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 Trac

Date: 02/13/18

Subject: Second Deliverable 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

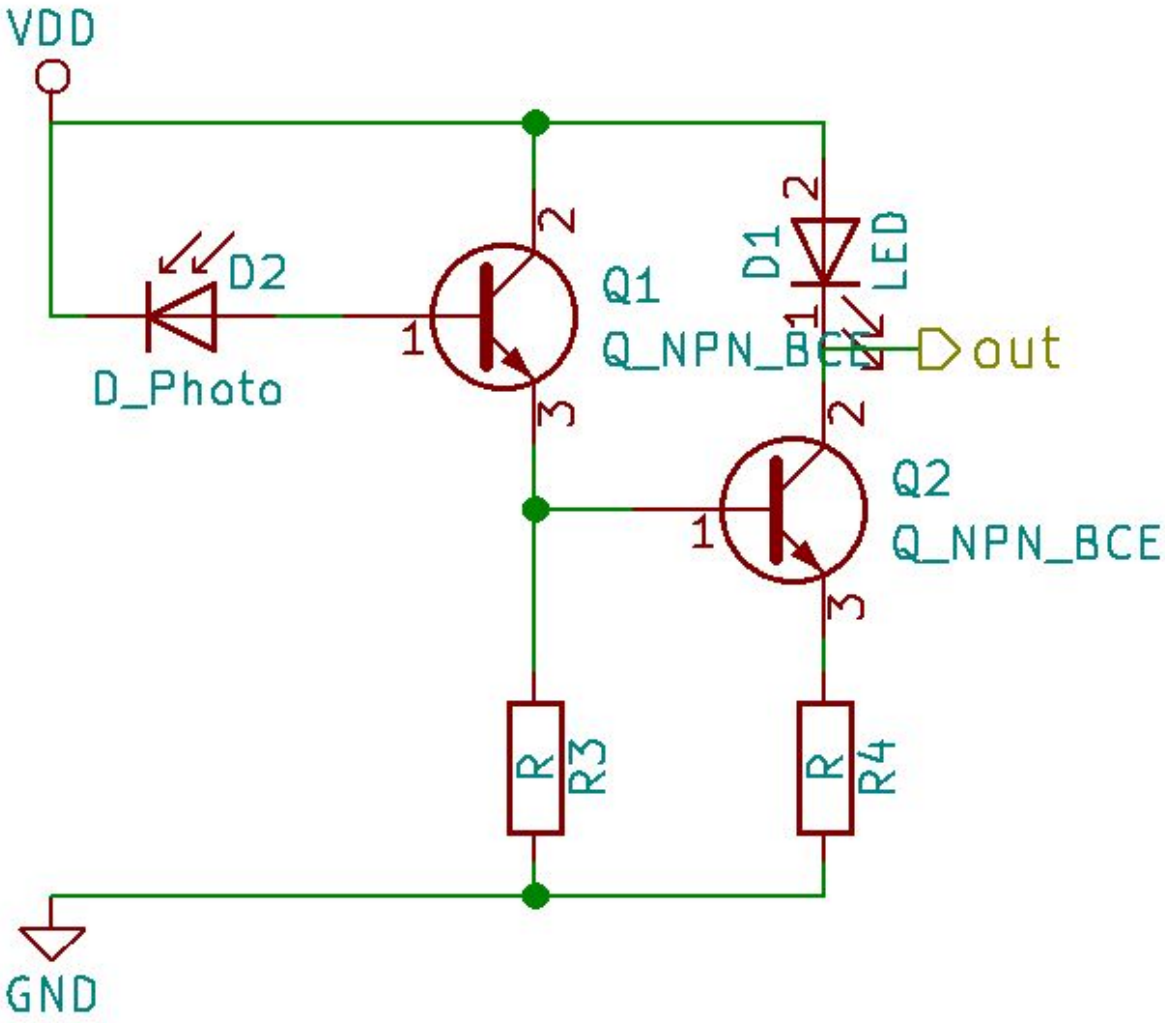
2.1. This test was significant because it demonstrated our ability to track the sensor, which is a fundamental component of our project. We demonstrated our closed loop control system, which included two wireless links -- the laser-to-sensor component and the RF feedback signal.

2.2. This test focused on achieving a rough example of a tracking mechanism. With this test we demonstrated an ability to track an object with a laser while utilizing a photodiode array in the way we hope to for our final project. We successfully demonstrated a single-photodiode, 2-dimensional tracking system which could follow the array at a reasonable speed. The most important part of this test is how we incorporated the photodiode array into the tracking process. This is a technique we hope to use in our final solution to the laser tracking problem.

3. Equipment and Setup

3.1. Receiver Circuit

This is the main hardware component of of our project.



The photodiode circuit is tiled 16 times in our sensor array. Each output is treated as a single analog value, which is pulled low when the photodiode detects the presence of a laser.
The LED in our circuit provides visual feedback for which signal is currently active. These indicators showed that 13 of the 16 circuit elements worked reliably, and 3 appeared to have broken photodiodes. We will fix this problem by replacing the failing units as they are detected in future versions of the circuit.
In our tests, we used (algorithm 1) 5 photodiodes and (algorithm 2) 1 photodiode, so the missing signals were not a problem.

3.2. Tracking Algorithm I

The first tracking algorithm is an extension of the naive tracking algorithm from first deliverable testing from one dimension to two dimensions. In this test we demonstrated a rudimentary tracking mechanism. A laser is pointed with a MEMS mirror at a cluster of 5 photodiodes on the receiver. The five photodiodes are arranged in a cross configuration. We demonstrated that as the receptor moves left and right or up and down, the laser is able to follow.

The MEMS is programmed to move at a constant speed when it is not pointing at the target. When the laser hits the left photodiode, an interrupt is triggered and the MEMS changes its direction to move right. When the laser hits the right photodiode, an interrupt is triggered and the MEMS turns left. Similar interrupts are programmed for the up and down photodiodes. This test is an improvement upon the testing done last semester, since the laser will not move when it is directly hitting the target, so the tracking algorithm is slightly more stable.

Code is included in appendix.

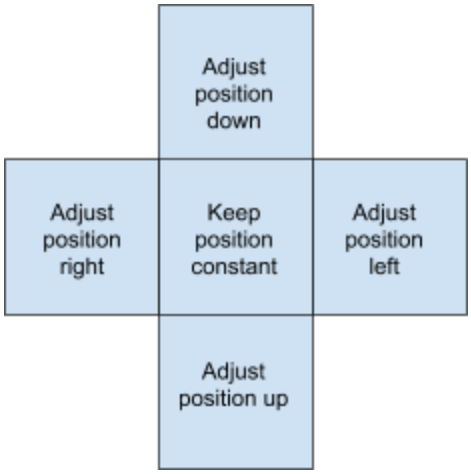


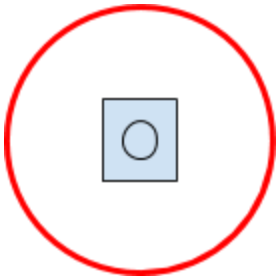
Figure 1. *Diagram of the five photodiodes used in this tracking algorithm along with expected response by the laser/MEMS.*

3.3. Tracking Algorithm II

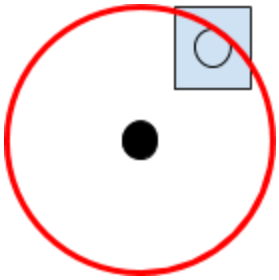
In this test we tested tracking using one photodiode. The laser is initially aimed at the target. While the laser is aimed at the target at any point during tracking, the laser will not move (same as the first algorithm). As soon as the target moves out of the laser beam, the MEMS mirror will draw a circle with the laser beam around the previous known position of the target. Specifically, this is done by moving the laser to 32 points along the circle counterclockwise at 200 microsecond increments. When the target hits the circle, an interrupt is triggered and the laser stops. At this point, we immediately

move the laser clockwise a fixed number of increments. This is where we think the laser hit the target (hence where the target is). We have to backtrack a certain number of increments because of latency between sending the target signal and receiving it.

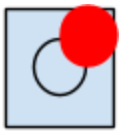
In practice, it looks like the target photodiode is dragging a circle around as it moves. Because we have not yet figured out a way to make latency deterministic, the laser occasionally over or under compensates for latency.



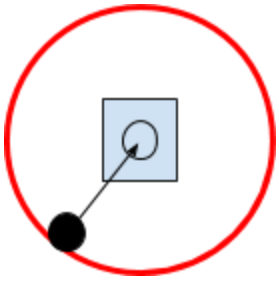
Initially, the laser ring is centered on the photodiode



When the laser ring hits the diode, we start to adjust the position



We move the photodiode back to where it was when it hit the photodiode



From that information, we know the direction to update the laser's centered position

Figure 2. *This diagram illustrates the sequence of events for this tracking algorithm. The black dot represents the last known location of the target. The red represents the laser beam.*

3.4. Block Diagram of Tracking Test

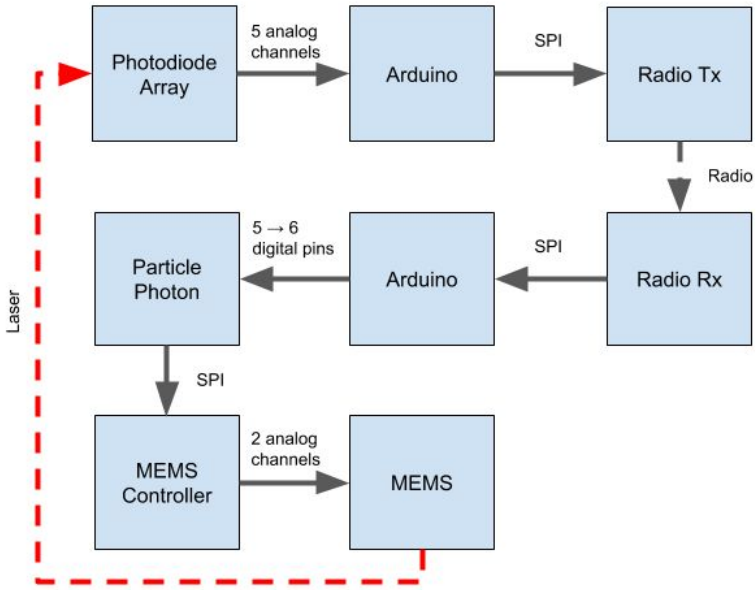


Figure 3. *This diagram illustrates the closed loop tracking system. Note that both tests use the same setup. Since the second tracking algorithm only uses one photodiode, it simply ignores the signals from the other four photodiodes. More detail is provided below.*

Description of signal path:

1. Photodiode circuit outputs analog value. Under ambient light, it outputs ~2.9 volts. When it sees the laser it outputs ~3.5 volts.
2. The output from the photodiode array is connected to 5 analog pins. In a loop, the values are read and compared against a threshold value to generate a digital value. The five bit values are shifted into one byte and communicated to the radio transmitter via SPI. The threshold value is determined on startup. All photodiodes are read and the threshold value is set 0.05 volts below the average analog value (the Arduino Leonardo uses 10 bit analog resolution, the analog threshold is set to be 50 steps below the average analog value of the photodiodes under ambient light).
3. The radio transmitter transmits the one byte payload. There is some transmission overhead.
4. The radio receiver receives the one byte payload.
5. The second Arduino reads the byte from radio receiver via SPI in an infinite loop. 5 digital pins are set to HIGH or LOW accordingly.
6. The input to the Particle Photon has 6 input pins. 5 pins trigger interrupts on the falling edge. The target photodiode needs an extra pin to trigger rising edge

interrupt, because we need to know when target is lost. The Photon is where the tracking algorithm lives. See the code in Appendix.

7. The rest of the signal path is provided off-the-shelf and is simplified in this diagram. The general idea is that everytime the Photon wants to move the MEMS, it sends a pair of voltage values to the MEMS controller via SPI (The voltage values correspond to degrees tilt around x and y axes). The controller then moves the MEMS accordingly.

4. Measurements

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

- Latency from shining light on photodiode to target interrupt triggered on Particle Photon: 1 - 2 ms. This a random distribution because of interference from WiFi and Bluetooth, which use the same frequency range, and potentially other factors that we have not yet considered.
- Distance across which tracking was achieved: 1 - 3 meters. We have not tested beyond 3 meters.
- Speed at which target can move at constant speed with reliable tracking: 1 - 2 m/s. Estimate is based purely on judgement.

5. Conclusion

5.1. In our test we successfully tracked an array of photodiodes. In future tests we will be passing bits from our ground station to the drone. This test shows that we have viable methods for optical wireless communication. In future tests 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. From the second test we take note that although we have successful tracking results in the azimuthal direction, we still have many areas of our tracking mechanism that require development. In future tests we will add onto the photodiode array techniques that we have developed in our second test. Some areas for future development include tracking in the elevation and azimuthal direction together and improving the tracking algorithms that tell the laser how much to shift when it hits an outer photodiode in the array.
We would like to track the drone's position and incorporate a model of its motion to improve the steady-state response of our tracking algorithm, which should allow us to track at a higher speed.

5.3. Summary of next steps:

- Establish deterministic latency.
- Incorporate a model of motion into tracking.
- Lens the laser to achieve wider collimated beam and wider field of view (series of diverging, collimating, and convex lenses).
- Shorten signal path leading to reduced latency.

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. Transmitter Code (Runs on First Arduino Leonardo)

```
/*
 * Adapted from code by Dejan Nedelkovski, www.HowToMechatronics.com
 * Library: TMRh20/RF24, https://github.com/tmrh20/RF24/
 */
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
RF24 radio(7, 8); // CE, CSN
const byte address[][6] = {"00001","00002"};
int threshold;
char data;

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(address[0]);
  radio.setPALevel(RF24_PA_MIN);
  radio.setAutoAck(0,false);
  radio.stopListening();

  pinMode(A0,INPUT_PULLUP);
  pinMode(A1,INPUT_PULLUP);
  pinMode(A2,INPUT_PULLUP);
  pinMode(A3,INPUT_PULLUP);
  pinMode(A4,INPUT_PULLUP);
  pinMode(A5,INPUT_PULLUP);
  threshold = analogRead(A1) - 20;
}
```

```
void loop() {
  data = (char)(analogRead(A1) < threshold);
  data |= (analogRead(A2) < threshold) << 1;
  data |= (analogRead(A3) < threshold) << 2;
  data |= (analogRead(A4) < threshold) << 3;
  data |= (analogRead(A5) < threshold) << 4;
  radio.write(&data, sizeof(data));
}
```

6.2. Receiver Code (Runs on Second Leonardo)

```
/*
 * Adapted from code by Dejan Nedelkovski, www.HowToMechatronics.com
 * Library: TMRh20/RF24, https://github.com/tmrh20/RF24/
 */
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
RF24 radio(7, 8); // CE, CSN
const byte address[][6] = {"00001","00002"};
unsigned long starttime;
char data;

void setup() {
  Serial.begin(115200);
  radio.begin();
  radio.openReadingPipe(0, address[0]);
  radio.setPALevel(RF24_PA_MIN);
  radio.setAutoAck(0,false);

  radio.startListening();
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
}
void loop() {
  if (radio.available()) {
    radio.read(&data, sizeof(data));
    Serial.println(data,BIN);
    if  (data & 1){
      digitalWrite(2,LOW);
    }
    else{
      digitalWrite(2,HIGH);
    }
```

```
    if  (data & (1<<1)){
      digitalWrite(3,LOW);
    }
    else{
      digitalWrite(3,HIGH);
    }
    if  (data & (1<<2)){
      digitalWrite(4,LOW);
    }
    else{
      digitalWrite(4,HIGH);
    }
    if  (data & (1<<3)){
      digitalWrite(5,LOW);
    }
    else{
      digitalWrite(5,HIGH);
    }
    if  (data & (1<<4)){
      digitalWrite(6,LOW);
    }
    else{
      digitalWrite(6,HIGH);
    }
  }
}
```

6.3. Tracking Algorithm I Code (Runs on Particle Photon)

```
// ------------
// Setup for Mirrorcle PicoAmp 4.6
// Code Adapted from Emily Lam
// ------------
#include "math.h"
SYSTEM_MODE(MANUAL);

int enable_pin  = D7; // (pin 1) pin to enable MEMS output
int fclk_pin    = D0; // (pin 2) pin to set frequency of Bessel low pass filter

int ss          = A2; // (pin 8)
int sck         = A3; // (pin 6)
int mosi        = A5; // (pin 4)
                      // Gnd (pin 3 and pin 5)
                      // +5V (pin 7)

int full_reset                  = 2621441; //Initialization to set up DAC
int enable_internal_reference   = 3670017; //Initialization to set up DAC
```

```
int enable_all_dac_channels     = 2097167; // B00100000 00000000 00001111//Initialization to
set up DAC
int enable_software_ldac        = 3145728; // B00110000 00000000 00000000 //Initialization
to set up DAC
int disable_all_dac_channels    = 2097215; //Power Down

#define AD56X4_COMMAND_WRITE_INPUT_REGISTER            0 //B00000000 //Set the channel(s)'s
input register.
#define AD56X4_COMMAND_UPDATE_DAC_REGISTER             8 //B00001000 //Update the DAC
register (output the set voltage)
#define AD56X4_COMMAND_WRITE_INPUT_REGISTER_UPDATE_ALL  16 //B00010000 //Set channel/s's
input register and then update all DAC registers from the input registers.

#define AD56X4_CHANNEL_A                               0 //B00000000
#define AD56X4_CHANNEL_B                               1 //B00000001
#define AD56X4_CHANNEL_C                               2 //B00000010
#define AD56X4_CHANNEL_D                               3 //B00000011
#define AD56X4_CHANNEL_ALL                             7 //B00000111

#define pi 3.1415926

// commands to write to LDAC buffer, then at channel_D write to DAC register
int writecommand[]  = {AD56X4_COMMAND_WRITE_INPUT_REGISTER  | AD56X4_CHANNEL_A,
                       AD56X4_COMMAND_WRITE_INPUT_REGISTER  | AD56X4_CHANNEL_B,
                       AD56X4_COMMAND_WRITE_INPUT_REGISTER  | AD56X4_CHANNEL_C,
                       AD56X4_COMMAND_WRITE_INPUT_REGISTER_UPDATE_ALL  | AD56X4_CHANNEL_D
                      };

// Set mirror dependent values
int freq            = 650; // 650 Hz recommended cutoff frequency
int fclk_freq       = 60*freq; //39000;    // Recommended 650 Hz * 60 - do not change!!!!!!
double max_voltage  = 157;      // Datasheet spec - do not change!!!!!!
int bias_voltage    = 80;       // Datasheet spec - do not change!!!!!!
double analog_bias  = (bias_voltage*65535)/160 + 0.5;
int dig_bias        = (int) analog_bias;
int data;

volatile bool left = true;
volatile bool up   = true;
volatile bool targeted = false;
volatile bool target_lost = false;
unsigned long counter = 0;
unsigned long target_time = 0;

// Required constants
int numloops        = 0;
int maxloops        = 10;
double difX, difY;
volatile double targetX, targetY;
```

```
/* interrupt service routines */
void move_left();
void move_right();
void move_up();
void move_down();
void set_target();
void release_target();

void setup() {
  pinMode(enable_pin, OUTPUT);
  pinMode(fclk_pin, OUTPUT);
  pinMode(ss, OUTPUT);
  pinMode(sck, OUTPUT);
  pinMode(mosi, OUTPUT);

  /* Power down pin */
  pinMode(A0,INPUT_PULLUP);

  /* Interrupt pins */
  pinMode(D6,INPUT);        //TARGET RELEASE
  pinMode(D5,INPUT_PULLUP); //LEFT
  pinMode(D4,INPUT_PULLUP); //RIGHT
  pinMode(D3,INPUT_PULLUP); //UP
  pinMode(D2,INPUT_PULLUP); //DOWN
  pinMode(D1,INPUT_PULLUP); //TARGET
  attachInterrupt(D5, move_right, FALLING);
  attachInterrupt(D4, move_left, FALLING);
  attachInterrupt(D3, move_up, FALLING);
  attachInterrupt(D2, move_down, FALLING);
  attachInterrupt(D1, set_target, FALLING);
  attachInterrupt(D6, release_target, RISING);

  SPI.begin();
  SPI.begin(SPI_MODE_MASTER); // SPI master
  SPI.setClockDivider(SPI_CLOCK_DIV4); // 60 MHz over 4 -> 15 MHz
  SPI.setBitOrder(MSBFIRST); // Bit 23 is first in buffer
  SPI.setDataMode(SPI_MODE1);  // According to github library

  analogWrite(fclk_pin, 127, fclk_freq); // Begin Bessel filter clock set at 50% duty cycle

  digitalWrite(enable_pin,LOW);
  initialization();
  Serial.begin();
}

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;
  targetX = 0;
  targetY = 0;

  while(1) {
    if (target_lost){
      target_time = counter;
      target_lost = false;
      setChannels(targetX, targetY);
      difX = targetX;
      difY = targetY;
    }
    if (targeted){
      setChannels(targetX, targetY);
      while(targeted){
        target_time = counter;
      }
    }
    if (left){
      difX -= 0.02;
      if (difX < -80) left = !left;
    }
    else{
      difX += 0.02;
      if (difX > 80) left = !left;
    }
    if (up){
      difY += 0.02;
      if (difY > 80) up = !up;
    }
    else{
      difY -=0.02;
      if (difX < -80) up = !up;
    }
    setChannels(difX, difY);
    if (counter - target_time > 100){
      target_lost = true;
     }
    if (digitalRead(A0) == LOW) break;
    delayMicroseconds(100);
    counter++;
  }
  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
}

void move_right(){
  left = false;
  targeted = false;
```

```
}

void move_left(){
  left = true;
  targeted = false;
}

void move_up(){
  up = true;
  targeted = false;
}

void move_down(){
  up = false;
  targeted = false;
}

void set_target(){
  targetX = difX;
  targetY = difY;
  targeted = true;
}

void release_target(){
  targeted = false;
}
```

6.4. Tracking Algorithm II Code (Runs on Particle Photon)

```
// ------------
// Setup for Mirrorcle PicoAmp 4.6
// Code Adapted from Emily Lam
// ------------
#include "math.h"
SYSTEM_MODE(MANUAL);

int enable_pin  = D7; // (pin 1) pin to enable MEMS output
int fclk_pin    = D0; // (pin 2) pin to set frequency of Bessel low pass filter
int ss          = A2; // (pin 8)
int sck         = A3; // (pin 6)
int mosi        = A5; // (pin 4)
                      // Gnd (pin 3 and pin 5)
                      // +5V (pin 7)

int full_reset                  = 2621441; //Initialization to set up DAC
int enable_internal_reference   = 3670017; //Initialization to set up DAC
int enable_all_dac_channels     = 2097167; // B00100000 00000000 00001111//Initialization to
```

```
set up DAC
int enable_software_ldac        = 3145728; // B00110000 00000000 00000000 //Initialization
to set up DAC
int disable_all_dac_channels    = 2097215; //Power Down

#define AD56X4_COMMAND_WRITE_INPUT_REGISTER             0 //B00000000 //Set the channel(s)'s
input register.
#define AD56X4_COMMAND_UPDATE_DAC_REGISTER              8 //B00001000 //Update the DAC
register (output the set voltage)
#define AD56X4_COMMAND_WRITE_INPUT_REGISTER_UPDATE_ALL  16 //B00010000 //Set channel/s's
input register and then update all DAC registers from the input registers.

#define AD56X4_CHANNEL_A                                0 //B00000000
#define AD56X4_CHANNEL_B                                1 //B00000001
#define AD56X4_CHANNEL_C                                2 //B00000010
#define AD56X4_CHANNEL_D                                3 //B00000011
#define AD56X4_CHANNEL_ALL                              7 //B00000111

#define pi 3.1415926
// commands to write to LDAC buffer, then at channel_D write to DAC register
int writecommand[]  = {AD56X4_COMMAND_WRITE_INPUT_REGISTER  | AD56X4_CHANNEL_A,
                       AD56X4_COMMAND_WRITE_INPUT_REGISTER  | AD56X4_CHANNEL_B,
                       AD56X4_COMMAND_WRITE_INPUT_REGISTER  | AD56X4_CHANNEL_C,
                       AD56X4_COMMAND_WRITE_INPUT_REGISTER_UPDATE_ALL  | AD56X4_CHANNEL_D
                     };

// Set mirror dependent values
int freq            = 650; // 650 Hz recommended cutoff frequency
int fclk_freq       = 60*freq; //39000;    // Recommended 650 Hz * 60 - do not change!!!!!!
double max_voltage  = 157;      // Datasheet spec - do not change!!!!!!
int bias_voltage    = 80;       // Datasheet spec - do not change!!!!!!
double analog_bias  = (bias_voltage*65535)/160 + 0.5;
int dig_bias        = (int) analog_bias;
int data;

volatile bool left        = true;
volatile bool up          = true;
volatile bool targeted    = false;
volatile bool target_lost = false;
unsigned long counter     = 0;
unsigned long target_time = 0;

// Required constants
int numloops        = 0;
int maxloops        = 10;
double difX, difY;
volatile double targetX, targetY;

/* interrupt service routines */
void set_target();
```

```
void release_target();

void setup() {
  pinMode(enable_pin, OUTPUT);
  pinMode(fclk_pin, OUTPUT);
  pinMode(ss, OUTPUT);
  pinMode(sck, OUTPUT);
  pinMode(mosi, OUTPUT);

  /* Power down pin */
  pinMode(A0,INPUT_PULLUP);

  /* Interrupt pins */
  pinMode(D6,INPUT);        //TARGET RELEASE
  pinMode(D1,INPUT_PULLUP); //TARGET
  attachInterrupt(D1, set_target, FALLING);
  attachInterrupt(D6, release_target, RISING);
  attachInterrupt(A0, powerdown, FALLING);

  SPI.begin();
  SPI.begin(SPI_MODE_MASTER); // SPI master
  SPI.setClockDivider(SPI_CLOCK_DIV4); // 60 MHz over 4 -> 15 MHz
  SPI.setBitOrder(MSBFIRST); // Bit 23 is first in buffer
  SPI.setDataMode(SPI_MODE1);  // According to github library

  analogWrite(fclk_pin, 127, fclk_freq); // Begin Bessel filter clock set at 50% duty cycle
  digitalWrite(enable_pin,LOW);
  initialization();
  Serial.begin();
}

void loop() {
  difX = 0; //Can be positive or negative
  difY = 0;
  targetX = 0;
  targetY = 0;
  long int start,done,finish;
  int latency, calc_delay, setchannel_delay;
  double inc           = 0.1;
  int angle_delay      = 100;
  long int latency_sum = 0;
  int latency_min      = 2147483647;
  int tmp              = 0;
  int count            = 0;
  long r               = random(0,100);
  double angle         = ((double)r)/100.0;
  double offsetX       = ((double)r)/100.0;
  double offsetY       = ((double)r)/100.0;
  double vX = 0; //speed in volts/100 micros
  double vY = 0; //speed in volts/100 micros
```

```
double cos_table [32];
double sin_table [32];

for (int i= 0 ; i<32 ; i++){
  cos_table[i] = cos(i*0.2);
  sin_table[i] = sin(i*0.2);
}

/************* DELAY TEST ***************/
for (int i=0 ; i<32 ; i++) {
    start = micros();
    difX = cos_table[i]+offsetX;
    difY = sin_table[i]+offsetY;
    setChannels(0, 0);
    finish = micros();
    latency_sum += finish - start;
}
Serial.println(tmp);
calc_delay = latency_sum / 32;
Serial.print("CALCULATION DELAY IN MICROS: ");
Serial.println(calc_delay);
// Previous measurement: 183 micros

//make circle around target in 0.1 rad increments with 100 micros delay
latency_sum = 0;
for (int i=0 ; i<100 ; i++) {
    start = micros();
    setChannels(0, 0);
    finish = micros();
    latency_sum += finish - start;
}
setchannel_delay = latency_sum / 100;
Serial.print("SETCHANNEL DELAY IN MICROS: ");
Serial.println(setchannel_delay);

/* set increment according to latency. */
latency = 2000;
angle_delay = 100 - calc_delay;
double fallback = ((double)latency/100.0);

/************* MAINLOOP ***************/
while(1) {
  while(targeted){
    setChannels(difX, difY);
  }
  int i = 0;
  offsetX = difX;
  offsetY = difY;
  Inc     = 2;
  count   = 0;
```

```
    while (!targeted){
      //make circle around target in 0.1 rad increments with 100 micros delay
      difX = inc*cos_table[i]+offsetX;
      difY = inc*sin_table[i]+offsetY;
      setChannels(difX, difY);
      i++;
      count ++;
      if (i > 32) i = 0;
      delayMicroseconds(angle_delay);
    }
    i = i - fallback;
    if (i<0) i = 32+i;
    difX = inc*3.0/4.0*cos_table[i]+offsetX;
    difY = inc*3.0/4.0*sin_table[i]+offsetY;
    setChannels(difX, difY);
  }
}

void set_target(){
  targeted = true;
}

void release_target(){
  targeted = false;
}
```

6.5. MEMS Library Functions Needed for Tracking Code to Work. Provided by Emily Lam.

```
// All functions for this to work
void initialization() {         //Initialization to set up DAC

    digitalWrite(ss,LOW);
    SPI.transfer((full_reset>>16) & 0xFF);  // bits 16-23
    SPI.transfer((full_reset>>8) & 0xFF);   // bits 8-15
    SPI.transfer(full_reset & 0xFF);        // bits 0-7
    digitalWrite(ss,HIGH);

    digitalWrite(ss,LOW);
    SPI.transfer((enable_internal_reference>>16) & 0xFF);
    SPI.transfer((enable_internal_reference>>8) & 0xFF);
    SPI.transfer(enable_internal_reference & 0xFF);
    digitalWrite(ss,HIGH);

    digitalWrite(ss,LOW);
    SPI.transfer((enable_all_dac_channels>>16) & 0xFF);
    SPI.transfer((enable_all_dac_channels>>8) & 0xFF);
```

```
    SPI.transfer(enable_all_dac_channels & 0xFF);
    digitalWrite(ss,HIGH);

    digitalWrite(ss,LOW);
    SPI.transfer((enable_software_ldac>>16) & 0xFF);
    SPI.transfer((enable_software_ldac>>8) & 0xFF);
    SPI.transfer(enable_software_ldac & 0xFF);
    digitalWrite(ss,HIGH);

    digitalWrite(enable_pin,HIGH); // Enable MEMS Output
    setChannels(0,0); // Set MEMS to bias offset of 80 V
}

void setChannels(double difX, double difY) {

  //check that the maximum voltage does not exceed 157 V

  double Xp = bias_voltage + difX;
  double Xm = bias_voltage - difX;
  double Yp = bias_voltage + difY;
  double Ym = bias_voltage - difY;

  if (Xp > max_voltage | Xm > max_voltage | Yp > max_voltage | Ym > max_voltage) {
    difX = 0; // If it exceeds 157 V, set the output to the bias voltage.
    difY = 0; // If it exceeds 157 V, set the output to the bias voltage.
  }

  //convert difX and difY to digital values
  double voltage[] = {Xp, Xm, Yp, Ym};

  digitalWrite(ss,HIGH); //pull out of idle

  for (int i = 0; i < 4; i++) {
    double analog_voltage = (voltage[i]*65535)/160 + 0.5; // Convert from voltage to digital
value
    data = (int) analog_voltage;

    digitalWrite(ss,LOW);
    SPI.transfer(writecommand[i]); //Defined above to write to LDAC and write to DAC
register on last channel
    SPI.transfer((data >> 8) & 0xFF);
    SPI.transfer(data & 0xFF);
    digitalWrite(ss,HIGH);
  }

  // DAC update all outputs simultaneously
  digitalWrite(ss,LOW);
  SPI.transfer(AD56X4_COMMAND_UPDATE_DAC_REGISTER | AD56X4_CHANNEL_ALL);
  SPI.transfer(0);
  SPI.transfer(0);
```

```
  digitalWrite(ss,HIGH);

  digitalWrite(ss,LOW); //idle for low current mode without powering down
}

void powerdown() {

  digitalWrite(enable_pin,LOW); //Disable MEMS output

  digitalWrite(ss,HIGH); //pull out of idle
  digitalWrite(ss,LOW);
  SPI.transfer((full_reset>>16) & 0xFF);  // bits 16-23
  SPI.transfer((full_reset>>8) & 0xFF);   // bits 8-15
  SPI.transfer(full_reset & 0xFF);        // bits 0-7
  digitalWrite(ss,HIGH);

  //Power down to idle
  digitalWrite(ss,LOW);
  SPI.transfer((disable_all_dac_channels>>16) & 0xFF);
  SPI.transfer((disable_all_dac_channels>>8) & 0xFF);
  SPI.transfer(disable_all_dac_channels & 0xFF);
  digitalWrite(ss,HIGH);
  digitalWrite(ss,LOW);
}
```