

BOBINADORA DE MICROINDUCTORES

DR. MARTÍN SARAVIA - ING. CLAUDIO GATTI

CONICET



Grupo de Investigación en Multifísica Aplicada - UTN FRBB

Noviembre de 2017 – Version 0.0

HARDWARE

1.1 INTRODUCCIÓN

La máquina descripta en el presente informe permite bobinar automáticamente inductores de dimensiones reducidas. El software y el hardware fueron desarrollados íntegramente por los autores, en el Grupo de Investigación de Multifísica Aplicada (GIMAP) de la Facultad Regional Bahía Blanca de la Universidad Tecnológica Nacional.

La máquina es automática; esto es: los parámetros de bobinado (longitud, diámetro de alambre, número de vueltas, etc) son seteados en un software, y luego este se encarga de controlar la ejecución del proceso. El movimiento es controlado por un par de motores paso a paso, los cuales son controlados por una placa Arduino Mega. Esta, aloja una placa RAMPS que contiene los drivers de los motores. El software de control está basado en un algoritmo que utiliza un enfoque pseudo-temporal para el conteo de ciclos de bobinado.

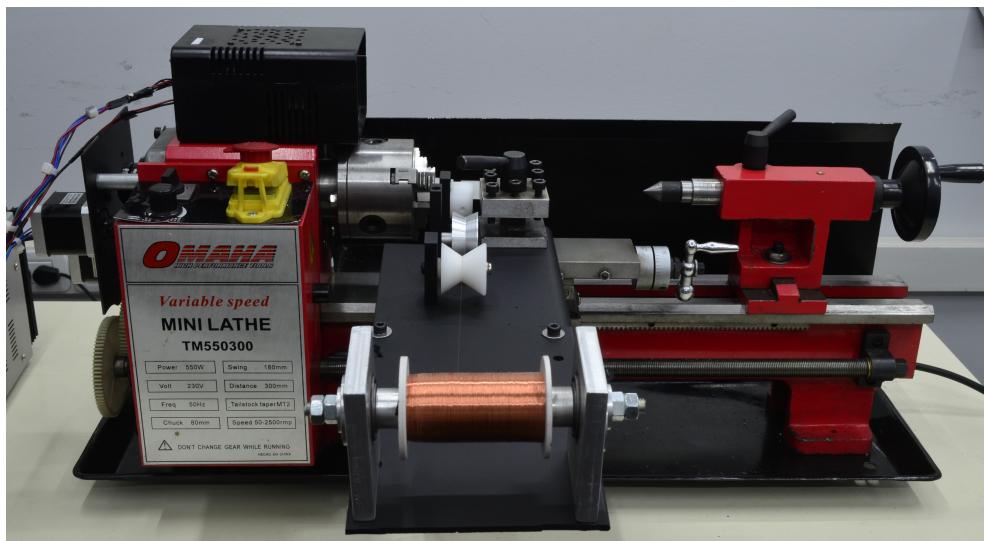


Figure 1.1: Bobinadora de microinductores.

El principal objetivo de este desarrollo es lograr la miniaturización de inductores utilizados en recolectores de energía electromagnéticos (Figura 1.2); este aspecto es fundamental para lograr prototipos funcionales. Los primeros prototipos del recolector eran bobinados con conductores de calibre AWG 30 (diámetro 0.25 mm); no obstante, de acuerdo a estimaciones computacionales, la corriente máxima que circula por un recolector de energía tamaño AA, el diámetro mínimo de conductor es 0.076 mm (AWG 42), es decir: 3 veces menor, ver Figura 1.3. Este diámetro es aproximadamente igual al diámetro de un cabello humano. Esta reducción de diámetro permite aumentar dramáticamente el número de vueltas en las bobinas, lo cual incrementa el voltaje generado proporcionalmente.



Figure 1.2: Microbobinas.



Figure 1.3: Recolectores de energía. Izquierda: Prototipo inicial; medio: prototipo con microbobinas; derecha: pila AA.

Con el objetivo de ahorrar costos, se decidió acoplar la máquina a un torno convencional; así, los motores paso a paso se acoplan al tren de engranajes del torno controlan el movimiento de la máquina. El conductor que forma la bobina es alimentado a través de una carretel que contiene un eje con fricción ajustable, sobre el cual se monta el carretel de alambre. El alambre atraviesa una serie de poleas cónicas que direccionan el alambre y el ajuste de un conjunto de rodamientos axiales permite controlar la tensión de bobinado.

La fabricación de microbobinas requirió del desarrollo de un proceso que permita obtener bobinas autocontenidas y desmontables. Esto es necesario para modificar su ubicación axial y así estudiar diferentes posibilidades de diseño de recolectores de energía. Luego de ensayar al menos cuatro procesos de fabricación, la metodología de cera perdida dio los mejores resultados. Esta metodología se basa en ejecutar el bobinado sobre una barra cilíndrica parafina aditivada con ácido esteárico, la cual es previamente mecanizada con las dimensiones del cuerpo de poliacetal del recolector y posteriormente recubierta con una delgada película de resina epoxy. Una vez completado el bobinado, la bobina se recubre con resina epoxi, y luego de 2 hs, se la coloca en horno a 150 grados centígrados para eliminar la parafina por fusión. De esta manera se logra una bobina de alta densidad y con excelente dimensionado.

Gran parte del tiempo de desarrollo de la máquina ha sido dedicado al diseño del sistema de control. Este está formado por una placa Arduino Mega acoplada a una placa RAMPS que aloja los drivers de los motores paso a paso. Para lograr un control efectivo de los parámetros de bobinado, se desarrolló un algoritmo que utiliza un enfoque pseudo-temporal para el conteo de ciclos de bobinado.

1.2 TENSIONADOR

Para lograr un bobinado con buena precisión es necesario controlar la deformación inicial del alambre. Para mantenerlo recto, es necesario aplicar una tensión controlada. Para ello, se diseñó un dispositivo basado en fricción por rodadura, para el cual, ajustando la tensión axial en un eje pretensado es posible controlar la fuerza tangencial en el carrete, ver Figura 1.4.

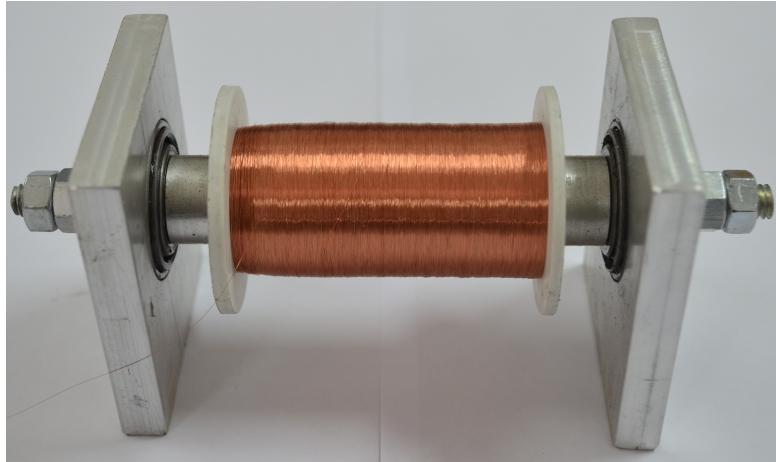


Figure 1.4: Tensionador con carrete.

El ajuste se realiza por medio de un sistema de rodamientos cónicos montados sobre un eje hueco; la tensión axial en los rodamientos es controlada por medio de una varilla roscada que atraviesa el eje y permite ajustar la carga axial, ver Figura 1.5.

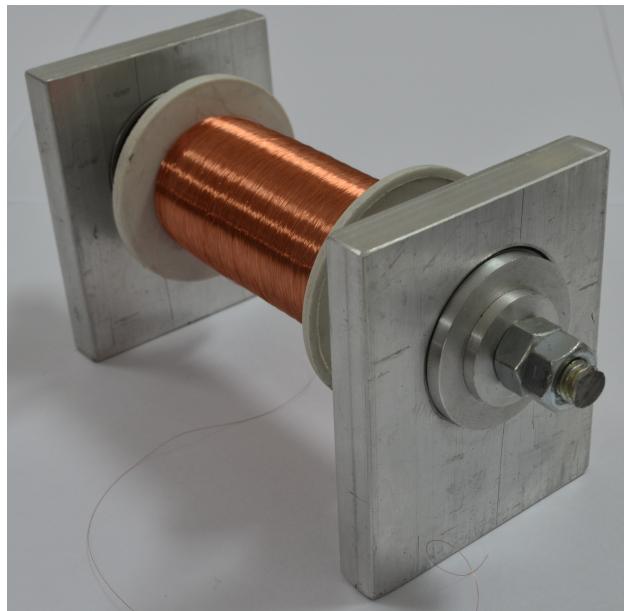


Figure 1.5: Eje del tensionador.

1.3 DIRECCIONADOR

El direccionamiento del conductor está controlado por un tren de dos poleas cónicas y un paso calibrado, ver Figura 1.6. El tren de poleas cumple la función de alinear el conductor; así, al llegar a la bobina, éste se encuentra perpendicular al eje de parafina sobre el cual se enrolla, independientemente de la posición transversal que ocupe el carretel. El paso calibrado evita los corrimientos en la dirección axial de la bobina originados por el la carrera del carro.



Figure 1.6: Direccionador.

1.4 MOTORES Y TREN DE ENGRANAJES

El movimiento de la máquina está controlado por dos motores paso a paso NEMA 17 (1.8° , 1.2A) de 4.8 kgcm de torque. Los motores se acoplan al tren de engranajes del torno por medio de un par de engranajes, ver Figuras 1.7 y 1.8. La reducción de las velocidades angulares es de 1.5 y 8 para los movimientos de bobinado (enrollado) y avance (carrera), respectivamente.



Figure 1.7: Motores paso a paso.

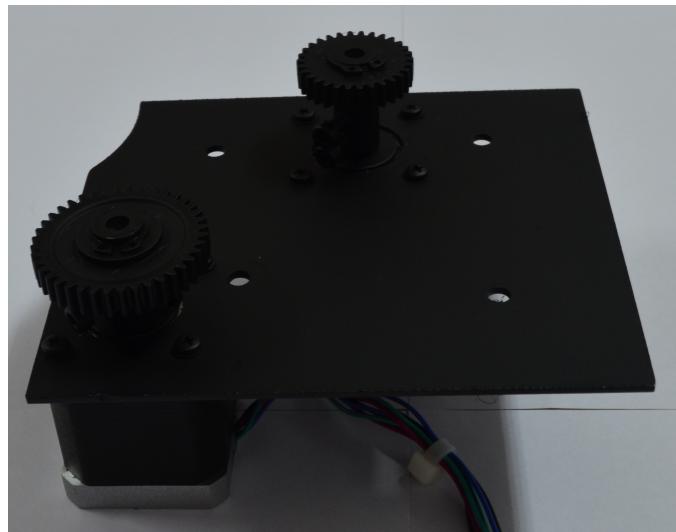


Figure 1.8: Transmisión.

1.5 CONTROLADOR

El controlador está formado por una placa Arduino Mega sobre la cual se acopla una placa RAMPS 1.4, ver Figura 1.9. La RAMPS aloja un par de drivers DRV8825 (originalmente A4988) que controlan los pines de paso y dirección de los motores paso a paso. La caja que aloja al controlador posee una llave de encendido y un pulsador de reinicio. El software es cargado al Arduino por medio de un puerto USB, ver Figura 1.10.

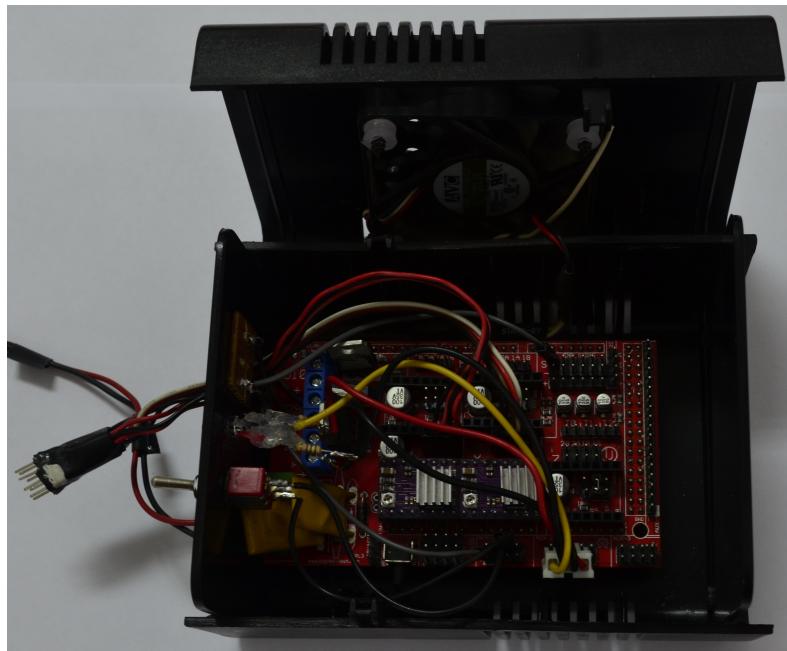


Figure 1.9: Controlador.

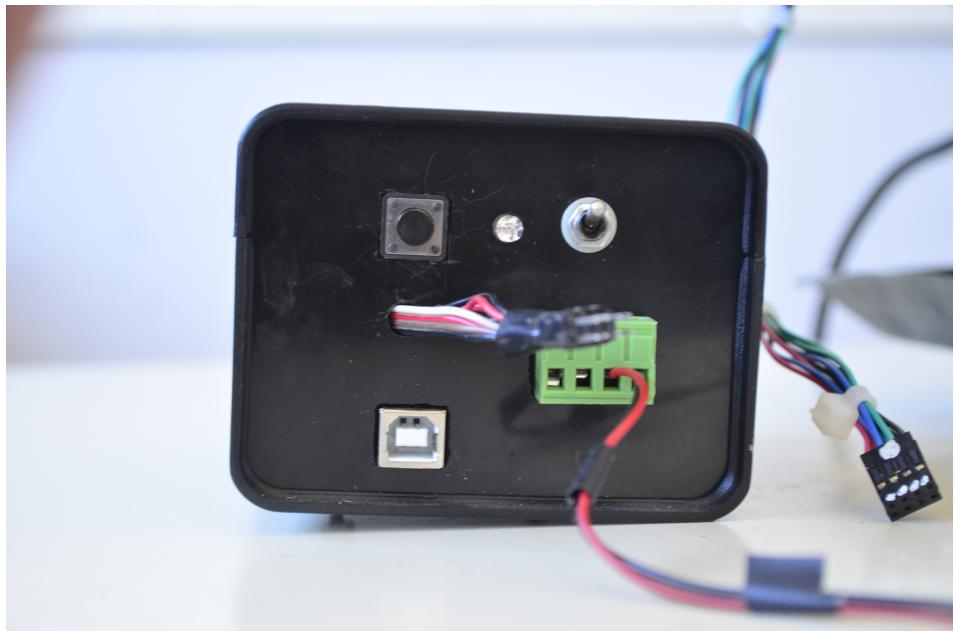


Figure 1.10: Comando del controlador.

SOFTWARE

El software que controla la bobinadora fue desarrollado en la plataforma Arduino. Como se ha mencionado, el esquema de control está basado en enfoque pseudo-temporal de lazo abierto; la siguiente sección muestra el código completo.

2.1 CÓDIGO

```
// Bobinadora 0.1

//-----
//          SETUP DE BOBINADO
//-----

unsigned int strokes = 10      ; // Total strokes
float wps      = 1.0          ; // Speed (revolutions per second)
float coild    = 11.0          ; // Internal coil diameter
float coilth   = 14.0          ; // Coil width
float wdiam    = 0.0635        ; // Wire diameter
float micro    = 4.0           ; // Micro Stepping
float tfactor  = 1.5           ; // % of increment in wire diameter
                                // to account for imperfections
float Lbacklash = 0.4          ; // Left Backlash mm
float Rbacklash = 0.4          ; // Right Backlash mm
boolean crev = false          ; // Reverse motor flag (false=
                                // starts to the right, true=starts to the left)
//-----


// Setup de control de motores
byte StartPin = 1  ;
byte KillPin  = 3  ;

byte WstepPin = 54 ;
byte WdirPin  = 55 ;
byte WenaPin  = 38 ;
byte CstepPin = 60 ;
byte CdirPin  = 61 ;
byte CenaPin  = 56 ;
byte WledPin   = 13 ;
byte FanPin   = 9  ;

byte RedLedPin = 44;
byte GreLedPin = 64;
byte BluLedPin = 59;

// Setup de control de tiempo
unsigned long tnow ;
```

```

unsigned long t0w ;
unsigned long t0c ;
unsigned long dtw ; // Tiempo entre pasos del motor
    bobinador (micros)
unsigned long dtc ; // Tiempo entre pasos del motor de
    carrera (micros)
unsigned long dtwr ; // Tiempo entre pasos del motor
    bobinador (micros) en el arranque
unsigned long dtcr ; // Tiempo entre pasos del motor de
    carrera (micros) en el arranque
unsigned long dtw0 ; // Tiempo entre pasos del motor
    bobinador (micros) en el arranque
unsigned long dtc0 ; // Tiempo entre pasos del motor de
    carrera (micros) en el arranque
float frac;
float spw;
float spc;
unsigned int backsteps ; // backlash steps
float cerror = 0.0 ;
unsigned int wloops ;
unsigned int wcount = 0 ; // loop count
unsigned int iw0 = 0 ; // loop count
unsigned int wstep = 0 ; // wounding step count
unsigned int ccount = 0 ; // stroke count
unsigned int cstep = 0 ; // stroking step count
boolean goloop = true ;
int temp = 0 ; // Temporary variable
int cota ;
unsigned long microdelay = 50;

unsigned long TotalTime;
unsigned long dtbk; //dt for the backlash velocity
unsigned int acbk; // Cutoff steps for backlash ramp

boolean PauseFlag = false;
//-----
//                      INIT
//-----
void setup() {

    Pre(); // Pre (jajaja)

    // Wait for Start Button pressed
    while(int Start = digitalRead(StartPin) == HIGH){
        BlinkLed('R');
    }

    // Let's go...
    TurnOnLed('G');
    digitalWrite(FanPin, HIGH); // Start the Fan
    digitalWrite(WenaPin, LOW); // Enable wounding motor
    digitalWrite(CenaPin, LOW); // Enable stroking motor
    digitalWrite(WdirPin,HIGH); // Set wounding dir
}

```

```

if (crev == false){
    digitalWrite(CdirPin,HIGH);} // Set stroking dir
if (crev == true){
    digitalWrite(CdirPin,LOW);} // Set stroking dir

digitalWrite(WledPin, LOW); // Turn on some LED (which?)

// Calculate constants
wdiam = wdiam * tfactor; // Effective wire
diameter = wire diameter + imperfection errors
spw = (micro * 200) * 1.5; // Steps per loop
    (1.5 is the reduction factor
spc = (micro * 200) * 8 * ( wdiam / 1.5 ) ; // Steps per
    unit stroke (wire diameter) (8 is the reduction factor,
    1.5 is the pitch of the screw)
dtwr = 1000000 * (1.0 / (wps * spw) ); // Time between
    wounder steps (microseconds)
dtcr = 1000000 * (1.0 / (wps * spc) ); // Time between
    stroker steps (microseconds)
// dtw0 = 4.0 * dtwr; // Time between
    wounder steps at start(microseconds)
// dtc0 = 4.0 * dtcr; // Time between
    stroker steps at start (microseconds)
wloops = int(coilth / wdiam );

// Start looping ....
t0w = micros() ; // Set the start time
t0c = t0w ;
cota = 450;
for (int ic = 1; ic <= strokes; ic++){
    iw0 = 1; // Restart the initial loop count
    for (int iw = 1; iw <= wloops; iw++){
        while (goloop){
            tnow = micros();
            if (iw == iw0 ){
                frac = ((wstep+cota) / (spw+cota));
                dtw = dtwr / frac;
                dtc = dtcr / frac;
            }
            if (iw == wloops ){
                frac = ((spw-wstep+cota) / (spw+cota));
                dtw = dtwr / frac;
                dtc = dtcr / frac;
            }
            if (iw > iw0 && iw < wloops ){
                dtw = dtwr;
                dtc = dtcr;
            }
            // Wound step test
            if (tnow - t0w >= dtw){
                PORTF |= B00000001;
                t0w += dtw;
            }
        }
    }
}

```

```

        wstep += 1;
        delayMicroseconds(microdelay);
        PORTF &= B11111110;
    }
    // Stroke step test
    if (tnow - t0c >= dtc){
        PORTF |= B01000000;
        t0c += dtc;
        cstep += 1;
        delayMicroseconds(microdelay);
        PORTF &= B10111111;
    }
    // Test for complete revolution
    if (wstep == spw){
        wcount += 1;
        //cerror = (cstep - wcount * spc); // Error in stroke
        //Serial.println(cerror);
        wstep = 0;
        // Stop the execution if the Start button is down
        if( digitalRead(StartPin) == HIGH ) {
            Stop();
            t0w = micros() ; // Reset the t0 instans
            t0c = t0w ;
            iw0 = iw + 1;
        }
        break;
    }
}
// Check for reverse stroke direction
ccount += 1;
if (crev == true){
    crev = false;
    digitalWrite(CdirPin, HIGH);
    backsteps = int(micro * 200) * 8 * ( Lbacklash / 1.5 ) ;
    // Backlash steps for Left stop
}
else{
    crev = true;
    digitalWrite(CdirPin, LOW);
    backsteps = int(micro * 200) * 8 * ( Rbacklash / 1.5 ) ;
    // Backlash steps for Left stop
}
// Reverse some steps to eliminate backlash

acbk = 50;
// Number of acceleration steps
for (int i = 1; i <= backsteps; i++){
    if(i > acbk && i < (backsteps - acbk)){
        BacklashStep(dtcr / 2.0); }
    else{
        // Acceleration loop
    }
}

```

```

        if(i <= acbk){
            dtbk = dtcr / ( (i+10.0) / (acbk+10.0) );} // Ojo
            // con la coma del 10, sino da division por cero
        if(i >= (backsteps - acbk)){
            dtbk = dtcr / ( (backsteps-i+10.0) / (acbk+10.0) );
            BacklashStep(dtbk / 2.0);
        }
    }

    // Write the end of stroke output data
    delay(100);
    // Print stroke end info
    //     Serial.print("---> Stroke ");
    //     Serial.print(ccount);
    //     Serial.print(" has ended. ");
    //     Serial.print("Total wounds is: ");
    //     Serial.print(wcount);
    //     Serial.print("\n");
    // Reset t0 (very very very very important)
    t0w = micros();
    t0c = t0w;
}
// -----
//     END THE EXECUTION ANF INFORM
//-----
// Turn off the motors
digitalWrite(WenaPin, HIGH);
digitalWrite(CenaPin, HIGH);
// Print data to serial port 4 at 115200 baud
Serial.println("---> LOOPING FINISHED... ");
Serial.print("---> Total execution time in seconds is: ");
TotalTime = millis() / 1000;
Serial.print(TotalTime);
Serial.print("\n");
Serial.print("---> Total wounds is: ");
Serial.print(wcount);
Serial.print("\n");

}

//-----
//             LOOP
//-----
void loop() {

}

void Pre(){
    // Start the communication variables
    //Serial.begin(9600);
    Serial.println("-----");
    Serial.println("----- GIMAP WOUNDING 0.0 -----");
    Serial.println("-----");
}

```

```

// Set the pins
pinMode(WdirPin, OUTPUT);
pinMode(WstepPin, OUTPUT);
pinMode(WledPin, OUTPUT);
pinMode(WenaPin, OUTPUT);
pinMode(CdirPin, OUTPUT);
pinMode(CstepPin, OUTPUT);
pinMode(CenaPin, OUTPUT);
pinMode(FanPin, OUTPUT);
pinMode(StartPin, INPUT_PULLUP);
pinMode(KillPin, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(KillPin), Kill, LOW);
pinMode(RedLedPin, OUTPUT);
pinMode(BluLedPin, OUTPUT);
pinMode(GreLedPin, OUTPUT);
digitalWrite(WenaPin, HIGH); // Disable Wound Motor
digitalWrite(CenaPin, HIGH); // Disable Stroke Motor
}

void BacklashStep(float dt){
PORTF |= B01000000;
delayMicroseconds(dt);
PORTF &= B10111111;
delayMicroseconds(dt);
}

void Stop(){
digitalWrite(WenaPin, HIGH); // Disable Wound Motor
digitalWrite(CenaPin, HIGH); // Disable Stroke Motor
while(digitalRead(StartPin) == HIGH){
    BlinkLed('B');
}
digitalWrite(WenaPin, LOW); // Disable Wound Motor
digitalWrite(CenaPin, LOW); // Disable Stroke Motor
TurnOnLed('G');
}

void Kill(){
digitalWrite(WenaPin, HIGH); // Disable Wound Motor
digitalWrite(CenaPin, HIGH); // Disable Stroke Motor
TurnOnLed('R');
digitalWrite(FanPin, LOW); // Stop the fan
while(1);
}

void TurnOnLed(char LedColor){
if(LedColor == 'R'){
    digitalWrite(RedLedPin, LOW);
    digitalWrite(BluLedPin, HIGH);
    digitalWrite(GreLedPin, HIGH);
}
}

```

```
}

if(LedColor == 'G'){
    digitalWrite(RedLedPin, HIGH);
    digitalWrite(BluLedPin, HIGH);
    digitalWrite(GreLedPin, LOW);
}
if(LedColor == 'B'){
    digitalWrite(RedLedPin, HIGH);
    digitalWrite(BluLedPin, LOW);
    digitalWrite(GreLedPin, HIGH);
}
if(LedColor == 'W'){
    digitalWrite(RedLedPin, HIGH);
    digitalWrite(BluLedPin, HIGH);
    digitalWrite(GreLedPin, HIGH);
}
}

unsigned int blinktime = 500;
void BlinkLed(char LedColor){
    if(LedColor == 'R'){
        TurnOnLed('R');
        delay(blinktime);
        TurnOnLed('W');
        delay(blinktime);
    }
    if(LedColor == 'B'){
        TurnOnLed('B');
        delay(blinktime);
        TurnOnLed('W');
        delay(blinktime);
    }
    if(LedColor == 'G'){
        TurnOnLed('G');
        delay(blinktime);
        TurnOnLed('W');
        delay(blinktime);
    }
}
```