Fall | 14

# Weekly Status Report


## Wesley Myers


October 21-27th, 2014


w e s l e y . y . m y e r s @ g m a i l . c o m

## Table of Contents

## Executive Summary

Back on track due to a requirement change on OpenCV.

- Serial Connection Work
  - Arduino Receive/Transmit Working
  - Raspberry Pi Receive/Transmit Working
  - Arduino to Laser Not Working
- Camera Integration and Testing
  - Laser Visible to Long Distance
  - OpenCV Detection of Laser

## Serial Connection Work

Raspberry Pi has a main topic post[1] to talk about the limitations of the Raspberry Pi USB ports. A relevant limitation of the Pi is the power supplied by the USB port. It is a maximum of 500 mA, which means we may need to use the 5 V line supplied on the GPIO to power the Arduino or another power source to draw enough current. This has yet to be determined.

```
pi@rpilaser ~ $ lsusb
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 7392:7811 Edimax Technology Co., Ltd EW-
7811Un 802.11n Wireless Adapter [Realtek RTL8188CUS]
Bus 001 Device 083: ID 2341:0043 Arduino SA Uno R3 (CDC ACM)
Bus 001 Device 005: ID 046d:c52b Logitech, Inc. Unifying Receiver
pi@rpilaser ~ $ dmesg
[ 2250.983950] usb 1-1.3: Manufacturer: Arduino (www.arduino.cc)
[ 2250.983965] usb 1-1.3: SerialNumber: 74134373733351314031
[ 2250.985880] cdc_acm 1-1.3:1.0: ttyACM0: USB ACM device
```

In order to power the Arduino separately to prevent any magic smoke, I used an old 5V FTDI cable and ran the power to the Vin port on the Arduino. I then plugged in the USB cable between the Raspberry Pi and the Arduino.

Using the PySerial library, I was able to establish communication between the Arduino and the Rasbperry Pi. Sounds trivial, but it was a bit of work. The Arduino code (Appendix A) was written to accept a standard set of commands. All we want to do are move servos and say when to read from the laser. On the Raspberry Pi end, I was able to cleanly create functions to send commands to the laser. In the future, these functions will be called when the UI has a button click.

---

[1] http://www.raspberrypi.org/forums/viewtopic.php?f=28&t=53832

### Arduino to Laser Communication

This area has been a bit troublesome. I've set up the code to communicate appropriately, but I don't get a response back. I'm digging further into the issue, but I might need to buy a windows laptop to use their application.

## Camera Integration and Testing

The camera has now been mounted thanks to the longer cable. This cable is an 18-inch cable for the Raspberry Pi camera to CSI bus. With careful taping, I was able to mount it on top. We can verify that the camera is aligned with the laser using the camera feed. Though this is pretty cool, we should investigate further on how to properly calibrate the camera-laser alignment. As for how far away we can see the laser, I was able to clearly see the laser out to 50 feet, which was the maximum length of my house.

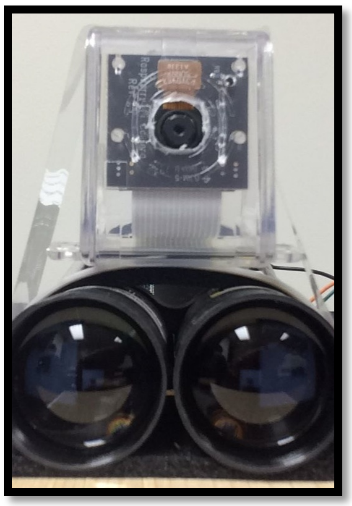

Figure 1 - Camera Mount

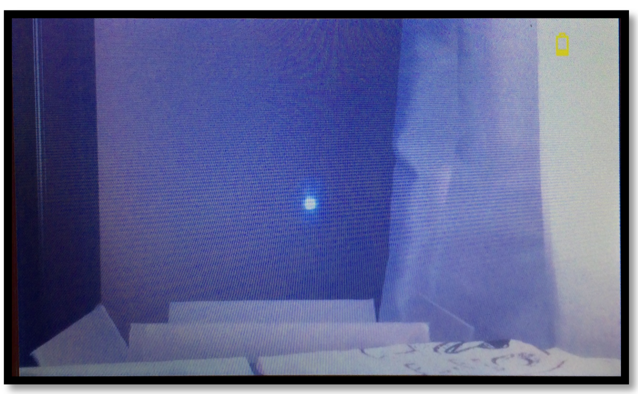

Figure 2 - Laser On Wall

## OpenCV

Given that we can see the laser, we should use this for verifying the laser is pointed in the correct direction on initial set up, as well as for verifying on future "points" that the laser is still properly configured.

In order to save time compiling and loading dependencies on the Raspberry Pi, I thought I'd be clever and went ahead and pulled a copy of OpenCV that was already precompiled[2] for the Raspberry Pi. The build date was in July of 2014, so it seemed recent enough to get the job done.

However, after trying to integrate, this library wasn't built to be used with python. So I had to scrub the installation.

To install OpenCV on the Raspberry Pi for python, it is as simple as running this the following line. It turns out that this only took about 15 minutes to install and we can start using OpenCV immediately.

```
sudo apt-get install libopencv-dev python-opencv
```

In order to actually use OpenCV at this point, we can use the OS library in python to take images with the camera.

```
import os

os.system("raspistill -o image.jpg")

img = cv2.imread("/home/pi/Desktop/image.jpg")
```

I found some documentation[3] online that talks about how to use OpenCV in this method. Given that we have a fresh image, we can now shove it through some OpenCV algorithms to check if we can find the laser.

As is typical with many image-processing techniques, as an end goal for processing, we want as few pixels as possible and ideally have something binary to work with. So the basic strategy was to reduce the image size by looking in the region we care about (the center). Next threshold the image using the basic method of saying the laser is going to be bright. Next we need to detect the circle. There are a couple of

[2] http://www.raspberrypi.org/forums/viewtopic.php?f=33&t=81503
[3] http://trevorappleton.blogspot.com/2013/11/python-getting-started-with-opencv.html

techniques, but the common one used was using a Hough Transform[4], which OpenCV provides[5].

Figure 3 - Whole Image

Image dimensions are 1944x2592. This is a lot of area to cover, so it'll be easier to work with something a bit smaller. Masking over to just work with a 300x300 image will be much faster and easier.

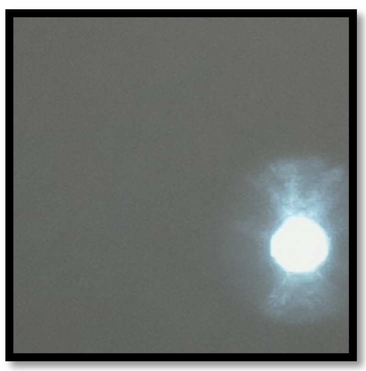

Figure 4 - Cropped Image

[4] http://en.wikipedia.org/wiki/Hough_transform
[5] http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html

Thresholding is a bit trickier. We want to only get the bright spot in the image. This is accomplished by using the thresholding[6] libraries in openCV.

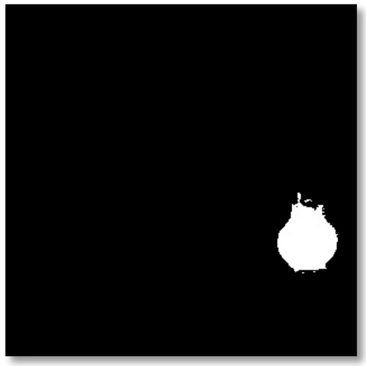

Figure 5 – Thresholded Image

The end result is a great circle labeling the target at hand. We now have a centroid for the laser in relation for the camera.

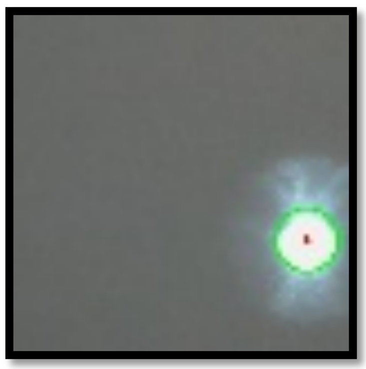

Figure 6 – Circled Result

Given that we have a centroid, we can tell the user/operator how off center the laser is in relation to the field of view.

```
===== Calibration Program Results =====
Blob Information:
x1 =  269.5
y1 =  208.5
[[[ 269.5         208.5          23.37734032]]]
X distance off by:  119.5  pixels
Y distance off by:  58.5  pixels
```

[6]http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

## OpenCV Requirement Change

After talking to New Spin leadership, they decided that they don't think automatic feature detection is practical, nor useful.  They felt that the user should go ahead and be the one who selects the points to operate the system.  The following is from the head researcher at New Spin

Auto-identification of targets by OpenCV may not be a good approach. "Good" features tend to be where there is a rapid change - edges and angles - whereas what we want in general will be nice flat surfaces, as distant from edges as possible. But that's okay, I don't think that we need to worry much about target selection.

# Appendix A – Arduino Serial Code

```
#include <Servo.h>

//#define DEBUG

//tilt
#define TILT_LEVEL 90
#define TILT_MAX_DOWN 75

//pan
#define PAN_MIDDLE 90
#define PAN_MAX_LEFT 50
#define PAN_MAX_RIGHT 130


Servo panServo;  // create servo object to control a servo
Servo tiltServo;

int pan_pos;    // variable to store the servo position
int tilt_pos;
int new_pos;

char command;

void setup()
{
  panServo.attach(5);  // attaches the servo on pin 9 to the servo object
  panServo.write(PAN_MIDDLE);

  tiltServo.attach(6);
  tiltServo.write(TILT_LEVEL);

  pan_pos = panServo.read();
  tilt_pos = tiltServo.read();

  Serial.begin(9600);
}


void loop()
{
  if (Serial.available())
  {
    //get command (i.e. read laser, move pan-tilt)
    command = Serial.read();

    //if there is a number trailing, then get that too
    new_pos = Serial.parseInt();

    if(command == 'p')
    {
      // we want to move the pan servo

      if(new_pos >= PAN_MAX_LEFT && new_pos <= PAN_MAX_RIGHT)
      {
        #ifdef DEBUG
        Serial.print("Moving Pan Servo: ");
        Serial.print(new_pos);
        Serial.print(" -> ");
        Serial.println(pan_pos);
        #endif
```

```
    if (new_pos > pan_pos)
    {
      while (pan_pos < new_pos)
      {
        pan_pos++;

        panServo.write(pan_pos);
        delay(50);

        #ifdef DEBUG
        Serial.println("right!");
        #endif
      }
    }
    else if(new_pos < pan_pos)
    {
      while (new_pos < pan_pos)
      {
        pan_pos--;

        panServo.write(pan_pos);
        delay(50);

        #ifdef DEBUG
        Serial.println("left!");
        #endif
      }
    }

    new_pos = panServo.read();
  }
  else
  {
    #ifdef DEBUG
    Serial.println("===== Bad Pan Servo input =====");
    Serial.print(" - Servo: ");
    Serial.println(command);
    Serial.print(" - Position: ");
    Serial.println(new_pos);
    #endif
  }
}
else if(command == 't')
{
  // We want to move the tilt servo

  if(new_pos >= TILT_MAX_DOWN && new_pos <= TILT_LEVEL)
    {
      if (new_pos > tilt_pos)
      {
        while (tilt_pos < new_pos)
        {
          tilt_pos++;

          tiltServo.write(tilt_pos);
          delay(50);

          #ifdef DEBUG
          Serial.println("up!");
          #endif
        }
      }
      else if(new_pos < tilt_pos)
```

```
          {
            while (new_pos < tilt_pos)
            {
              tilt_pos--;

              tiltServo.write(tilt_pos);
              delay(50);

              #ifdef DEBUG
              Serial.println("down!");
              #endif
            }
          }

          new_pos = tiltServo.read();
        }
        else
        {
          #ifdef DEBUG
          Serial.println("===== Bad Tilt Servo input =====");
          Serial.print(" - Servo: ");
          Serial.println(command);
          Serial.print(" - Position: ");
          Serial.println(new_pos);
          #endif
        }
    }
    else if(command == 'r')
    {
      // We want to read the LRF

      //test response
      Serial.println(123.45);

      #ifdef DEBUG
      Serial.println("===== Read LRF =====");
      #endif
    }
    else
    {
      #ifdef DEBUG
      Serial.println("===== Bad Input =====");
      Serial.print("Command: ");
      Serial.println(command);
      #endif
    }
  }
}
```

## Appendix B – Raspberry Pi Serial Code

```
import serial
import time

DEVICE = '/dev/ttyACM0'
BAUD = 9600
ser = serial.Serial(DEVICE, BAUD)

#delay to get the serial port set up
time.sleep(3)

PAN_SERVO = 'p'
TILT_SERVO = 't'
LRF_READ = 'r'

tilt_servo_pos = 90
pan_servo_pos = 90

def moveServo(servo, position) :
  ser.write(servo + str(position))
  ser.flush()
  time.sleep(1)
  return

def pTest() :
  moveServo(PAN_SERVO, 100)

  moveServo(PAN_SERVO, 80)

  moveServo(PAN_SERVO, 100)

  moveServo(PAN_SERVO, 80)
  return

def laserTest() :
  ser.flushInput()
  ser.write(LRF_READ)
  time.sleep(1)
  result = ser.readline()
  print float(result)
  return float(result)

pTest()

laserTest()
```

## Appendix C – OpenCV Code

```
import os
import cv2
import math
import time

##Resize with resize command
def resizeImage(img):
    dst = cv2.resize(img,None, fx=0.25, fy=0.25, interpolation = cv2.INTER_LINEAR)
    return dst

##Take image with Raspberry Pi camera
img_name = str(int(time.time())) + 'image.jpg'
os.system('raspistill -o images/' + img_name)

##Load image
img = cv2.imread("/home/pi/opencv/images/" + img_name)
grey = cv2.imread("/home/pi/opencv/images/" + img_name, 0) #0 for grayscale

height, width = img.shape[:2]

total_radius = 300

w_y1 = (height/2) - (total_radius/2)
w_y2 = (height/2) + (total_radius/2)
w_x1 = (width/2) - (total_radius/2)
w_x2 = (width/2) + (total_radius/2)

#want to reduce the amount of the image we are working with
cropped = img[w_y1:w_y2, w_x1:w_x2]
cv2.imwrite('laser.jpg', cropped)

##Run Threshold on image to make it black and white
ret, thresh = cv2.threshold(grey[w_y1:w_y2, w_x1:w_x2], 200, 255, cv2.THRESH_BINARY)

cv2.imwrite('thresh_laser.jpg', thresh)

#thresh = cv2.adaptiveThreshold(cropped,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)

##Use houghcircles to determine centre of circle
circles = cv2.HoughCircles(thresh, cv2.cv.CV_HOUGH_GRADIENT, 1, 75, param1=50, param2=13, minRadius=0, maxRadius=175)
for i in circles[0,:]:
    #draw the outer circle
    cv2.circle(cropped,(i[0], i[1]), i[2], (0, 255, 0), 2)
```

```
    #draw the centre of the circle
    cv2.circle(cropped,(i[0], i[1]), 2, (0,0,255), 3)

##Determine coordinates for center of circle
x1 = circles[0][0][0]
y1 = circles[0][0][1]

##print information
print "===== Calibration Program Results ====="

print "Blob Information: "
print "x1 = ", x1
print "y1 = ", y1
print circles

if math.fabs((total_radius/2) - x1) > 0:
  print "X distance off by: ", math.fabs((total_radius/2) - x1), " pixels"

if math.fabs((total_radius/2) - y1) > 0:
  print "Y distance off by: ", math.fabs((total_radius/2) - y1), " pixels"

##Resize image
img = resizeImage(img)
thresh = resizeImage(thresh)
cropped = resizeImage(cropped)

##Show Images
cv2.imwrite("thresh.jpg",thresh)
cv2.imwrite("img.jpg",img)
cv2.imwrite("grey.jpg",grey)
cv2.imwrite("cropped.jpg", cropped)
cv2.waitKey(0)
```