ONDOKUZ MAYIS ÜNİVERSİTESİ

ELEKTRİK - ELEKTRONİK MÜHENDİSLİĞİ

Algılayıcılar ve Dönüştürücüler

Konu:Yük Hücresi Nedir ? Yük Hücresinden gelen verilerin Okunması

Hazırlayanlar:

BATUHAN TOPAL / 17060071

Caner ADSOY / 17060032

MUHAMMET MERT ÇAKIR /17060101

ÖMER FARUK BOZ / 17060057

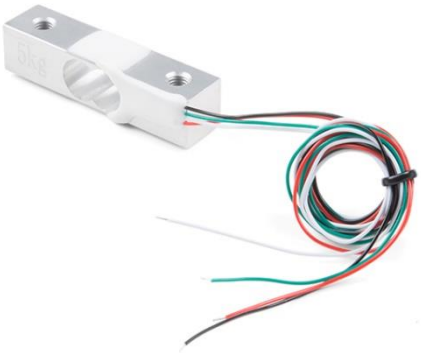
Ders Sorumlusu : Dr. Öğretim Üyesi İLYAS EMİNOĞLU

Samsun

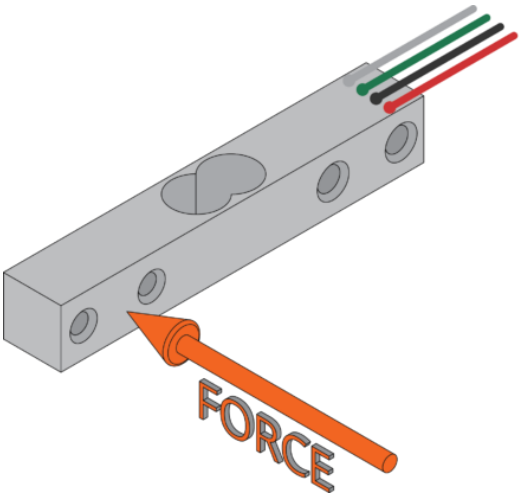
2021

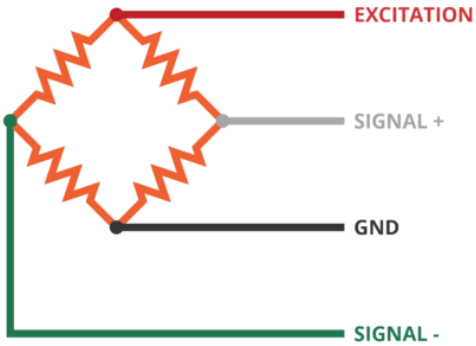
Yük Hücresi Nedir ?

Yük hücresi, bir kuvveti elektrik sinyali hâline dönüştürmek için kullanılan dönüştürücüdür. Bu dönüşüm dolaylı ve iki aşamada olur. Mekanik bir düzenleme ile, algılanan kuvvet bir gerinim ölçerin şeklini değiştirir. Gerinim ölçer şekil değişikliğini (gerinim) bir elektrik sinyali olarak ölçer. Çünkü gerinim, telin etkin elektriksel direncini değiştirir.



Yük hücreleri, esaslı ağırlık göstergeleri kısaca **elektronik teraziler** zemberekli terazilere benzerler. Yay elemanı olarak **çelik ve alüminyum** kullanılır. Yük hücrelerinde **4 adet gerilme ölçer** kullanılır. Bu gerilme ölçerler **Wheatstone köprüsü** şeklinde oluşturulurlar. Köprüye bir gerilim uygulandığında çıkış gerilimi uygulanan yüke orantılı bir gerilim olur. Bu oluşan küçük gerilim sayesinde uygulanan basıncın ağırlığı ölçülür. Yük hücrelerinde 2 giriş ve 2 çıkış olmak üzere toplam **4 uç bulunur**. Bazı yük hücrelerinde **toplam uç sayısı 6 olabilmektedir**. 6 uçlu yük hücrelerinde giriş uçlarına 2 adet uç daha bağlanmıştır. Bu bağlanan ek uçlar **+duyu(+sense)** ve **-duyu(-sense)** olarak isimlendirilir. Amaçları ise yük hücrelerinde **herhangi bir kopukluk** meydana gelip gelmediğini saptamaktır.





Yük Hücresi Modelleri

Yük hücreleri kullanım alanlarının gerektirdiği şekilde imal edilirler bu yüzden farklı ve çeşitli modelde yük hücresine rastlanılır. Günümüzde 50-100gr dan 1000-2000ton a kadar geniş bir kapasite aralığında yük hücresi imal edilebilmektedir.

1- Lama Tipi Yük Hücreleri

Lama tipi yük hücreleri, eğme kuvveti prensibi ile çalışan elektronik ağırlık ve kuvvet ölçme uygulamalarında, endüstriyel ortamlarda kullanılmak üzere geliştirilmiştir. Çoğunlukla düşük kapasitelerdeki tank tartım sistemleri, platform kantarları, torbalama ve dozajlama makineleri ve bant kantarlarında kullanılır. Tamamen paslanmaz çelik olan gövdesindeki ölçüm bölgesinin körük malzeme ile kaynakla kapatılmış olması, çok iyi geçirmezlik sağlar.

BB Yük Hücresi

20-50-100-200 kg

BS Yük Hücresi

200-500-1000-2000 kg

SBS Yük Hücresi

200-500-1000-2000 kg

2- S Tipi Yük Hücreleri

Ağırlık kapasitesi yaklaşık olarak **500 kg - 2 Ton** olan yerler için kullanılır. Kesme kuvveti prensibi ile basma ve çekme yönünde çalışan kuvvet ölçüm uygulamaları için geliştirilmiştir. Özellikle bant kantarlarında kullanılırlar.

STCS Yük Hücresi

50-100-200-300 kg

TB Yük Hücresi

500-1000-2000-3000-5000 kg

3- Platform Tipi Yük Hücreleri

Düşük kapasiteli ve tek yük hücreli platform kanatlarında kullanılırlar. Çalışma ağırlıkları yaklaşık olarak **6 kg - 600 kg** civarındadır.

SSP Yük Hücresi

20-40-80-120 kg

SP Yük Hücresi

200-500-1000 kg

SPA 3-50 Yük Hücresi

3-6-10-20-30-50 kg

SPA 100-350 Yük Hücresi

100-200-350 kg

SPA 600 Yük Hücresi

600 kg

MSP 8-100 Yük Hücresi

8-15-20-30-40-60-80-100 kg

4- Bası Tipi Yük Hücreleri

Kesme kuvveti ile basma yönünde çalışan **yüksek hassasiyet ve yüksek kapasiteli** yerler için geliştirilmiştir. Genellikle tank ve vagon tartım işlemlerinde kullanılırlar. Çalışma ağırlıkları **10 Ton - 200 Ton** aralığındadır.

CAD Dijital Yük Hücresi

10-20-30 t

SC Yük Hücresi

10-20 t

SC-V Yük Hücresi

10-20 t

CD Yük Hücresi

25-30-40-60-100 t

CA Yük Hücresi

10-20-25 t

CA-V Yük Hücresi

10-20-25 t

5- Özel Yük Hücreleri

5.1- PLC Yük Hücresi (2-5-10-15-20-30-60 t Taşıma Kapasitesi)

PLC yük hücreleri, vinç sistemleri için geliştirilmiş pim tipi yük hücresidir. Vinçlerin kaldırma mekanizmasında bulunan denge makarasına pim olarak takılır ve makaraya gelen yükü algılar.

Alaşım çeliğinden gövdesi korozyona uzun süreler dayanabilen özel boya ile kaplanmış, ölçüm bölgesi ise paslanmaz çelik kapakla kaynaklı olarak kapatılmış olan PLC yük hücreleri, IP68 sınıfında suya karşı tam korumalıdır. Vinç ve kaldırma makinelerinde aşırı yük kontrolü veya kaldırılan yükün ağırlığını ölçme amacı ile kullanılır.

5.2- WTB Yük Hücresi (4500 kg Taşıma Kapasitesi)

WTB yük hücreleri, asansör ve vinç sistemleri için geliştirilmiş halat tipi yük hücresidir. Kaldırma mekanizmasında bulunan hareketsiz çelik halatlara takılır ve halata gelen yükü algılar.

Paslanmaz çelik gövdeli WTB yük hücreleri, kaynakla kapatılmış IP 68 koruma sınıfındadır, toz, su ve diğer kimyasallardan etkilenmez.

Asansör ve vinç gibi kaldırma makinelerinde aşırı yük kontrolü veya kaldırılan yükün ağırlığını ölçme amacı ile kullanılır

5.3- PFT Yük Hücresi (100 kg Taşıma Kapasitesi)

PFT, özel uygulamalar için geliştirilmiş rulo tipi yük hücresidir. Sarma sistemlerde bulunan rulonun miline takılarak merdane üzerine gelen yükü algılar.

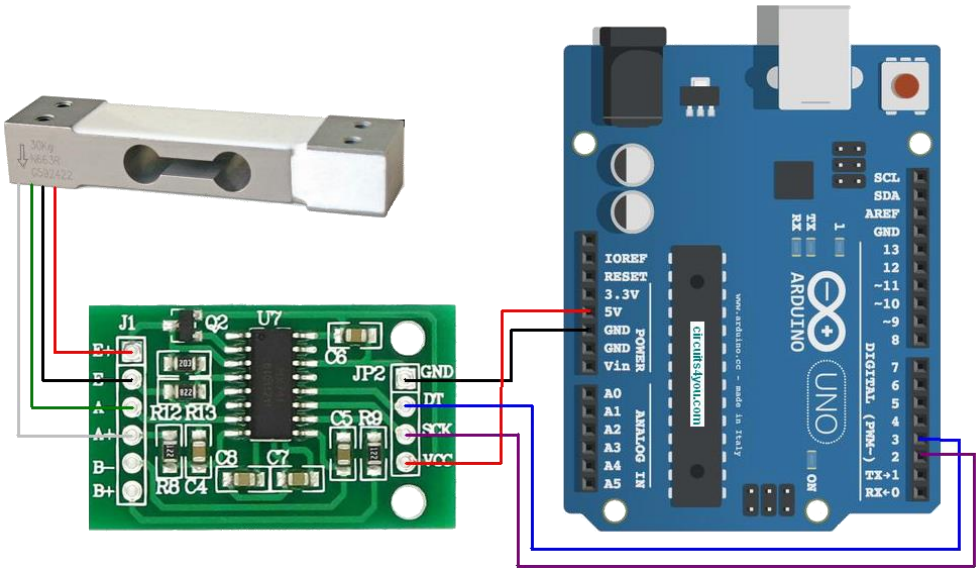
Alaşım çeliğinden gövdesi korozyona dayanıklı özel boya ile kaplanmış olan PFT yük hücreleri IP55 sınıfında toza ve suya karşı korumalıdır. PFT yük hücresi, içine yerleştirilen rulman ile kolaylıkla monte edilir. Ruloya sarılan malzemenin gerginliğini algılamak ve ayarlamakta kullanılır.

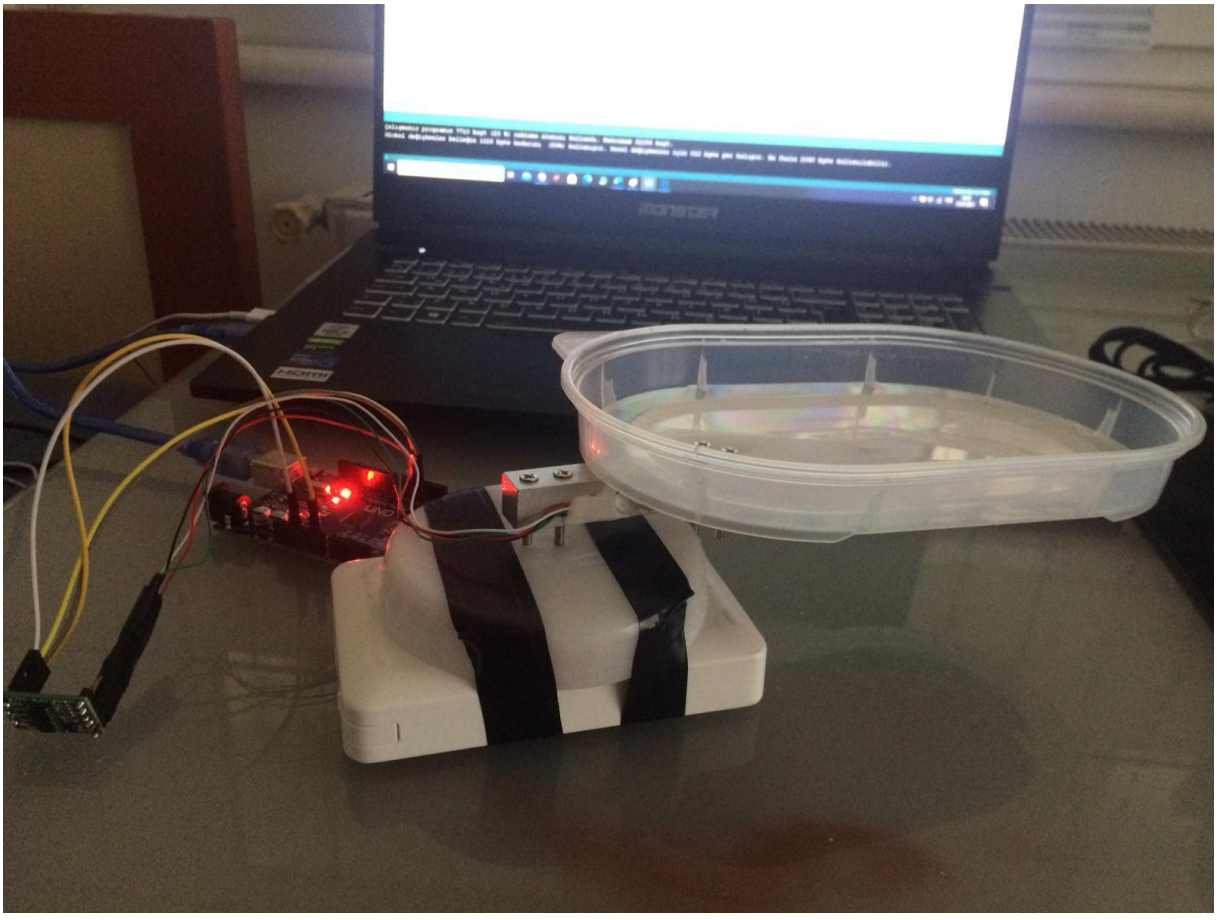
Yük Hücresinden Gelen Verilerin Okunması

Yük Hücresi Teknik Özellikleri

- Yük Hücresi Kapasitesi: 5 kg
- Ayırıcı Giriş Gerilimi: ± 40mV
- Veri Hassasiyeti: 24 bit (24 bit A / D Çevirici Entegre)
- Yenileme Frekansı: 80 Hz
- Çalışma Gerilimi: 4.8 - 5.5V
- Çıkış gerilim aralığı: 2.6V ~ 5.5V
- Çalışma akımı: <10 mA
- Boyutu: 34.5mm x 20.5mm x 1.1mm

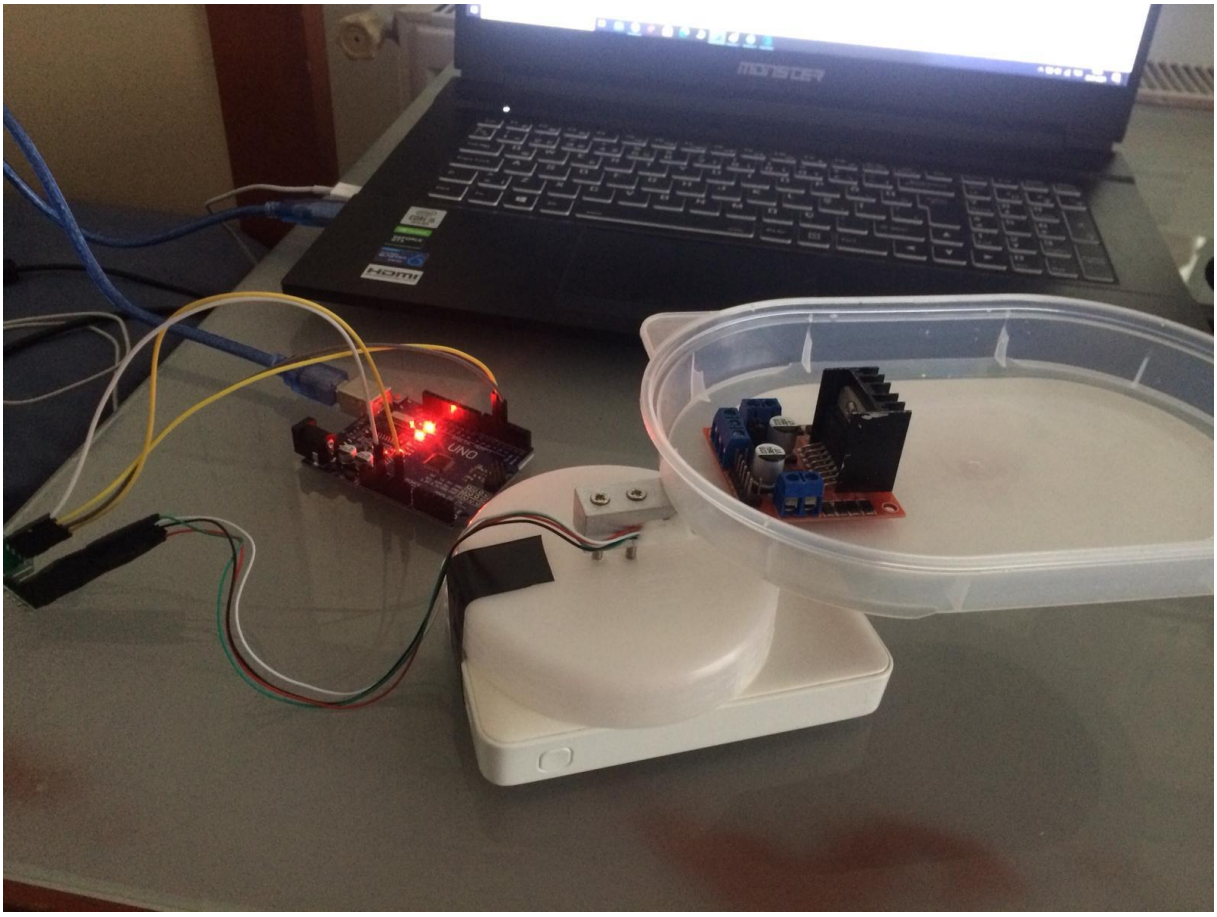
Arduino Bağlantı Şeması

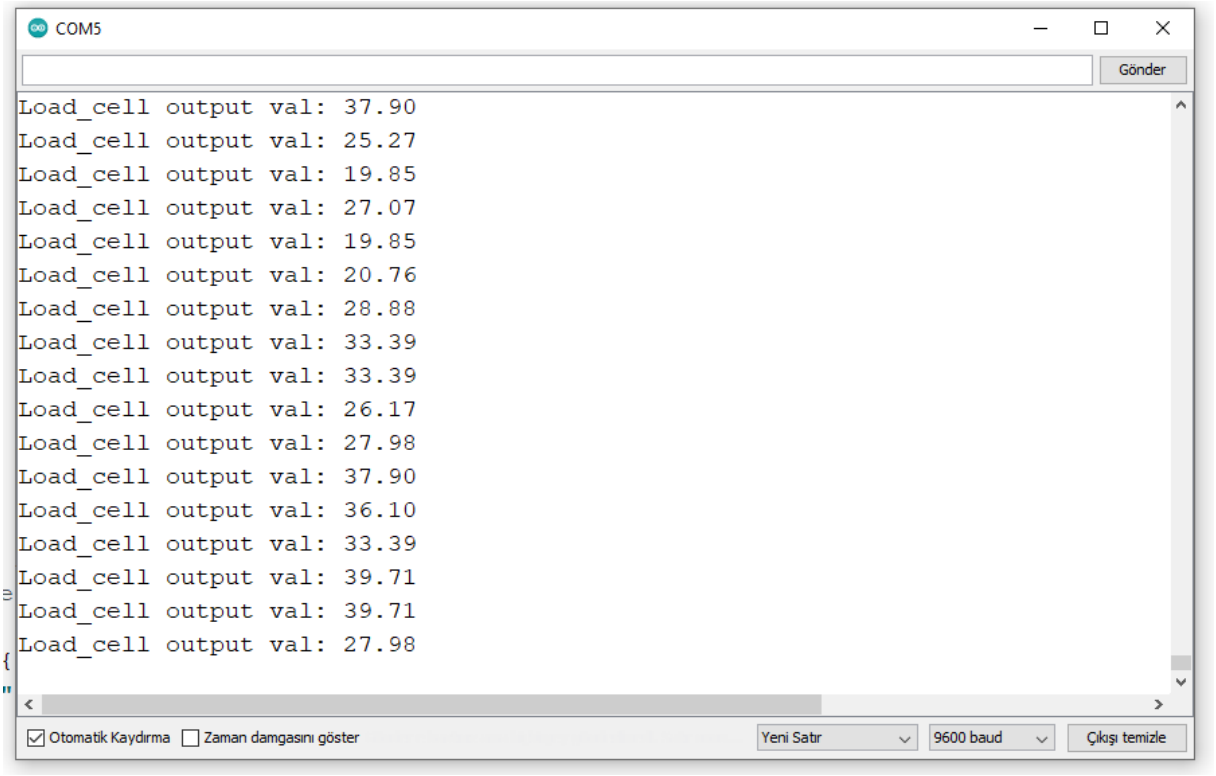




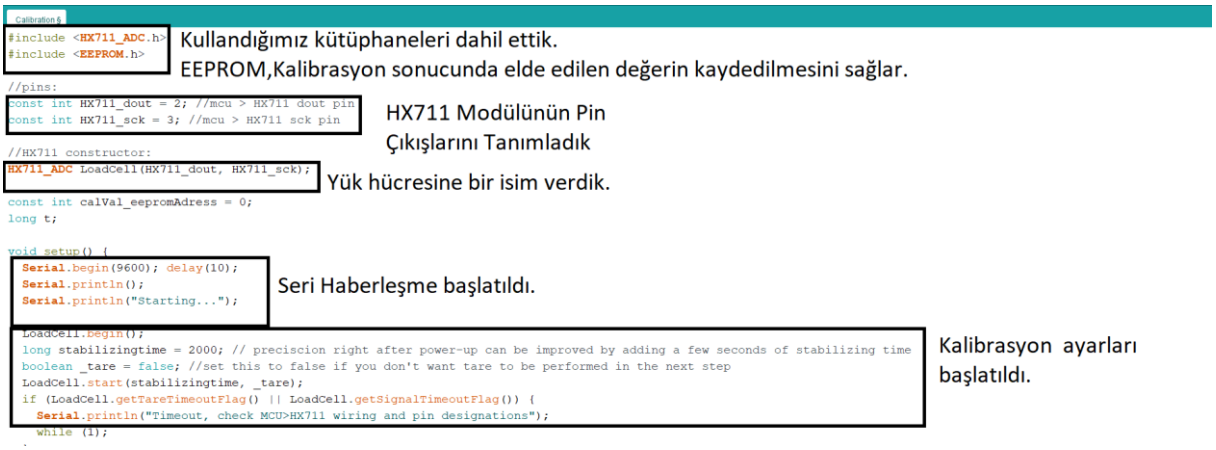
Üzerine Ağırlığını bildiğimiz bir nesne koyalım…

L298N Motor Sürücü = 37 gr





Arduino Kodları




```
}
 else {
  LoadCell.setCalFactor(1.0); // user set calibration value (float), initial value 1.0 may be used for this sketch
  Serial.println("Startup is complete");
 }
 while (!LoadCell.update());
 calibrate(); //start calibration procedure
}
```

```
void loop() {
  static boolean newDataReady = 0;
  const int serialPrintInterval = 0; //increase value to slow down serial print activity

  // check for new data/start next conversion:
  if (LoadCell.update()) newDataReady = true;

  // get smoothed value from the dataset:
  if (newDataReady) {
    if (millis() > t + serialPrintInterval) {
      float i = LoadCell.getData();
      Serial.print("Load_cell output val: ");
      Serial.println(i);
      newDataReady = 0;
      t = millis();
    }
  }

  // receive command from serial terminal
  if (Serial.available() > 0) {
    float i;
    char inByte = Serial.read();
    if (inByte == 't') LoadCell.tareNoDelay(); //tare
    else if (inByte == 'r') calibrate(); //calibrate
    else if (inByte == 'c') changeSavedCalFactor(); //edit calibration value manually
  }

  // check if last tare operation is complete
  if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
  }
```

```
}

void calibrate() {
  Serial.println("***");
  Serial.println("Start calibration:");
  Serial.println("Place the load cell an a level stable surface.");
  Serial.println("Remove any load applied to the load cell.");
  Serial.println("Send 't' from serial monitor to set the tare offset.");

  boolean _resume = false;
  while (_resume == false) {
    LoadCell.update();
    if (Serial.available() > 0) {
      if (Serial.available() > 0) {
        float i;
        char inByte = Serial.read();
        if (inByte == 't') LoadCell.tareNoDelay();
      }
    }
    if (LoadCell.getTareStatus() == true) {
      Serial.println("Tare complete");
      _resume = true;
    }
  }

  Serial.println("Now, place your known mass on the loadcell.");
  Serial.println("Then send the weight of this mass (i.e. 100.0) from serial monitor.");

  float known_mass = 0;
  _resume = false;
  while (_resume == false) {
    LoadCell.update();
    if (Serial.available() > 0) {
```

```
    known_mass = Serial.parseFloat();
    if (known_mass != 0) {
      Serial.print("Known mass is: ");
      Serial.println(known_mass);
      _resume = true;
    }
  }
 }

 LoadCell.refreshDataSet(); //refresh the dataset to be sure that the known mass is measured correct
 float newCalibrationValue = LoadCell.getNewCalibration(known_mass); //get the new calibration value

 Serial.print("New calibration value has been set to: ");
 Serial.print(newCalibrationValue);
 Serial.println(", use this as calibration value (calFactor) in your project sketch.");
 Serial.print("Save this value to EEPROM adress ");
 Serial.print(calVal_eepromAdress);
 Serial.println("? y/n");

 _resume = false;
 while (_resume == false) {
   if (Serial.available() > 0) {
     char inByte = Serial.read();
     if (inByte == 'y') {
#if defined(ESP8266)|| defined(ESP32)
       EEPROM.begin(512);
#endif
       EEPROM.put(calVal_eepromAdress, newCalibrationValue);
#if defined(ESP8266)|| defined(ESP32)
       EEPROM.commit();
#endif
       EEPROM.get(calVal_eepromAdress, newCalibrationValue);
       Serial.print("Value ");
```

```
        Serial.print(newCalibrationValue);
        Serial.print(" saved to EEPROM address: ");
        Serial.println(calVal_eepromAdress);
        _resume = true;

      }
      else if (inByte == 'n') {
        Serial.println("Value not saved to EEPROM");
        _resume = true;
      }
    }
  }

  Serial.println("End calibration");
  Serial.println("***");
  Serial.println("To re-calibrate, send 'r' from serial monitor.");
  Serial.println("For manual edit of the calibration value, send 'c' from serial monitor.");
  Serial.println("***");
}

void changeSavedCalFactor() {
  float oldCalibrationValue = LoadCell.getCalFactor();
  boolean _resume = false;
  Serial.println("***");
  Serial.print("Current value is: ");
  Serial.println(oldCalibrationValue);
  Serial.println("Now, send the new value from serial monitor, i.e. 696.0");
  float newCalibrationValue;
  while (_resume == false) {
    if (Serial.available() > 0) {
      newCalibrationValue = Serial.parseFloat();
      if (newCalibrationValue != 0) {
        Serial.print("New calibration value is: ");
```

```
        Serial.println(newCalibrationValue);
        LoadCell.setCalFactor(newCalibrationValue);
        _resume = true;
      }
     }
    }
   _resume = false;
   Serial.print("Save this value to EEPROM adress ");
   Serial.print(calVal_eepromAdress);
   Serial.println("? y/n");
   while (_resume == false) {
    if (Serial.available() > 0) {
      char inByte = Serial.read();
      if (inByte == 'y') {
#if defined(ESP8266)|| defined(ESP32)
        EEPROM.begin(512);
#endif
        EEPROM.put(calVal_eepromAdress, newCalibrationValue);
#if defined(ESP8266)|| defined(ESP32)
        EEPROM.commit();
#endif
        EEPROM.get(calVal_eepromAdress, newCalibrationValue);
        Serial.print("Value ");
        Serial.print(newCalibrationValue);
        Serial.print(" saved to EEPROM address: ");
        Serial.println(calVal_eepromAdress);
        _resume = true;
      }
      else if (inByte == 'n') {
        Serial.println("Value not saved to EEPROM");
        _resume = true;
      }
    }
```

```
  }
  Serial.println("End change calibration value");
  Serial.println("***");
}
```