Final Report
Project: RasBot

Advisor:
Prof. Yanyong Zhang

Team Members:
Alexander Hill,
Haoyang Yu,
Stefan Juang,
Lingnan Meng

Table of Contents
Abstract:

For our project we created a rover, RasBot, that is capable of sending video through wireless internet to an external computing device that does the image processing. This cloud based computing platform allows its user a cheap and easy way to do remote image processing. Multiple users can also interact with the rover through our website. In case if
the user want more than just drive around the rover, there is also the possibility of end-user add on detection.

The RasBot can stream real time video with little to no delay. The live streaming video can be viewed on our control panel web page. The aim of this project is to design a user-affordable and easy-to-use device that can replace people going to places where potential danger lies. Soldier can also use it in search and rescue missions of a disaster and/or on war field.

Of course what we have done is only one of many utilities that is possible to realize on this amazing platform. In the future work could include customizing the image processing to fit the individual needs of the application.

**Cost/Sustainability Analysis:**

One of the major motivations for this project was reducing the overall system cost by reducing the CPU power needed on the ground for robots that go into dangerous environments. For example, a system that previously would need high end CPU performance for the desired image processing on the ground is replaced with a relatively inexpensive raspberry pi. The Raspberry pi allows for a linux operating system to control the peripherals while still giving access to basic embedded system I/O ports including SPI
and I2C. This means that overall development time can be shortened, by utilizing the already developed drivers for many common peripherals.

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-Project 16 Value Resistor Kit</td>
<td>9.99</td>
</tr>
<tr>
<td>Raspberry Pi 5MP Camera Board Module</td>
<td>30.98</td>
</tr>
<tr>
<td>2 Duracell 9V Rechargeable Batteries</td>
<td>24.30</td>
</tr>
<tr>
<td>Raspberry Pi Model B</td>
<td>40.49</td>
</tr>
<tr>
<td>4 Amico DC Geared Motor w Rubber Wheel</td>
<td>49.04</td>
</tr>
<tr>
<td>TN141 2 Bay 9V Smart Battery Charger</td>
<td>10.90</td>
</tr>
<tr>
<td>Long Range WiFi USB with Antenna for Raspberry Pi</td>
<td>22.95</td>
</tr>
<tr>
<td>Transcend 16GB Class 10 Flash Memory Card</td>
<td>11.99</td>
</tr>
<tr>
<td><strong>Total Amount</strong></td>
<td><strong>200.64</strong></td>
</tr>
</tbody>
</table>

System Overview:
In general the rasbot relies heavily on the internet and modern internet technologies to allow for the remote data processing and control. When the rasbot is started it connects to an available wireless network and contacts the public server to begin listening for commands and uploading live video. The public server receives the video stream from the rasbot and splits the stream into two streams; one for image processing and the other for the web site. In addition to handling the video, the server receives data and commands from clients and echos those commands out to all of the clients. Commands for the rasbot can be generated from scripts running on the server.
or from users logged in on the website. End users can control and monitor the rasbot through the website running javascript code that establishes a connection between the server and the end user and allows for commands to be sent to the server to be echoed back to the rasbot.

1. Hardware implementation
1.1 Control Part - Raspberry Pi Board (Model B)
Raspberry Pi is a credit card-sized single board computer whose operating system is Arch Linux. In this project, it is the brain of the Rosbot, using CSI connector camera, USB 2.0 and GPIO headers to give commands to control the robot. CSI connector connects to raspberry pi 5MP board module to get instant video. USB 2.0 connects to long range WiFi Antennas to implement wireless communication with the server. The moving system’s control depends on the voltage of the GPIO pins which is set by broad chip.

1.2 Motor system
The moving system is made up of seven members: two L298N stepper motor driver controller boards, one protection circuit and four 1500 - 10000RPM 6V DC geared motor w rubber wheels. To control the moving operations of the Rosbot, GPIO Pins give changing voltages to two motor drivers. Then one motor driver connects to two motors attached in two wheels to make them go forward, backward or stop through the protection circuit.

1.3 Wireless Part
In this part, Long range WiFi USB with Antenna for Raspberry Pi is used as hardware to do wireless communication with the server. Its interface is high speed 2.0 and its frequency range is between 2412 and 2462 MHZ. The network can auto-switch to use 802.11n or 802.11g or 802.11b mode.

1.4 Power Support System

There are two parts in the power support system: two 9V rechargeable batteries, one 10000mAh USB Mobile charger. The mobile charger provides power for the raspberry pi to support its decoding, wireless communication and video collecting. The two batteries can keep four motor running at the same time. This power system is very powerful and can keep the Rasbot running for 8 ~ 10 hours.

2.1 Raspberry Pi System

![Figure 6. The control system in Raspberry Pi](image)

2.1.1 GPIO Setting

In this part, we convert the five moving commands from the server( "forward", "backward", “turn left”, “turn right”, “stop”) to GPIO output voltages for the moving system to recognize.

Since there are four inputs for both motor driver circuits, we use four pins as voltage outputs in the raspberry pi. In this project, we use pin 19, 21, 22, 23. Raspberry pi is a mini Linux computer and both python code and java script code are written to implement the GPIO control. Because the last edition used in the raspberry pi is java script code, java script implementation will be mainly introduced below.
First, a GPIO library which is designed by the company especially for GPIO commands should be installed in the raspberry pi. We used pi-gpio library in this project and the installation command in arch Linux is "$ npm install pi-gpio”. After successful installation, we can use the commands in the library. The main commands are as below.

`gpio.open(pinNumber, [options], [callback])`  //open the GPIO pin

`gpio.close(pinNumber, [callback])`  //close the GPIO pin

`gpio.read(pinNumber, [callback])`  //read the value of the pin

`gpio.write(pinNumber, value, [callback])`  //set the pin value

In the Rasbot, the left two wheels share two pin outputs and the right two share two pin outputs. If the Rosbot should turn left, just set the left two pins “go backward”( backward pin high and forward low) and the right two pins “go forward”( forward pin high and backward pin low) and vice versa. In this part, I have two warnings: 1) The GPIO pin number used in the commands should be physical pin numbers, not functional pin numbers. 2) Once you use gpio.open command to open one pin, you can not open one pin twice, or it will give you an error. 3) When you install the GPIO library, please make sure the time setting in the raspberry pi is right.

### 2.1.2 Wireless Communication

In wireless part, we use web socket to do the communication with the server. Web socket is a protocol providing full-duplex communication channels over a single TCP connection. To establish a Web Socket handshake request, for which the server returns a web socket handshake response. There we can have an example:

Raspberry pi request:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JjHMbDL1Ezlkh9GBhXDw==
```
In the project, we implement a web socket communication using java script code and combine it with the GPIO decoding code. So every time the raspberry pi get a command from the server (like “forward”), the GPIO voltage changes and the Rasbot moves as order. We have already attached our code at the appendix part, everyone can read it as a reference.

**Video System:**

In order to allow for computationally heavy image processing to be off loaded to a remote server, and for global control and monitoring, it was critical to develop a method of streaming video with low latency. For point to point applications in a local subnet it is possible to stream the raw data using ncat, ssh or some other similar application and then play the video on the remote computer using a media player. This approach causes several limitations including: (1) network limitations from firewalls and routing standards implemented, (2) in general only one computer receives the video, (3) end users must have a media player that can decode h.264 video, and (4) the video stream frequently suffers from several seconds of latency from the media player buffering the video.

After trying many derivations of the above topology it became clear that the system would need to be structured differently to achieve the overall goals. This new layout included a publicly addressable server that could receive the video from the rasbot and then split the stream for web and image processing use. The raspberry pi allows for the camera to be started by a program called raspvid; the output of this command is h.264 video and is sent directly to the server via the netcat utility. Initial streaming attempts used ffmpeg to transcode the video from h.264 to mjpeg on the raspberry pi before sending the images to the server. This method proved too involved for the raspberry pi, often causing the frames to freeze even if the image size was set to 340 x 280. To resolve these poor performance problems the transcoding was shifted from the raspberry pi to the remote server using netcat. The command that listens for the video on the server side is as follows:

```
ncat -v -l 5001 | ~/bin/ffmpeg -i -f mpeg1video -b 900k
http://ram.labhill.com:8082/password/620/480
```

This accepts the raw h.264 video on port 5001 and pipes the output to ffmpeg to be transcoded into mpeg1 video and then sent to the second process listening on port 8082 for the video to split out for the web page. If image processing is used it can be added by
having a second output from the ffmpeg command giving the data to the image processing code. The video is sent out to the web page using a websocket to transport the video to the clients browser where it is decoded and displayed using a javascript image decoding library. One of the challenges experienced while testing this system was the fact that ffmpeg supplied by the package manager pacman is often out of date, causing many image streaming attempts to fail. To resolve this, the source code for ffmpeg and its needed libraries had to be compiled and installed manually. After learning these things, the video system meet all expectations with low latency in the range of a few tenths of a second.

The Data / Command System:
To allow for flexible remote control, it was necessary to create a system that could send arbitrary data to all devices in a timely manner. This is accomplished by the CommandServer.js code show in the appendix. This code listens on a port for all connections and relays any received data to all other devices through a bi-directional websocket. This method proved useful as most high level languages have libraries for websockets allowing for application specific code to access send and receive data from the robot. In cases where the programming environment does not support websockets, it is also possible to pass data into the system via a http get request. When this method is used the data is taken from the GET request and send in the data system just as if it came from a client that supported websockets. The rasbot runs a javascript program that connects to this system and interprets some basic commands like stop, forward, and close into actions this however can be added to easily.

Results and Future Work:
- Results:
Overall the rasbot system fulfills many of the the basic functions that it was set out to complete. However, unexpected complications caused more advanced areas to suffer. The system allows for users around the globe both the ability to have a live first person view from the rasbot and to control the rasbot.

- Future Work:
Future work would include perfecting the image processing system and improving the web interface. Of course, other than the software improvements, there are many other devices that we hope to add to our rasbot as well. Such as ultrasonic tracker, microphone and perhaps a pair of robotic arms to help user reach out to the environment with greater degrees of control and freedom.

Appendix A -- Code Sections

MatlabImageProcessing.m
A part of the laser tracking code is an open source from Arindam Bose, I have modified the original program to what is now shown below to fit our project needs. The link to the original program can be found in the bibliography.

```matlab
% Program Name : Red Object Detection and Tracking
% Author       : Arindam Bose
% Version      : 1.15
% Description  : How to detect and track red, green and blue objects in Live Video
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

- %% Initialization
- redThresh = 1; % Threshold for red detection
- greenThresh = 0.05; % Threshold for green detection
- blueThresh = 0.15; % Threshold for blue detection

- vidDevice = imaq.VideoDevice('macvideo', 1, 'YCbCr422_640x480', ...
  % Acquire input video stream ARGB32_1920x1080 YCbCr422_640x480
  'ROI', [1 1 640 480], ...
  'ReturnedColorSpace', 'rgb');
- vidInfo = imaqhwinfo(vidDevice); % Acquire input video property
- hblob = vision.BlobAnalysis('AreaOutputPort', false, ...
  % Set blob analysis handling
  'CentroidOutputPort', true, ...
  'BoundingBoxOutputPort', true, ...
  'MinimumBlobArea', 600, ...
  'MaximumBlobArea', 3000, ...
  'MaximumCount', 10);
- hshapeinsBox = vision.ShapeInserter('BorderColorSource', 'Input port', ...
  % Set box handling
  'Fill', true, ...
  'FillColorSource', 'Input port', ...
  'Opacity', 0.4);
- htextinsRed = vision.TextInserter('Text', 'Red : %2d', ...
  % Set text for number of blobs
  'Location', [5 2], ...
  'Color', [1 0 0], ... // red color
  'Font', 'Courier New', ...
  'FontSize', 14);
- htextinsGreen = vision.TextInserter('Text', 'Green : %2d', ...
  % Set text for number of blobs
  'Location', [5 18], ...
```
'Color', [0 1 0], ... // green color
'Font', 'Courier New', ...
'FontSize', 14);
htextinsBlue = vision.TextInserter('Text', 'Blue : %d', ... % Set text for number of blobs
'Location', [5 34], ...
'Color', [0 0 1], ... // blue color
'Font', 'Courier New', ...
'FontSize', 14);
htextinsCent = vision.TextInserter('Text', '+ X:%d, F:%d', ...
% set text for centroid
'LocationSource', 'Input port', ...
'Color', [1 1 0], ... // yellow color
'Font', 'Courier New', ...
'FontSize', 14);
hVideoIn = vision.VideoPlayer('Name', 'Final Video', ... % Output video player
'Position', [100 100 vidInfo.MaxWidth+20 vidInfo.MaxHeight+30]);
nFrame = 0; % Frame number initialization
storeGreen = zeros(2); % initialize empty matrix
% Processing Loop
while (nFrame < 450)
    rgbFrame = step(vidDevice); % Acquire single frame
    rgbFrame = flipdim(rgbFrame,2); % obtain the mirror image for displaying
    
    diffFrameRed = imsubtract(rgbFrame(:,:,1), rgb2gray(rgbFrame)); % Get red component of the image
    diffFrameRed = medfilt2(diffFrameRed, [3 3]); % Filter out the noise by using median filter
    binFrameRed = im2bw(diffFrameRed, redThresh); % Convert the image into binary image with the red objects as white

    diffFrameGreen = imsubtract(rgbFrame(:,:,2), rgb2gray(rgbFrame)); % Get green component of the image
    diffFrameGreen = medfilt2(diffFrameGreen, [3 3]); % Filter out the noise by using median filter
    binFrameGreen = im2bw(diffFrameGreen, greenThresh); % Convert the image into binary image with the green objects as white
diffFrameBlue = imsubtract(rgbFrame(:,:,3), rgb2gray(rgbFrame));  % Get blue component of the image

diffFrameBlue = medfilt2(diffFrameBlue, [3 3]);  % Filter out the noise by using median filter

binFrameBlue = im2bw(diffFrameBlue, blueThresh);  % Convert the image into binary image with the blue objects as white

[centroidRed, bboxRed] = step(hblob, binFrameRed);  % Get the centroids and bounding boxes of the red blobs

centroidRed = uint16(centroidRed);  % Convert the centroids into Integer for further steps

[centroidGreen, bboxGreen] = step(hblob, binFrameGreen);  % Get the centroids and bounding boxes of the green blobs

centroidGreen = uint16(centroidGreen);  % Convert the centroids into Integer for further steps

if (nFrame > 30)  %set up time
    if (nFrame == 31)
        disp ('Detection has started')
    end
    fprintf('nFrame is', nFrame, 'n');

if (storeGreen == zeros(2))  %attention for laser position
    disp ('Reposition the laser, no laser point in sight')
end

if (numel(centroidGreen) ~= 0)
    if mod(nFrame,2)  %test if frame is odd
        storeGreen(1,1) = centroidGreen(1);  %store x value when frame is odd
        storeGreen(1,2) = centroidGreen(2);  %store y value when frame is odd
    else  %frame that is even
        storeGreen(2,1) = centroidGreen(1);  %store x value when frame is even
        storeGreen(2,2) = centroidGreen(2);  %store y value when frame is even
    end

    % Detect if laser point drift up

    height_diff(nFrame + 3) = storeGreen(1,2) - storeGreen(2,2);

    if (height_diff(nFrame + 3) > height_diff(nFrame + 2))  %compare current frame with the previous frame
        if (height_diff(nFrame + 2) > height_diff(nFrame + 1))
            %web
            ('https://ram.labhill.com/data?user=matlab&up=one&down=zero')
- str = urlread('http://ram.labhill.com:8000/','Get',{user,'matlab',up,'one','down','zero'});
- % imwrite(X,map,'myclown.png')
- disp('laser moved up')
- end
- end
- if (height_diff(nFrame + 3) < height_diff(nFrame + 2))
  % compare current frame with the previous frame
  if (height_diff(nFrame + 2) < height_diff(nFrame + 1))
    web('https://ram.labhill.com/data?user=matlab&up=zero&down=one')
    str = urlread('http://ram.labhill.com:8000/','Get',{user,'matlab',up,'zero','down','one'});
    % imwrite(X,map,'myclown.png')
    disp('laser moved down')
  end
- end
- end
- [centroidBlue, bboxBlue] = step(hblob, binFrameBlue); % Get the centroids and bounding boxes of the blue blobs
- centroidBlue = uint16(centroidBlue); % Convert the centroids into Integer for further steps
- rgbFrame(1:50,1:90,:) = 0; % put a black region on the output stream
- vidIn = step(hshapeinsBox, rgbFrame, bboxRed, single([1 0 0])); % Insert the red box
- vidIn = step(hshapeinsBox, vidIn, bboxGreen, single([0 1 0])); % Insert the green box
- vidIn = step(hshapeinsBox, vidIn, bboxBlue, single([0 0 1])); % Insert the blue box
  for object = 1:length(bboxRed(:,1)) % Write the corresponding centroids for red
    centXRed = centroidRed(object,1); centYRed = centroidRed(object,2);
    vidIn = step(htextinsCent, vidIn, [centXRed centYRed], [centXRed-6 centYRed-9]);
- end
- for object = 1:length(bboxGreen(:,1)) % Write the corresponding centroids for green
    centXGreen = centroidGreen(object,1); centYGreen = centroidGreen(object,2);
    vidIn = step(htextinsCent, vidIn, [centXGreen centYGreen], [centXGreen-6 centYGreen-9]);
- end
for object = 1:length(bboxBlue(:,1))  % Write the corresponding centroids for blue
    centXBlue = centroidBlue(object,1); centYBlue = centroidBlue(object,2);
    vidIn = step(htextinsCent, vidIn, [centXBlue centYBlue], [centXBlue-6 centYBlue-9]);
end
vidIn = step(htextinsRed, vidIn, uint8(length(bboxRed(:,1))));  % Count the number of red blobs
vidIn = step(htextinsGreen, vidIn, uint8(length(bboxGreen(:,1)))); % Count the number of green blobs
vidIn = step(htextinsBlue, vidIn, uint8(length(bboxBlue(:,1))));  % Count the number of blue blobs
step(hVideoIn, vidIn);  % Output video stream
nFrame = nFrame+1;
end

%% Clearing Memory
release(hVideoIn);  % Release all memory and buffer used
release(vidDevice);
%clear all;
%clc;

ColorTracking.py

#Laser Tracking
#Haoyang Yu
#We should carefully use THE HSV color detection code written by achuwilson to get exact HSV for what we want to detect
import cv2
import cv
import urllib
import numpy as np

def getthresholdedimg(hsv):
    #blue = cv2.inRange(hsv,np.array((100,100,100)),np.array((120,255,255)))
    #green = cv2.inRange(hsv,np.array((75,10,230)),np.array((10,50,255)))
    #red = cv2.inRange(hsv,np.array((0,10,230)),np.array((10,50,255)))

    green = cv2.inRange(hsv,np.array((35,100,100)),np.array((77,255,255)))
    return green

#Alex, you just need to change the URL value to test!
stream = urllib.urlopen('http://myapplecam.com/mjpg/video.mjpg')
bytes=''
while True:
    bytes+=stream.read(1024)
    # Find the mjep one by one!
    a = bytes.find('\xff\xd8')
    b = bytes.find('\xff\xd9')
    if a!=-1 and b!=-1:
        jpg = bytes[a:b+2]
        bytes= bytes[b+2:]
        i = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8),cv2.CV_LOAD_IMAGE_COLOR)
        #cv2.imshow('img',i)
        #width,height = i.get(3),i.get(4)
        #print "frame width and height : ", width, height

        #_f = i.read()
        #i = cv2.flip(i,1)
        blur = cv2.medianBlur(i,5)
        hsv = cv2.cvtColor(i,cv2.COLOR_BGR2HSV)
        both = getthresholdedimg(hsv)
        # Do image improvement
        # Erode and dilate
        erode = cv2.erode(both,None,iterations = 3)
        dilate = cv2.dilate(erode,None,iterations = 10)
        contours,hierarchy = cv2.findContours(dilate,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

        for cnt in contours:
            x,y,w,h = cv2.boundingRect(cnt)
            cx,cy = x+w/2, y+h/2

            if 35 < hsv.item(cy,cx,0) < 77:
                cv2.rectangle(i,(x,y),(x+w,y+h),[255,0,0],2)
                print "green :", x,y,cx,cy
                cv2.imshow('img',i)

    if cv2.waitKey(25) == 27:
        break

cv2.destroyAllWindows()
#c.release()
// Haoyang Yu: websocket Lingnan Meng: gpio_control
// Act as
// Include the "socket.io-client"
var io = require('socket.io-client');
var socket = io.connect('http://ram.labhill.com:8080/');

// Define the GPOS
var gpio = require('pi-gpio');
var message = "forward";

var gpioPin19 = 11; // 19 -> 11 [ Blue ]
var gpioPin22 = 15; // 22 -> 15 [ Green ]
var gpioPin23 = 16; // 23 -> 16 [ Orange ]
var gpioPin24 = 18; // 24 -> 18 [ White ]

socket.on('connect', function() {
  console.log('Connection Established! Ready to receive data!');
  socket.emit('MessageFromClient', {user: 'PI', Message: "Rpi is Awake and connected:'})
  gpio.open(gpioPin19, "output", function(err) {});
  gpio.open(gpioPin22, "output", function(err) {});
  gpio.open(gpioPin23, "output", function(err) {});
  gpio.open(gpioPin24, "output", function(err) {});
  console.log("open_test");
});

socket.on('message', function(message) {
  console.log(message.Message);
  switch(message.Message) {
    case "open":
      // Open all the Gpios pins.
      // Make them to Output Direction
      gpio.open(gpioPin19, "output", function(err) {});
      gpio.open(gpioPin22, "output", function(err) {});
      gpio.open(gpioPin23, "output", function(err) {});
      gpio.open(gpioPin24, "output", function(err) {});
      console.log("open_test");
      break;
    case "stop":
      gpio.write(gpioPin19, 0, function(err, value) {});
  }
});
case "forward":
    gpio.write(gpioPin19,1,function(err,value){});
gpio.write(gpioPin22,0,function(err,value){});
gpio.write(gpioPin23,1,function(err,value){});
gpio.write(gpioPin24,0,function(err,value){});
    console.log("forward_test");
    break;

case "backward":
    gpio.write(gpioPin19,0,function(err,value){});
gpio.write(gpioPin22,1,function(err,value){});
gpio.write(gpioPin23,0,function(err,value){});
gpio.write(gpioPin24,1,function(err,value){});
    console.log("backward_test");
    break;

case "left":
    gpio.write(gpioPin19,0,function(err,value){});
gpio.write(gpioPin22,1,function(err,value){});
gpio.write(gpioPin23,1,function(err,value){});
gpio.write(gpioPin24,0,function(err,value){});
    console.log("left_test");
    break;

case "right":
    gpio.write(gpioPin19,1,function(err,value){});
gpio.write(gpioPin22,0,function(err,value){});
gpio.write(gpioPin23,0,function(err,value){});
gpio.write(gpioPin24,1,function(err,value){});
    console.log("right_test");
    break;

case "close":
    gpio.close(gpioPin19);
gpio.close(gpioPin22);
gpio.close(gpioPin23);
gpio.close(gpioPin24);
    console.log("close_test");
    break;

socket.emit('MessageFromClient',{user:"RPI",Message:"Shutting Down NOW!"});
var spawn = require('child_process').spawn;
ls = spawn('shutdown', ['-h','now']);
break;
}
socket.on('disconnect', function() {
  console.log('my connection dropped');
});

var UpTime = 0;
function SendHB(){
  UpTime = UpTime + 1;
  socket.emit('MessageFromClent',{user:"HB",Message:UpTime});
  console.log("HB");
}

function StartHB() {
  SendHB();
  setInterval(SendHB, 1000);
}
StartHB();

StartSystem.sh located under /usr/local/bin

#!/bin/bash

echo "Starting the system"
LOGFILE=/root/StartSystem.log

cp $LOGFILE $LOGFILE.old

echo "This is an auto generated log file from the start up script /usr/local/bin/StartSystem.sh" > $LOGFILE

##Check networks: if eth0 is up use that otherwise use wireless
rm /tmp/eth0.txt
/usr/bin/ip link set eth0 up
dhcpcd eth0
sleep 10
ifconfig eth0 > /tmp/eth0.txt

grep inet /tmp/eth0.txt

if [[ $? -eq 0 ]] ## We have eth0 up so lets use it then
  ## any configuration needed for ethernet
  echo "Using Ethernet..." >> $LOGFILE
else
  ## find avalible wireless
  ## Is LAWN avalable

20 of 27
```bash
rm /tmp/nets.txt
/usr/bin/ip link set wlan0 up
iwlist wlan0 scan > /tmp/nets.txt
grep Yavanna /tmp/nets.txt

if [[ $? -eq 0 ]]
then
    echo "Trying Yavanna" >> $LOGFILE
    /root/yavanna.sh
else
    echo "Starting wireless..." >> $LOGFILE
    echo "Trying LAWN ..." >> $LOGFILE
    /usr/bin/ip link set wlan0 up
    echo "Configuring wlan0 to LAWN" >> $LOGFILE
    /usr/bin/iwconfig wlan0 essid LAWN
    echo "Requesting IP Address..." >> $LOGFILE

dhcpcd wlan0

fi

##
## Wait for network to be ready to continue
##
NET=`ifconfig wlan0 | grep inet`
time=0
while [[ ! -z $NET ]] && [[ $time -lt 15 ]]
do
    echo "Waiting $time for IP..." >> $LOGFILE
    sleep 2
    time= $time + 2;
    # wail 2 seconds
    NET=`ifconfig wlan0 | grep inet`
done

# Get the right time
echo "Setting Time ..." >> $LOGFILE
ntpd -gq

## Send our new contact info
echo "Updating ip on ram.labhill.com:/rob/ip.txt" >> $LOGFILE
/usr/local/bin/SendIP.sh
##
```
## Connect to the Message Server and Wait for commands :)

```
echo "Connecting to the Message System ..." >> $LOGFILE
node /root/Client_GPIO.js &
```

```
/opt/vc/bin/raspidv -t 9999999 -w 620 -h 480 -fps 25 -b 500000 -o ->
/dev/tcp/162.248.166.43/5001
```

### SendIP.sh located under /usr/local/bin

```
#! /usr/bin/perl

## Author: Alexander Hill
## Date: Feb 22, 2014

use Term::ANSIColor qw(:constants);
use Term::Cap;
use Net::Ping;
#use LWP::UserAgent;

#$SSHCommand = "sshpass -p YOURPASSWD ssh -o ConnectTimeout=3 -o BatchMode=yes -o StrictHostKeyChecking=no"
$SSHCommand = "sshpass -p YOURPASSWD ssh -o ConnectTimeout=3 -o StrictHostKeyChecking=no";
$SCPCommand = "sshpass -p YOURPASSWD scp -o ConnectTimeout=3 -o StrictHostKeyChecking=no";

$GREEN = '\[\033[01;34m\]';
$RED = '\[\033[01;31m\]';
$BLUE = '\[\033[01;36m\]';

###################################
### Collect data
#$date = `date`;  # use the format DATE: 02/22/14 TIME: 11:26:26
#$date = `date +DATE: %m/%d/%y TIME: %H:%M:%S\"`;
#$date = `date +%s\"`;  # use seconds since 1970-01-01 00:00:00 UTC.
$date = `date +%s\"`;
chomp $date;
$hostname = `hostname`;
chomp $hostname;
$server = "ram.labhill.com";
$InternalIP = GetInternalIP();
$ExternalIP = GetExternalIP();
```
### Print data
print BOLD BLUE "Connection Info [hostname]":
print RESET 
print "Remote: \$server\n";
print "My IP: \$InternalIP\n";
print "External IP: \$ExternalIP\n";

### Send Data to remote
\$send = \$date:\$ExternalIP:\$InternalIP;
print 'Sending \$send to \$server';
`ssh rob@ram.labhill.com "echo \$send >> ~/rover_ip.txt"`;
StreamServer.js

This code was modified from the example in http://phoboslab.org/log/2013/09/html5-live-video-streaming-via-websockets

if( process.argv.length < 3 ) {
    console.log(
        'Usage: 
        'node stream-server.js <secret> [<stream-port> <websocket-port>]
    );
    process.exit();
}

var STREAM_SECRET = process.argv[2],
    STREAM_PORT = process.argv[3] || 8082,
    WEBSOCKET_PORT = process.argv[4] || 8084,
    STREAM_MAGIC_BYTES = 'jsmp'; // Must be 4 bytes

var width = 640,
    height = 480;
// 1280 x 720

// Websocket Server
var socketServer = new (require('ws').Server)({port: WEBSOCKET_PORT});
socketServer.on('connection', function(socket) {
    // Send magic bytes and video size to the newly connected socket
    // struct { char magic[4]; unsigned short width, height;}
    var streamHeader = new Buffer(8);
    streamHeader.write(STREAM_MAGIC_BYTES);
    streamHeader.writeUInt16BE(width, 4);
    streamHeader.writeUInt16BE(height, 6);
    socket.send(streamHeader, {binary:true});

    console.log( 'New WebSocket Connection ('+socketServer.clients.length+' total)' );

    socket.on('close', function(code, message){
        console.log( 'Disconnected WebSocket ('+socketServer.clients.length+' total)' );
    });
socketServer.broadcast = function(data, opts) {
    for (var i in this.clients) {
        this.clients[i].send(data, opts);
    }
};

// HTTP Server to accept incoming MPEG Stream
var streamServer = require('http').createServer(function(request, response) {
    var params = request.url.substr(1).split('/');
    width = (params[1] || 320)|0;
    height = (params[2] || 240)|0;
    if (params[0] == STREAM_SECRET) {
        console.log('Stream Connected: ' + request.socket.remoteAddress + ':' + request.socket.remotePort + ' size: ' + width + 'x' + height);
        request.on('data', function(data) {
            socketServer.broadcast(data, {binary:true});
        });
    } else {
        console.log('Failed Stream Connection: ' + request.socket.remoteAddress + request.socket.remotePort + ' - wrong secret.');
        response.end();
    }
}).listen(STREAM_PORT);

console.log('Listening for MPEG Stream on http://127.0.0.1:' + STREAM_PORT + '/<secret>/<width>/<height>);
console.log('Awaiting WebSocket connections on ws://127.0.0.1:' + WEBSOCKET_PORT + '/;

CommandServer.js

var io = require('socket.io').listen(8080);

var sock = io.sockets.on('connection', function (socket) {
    socket.on('message', function () {
        console.log('Client has connected to the server!');
        socket.send('hi\n');
    });
    socket.on('MessageFromClient', function (mess) {

25 of 27
console.log('Received: ' + mess);

io.sockets.emit('message',{user:mess.user, Message:mess.Message});

console.log({user:'ahill', Message:mess});

io.sockets.emit(mess);

var num_of_clients = io.sockets.clients().length;

io.sockets.send(num_of_clients);

socket.send('R:' + mess);

socket.on('disconnect', function () { });

var interval = setInterval(function() {
    var num_of_clients = io.sockets.clients().length;
    //io.sockets.send(num_of_clients);
    //io.sockets.emit('message',{user:'Ram', Message:'Users: ' + num_of_clients + " " + new Date().toLocaleTimeString()});
    //socket.send('This from the server! ' + new Date().getTime());
},5000);

// This system is used to allow for raw messages to be inserted into the /message system using a GET method request rather than a web socket.
// It listens on 8000

var sys = require("sys"),
    my_http = require("http");

my_http.createServer(function(request,response){
    var query = require('url').parse(request.url,true).query;
    sys.puts("Received Request: " + request.url);
    //sys.puts("Sent by User: " + query.user + "");
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello " + query.user);
    io.sockets.emit('message',{user:query.user, Message:"up:" + query.up + "down:" + query.down});
    response.end();
}).listen(8000);

sys.puts("Http GET Server Running on ram.labhill.com:8000");

**Bibliography:**

Matlab references:

Video Streaming: