Wesley Myers
18-578
Team F
Individual Lab Report #5

Individual Progress

So the past two weeks were spring break and preparing for the mid-semester presentation. Progress on the project is now basically determined on getting the CMUcam4 ready and the mechanical parts manufactured.

For the mid-semester presentation, I was responsible for the Physical and Functional Architecture sections. I created a new Physical Architecture diagram given our current sensors/motors. This is shown in Figure 2. I also created a new Functional Architecture diagram, giving the high level aspect of our system. Note that many of the sub-blocks are Finite State Machines themselves. This is shown in Figure 1.

Curtis Layton made new targets in the lab with a non-reflective color on the outside ring. This should give us a more consistent color to detect. I was unable to take new images with the CMUcam4 before I left campus for break.

I did bring the CMUcam4 with me so that I can do testing over break. I developed new code that basically cycles through the different color models and grabbed information to determine the location of the target in relation to the x-coordinate. The code also figures out if a target is moving. The code is in the code section of the ILR.

I received yet another version of the CMUcam4. This is the version that is being distributed to the other teams. I met with the engineer developing the CMUcam4 and we tested the first unit. After the initial test, we realized that the camera was malfunctioning. I assisted him in testing all 15 units he had with them. We discovered that it was an issue with the timing of the system. The propeller chip on the CMUcam4 is being slightly overclocked. Given the specification document, this is okay. However, there is some latency so the only way to solve this is by turning down the clock in the firmware.

Given these issues, I decided to push on with the CMUcam4 that I had. I continued testing and refinement of the image quality.

Most of my testing this week has been devoted to working with the new models. I took some sample images of the new targets as shown in Figure 3. I also determined the necessary bounding box to eliminate parts of the image that are unnecessary. This is illustrated in figure 4. I take a bounding box of the top, middle, and bottom section. This lets me focus entirely on the color and I don't have to worry about other locations. I show this with the colored rectangle. I also took

sample images to illustrate the issue with the reflective tape. This can be shown in figure 5. Clearly the light causes a glare that makes the targets white. Testing showed that glare is a significant factor. Figures 6-8 illustrate the targets I got with the bounding box technique. I found that it was able to track the targets almost instantaneously as it moved too.

On the System Demonstration day, we ran into a problem. We powered up our system and nothing worked as expected. We then noticed that the stepper larger stepper driver was extremely hot and that an LED on the MEGA was not turning on. We then noticed that if we plugged the cord into my computer, the computer would shut off instantly. This caused me to look at the wiring and I noticed that the 12-volt line to the stepper driver was not attached. What this meant was that the driver was pulling a lot of amps through the arduino and basically fried the two stepper drivers and the MEGA. This is rather unfortunate. Richard and I were able to get a new stepper driver and use it to power our weaker stepper for the panning part of the demonstration. We also borrowed a MEGA for the demonstration.

I assisted with the overall state machine programming for the lab, along with modifications needed for the demonstration day miss-hap.

Challenges

It was definitely a challenge trying to get the project back up and running after break. However, machining and coding went well.

The main challenge we faced was the System Demonstration failure with the electronics. This is described in some detail in the second to last paragraph in the individual progress section. A summary of the failure is described below. It was hard due to the fact that Richard and I were the only two people there until SooHyun and David showed up at around 2:30 pm. The failure happened at roughly 10:30 am.

The failure is described as follows: The power line had slipped out due to a new breadboard we were using that is much smaller. It is perfect except for the fact that the header pins don't fit snugly into the board, meaning that they are loose and it is easy to slip out. Thus this could have happened at any point. Thus when it was powered, a surge of current went through the arduino to power the stepper which exceeded the current limit of the arduino.

Teamwork

For the mid-semester presentation, we split up the work. Rick took the slides that required mechanical aspects, such as the concept. I took the slides dealing with the architecture of the system, as mentioned in the Individual Progress section. SooHyun and David split up the rest of the slides.

As for the work over break, I decided to make progress on the CMUcam4. Rick tried to do as much machining as possible before he left. He is returning to campus early, so he'll finish machining hopefully before most of us return. SooHyun and David weren't tasked with anything since they didn't have a physical system to test with.

After break, Rick continued machining of more parts. I continued work with the CMUcam4. SooHyun and David were tasked with programming the system after Rick was done machining parts.

Future Work

Everything we've done so far is contributing to the overall system. The mechanical part is making progress as more parts are done. The CMUcam4 is steadily making progress.

System demo 2 showed us the importance of checking everything before powering on. This mistake also showed us that we need to change breadboards or go to perfboard.

With System Demo 3, we are continuing along the same path. We are incorporating the rack and pinion part into the system. We will also have the CMUcam4 track the targets rather than having canned shooting positions.

Figures

Please note that any images from the CMUcam4 are incorrectly rotated. This is due to the way it takes a bmp image. I do not change it for the sake of not altering the data.

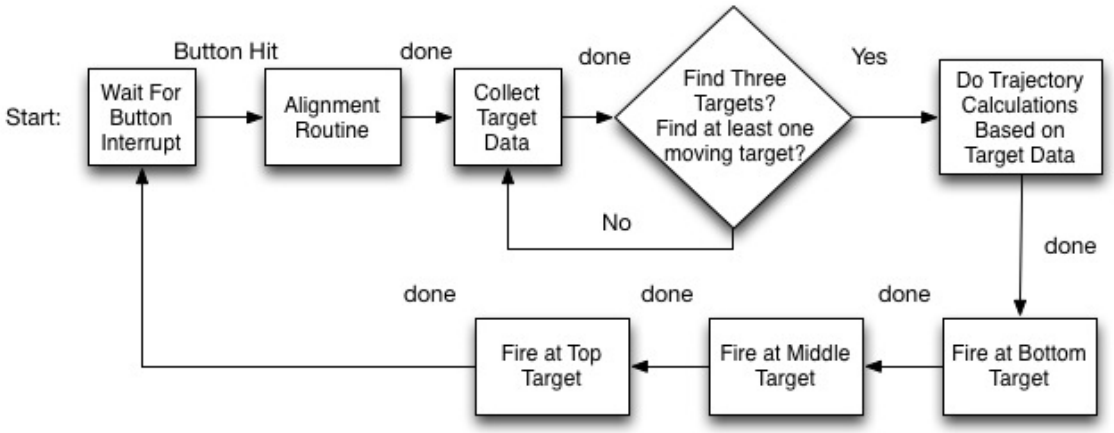

Figure 1. New Functional Architecture

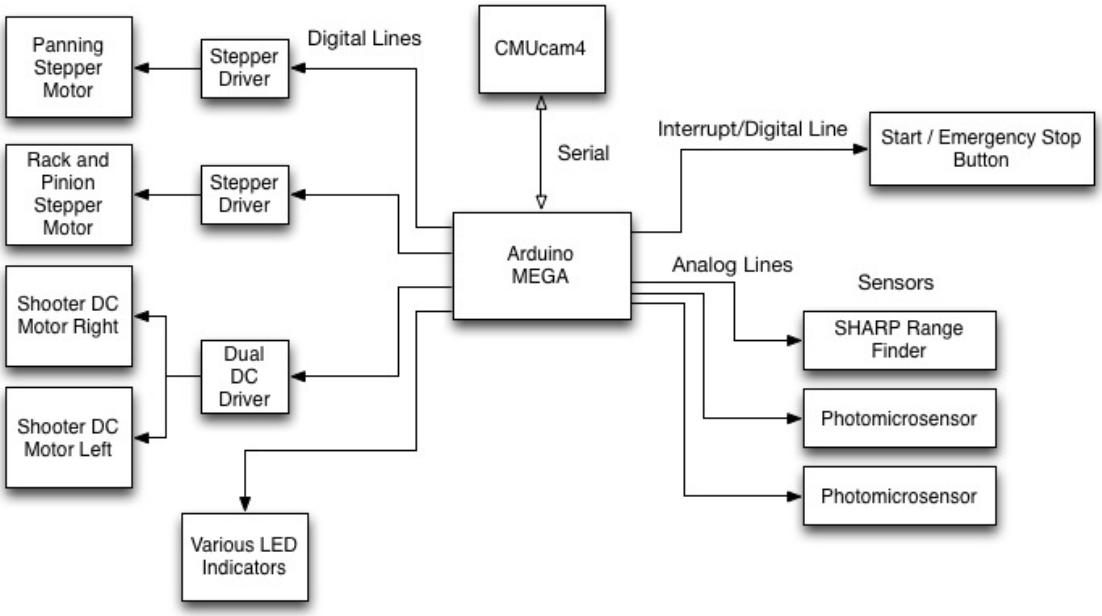

Figure 2. New Physical Architecture

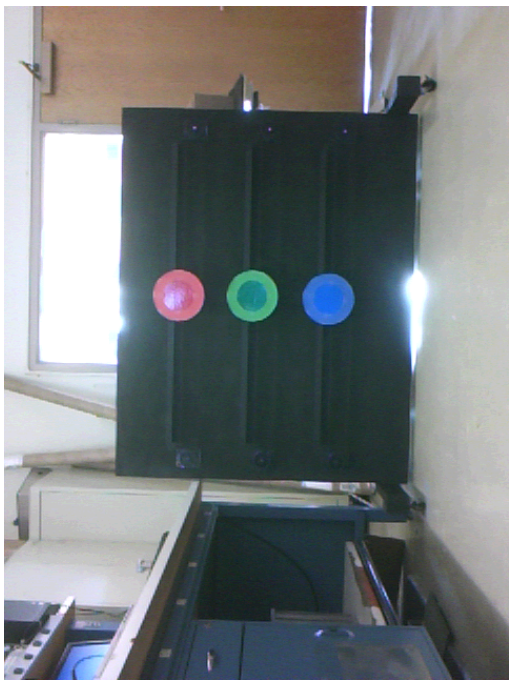

Figure 3. First Image of New Targets

Figure 4. Bounding Boxes on Targets

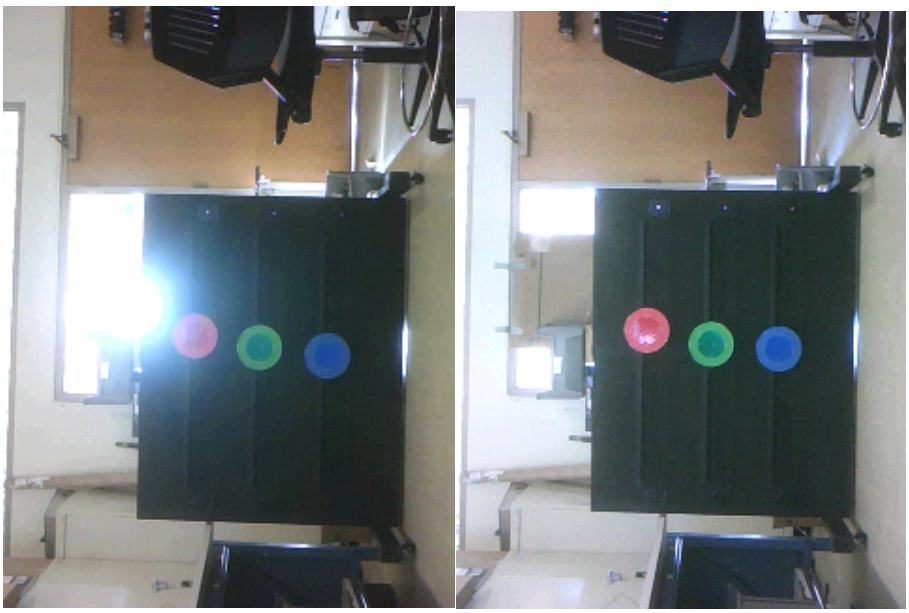

Figure 5. Window Glare Example

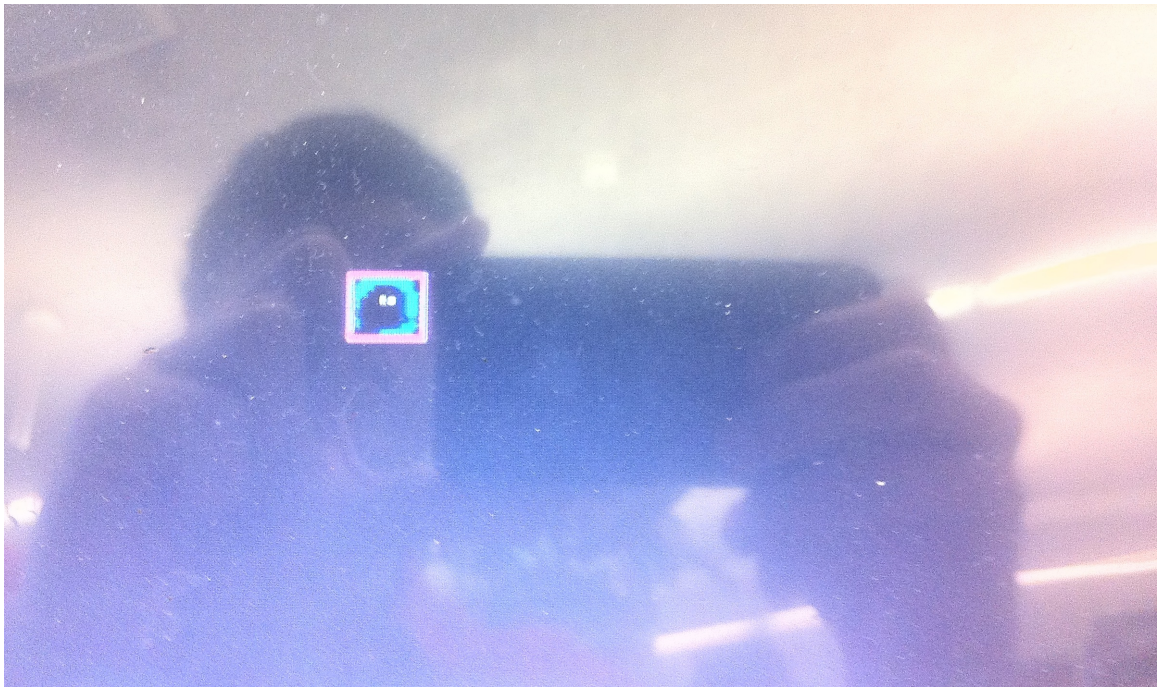

Figure 6. Red Target

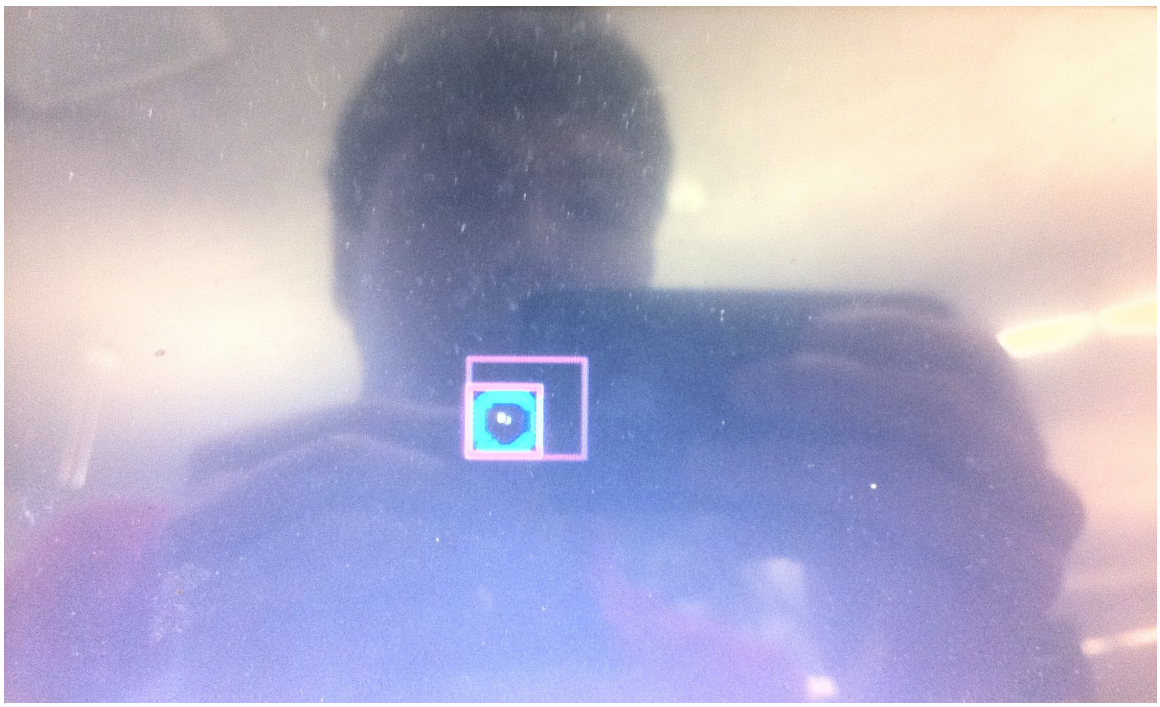

Figure 7. Green Target

Figure 8. Blue Target

Code

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

/*
 *  Wesley Myers
 *
 *  This version of the code grabs the location of all three targets and if
 *  any of the targets are moving.
 */

//Tracking States

#define TRACK_DYNAMIC false

#define done_led        5
#define error_led       11

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

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

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

CMUCam4 camera;

unsigned char blob_x;
unsigned char blob_y;

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_window_t trackWindowSize;
CMUCam4::CMUCAM4_track_data_t trackingData;

struct target {
  unsigned int xPosition;
  unsigned int yPosition;

  unsigned int xPos2;

  boolean stationary;

  unsigned int verticalPosition;  //0 bottom, 1 middle, 2 highest
};

#define RED 0
#define GREEN 1
#define BLUE 2

target targetArray[3];  //0 -> red, 1 -> green, 2 -> blue

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

int trackState;

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

  camera.init();

  camera.setNoiseFilterMode(2);  //was 4
  camera.setTrackWindow(&trackWindowSize);
```

```
  pinMode(done_led, OUTPUT);
  pinMode(error_led, OUTPUT);

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

  trackState = 0;

  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;

  //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;

  //as measured from the top left corner
  trackWindowSize.boundX1 = (unsigned char) 25;
  trackWindowSize.boundY1 = (unsigned char) 35;
  trackWindowSize.boundX2 = (unsigned char) 110;
  trackWindowSize.boundY2 = (unsigned char) 95;

  targetArray[RED].xPosition = 0;
  targetArray[RED].yPosition = 0;
  targetArray[RED].xPos2 = 0;
  targetArray[RED].stationary = true;

  targetArray[GREEN].xPosition = 0;
  targetArray[GREEN].yPosition = 0;
  targetArray[GREEN].xPos2 = 0;
  targetArray[GREEN].stationary = true;

  targetArray[BLUE].xPosition = 0;
  targetArray[BLUE].yPosition = 0;
  targetArray[BLUE].xPos2 = 0;
  targetArray[BLUE].stationary = true;

}

void loop()
{

  camera.idle();
  digitalWrite(error_led, LOW);

  //switches dictate which color we are tracking
  switch(trackState)
  {
    case 0:
    {
      camera.startTrackColor(&redTracking);
      break;
    }
    case 1:
    {
      camera.startTrackColor(&greenTracking);
      break;
    }
    case 2:
```

```
{
  camera.startTrackColor(&blueTracking);
  break;
}
case 3:
{
  camera.startTrackColor(&redTracking);
  break;
}
case 4:
{
  camera.startTrackColor(&greenTracking);
  break;
}
case 5:
{
  camera.startTrackColor(&blueTracking);
  break;
}
case 6:
{
  //data processing phase
  int diffRed = abs(targetArray[RED].xPosition - targetArray[RED].xPos2);
  int diffGreen = abs(targetArray[GREEN].xPosition - targetArray[GREEN].xPos2);
  int diffBlue = abs(targetArray[BLUE].xPosition - targetArray[BLUE].xPos2);

  if(diffRed > 3)
  {
    targetArray[RED].stationary = false;
  }

  if(diffGreen > 3)
  {
    targetArray[GREEN].stationary = false;
  }

  if(diffBlue > 3)
  {
    targetArray[BLUE].stationary = false;
  }


  if (TRACK_DYNAMIC == true &&
     ((targetArray[RED].stationary == false && targetArray[GREEN].stationary == false) ||
     (targetArray[GREEN].stationary == false && targetArray[BLUE].stationary == false) ||
     (targetArray[RED].stationary == false && targetArray[BLUE].stationary == false))
     )
  {
    trackState = 0;
    break;
  }

  if(targetArray[RED].yPosition < targetArray[GREEN].yPosition < targetArray[BLUE].yPosition)
  {
    targetArray[RED].verticalPosition = 0;
    targetArray[GREEN].verticalPosition = 1;
    targetArray[BLUE].verticalPosition = 2;
  }
  else if(targetArray[GREEN].yPosition < targetArray[RED].yPosition < targetArray[BLUE].yPosition)
  {
    targetArray[RED].verticalPosition = 1;
    targetArray[GREEN].verticalPosition = 0;
    targetArray[BLUE].verticalPosition = 2;
  }
  else if(targetArray[RED].yPosition < targetArray[BLUE].yPosition < targetArray[GREEN].yPosition)
  {
    targetArray[RED].verticalPosition = 0;
    targetArray[GREEN].verticalPosition = 2;
    targetArray[BLUE].verticalPosition = 1;
  }
  else if(targetArray[BLUE].yPosition < targetArray[RED].yPosition < targetArray[GREEN].yPosition)
  {
    targetArray[RED].verticalPosition = 1;
    targetArray[GREEN].verticalPosition = 2;
    targetArray[BLUE].verticalPosition = 1;
  }
  else if(targetArray[BLUE].yPosition < targetArray[GREEN].yPosition < targetArray[RED].yPosition)
  {
    targetArray[RED].verticalPosition = 2;
    targetArray[GREEN].verticalPosition = 1;
    targetArray[BLUE].verticalPosition = 0;
  }
  else if(targetArray[GREEN].yPosition < targetArray[BLUE].yPosition < targetArray[RED].yPosition)
```

```
    {
      targetArray[RED].verticalPosition = 2;
      targetArray[GREEN].verticalPosition = 0;
      targetArray[BLUE].verticalPosition = 1;
    }
  }
  case 7:
  {
    //print out data

    Serial.print("Red X Position: ");
    Serial.println(targetArray[RED].xPosition);

    Serial.print("Red Moving: ");
    Serial.println(targetArray[RED].stationary);

    Serial.print("Red Vertical Position: ");
    Serial.println(targetArray[RED].verticalPosition);

    Serial.print("Green X Position: ");
    Serial.println(targetArray[GREEN].xPosition);

    Serial.print("Green Moving: ");
    Serial.println(targetArray[GREEN].stationary);

    Serial.print("Green Vertical Position: ");
    Serial.println(targetArray[GREEN].verticalPosition);

    Serial.print("Blue X Position: ");
    Serial.println(targetArray[BLUE].xPosition);

    Serial.print("Blue Moving: ");
    Serial.println(targetArray[BLUE].stationary);

    Serial.print("Blue Vertical Position: ");
    Serial.println(targetArray[BLUE].verticalPosition);

    trackState++;
  }
  case 8:
  {
    digitalWrite(done_led, HIGH);
  }
}

digitalWrite(error_led, LOW);

if(trackState < 6)
{
  error = camera.getTrackData(&trackingData);

  if(error < 0)
  {
    digitalWrite(error_led, HIGH);
    DEBUG_PRINT("Error: ");
    DEBUG_PRINT(error);
    delay(250);
  }

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

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

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

  switch(trackState)
  {
    case 0:
    {
      targetArray[RED].xPosition = blob_x;
      targetArray[RED].yPosition = blob_y;
      break;
    }
    case 1:
    {
      targetArray[GREEN].xPosition = blob_x;
      targetArray[GREEN].yPosition = blob_y;
      break;
    }
    case 2:
```

```
      {
        targetArray[BLUE].xPosition = blob_x;
        targetArray[BLUE].yPosition = blob_y;
        break;
      }
      case 3:
      {
        targetArray[RED].xPos2 = blob_x;  //no need to check second y since y is generally the same
        break;
      }
      case 4:
      {
        targetArray[GREEN].xPos2 = blob_x;
        break;
      }
      case 5:
      {
        targetArray[BLUE].xPos2 = blob_x;
        break;
      }
    }

    if(error == 0)
    {
      trackState++;
    }
    delay(250);
  }
}
```