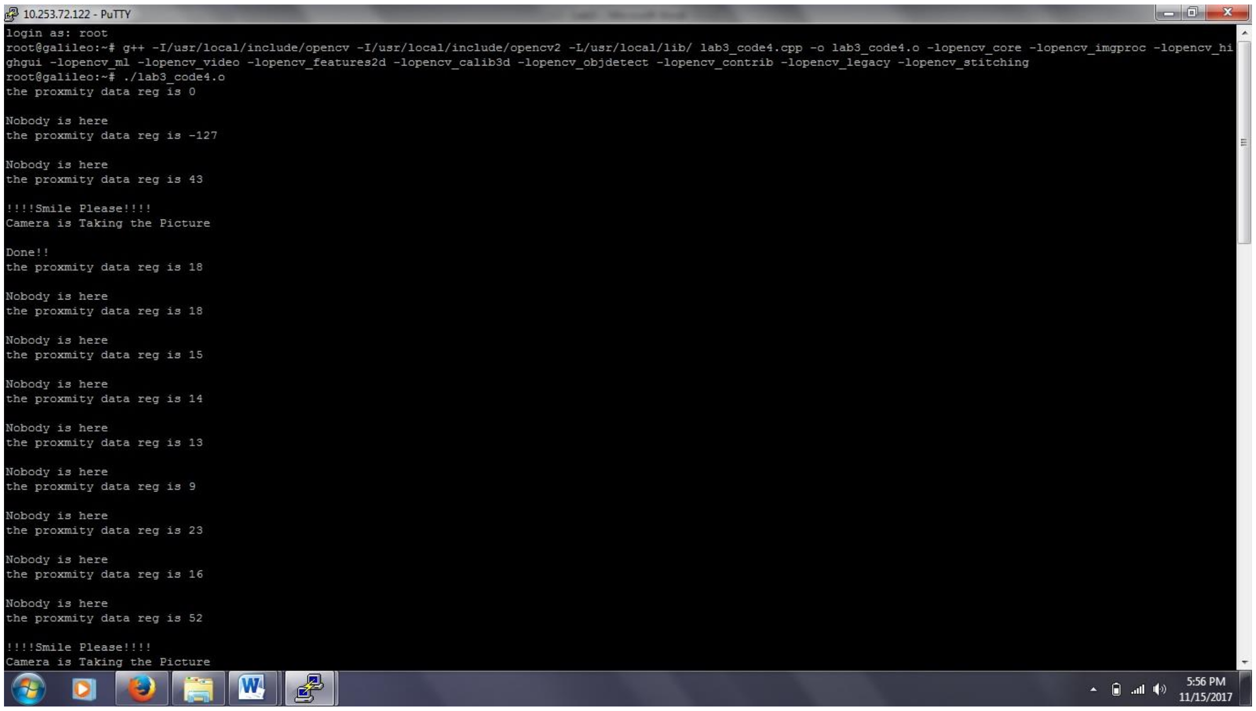
Section 9: Results /0.5 points

Result1:

Compiling and executing the program.

The proximity data getting varied depending on the nearness of the object and the picture being taken when the proximity data is above threshold i.e we have set the threshold to 30.

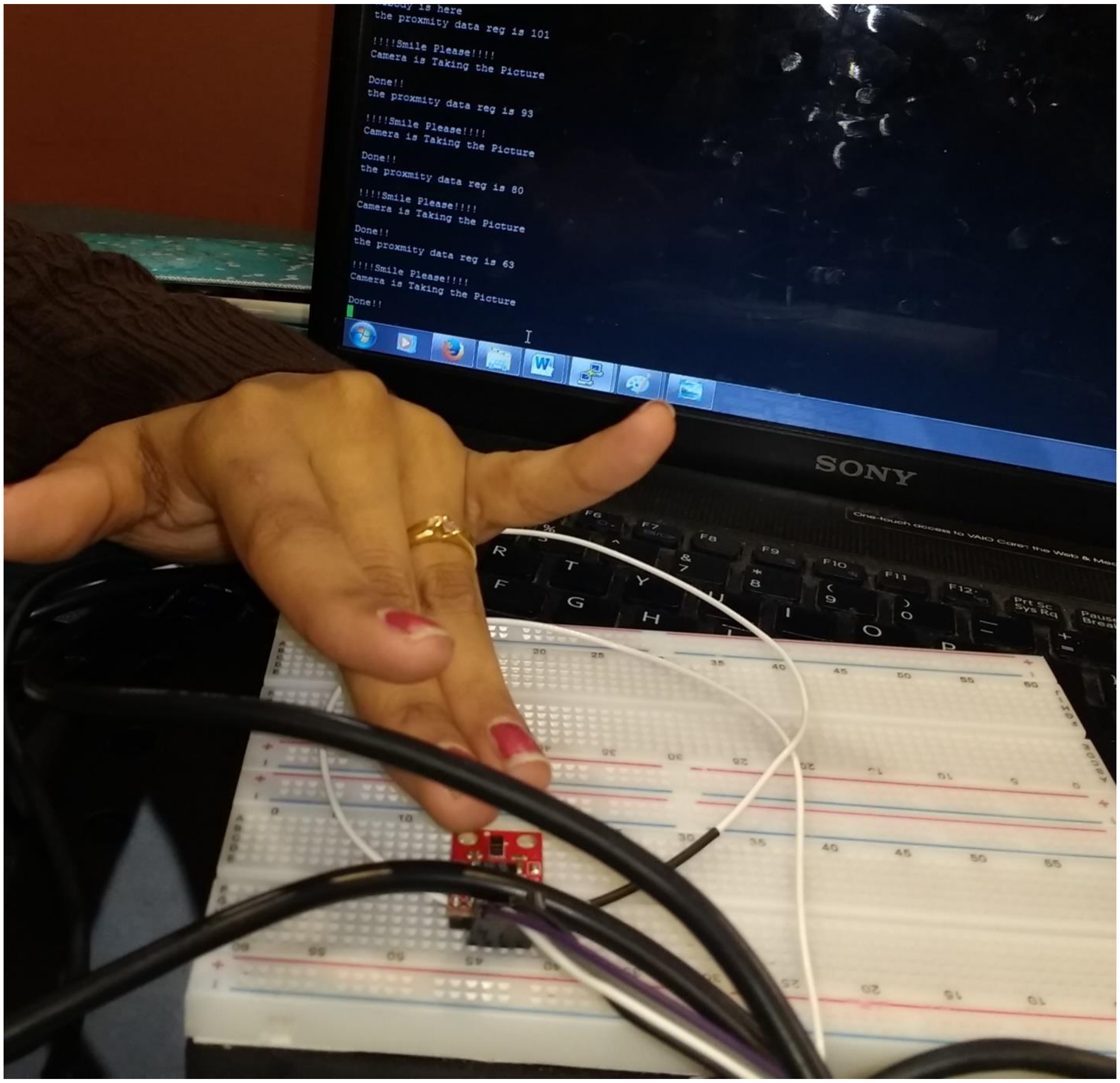
The program will make the webcam to take a picture when the proximity data is above threshold. Else, the program will print “Nobody is here”.



```
login as: root
root@galileo:~# g++ -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/usr/local/lib/ lab3_code4.cpp -o lab3_code4.o -lopencv_core -lopencv_imgproc -lopencv_hi
ghgui -lopencv_ml -lopencv_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_stitching
root@galileo:~# ./lab3_code4.o
the proxmity data reg is 0

Nobody is here
the proxmity data reg is -127

Nobody is here
the proxmity data reg is 43

!!!!Smile Please!!!!
Camera is Taking the Picture

Done!!
the proxmity data reg is 18

Nobody is here
the proxmity data reg is 18

Nobody is here
the proxmity data reg is 15

Nobody is here
the proxmity data reg is 14

Nobody is here
the proxmity data reg is 13

Nobody is here
the proxmity data reg is 9

Nobody is here
the proxmity data reg is 23

Nobody is here
the proxmity data reg is 16

Nobody is here
the proxmity data reg is 52

!!!!Smile Please!!!!
Camera is Taking the Picture
```

Result 2:

Gesture sensor detecting the proximity of the human hand.



```
is here
the proxmity data reg is 101

!!!!Smile Please!!!!
Camera is Taking the Picture

Done!!
the proxmity data reg is 93

!!!!Smile Please!!!!
Camera is Taking the Picture

Done!!
the proxmity data reg is 80

!!!!Smile Please!!!!
Camera is Taking the Picture

Done!!
the proxmity data reg is 63

!!!!Smile Please!!!!
Camera is Taking the Picture

Done!!
```

Result3:

Photo captured by the webcam.

Section 10: Appendix

A1.Code for Webcam using OpenCV library:

```
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
//*** Opencv libraries ***
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
//*** Normal c/c++ code libraries ***
#include <fcntl.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <math.h>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <unistd.h>

int main()//main function
{
```

```
char buffer[100];//Buffer used for image file name and location of SD drive

sprintf(buffer,"phot1.jpg");//filling buffer

//Create pointer and initialize detected camera "0"

CvCapture *capture = cvCaptureFromCAM(CV_CAP_ANY);

printf("\n!!!!Smile please!!!!\n");//Tell everyone to Smile

if (capture == NULL)//Error to detect if unable to open camera

{

printf("fail\n");

}

else{//Indicate if camera was initialized

printf("Camera is taking the picture\n");

}

//Create pointer to IplImage structure for storing image data in c style interface

IplImage *iplImg;

iplImg = cvQueryFrame( capture );//Capture single frame and store in iplImg using camera
address

sleep(.5);

cvSaveImage(buffer, iplImg);//Save image data (iplImg) to buffer location on SD drive

cvReleaseCapture(&capture);//De-initialize camera before exiting

}
```

A2. Code for Proximity detection using Gesture sensor.

```
#include<linux/i2c-dev.h>
#include<sys/ioctl.h>
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>
#include<string.h>
#include<sys/types.h>
int main()
{
int file;
int adapter_nr = 0;
char filename[20];
char buffer[200];
 int res;
snprintf(filename,19, "/dev/i2c-%d",adapter_nr);
file = open(filename,O_RDWR);
if(file<0)
        {
                        printf("cannot open file for sensor\n");
                        return 1;
            }
int addr = 0x39;
if(ioctl(file,I2C_SLAVE,addr)<0)
            {
                        printf("error occurred while writing sensor address");
                        return 1;
            }

char buf[10];
printf("error happened before proximity enable register\n");
sprintf(buffer,"i2cset -y -ff 0 0x39 0x80 0x05");
system(buffer);
buf[0] = 0x9c;
if(write(file,buf,1)!=1)
       {
                   printf("write failed\n");
            }
 if(read(file,buf,1)!= 1)
            {
                printf("error in reading\n");
            }
       else
            {
                        res = buf[0];
            }
printf("the proxmity reg is %d\n",res);
return 0;
 }
```

A3. Code for triggering webcam using Gesture sensor

```
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
//*** Opencv libraries ***
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>
//*** Normal c/c++ code libraries ***
#include <fcntl.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <math.h>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <unistd.h>
int main()//main function
{

            int file;
            int adapter_nr = 0;
            char filename[20];
            char buffer[200];
            int res;
            int count =0;
            while(1)
            {
            snprintf(filename,19, "/dev/i2c-%d",adapter_nr);
            file = open(filename,O_RDWR);

            if(file<0)
             {
                        printf("cannot open file\n");
```

```
                return 1;
        }
        int addr = 0x39;
        if(ioctl(file,I2C_SLAVE,addr)<0)
        {
                printf("error");
                return 1;
        }
       char buf[10];
       sprintf(buffer,"i2cset -y -ff 0 0x39 0x80 0x05");
        system(buffer);

         buf[0] = 0x9c;
       if(write(file,buf,1)!=1)
      {
              printf("write failed\n");
        }
     if(read(file,buf,1)!= 1)
        {
                printf("error in reading\n");
        }
   else
        {
                res = buf[0];
        }

     printf("the proxmity data reg is %d\n",res);
         if(res > 30)
         {
     sprintf(buffer,"photo%d.jpg",count);//filling buffer
    //Create pointer and initialize detected camera "0"
     CvCapture *capture = cvCaptureFromCAM(CV_CAP_ANY);
     printf("\n!!!!Smile Please!!!!\n");//Tell everyone to Smile
     if (capture == NULL)//Error to detect if unable to open camera
     {
         printf("fail\n");
     }
     else{//Indicate if camera was initialized
     printf("Camera is Taking the Picture\n");
     }
    //Create pointer to IplImage structure for storing image data in c style interface
      IplImage *iplImg;
      iplImg = cvQueryFrame( capture );//Capture single frame and store in iplImg using camera
address
      sleep(.5);
      cvSaveImage(buffer, iplImg);//Save image data (iplImg) to buffer location on SD drive
```

```
          cvReleaseCapture(&capture);//De-initialize camera before exiting
              printf("\nDone!!\n");
              count++;
         }
            else
         {
                        printf("\nNobody is here\n");
              }

            sleep(5);

      }

}
```