Junior Design: Home Security System

Matthew Dwyer
Richard Saganey
Ben Tinelli

Embedded and Mobile Systems Laboratory
CSE 398
Dr. Marcy

Table of Contents

Introduction:

In this lab, we decided for our Junior Design Project to make a home security system. Due to time restraints we could only do one part of a modern home security system today. We choose to do a security camera that centered around a Raspberry Pi. It takes pictures of a person outside your home using facial recognition technology and sends an email to you when this happens. The security system was also supposed to support a live stream component to a webpage that you would be able to access via your local area network. Due to some unforeseen issues before building our mobile system we were unable to run the 24/7 live stream in parallel with our email notification protocol. The details of this are in the discussion.

Method:

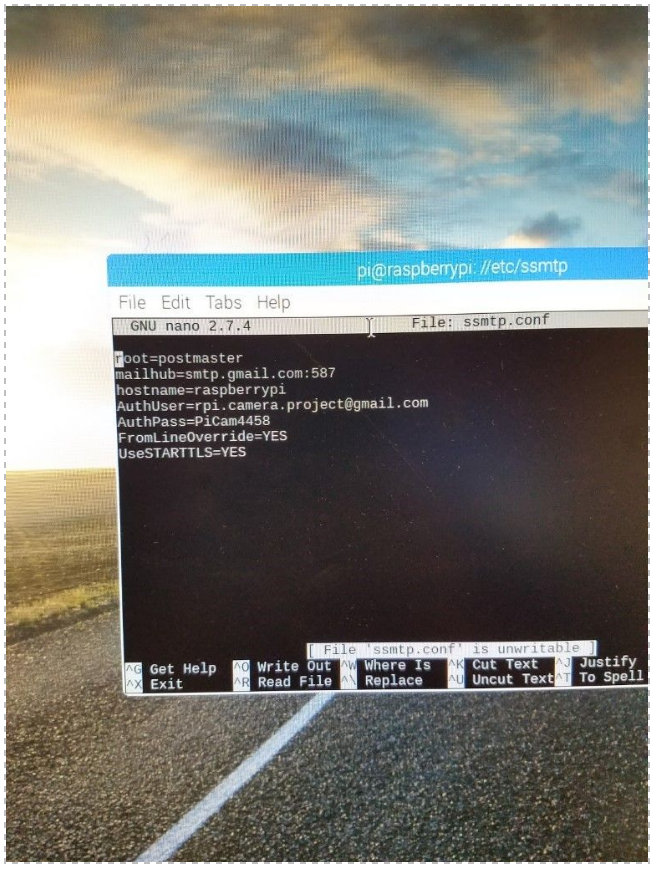
The first part of this project starts like any other lab would, with getting our equipment ready to work with. This starts with flashing the Raspberry Pi 3B+. This was without a doubt the easiest part of the lab. We had already had a flashed SD with the latest image of the OS that we could use from the last lab. We just plugged the SD card into the 3Bs SD slot and booted up the Pi. (This is of course after powering it and setting it up with our computer screen.)

The next part of the project was getting the Raspberry Pi cam working. That way it could take images and video while connected to the Pi. First, we had to connect the camera to the pi. Then we had to go into the raspberry OS and config it so that it would accept and recognize that there was in fact a camera connected to that port. After that we rebooted the board and used the raspistill –o imageNameHere.jpg command to take a picture and the gpicview ImagineNameHere.jpg to view it. We could also use raspvid -t 0 to view a live video of our camera.

For the next part of the lab, we needed to connect to the internet so we could download utilities that will be very useful in our project. For most people this part takes only a few seconds but due to our network's protocol, connecting to the internet takes a little bit longer than usual. We had to follow instructions given by the university to get our Pi connected to our university's network. This involved us finding a file preloaded on our Pi and editing it so that it would automatically connect to the internet. We had to put in the ssid and our username and password into this document. Once this was done we rebooted the Pi and we were good to go.

In this part of the lab, we just created a new gmail for the Pi to use for testing. Since you have to enable letting low trust level applications email you in order to receive automated messages from the Pi, none of us wanted to use our personal or school emails. While this email will act as the receiver it will also be the sender of the email letting the user know that someone is outside.

Now we will setup the automatic emailing feature of our Home Security System. To do this, we first have to download the utilities to create a SSMTP server to send emails. Using the sudo apt-get install ssmtp and sudo apt-get install mailutils commands we can download theses unilties. Once they are downloaded, we need to go into the postmaster file. This can be found in /etc/ssmtp/ssmtp.conf. Once inside the postmaster file, we need to write it to be exactly as it is seen down below.



```
  GNU nano 2.7.4                  File: ssmtp.conf

root=postmaster
mailhub=smtp.gmail.com:587
hostname=raspberrypi
AuthUser=rpi.camera.project@gmail.com
AuthPass=PiCam4458
FromLineOverride=YES
UseSTARTTLS=YES

                          [ File 'ssmtp.conf' is unwritable ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell
```

The only changes you should make is to put in your email you want the email to be sent from and the password to this email. We only tested the Home Security System using Google services so we are not sure what the mail hub port would be for other services.

Now, we will download the library we are going to be using to help us with the facial recognition part of this project. To do this we should use the following commands in this order:

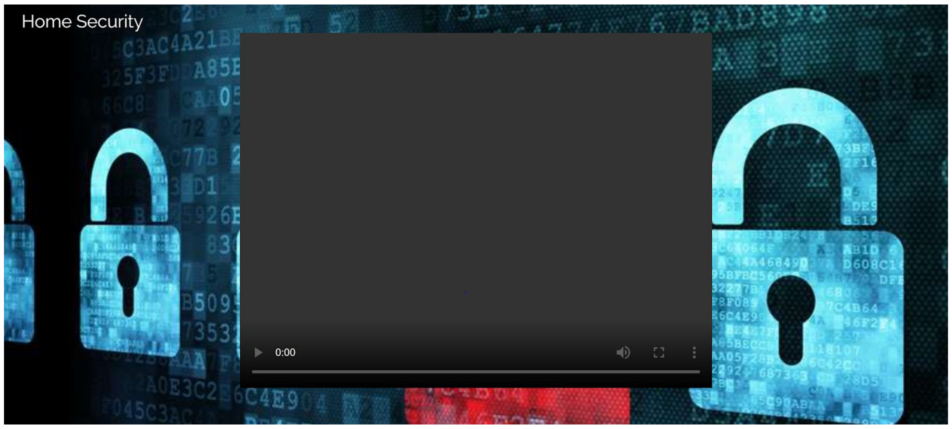
```
sudo apt install libopencv-dev
sudo apt update
sudo apt upgrade
Sudo modprobe bcm2835-v4l2
```

All of these commands will take some time to download, If you have to you to, you can leave your Pi and let the download progress since you only have to give permission to the Pi once. Once this is done, we can move on to the coding part of this project.

This part of the project was by far the hardest part. We first have to go into the OpenCV directory that you should have downloaded in the last part. Once you are there, find your way to the chp15 folder and then go into the OpenCV folder. Once there you will see a program called face.cpp. This is the program that we will be working with to make our camera take pictures and email them to us when it sees a face. Making the new face.cpp involved very few new lines, but the lines we had to add were somewhat complex. We had to add System() lines to call the command line calls we added when we downloaded the mail utilities so we can sent emails. We also needed to add a spam prevention measure to our program, so that the camera wouldn't spam the user with a picture of the same person multiple times. So we used the chrono class to create a timer that would start ticking when the program would run. When this timer would run it had a certain value that would reset and allow you to send an email. For the first email sent this timer would have no effect, but after that the timer would run and prevent emails from being sent out for however long the timer was set for. All of this logic would be within the "face detected" statement in the program. In testing, we used sleep(1000000) or sleep(any number in here really) to make it so that the whole program would stop and no emails would be sent. The issue with this was that the livestream would have to run all the time so we couldn't use sleep otherwise it would stop the livestream as well.

After the facial recognition software was working, we now had to get the web server up and running. To host the website we are going to need to download apache2. If you google "apache2 web server" and click the first link, you should be able to find their website. Once there download apache2 for the linux OS or whatever OS your Pi is running. Once that is done, we will have to configure the web server to host the webpage we want. To do this we go into the apache2 directory and go to www/. There you will find a default .html or .php file. Remove this and replace it with the html page that you are going to be using for our website. Once the web server is working we need to download RPi_Cam_Web_Interface library. This will be used to send the source frames we get from the Pi camera to the web server. Once that is downloaded, you may need to change the source file path so that it can detect your camera. Your camera path should be dev/shm/mjepg/cam. Just add that camera path to the paths.php file and you should be good to go.

For this part of the lab, we had to create a webpage to receive and play the 24/7 live stream being sent out by the Apache2 server on the Pi. This was another simple part of this lab if you know how to write HTML and PHP code. The webpage consisted of three parts. A background for a bit of flair, our project name/ name of the webpage and the live stream.



The live stream was easier than we expected due to a new line we found for live stream playing.
<source src="http://demo.viewcam.me:8000/++hls?cameraNum=1&auth=dXNlcjpwYXNz" type="application/x-mpegURL">
For example you put the link of what server you are sending your data from and it receives and plays it in whatever window size and area you previously wrote in your website.

For the this part of the project we used a Teensy LC and a remote to control the camera mount. This would allow for the camera to cover any angle. We wanted to be able to adjust the angle of our camera remotely, so we first had to figure out how our IR remote communicated information. We figured out that our specific remote used NEC IR protocol. Unfortunately, we could not find any Arduino Libraries that already had decoding for this, so we had to make our own. In order to do this we wired an IR receiver to the Teensy LC such that the signals could be analyzed by the Teesy. We wrote code that had the Teensy print out the values that it interpreted from the receiver. These values were strange characters that did not make much sense, so we added code that converted the symbols into integers. This worked perfectly had gave us unique integers for each button press of the remote.

Each unique value of the remote was used to write if statements that were used to control the cameras pan and tilt. Along with our IR decoding/case code, we integrated the Arduino servo library as well. Aspects of this library were used inside each IR if case. For example, one of these cases made the camera return to its middle position. The if statement had the integer for a button that we desired to do this function. Inside the statement is code that set the x and y position of the camera servo mount to equal 90 (middle position as each servo has a range of 0-180).

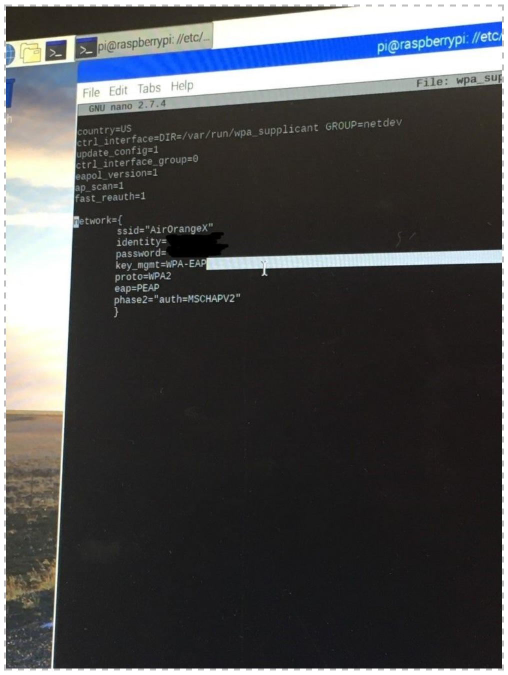
After determining that our code did what we desired, we had to create an enclosure and power supply for the Teensy/servo system. To do this we used a mini breadboard, a black enclosure, a 5v switching power adapter, and lots of hot glue. We got glued the breadboard to the inside of the enclosure and wired up our Teensy and IR receiver. Two holes needed to be drilled into the enclosure so that the power source could be wired to the breadboard and so that the IR receiver wasn't stuck inside the enclosure. After everything was in place, hot glue was added to insulate our circuit and be certain that none of our wires came out or shorted. This completed this aspect of our design.

Results:

For the outcome of the first part, we had a working Raspberry Pi that is all ready to be used and programmed to fit our needs. This was by far the easiest part of the lab.

The outcome of the second part of the lab is a picture taken by the Pi cam and working video stream that is outputted by the terminal command raspvid -t 0 in the Raspberry Pi Terminal.

The results of the third part of the lab was a Raspberry Pi that would automatically connect to the internet when it booted up. Being able to connect to the internet makes the rest of this project possible and without this part the lab would be very hard to do.



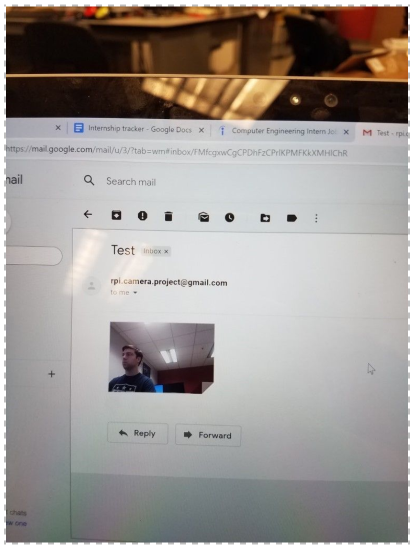
```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
ctrl_interface_group=0
eapol_version=1
ap_scan=1
fast_reauth=1

network={
        ssid="AirOrangeX"
        identity=[redacted]
        password=
        key_mgmt=WPA-EAP
        proto=WPA2
        eap=PEAP
        phase2="auth=MSCHAPV2"
        }
```

This is the file we had to change

The output of the fourth part of the project was a brand new gmail account that could receive emails from low trust applications, such as our Home Security System SSTMP server. We also now had an account to sent emails to users from, so we can effectively test our projects email capabilities.

The outcome of the fifth part of this lab is actually make the SSTMP email server. This server will then handle all of our email notification requests. The server will output emails with images to whoever the postmaster tells it to. The postmaster is just a .conf file in our email program.


Internship tracker - Google Docs
Computer Engineering Intern Jo
Test - rpi
https://mail.google.com/mail/u/3/?tab=wm#inbox/FMfcgxwCgCPDhFzCPrlKPMFKkXMHlChR
Search mail
Test
Inbox
rpi.camera.project@gmail.com
to me
Reply
Forward

An Example of an Email the user would get

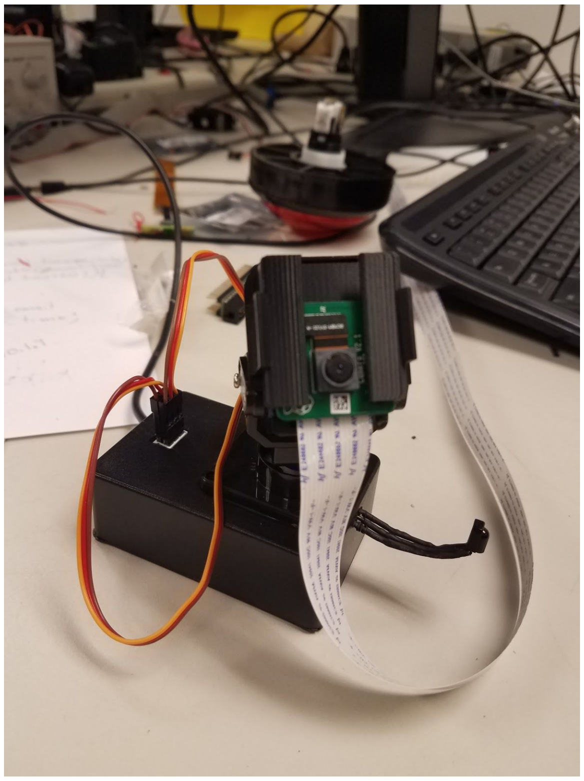
The results of the next part of the lab is that we now have the OpenCV libraries to work with. These libraries will help give us a foundation for our facial recognition software. We can now run basic facial recognition tasks, without any additional code. We will add to this in the next part.

In this part of the lab, the outcome is a modified face.cpp program that can scan a video feed for a face, then take a picture of that face and send the user a picture of that face. The new face.cpp also has a timer that has spam prevention, so that a picture is only sent a maximum of once every X amount of seconds. For testing we used only 100.

The outcome of this part of the lab was a working web server that we could use to host our webpage and a live stream source that we could reference from the camera. The Web server could host any php based or html based page.

The results of the second to last part of the project was a working web page that would take our cam source feed that we are sending to our web server and display it on our website. The webpage also comes with some basic CSS that we borrowed from a free open source web starter.

The results of the final part of the project was a small enclose that had a Teensy LC in it that would control the movement of the camera mount. We made it so we could control it wireless using a basic IR remote. There was, of course an IR receiver/decoder inside the enclosure as well so that the mount would know where the user wanted it to turn.



The Final Product

Discussion:

In this lab we found that researching a product before implementation and design is extremely important. Sadly we learned this the hard way when we determined the reason we couldn't get both of our ideas to work for our security camera was do to the limitations of the camera. The Raspberry Pi camera is fine for doing one task at a time but falls short when attempting two. Most cameras we found out fall short at this. Most cameras can not communicate data to two different file paths. Due to finding this out late in the game we decided that the way to fix this without buying a new camera was to try and put our live stream and email protocol together in one C++ program. After many attempts and many download libraries and programs we found that there was just no way we could find that would allow us to do what we originally wanted. Due to this we determined we could sell the product differently. Instead of having and all in one camera, which is of course the better design, we came up with a pick your own camera design. What we did was make an entirely new flashed SD card. On this one we redid all the steps in the method to get back to where we had a working facial recognition email protocol. Once we had this we saved it all and unplugged the card. Now what we had was a raspberry Pi with the option to change out the SD card depending on what, you the cameras owner wanted. A 24/7 live stream or an email notification with facial recognition .

Appendix:

Our code for the new face.cpp:

```
#include <iostream>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <string.h>
#include <string>
#include <stdlib.h>
#include "RPi.cpp"
#include <chrono>
#include <time.h>
using namespace std;
using namespace std::chrono;
using namespace cv;
#define FALSE 0
#define TRUE 1


int main(int argc, char *args[])
{
```

```
//start the span prevention timer
//auto start = system_clock::now();
//double emailTime = system_clock::now()+60;



//clock_t currentTime =time(0); //timer starts at zero
//double timerTime = difftime(time(0), currentTime);//time that has past since the program has started
//double emailTime = 1.00;



 while(1){
//cout << "The current time is " << endl;
//cout << currentTime  << endl;
  Mat frame;
  VideoCapture *capture;  // capture needs full scope of main(), using ptr
  cout << "Starting face detection" << endl;
  string point="";
  if(argc==2){  // loading image from a file
    cout << "Loading the image " << args[1] << endl;
        frame = imread(args[1], CV_LOAD_IMAGE_COLOR);
  }
  else {
    cout << "Capturing from the webcam" << endl;
        capture = new VideoCapture(0);
        // set any  properties in the VideoCapture object
        capture->set(CV_CAP_PROP_FRAME_WIDTH,1280);   // width pixels
        capture->set(CV_CAP_PROP_FRAME_HEIGHT,720);   // height pixels
        if(!capture->isOpened()){   // connect to the camera
      cout << "Failed to connect to the camera." << endl;
        return 1;
        }
        *capture >> frame;          // populate the frame with captured image
    cout << "Successfully captured a frame." << endl;
  }
  if (!frame.data){
    cout << "Invalid image data... exiting!" << endl;
        return 1;
  }
  // loading the face classifier from a file (standard OpenCV example)
  CascadeClassifier faceCascade;
  faceCascade.load("haarcascade_frontalface.xml");



  // faces is a STL vector of faces - will store the detected faces
  std::vector<Rect> faces;
  // detect objects in the scene using the classifier above (frame,

  faceCascade.detectMultiScale(frame, faces, 1.1, 3,
                      0 | CV_HAAR_SCALE_IMAGE, Size(50,50));
  if(faces.size()==0){
     cout << "No faces detected!" << endl;   // display the image
//test code remove later
//imwrite("capturetest.png",frame);



  }
  // draw oval around the detected faces in the faces vector

  int recsize=0;

  if(faces.size()==1){
            for(int i=0; i<faces.size(); i++)
            {
            // Using the center point and a rectangle to create an ellipse
            Point cent(faces[i].x+faces[i].width*0.5,
                      faces[i].y+faces[i].height*0.5);
      RotatedRect rect(cent, Size(faces[i].width,faces[i].width),0);
            // image, rectangle, color=green, thickness=3, linetype=8
            ellipse(frame, rect, Scalar(0,255,0), 3, 8);
            //cout << "Face at: (" << faces[i].x << "," <<faces[i].y << ")" << endl;
      cout << "face detected" << endl;



 //      cout << "The face is at: (" << faces[i].x+faces[i].width*0.5<< "," <<faces[i].y+faces[i].height*0.5<< ")"
<< endl;



//now we need spam prevention,use a timer to prevent this
//timer code::





//auto start = system_clock::now();
//double emailTime = system_clock::now();+50;



//if(timerTime >= emailTime /*timer is above a certain value */){//time to send an email
//code to take a picture
imwrite("capturetest.png",frame);
      cout <<"Picture taken " << endl;





      cout << "sending Email"<< endl;
//code to send an email
system("mpack -s \"test\" capturetest.png rpi.camera.project@gmail.com");
//simple sleep to prevent spam emails
usleep(100000000);



//Now the email is sent we need to restart the system timer
//timerTime = time(0);
//}
//else{//if the timer is not above that value, do nothing



//auto start = system_clock::now();
//double emailTime = system_clock::now();+60;


//}


        recsize=faces[i].width * faces[i].height;
            int widpoint=faces[i].x+faces[i].width*0.5;
            int htpoint=faces[i].y+faces[i].height*0.5;

            if (widpoint<1000)point+="0";
            if (widpoint<100)point+="0";
            if (widpoint<10)point+="0";
            point+=to_string(widpoint)+" ";
            if (htpoint<1000)point+="0";
            if (htpoint<100)point+="0";
            if (htpoint<10)point+="0";
            point+=to_string(htpoint);

            }

      imwrite("faceOutput.png", frame);


            if (recsize>=50000){
        RPi *r1=new RPi(point);
            string r1m=r1->GetPacket();
            // const char *c1=r1m.c_str();
            }
  }
  capture->release();
  delete capture;
```

```
 }
  //End while(1) loop

  return 0;
}
```

Code object we needed to run face.cpp (from last lab)

Final code for RPi.cpp:

```
#include <stdio.h>
#include <string.h>
#include <string>
#include <iostream>
using namespace std;
class RPi{
public:
RPi(string m){
Data=m;
length_data_int=m.length();
combination=Header;
string zero="0";
if(length_data_int<100){combination+=zero;}
if(length_data_int<10){combination+=zero;}
combination=combination+to_string(length_data_int)+Data;
int sum=0;
for(int j=0;j<combination.length();j++){
sum+=combination.at(j);
}
checksum=sum%256;
combination_real=combination;
if(checksum<100){combination_real+=zero;}
if(checksum<10){combination_real+=zero;}
combination_real=combination_real+to_string(checksum)+"\n";
}
void SetChecksum(int i){
string zero="0";
combination_real=combination;
if(i<100){combination_real+=zero;}
if(i<10){combination_real+=zero;}
combination_real=combination_real+to_string(i)+"\n";//might be atoi (i)
}
string GetPacket(){
return combination_real;
}
~RPi();
private:
string Header="RPi";
string Data;
int length_data_int;
string combination;
int checksum;
string combination_real;
};
```

HMTL code for our website:

HTML code for our webpage:

```
<!DOCTYPE html>
<html>
<title>Home Security</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
<style>
body,h1 {font-family: "Raleway", sans-serif}
body, html {height: 100%}
.bgimg {
  background-image:
url('https://www.rockwellautomation.com/resources/images/rockwellautomation/publication/SecurityCap
ability-Feature2--photograph_848w477h.jpg');
  min-height: 100%;
  background-position: center ;
  background-size: cover;
}
</style>
<body>


<div class="bgimg w3-display-container w3-animate-opacity w3-text-white">
  <div class="w3-display-topleft w3-padding-large w3-xlarge">
```

```
            Home Security
  </div>
  <div class="w3-display-middle">
            <video controls autoplay width="640" height="480">
<source src="http://demo.viewcam.me:8000/++hls?cameraNum=1&auth=dXNlcjpwYXNz"
type="application/x-mpegURL">
  </video>
  </div>
  </div>



</body>
</html>
```

PHP code from web server and stream was sourced from apache2 and the RPi_Cam_Interface.

```
#include <IRremote.h>
#include <Servo.h>

Servo updown;
Servo leftright;

const int ir= 7;
IRrecv irrecv(ir);
decode_results results;

int input=0;
int vpos=90;
int hpos=90;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
  leftright.attach(9);
  updown.attach(10);
  updown.write(90);
  leftright.write(90);
```

```
}



void loop(){
 if (irrecv.decode(&results)){
     Serial.println(results.value, DEC);//
     input=(results.value);
     Serial.println(input);
     moveCam();
     irrecv.resume();
     //delay(600);
     }
}




void moveCam(){
 if(input==16625743)//down
      {
      if(vpos<180){
        vpos+=10;
        updown.write(vpos);
        return;
        }
      }//end down

 else if(input==16621663)//up
      {
      if(vpos>0){
        vpos-=10;
        updown.write(vpos);
        return;
      }
      }//end up
 else if(input==16584943)//left
     {
        if(hpos<180){
          hpos+=10;
          leftright.write(hpos);
          return;
          }
        }//end left

else if(input==16601263)//right
        {
        if(hpos>0){
          hpos-=10;
          leftright.write(hpos);
          return;
          }
         }//end right
 else if(input==16617583){
  updown.write(90);
  leftright.write(90);
  hpos=90;
  vpos=90;
  }
}
```