Supplementary Material for:

Complete Stokes vector analysis with a compact, portable rotating waveplate polarimeter

T.A. Wilkinson, C.E. Maurer, C.J. Flood, G. Lander, S. Chafin, and E.B. Flagg
Department of Physics and Astronomy, West Virginia University, Morgantown, West Virginia 26506, USA

Sections*

1) Drawings of Mechanical System
2) Calibrating the Polarimeter
3) Circuit Diagram and Printed Circuit Board (PCB)
4) Arduino Code
5) LCVR Calibration
6) Bill of Materials
7) Calibrating the Azimuth of an Optic to a Rotation Mount

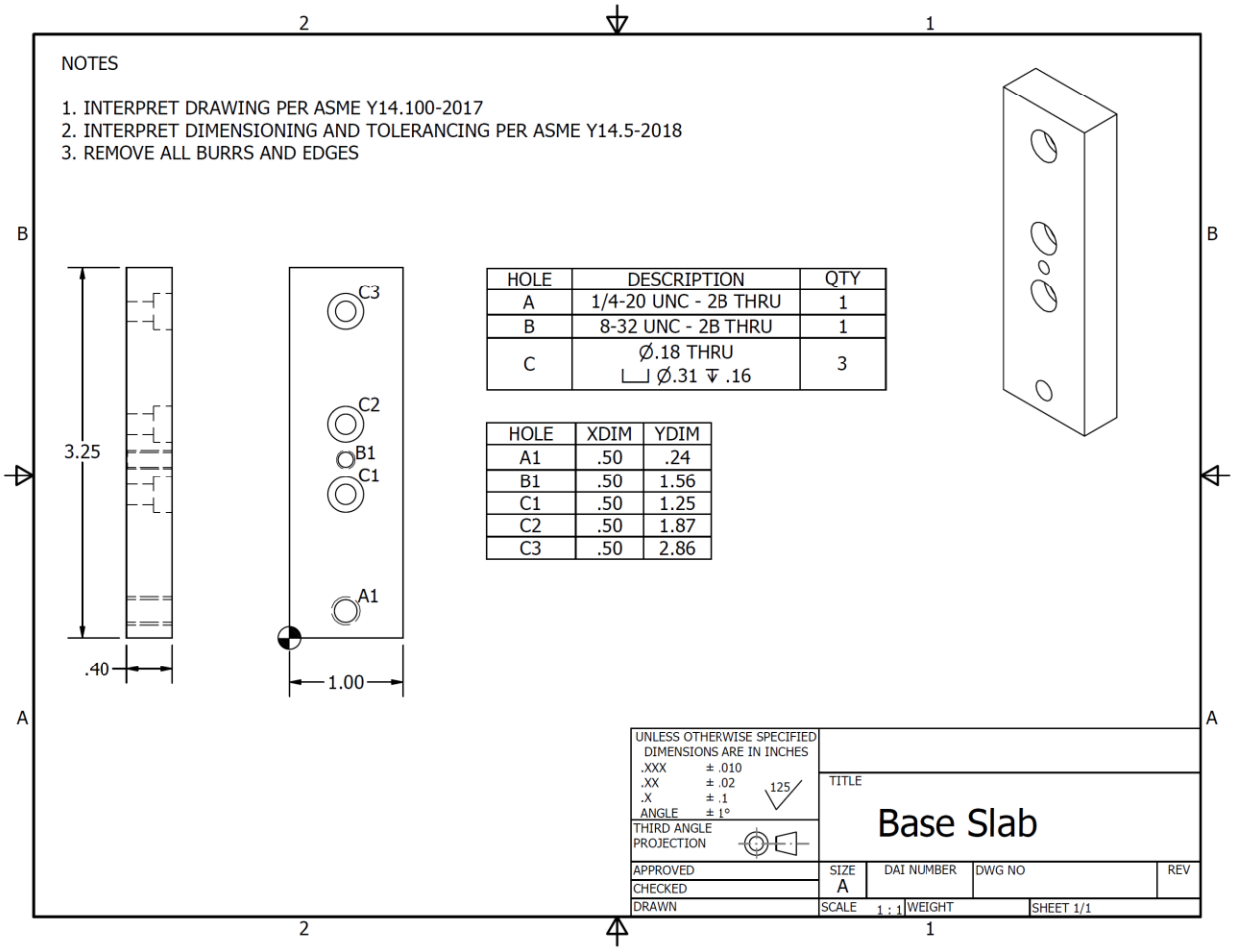
*Note: the order of sections corresponds to the order in which they are referenced in the main text.

Drawings of Mechanical System

We include here a drawing for each of the mechanical parts necessary to construct the measurement part of the device, as seen in the main text Fig. 2(b). The final drawing is an exploded view of the entire device, which shows explicitly how each part fits together

Base Slab

All components of the measurement device are mounted on this piece. The three counterbored 8-32 holes (C1-C3) are for the motor holder, the analyzer, and the photodiode. A standard optical post mounts to the central threaded 8-32 hole (B1). The threaded ¼-20 hole (A1) on the end mounts the photo-interrupter holder.



Motor Holder

This piece mounts to the base slab directly in front of the analyzer. The hollow axle motor (Tiger GB2208-80) is secured via the 4 counterbored holes for M3 screws (B1-B4).

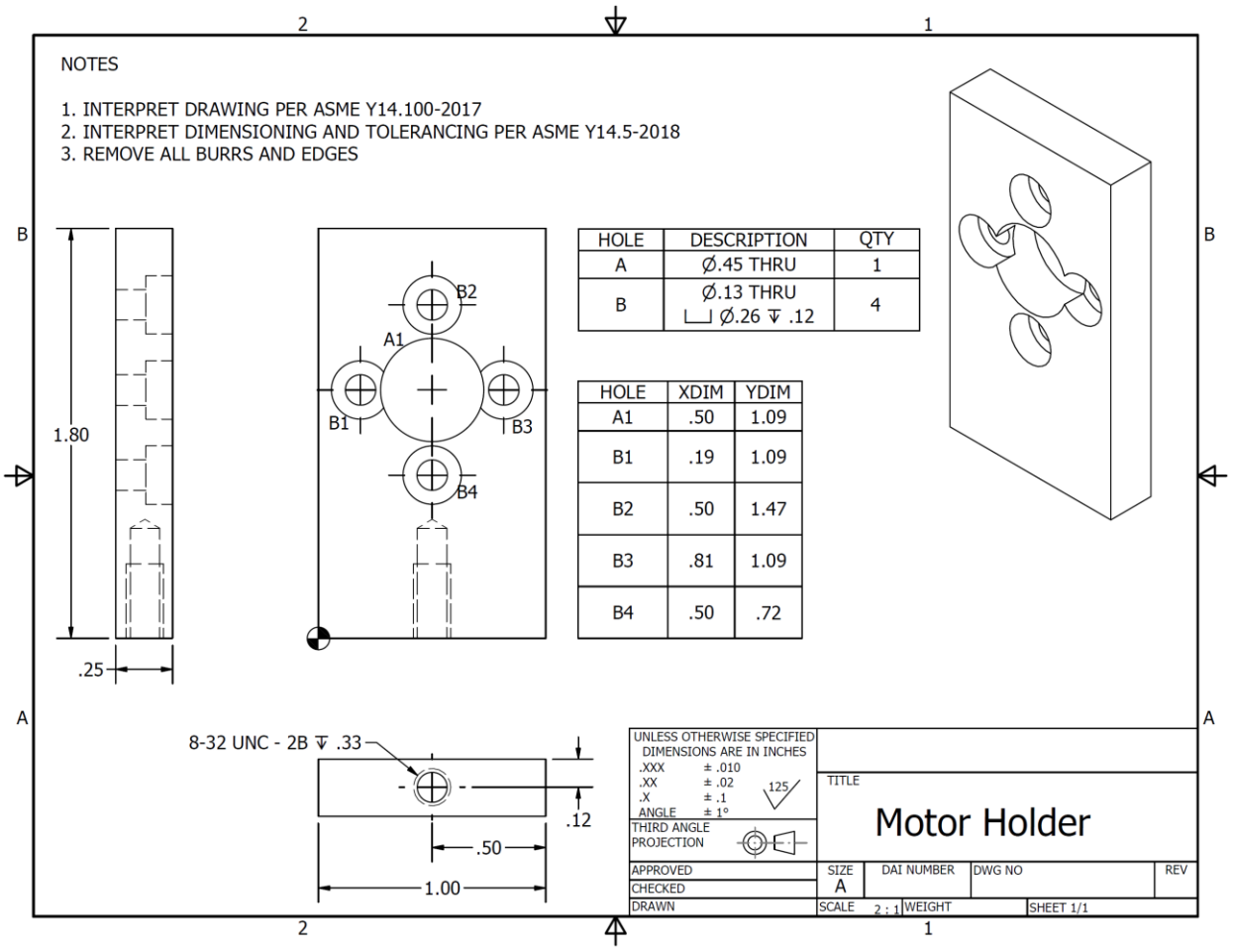
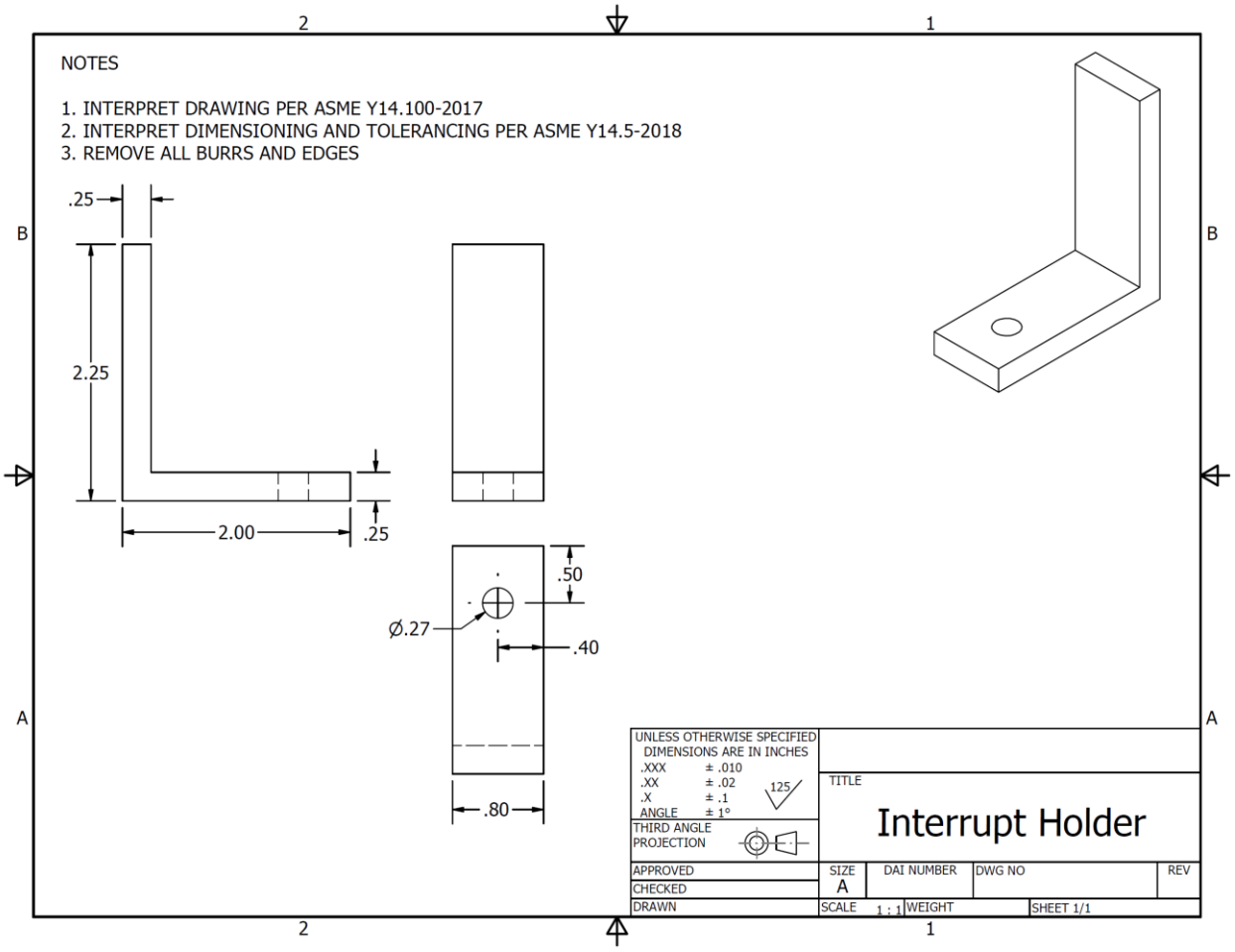


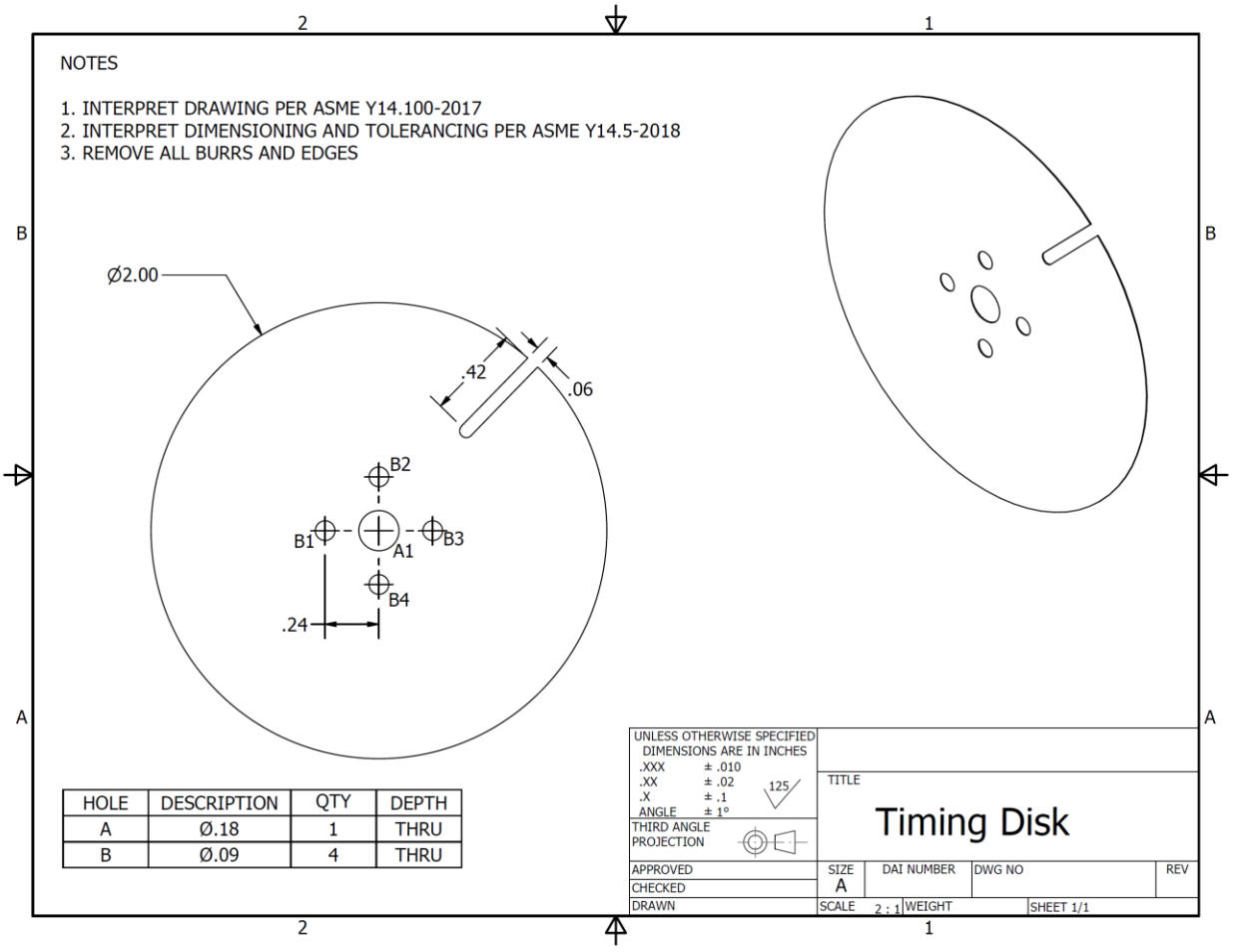
Photo-Interrupt Mount

This piece mounts to the base slab via the threaded ¼-20 hole near the end (A1). The photo-interrupter is attached to a small piece of PCB that is secured to the vertical arm of the mount using rubber bands, which allows a firm hold while also allowing small movements for calibration.



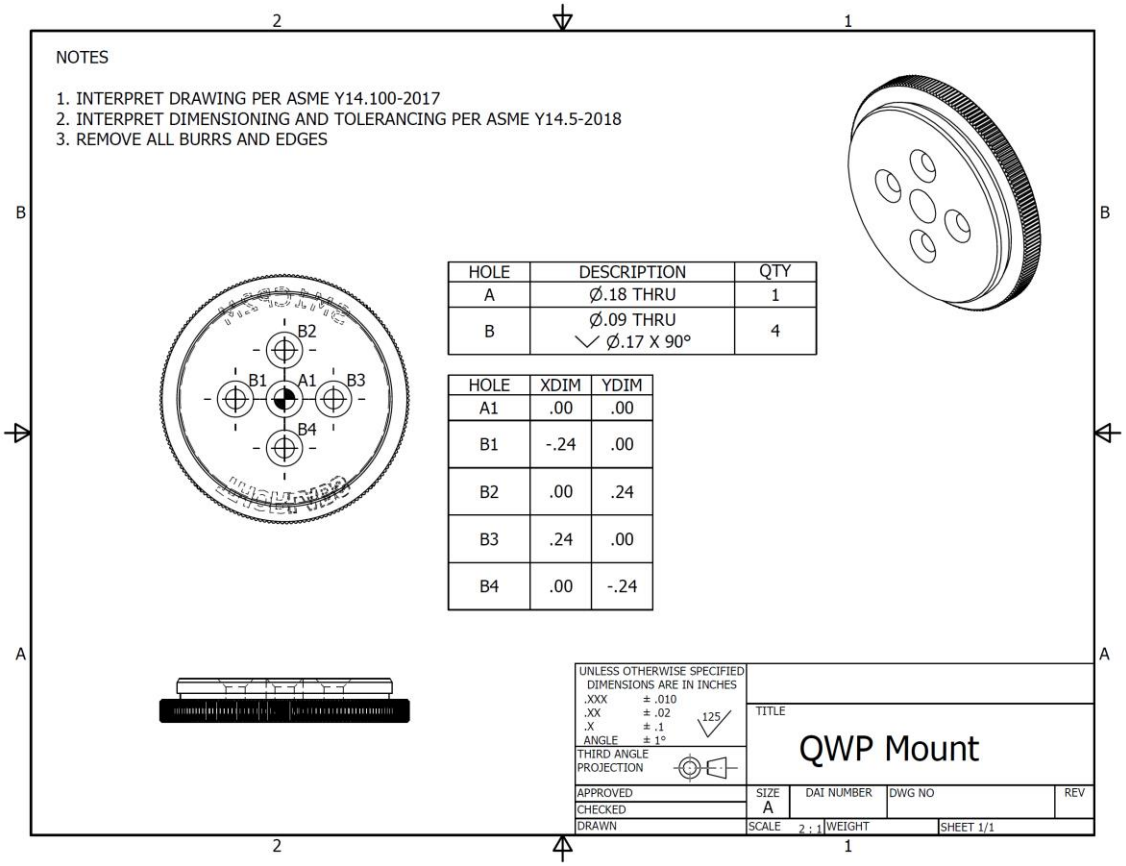
Timing Disk

The timing disk is secured between the custom QWP mount and the motor.

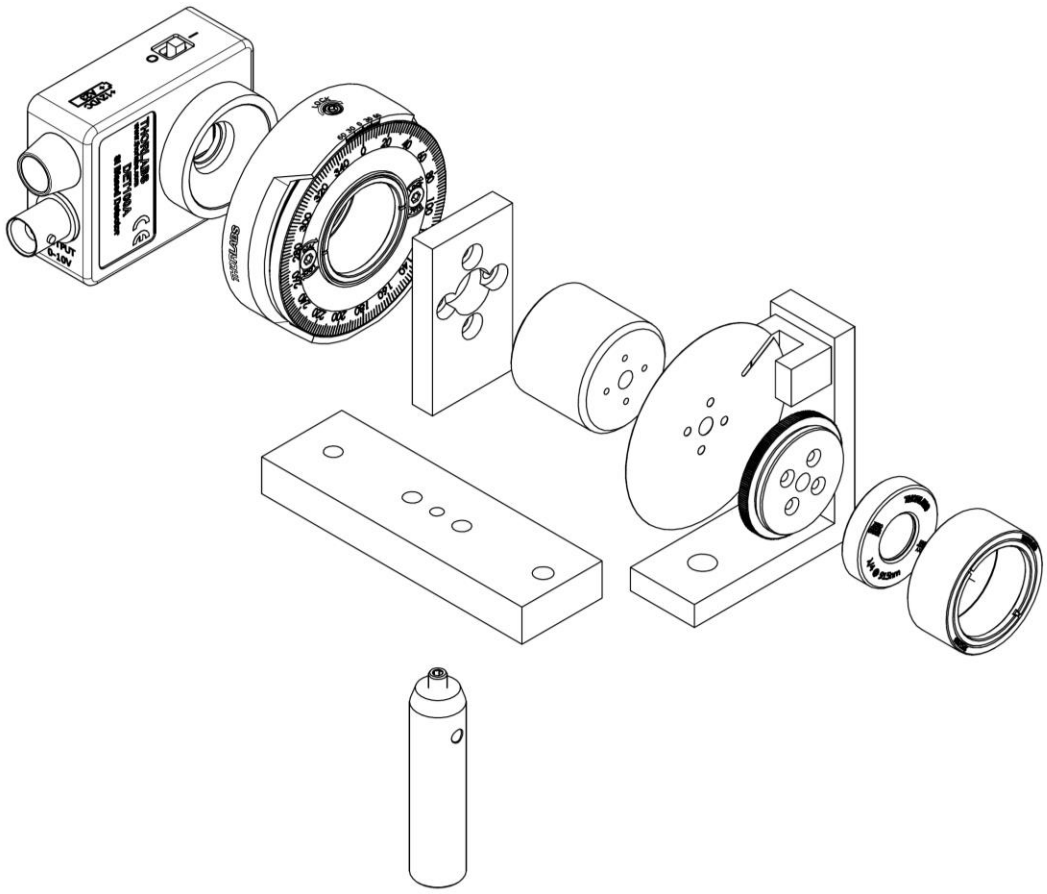


Custom Quarter-wave Plate Mount

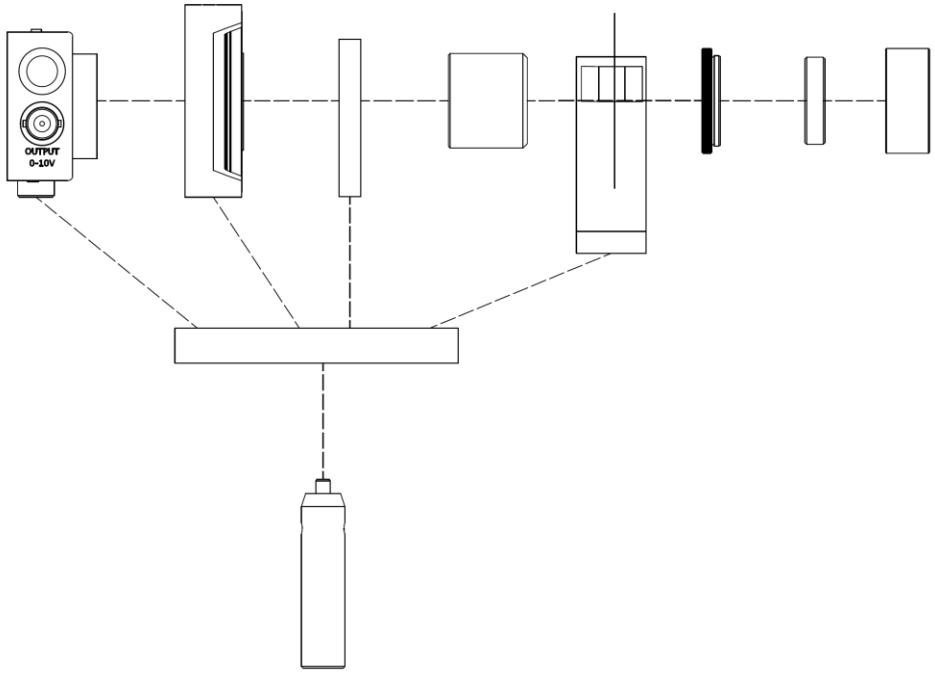
These modifications are made to the Thorlabs part SM1CP2M to match the hollow axle motor (Tiger GB2208-80) used in this work. The assembled mount is show in the exploded views on the following page. It uses an SM1 lens tube and retaining ring from Thorlabs.



Exploded Orthographic View of Device



Side View

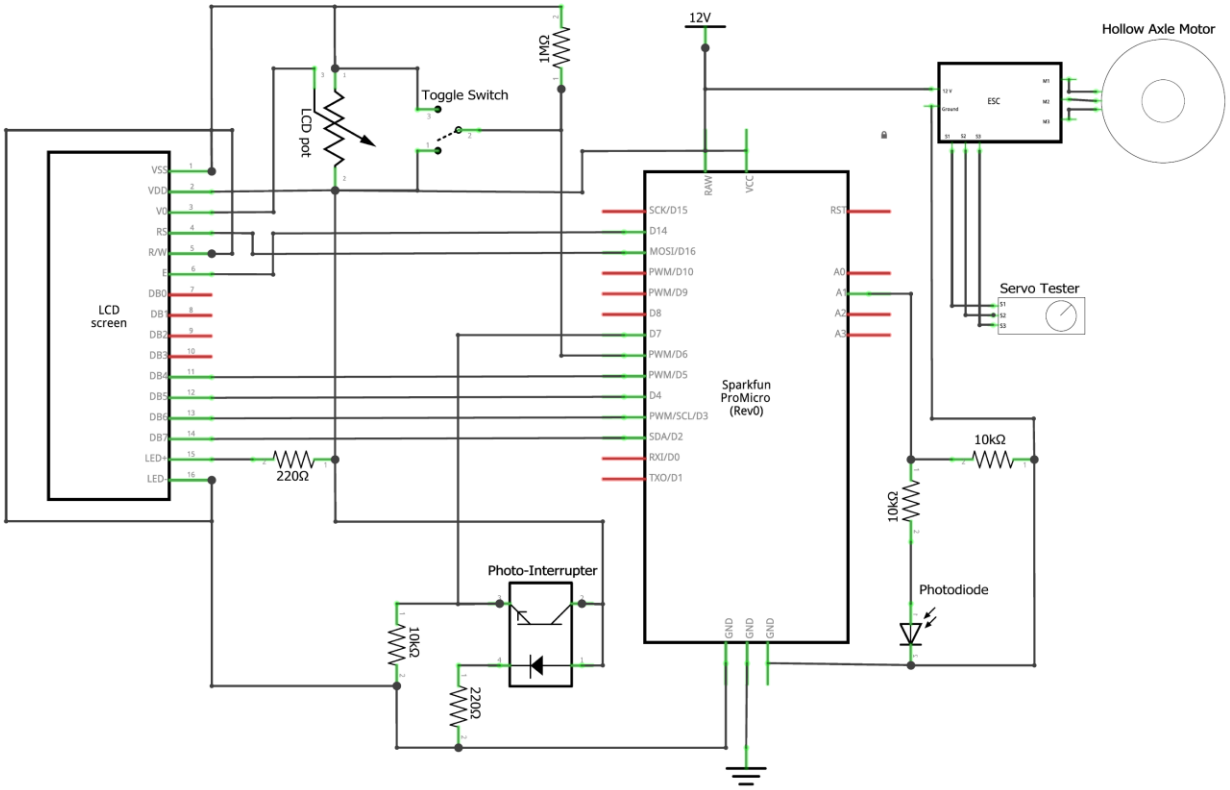


Calibrating the Polarimeter

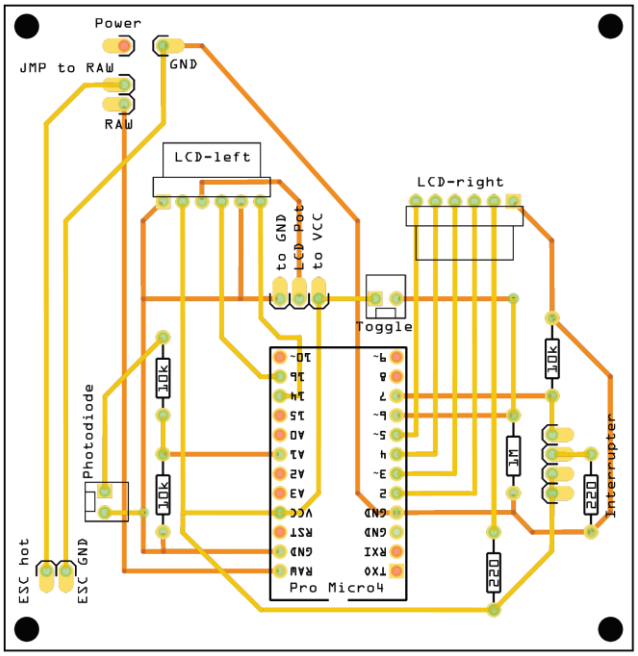
This calibration procedure is done entirely with X-polarized light from a calibrated polarizer upstream of the polarimeter.

1) Align the fast axis (FA) of the quarter-wave plate (QWP) to be roughly horizontally oriented (X) when the trigger will fire.
2) Set the toggle switch so that the Poincaré sphere angles are visible on the LCD screen. Physically move the position of the photo-interrupter up and down on its mount until the longitude angle is $(0 \pm 1)^{\circ}$.

Circuit Diagram and Printed Circuit Board (PCB)



The PCB was designed in the program Fritzing. This program can be used create PDF etch files to send to a PCB manufacturer.

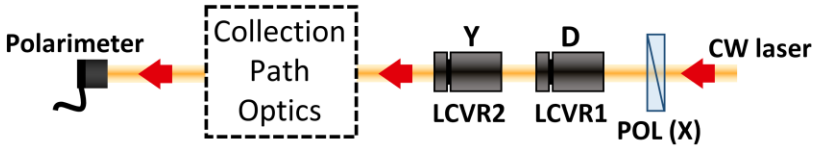


Arduino Code

The Arduino code can be found at the end of this document beginning on page 12.

LCVR Calibration

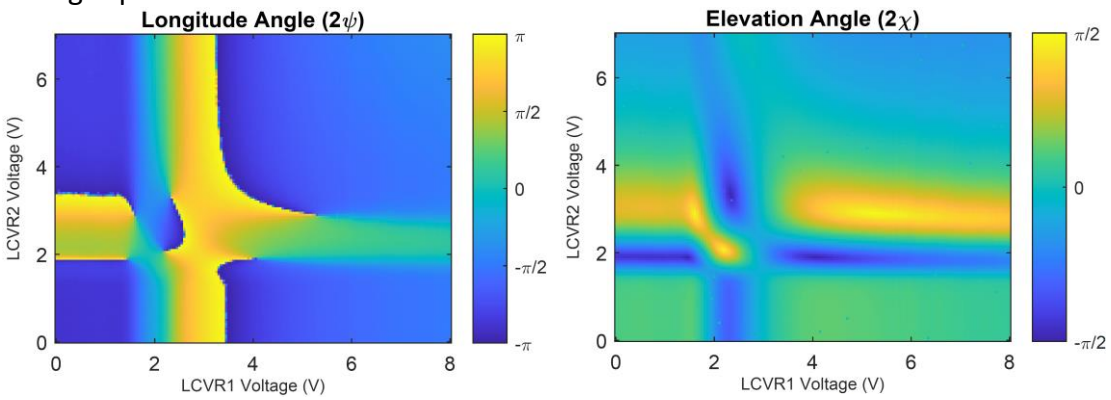
We use the computer interface enabled by the Arduino board to calibrate a pair of sequential LCVRs. We use the LCVRs to make polarization-sensitive measurements of fluorescence emitted by an optically excited material sample. In the light collection path between the sample and the LCVRs there are many optical elements like mirrors and lenses that have unintentional effects on the polarization. The result is that emission of a certain polarization ends up with a different polarization when it reaches the LCVRs. The purpose of this calibration procedure is to find settings of the LCVRs that enable measurements equivalent to measuring the polarization at the location of the sample. The optical setup for this calibration procedure is shown below.



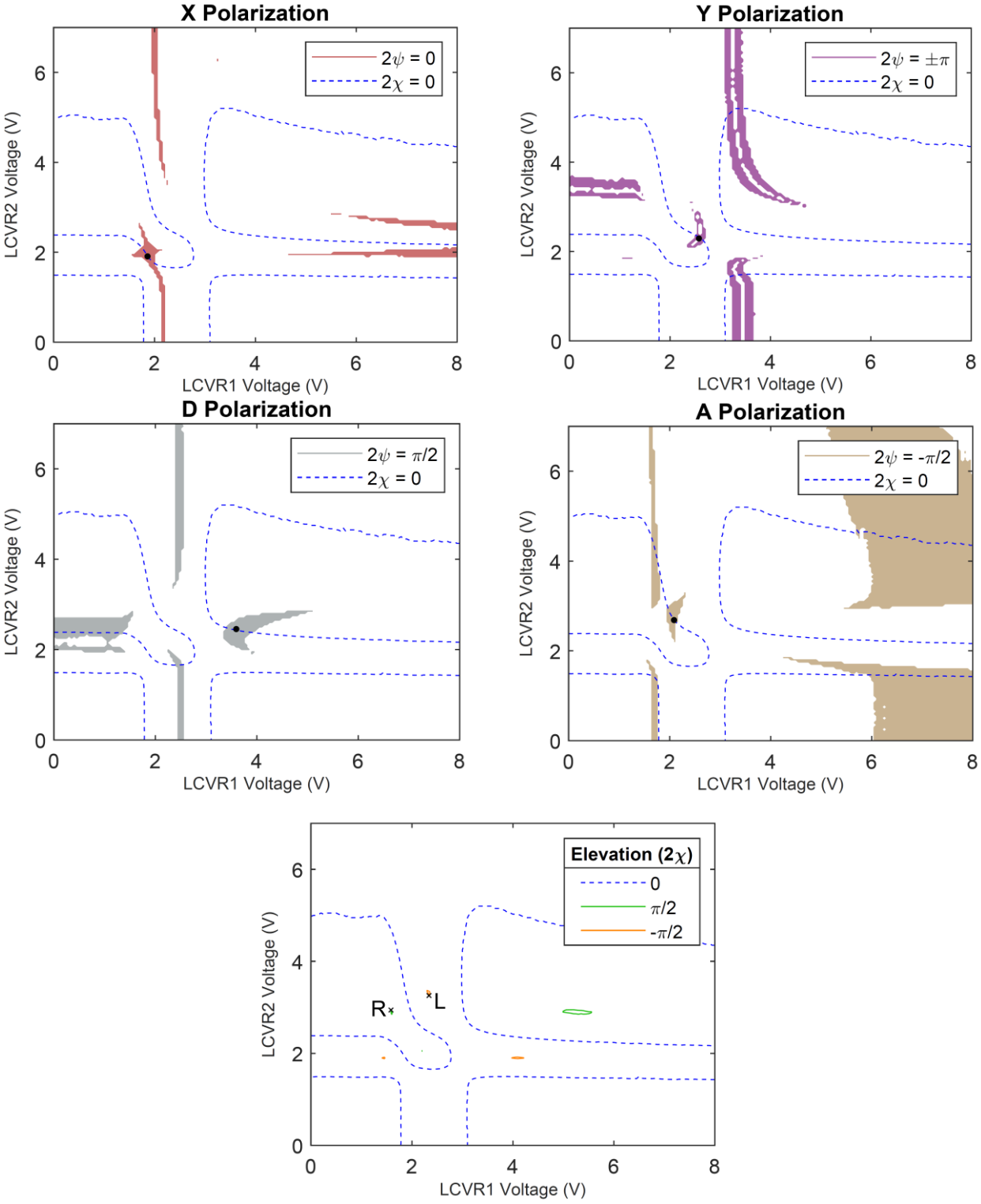
We direct a collimated beam backwards through our collection path to the polarimeter, which is placed directly in front of the sample. The reversibility of the effect of the polarization changes allows us to calibrate the LCVRs by setting the LCVR voltage values such that the polarimeter measures the desired polarization (X, Y, D, etc.). The desired polarizations can be specified by different combinations of only two parameters of the Stokes vector in the Poincaré sphere: the longitude angle 2ψ and the elevation angle 2χ. These values are shown in the table below.

Polarization	Longitude Angle (2ψ)	Elevation Angle (2ψ)
X	0	0
Y	$\pm\pi$	0
D	$\pi/2$	0
A	$-\pi/2$	0
R		$\pi/2$
L		$-\pi/2$

The circular polarizations (R, L) can be fully specified by the elevation angle alone, while the linear polarizations each require a certain value of longitude angle and 0 elevation angle. To begin, we take an automated scan of the entire voltage range of both LCVRs while recording the Stokes parameters via USB serial interface between the polarimeter and a computer. Shown below are intensity plots for both the longitude angle and the elevation angle for all LCVR voltage space.



To find the voltage values corresponding to each polarization, we plot five maps: one for each linear polarization and one for both circular polarizations (elevation contours only). For the longitude angle we plot regions around the specified values as the phase flip at $\pm\pi$ leads to erroneous contours. The chosen voltage values are marked on each plot as dots (crosses) for the linear (circular) cases. These values are chosen after a small bit of manual optimization.



Bill of Materials

Part Name	Vendor	Part Number	Price ($)
Optics			
Photodiode detector	Thorlabs	DET100A (DET100A2)*	174.22*
Polarizer (600-1100 nm)	Thorlabs	LPNIRE100-B	119.04
Broadband λ/4 plate (690-1200 nm)	Thorlabs	AQWP05M-980	864.61
		Optics Subtotal	1157.87
Opto-Mechanics			
Rotation Mount	Thorlabs	RSP1D	100.37
Custom λ/4 plate holder	Thorlabs	SM1M05 and SM1CP2M	13.32 and 19.13
Post	Thorlabs	TR3	5.58
Post holder	Thorlabs	PH3	8.52
		Opto-mechanics Subtotal	146.92
Motor Components			
Brushless Gimbal Hollow Motor	GetFPV	Tiger GB2208-80	34.99
Servo/ESC Tester	GetFPV	EMAX AC-0986	4.99
Tiger Motor Electronic Speed Controller (ESC)	GetFPV	S12A ESC 600 Hz 2-4s	16.99
		Motor Subtotal	56.97
Arduino			
Arduino Pro Micro	SparkFun	DEV-12640	17.95
Basic 16x2 Character LCD	SparkFun	LCD-00709	18.95
		Arduino Subtotal	36.90
Electronics			
Rotary Potentiometer – Linear (10k) x2	SparkFun	COM-09288	1.90
Momentary Button – Panel Mount	SparkFun	COM-11992	0.95
PCB Circuit Board	Aisler	Custom order	15.00
Plastic box to house circuit	DigiKey	SRRB66P06G16G-ND	21.30
Photo-Interrupter	DigiKey	365-1017-ND	1.78
Through Hole Resistor 10k x2	DigiKey	2197-293-10K-RC-ND	0.20
Through Hole Resistor 1M	DigiKey	2197-293-1M-RC-ND	0.10
Through Hole Resistor 220 x2	DigiKey	2197-293-220-RC-ND	0.20
DSUB cable	DigiKey	1175-1147-ND	4.90
12 V AC/DC Adapter	DigiKey	237-1454-ND	8.05
		Electronics Subtotal	54.38
		Total	1453.04

*The photodiode used in this work, DET100A, is now designated as obsolete by Thorlabs and has been replaced by part DET100A2. The price quoted here is for DET100A2.

Calibrating the Azimuth of an Optic to a Rotation Mount

Polarizer Calibration

Set up the optics in the following order:

1) Laser
2) Polarizer P_1 (does not need to be calibrated; oriented at approximately 45 degrees)
3) Polarizer P_2 to calibrate, held in a rotation mount (e.g., Thorlabs RSPD1)
4) Power meter or photodiode to measure the signal

To calibrate the polarizer P_2:

1) Record the intensity using the power meter or photodiode (I_1).
2) Rotate the polarizer P_2 $180°$ about the *vertical* axis (i.e., rotate it on its optical post).
3) Record the intensity again (I_2).
4) Rotate the polarizer P_2 in its mount (i.e., rotate it about the horizontal axis) so the intensity is halfway between I_1 and I_2.
5) Repeat steps 1-4 until I_1 and I_2 are equal.

Notes: The method will work for any choice of angle for the polarizer P_1 excepting $0°$ or $90°$ (X or Y), however it is most easily done with the angle at $\pm 45°$ (D or A).

Quarter-wave plate (QWP) Calibration

This method works best with two already calibrated polarizers. The above method can be used to calibrate these polarizers.

Set up the optics in the following order:

1) Laser
2) Calibrated polarizer P_1 in a rotation mount
3) QWP to calibrate, held in a rotation mount
4) Calibrated polarizer P_2 in a rotation mount
5) Power meter or photodiode to measure the signal

To Calibrate the QWP:

1) Set the angles of both P_1 and P_2 to be $45°$ (D).
2) Minimize the measured intensity by rotating the QWP azimuth.
3) Set the angle of P_2 to be $0°$ (X).
4) Record the intensity (I_1).
5) Rotate the QWP $180°$ about the *vertical* axis (rotate its optical post in the post holder).
6) Record the intensity again (I_2)
7) Rotate the QWP in its mount (i.e., rotate it about the horizontal axis) so the intensity is halfway between I_1 and I_2.
8) Repeat steps 4-7 until I_1 and I_2 are equal.

Notes: Depending on the initial orientation of the QWP, steps 4-7 may align the fast or slow axis to be $45°$; steps 1-2 ensure that this does not occur.

```
// Polarimeter_final.ino
// This code processes the voltage signals from the photodiode and
// the photo-interrupter with a fast Fourier transform (FFT), calculates
// the Stokes vectors, and outputs the results to the LCD screen.
//
// INPUTS
// photodiode signal:         adcPin = A1
// photo-interrupter signal:  interruptPin = 7
// toggle switch:             switchPin = 6
//
// The toggle switch controls whether the Stokes vector parameters
// or angles are shown on the LCD.
//
// OUTPUTS
// S0 Stokes:                 avg_Stokes[0]
// S1 Stokes:                 avg_Stokes[1]
// S2 Stokes:                 avg_Stokes[2]
// S3 Stokes:                 avg_Stokes[3]
// longitude angle (2*psi):   avg_longitude
// elevation angle (2*chi):   avg_elevation
//
// The outputs are displayed on the LCD screen and sent via Serial output
// to a connected computer.

/*
 * Library sources
 * AnalogReadFast.h  http://www.avdweb.nl/arduino/libraries/fast-10-bit-adc.html
 * TimerOne.h        https://www.pjrc.com/teensy/td_libs_TimerOne.html
 * arduinoFFT.h      https://github.com/kosme/arduinoFFT (loaded via Library manager)
 */

#include <avdweb_AnalogReadFast.h> // Allows using a faster analog read function
#include <TimerOne.h> // Allows using the built-in timers to throw interrupts
#include <LiquidCrystal.h> // Allows displaying to an LCD screen
#include "arduinoFFT.h" // Allows Fast Fourier Transform

// These factors are used to average the rotation period,
// where N is the number of cycles to average over. (Here N=10)
#define OLDFACTOR 0.9 // = (N - 1) / N
#define NEWFACTOR 0.1 // = 1/N

// For use in changing radians to degrees
#define DEGREES 57.2957795 // = 180 / pi

// Number of points to sample in one trigger cycle
// Should be a power of 2 that is >= 32
#define NPTS 64

// For output of the Fourier transform.
#define M_PI 3.14159265358979323846264338

// For the number of measurement cycles to average (N=10)
#define arraySetNumber 10


//****************** DECLARE VARIABLES *******************************

int interruptPin = 7; // Attach the trigger output to this pin.

volatile unsigned long prev_trigger = 0; // previous trigger, in microseconds
volatile unsigned long curr_trigger = 0; // current trigger, in microseconds
volatile unsigned long rotation_period = 0; // rotation period, in microseconds
```

```
// Flags to control program flow
volatile boolean startMeasuring = false; // Whether to start measuring the photodiode
signal
volatile boolean processing = true; // Whether the loop is processing the photodiode
signal data
                                    // Starts true so no measurement until setup() is
done
volatile boolean updateLCD = false; // Whether to update the LCD display

// For storing and FFT'ing data
volatile double vReal[NPTS]; // Array for actual photodiode data and real part of FFT
volatile double vImag[NPTS]; // Array for imag part of FFT
volatile int j = 0; // Keep track of which bin stores the photodiode voltage
const int adcPin = A1; // Analog input pin for photodiode signal

// For calculating the Stokes vector
double A = 0; // DC peak
double B = 0; // Imaginary part of 2*f peak
double C = 0; // Real part of 4*f peak
double D = 0; // Imaginary park of 4*f peak
float beta = 0.0524; // Compensate for imperfect QWP retardance
float delta = (M_PI/2) + beta; // Actual measured retardance
double stokes[] = {0, 0, 0, 0}; // Stokes vector

// Parameters calculated from the Stokes vector
double DOP = 0; // Degree of polarization
double longitude = 0; // Longitude of Stokes vector, 2*psi
double elevation = 0; // Elevation of Stokes vector from equator, 2*chi
double orientation = 0; // Angle of polarization ellipse major axis relative to X-axis
double ellipticityAngle = 0; // atan( minor axis / major axis )

// AVERAGING LAST 10 MEASUREMENTS
// Arrays for storing last 10 measurements
double set_stokes_0[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_stokes_1[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_stokes_2[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_stokes_3[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_DOP[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_longitude[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_elevation[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_orientation[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
double set_ellipticityAngle[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

// Index for counting assignment to arrays
int indexArr = 0;

// Sums for averages
double sum_stokes[] = {0, 0, 0, 0};
double sum_DOP = 0;
double sum_longitude = 0;
double sum_elevation = 0;
double sum_orientation = 0;
double sum_ellipticityAngle = 0;

// Average values
double avg_stokes[] = {0, 0, 0, 0};
double avg_DOP = 0;
double avg_longitude = 0;
double avg_elevation = 0;
double avg_orientation = 0;
double avg_ellipticityAngle = 0;

// Strings with which to display the results
```

```
String S0_str = String(stokes[0], 0); // Light intensity, 2 decimals
String S1_str = String(stokes[1], 2); // Rectilinear Stokes parameter
String S2_str = String(stokes[2], 2); // Diagonal Stokes parameter
String S3_str = String(stokes[3], 2); // Circular Stokes parameter
String orient_str = String(orientation * DEGREES, 0); // Angle of ellipse
String long_str = String(longitude * DEGREES, 0); // Longitude of Stokes vector
String elev_str = String(elevation * DEGREES, 0); // Elevation of Stokes vector
String message1_str = "Measuring period"; // Display on first line of LCD
String message2_str = "please wait";      // Display on second line of LCD

// For controlling which screen is shown with the toggle button
int switchPin = 6; // Toggle switch input
int switchVal = 0; // Value of toggle switch (HIGH or LOW)
int screen = 0; // Controls which screen is shown

// For controlling when the LCD updates
double Tstamp = 0; // Current program time, in ms
double updateFreq = 250; // How often to update the LCD, in ms
int updateNum = 1; // Keep track of updates
double T0 = 0; // Start of program time, in ms


// ******************** CREATE OBJECTS ************************

// Create FFT object
arduinoFFT FFT = arduinoFFT();

// Create LCD object
const int rs = 16, en = 14, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


// ******************** S E T U P ****************************

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD; will be "Measuring period" at first
  lcd.print( message1_str );
  lcd.setCursor(0, 1); // Move to beginning of second line
  lcd.print( message2_str );

  Serial.begin(9600);

  // Wipe the data in prep for the first measurement and FFT
  for (int k=0; k < NPTS; k++) {
    vReal[k] = 0;
    vImag[k] = 0;
  }

  // Set up the input pin for the photodiode signal
  pinMode( adcPin, INPUT );

  // Set up the input pin for the toggle switch to change LCD display
  pinMode( switchPin, INPUT );

  // Set up the interrupt pin as an input
  pinMode( interruptPin, INPUT );
  // Attach the external interrupt to the trigger_ISR
  attachInterrupt( digitalPinToInterrupt ( interruptPin ), trigger_ISR, RISING );

  // Wait a second so the trigger period is well characterized by trigger_ISR.
  delay( 1000 ); // Delay in ms
```

```
  // Initialize a timer to interrupt NPTS times each rotation period
  noInterrupts(); // Turn off interrupts
  Timer1.initialize( rotation_period / NPTS );
  interrupts(); // Turn interrupts back on again

  // Attach the timer interrupt to the sampling_ISR
  Timer1.attachInterrupt( sampling_ISR );

  // Set the processing flag so the next trigger will begin measurement
  processing = false;

}


// ********************** L O O P ***************************

void loop() {
  // The loop only processes the data if the 'processing' flag is set to TRUE
  if ( processing ) {

    // Output detected intensity signal to the Serial Plotter
    //for (int k=0; k < NPTS; k++)  {
      //Serial.println(vReal[k]);
    //}

    // P R O C E S S I N G
    // Process 'vReal' and 'vImag' to obtain the Stokes parameters
    FFT.Compute(vReal, vImag, NPTS, FFT_FORWARD); // Compute FFT in-place

    A =  2 * vReal[0] / NPTS; // DC peak
    B =  4 * vImag[2] / NPTS; // Imaginary part of 2*f peak
    C =  4 * vReal[4] / NPTS; // Real part of 4*f peak
    D =  4 * vImag[4] / NPTS; // Imaginary part of 4*f peak
    // S0: total light intensity
    stokes[0] = (A - (C / ( tan(delta/2)*tan(delta/2) ) )) / (sqrt(2*M_PI) );
    // S1: rectilinear Stokes parameter
    stokes[1] = C / ( stokes[0] * sin(delta/2)*sin(delta/2) * sqrt(2*M_PI) );
    // S2: diagonal Stokes parameter
    stokes[2] = D / ( stokes[0] * sin(delta/2)*sin(delta/2) * sqrt(2*M_PI) );
    // S3: circular Stokes parameter
    stokes[3] = B / ( stokes[0] * sin(delta) * sqrt(2 * M_PI) );
    DOP = sqrt(stokes[1]*stokes[1]+stokes[2]*stokes[2]+stokes[3]*stokes[3])/stokes[0];
    // Poincare sphere angles
    longitude = atan2( stokes[2], stokes[1] );
    elevation = atan2( stokes[3], sqrt( stokes[1]*stokes[1] + stokes[2]*stokes[2] ) );
    // Polarization ellipse angles
    orientation = 0.5 * longitude; // Angle relative to X-axis
    ellipticityAngle = 0.5 * elevation; // atan( minor axis / major axis )

    // A V E R A G I N G
    set_stokes_0[indexArr] = stokes[0];
    set_stokes_1[indexArr] = stokes[1];
    set_stokes_2[indexArr] = stokes[2];
    set_stokes_3[indexArr] = stokes[3];
    set_DOP[indexArr] = DOP;
    set_longitude[indexArr] = longitude;
    set_elevation[indexArr] = elevation;
    set_orientation[indexArr] = orientation;
    set_ellipticityAngle[indexArr] = ellipticityAngle;

    // Index control and set back to zero after the 10th measurement
    indexArr++; // increment index
```

```
if( indexArr == 10){
  indexArr = 0;
}

// Zero the sum arrays
sum_stokes[0] = 0;
sum_stokes[1] = 0;
sum_stokes[2] = 0;
sum_stokes[3] = 0;
sum_DOP = 0;
sum_longitude = 0;
sum_elevation = 0;
sum_orientation = 0;
sum_ellipticityAngle = 0;

// Calculate the averages
for (int k = 0; k <= (arraySetNumber-1); k++){
  sum_stokes[0] += set_stokes_0[k];
  sum_stokes[1] += set_stokes_1[k];
  sum_stokes[2] += set_stokes_2[k];
  sum_stokes[3] += set_stokes_3[k];
  sum_DOP += set_DOP[k];
  sum_longitude += set_longitude[k];
  sum_elevation += set_elevation[k];
  sum_orientation += set_orientation[k];
  sum_ellipticityAngle += set_ellipticityAngle[k];
}

avg_stokes[0] = sum_stokes[0] / arraySetNumber;
avg_stokes[1] = sum_stokes[1] / arraySetNumber;
avg_stokes[2] = sum_stokes[2] / arraySetNumber;
avg_stokes[3] = sum_stokes[3] / arraySetNumber;
avg_DOP = sum_DOP / arraySetNumber;
avg_longitude = sum_longitude / arraySetNumber;
avg_elevation = sum_elevation / arraySetNumber;
avg_orientation = sum_orientation / arraySetNumber;
avg_ellipticityAngle = sum_ellipticityAngle / arraySetNumber;

// Output Stokes parameters to Serial output for interfacing with a computer
Serial.print(avg_stokes[0]);
Serial.print(" ");
Serial.print(avg_stokes[1]);
Serial.print(" ");
Serial.print(avg_stokes[2]);
Serial.print(" ");
Serial.print(avg_stokes[3]);
Serial.print(" ");
Serial.print(avg_longitude);
Serial.print(" ");
Serial.print(avg_elevation);
Serial.print(" ");

// Determine whether to update the LCD screen
if ( Tstamp == 0 )  {
  T0 = millis();
}
Tstamp = millis();
if ( (Tstamp-T0) > updateNum*updateFreq )  {
  updateLCD = true;
}
else  {
  updateLCD = false;
}
```

```
// The loop only updates the LCD screen if the 'updateLCD' flag is set to TRUE
if ( updateLCD )  {
  // D I S P L A Y
  // Display the results of the data analysis
  // Prepare the String objects to be displayed
  // Stokes parameters
  S0_str = String(avg_stokes[0], 0); // Light intensity, 2 decimals
  S1_str = String(avg_stokes[1], 2); // Rectilinear Stokes parameter
  S2_str = String(avg_stokes[2], 2); // Diagonal Stokes parameter
  S3_str = String(avg_stokes[3], 2); // Circular Stokes parameter
  // Poincaré sphere angles
  orient_str = String(avg_orientation * DEGREES, 0); // Angle of ellipse
  long_str = String(avg_longitude * DEGREES, 1); // Longitude angle
  elev_str = String(avg_elevation * DEGREES, 1); // Elevation angle
  // Concatenate the Strings into the message to display
  message1_str = S1_str + " " + S2_str + " " + S3_str;
  message2_str = orient_str + " " + long_str + " " + elev_str;

  lcd.clear(); // Clear the LCD screen so no remnants from last message
  lcd.home(); // Set the cursor to beginning of first line

  // Detect the toggle switch to switch between LCD display types
  switchVal = digitalRead(switchPin);
  if ( switchVal == HIGH )  {
    screen = 2;
  }
  else  {
    screen = 1;
  }

  switch(screen){
    case 1:
    // Display the Stokes parameters
    lcd.print(S0_str);
    lcd.setCursor(6,0);
    lcd.print("Stokes:");
    lcd.setCursor(0,1); // Move cursor to beginning of second line
    lcd.print( message1_str );
    break;

    case 2:
    // Display the intensity, ellipse orientation, longitude, and elevation
    lcd.print(S0_str);
    lcd.setCursor(6,0);
    lcd.print("Angles:");
    lcd.setCursor(1,1); // Move cursor to beginning of second line
    lcd.print( message2_str );
    break;

    default:
    // Calibration screen shown at the beginning
    lcd.print("  Phase: ");
    lcd.setCursor(10,0);
    lcd.print(long_str);
    lcd.setCursor(0,1);
    lcd.print("Flip the switch to cont");
    break;
  }
  updateNum++; // Increment updateNum for next loop cycle
  updateLCD = false; // Don't update LCD until it is time again
}
```

```
    // C L E A N   U P
    // Wipe the imaginary part of the FFT in prep for the next loop
    for (int k=0; k < NPTS; k++) {
      vImag[k] = 0;
    }
    // Notify the ISRs that we're done processing the data
    processing = false;

    }

    // Update the timer interrupt period based on the current trigger period
    noInterrupts(); // Turn off interrupts
    Timer1.setPeriod( rotation_period / NPTS );
    interrupts(); // Turn interrupts back on again.
  }


// ************** INTERRUPT SERVICE ROUTINES ********************

void trigger_ISR() {
  curr_trigger = micros();
  // Microseconds since the Arduino board began running the current program.

  rotation_period = OLDFACTOR * rotation_period + NEWFACTOR * (curr_trigger -
prev_trigger);
  // Average rotation period over the last N triggers (N=10)
  // It takes N triggers to reach a stable value after start-up

  // Save current trigger time for the next call of trigger_ISR
  prev_trigger = curr_trigger;

  if ((!processing) && (!startMeasuring)) { // If the loop is done processing
    startMeasuring = true; // ...then start measuring the next cycle.

    // Measure the first sample of photodiode voltage and store it
    // j should have been set to 0 in sampling_ISR after the last measurement
    vReal[j] = analogReadFast( adcPin ); // 10-bit resolution (0-1023)
    j++; // increment the bin number for the next sample
    Timer1.restart(); // Restart the timer at the beginning of a new period
                      // This keeps the timer in phase with the trigger
  }
}

void sampling_ISR() { // Runs NPTS - 1 times per of rotation
  // Measure only when the trigger says go
  if ( startMeasuring ) {
    if ( j == NPTS - 1 ) { // If this is the last measurement in this cycle...
      Timer1.stop(); // Stop the timer, which prevents the ISR from running again
      startMeasuring = false; // Set the flag to prevent further measurement
      processing = true; // Start processing of data in the loop()
    }

    // Measure the photodiode voltage and store it
    vReal[j] = double( analogReadFast( adcPin ) ); // 10-bit resolution (0-1023)
    // Increment the bin number modulo the number of bins
    // This makes j=0 after the last measurement
    j = (j + 1) % NPTS;
  }
}

// end Polarimeter_final.ino
```