Wesley Myers
18-578
Team F
Individual Lab Report 4

Individual Progress

One of the first things I did was create a Finite State Machine diagram illustrating how we should implement the FSM for this week's demonstration. Since the parts we are assembling are for our actual machine, the FSM is actually simpler than what we needed for last week. For this week's demonstration, we promised that we would have an assembled base, the CMUcam4 tower, and the part of the system that turns to shoot the ball in real time. So the FSM, illustrated in Figure 1, does the basic calibration and then points the shooter in real time at the targets. The FSM also calls for a button interrupt to start and halt the system. I thought this would be a good idea since this is how most machines function, press to go and press to stop for an emergency or just off.

Since last week all we demonstrated was being able to see a green target, this week I wanted to show that we can indeed see all three targets accurately. What this meant was that I needed a way to be able to quickly switch between colors during our demonstration and for testing. For this, I created a board with switches on it to signify which color I was tracking. LEDs on the board allow a visual confirmation of which switch is on. This is illustrated in Figure 3. SooHyun assisted in the soldering of the board. This board was also good for demonstration of the actual system capability.

I assembled a new test bench to be able to test and demonstrate the capability of tracking colors. This involved an Arduino UNO, LED array, RGB Switch, CMUcam4, and a servo. The LED array was for being able to see what was happening in the code since printing is not available. The reasoning for the RGB switch is described in the previous paragraph. The servo served as a way to demonstrate to the team members that we can point something at the targets. Figure 5 illustrates the set up. Figure 4 is a close up of what the servo pointing top view looked like.

Color tracking was a bit of a challenge. The coding was rather simple for this new system. The hard part was actually getting the correct colors for each of the targets. Green and Blue are relatively similar colors, which means that the colors overlap quite a bit. Thus being able to identify which color is which took a bit of a trial and error process. One of the new things I did was take an image that doesn't have as many pixels. The reason for this is that it gives more of an average value for the pixels. When I have full resolution, I'm looking at individual pixels, which RGB values vary quite a bit. I'd rather look at values that are more of an average, which this gives. Figure 2 shows the image I used.

There was a large challenge in figuring out why I couldn't poll the data more than once. I describe this challenge further in the Challenges section. I also received an update for the Arduino Library code. With this code, I implemented some filtering to eliminate noise from the system. This helped grab a better image of the target.

For the System Demonstration, Rick was able to fabricate and machine acrylic parts. We also received the stepper motor and driver for the panning part of the system. Rick assembled the system and I assisted with getting the stepper motor working. Once it was working, I integrated the camera code, the finite state machine, and the new stepper motor/driver. The final system that we demonstrated on February 29 is show in Figure 6.

Challenges

As mentioned in last week's ILR, I received Arduino Library code to communicate with the CMUcam4. Since it is still under development, I am basically acting as a tester for this library. As with any library under development, there will be bugs in the code. What this meant for me is that I encountered quite a few bugs.

I'm doing my testing with an Arduino UNO. The UNO has only one UART, which means that if I print for output, then it'll write to the Arduino terminal and to the CMUcam4. This is bad since I'm basically sending garbage to the CMUcam4 when I really just want to see the internal state. I made a rule for myself to not do this last week, but I broke it. There was a weird error code I got for a while that was unclear. I discovered CMUcam4 was saying that I was passing it garbage. So I switched back to my LED array that I made last week to signal me the internal state.

The most significant challenge was a bug that involved a state system overlaid in the library. What the CMUcam4 state system does is keep track of what "mode" the camera is in. The actual CMUcam4 doesn't do this, so it doesn't make sense that the state system is there. If you connected the CMUcam4 with a Parallax Serial Terminal, then you get a constant stream of data coming from it about the color you're tracking. What this means is that you should be able to poll that stream whenever you want. However, the code sent the system into a weird state immediately after the first poll, meaning a second poll couldn't happen. This was hard to discover since the code does not explicitly change to a new state after polling. Making the CMUcam4 go into an idle state solved this problem. A rather simple fix to a unclear problem.

Teamwork

We again split up the work this week. Richard was tasked with machining all the parts for the system demonstration. All of the parts necessary for this have come in, so we decided that it would be a good idea to assemble and manufacture as much as possible. David and SooHyun were responsible for getting the coding

aspect of the system done. I was tasked with getting the CMUcam4 capable of tracking three different colors in real time and integrating it with the finite state machine.

As for next week, we will be splitting up the work for the mid-semester demonstration. This is described further in the next section.

Future Work

Everything we've done this week is applicable to future work. The structure we've made so far is going to be built upon for our mid-semester. One thing we learned from this is that the specification for the nerf balls we received was incorrect. It was off by one quarter of an inch, thus we need to re-machine the parts. It is good to learn of this now rather than later, and it also means that we should also always check all measurements in the future first before machining.

The camera is now at a state where it can determine x and y position for the individual targets. This will be built upon for our final system and for our mid-semester demo. The x position will be used to point at the target, and the y will determine the speed the motors will turn at. This will be finer tuned to be able to grab images/positions at a faster rate. The big thing is being able to get the colors correct. This will be an ongoing problem.

Next week is the mid-semester presentation. We will be splitting up the work for the slides. I'll be taking the slides about physical and functional architecture. Rick will write the slides about design concepts and depiction of the system. SooHyun is our treasurer and will discuss the parts list and budget. David will talk about risk management.

In order to continue progress with the CMUcam4, I will be trying to get the system to look for all three targets and put the coordinates in an array. The data will say if the target is also moving. Rick is going to continue machining parts and assembly.

Figures

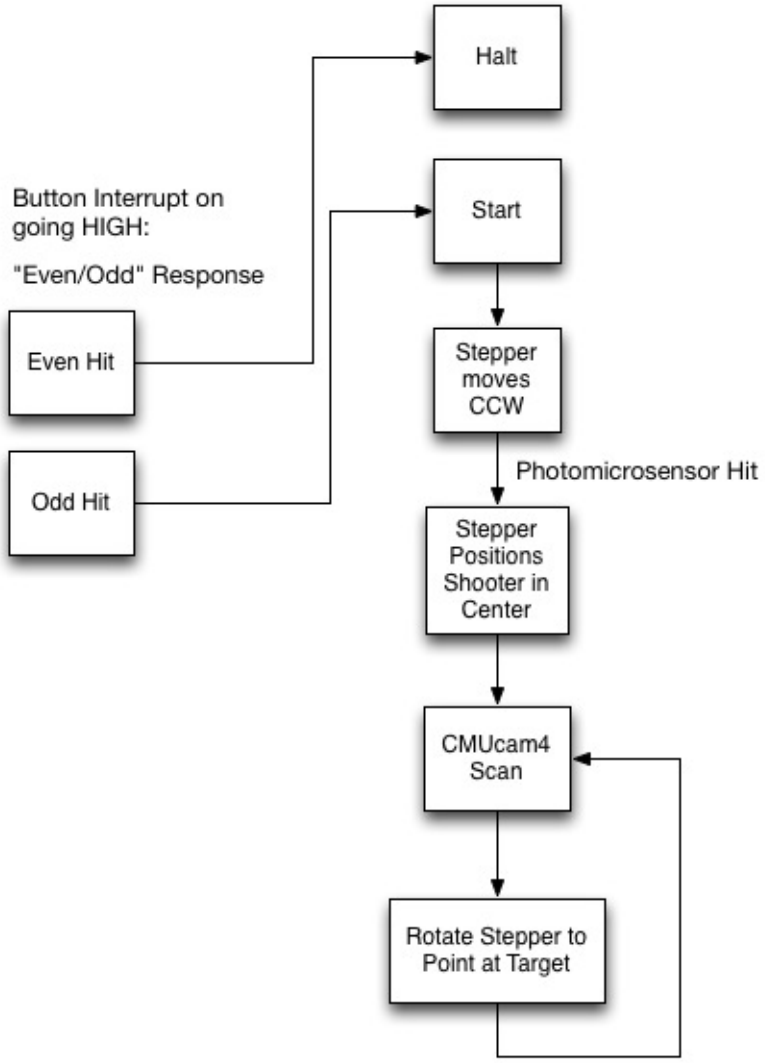

Figure 1. State Machine for System Demonstration One

Figure 2. Scaled Image from CMUcam4

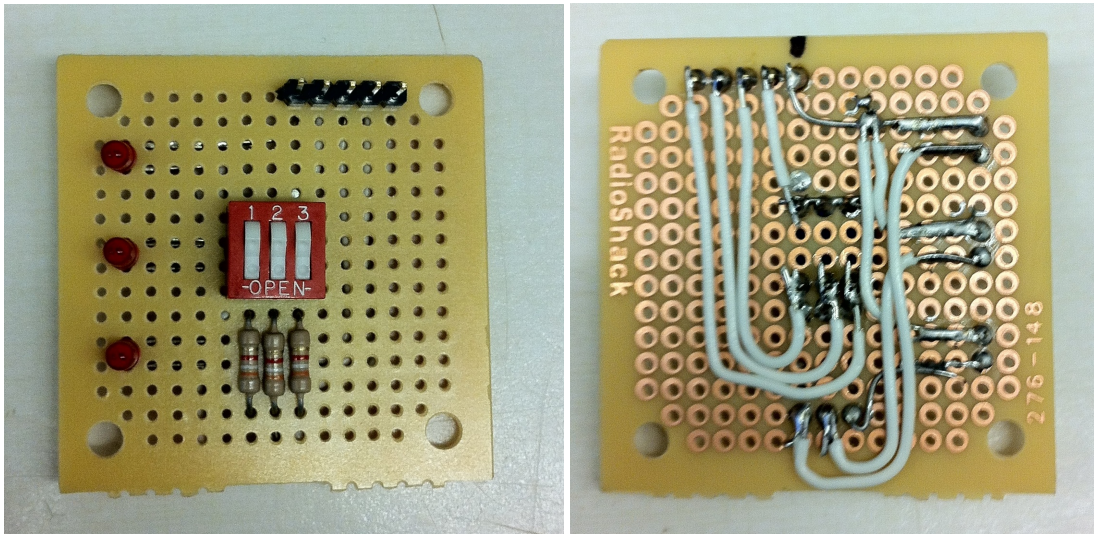

Figure 3. RGB Switches for Fast Color Tracking Switching (Top/Bottom)

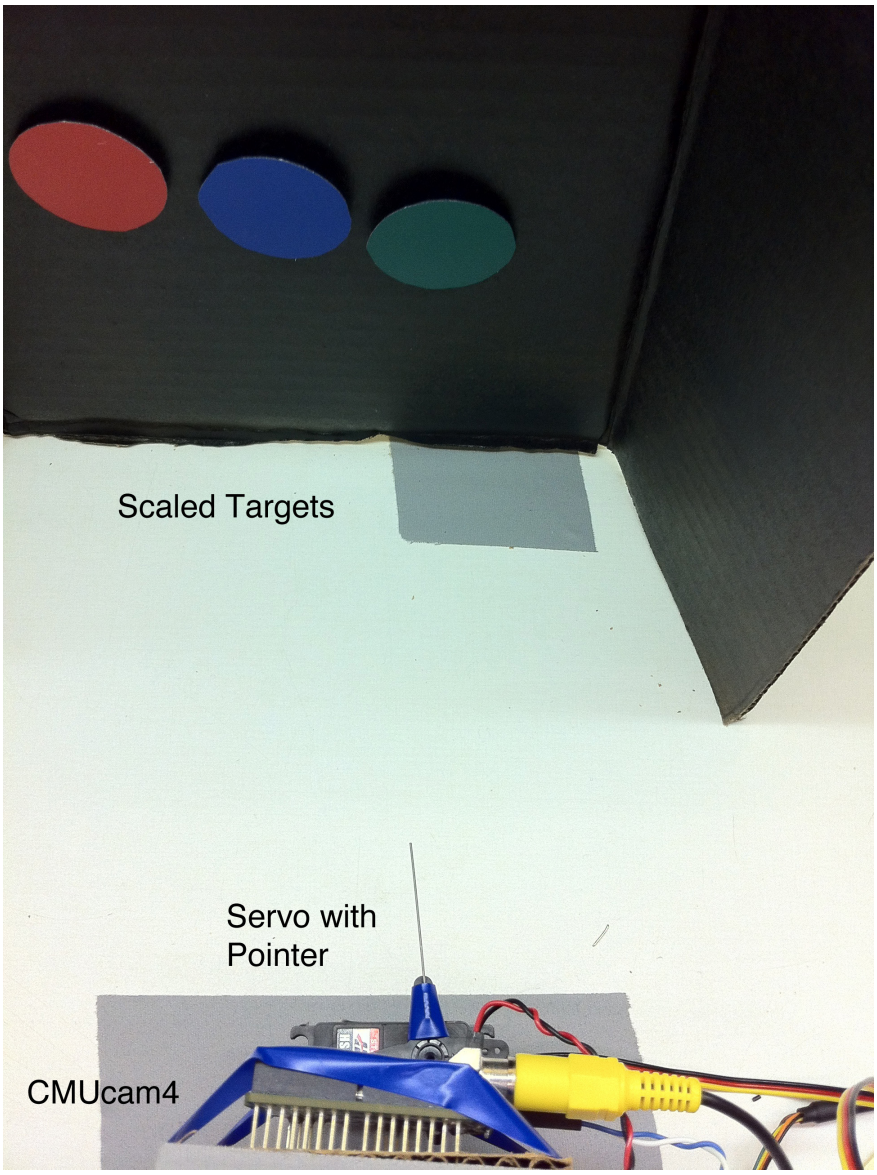

Figure 4. Pointing Demonstration

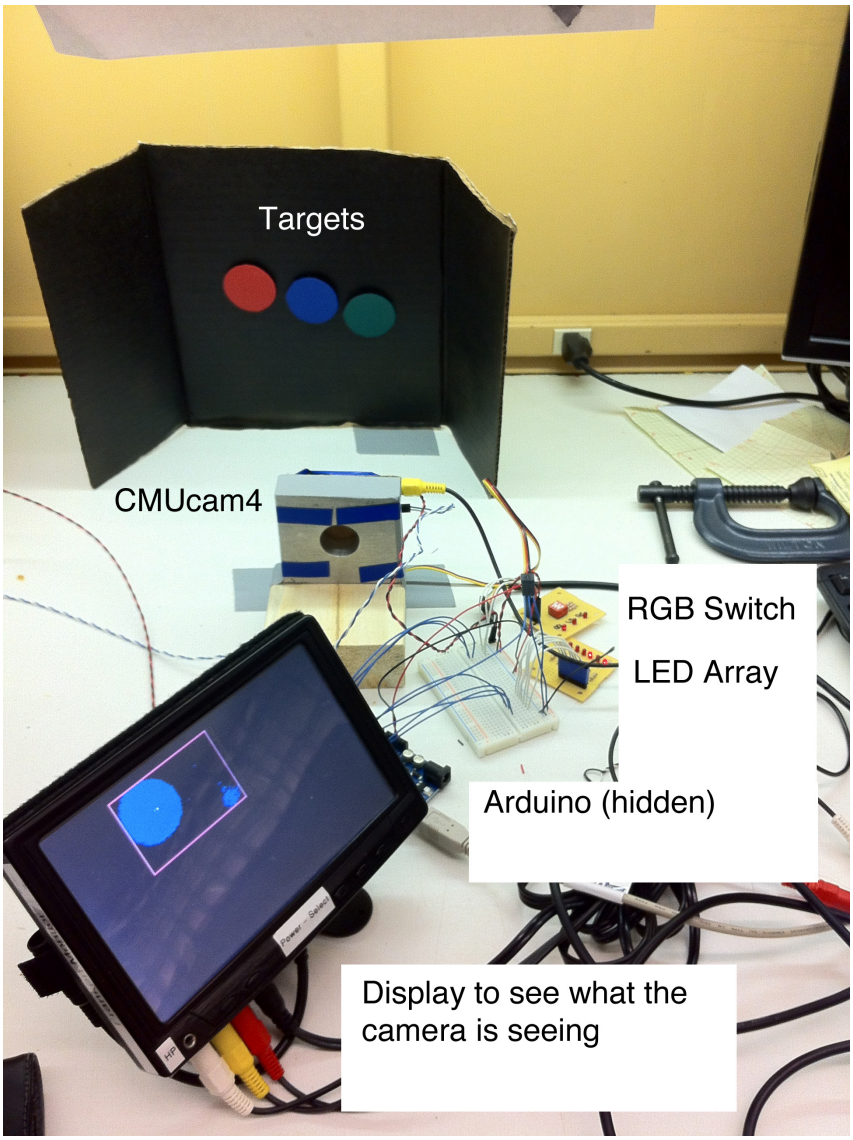

Figure 5. Setup for Testing/Demonstration

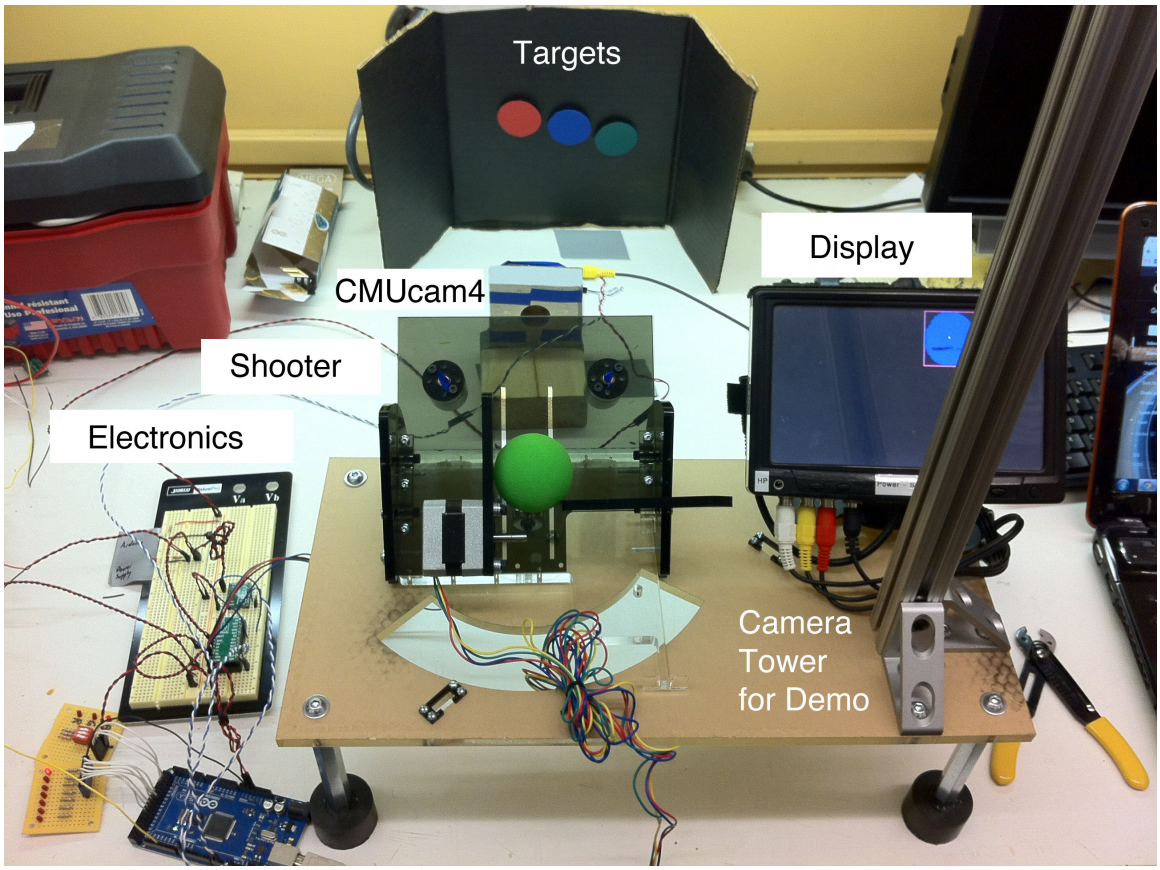

Figure 6. System Demonstration One Setup

Code

```
#include "CMUCam4.h"
#include "Servo.h"

/*
 *  Wesley Myers
 *
 *  This version of the code tracks red, green, or blue based upon external switches.
 */

#define led1 5
#define led2 6
#define led3 7
#define blob_left_led   8
#define blob_middle_led 9
#define blob_right_led   10
#define error_led       11

#define redSwitch   2
#define greenSwitch 3
#define blueSwitch  4

#ifdef DEBUG
#define DEBUG_PRINT(...) Serial.println( __VA_ARGS__ );
#else
#define DEBUG_PRINT(...) do {} while(false);
#endif

CMUCam4 camera;

unsigned char blob_x;
unsigned char blob_y;
unsigned char prev_blob_x = 0;

int error;

Servo s;

CMUCam4::CMUCAM4_color_range_t redTracking;
CMUCam4::CMUCAM4_color_range_t greenTracking;
CMUCam4::CMUCAM4_color_range_t blueTracking;
CMUCam4::CMUCAM4_color_range_t nullTracking;

CMUCam4::CMUCAM4_track_data_t trackingData;

enum COLORS {red, blue, green, none};
COLORS colorTrackingMode;

boolean collectData;

void setup()
{
  s.attach(6);

  camera.init();
  camera.setNoiseFilterMode(4);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(blob_left_led, OUTPUT);
  pinMode(blob_middle_led, OUTPUT);
  pinMode(blob_right_led, OUTPUT);
  pinMode(error_led, OUTPUT);

  pinMode(redSwitch, INPUT);
  pinMode(greenSwitch, INPUT);
  pinMode(blueSwitch, INPUT);

  collectData = false;
  colorTrackingMode = none;
```

```
  redTracking.redLow = 170;  //170
  redTracking.redHigh = 255;
  redTracking.greenLow = 100;
  redTracking.greenHigh = 160;
  redTracking.blueLow = 110;
  redTracking.blueHigh = 170;

  greenTracking.redLow = 50;
  greenTracking.redHigh = 110;
  greenTracking.greenLow = 0;  //140
  greenTracking.greenHigh = 230;  //180
  greenTracking.blueLow = 140;  //170
  greenTracking.blueHigh = 255;  //255

  blueTracking.redLow = 40;
  blueTracking.redHigh = 120;
  blueTracking.greenLow = 190;
  blueTracking.greenHigh = 255;
  blueTracking.blueLow = 0;
  blueTracking.blueHigh = 255;

  nullTracking.redLow = 0;
  nullTracking.redHigh = 0;
  nullTracking.greenLow = 0;
  nullTracking.greenHigh = 0;
  nullTracking.blueLow = 0;
  nullTracking.blueHigh = 0;

  //initialize struct
  trackingData.centroidX = 0;
  trackingData.centroidY = 0;
  trackingData.boundX1 = 0;
  trackingData.boundY1 = 0;
  trackingData.boundX2 = 0;
  trackingData.boundY2 = 0;
  trackingData.trackedPixels = 0;
  trackingData.trackingConfidence = 0;

}

void loop()
{
  camera.idle();

  //switches dictate which color we are tracking
  if(digitalRead(redSwitch) == HIGH &&
     digitalRead(greenSwitch) == LOW &&
     digitalRead(blueSwitch) == LOW)
  {
    colorTrackingMode = red;
    camera.startTrackColor(&redTracking);
    collectData = true;
  }
  else if(digitalRead(redSwitch) == LOW &&
     digitalRead(greenSwitch) == HIGH &&
     digitalRead(blueSwitch) == LOW)
  {
    colorTrackingMode = green;
    camera.startTrackColor(&greenTracking);
    collectData = true;
  }
  else if(digitalRead(redSwitch) == LOW &&
     digitalRead(greenSwitch) == LOW &&
     digitalRead(blueSwitch) == HIGH)
  {
    colorTrackingMode = blue;
    camera.startTrackColor(&blueTracking);
    collectData = true;
  }
  else
  {
    //do nothing
    colorTrackingMode = none;
```

```
    collectData = false;
  }

  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);
  digitalWrite(error_led, LOW);

  if(collectData == true)
  {
    collectData = false;

    error = camera.getTrackData(&trackingData);

    if(error < 0)
      digitalWrite(error_led, HIGH);
    else
      digitalWrite(error_led, LOW);

    blob_x = trackingData.centroidX;
    blob_y = trackingData.centroidY;

    if(abs(blob_x - prev_blob_x) <= 3)
      digitalWrite(10, HIGH);
    else if(blob_x > prev_blob_x)
      digitalWrite(8, HIGH);
    else if(blob_x < prev_blob_x)
      digitalWrite(9, HIGH);

    DEBUG_PRINT("Error: ");
    DEBUG_PRINT(error);

    DEBUG_PRINT("Centroid X: ");
    DEBUG_PRINT((int)blob_x);

    DEBUG_PRINT("Centroid Y: ");
    DEBUG_PRINT((int)blob_y);

    prev_blob_x = blob_x;

    delay(250);
    aimPointer((int)blob_x);
  }
}

/*
 Function points a pointer at the target based upon the position of the target
 */
void aimPointer(int targetPos)
{
  int i;

  for(i = 2; i <= 120; i += 2)
  {
    if(targetPos < i)
    {
      s.write(60 + (i*2)/5);
      return;
    }
  }
}
```