```
/*
* Supplementary information for “Design of Flexible Organic Photovoltaic Fibers.”
* This is version 7 of the SOL program that activates the big motor to ensure constant tension
* in the wire as it progresses along the spool.  Additional code was added to the program to ensure the
* bumpers are active during run mode, previously they were only active during the tare state
*/

/*
 * These constants control the motor
 */

//* Include the library */
#include "HCMotor.h"

//Motor 0
#define DIR_PIN_M0 8 //Connect to drive modules 'direction' input.
#define CLK_PIN_M0 9 //Connect to drive modules 'step' or 'CLK' input.

//Motor 1
#define DIR_PIN_M1 10 //Connect to drive modules 'direction' input.
#define CLK_PIN_M1 11 //Connect to drive modules 'step' or 'CLK' input.

//Motor 2
#define DIR_PIN_M2 6
#define CLK_PIN_M2 7

//Motor 3
#define DIR_PIN_M3 4
#define CLK_PIN_M3 5

// Set the analogue pin the potentiometer will be connected to. */
#define POT_PIN A0
#define POT_PIN_1 A1

// Set a dead area at the centre of the pot where it crosses from forward to reverse */
#define DEADZONE 20

// The analogue pin will return values between 0 and 1024 so divide this up between forward and reverse
#define POT_REV_MIN 0
#define POT_REV_MAX (512 - DEADZONE)
#define POT_FWD_MIN (512 + DEADZONE)
#define POT_FWD_MAX 1024

const int distancePin = 22;

int distanceCounter = 0;
int distanceState = 0;
int lastDistanceState = 0;

int initialSpeedState = 0;
int speedStatePrevious = 0;
double setpoint = 3.00;
double tensionBuffer = 0.30;
```

```
57          //* Create an instance of the library */
58          HCMotor HCMotor;
59
60
61          /*
62           * These constants correlate with the buttons that control start/stop, run, tare, tare direction and pause
63           */
64          int powerPin = 52;        // the number of the input pin
65          int tarePin = 50;
66          int tareDirectionPin = 48;
67          int runPin = 49;
68          int wireWindingPin = 47;
69          int spoolButtonPin = 53;
70          int leftMotorRearBumperPin = 22;
71          int leftMotorFrontBumperPin = 24;
72          int rightMotorRearBumperPin = 31;
73          int rightMotorFrontBumperPin = 33;
74          int spaceDebounce = 200; //debounce time
75
76
77          int greenLed = 34;
78          int yellowLedOne = 36;
79          int yellowLedTwo = 38;
80          int yellowLedThree = 51;
81          int redLed = 40;
82          int orangeLed = 42;
83
84
85          int powerState = LOW;     // the current powerState of the output pin
86          int powerReading;         // the current powerReading from the input pin
87          int powerPrevious = HIGH;    // the powerPrevious powerReading from the input pin
88
89          int tareState = LOW;
90          int tareReading;
91          int tarePrevious = HIGH;
92
93          int tareDirectionState = LOW;
94          int tareDirectionReading;
95          int tareDirectionPrevious = HIGH;
96
97          int runState = LOW;
98          int runReading;
99          int runPrevious = HIGH;
100
101         int wireWindingState = LOW;
102         int wireWindingReading;
103         int wireWindingPrevious = HIGH;
104
105         int spoolButtonState = LOW;
106         int spoolButtonReading;
107         int spoolButtonPrevious = HIGH;
108
109         /*
110          * These variables control the bumpers preventing the machine from binding up.
111          */
112
```

```
113         int leftMotorRearState = LOW;
114         int leftMotorRearReading;
115         int leftMotorRearPrevious = HIGH;
116
117         int leftMotorFrontState = LOW;
118         int leftMotorFrontReading;
119         int leftMotorFrontPrevious = HIGH;
120
121         int rightMotorRearState = LOW;
122         int rightMotorRearReading;
123         int rightMotorRearPrevious = HIGH;
124
125         int rightMotorFrontState = LOW;
126         int rightMotorFrontReading;
127         int rightMotorFrontPrevious = HIGH;
128
129         // the follow variables are long's because the powerTime, measured in miliseconds,
130         // will quickly become a bigger number than can be stored in an int.
131         long powerTime = 0;        // the last powerTime the output pin was toggled
132         long powerDebounce = 200;   // the powerDebounce powerTime, increase if the output flickers
133
134         long tareTime = 0;
135         long tareDebounce = 200;
136
137         long tareDirectionTime = 0;
138         long tareDirectionDebounce = 200;
139
140         long runTime = 0;
141         long runDebounce = 200;
142
143         long wireWindingTime = 0;
144         long wireWindingDebounce = 200;
145
146         long spoolButtonTime = 0;
147         long spoolButtonDebounce = 200;
148
149         long leftMotorRearTime = 0;
150         long leftMotorRearDebounce = spaceDebounce;
151
152         long leftMotorFrontTime = 0;
153         long leftMotorFrontDebounce = spaceDebounce;
154
155         long rightMotorRearTime = 0;
156         long rightMotorRearDebounce = spaceDebounce;
157
158         long rightMotorFrontTime = 0;
159         long rightMotorFrontDebounce = spaceDebounce;
160
161         /*
162          * These constants control the tension sensor
163          */
164         int  fsrPin = A2;
165         int fsrReading;
166         float fsrVoltage;
167
168         void setup()
```

```
{
  // Initialise the library */
  HCMotor.Init();
  Serial.begin(9600);

  // Attach motor 0 to digital pins 8 & 9. The first parameter specifies the
  // motor number, the second is the motor type, and the third and forth are the
  // digital pins that will control the motor
  HCMotor.attach(0, STEPPER, CLK_PIN_M0, DIR_PIN_M0);
  HCMotor.attach(1, STEPPER, CLK_PIN_M1, DIR_PIN_M1);
  HCMotor.attach(2, STEPPER, CLK_PIN_M2, DIR_PIN_M2);
  HCMotor.attach(3, STEPPER, CLK_PIN_M3, DIR_PIN_M3);

  /* Set the number of steps to continuous so the the motor is always turning while
   * not int he dead zone */
  HCMotor.Steps(0,CONTINUOUS);
  HCMotor.Steps(1,CONTINUOUS);
  HCMotor.Steps(2,CONTINUOUS);
  HCMotor.Steps(3,CONTINUOUS);


  pinMode(powerPin, INPUT);
  pinMode(tarePin, INPUT);
  pinMode(tareDirectionPin, INPUT);
  pinMode(runPin, INPUT);
  pinMode(wireWindingPin, INPUT);
  pinMode(spoolButtonPin, INPUT);

  pinMode(greenLed, OUTPUT);
  pinMode(yellowLedOne, OUTPUT);
  pinMode(yellowLedTwo, OUTPUT);
  pinMode(yellowLedThree, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(orangeLed, OUTPUT);


  pinMode(leftMotorRearBumperPin, INPUT);
  pinMode(leftMotorFrontBumperPin, INPUT);
  pinMode(rightMotorRearBumperPin, INPUT);
  pinMode(rightMotorFrontBumperPin, INPUT);
}

double  RPM_0, RPM_2, RPM_3, MRPM_2;
double RPM_Buffer = 0.01;
int rpmCounter;
int rpmCounterBuffer;
int testTimer;
double Speed, Speed_M0, Speed_M2;
double wire_dia = 100 * pow(10,-6);
double ACME_Thread = 8 * pow(10,-3);


void loop()
{
  /*
   * These parameters control the motor functionality
```

```
 */
//int Fast = 50;

int Pot, Pot2;

double PPS_0, PPS_2,PPS_3;
double SM;
static double wire_dia = 100 * pow(10,-6);
double ACME_Thread = 8 * pow(10,-3);
double distance = 10;
static double calcDistance = 0;
static int directionState = 0;
static int spoolPass = 0;
static int initialSpeed = 0;

/*
 * These parameters take measurements from the tension sensor
 */

fsrReading = analogRead(fsrPin);
fsrVoltage = fsrReading * (5.0 / 1023.0);

/*
 * These parameters control the bumper switches and the the control system switches
 */
powerReading = digitalRead(powerPin);
tareReading = digitalRead(tarePin);
tareDirectionReading = digitalRead(tareDirectionPin);
runReading = digitalRead(runPin);
wireWindingReading = digitalRead(wireWindingPin);
spoolButtonReading = digitalRead(spoolButtonPin);

leftMotorRearReading = digitalRead(leftMotorRearBumperPin);
leftMotorFrontReading = digitalRead(leftMotorFrontBumperPin);
rightMotorRearReading = digitalRead(rightMotorRearBumperPin);
rightMotorFrontReading = digitalRead(rightMotorFrontBumperPin);


// if the input just went from LOW and HIGH and we've waited long enough
// to ignore any noise on the circuit, toggle the output pin and remember
// the powerTime
if (powerReading == LOW && powerPrevious == HIGH && millis() - powerTime > powerDebounce)
{
  if (powerState == HIGH)
  {
    powerState = LOW;
    digitalWrite(greenLed, LOW);
  }
  else
  {
    powerState = HIGH;
    digitalWrite(greenLed, HIGH);
  }
  powerTime = millis();
}

```

```
  if(tareDirectionReading == LOW && tareDirectionPrevious == HIGH && millis() - tareDirectionTime >
tareDirectionDebounce)
  {
   if(tareDirectionState == HIGH)
   {
    tareDirectionState = LOW;
    digitalWrite(yellowLedTwo, LOW);

   }
   else
   {
    tareDirectionState = HIGH;
    digitalWrite(yellowLedTwo, HIGH);
   }
   tareDirectionTime = millis();
  }

  if(tareReading == LOW && tarePrevious == HIGH && millis() - tareTime > tareDebounce)
  {
   if(tareState == HIGH)
   {
    tareState = LOW;
    digitalWrite(yellowLedOne, LOW);
   }
   else
   {
    tareState = HIGH;
    digitalWrite(yellowLedOne, HIGH);
   }
   tareTime = millis();
  }

  if(runReading == LOW && runPrevious == HIGH && millis() - runTime > runDebounce)
  {
   if(runState == HIGH)
   {
    runState = LOW;
    digitalWrite(redLed, LOW);
   }
   else
   {
    runState = HIGH;
    digitalWrite(redLed, HIGH);
   }
   runTime = millis();
  }

  if(wireWindingReading == LOW && wireWindingPrevious == HIGH && millis() - wireWindingTime
> wireWindingDebounce)
  {
   if(wireWindingState == HIGH)
   {
    wireWindingState = LOW;
    digitalWrite(orangeLed, LOW);
   }
   else
```

```
337             {
338               wireWindingState = HIGH;
339               digitalWrite(orangeLed, HIGH);
340             }
341             wireWindingTime = millis();
342           }
343
344           if(spoolButtonReading == LOW && spoolButtonPrevious == HIGH && millis() - spoolButtonTime >
345         spoolButtonDebounce)
346           {
347             if(spoolButtonState == HIGH)
348             {
349               spoolButtonState = LOW;
350               digitalWrite(yellowLedThree, LOW);
351             }
352             else
353             {
354               spoolButtonState = HIGH;
355               digitalWrite(yellowLedThree, HIGH);
356             }
357             spoolButtonTime = millis();
358           }
359
360           if(leftMotorRearReading == LOW && leftMotorRearPrevious == HIGH && millis() -
361         leftMotorRearTime > leftMotorRearDebounce)
362           {
363             if(leftMotorRearState == HIGH)
364             {
365               leftMotorRearState = LOW;
366             }
367             else
368             {
369               leftMotorRearState = HIGH;
370             }
371             leftMotorRearTime = millis();
372           }
373
374             if(leftMotorFrontReading == LOW && leftMotorFrontPrevious == HIGH && millis() -
375         leftMotorFrontTime > leftMotorFrontDebounce)
376           {
377             if(leftMotorFrontState == HIGH)
378             {
379               leftMotorFrontState = LOW;
380             }
381             else
382             {
383               leftMotorFrontState = HIGH;
384             }
385             leftMotorFrontTime = millis();
386           }
387
388           if(rightMotorRearReading == LOW && rightMotorRearPrevious == HIGH && millis() -
389         rightMotorRearTime > rightMotorRearDebounce)
390           {
391             if(rightMotorRearState == HIGH)
392             {
```

```
393               rightMotorRearState = LOW;
394             }
395             else
396             {
397               rightMotorRearState = HIGH;
398             }
399             rightMotorRearTime = millis();
400           }
401
402           if(rightMotorFrontReading == LOW && rightMotorFrontPrevious == HIGH && millis() -
403         rightMotorFrontTime > rightMotorFrontDebounce)
404           {
405             if(rightMotorFrontState == HIGH)
406             {
407               rightMotorFrontState = LOW;
408             }
409             else
410             {
411               rightMotorFrontState = HIGH;
412             }
413             rightMotorFrontTime = millis();
414           }
415
416
417           if(powerState == HIGH)
418           {
419             /*
420              * Need to detect front bumper sensors
421              * Small motors turned off
422              * Tension sensor inactive
423              */
424             if((tareState == HIGH) && (runState == LOW))
425             {
426               if(tareDirectionState == HIGH)
427               {
428                 //directionState = 0;
429                 HCMotor.Direction(0, FORWARD);
430                 HCMotor.Direction(1, FORWARD);
431                 HCMotor.DutyCycle(0, 560.0); //Need to put a non variable unit here
432                 HCMotor.DutyCycle(1, 560.0); //Need to put a non variable unit here, same unit as above
433
434                 Serial.println("Motors move back");
435
436                 if((leftMotorRearState == HIGH) || (rightMotorRearState == HIGH))
437                 {
438                   digitalWrite(tarePin, LOW);
439                   digitalWrite(leftMotorRearBumperPin, LOW);
440                   digitalWrite(rightMotorRearBumperPin, LOW);
441                   tareState = LOW;
442                   tarePrevious = LOW;
443                   leftMotorRearState = LOW;
444                   leftMotorRearPrevious = LOW;
445                   rightMotorRearState = LOW;
446                   rightMotorRearPrevious = LOW;
447                   HCMotor.DutyCycle(0, 0);
448                   HCMotor.DutyCycle(1, 0);
```

```
449                 Serial.println("Motors moving towards the back stopped");
450                 Serial.println(tareState);
451               }
452
453
454               /*
455                * Need to detect rear bumper sensors
456                * Small motors turned off
457                * Tension sensor inactive
458                */
459             }
460             else
461             {
462               HCMotor.Direction(0, REVERSE);
463               HCMotor.Direction(1, REVERSE);
464               HCMotor.DutyCycle(0, 560.0); //Need to put a non variable unit here
465               HCMotor.DutyCycle(1, 560.0); //Need to put a non variable unit here, same unit as above
466
467               Serial.println("Motors move forward");
468
469               if((leftMotorFrontState == HIGH) || (rightMotorFrontState == HIGH))
470               {
471                 digitalWrite(tarePin, LOW);
472                 digitalWrite(leftMotorFrontBumperPin, LOW);
473                 digitalWrite(rightMotorFrontBumperPin, LOW);
474                 tareState = LOW;
475                 tarePrevious = LOW;
476                 leftMotorFrontState = LOW;
477                 leftMotorFrontPrevious = LOW;
478                 rightMotorFrontState = LOW;
479                 rightMotorFrontPrevious = LOW;
480
481                 HCMotor.DutyCycle(0, 0);
482                 HCMotor.DutyCycle(1, 0);
483                 Serial.println("Motors moving towards the front stopped");
484                 Serial.println(tareState);
485               }
486
487             }
488           }
489           else if(tareState == LOW && tarePrevious == HIGH)
490           {
491             HCMotor.DutyCycle(0, 0);
492             HCMotor.DutyCycle(1, 0);
493           }
494
495           if(runState == HIGH && tareState == LOW && wireWindingState == LOW)
496           {
497             /*
498              *  Run state goes here
499              *  Tension Sensor Active
500              *  Optical sensors goes here
501              *  Small Motors Active
502              *  Big motors moving at a rate that correlates with wire diameter
503              */
504
```

```
505
506             /* if tension sensor < 500g and detect the optical sensors for wire reaching the end of the spool
507              *
508              */
509
510             // The speed is now a user input
511             Speed = 1000;
512
513             //There should be a section here once the LCD user input is added to the program
514
515
516             // Motor 3 PPS and RPM calculations
517             PPS_3 = 1 / (Speed * 100 * pow(10,-6));
518             RPM_3 = ((1 / (Speed * 100 * pow(10,-6))) / 200) * 60;
519
520             // Calculate the RPM of the tracking motors Motor 0 and Motor 1
521             RPM_0 = RPM_3 * (wire_dia / ACME_Thread);
522             Speed_M0 = 60 / (200 * RPM_0 * 100 * pow(10,-6));
523             PPS_0 = 1 / (Speed_M0 * 100 * pow(10,-6));
524
525             /*
526              * This is the code that controls the main running portion of the program.
527              */
528             if(runState == HIGH)
529             {
530               /*
531                * Bumper controls and optical sensors
532                */
533               if(directionState == 0) // add optical sensor controls to this
534               {
535                 HCMotor.Direction(0, FORWARD);
536                 HCMotor.Direction(1, FORWARD);
537                 if((leftMotorRearState == HIGH) || (rightMotorRearState == HIGH))
538                 {
539                   directionState = 1;
540                 }
541               }
542               else if(directionState == 1) // add optical sensor controls to this
543               {
544                 HCMotor.Direction(0, REVERSE);
545                 HCMotor.Direction(1, REVERSE);
546                 if((leftMotorFrontState == HIGH) || (rightMotorFrontState == HIGH))
547                 {
548                   directionState = 0;
549                 }
550               }
551               HCMotor.Direction(2, FORWARD);
552               HCMotor.Direction(3, FORWARD);
553               if(initialSpeedState == speedStatePrevious)
554               {
555                 HCMotor.DutyCycle(3, Speed);
556                 HCMotor.DutyCycle(2, Speed); //Speed_M2
557                 HCMotor.DutyCycle(0, Speed_M0);
558                 HCMotor.DutyCycle(1, Speed_M0);
559                 Speed_M2 = Speed;
560                 PPS_2 = 1 / (Speed_M2 * 100 * pow(10,-6));
```

```
561                 RPM_2 = ((1.000 / (Speed_M2 * 100 * pow(10,-6)))/200) * 60;
562
563                 initialSpeedState = 1;
564                 if (RPM_3 < 3.0)
565                 {
566                   rpmCounterBuffer = 1;
567                 }
568                 else if (RPM_3 >= 3.0)
569                 {
570                   rpmCounterBuffer = 1;
571                 }
572                 rpmCounter = 0;
573                 testTimer = 0;
574               }
575               else if((fsrVoltage >= setpoint + tensionBuffer) || (fsrVoltage <= setpoint - tensionBuffer))
576               {
577                 if((fsrVoltage <= setpoint - tensionBuffer) && (rpmCounter == rpmCounterBuffer))
578                 {
579                   RPM_2 = RPM_2 - RPM_Buffer;
580                   rpmCounter = 0;
581                 }
582                 else if ((fsrVoltage >= setpoint + tensionBuffer) && (rpmCounter == rpmCounterBuffer) &&
583           (RPM_2 <= (setpoint + 0.04)))
584                 {
585                   RPM_2 = RPM_2 + RPM_Buffer;
586                   rpmCounter = 0;
587                 }
588                 else if ((fsrVoltage >= setpoint + tensionBuffer) && (rpmCounter == rpmCounterBuffer) &&
589           (RPM_2 >= (setpoint + 0.04)))
590                 {
591                   rpmCounter = 0;
592                 }
593                 rpmCounter = rpmCounter + 1;
594                 Speed_M2 = 60 / (200 * RPM_2 * 100 * pow(10,-6));
595                 HCMotor.DutyCycle(2, Speed_M2); //Speed_M2
596               }
597               else if ((leftMotorRearState == HIGH) || (rightMotorRearState == HIGH))
598               {
599                 HCMotor.Direction(0, REVERSE);
600                 HCMotor.Direction(1, REVERSE);
601                 digitalWrite(leftMotorRearBumperPin, LOW);
602                 digitalWrite(rightMotorRearBumperPin, LOW);
603                 leftMotorRearState = LOW;
604                 leftMotorRearPrevious = LOW;
605                 rightMotorRearState = LOW;
606                 rightMotorRearPrevious = LOW;
607                 //Serial.println("Cart moving reverse");
608               }
609               if((leftMotorFrontState == HIGH) || (rightMotorFrontState == HIGH))
610               {
611                 HCMotor.Direction(0, FORWARD);
612                 HCMotor.Direction(1, FORWARD);
613                 digitalWrite(leftMotorFrontBumperPin, LOW);
614                 digitalWrite(rightMotorFrontBumperPin, LOW);
615                 leftMotorFrontState = LOW;
616                 leftMotorFrontPrevious = LOW;
```

```
                rightMotorFrontState = LOW;
                rightMotorFrontPrevious = LOW;
                //Serial.println("Cart moving forward");
              }
              testTimer = testTimer + 1;
              Serial.print(testTimer);
              Serial.print("\t");
              Serial.print(RPM_2,4);
              Serial.print("\t");
              Serial.print(RPM_3,4);
              Serial.print("\t");
              Serial.print(fsrVoltage);

              if (testTimer >= 180000)
              {
                digitalWrite(runPin, LOW);
                runState = LOW;
                runPrevious = LOW;
                HCMotor.DutyCycle(2, 0);
                HCMotor.DutyCycle(3, 0);
                Serial.println("in here");
              }
              else if (Speed_M2 == 0)
              {
                Serial.println("never stood a chance");
                digitalWrite(runPin, LOW);
                runState = LOW;
                runPrevious = LOW;
                HCMotor.DutyCycle(2, 0);
                HCMotor.DutyCycle(3, 0);
              }
            }
          }
          else if(runPrevious == HIGH && runState == LOW)
          {
            HCMotor.DutyCycle(2, 0);
            HCMotor.DutyCycle(3, 0);
            HCMotor.DutyCycle(0, 0);
            HCMotor.DutyCycle(1, 0);
          }
           /*
            * Wire tightening state goes here
            * Small Motor control
            * Tension sensor is active
            */
          if(wireWindingState == HIGH && runState == LOW && tareState == LOW)//this needs to have a
        wiretightening state added to low to if necessary
          {
            if(fsrVoltage <= setpoint)
            {
              HCMotor.Direction(2, FORWARD);
              HCMotor.Direction(3, FORWARD);
              HCMotor.DutyCycle(2, 0);
              HCMotor.DutyCycle(3, 10000);

            }
```

```
673             else
674             {
675               digitalWrite(wireWindingPin, LOW);
676               wireWindingState = LOW;
677               wireWindingPrevious = LOW;
678               digitalWrite(orangeLed, LOW);
679               HCMotor.DutyCycle(2, 0);
680               HCMotor.DutyCycle(3, 0);
681               Serial.println("ready");
682             }
683             //Serial.println(fsrVoltage);
684           }
685
686           if(spoolButtonState == HIGH && wireWindingState == LOW && runState == LOW && tareState ==
687         LOW)//this needs to have a wiretightening state added to low to if necessary
688           {
689             //Serial.println(HCMotor.clockCounter());
690
691             /*
692              * This snippet allows the wire to be spooled twice along the length of the spool creating a buffer layer
693              */
694             if (spoolPass < 100)
695             {
696               Speed = 100.0;
697               if (spoolPass == 0 && initialSpeed == 0)
698               {
699                 HCMotor.Direction(2, REVERSE);
700                 HCMotor.Direction(3, REVERSE);
701                 HCMotor.Direction(1, REVERSE);
702                 HCMotor.DutyCycle(2, Speed);
703                 HCMotor.DutyCycle(3, Speed);
704                 HCMotor.DutyCycle(1, Speed);
705                 initialSpeed = 1;
706                 //spoolPass++;
707                 Serial.println("First Phase");
708               }
709               else if (HCMotor.clockCounter(1) >= 3176 && (spoolPass == 0 ))
710               {
711                 HCMotor.Direction(1, FORWARD);
712                 spoolPass++;
713                 Serial.println("Second Phase");
714                 HCMotor.clearClockCounter(1);
715               }
716               else if  (HCMotor.clockCounter(1) >= 3176 && (((spoolPass % 2) > 0) && (spoolPass < 6)))
717               {
718                 HCMotor.Direction(1, REVERSE);
719                 HCMotor.DutyCycle(1, Speed * pow(2,spoolPass));
720                 spoolPass++;
721                 Serial.println(spoolPass);
722                 HCMotor.clearClockCounter(1);
723                 Serial.println("Third Phase");
724
725               }
726               else if (HCMotor.clockCounter(1) >= 3176 && (((spoolPass % 2) == 0) && (spoolPass < 6)))
727               {
728                 HCMotor.Direction(1, FORWARD);
```

```
729                 HCMotor.DutyCycle(1, Speed * pow(2,spoolPass));
730                 spoolPass++;
731                 Serial.println(spoolPass);
732                 HCMotor.clearClockCounter(1);
733                 Serial.println("Fourth Phase");
734               }
735               else if (HCMotor.clockCounter(1) >= 3176 && (spoolPass >= 6) && ((spoolPass % 2) > 0))
736               {
737                 HCMotor.Direction(1, REVERSE);
738                 Speed_M0 = bigMotorSpeed(Speed);
739                 HCMotor.DutyCycle(1, Speed_M0);
740                 spoolPass++;
741                 HCMotor.clearClockCounter(1);
742                 //Serial.println("Fifth Phase");
743               }
744               else if (HCMotor.clockCounter(1) >= 3176 && (spoolPass >= 6) && ((spoolPass % 2) == 0))
745               {
746                 HCMotor.Direction(1, FORWARD);
747                 spoolPass++;
748                 HCMotor.clearClockCounter(1);
749                 //Serial.println("Sixth Phase");
750               }
751               Serial.println(fsrVoltage);
752             }
753
754
755             }
756             else if(spoolButtonPrevious == HIGH && spoolButtonState == LOW)
757             {
758               digitalWrite(spoolButtonPin, LOW);
759               spoolButtonState = LOW;
760               spoolButtonPrevious = LOW;
761               digitalWrite(yellowLedThree, LOW);
762               HCMotor.DutyCycle(2, 0);
763               HCMotor.DutyCycle(3, 0);
764               HCMotor.DutyCycle(1, 0);
765             }
766           }
767           else
768           {
769             HCMotor.DutyCycle(2, 0);
770             HCMotor.DutyCycle(3, 0);
771             HCMotor.DutyCycle(0, 0);
772             HCMotor.DutyCycle(1, 0);
773           }
774           powerPrevious = powerReading;
775           tarePrevious = tareReading;
776           tareDirectionPrevious = tareDirectionReading;
777           runPrevious = runReading;
778           wireWindingPrevious = wireWindingReading;
779           spoolButtonPrevious = spoolButtonReading;
780
781           leftMotorRearPrevious = leftMotorRearReading;
782           leftMotorFrontPrevious = leftMotorFrontReading;
783           rightMotorRearPrevious = rightMotorRearReading;
784           rightMotorFrontPrevious = rightMotorFrontReading;
```

```
785         }
786
787         double bigMotorSpeed(double velocitySmallMotor)
788         {
789           double RPM_M0 = (60 / (velocitySmallMotor * 200 * (100 * pow(10,-6))) *          (wire_dia /
790         ACME_Thread));
791
792           Speed_M0 = 60 / (200 * RPM_M0 * 100 * pow(10,-6));
793           return Speed_M0;
794         }
```