Senior Design
ENG EC 463

To: Professor Pisano, Professor Alshaykh, Professor Hirsch

From: Christopher Liao, Anton Paquin, Jeffrey Lin, Eduardo Portet, Aviva Englander

Team: 15 - Laser Tracking

Date: 04/01/18

Subject: Functional Test Report

---

1. Project Objective

1.1. The overall objective of our project is to create an optical based tracking and communication system from a ground station to a UAV, allowing users to interact with their vehicles without using Radio Frequency communication.

2. Test Objective and Significance

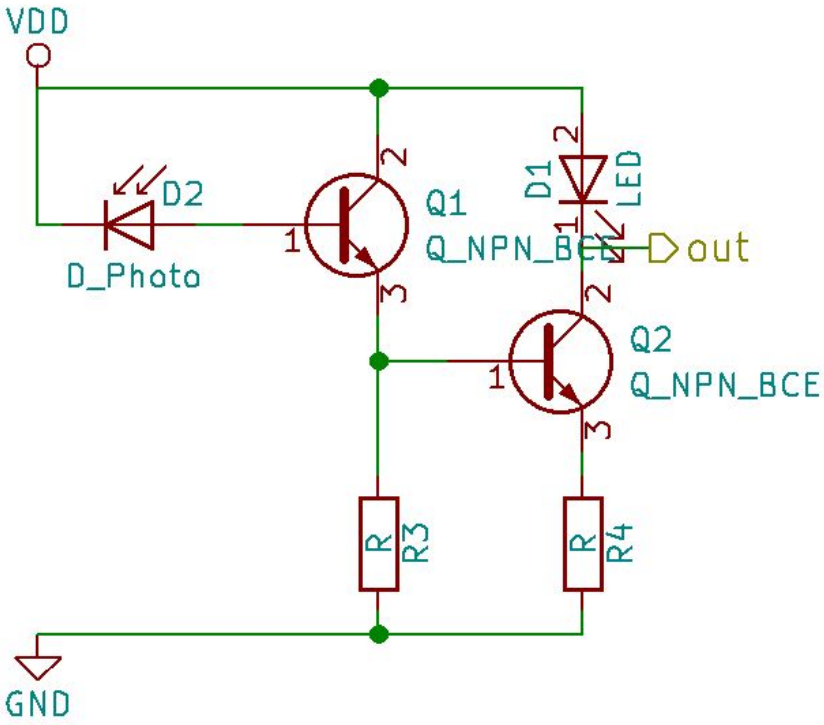
2.1. This test was significant because it demonstrated our ability to track the sensor, which is a fundamental component of our project. Our core deliverable is a tracking algorithm, which we demonstrated. This test focused on achieving a fine tracking mechanism. With this test we demonstrated an ability to track an object with a laser once we had acquired it. We utilized our printed circuit photodiode array and radio frequency transmitter to do so. We successfully demonstrated a 3-dimensional tracking system which could follow the array at the expected speeds.

2.2. We have also demonstrated a rough coarse tracking algorithm that uses the Camera mounted on a Raspberry Pi. This tracking system will be integrated with the fine tracking before or during installation to assist in the initial localization of the photodiode array.

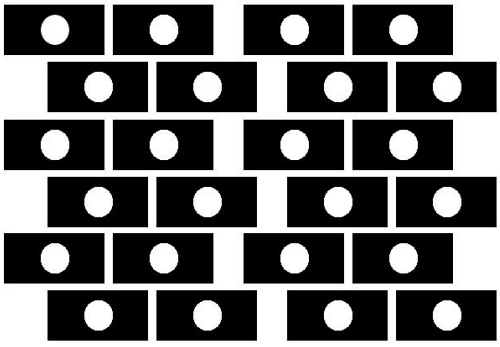
3. Equipment and Setup

3.1. PCB with Updated Photodiode Array

This is the main hardware component of of our project. The same circuit is used our current version of the PCB as the previous one.



This time, the photodiode circuit is tiled 24 times in our sensor array, arranged in the following pattern.



This design was created to minimize the likelihood of our laser passing in between the gaps of the photodiodes. Each output is treated as a single analog value, which is pulled low when the photodiode detects the presence of a laser.

3.2. Fine Tracking (Using MEMS Mirror)

For this test, the laser is steered by MEMS mirror, which is mounted on top of a servo mechanism. We use both MEMS mirror and servo mechanism since the MEMS is able to move quickly and precisely, but has a limited field-of-view (6 degrees in either direction). The servo is imprecise due to limitations of the potentiometer used for

positioning and internal gears, but is able to extend the field-of-view of the MEMS to cover every point above the horizontal plane. The details of how this control is done is illustrated below.

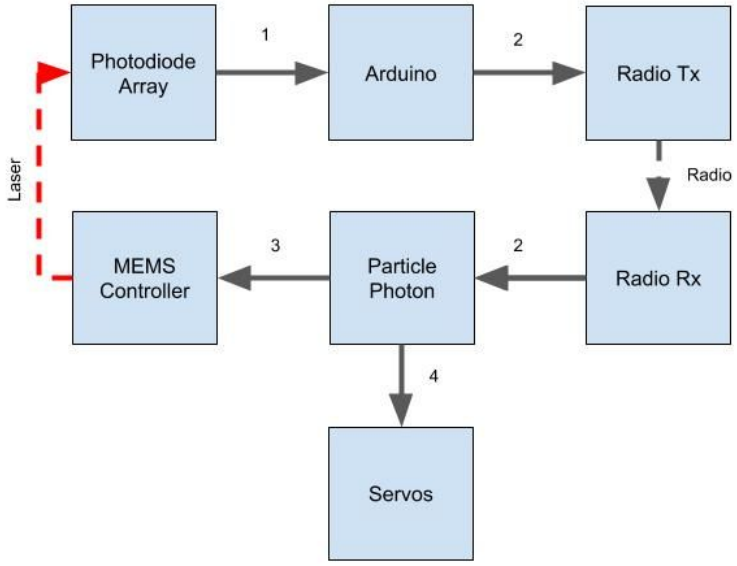


Figure 4. *This diagram illustrates the closed loop tracking system.*

Description of signal path:

1. Photodiode communicates the digital state of each of the 24 photodiodes to the Arduino on the target. This is done by connecting the output of each photodiode to a multiplexer input. The Arduino shifts through all the diode ids (from 1 to 24) and reads the output.

2. The Arduino averages the coordinates of the diodes that are on to obtain the approximate coordinate of the center of the laser beam with respect to the center of the photodiode array. This coordinate (8 bits for x and 8 bits for y) is transmitted to the particle photon.

3. Using the error coordinates from the photodiode array, we generate the new pair of voltages to send to the MEMS controller. More precisely, let the error coordinates received be vector **e**. Then, the voltage output vector **o** can be expressed as,

$$\mathbf{o}(t) = \mathbf{o}(t - 1) + k * \mathbf{e}(t) + (\mathbf{o}(t) - \mathbf{o}(t - i)) / i$$

All vectors are pairs of x-y positions. Constant *k* can be tuned to change how the MEMS responds to error coordinates. Constant *i*, which we set to 100, adjusts how the velocity vector is calculated. Large *i* will make MEMS slow to respond to

sudden changes in velocity, but low *i* will make it susceptible to randomness (noise) in the position of the laser.

4. **This step was not demonstrated during functional testing.** Using the position of the MEMS as the error function for PID control, we generate a new pair of coordinates for the Servos to move to.

3.3. Coarse Tracking Algorithm

In this test we demonstrated our ability to track the coarse position of an object using the Raspberry Pi and a the Pi Camera. The test analyzed the input from the camera to isolate the desired object by looking at specific ranges of the hue, saturation and color value found in the image. The output of this operation would allow us to know the x,y position of the object tracked in terms of the image's resolution (i.e. pixel height, pixel width).

The Raspberry Pi will be able to control the servo movement to track the position of the object as it moves to prevent it from escaping the camera's field of view. As we track the object and get it's relative position we can communicate with the Photon via USB Serial to transmit this information, once an object to track has been defined and its hue, saturation and value ranges have been defined.

Once the object that we want to track enters the camera's field of view, the image displayed will show a circle surrounding the object tracked and a red dot at the center of mass of the object. If the object moves outside of the central area of the camera's view it will trigger the servos to maintain the object at the center.

The biggest limitations of this tracking mechanism is that it is very dependant on the lighting that the object is receiving. The HSV range need to be very flexible in order to track in most environments, but this will also cause the algorithm to get confused with other objects that might also fall within the same HSV range.

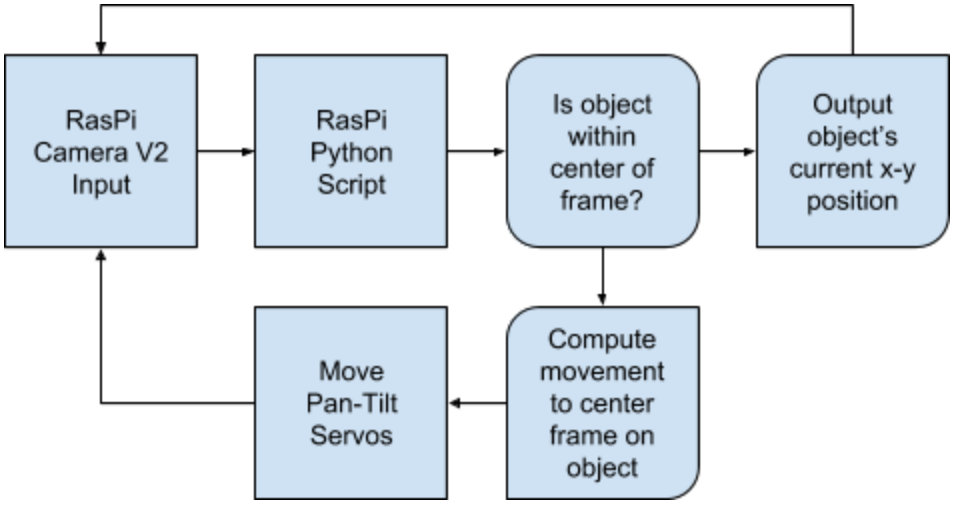


Figure 3. *This diagram illustrates the control diagram that follows this algorithm.*

4. Measurements

Due to the nature of our project, we are more concerned with building reliable hardware and software components to make closed loop tracking consistently, rather than taking concrete measurements. Here are some rough numerical measurements that are pertinent:

- Latency from shining light on photodiode to response from MEMS: **2 ms**.
- Minimum distance across which tracking can be achieved: **2.5 meters.**
- Maximum constant speed at which target can move with reliable tracking: **10 m/s**.
- Maximum angle of incidence measured with respect to the vector normal to the plane of the photodiodes: **30 degrees.**
- Maximum magnitude of the acceleration: **½*g.**

All of the measurements taken above are rough estimates. We are unable to make precise measurements, since we lack the equipment to precisely measure quantities such as target acceleration and velocity.

5. Conclusion

5.1. In our test we successfully tracked an array of photodiodes. By the time of installation, we will be passing bits from our ground station to the drone. This test shows that we have viable methods for optical wireless communication. For installation, we will have to mount the photodiode array on the drone, and improve our tracking solution. However, having the optical wireless

communications working is a fundamental part of our project which we have successfully implemented.

5.2. For the final test we implemented a coarse acquisition algorithm using a Raspberry Pi camera that was able to track an object and follow it using the servos on the mount.

5.3. Summary of next steps:

- Integrate coarse and fine tracking so that they work together before or during installation
- Send bits and track at the same time
- Integrate Lidar to fine tune tracking by getting more exact angle for moving the laser
- Integrate motion of servos with MEMS mirror in order to do fine tracking for wider range of angles

6. Appendix

Please note that for the Particle Photon Code, all tracking is done in the loop() function. The rest of the code declares various constants, variables, and support functions (most of which we did not write). The MEMS code is based on code provided by Emily Lam.

6.1. Fine Tracking Algorithm (Runs on Particle Photon)

```
void loop() {
  // Set analog voltage X+, X-, Y+, Y- from center (analog_bias) always at 80 V
  // Max voltage is 157 so max analog voltage is 157 - if you go beyond 157, the DAC will set to 80 V
and your code won't work.

  difX = 0; //Can be positive or negative
  difY = 0;
  incX = 0;
  incY = 0;
  setChannels(difX, difY);

  while(1) {
    while(! radio.available()); //wait for new communication
    radio.read(&radio_data, sizeof(radio_data[0])*2);
    p++;
    bufX[p] = difX;
    bufY[p] = difY;
    q = p+1;
    if (p>=BUF_SIZE) p=0;
    if (q>=BUF_SIZE) q=0;

    incX = (bufX[p]-bufX[q])/BUF_SIZE;
    incY = (bufY[p]-bufY[q])/BUF_SIZE;
    difX += incX;
    difY += incY;
    difX += (double)(radio_data[0])/127.0 * inc;
    difY += (double)(radio_data[1])/127.0 * inc;

    azimuth_servo_pos = azimuth_controller_update(difX);
    azimuth_write(azimuth_servo_pos);
    setChannels(difX, difY);
    serial_counter++;
    if (serial_counter > 50){
      Serial.print(difX);
      Serial.print(',');
      Serial.print(difY);
      Serial.print(';');
      Serial.println(azimuth_servo_pos);
      serial_counter = 0;
    }
  }
  powerdown(); //Always power down at the end to disable the MEMS output.. If you don't do this and
disconnect the power you may break the MEMS
}

int azimuth_controller_update(double mems_pos) {
    const double ki = 0.0, kp=0.1, kd=0.0;
    static double err_i = 0;
    static double mems_last = 0;
    double err_p, err_d;

    err_i += mems_pos;
```

```
    err_p = mems_pos;
    err_d = mems_pos - mems_last;

    mems_last = mems_pos;

    return (int)(1500 + (err_i * ki) + (err_p * kp) + (err_d * kd));
}

void azimuth_write(int pos){
  if (pos < 1200 or pos > 1800) return;
  azimuth_servo.writeMicroseconds(pos);
}
```

6.2. Coarse Tracking Algorithm (Runs on Raspberry Pi)

```
def main():

        usb = False             # True : Try to communicate with serial
                                # False: Do not communicate

        servo = True    # True : Communicate with the servos
                                # False: Do not initiate servos

        windows = 2             # 0: Do not show any windows
                                # 1: Show only the tracked window
                                # 2: Show all the windows

        # Set the (h,s,v) lower and upper bounds
```

```
lowerbound = (0,80,80)
upperbound = (10,255,200)

# Duty Cycle Values
# set to rest values
dc_p = 6.0
dc_t = 4.5

cap, port, pwm_p, pwm_t = setup(usb, servo, windows, lowerbound, upperbound)


while(1):

        # MOVING SERVO
        #pwm_p.ChangeDutyCycle(dc_p)
        pwm_t.ChangeDutyCycle(dc_t)

        # Read frame
        _, frame = cap.read()

        # Blur the frame
        blurred = cv2.GaussianBlur(frame, (5,5), 0)

        # Converting to HSV
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
        hue,sat,val = cv2.split(hsv)

        # Get info from track bar and appy to result
        if windows == 2:
                lowerbound, upperbound = getBounds()

        # construct a mask from the bounds obtained from trackbars, then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        mask = cv2.inRange(hsv, lowerbound, upperbound)
        mask = cv2.erode(mask, None, iterations=2)
        closing = cv2.dilate(mask, None, iterations=2)

        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
        center = None

        # only proceed if at least one contour was found
        if len(cnts) > 0:
                # find the largest contour in the mask, then use
                # it to compute the minimum enclosing circle and
                # centroid
                c = max(cnts, key=cv2.contourArea)
                ((x, y), radius) = cv2.minEnclosingCircle(c)
                M = cv2.moments(c)
                if M["m00"] != 0:
                        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
                        #dc_p, dc_t = moveServo(center[0],center[1], dc_p, dc_t)
                        port.write(str(center)+"\n") if usb else None

                # only proceed if the radius meets a minimum size
                if radius > 10:
                        # draw the circle and centroid on the frame
                        cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255, 255), 2)
                        cv2.circle(frame, center, 5, (0, 0, 255), -1)

        # display wanted windows
        if windows == 1 or windows == 2:
```

```
                        frame = np.rot90(np.rot90(frame)) # Rotate frame
                        cv2.imshow('tracking',frame)

                        if windows == 2:
                                hthresh = cv2.inRange(np.array(hue), np.array(lowerbound[0]),
np.array(upperbound[0]))
                                sthresh = cv2.inRange(np.array(sat), np.array(lowerbound[1]),
np.array(upperbound[1]))
                                vthresh = cv2.inRange(np.array(val), np.array(lowerbound[2]),
np.array(upperbound[2]))

                                hthresh = np.rot90(np.rot90(hthresh)) # Rotate frame
                                sthresh = np.rot90(np.rot90(sthresh)) # Rotate frame
                                vthresh = np.rot90(np.rot90(vthresh)) # Rotate frame

                                #Show the result in frames
                                cv2.imshow('HueComp',hthresh)
                                cv2.imshow('SatComp',sthresh)
                                cv2.imshow('ValComp',vthresh)
                                cv2.imshow('closing',closing)

                # exit when 'q' key is pressed
                k = cv2.waitKey(1) & 0xFF
                if k == ord("q"):
                        break

        #GPIO.cleanup()
        cap.release()
        cv2.destroyAllWindows()
```