```
#include "HX711.h"

// CHANGE CALIBRATION CONDITIONS BASED ON BEST FIT LINE!!!
// For example, best fit equation is: y = mx + b.   (where x is raw sensor value, y is the value in grams
double calibration_YIntercept = 272700; // TODO: Need to define best fit y intercept
double calibration_Slope = -432.7;   // TODO: Need to define best fit slope
//-----------------------------------------------------------------------------

// TODO: -Show connecting potentiometer to adjust Y_intercept
//       -Button to show slope
//       -Show state machines to support calibration steps

const double NEXT_WEIGHT = 0.1; // how much to weigh the next sample as part of the moving average
const int DELAY_MS = 10; //how many milliseconds between samples

const bool STRICTLY_INCREASING = true;

// HX711 circuit wiring
const int LOADCELL_DOUT_PIN = 4;
const int LOADCELL_SCK_PIN = 5;

// Ultrasonic Range Sensor
const int ECHO_PIN = 2;
const int TRIG_PIN = 3;

double load, distance;
double prevDistance = 0.0;

typedef enum {START, WAIT_TILL_READY1, CALIBRATE_BASELINE, WAIT_TILL_READY2, CALIBRATE_1KG, WAIT_TILL_READY3, COLLECT_DATA, FINISHED} SystemState;
SystemState sysState = START;

int incomingByte = 0;

//Object representing load cell
HX711 scale;
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,20,4);
```

```
void setup() {

  //Define BAUD rate (Async. Communication Speed)
  Serial.begin(9600);

  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);

  pinMode(TRIG_PIN, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(ECHO_PIN, INPUT); // Sets the echoPin as an INPUT

  //Display flie first row in CSV data standard (comma deliminator)
  //Serial.println("Distance (in cm), Load (in g), ");
  lcd.init();                      // initialize the lcd
  lcd.init();



}


void ticFunc(){
  // Print a message to the LCD.
  lcd.backlight();

  //State Transitions
  switch(sysState){
      case START:{
        sysState = WAIT_TILL_READY1;
        break;
      }
      case WAIT_TILL_READY1:{
        sysState = CALIBRATE_BASELINE;
        break;
      }
      case CALIBRATE_BASELINE:{
        sysState = WAIT_TILL_READY2;
        break;
      }
      case WAIT_TILL_READY2:
        sysState = CALIBRATE_1KG;
        break;
      case CALIBRATE_1KG:{
        sysState = WAIT_TILL_READY3;
        break;
      }
     case WAIT_TILL_READY3:
        sysState = COLLECT_DATA;
        break;

  }

  //State Actions
  switch(sysState){

    case WAIT_TILL_READY1: {
        Serial.println("CALIBRATE:  PULL  ROPE  SLIGHTLY  UNTIL  MATERIAL  IS  BAREL Y  IN
TENSION");
        lcd.setCursor(1,0);
        lcd.print("PULL ROPE");
        waitTillReady();


        break;
    }
    case CALIBRATE_BASELINE:{
         distance = getDistance();
         load = getLoad();

        // Baseline values
        Serial.println("Calibrating.... Continue to hold material in place....");
        lcd.setCursor(1,0);
        lcd.print("Calibrating...");
        lcd.setCursor(1,1);
        lcd.print("Hold material");

        for(int i=0;i<20;i++){
          updateDistance();
          updateLoad();
        }
        double baseDistance = distance;
        calibration_YIntercept = load;
        Serial.print("ACCEPTED BASELINE VALUES: ");
        Serial.print("Load (raw): ");
        Serial.print(calibration_YIntercept);
        Serial.print(" , distance (cm): ");
        Serial.println(baseDistance);
        waitTillReady();

        break;
    }
    case WAIT_TILL_READY2: {
        Serial.println("CALIBRATE: CONTINUE KEEPING MATERIAL IN TENSION, PLACE 1kg ON
BED");
        lcd.setCursor(1,0);
        lcd.print("PLACE 1kg.....");
        lcd.setCursor(1,1);
        lcd.print("Press key.....");
        waitTillReady();
        break;
    }
    case CALIBRATE_1KG:{
        load = getLoad();
        distance = getDistance();
        Serial.println("Calibrating.... Continue to hold material in place....");
        for(int i=0;i<20;i++){
          updateDistance();
          updateLoad();
        }
        double deltaY = (load - calibration_YIntercept) ;
        double deltaX = 1000; //1kg
        double slope = deltaY / deltaX;
        calibration_Slope = slope;
        Serial.print("Load: ");
        Serial.println(load);
        Serial.print("Slope: ");
        Serial.println(slope);
        break;
    }
    case WAIT_TILL_READY3:{
      Serial.println("Remove 1kg, then pull rope very slowly to start collecting stress-strain
curve....");
      lcd.setCursor(1,0);
      lcd.print("Remove 1kg.....");
      lcd.setCursor(1,1);
      lcd.print("Press key.....");
      waitTillReady();
      break;
    }
    case COLLECT_DATA:{
       load = getLoad();
       distance = getDistance();
       for(int i=0;i<10;i++){
        updateDistance();
        updateLoad();
       }

      if ( !STRICTLY_INCREASING || distance > prevDistance){
        //x =(y-b) /m
        double x = (load - calibration_YIntercept) / calibration_Slope;
        //Serial.print("Grams: ");
        //Serial.println(x);
        printVals();
        prevDistance = distance;
      }
      break;
    }

  }

}


void loop() {

  ticFunc();
  delay(DELAY_MS);
}


void printVals(){
  Serial.print(distance);
  Serial.print("\t");
  Serial.println( convertLoadToGrams(load) );
```

```
          lcd.setCursor(1,0);
          lcd.print("Distance: ");
          lcd.print(distance);
          lcd.setCursor(1,1);
          lcd.print("Grams: ");
          lcd.print(convertLoadToGrams(load));
}


//Rolling average of distance (10% weight for each new sample)
void updateDistance(){
   double currDist = getDistance();
   distance = ((1.0-NEXT_WEIGHT)*distance)+(NEXT_WEIGHT*currDist);
}


//Returns the current distance of the ultrasonic range sensor
double getDistance(){

   digitalWrite(TRIG_PIN, LOW);
   delayMicroseconds(2);

   // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
   digitalWrite(TRIG_PIN, HIGH);
   delayMicroseconds(10);
   digitalWrite(TRIG_PIN, LOW);

   // Reads the echoPin, returns the sound wave travel time in microseconds
   double duration = pulseIn(ECHO_PIN, HIGH);

   // Calculating the distance
   double dist = (duration * 0.034) / 2.0; // Speed of sound wave divided by 2 (go and back)

   return dist;
}


double updateLoad(){
   double currLoad = getLoad();
   load = ((1.0-NEXT_WEIGHT)*load)+(NEXT_WEIGHT*currLoad);
```

```
}


void waitTillReady(){
    String str = "";
    Serial.println("WHEN READY.... PRESS ANY KEY....");
    while( str.equals("") )    {
        str = Serial.readString();// read the incoming data as string
        //Serial.println(str);
    }
}

double convertLoadToGrams(double l){
        //x =(y-b) /m
        double x = (load - calibration_YIntercept) / calibration_Slope;
        return x;
}

double getLoad(){
    double avg = scale.read();

    //double diff = avg - calibration_YIntercept;
    //double grams = diff / calibration_Slope;

    //y = mx + b (where y is the value in grams and x is the raw value)
    //    double grams = (calibration_Slope)*avg + calibration_YIntercept;

    return avg;
    //return grams;
}
```