Control Codes of Stepper1, Stepper2 and Monitoring System

1- Control Script for the Stepper1 Main Linear Actuator

```
// ==================== stepper actuator Nema 23 ========================
/*
To determine the stepper distance, speed and acceleration of the linear
motion are in steps. From the stepper's datasheet, rotor has 50 teeth,
stator has 48 teeth, steps/rev = 200 ==>  step = 1.8 degree.
// Step angle=1/4 (angular bitch), angular bitch=360/50
Resolution = 10e-6 m/step.
// To convert the resolution from m/step to m/rev
Resolution = 10e-6 m/step * (200 step/rev) = 2 mm/rev *
Thread lead =2 mm                          // from the stepper's data sheet
2 mm linear distance = 2 mm/(2 mm/rev) = 1 rev
3 cm = 30 mm = 30 / 2 = 15 rev
15 rev * 200 steps = 3000 steps               // distance in steps
15 rev * 400 steps = 6000 steps               // 1/2 micro stepping using
the stepper drive unit, 1 step=0.9 degree
15 rev * 800 steps = 12000 steps              // 1/4 micro stepping using
the stepper drive unit, 1 step = 0.45 degree
speed = 80 mm/s = (80 mm/s) / 2 mm/rev = 40 rev/sec * 200 steps/rev =8000
steps/s // nominal stepper speed in steps/s
*/
 // ========================= Libraries ================================

 #include <AccelStepper.h>   //Supported with acceleration and deceleration
 #include <Stepper.h>               // Stepper library
 #include <WheatstoneBridge.h>      // To interface the load cell to arduino
 #include <NewPing.h>               // Ultrasonic sensor library
 #include <Wire.h>                  // Serial communication library
 #include <LiquidCrystal_I2C.h>     // LCD 20X4 library
 #include <PID_v2.h>                // PID on arduino library

// ====================== Define connections ============================

 #define buttonPin 2                // Digital input (DI), push botton
 #define reverseSwitchPin 3         // DI, to reverse the rotation direction
 #define encoderOutputA 4
 #define encoderOutputB 5
 #define stepPin 6         // DO, to drive pulses to the stepper drive unit
 #define dirPin 7          // DO, to determin the rotation direction
 #define ledPin 8          // Digital output (DO), led
 #define triggerPin 9      // DO, to send ultrasonic signal, HC-SR04 trig
 #define echoPin 10        // DI, to receive ultrasonic signal, HC-SR04 echo
 #define scanningRelayPin 11 // DO, to trigger stepper2 to detect the rails
 #define trackingPin 12      // DI from stepper2 to start tracking.
 #define loadCellPin A0      // Analog input, to monitor the CF
 #define swingingRelayPin 15 // DO, to stepper 2 to start swinging, A1.

 // ====================== Define variables ============================

         //-------------- Push Button variables --------------//

 int  ledState = LOW;               // current state of the output pin
 int  currentButtonState;           // current reading from the input pin
 int  lastButtonState = HIGH ;// previous reading from the PULLUP input pin
 int  buttonReading;
unsigned long lastDebounceTime = 0; // last time output pin was toggled
unsigned long debounceDelay = 100;  // the debounce time
bool start;
bool stop ;
```

```
long CurrentPosition = 0;                    // stepper current position
bool volatile setDir = 0 ;                   // bool volatile for ISR
        //------------ Serial monitor variables ------------- //

long receivedSteps = 0;                              // Number of steps
long receivedSpeed = 0;                              // Steps / second
long receivedAcceleration = 0;                       // Steps / second^2
char serialCommand;
int directionMultiplier = 1;
long NewSetpoint=0;

        //----------------- Global variables -----------------//

int dt = 500
bool newData, runAllowed,scanningAllowed, trackingAllowed, swingingAllowed
     = false;                    // booleans for (1 bit) true/false flags
bool forceControlAllowed, us_controlAllowed = false ;

        //------------- Ultrasonic sensor variables -----------//

int maxDistance = 400 ;  // Maximum distance (in cm) sensor can measure.
int distance ;
float duration;

        //-------------- Digital encoder variables ------------//

int counter = 0;                  // Incremental digital encoder's counter
int stepper1Angle = 0;            // rotor angle
int currentAstate;
int lastAstate;

// =================  PID controllers' parameters ====================

// CF PID controller variables
double Setpoint=10 , C_F_Setpoint , Input , Output , error  ;
// Force PID parameters
double Kp1 = 1.5 , Ki1 = 0.15 , Kd1 = 0.000005;
double Kp2 = 0.5 , Ki2 = 0.075 , Kd2 = 0.000025 ;

PID_v2 F_PID(Kp1, Ki1, Kd1, PID::Direct);
//PID F_PID(&Input, &Output, &Setpoint, Kp, Ki, Kd,P_ON_M,DIRECT

// ==================  Define functions prototype ====================

void reverseSwitch()                // Interrupt Service Rutine (ISR)
{
   setDir = !setDir ;
   //detachInterrrupt(0);           // 0 means pin 2, 1 means pin 3
}

AccelStepper stepper1(1, stepPin, dirPin);        //Create stepper object

NewPing sonar(triggerPin, echoPin, maxDistance);
// NewPing setup of pins and maximum distance, ultrasonic sonic.

LiquidCrystal_I2C lcd(0x27,20,4);
// set the LCD address to 0x27 for a 20 chars and 4 line display

WheatstoneBridge wsb_strain1(A0, 340, 595, 0, 255) ;

// With map factor = 1.00,  14 ADC == 14 N  full range=255 steps/14 = 180
N (C.F=valForce*10/14)
```

```
// ========================== Setup function ============================

 void setup()
 {

    Serial.begin(115200);       // Begin serial communication (Baud rate)
    pinMode(buttonPin, INPUT_PULLUP);
    // Set buttonPin as an input,internal PULLUP resistor.
    pinMode(reverseSwitchPin, INPUT_PULLUP); // Internal pull up resistor
    pinMode(encoderOutputA,INPUT);
    pinMode(encoderOutputB,INPUT);
    pinMode(ledPin, OUTPUT);
    pinMode(triggerPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(scanningRelayPin, OUTPUT);    // To stepper 2
    pinMode(trackingPin, INPUT);// From stepper2,external pulldown resistor
    pinMode(swingingRelayPin, OUTPUT);    // To stepper 2

    digitalWrite(dirPin, setDir);         // To set the stepper direction
    stepper1.setCurrentPosition(0);       // Home position
    stepper1.setMaxSpeed(2000);           // To set maximum speed in Steps/s
    stepper1.setAcceleration(1000);     // To set acceleration in Steps /s^2
    stepper1.disableOutputs();
    // disable outputs, so the motor is not getting warm (no current)

    attachInterrupt(1,reverseSwitch,FALLING);
    // ISR of reversing the stepper rotation direction.

    // int SetSampleTime(1000);  // 1 sec, (500 to 1000) for PID controller

    lcd.init();                           // initialize the lcd
    lcd.backlight();                      // Print a message to the LCD

    lastAstate = digitalRead(encoderOutputA);
    // Last state of the digital encoder output A
  }

// ============================================================================
//**************************  Mian Loop  ****************************
// ============================================================================

 void loop()
 {
     pushButton();
     // Start / Stop Push Button

     Extend_To_Scanning_Position();
     // To extend the pantograph upto rails scanning level, 50000 steps.

     Move_To_Tracking_Position();
     // To extend the pantograph upto rails tracking level, 60000 steps.

     contact_Force_Control();
     // To control the CF between sliding contacts and rails.

     ultraSonicSensor();
     // To measure the distance between head of the pantograph and rails.

     encoder();
     // For fine tunning of CF, to make stepper following the
        Encoder nub rotation direction.

     checkSerial();
```

```
    // check serial port for new commands // PB needs to be in stop mode

    driveStepper1();         // Motor drive function for serial commands

  }

//=========================================================================

 void pushButton()
 {

    buttonReading = digitalRead(buttonPin);
    // read the state of the PB contact into a variable.

    if ( buttonReading != lastButtonState)        // Switching state
    {
       lastDebounceTime = millis();
       // To reset the debouncing timer If the switch changed
    }

    if ((millis() - lastDebounceTime) > debounceDelay)
       // Debouncing time delay.
    {

        if ( buttonReading != currentButtonState)
            // if the button state has changed.
        {
            currentButtonState =  buttonReading;

            if (currentButtonState == LOW)      // PULLUP resistor PB
             {
                ledState = !ledState;
                // only toggle the LED if the new button state is LOW
             }
         }
      }

    digitalWrite(ledPin, ledState);       // set the LED

    if(ledState == HIGH)     start = true;

    else       start=false;

  lastButtonState = buttonReading;
  // save the reading, next time it'll be the lastButtonState

}

//=========================================================================

void Extend_To_Scanning_Position()
{
  /*When PB is first pressed, it extends the pantograph to the stated
   distance because initial setDir is declared 0, which is case(0)
   then reverse PB is used to extend and retract it using setDir(),
     while start PB led is on.
  motor can not be stopped while running, ARDUINO can not execute two
     Commands at a time.*/

  if(start == true )

  {
```

```
  stepper1.enableOutputs();

  switch(setDir)                        // digitalRead(reverseSwitchPin)
   {

      case( 0 ):                       // initial default value

         stepper1.moveTo(55000);
         while (stepper1.currentPosition() < 55000 && setDir == 0  )
          // The inside while loop setDir is to enable reversing the
          // direction before reaching 55000 position
         stepper1.run();
          // Without using the comparison operator" < "the stepper start
          // swinging between 60000 and 55000.
         if(stepper1.distanceToGo() == 0)
           {
              digitalWrite(scanningRelayPin,HIGH) ;
        // To be sent to stepper 2 to start detecting the conduction rails

           }

         break;

      case( 1 ):  /* The switch(1)setDir is to reverse the rotation
                     direction after reaching the position 55000 */

         digitalWrite(scanningRelayPin,LOW) ;
         digitalWrite(swingingRelayPin,LOW) ;
         forceControlAllowed = false;

         stepper1.moveTo(0);
         while (stepper1.currentPosition() != 0  && setDir == 1 )
        // The in loop setDir is to reverse the direction before reaching
           zero position.
         stepper1.run();

         if(stepper1.distanceToGo() == 0)

         break;
    }

  }

}

//=======================================================================

void  Move_To_Tracking_Position()
{
  int trackingPinReading = digitalRead(trackingPin);     // Local variable

  if( trackingPinReading == HIGH)   trackingAllowed = true;

  else trackingAllowed = false;

  if(start == true && scanningAllowed == true && trackingAllowed == true)
   {

  stepper1.enableOutputs();

  switch(setDir)                    // digitalRead(reverseSwitchPin)
   {
```

```
      case( 0 ):// The same direction as the previous step in the sequence

         stepper1.setSpeed(1000);
         stepper1.setAcceleration(1000);

         stepper1.moveTo(60000);                        // Absolute distace
         while (stepper1.currentPosition() != 60000 && setDir == 0  )
         stepper1.run();
         if(stepper1.distanceToGo() == 0)
          {

                forceControlAllowed = true;
        // This condition can be controlled by serial command

           }

         break;

      case( 1 ):// setDir is to reverse the direction after 60000 steps

         digitalWrite(scanningRelayPin,LOW) ;
         forceControlAllowed = false;

         stepper1.moveTo(0);
         while (stepper1.currentPosition() != 0  && setDir == 1 )
// setDir is to reverse the direction before reaching 0 position
         stepper1.run();

          if(stepper1.distanceToGo() == 0)

         break;
     }

   }

 }

// ---------------------------------------------------------------------------
//******************** Serial monitor commands  ************************
// ---------------------------------------------------------------------------

oid checkSerial()                  // function for receiving the commands
{
    if (Serial.available() > 0)    // if something comes from the computer
    {
        serialCommand = Serial.read();
        // pass the value to the receivedCommad variable
        newData = true;
        // indicate that there is a new data by setting this bool to true

       if (newData == true)
        // this long switch-case statement is only entred if there is a
           new command from the computer
        {
            switch (serialCommand)          // To check what the command is
            {


            case 'H':                       // H: Homing

                runAllowed = true;
                Serial.println("Homing");   // Print the message
                GoHome();                   // Run the function
```

```
            break;

        case 'L':                       // L: Location

            runAllowed = false;         // Motor drive disabled
            stepper1.disableOutputs(); // disable power
            Serial.print("Current location of the motor: ");
            Serial.println(stepper1.currentPosition());
            // Printing the current position in steps.

            break;

        case 'P':

            PrintCommands();    // Print all serial monitor commands

             break;

        case 'E':
             // Extend the pantograph,uses the move() function of the
             // AccelStepper library, which means that it moves
             // relatively to the current position.

            receivedSteps = Serial.parseFloat();  //value for the steps
            receivedSpeed = Serial.parseFloat();  //value for the speed
            directionMultiplier = 1;             //define the direction
            Serial.println("Positive direction.");//print the action
            RotateRelative();                     //Run the function

// example:" E2000 400", distance = 2000 steps  and speed = 400 steps/s

            break;

         case 'R':

            receivedSteps = Serial.parseFloat();
            receivedSpeed = Serial.parseFloat();
            directionMultiplier = -1;
            Serial.println("Negative direction.");
            RotateRelative();

           break;

         case 'F':
// controlling the main stepper to maintain the CF

            forceControlAllowed = true // sliding contacts on track
            Serial.println("Force control in operation");
            contact_Force_Control();

            break;
         case 'Q':  // To set the setpoint value of the CF

          NewSetpoint = Serial.parseFloat();
          contact_Force_Control();

             break;

         case 'O':                          // Off, stops the operation

            stepper1.stop();                // stop motor
            stepper1.disableOutputs();      // disable power
```

```
                Serial.println("Stopped.");    // print action
                runAllowed = false;            // disable running
                forceControlAllowed = false;
                us_controlAllowed = false;

                break;

              case 'D':     // Measuring distance using ultrasonic sensor

                us_controlAllowed = true;
                Serial.println("Ultra sonic in operation");
                ultraSonicSensor() ;

                break;

                default:

                break;

            }
        }

        newData = false;
        //The newData is set to false again, to receive new commands again

    }
}

//---------------------------------------------------------------------------
//******************** CF Control Function *********************
//---------------------------------------------------------------------------

void contact_Force_Control()
{

  int valRaw ; // Load cell,ADC measured value (0 - 1023),controller input
  int valForce; // Measured force, controller input (N)
  int contactForce; // Calculated CF for LCD

                //---------- Force measurements -----------//

 if (start==true && forceControlAllowed==true && trackingAllowed==true)
  {
// Setpoint = NewSetpoint;  // uncommented when serial command is used
    C_F_Setpoint = (Setpoint*10/5);               // 1 bit = 2 N
    F_PID.Start(Input,  Output,  Setpoint );      // setpoint

    valForce = wsb_strain1.measureForce();// Read strain 1, Arduino library
    valRaw = wsb_strain1.getLastForceRawADC();
    valRaw = constrain(valRaw, 340, 595);  // 255 Raw value
    Serial.print("CF control in effect ");

                //----------- Average of valRaw -----------//

     int sum_valRaw , avg_valRaw;
     for (int i=0; i<10; i++)
        { sum_valRaw += valRaw;
        avg_valRaw = sum_valRaw/10;
        }
               //---------------------------------------//

     Input = avg_valRaw;   // uncommented if averaging the samples is used
  // Input = valRaw;   // uncommented if averaging the samples is not used
```

```
   Input = map(Input, 340, 595, 0, 255);

              //-------- Load cell calibration ---------//


// contactForce = (valRaw*10/14)-243 ;
  /* 1 N = 1.4 bit, Gain = 100 N/140 bits (m), Offset = (340/1.4)= -243
     (B), F = mX+b, X=valRaw*/

// contactForce = (valRaw*10/12)-283 ;
  /* 1 N = 1.2 bit, Gain = 120/100 bits (m), offset = (340/1.2) = -
     283  (for both DAQ and serial), Foe DAQ: 1 bit = 1/1.2 = 0.8333N =
     4.88mV , m = 0.8333N/0.00488 V = 170 N/V*/

// contactForce = (valRaw*10/11)-309 ;         // 1 N = 1.1 bit

// contactForce = (valRaw*10/10)-340 ;
   /* 1 N = 1.0 bit,  equivalent to valForce ,  for DAQ: 1bit = 1N =
    4.88mV, m =1/0.00488 = 204N/V,  b is the same for ADC and DAQ = -340*/

// contactForce = ((valRaw*10/10)-340)*3 ;
   /* 1 N * 3 = 1 bit , offset = 340 * 3 = 1020 , for DAQ: 1 bit = 3N =
    4.88mV , m = 3N/ 0.00488V = 615 N/V*/

// contactForce = ((valRaw*10/10)-340)*2 ;

// contactForce = (valRaw*10/9)-378  ;
   /*1 N = 0.9 bit,  for DAQ m = 227 N/V,1 bit = 1/0.9 = 1.11N = 4.88mV
     , m = 1.11 N/0.00488 V = 227 N/V*/

// contactForce = (valRaw*10/8)-425  ;
   /* 1 N = 0.8 bit   for DAQ m = 255 N/V,1 bit = 1/0.8 = 1.25N = 4.88mV
    , m = 1.25 N/0.00488 V = 255 N/V*/

// contactForce = (valRaw*10/7)-486;
   /* 1 N = 0.7 bit,  for DAQ m = 292 N/V*/

// contactForce = (valRaw*10/6)-566  ;
   /* 1 N = 0.6 bit,  for DAQ m = 340 N/V*/

       contactForce = (valRaw*10/5)-680  ;
   /* 1 N = 0.5 bit,  for DAQ m = 408 N/V*/

//------------- Rotation direction of the stepper motor --------------//

   if (Setpoint > Input )  digitalWrite(dirPin,HIGH);  // Set Dir high

   else   digitalWrite(dirPin,LOW);                    // Set Dir low

 //---------------------- CF control ---------------------//

  error = abs(Setpoint-Input);  // comput absolute proportional raw value

  if(error >= 255 )error = 255; // Limiter, anti integrator winding up

  if(error < 3 )   {   error = 0;  analogWrite(6,LOW);   }

    if(error <10 )       // Adaptive PID parameters
      /* error < 7 ADC value (around 10 N) will be ignored othewise
         the controller will be bouncing*/
        {
           F_PID.SetTunings(Kp2, Ki2, Kd2);
```

```
            }
          else
              {
                 F_PID.SetTunings(Kp1, Ki1, Kd1);

              }

           Output = F_PID.Run(Input);            //pid contollr output

//        analogWrite(6,Output);

          for(int x = 50*error; x > 0; x--)
          /* Loop step times, at low foce deviation linear motion very ittle,
             to show the effect of the deviation on linear motion in both
             directions use 50*error */

            {
               digitalWrite(stepPin,HIGH);             // Output high
               delayMicroseconds(Output*2/5);
               // On duty cycle 50%, T=1000 us, f=1 k Hz   (255-Output)
               digitalWrite(stepPin,LOW);               // Output low
               delayMicroseconds(Output*1/5);
               // Off duty cycle 50%   (0.5*(255+output))

               if(error<=5){
                          Serial.print("Power disc operation is allowed ");
                          break;
                           }
             }

//---To plot the controller setpoint, input and output in ADC values ---//

          Serial.print("Setpoint:");
          Serial.print(Setpoint);
          Serial.print("\tInput:");
          Serial.print(Input);
          Serial.print("\tOutput:");
          Serial.print(Output);
          Serial.print("\tC_F_Setpoint:");
          Serial.print(Setpoint*10/5);
          Serial.print("\tcontactForce:");
          Serial.println(contactForce);

       delay(dt);

       // --------------- For the display LCD 4x20 ----------------//

      lcd.setCursor(0,3);
      lcd.print("CF = ");
      lcd.setCursor(16,3);
      lcd.print(contactForce);
      lcd.setCursor(18,3);
      lcd.print("N");
     }

if(error<=5 && forceControlAllowed==true)
digitalWrite(swingingRelayPin,HIGH) ;
else digitalWrite(swingingRelayPin,LOW) ;

}

//=====================================================================
```

```
void ultraSonicSensor()
{
  int ref_distance = 10;
  int error;
  int us_controlAllowed = false ;  // Triggried from the serial monitor
  if (start == true &&  us_controlAllowed == true)
   {

// distance = sonar.ping_cm(); /*To calculate distance using Library
                               Function. Send ping, get the distance in cm*/

    digitalWrite(triggerPin, LOW); // Distance calculation using time and
    delayMicroseconds(2);          // sound speed (343 mm/s)
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration/2) / 29.1;
    // 1/34300 cm/s = 29.1 s/cm , duration in us

     Serial.print("Distance = ");   // It disturbs the Force data.
     Serial.print(distance);
     Serial.println("cm");
     delay(dt);

     if (distance >= 400 || distance <= 2)
     {
      Serial.println("Out of range");

      lcd.setCursor(4,1);
      lcd.print("Out of range");
      digitalWrite(6,0);
     }
     else
     {
      lcd.setCursor(0,0);
      lcd.print("Distance = ");
      lcd.setCursor(11,0);
      lcd.print(distance);
      lcd.setCursor(14,0);
      lcd.print("cm");
     }
   delay(dt);

   }
  }

//=========================================================================

void encoder()
{
  currentAstate = digitalRead(encoderOutputA);

  if (currentAstate != lastAstate)
  {
     if (digitalRead(encoderOutputB) != currentAstate)
     {
       counter ++;
       stepper1Angle ++;
       rotateCW();            // driveStepper1();
     }
     else {
```

```
        counter --;
        stepper1Angle --;
        rotateCCW();          // driveStepper1();

           }

      Serial.print("Position: ");
      Serial.print(counter);
      Serial.print("Stepper1 angle: ");
      Serial.print(stepper1Angle*9 );
      Serial.print("  deg");
      Serial.println();
/* 1/4 *1.8 degree/step = 0.45 degree/step, 1 count =10 steps (from for
loop, then, 1 count = 9 degrees. 2 rotations of the rotary encoder nub = 1
rotation of the Nema 17 stepper motor*/
//     if (counter >=30 ) counter =0;

      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("counter: ");
      lcd.print(counter);
      lcd.setCursor(0,1);
      lcd.print("Position: ");
      lcd.print(stepper1Angle*9);
      lcd.print("deg");
      lcd.setCursor(0,0);

   }

      lastAstate = currentAstate;
}

// ----------------------------------------------------------------------

void rotateCW()
{
    digitalWrite(dirPin,LOW);
    for(int x=1; x<=10; x++)
    {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
    }
}

void rotateCCW()
{
    digitalWrite(dirPin,HIGH);
    for(int x=1; x<=10; x++)
    {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
    }
}

//======================================================================

void driveStepper1()                          // function for the motor
{
    if (runAllowed == true)
```

```
    {
        stepper1.enableOutputs();              // enable pins
        stepper1.run();
  // step the motor (this will step the motor by 1 step at each loop)
    }
    else
 // program enters this part if the runAllowed is false
    {
        stepper1.disableOutputs();            // disable outputs
        return;
    }
}

//=====================================================================

void GoHome()
{
    if (stepper1.currentPosition() == 0)
    {
        Serial.println("We are at the home position.");
        stepper1.disableOutputs();    //disable power
    }
    else
    {
        stepper1.setMaxSpeed(2000);
        stepper1.moveTo(0);           //set abolute distance to move
    }

    digitalWrite(scanningRelayPin,LOW);
    digitalWrite(swingingRelayPin,LOW) ;
}

//=====================================================================

void RotateRelative()
{
   //To move X steps from the current rotor position in a given direction.
   //The direction is determined by the multiplier (+1 or -1)

    runAllowed = true;
   //allow running - this allows entering driveStepper1() function.
    stepper1.setMaxSpeed(receivedSpeed);         //set speed
    stepper1.move(directionMultiplier * receivedSteps);
   //set relative distance and direction
}

//=====================================================================

void PrintCommands()
{
  //Printing the serial monitor commands
  Serial.println(" 'P' : Prints all the commands and their functions.");
  Serial.println(" 'H' : Go back to 0 position from the current position.");
  Serial.println(" 'L' : Prints the current position/location of the motor.");
  Serial.println(" 'E' : Extend the mini pantograph");
  Serial.println(" 'R' : Retract the mini pantograph");
  Serial.println(" 'F' : CF Control");
  Serial.println(" 'Q' : Setpoint value of the CF ");
  Serial.println(" 'D' : Measuring distance using ultrasonic sensor ");
}

//============================  end  ================================
```

2- Control Script for the Stepper2 Auxiliary Linear Actuator

```
// ================== stepper actuator Nema 17 ==========================
/*
To determin the stepper distance, speed and acceleration of the linear
motion in steps
steps/rev = 200 ==>  step = 1.8 degree   // from the stepper's data sheet
Resolution = 7.9e-6 m/step               // from the stepper's data sheet
Resolution = 7.9e-6 m/step * (200 step/rev) = 1.580 mm/rev
// To convert the resolution from m/step to m/rev
Thread lead = 1.59 mm                    // from the stepper's data sheet
2 mm linear distance = 2 mm/(1.58 mm/rev) = 1.266 rev
2 cm = 20 mm = 20 / 1.58 = 12.66 rev
12.66 rev * 200 steps = 2,532 steps                  // distance in steps
12.66 rev * 400 steps = 5,064 step
// 1/2 micro stepping using the stepper drive unit, 1 step = 0.9 degree
12.66 rev * 800 steps = 10,128 steps
// 1/4 micro stepping using the stepper drive unit, 1 step = 0.45 degree
speed = 36 mm/s = (36 mm/s) / 1.58 mm/rev = 22.64 rev/sec * 200 steps/rev =
4528 step/s                              // nominal stepper speed in steps/s
*/
// ========================== Libraries ================================

 #include <AccelStepper.h>
 #include <Stepper.h>
 #include <NewPing.h>
 #include <Wire.h>
 #include <LiquidCrystal_I2C.h>
 #include <PID_v2.h>

// ====================== Define connections ==========================

 #define buttonPin 2           // Digital input (DI), PB
// #define reverseSwitchPin 3  // DI is in parallel to the proximity switch
 #define proximitySensorPin 3
 #define encoderOutputA 4      // Incremental digital encoder
 #define encoderOutputB 5
 #define stepPin 6      // DO, to drive pulses to the stepper drive unit
 #define dirPin 7       // DO, to determine the rotation direction
 #define ledPin 13
 #define triggerPin 9   // DO (DO), to send ultrasonic signal, HC-SR04 trig
 #define echoPin 10          // DO, to receive the ultra sonic signal
 #define scanningPin 11      // DI, from stepper1 to detect the rails
 #define trackingRelayPin 12 // DO, to to stepper 1 to start tracking
 #define swingingPin 15      // DI, to receive a signal from stepper1, A1
 #define powerSupplyRelayPin 16 // Do, to energise the power disc, A2

// ====================== Define variables ============================

          //-------------- Push Button variables --------------//

 int  ledState = LOW;               // current state of the output pin
 int  currentButtonState;           // current reading from the input pin
 int  lastButtonState = HIGH ;// previous reading from the PULLUP input pin
 int  buttonReading;
 unsigned long lastDebounceTime = 0; // last time output pin was toggled
 unsigned long debounceDelay = 100;  // the debounce time
 bool start;
 bool stop ;
 long CurrentPosition = 0;                  // stepper current position
 bool volatile setDir = 0 ;                 // bool volatile for ISR

          //------------ Serial monitor variables ------------ //
```

```
long receivedSteps = 0;                              // Number of steps
long receivedSpeed = 0;                              // Steps / second
long receivedAcceleration = 0;                       // Steps / second^2
char serialCommand;
int directionMultiplier = 1;
long NewSetpoint=0;

        //----------------- Global variables -----------------//

int dt = 500                      // Time delay
bool newData, runAllowed,scanningAllowed, trackingAllowed, swingingAllowed
     = false;                     // booleans for (1 bit) true/false
bool bouncingAllowed, us_controlAllowed = false ;

        //------------- Ultrasonic sensor variables -----------//

int maxDistance = 400 ;  // Maximum distance (in cm) sensor can measure.
int distance ;
float duration;

        //-------------- Digital encoder variables ------------//

int counter = 0;                  // Incremental digital encoder's counter
int stepper2Angle = 0;            // rotor angle
int currentAstate;
int lastAstate;

// =================  PID controllers' parameters ====================

// Distance PID controller variables
double Setpoint , Input , Output , error  ;
// Distance PID parameters
double Kp1 = 1.5 , Ki1 = 0.15 , Kd1 = 0.000005;
double Kp2 = 0.5 , Ki2 = 0.075 , Kd2 = 0.000025 ;

PID_v2 D_PID(Kp1, Ki1, Kd1, PID::Direct);

// PID D_PID(&Input, &Output, &Setpoint, Kp, Ki, Kd,P_ON_M,DIRECT

 // =================  Define functions prototype ====================

 void proximitySensor()            // Interrupt Service Rutine (ISR)
 {
  setDir = !setDir;
 }

 AccelStepper stepper2(1, stepPin, dirPin);    //Create stepper object

 NewPing sonar(triggerPin, echoPin, maxDistance);
 // NewPing setup of pins and maximum distance, ultrasonic sensor.

 LiquidCrystal_I2C lcd(0x27,20,4);
 // set the LCD address to 0x27 for a 20 chars and 4 line display

// ========================== Setup function ============================


 void setup()
 {

    Serial.begin(115200);      // Begin serial communication (Baud rate)
    Serial.println ("Send 'P' for printing the commands.");
```

```
    pinMode(buttonPin, INPUT_PULLUP);
    /* Set button Pin as an input, internal PULLUP resistor
       pinMode(reverseSwitchPin, INPUT_PULLUP);
       When it is connected in parallel to the prox. Sensor, it needs to be
       defined as INPUT! */
    pinMode(proximitySensorPin, INPUT);        // External PULLUP resistor
    pinMode(encoderOutputA,INPUT);
    pinMode(encoderOutputB,INPUT);
    pinMode(ledPin, OUTPUT);
    pinMode(triggerPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(scanningPin, INPUT);              // From stepper1 control unit
    pinMode(trackingRelayPin, OUTPUT);
    // To stepper1 when scanning is done and conduction rails detected
    pinMode(swingingPin, INPUT);// From stepper1 to start and stop swinging
    pinMode(powerSupplyRelayPin, OUTPUT);

    digitalWrite(dirPin, setDir);          // To set the stepper direction

    stepper2.setCurrentPosition(0);        // Home position
    stepper2.setMaxSpeed(4000);            // Speed = Steps / second
    stepper2.setAcceleration(2000);   // Acceleration = Steps /(second)^2

    stepper2.disableOutputs();
    // disable outputs, so the motor is not getting warm (no current)

    attachInterrupt(1, proximitySensor, FALLING);
 // ISR to reverse the stepper rotation direction when detecting the rails
 // int SetSampleTime(1000);  // 1 sec, (500 to 1000) for PID controller

    lcd.init();                            // initialize the lcd
    lcd.backlight();                       // Print a message to the LCD.

    lastAstate = digitalRead(encoderOutputA);
    // Last state of the digital encoder output A

 }

// ================================================================================
//***************************  Mian Loop  ******************************
// ================================================================================

 void loop()
 {

   pushButton();
   // Start/Stop push button

   Scanning ();
   /* To start moving up and down continuously until the rails are
      detected, then it goes to home position, or, by pressing the reverse
      switch*/

   Swinging();
   // It can be activated be pressing the reverse switch

   Bouncing();
   /* comment swinging and bouncing functions to run scanning. De-comment
      them one by one otherwise, during swinging stepper sometimes bounce
      by bouncing function*/

    encoder();
     /* For fine tuning of CF, to make the stepper follow the
        Encoder nub rotation direction. PB needs to be in stop mode in
        order to use the digital encoder*/
```

```
    checkSerial();
    // check serial port for new commands // PB needs to be in stop mode.

    driveStepper2();  // Motor drive function for serial commands

 }

//=========================================================================

void pushButton()
 {

     buttonReading = digitalRead(buttonPin);
     // read the state of the PB contact into a variable.

     if ( buttonReading != lastButtonState)         // Switching state
     {
        lastDebounceTime = millis();
        // To reset the debouncing timer If the switch changed
     }

     if ((millis() - lastDebounceTime) > debounceDelay)
        // Debouncing time delay.
     {

         if ( buttonReading != currentButtonState)
             // if the button state has changed.
         {
             currentButtonState =  buttonReading;

             if (currentButtonState == LOW)      // PULLUP resistor PB
              {
                 ledState = !ledState;
                 // only toggle the LED if the new button state is LOW
              }
          }
       }

     digitalWrite(ledPin, ledState);       // set the LED

     if(ledState == HIGH)     start = true;

     else       start=false;

  lastButtonState = buttonReading;
  // save the reading, next time it'll be the lastButtonState

}

//=========================================================================

void Scanning ()
{

  int scanningPinReading = digitalRead (scanningPin);

  if( scanningPinReading == HIGH)scanningAllowed = true;
  else scanningAllowed = false;

  if(start == true && scanningAllowed == true )
   // To be received from stepper 2
   {
```

```
    stepper2.enableOutputs();

    switch(setDir)
      {

        case( 0 ):
/*Swinging is continuous until the reverse switch is pressed, then
Stepper drives to zero position and stop,  as long as start true
and  swingingPinReading high continuously*/

         stepper2.moveTo(15000);
         while (stepper2.currentPosition() != 15000 && setDir == 0)
         // Full speed up to 5000    setDir = !setDir
         stepper2.run();
         if(stepper2.distanceToGo() == 0)
        {
              stepper2.moveTo(-15000);
              while (stepper2.currentPosition() != -15000 && setDir == 0  )
              stepper2.run();

              if(stepper2.distanceToGo() == 0)
               {
                  stepper2.moveTo(0);
                  while (stepper2.currentPosition() != 0 && setDir == 0  )
                  stepper2.run();
               }
                 Serial.print("The conduction rails are not identified! ")
         //after one full cycle, scanning can be stopped using rev. Switch.

                 lcd.clear();
                 lcd.setCursor(0,0);
                 lcd.print("No conduction rails");
         }

          break;

        case( 1 ):
  /* when proximity sensor detects conduction rails, motor reverses
     the direction and stops at zero position, or could use an
     external switch in parallel with it to stop the scanning, in
     case the proximity sensor detects nothing*/

   stepper2.moveTo(0);
   while (stepper2.currentPosition() != 0  && setDir == 1)
   stepper2.run();

          if(stepper2.distanceToGo() == 0 && scanningAllowed == true )

          {
              Serial.println(" The conduction rails are detected ");

                 lcd.clear();
                 lcd.setCursor(0,0);
                 lcd.print("cond. rails detected");

               delay(dt);
              digitalWrite(trackingRelayPin,HIGH);

          }


         break;
```

```
    }

  }
else digitalWrite(trackingRelayPin,LOW);
}

//=========================================================================

void Swinging()

// The reverse switch needs to be pressed to reset the direction at
detecting the conduction rails
{

  int swingingPinReading = digitalRead(swingingPin);
  if( swingingPinReading == HIGH) swingingAllowed = true;
  else swingingAllowed = false;

  if(start == true && swingingAllowed == true)
    // To be received from stepper 2
   {
     stepper2.enableOutputs();

      switch(setDir)                      // digitalRead(reverseSwitchPin)
      {

      case( 0 ):
 /* Swinging is continuous until the reverse switch is pressed, which drive
to zero position and stop it, as long as start true and  swingingPinReading
HIGH continuously*/
        for(int j=0; j<2; j++)
        {
         Serial.println(" \tSwinging in operation ");
        stepper2.moveTo(15000);
        while (stepper2.currentPosition() != 15000 && setDir == 0 &&
               swingingAllowed == true  )  // Full speed up to 15000
        stepper2.run();
          if(stepper2.distanceToGo() == 0)
          {
              stepper2.moveTo(- 15000);     // for bouncing
              while (stepper2.currentPosition() != -15000 && setDir == 0 &&
                     swingingAllowed == true  )
              stepper2.run();
           }
          }
              if(stepper2.distanceToGo() == 0)
              {
              stepper2.moveTo(0);
              while (stepper2.currentPosition() != 0
              stepper2.run();
              }



        break;

      case( 1 ):       // To stop the swinging using the rev. Switch.

         stepper2.moveTo(0);
         while (stepper2.currentPosition() != 0  && setDir == 1 &&
                swingingAllowed == true)     // Full speed up to 0
         stepper2.run();
         Serial.println(" The swinging is terminated ");
```

```
          break;
        }
    }

 }

// ---------------------------------------------------------------------------
//******************** Serial monitor commands  ************************
// ---------------------------------------------------------------------------

Void checkSerial()                  // function for receiving the commands
{
    if (Serial.available() > 0)    // if something comes from the computer
    {
        serialCommand = Serial.read();
        // pass the value to the receivedCommad variable
        newData = true;
        // indicate that there is a new data by setting this bool to true

       if (newData == true)
         // this long switch-case statement is only entred if there is a
            new command from the computer
        {
            switch (serialCommand)          // To check what the command is
            {

         case 'S':

            /* To scan for conduction rails when the sliding contacts at
            scanning position, scanningAllowed*/

            //receivedSteps = Serial.parseFloat();   // value for the steps
            //receivedSpeed = Serial.parseFloat();   // value for the speed
            Serial.println("Scan for conduction rails");// print the action
            Scanning ();            // Run the function
                                    // example: P2000 400 - 2000 steps
            break;

         case 'W':

            /* To make the sliding contacts swinging while operation in
            order to distribute the friction evenly */
            runAllowed = true ;
            swingingAllowed = true;
            Serial.println("Start swinging");           // print action
            Swinging();

            break;

          case 'B':

             // Measuring distance using ultrasonic sensor

             us_controlAllowed = true;
             Serial.println("Ultra sonic in operation");
             ultraSonicSensor() ;

              break;

            case 'H':                          // H: Homing

               runAllowed = true;
               Serial.println("Homing");  // Print the message
```

```
            GoHome();                   // Run the function

            break;

        case 'L':                        // L: Location

            runAllowed = false;         // Motor drive disabled
            stepper2.disableOutputs(); // disable power
            Serial.print("Current location of the motor: ");
            Serial.println(stepper1.currentPosition());
            // Printing the current position in steps.

            break;

        case 'P':

            PrintCommands();    // Print all serial monitor commands

            break;

        case 'D':
             // Extend the pantograph,uses the move() function of the
             // AccelStepper library, which means that it moves
             // relatively to the current position.

            receivedSteps = Serial.parseFloat();  //value for the steps
            receivedSpeed = Serial.parseFloat();  //value for the speed
            directionMultiplier = 1;             //define the direction
            Serial.println("Positive direction.");//print the action
            RotateRelative();                     //Run the function

// example:" E2000 400", distance = 2000 steps  and speed = 400 steps/s

            break;

        case 'U':

            receivedSteps = Serial.parseFloat();
            receivedSpeed = Serial.parseFloat();
            directionMultiplier = -1;
            Serial.println("Negative direction.");
            RotateRelative();

            break;

        case 'Q':  // To set the setpoint value of the CF

            NewSetpoint = Serial.parseFloat();
            contact_Force_Control();

            break;

        case 'O':                          // Off, stops the operation

            stepper2.stop();               // stop motor
            stepper2.disableOutputs();     // disable power
            Serial.println("Stopped.");    // print action
            runAllowed = false;            // disable running
            us_controlAllowed = false;

            break;

        case 'N':
```

```
                    encoder();   // To control stepper2 using digital encoder

                    break;

                    default:

                    break;

                }
            }

            newData = false;
            //The newData is set to false again, to receive new commands agai
        }
}
//------------------------------------------------------------------------------
//**************** Ultrasonic Distance Control Function ********************
//------------------------------------------------------------------------------

void Bouncing()
{

// distance = sonar.ping_cm(); /*To calculate distance using Library
                               Function. Send ping, get the distance in cm*/

    digitalWrite(triggerPin, LOW); // Distance calculation using time and
    delayMicroseconds(2);          // sound speed (343 mm/s)
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration/2) / 29.1;
    // 1/34300 cm/s = 29.1 s/cm , duration in us

    if(us_controlAllowed == true && swingingAllowed == true)
       bouncingAllowed = true;  // us_controlAllowed from serial monitor

    if (start == true && bouncingAllowed == true)
     {

       // ------ PID controller, calculating output signal ------//

    Setpoint = 10*NewSetpoint;  // Adjstible setpoint via serial monitor
   // Setpoint = 10;            // Constant setpoint

    Input = 10*distance;

    D_PID.Start(Input,  Output,  Setpoint );

   //---------------- Average of the Raw value -----------------------//

    int sum_distance , avg_distance;
     for (int i=0; i<10; i++)
{
    sum_distance += distance;
    avg_distance = sum_distance/10;
}

// ------------------------------------------------------------------ //
     Input = avg_distance*10;
//    Input = distance*10;
```

```
//------------- Rotation direction of the stepper motor -------------//

    if (Setpoint > Input )  digitalWrite(dirPin,HIGH);  // Set Dir high

    else   digitalWrite(dirPin,LOW);                   // Set Dir low

 //----------------------    Distance control   ---------------------//

    error = abs(Setpoint- Input);      // distance away from setpoint

    if(error > = 25 )error = 25;        // Ant integrator winding up
    if(error < = 5)  error = 0 ;

    if(error > 10)

            {
               D_PID.SetTunings(Kp2, Ki2, Kd2);

            }
        else
            {
               D_PID.SetTunings(Kp1, Ki1, Kd1);

            }

            Output = D_PID.Run(Input);

//        analogWrite(6,Output);
//     Output=map(Output,0,1300,0,255);

         for(int x = 253.2*error; x > 0; x--)
             // 253.2 step/mm, at 400step/rev setting

           {
              digitalWrite(stepPin,HIGH);                // Output high
              delayMicroseconds(Output*2/5);
               // On duty cycle 50%, T=1000 us, f=1 k Hz   (255-Output)
              digitalWrite(stepPin,LOW);                  // Output low
              delayMicroseconds(Output*1/5);
               // Off duty cycle 50%    (0.5*(255+output))
              if(error<=5)break;
           }
    }

     bouncingAllowed = false;
 //     else digitalWrite(6,0);

 //---To plot the controller setpoint, input and output in ADC values ---//

    Serial.print("Setpoint:");
    Serial.print(Setpoint);
    Serial.print("\Input:");
    Serial.print(Input);
    Serial.print("\tOutput:");
    Serial.println(Output);

     delay(dt);

     Serial.print("Distance = ");
     Serial.print(distance);
     Serial.println("cm");

     if (distance >= 20 || distance <= 5)
```

```
      {
      Serial.println("Out of range");

       lcd.setCursor(4,1);
       lcd.print("Out of range");
       digitalWrite(6,0);
      }
      else
      {
       lcd.setCursor(0,0);
       lcd.print("Distance = ");
       lcd.setCursor(11,0);
       lcd.print(distance);
       lcd.setCursor(14,0);
       lcd.print("cm");
      }
   delay(dt);
}
//=============================================================================

void encoder()
{
  currentAstate = digitalRead(encoderOutputA);

  if (currentAstate != lastAstate)
  {
     if (digitalRead(encoderOutputB) != currentAstate)
     {
       counter ++;
       stepper2Angle ++;
              rotateCW();
     //RunTheMotor();
     }
     else {
       counter --;
       stepper2Angle --;
       rotateCCW();
       //RunTheMotor();
          }

     Serial.print("Position: ");
     Serial.print(counter);
     Serial.print("     Stepper2 angle: ");
     Serial.print(stepper2Angle*9 );
     Serial.print("  deg");
     Serial.println();

//     if (counter >=30 ) counter =0;

     lcd.clear();
     lcd.setCursor(0,0);
     lcd.print("counter: ");
     lcd.print(counter);
     lcd.setCursor(0,1);
     lcd.print("Position: ");
     lcd.print(stepper2Angle*9);
  /* 1/4 *1.8 degree/step = 0.45 degree/step, 1 count =10 steps
    (from the   for loop, then, 1 count = 9 degrees */
     lcd.print("deg");
  /* 2 rotations of the rotary encoder nub = 1 rotation of the
     Nema 17 stepper motor */
     lcd.setCursor(0,0);
```

```
   }

     lastAstate = currentAstate;
}

// ================================================================

void rotateCW()
{
    digitalWrite(dirPin,LOW);
    for(int x=1; x<=10; x++)
    {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
    }
}
void rotateCCW()
{
    digitalWrite(dirPin,HIGH);
    for(int x=1; x<=10; x++)
    {
    digitalWrite(stepPin,HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin,LOW);
    delayMicroseconds(500);
    }
}

// ================================================================

void driveStepper2()                          // function for the motor
{
    if (runAllowed == true)
    {
        stepper2.enableOutputs();            // enable pins
        stepper2.run();
 // step the motor (this will step the motor by 1 step at each loop)
    }
    else
 // program enters this part if the runAllowed is false
    {
        stepper2.disableOutputs();            // disable outputs
        return;
    }
}

// ================================================================

void GoHome()
{
    if (stepper2.currentPosition() == 0)
    {
        Serial.println("We are at the home position.");
        stepper2.disableOutputs(); //disable power
    }
    else
    {
        stepper2.setMaxSpeed(2000);
        stepper2.moveTo(0);          //set abolute distance to move
    }
```

```
    digitalWrite(trackingRelayPin,LOW);

}

// ========================================================================

void RotateRelative()
{
   //To move X steps from the current rotor position in a given direction.
   //The direction is determined by the multiplier (+1 or -1)

    runAllowed = true;
   //allow running - this allows entering driveStepper1() function.
    Stepper2.setMaxSpeed(receivedSpeed);          //set speed
    Stepper2.move(directionMultiplier * receivedSteps);
   //set relative distance and direction
}

//========================================================================
void PrintCommands()
{
  //Printing the serial monitor commands
  Serial.println(" 'P' : Prints all the commands and their functions.");
  Serial.println(" 'S' : Scan for conduction rails.");
  Serial.println(" 'W' : Start swinging");
  Serial.println(" 'B' : Bouncing in operation");
  Serial.println(" 'O' : Off, stop the motor immediately");
  Serial.println(" 'H' : Go back to 0 position from the current position.");
  Serial.println(" 'L' : Prints the current position/location of the motor.");
  Serial.println(" 'D' : Extend the mini pantograph");
  Serial.println(" 'U' : Retract the mini pantograph");
  Serial.println(" 'Q' : Setpoint distance in cm");
  Serial.println(" 'N' : Stepper2 control using digital encoder");

}

//============================  end  ==================================
```

3- Vibration and Temperature Measurements

```
   // ========================= Libraries ================================

 #include <NewPing.h>                          // Ultrasonic sensor library
 #include <Wire.h>                             // Serial communication library
 #include <LiquidCrystal_I2C.h>                // LCD 20X4 library
 #include <Adafruit_MLX90614.h>                // Temperature sensor library
 #include <BMA220.h>                           // Digital accelerometer
 #include <math.h>

// ====================== Define connections ============================

 #define D4_pin 4
 #define D5_pin 5
 #define triggerPin 9
// Digital output, to send ultrasonic signal,HC-SR04 trig.
 #define echoPin 10
// Digital input, to receive the ultra sonic signal, HC-SR04 echo.

// ====================== Define variables ==============================
int RawMin = 0;  // initialize minimum and maximum Raw Ranges for each axis
int RawMax = 652;                              // 3.3 V * (1023/5)
const int sampleSize = 10;      // Take multiple samples to reduce noise

int dt=1000;

int maxDistance = 400 ;// Maximum distance we want to ping for (in
centimeters).
float duration, distance;

bool runAllowed,scanningAllowed, trackingAllowed, swingingAllowed = false;
// booleans for new data from serial, runallowed, trackingAllowed,
bouncingAllowed flags

// =================  Define functions prototype ====================

 NewPing sonar(triggerPin, echoPin, maxDistance);
  // NewPing setup of pins and maximum distance.

 LiquidCrystal_I2C lcd(0x27,20,4);
 // set the LCD address to 0x27 for a 16 chars and 2 line display

 Adafruit_MLX90614 mlx = Adafruit_MLX90614();
   // Infra Red temprature sensor

// BMA220 bma;
    // Triple axis accelerometer BMA 220, digital vibration sensor

// ========================== Setup function ============================

 void setup()
 {

    Serial.begin(115200);       // Begin serial communication (Baud rate)

//    analogReference(EXTERNAL);
 // as we have connected 3.3V to the AREF pin on Arduino

   // pinMode(ledPin, OUTPUT);

   // attachInterrupt(1,reverseSwitch,RISING);
```

```
    lcd.init();                              // initialize the lcd
    lcd.backlight();                        // Print a message to the LCD.
    mlx.begin();                           // Infra Red temprature sensor
}
// ========================================================================
//*************************  Mian Loop  ****************************
// ========================================================================

 void loop()
 {

      Infra_Red_temprature_sensor();
      vibration();
 }


//========================================================================

void Infra_Red_temprature_sensor()
{

  Serial.print("Ambient = ");
  Serial.print(mlx.readAmbientTempC());
  Serial.print("*C\tObject = ");
  Serial.print(mlx.readObjectTempC());
  Serial.println("*C");
  Serial.println();

  lcd.setCursor(0,1);
  lcd.print("Ambient T = ");
  lcd.setCursor(12,1);
  lcd.print(mlx.readAmbientTempC());
  lcd.setCursor(19,1);
  lcd.print("C");

  lcd.setCursor(0,2);
  lcd.print("Object T  = ");
  lcd.setCursor(12,2);
  lcd.print(mlx.readObjectTempC());
  lcd.setCursor(19,2);
  lcd.print("C");

  delay(dt);

 }

//========================================================================

void vibration()
{
  const int xInput = A1;
  const int yInput = A2;
  const int zInput = A3;

  int xRaw = ReadAxis(xInput)  // Read raw values
  int yRaw = ReadAxis(yInput);
  int zRaw = ReadAxis(zInput); //ReadAxis()instead of analogRead()function.


  long xScaled = map(xRaw, RawMin, RawMax, -3000, 3000);
  // Convert raw values to 'milli-Gs" (0-650 to -3g - +3g), sensor's output
     voltage(3.3V)*ADC resolution(1023/5)=650
  long yScaled = map(yRaw, RawMin, RawMax, -3000, 3000);
```

```
  // When the sensor outputs 0 volts on x-axis i.e. xRaw=0, the map()
     function will return -3000 representing -3g.
  long zScaled = map(zRaw, RawMin, RawMax, -3000, 3000);
  // When the sensor outputs 3.3 volts on x-axis i.e. xRaw=1023, the map()
     function will return 3000 representing +3g.

  // When the sensor outputs 1.65 volts on x-axis i.e. xRaw=511, the map()
     function will return 0 representing 0g.

  float xAccel = xScaled / 1000.0*9.8;
  // the sensor's output is scaled down to fractional Gs by dividing it by
     1000 and displayed on the serial monitor, then multiplied by 9.8 mm/s.
  float yAccel = yScaled / 1000.0*9.8;
  float zAccel = zScaled / 1000.0*9.8;

  Serial.print("X, Y, Z  :: ");
  Serial.print(xRaw);
  Serial.print(", ");
  Serial.print(yRaw);
  Serial.print(", ");
  Serial.print(zRaw);
  Serial.print(" :: ");
  Serial.print(xAccel,0);
  Serial.print("mm/s2, ");
  Serial.print(yAccel,0);
  Serial.print("mm/s2, ");
  Serial.print(zAccel,0);
  Serial.print("mm/s2");

  Serial.print("\t");

  lcd.setCursor(0,0);
  lcd.print("X = ");
  lcd.setCursor(4,0);
  lcd.print(xAccel);
  lcd.setCursor(8,0);
  lcd.print("mm/s2");

  lcd.setCursor(0,1);
  lcd.print("y = ");
  lcd.setCursor(4,1);
  lcd.print(yAccel);
  lcd.setCursor(8,1);
  lcd.print("mm/s2");

  lcd.setCursor(0,2);
  lcd.print("z = ");
  lcd.setCursor(4,2);
  lcd.print(zAccel);
  lcd.setCursor(8,2);
  lcd.print("mm/s2");

  delay(dt);
}

// end ================================================================
```