Lab 4: Multithreaded Programming and Image Processing

Group 9 – Deep Blue

Hema Nalini Routhu
Paridhi Saxena
Kunwar Taha
Hand in Date 12/12/2016
Due Data 12/12/2016

2. Contributions:

Hema – I worked on coding of threads using POSIX threads. I used the example program given by the TA, modified it such that each thread has a definite function. I also had to use mutex, synchronization of threads to make sure that thread 2 waits for thread 1 condition to be satisfied before sending the file to web server. I worked with Paridhi and Taha as I had to integrate their code as well as lab 2 code in different threads and test it for proper functionality. I worked with Paridhi on soldering the temperature and gesture sensor to header pins.

Paridhi – She worked on running the webserver using the given Python script. She modified the test program to send data to server by adding a function for date and time stamp, updating ip address of webserver, and making the file name to be passed dynamic.

Taha – He worked on face recognition part of the lab. He used haarcascades_fontalface.xml and training data set from AT&T public data base for training.

3. Purpose:

In this lab, we aim to design a multithreaded program for an embedded multicore system. We are using POSIX thread libraries for multithreaded programming. We also understand and use mutual exclusion and synchronization of threads to ensure that the program executes in desired manner. We aim to send the sensor data, and other data like pictures taken by webcam using libcurl library, HTTP protocol. We also process the data (webcam picture) using OpenCV functions and make a decision according to the processing result.

4. Introduction:

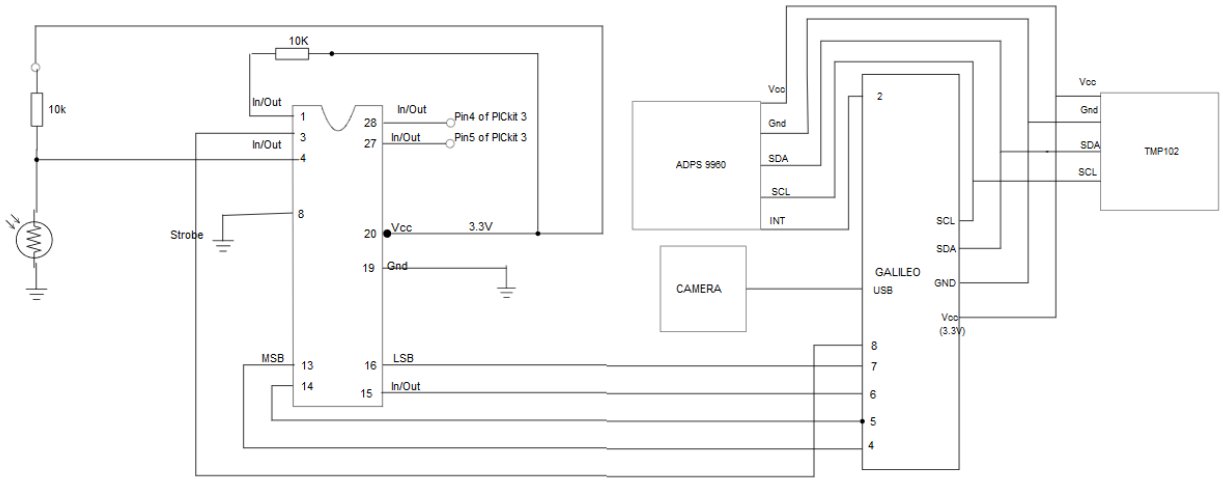
Galileo Gen 2 is now connected to sensors like light intensity sensor, temperature and gesture sensor. A webcam is also connected to Galileo which can be used to obtain images. We use pthreads to create three threads to control different sensors and send data to the webserver. First thread takes the command from the user, communicates with the PIC 16F18857 microcontroller and shows the response and status from light sensor. The second thread monitors the gesture and temperature sensors and if the sensor value exceeds a certain limit, uses the webcam to take a picture and process it for any human faces (this is done by using OpenCV library), on detection of a human face sends a signal to the third thread. The third thread waits for the signal from second thread and on receiving a signal sends the image and the sensor data to the webserver.

5. Materials, Devices and Instruments:

Galileo Gen 2 Board
Serial connector cable
Micro SD card for loading the Linux OS
Computer with putty to connect to Galileo through serial cable or ssh
TMP102 temperature sensor
APDS 9960 Gesture and Proximity sensor
Intel® Wi-Fi Link 5300
2 Wi-Fi Antennas
5 MP webcam
Connecting wires

Bread Board
PIC 16F18857 microcontroller
PicKit3
10KΩ resistors - 2
A Photoresistor

6. Schematics:



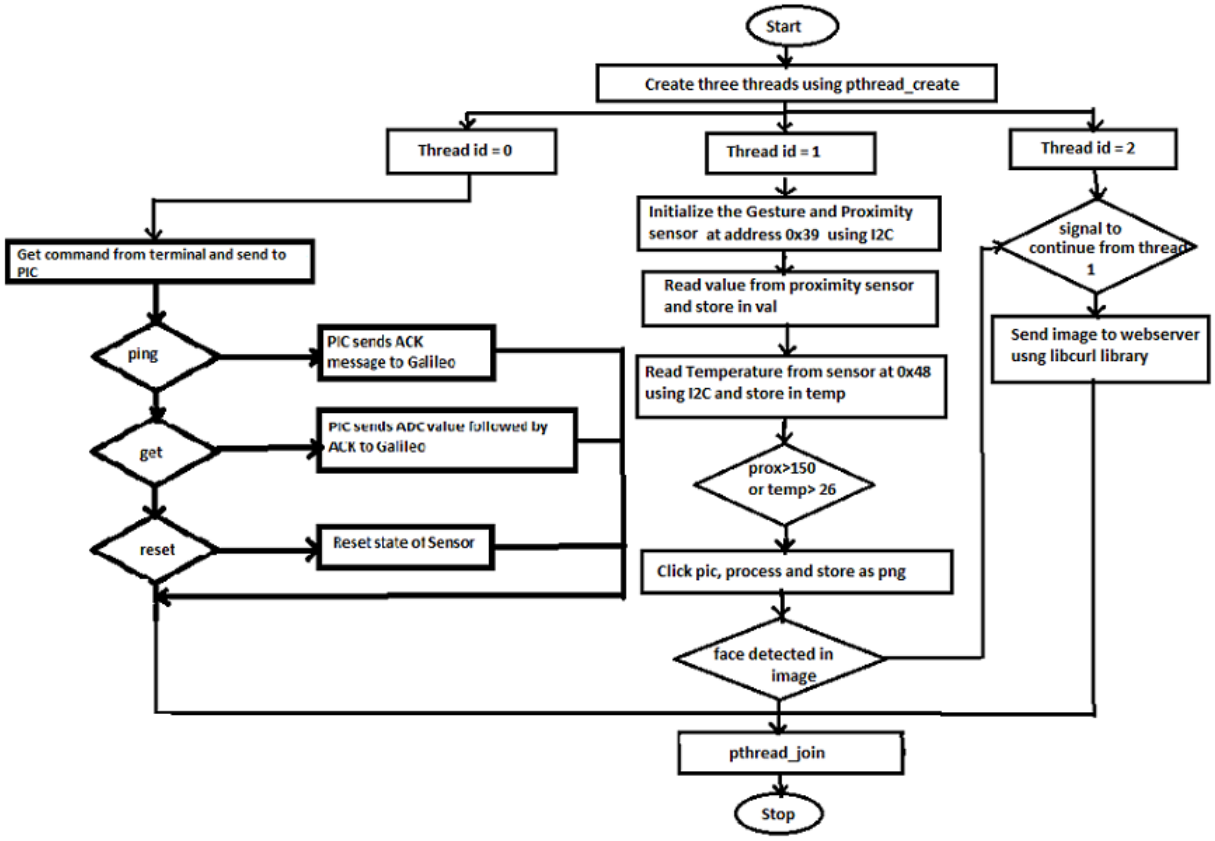
7. Lab Methods and Procedure:

Hardware Design:

The PIC 16F18857 microcontroller and light sensor circuit is connected to Galileo board as in lab 2. The temperature and gesture sensor circuit from lab 3 is also connected to Galileo so that all three sensors are connected to Galileo board. The PIC 16F18857 microcontroller is connected through PicKit3 to a computer with MPLABX to be able to program the PIC microcontroller. We have soldered the temperature and gesture sensor to header pins which helped us in detecting the sensors continuously and without issues.

Software Design:

Flowchart:



In main.c we create three threads, each thread has a specific function which it has to perform before exiting. Thread 0 accepts user commands from the console and sends the command to the PIC microcontroller and displays the result from the PIC microcontroller. Thread 1 communicates with temperature and gesture sensor and based on the values from the sensors signals thread 2 accordingly if a specific condition is met or not. Thread 2 on receiving the signal from thread 1, sends the image captured by thread 1 and sends it to the webserver.

We modified the program from lab 2 for communication with the PIC 16F18857 microcontroller so that we can call from main.c and pass the command given by the user from the terminal. We are also using the modified lab 3 program to communicate with the temperature and gesture sensors and getting the value in man.c so that using `pthread_cond_signal, pthread_cond_wait` we can signal thread 2 to send the image to the webserver using libcurl library. The webserver is started using a Python script provided by the TA.

```
pthread_mutex_t cmutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_var = PTHREAD_COND_INITIALIZER;

int initSensor();                          //Functions to be called
int getProxVal();                          //for sensor data, image
float getTempVal();                        //capture and
int captureAndProcess();                   //processing from
int picCommand(int select);                //external files
extern int call_POST(char* datafile, int adcvalue, char*
sensorstatus);
```

```
void *ThreadWait(void *t) {

    if(threadid == 0) {                     //thread 0
     printf("Please enter the digit corresponding to the command:\n");
     printf("Ping - 1");                    //Ping - 1
     printf("Get ADC value - 2");           //Get ADC value - 2
     scanf("%d", &sel);
     picCommand(sel);                       //Send command to PIC
    }

    if(threadid == 1) {                     //thread 1
        for(k=0; k<10;k++) {
          prox = getProxVal();              //get proximity value
          tempVal = getTempVal();           //get temperature
          FILE *fp =
     fopen("sensor_data.txt", "w+");        //Put sensor values in file
          fprintf(fp, "Proximity value : %d \n", prox);
          fprintf(fp, "Temperature : %f\n", tempVal);
          fclose(fp);
          if(prox > 150 || tempVal > 26) { //condition for image
               toSend =
                    captureAndProcess();  //capture image and
                                          //process for human face
               if(toSend == 1) {          //condition met
                    pthread_mutex_lock(&cmutex);
                    pthread_cond_signal(&cond_var);//signal thread 2
                    pthread_mutex_unlock(&cmutex);
               }
               break;
          }
        }
    }

    if(threadid == 2) {                     //thread 2
     printf("Start sending\n");
     pthread_mutex_lock(&cmutex);
     pthread_cond_wait(&cond_var, &cmutex);//receive signal from
                                          //thread 1
     call_POST("face_recognizer.png", 234, "camera-ok");
//   call_POST("sensor_data.txt", 0, "sensor-data");
     pthread_mutex_unlock(&cmutex);
    }

    pthread_exit(NULL);
}

int main () {

    // Initialize and set thread attributes to "joinable"
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

```
    for( i=0; i < NUM_THREADS; i++ ){       //create 3 threads in loop
        printf("main() is creating thread %d \n", i);
        ret = pthread_create(&threads[i], NULL, ThreadWait, (void *)i
);
        if (ret){
            printf("Unable to create thread %d\n", i);
            return -1;
        }
    }

    // deallocate attribute and wait for the other threads
    pthread_exit(NULL);
}
```

In test_client.h, we are getting the file name and putting the contents of the file in a buffer to be sent to the webserver using curl library. The ip address, id, correct password are needed to post data to the webserver.

```
char *getDateTime() {                   //get time and format timestamp
    char *array = (char*)malloc(sizeof(char)*20);
    memset (array, 0, 20);
    time_t rawtime;
    rawtime = time(NULL);
    struct tm  *timeinfo = localtime (&rawtime);
    strftime(array, 20, "%d.%m.%Y-%H:%M:%S", timeinfo);
    array[20] = '\0';
    return array;
}

void HTTP_POST(const char* url, const char* data, int size) {
     //Use curl library to post the data, file to the webserver
}

int call_POST(char* datafile, int adc, char* sensorstatus){
     const char* hostname="192.168.0.19";  //ip address of webserver
     const int   port=8000;                //port number
     const int   id=9;                     //id
     const char* password="deepblue";      //password
     const char* name="Team+Awesome";
     const int   adcval=adc;               //ADC value if present
     const char* status=sensorstatus;
     const char* timestamp=getDateTime();  //timestamp
     const char* filename=datafile;        //filename

     char buf[1024];
     snprintf(buf, 1024,
"http://%s:%d/update?id=%d&password=%s&name=%s&data=%d&status=%s&times
tamp=%s&filename=%s", hostname, port, id, password, name, adcval,
status, timestamp, filename);
     FILE *fp;
```

```
      //Get the file size and put the data in a buffer and pass it to
      //HTTP_POST

      HTTP_POST(buf, buffer, size);
      fclose(fp);
      //free(buffer);
      return 0;
}
```

In facerec_video.hpp we are capturing the device, taking a frame and analyzing the frame for any human faces. We are doing this by using OpenCV libraries, haarcascade_frontalface_default.xml, training set from AT & T public data set.

```
using namespace cv;
using namespace std;

static void read_csv(const string& filename, vector<Mat>& images,
vector<int>& labels, char separator = ';') {
      //Read the csv file which has paths to the
      //training set
}

int captureAndProcess() {
    string fn_haar =             //haarcascade file
      "haarcascade_frontalface_default.xml";
    string fn_csv = "csv.txt";
    // These vectors hold the images and corresponding labels:
    try {
        read_csv(fn_csv, images, labels);
    } catch (cv::Exception& e) {
        exit(1);
    }
    int im_width = images[0].cols;
    int im_height = images[0].rows;
    // Create a FaceRecognizer and train it on the given images:
    Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
    model->train(images, labels);
    CascadeClassifier haar_cascade;
    haar_cascade.load(fn_haar);

    VideoCapture cap(deviceId);  // Get a handle to the Video device
    if(!cap.isOpened()) {    // Check if we can use this device at all
        return -1;
    }
    Mat frame;                   //Holds the current frame from the
                                 //Video device
    for(i =0;i<1;i++) {
        cap >> frame;
        Mat original = frame.clone(); // Clone the current frame
        Mat gray;                //Convert the frame to grayscale
```

```
        cvtColor(original, gray, CV_BGR2GRAY);
        vector< Rect_<int> > faces; // Find the faces in the frame
        haar_cascade.detectMultiScale(gray, faces);
        if(faces.size() > 0) {
           faceFound = 1;
        }
       for(int i = 0; i < faces.size(); i++) {
            // Process face by face:
            Rect face_i = faces[i];
            // Crop the face from the image. So simple with OpenCV
C++:
            Mat face = gray(face_i);
            Mat face_resized;
            cv::resize(face, face_resized, Size(im_width, im_height),
1.0, 1.0, INTER_CUBIC);
            // Now perform the prediction, see how easy that is:
            int prediction = model->predict(face_resized);
            rectangle(original, face_i, CV_RGB(0, 255,0), 1);
            // Create the text we will annotate the box with:
            string box_text = format("Prediction = %d", prediction);
            // Calculate the position for annotated text (make sure we
don't
            // And now put it into the image:
            putText(original, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0,255,0), 2.0);
        }
      vector<int> cp;
      cp.push_back(CV_IMWRITE_PNG_COMPRESSION);
      cp.push_back(3);
      imwrite("face_recognizer.png", original,cp);//save image
      if(faceFound == 1) {
           break;
      }
    }
    return faceFound;
}
```

8. Trouble Shooting:

We were able to use the libcurl to send data to the webserver using network at home. When doing the same at the University using eduroam network, the sending of the image was failing. The error message specified that the server was not found. As the server was running, we checked the credentials used and the ip address used. We found that the ip address of the server changed as the network connection changed, changing the ip address to the new one fixed the problem and we were able to send the data.

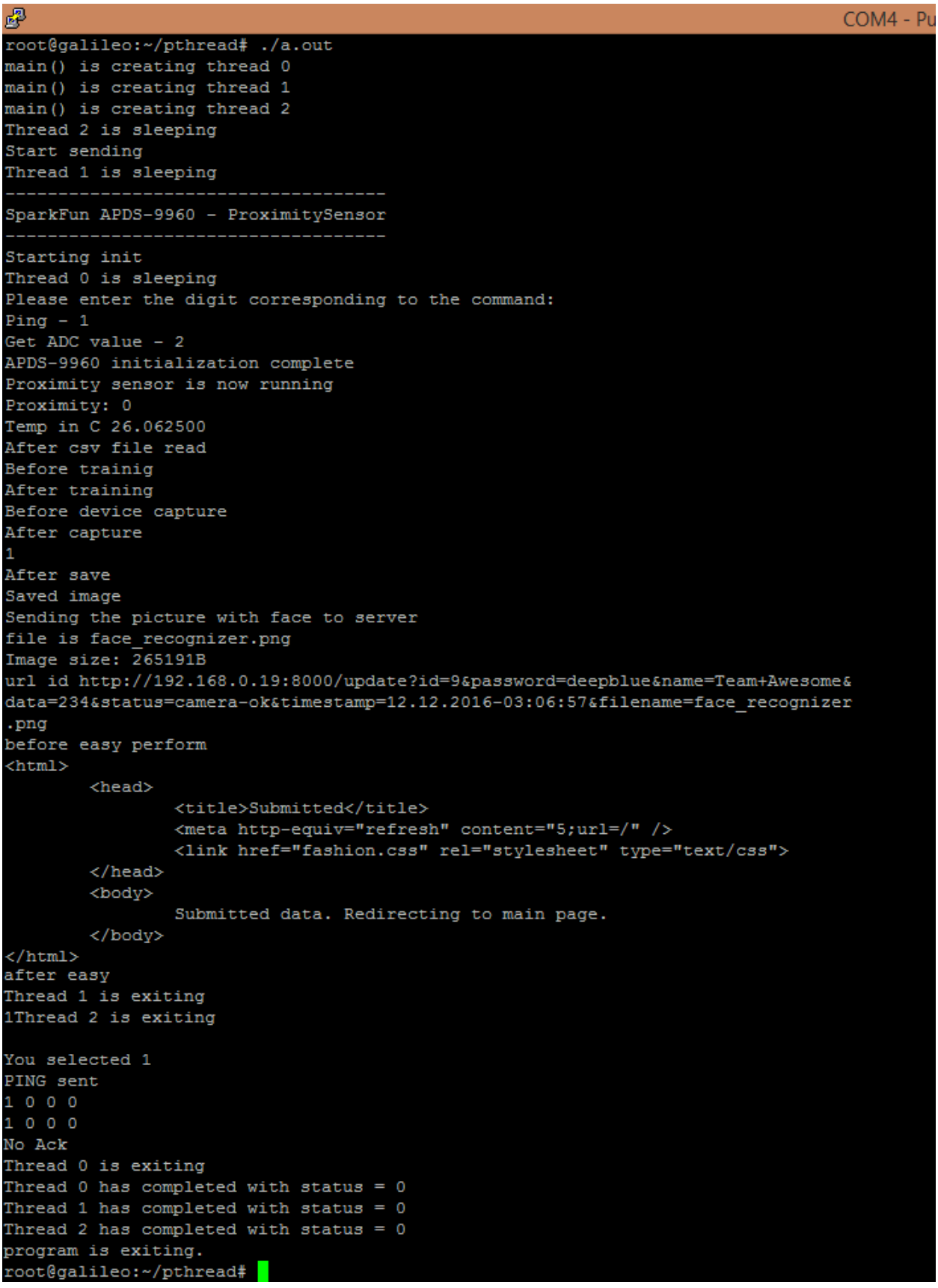
For the face recognition to work, we need a training set of images. These images need to be aligned properly for the program to work. This can be done using normal images and a python script. We also found that AT & T has a public database of images which can be used for face recognition training. This database saved us the effort of aligning and scaling the images.

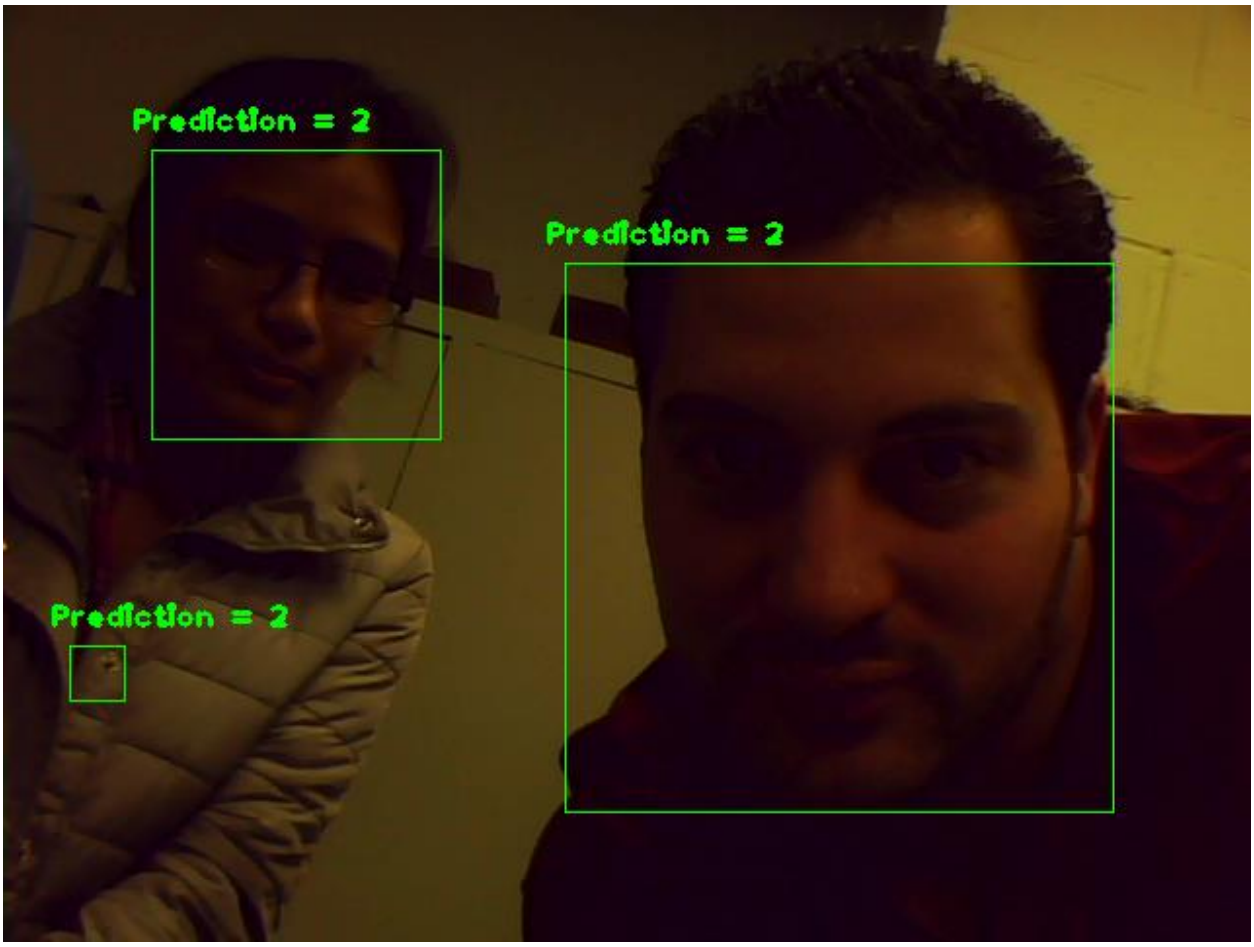
The third thread should wait for the sensor values (monitored my second thread) to reach a certain value to send the image to the webserver. We do not get correct values if we check the condition in the third thread on the values from second thread. Instead we found that we can use `pthread_cond_signal, pthread_cond_wait` along with mutex to ensure that third thread waits for the values of the sensors reach a certain level which is tested in second thread.

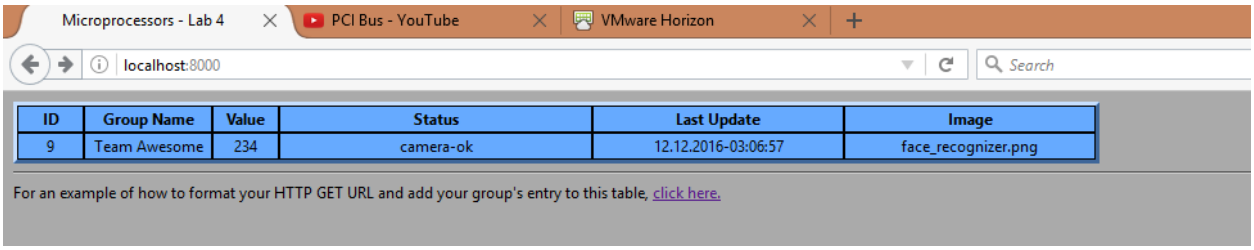
9. Results:

Below is the screenshot of the console output in Galileo, the image captured and sent to webserver. In the image the faces are detected and highlighted.

The last screenshot is of the webserver page which shows the data received from the Galileo.


COM4 - Pu
root@galileo:~/pthread# ./a.out
main() is creating thread 0
main() is creating thread 1
main() is creating thread 2
Thread 2 is sleeping
Start sending
Thread 1 is sleeping
SparkFun APDS-9960 - ProximitySensor
Starting init
Thread 0 is sleeping
Please enter the digit corresponding to the command:
Ping - 1
Get ADC value - 2
APDS-9960 initialization complete
Proximity sensor is now running
Proximity: 0
Temp in C 26.062500
After csv file read
Before trainig
After training
Before device capture
After capture
1
After save
Saved image
Sending the picture with face to server
file is face_recognizer.png
Image size: 265191B
url id http://192.168.0.19:8000/update?id=9&password=deepblue&name=Team+Awesome&
data=234&status=camera-ok&timestamp=12.12.2016-03:06:57&filename=face_recognizer
.png
before easy perform
<html>
<head>
<title>Submitted</title>
<meta http-equiv="refresh" content="5;url=/" />
<link href="fashion.css" rel="stylesheet" type="text/css">
</head>
<body>
Submitted data. Redirecting to main page.
</body>
</html>
after easy
Thread 1 is exiting
1Thread 2 is exiting
You selected 1
PING sent
1 0 0 0
1 0 0 0
No Ack
Thread 0 is exiting
Thread 0 has completed with status = 0
Thread 1 has completed with status = 0
Thread 2 has completed with status = 0
program is exiting.
root@galileo:~/pthread#





10. Appendix:

We are using images from AT&T Database of Faces for training in face recognition.

http://www.cl.cam.ac.uk/research/dtg/attarchive/pub/data/att_faces.zip

To compile the code

g++ main.c test_client.h facerec_video.hpp SparkFun_APDS9960.cpp SparkFun_APDS9960.h temp.c i2c-dev.h i2c-id.h i2c.h ProxSensorCamTemp.hpp gpio-galileo.h -lcurl -lpthread `pkg-config --cflags opencv` `pkg-config --libs opencv`

http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

https://github.com/opencv/opencv/tree/master/data/haarcascades