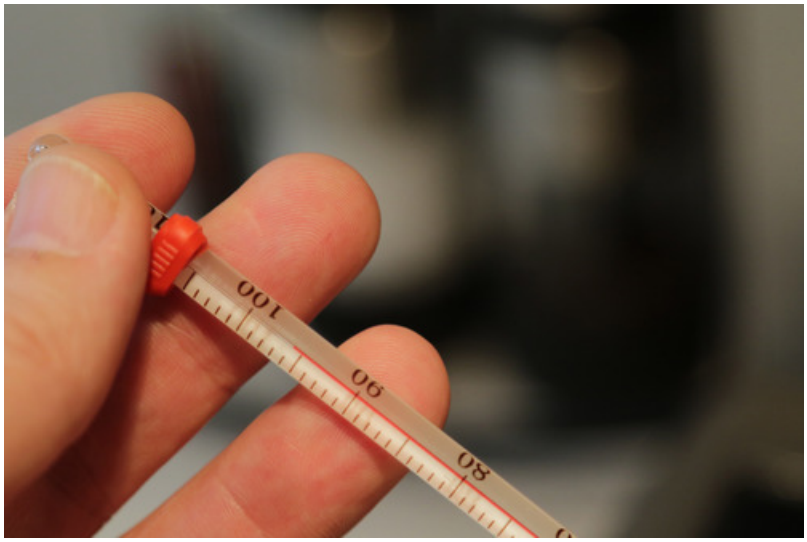
adafruit learning system

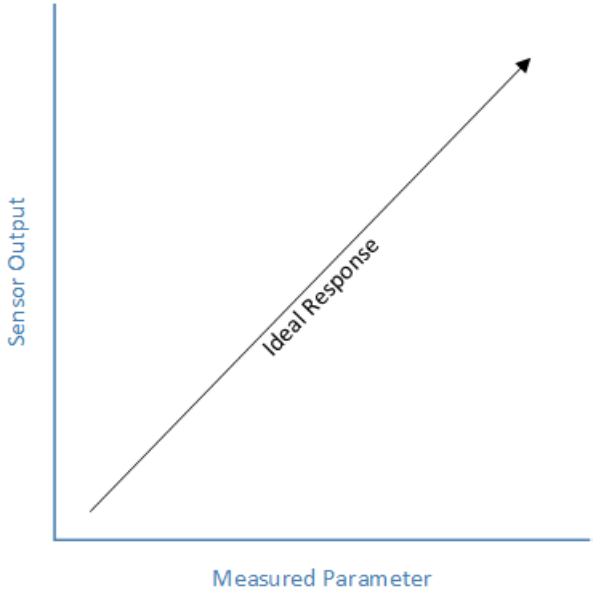
Calibrating Sensors

Created by Bill Earl



Last updated on 2020-03-21 11:34:02 PM UTC

Why Calibrate?



Why do we need to calibrate sensors?

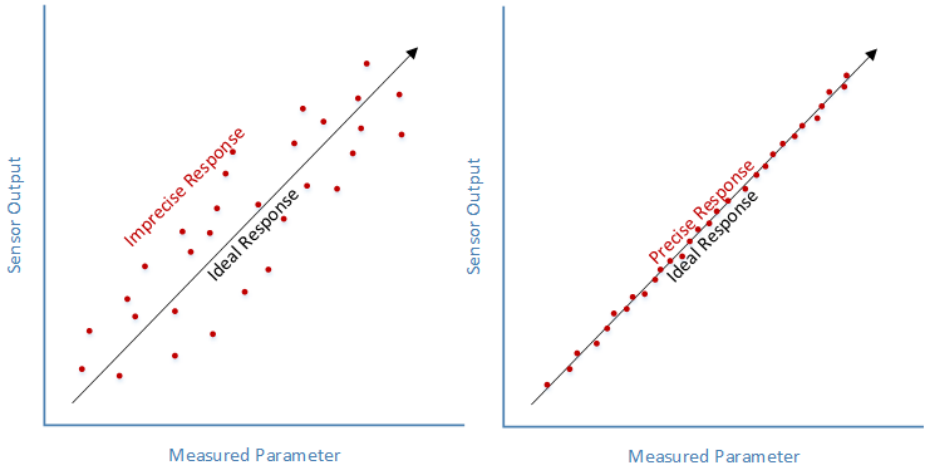
There are a lot of good sensors these days and many are 'good enough' out of the box for many non-critical applications. But in order to achieve the best possible accuracy, a sensor should be calibrated in the system where it will be used. This is because:

- **No sensor is perfect.**
  - Sample to sample manufacturing variations mean that even two sensors from the same manufacturer production run may yield slightly different readings.
  - Differences in sensor design mean two different sensors may respond differently in similar conditions. This is especially true of 'indirect' sensors that calculate a measurement based on one or more actual measurements of some different, but related parameter.
  - Sensors subject to heat, cold, shock, humidity etc. during storage, shipment and/or assembly may show a change in response.
  - Some sensor technologies 'age' and their response will naturally change over time - requiring periodic re-calibration.
- **The Sensor is only one component in the measurement system.** For example:
  - With analog sensors, your ADC is part of the measurement system and subject to variability as well.
  - Temperature measurements are subject to thermal gradients between the sensor and the measurement point.
  - Light and color sensors can be affected by spectral distribution, ambient light, specular reflections and other optical phenomena.
  - Inertial sensors almost always have some 'zero offset' error and are sensitive to alignment with the system being measured

What makes a good sensor?

The two most important characteristic of a sensor are:

- **Precision** - The ideal sensor will always produce the same output for the same input.
- **Resolution** - A good sensor will be able to reliably detect small changes in the measured parameter.

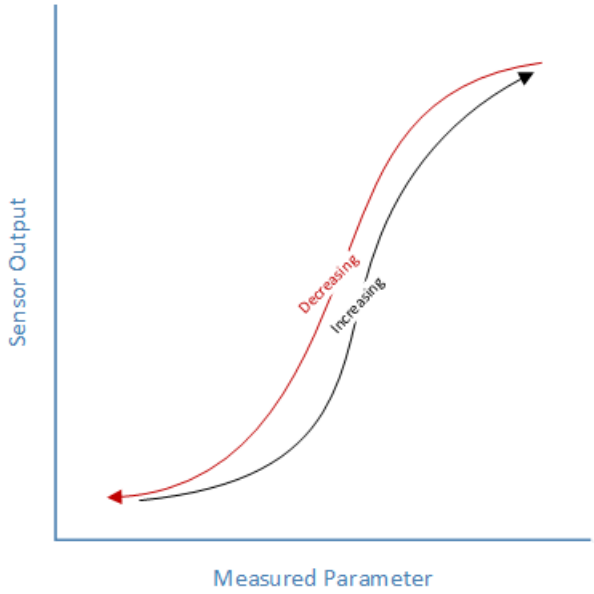


What affects precision?

- **Noise** - All measurement systems are subject to random noise to some degree. Measurement systems with a low Signal to Noise Ratio will have problems making repeatable measurements. In the diagrams above, the sensor on the right shows much better precision than the noisy one on the left.
- **Hysteresis** - Some types of sensors also exhibit hysteresis. The sensor will tend to read low with an increasing signal and high with a decreasing signal as shown in the graph below. Hysteresis is a common problem with many pressure sensors.

To paraphrase George Santayana:

"Those who ingnore hysteresis are doomed to unrepeatable results."

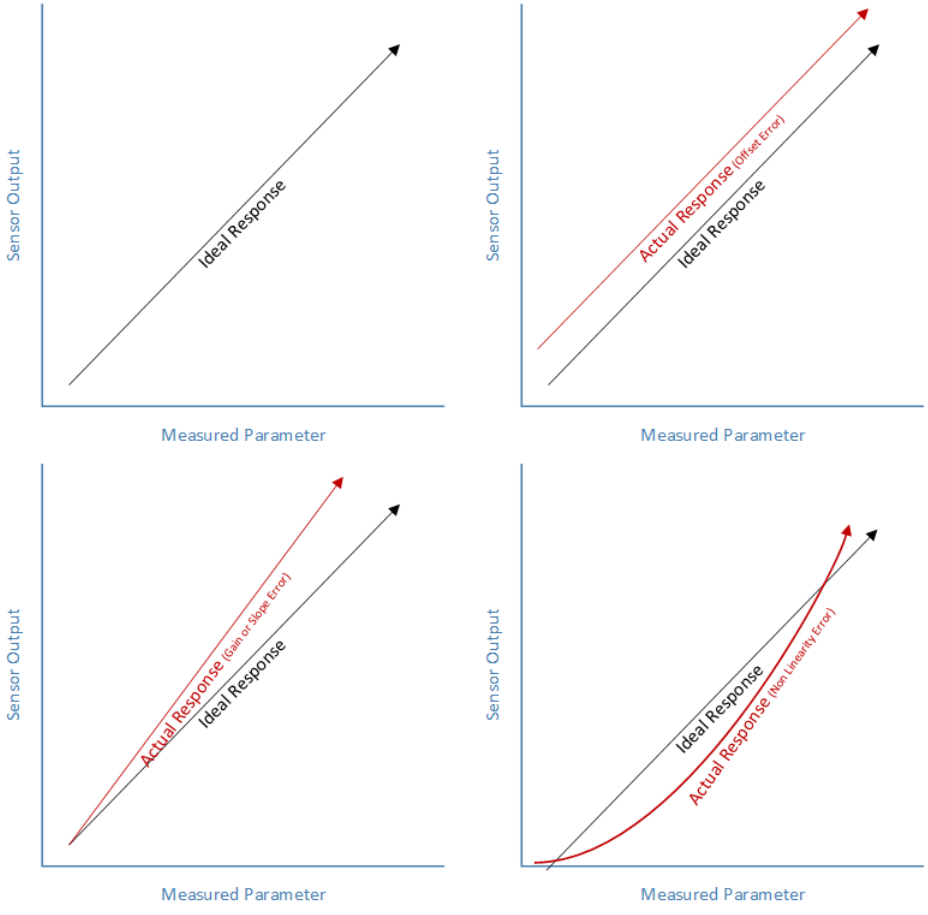


Are there any other important qualities in a sensor?

Precision and resolution are the real 'must have' qualities. But there are a couple of other 'nice-to-have' qualities:

Linearity - A sensor whose output is directly proportional to the input is said to be linear. This eliminates the need to do any complex curve-fitting and simplifies the calibration process.

Speed - All else being equal, a sensor that can produce precise readings faster is a good thing to have.



What about accuracy? Isn't accuracy the most important thing?

Accuracy is a combination of precision, resolution and calibration. **If you have a sensor that gives you repeatable measurements with good resolution, you can calibrate it for accuracy.**

What about digital sensors? Aren't they calibrated at the factory?

To some degree, yes. Nevertheless, digital sensors are still subject to manufacturing and operating condition variability. For critical measurements, you need to calibrate the total system.

- But the manufacturer's spec sheet says it is accurate to 0.00000001%

  And it probably is - when measured in their QA test fixture using their test procedures and according to their definition of 'accuracy'.

  Your mileage may vary!

So, How Do We Calibrate?

The first thing to decide is what your calibration reference will be.

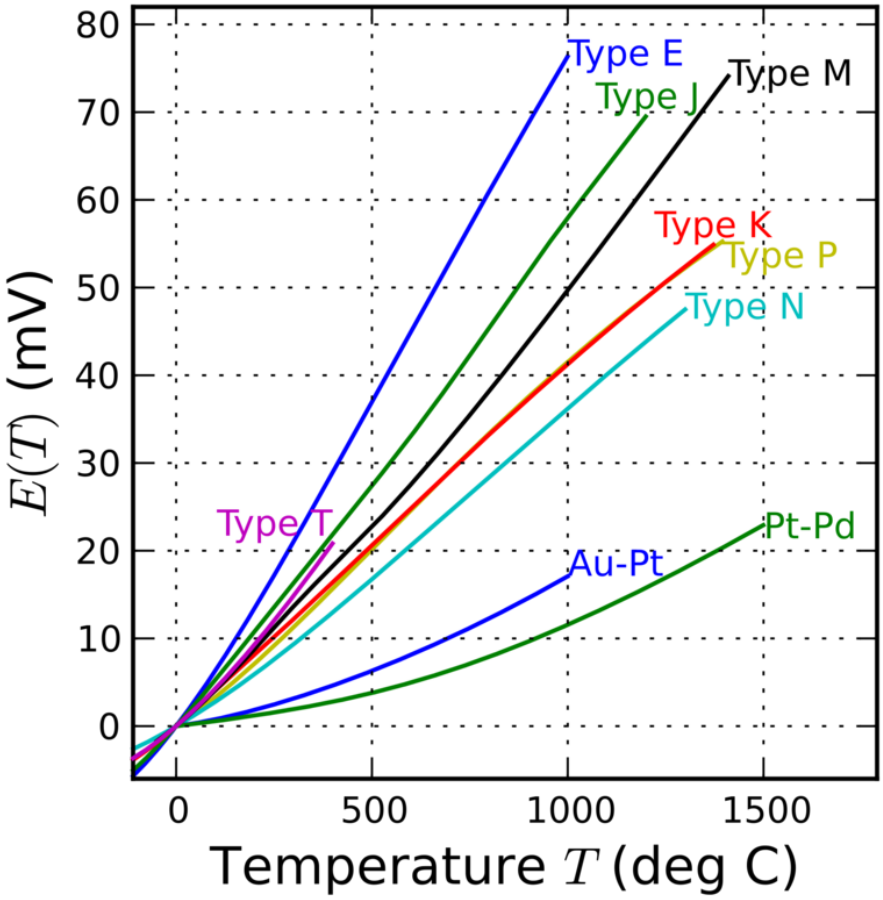
Standard References

If it is important to get accurate readings in some standard units, you will need a Standard Reference to calibrate against. This can be:

A calibrated sensor - If you have a sensor or instrument that is known to be accurate. It can be used to make reference readings for comparison. Most laboratories will have instruments that have been calibrated against NIST standards. These will have documentation including the specfic reference against which they were calibrated, as well as any correction factors that need to be applied to the output.

A standard physical reference - Reasonably accurate physical standards can be used as standard references for some types of sensors

- **Rangefinders**
  - Rulers and Meter sticks
- **Temperature Sensors**
  - Boiling Water - 100°C at sea-level
  - Ice-water Bath - The "Triple Point" of water is 0.01°C at sea-level
- **Accelerometers**
  - Gravity is a constant 1G on the surface of the earth.



Thermocouple Characteristic Curves

CC0 Public Domain via Wikimedia Commons (https://adafru.it/fcE)

The Characteristic Curve

Characteristic Curve – Each sensor will have a 'characteristic curve' that defines the sensor's response to an input. The calibration process maps the sensor's response to an ideal linear response. How to best accomplish that depends on the nature of the characteristic curve.

- **Offset** – An offset means that the sensor output is higher or lower than the ideal output. Offsets are easy to correct with a single-point calibration.
- **Sensitivity or Slope** – A difference in slope means that the sensor output changes at a different rate than the ideal. The Two-point calibration process can correct differences in slope.
- **Linearity** – Very few sensors have a completely linear characteristic curve. Some are linear enough over the measurement range that it is not a problem. But some sensors will require more complex calculations to linearize the output.

Calibration Methods

In the next few pages, we'll look at three different types of calibration:

- **One Point Calibration**
- **Two Point Calibration**
- **Multi-Point Curve Fitting**

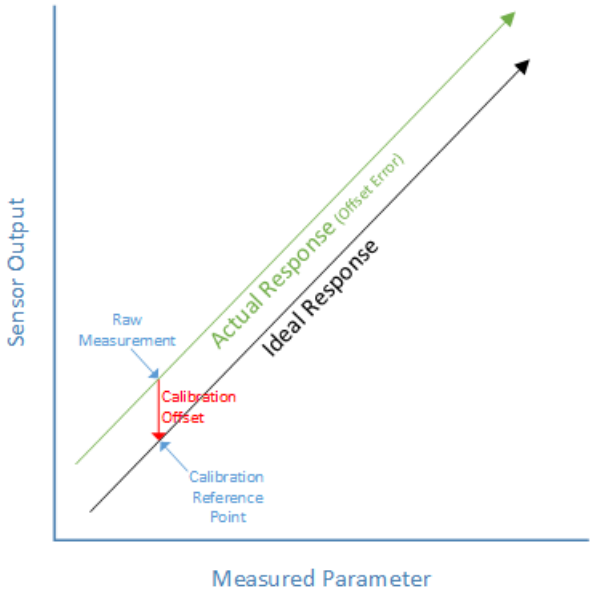
One Point Calibration

One point calibration is the simplest type of calibration. If your sensor output is already scaled to useful measurement units, a one point calibration can be used to correct for sensor offset errors in the following cases:

- **Only one measurement point is needed.** If you have an application that only requires accurate measurement of a single level, there is no need to worry about the rest of the measurement range. An example might be a temperature control system that needs to maintain the same temperature continuously.
- **The sensor is known to be linear and have the correct slope over the desired measurement range.** In this case, it is only necessary to calibrate one point in the measurement range and adjust the offset if necessary. Many temperature sensors are good candidates for one-point calibration.

A one point calibration can also be used as a "drift check" to detect changes in response and/or deterioration in sensor performance.

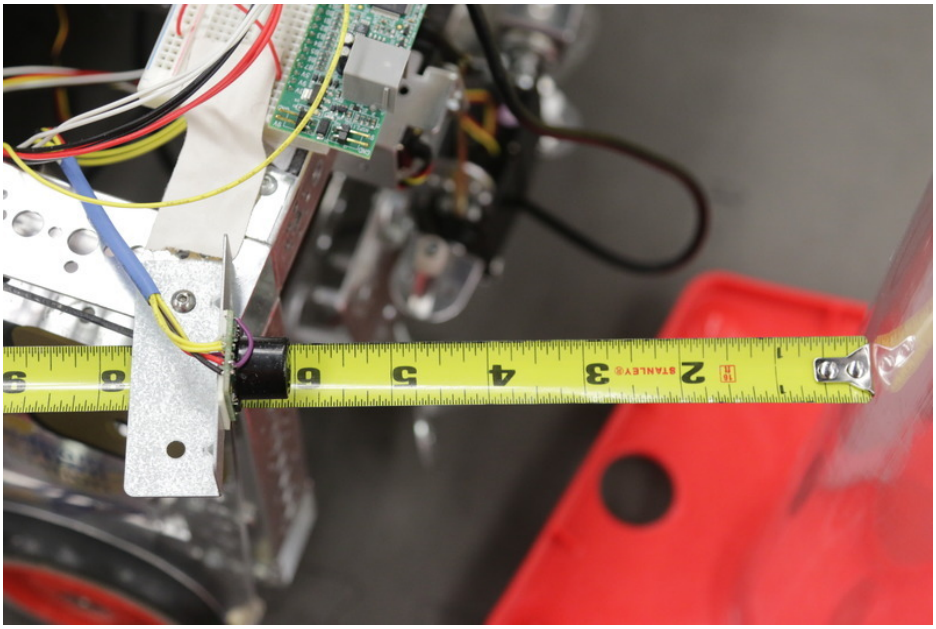
For example, thermocouples used at very high temperatures exhibit an 'aging' effect. This can be detected by performing periodic one point calibrations, and comparing the resulting offset with the previous calibration.



How to do it:

To perform a one point calibration:

1. Take a measurement with your sensor.
2. Compare that measurement with your reference standard.
3. Subtract the sensor reading from the reference reading to get the offet.
4. In your code, add the offset to every sensor reading to obtain the calibrated value.



Example:

Imagine that you have a competition robot that needs to position itself exactly 6" from a goal in preparation for scoring.

You have an ultrasonic rangefinder for your distance sensor. Since you only require maximum accuracy at one distance, a one point calibration is a simple and effictive solution.

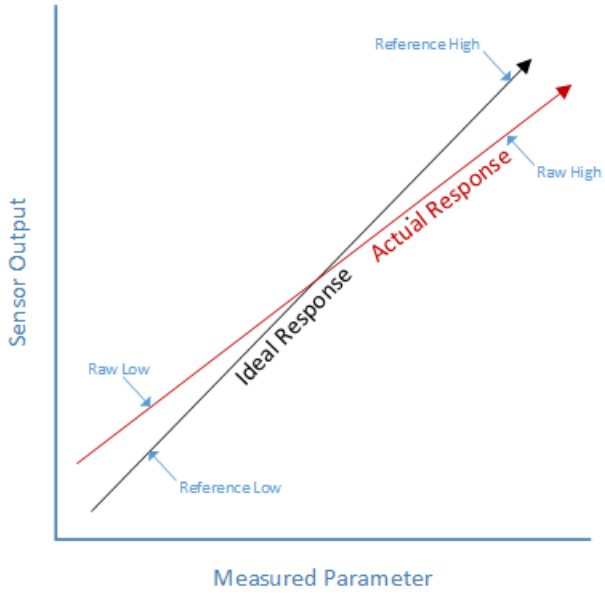
Using a measuring tape as your reference standard, position the robot exactly 6" from the goal.

If you take a reading with your sensor and it says 6.3", then you have a -0.3" offset.

Now edit your code to subtract 0.3" from every reading. Since this is known to be a linear sensor it will likely be pretty accurate over most of its range. But you know with great confidence that it will be spot-on at the critical distance of 6".

Two Point Calibration

A Two Point Calibration is a little more complex. But it can be applied to either raw or scaled sensor outputs. A Two Point calibration essentially re-scales the output and is capable of correcting both slope and offset errors. Two point calibration can be used in cases where the sensor output is known to be reasonably linear over the measurement range.



How to do it:

To perform a two point calibration:

1. Take two measurements with your sensor: One near the low end of the measurement range and one near the high end of the measurement range. Record these readings as "RawLow" and "RawHigh"
2. Repeat these measurements with your reference instrument. Record these readings as "ReferenceLow" and "ReferenceHigh"
3. Calculate "RawRange" as RawHigh – RawLow.
4. Calculate "ReferenceRange" as ReferenceHigh – ReferenceLow
5. In your code, calculate the "CorrectedValue" using the formula below:

CorrectedValue = (((RawValue – RawLow) * ReferenceRange) / RawRange) + ReferenceLow

Example

A common example of a two-point calibration is to calibrate a temperature sensor using an ice-water bath and boiling water for the two references. Thermocouples and other temperature sensors are quite linear within this temperature range, so two point calibration should produce good resuts.

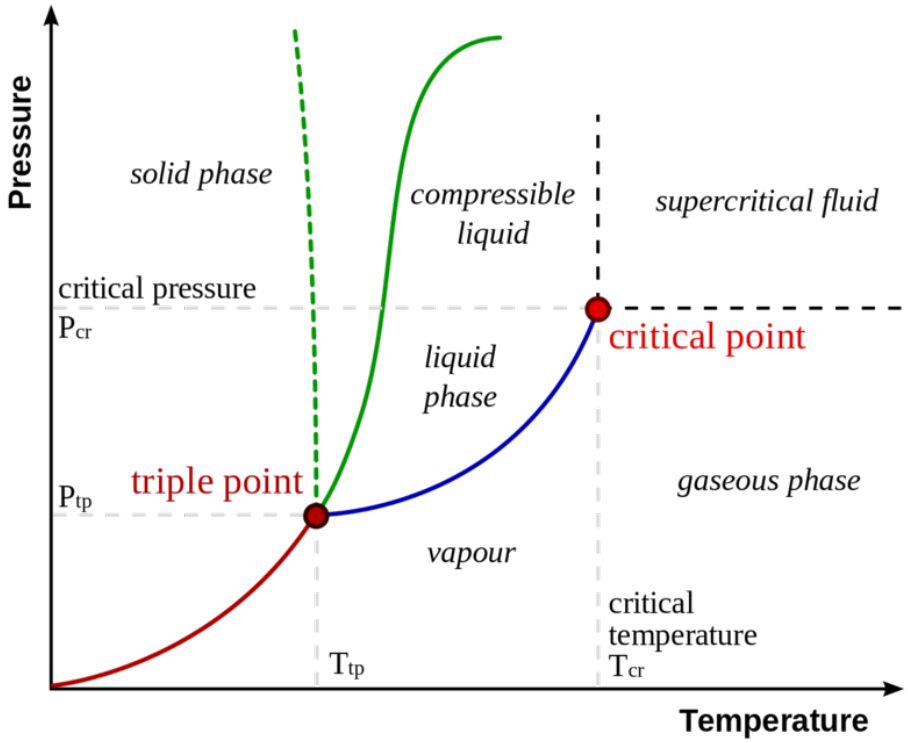
Since these are physical standards, we know that at normal sea-level atmospheric pressure, water boils at 100°C and the "triple point" (https://adafru.it/fcl) is 0.01°C. We can use these known values as our reference values:

ReferenceLow = 0.01°C

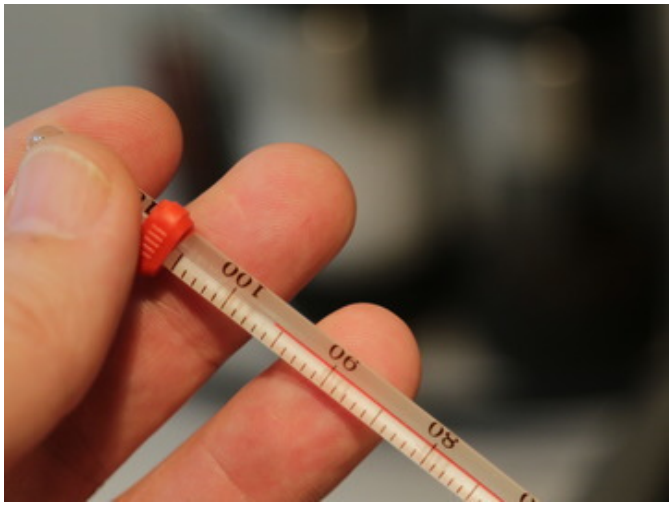
ReferenceHigh = 100°C

ReferenceRange = 99.99

Here we'll show a two point calibration of a laboratory thermometer. But the same principles apply to any temperature sensor:



Phase diagram by Matthieumarechal via Wikimedia Commons (https://adafru.it/fcJ)



Boiling Water

As you can see this lab thermometer is off by -4 degrees at the boiling point of water.

"Triple Point" Ice-Water Bath

The same thermometer registers a little less than zero degrees in the ice-water bath.

So the "Raw" readings are:

RawLow = **-0.5°C**

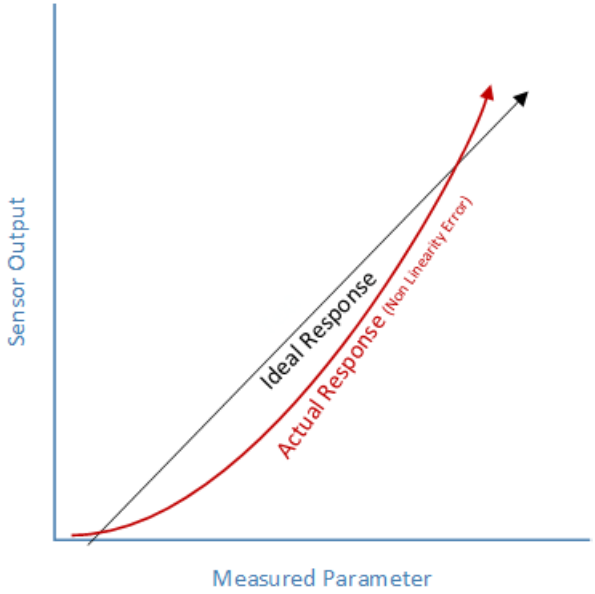
RawHigh = **96.0°C**

RawRange = **96.5°C**

So, if we get a raw reading of **37°C** with this thermometer, we can plug the numbers into the equation to get the corrected reading:

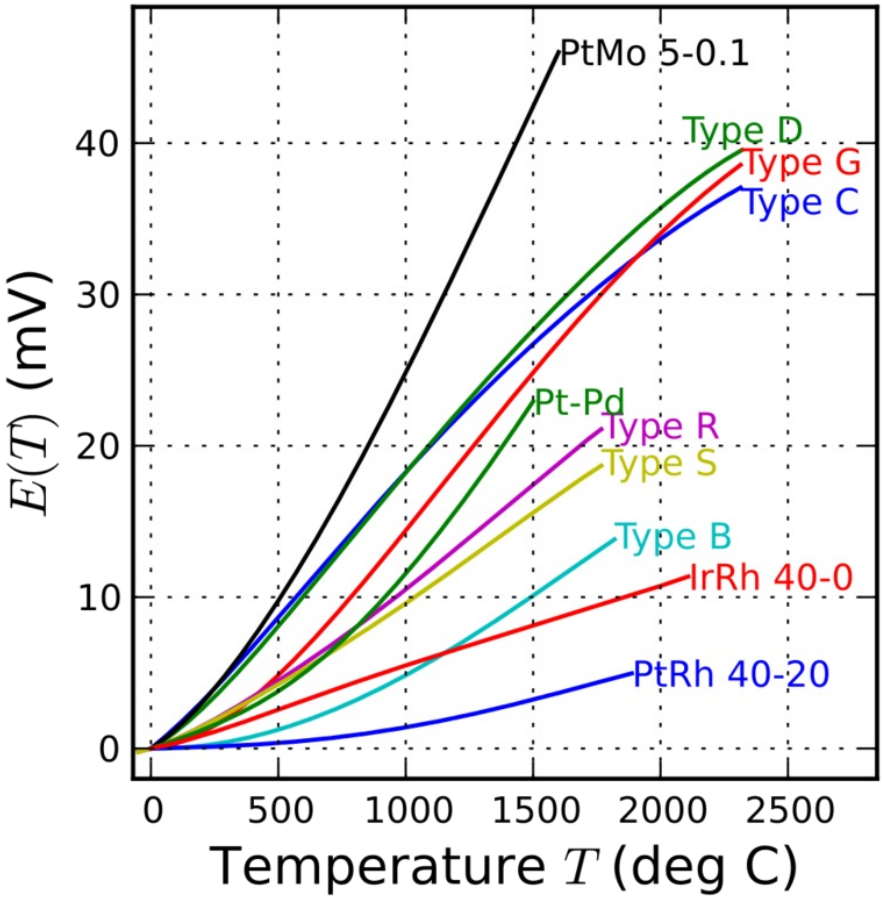
(((37 + 0.5) * 99.99) / 96.5) + 0.01 = **38.9°C**

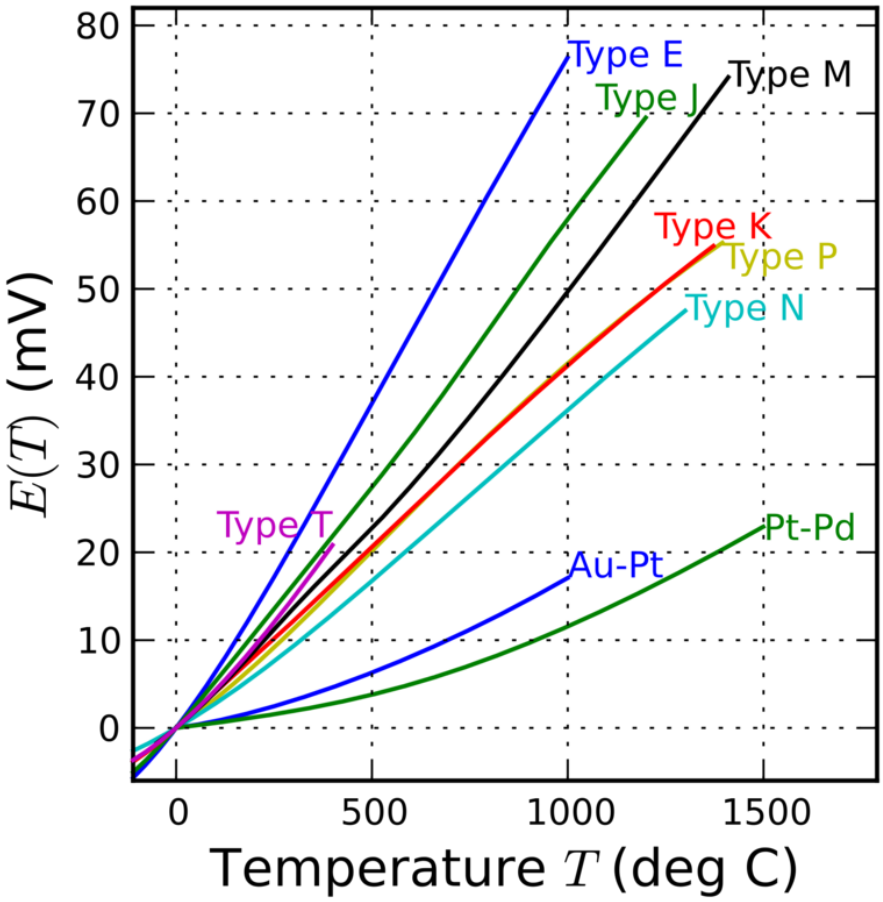
Multi-Point Curve Fitting

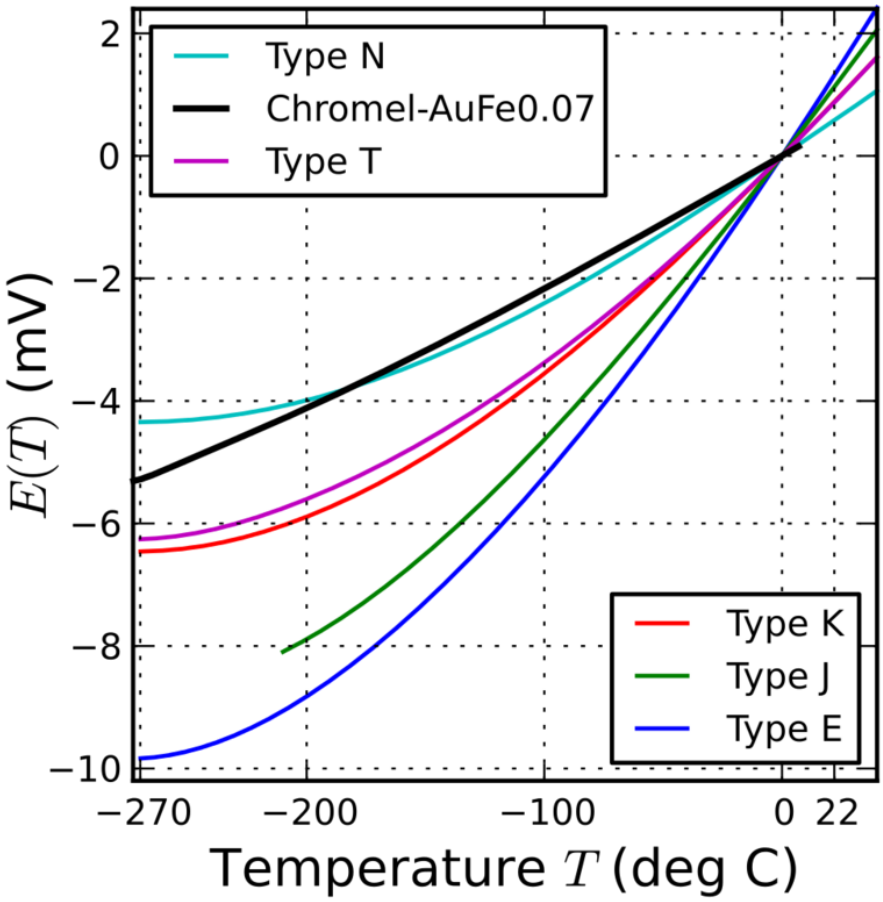


Sensors that are not linear over the measurement range require some curve-fitting to achieve accurate measurements over the measurement range. A common case requiring curve-fitting is thermocouples at extremely hot or cold temperatures. While nearly linear over a fairly wide range, they do deviate significantly at extreme temperatures.

The graphs below show the characteristic curves of high, intermediate and low temperature thermocouples. Note how the lines start to curve more at the extremes.


PtMo 5-0.1
Type D
Type G
Type C
Pt-Pd
Type R
Type S
Type B
IrRh 40-0
PtRh 40-20
$E(T)$ (mV)
0
10
20
30
40
0
500
1000
1500
2000
2500
Temperature T (deg C)


80
70
60
50
40
30
20
10
0
$E(T)$ (mV)
0
500
1000
1500
Temperature T (deg C)
Type E
Type J
Type M
Type K
Type P
Type N
Type T
Au-Pt
Pt-Pd



Thermocouple Characteristic Curves

Thermocouple curves CC0 Public Domain via Wikimedia Commons (https://adafru.it/fcE)

Fortunately, the characteristic curve of standard thermocouple types are well understood and curve-fitting coefficients are available from NIST and other sources.

If you are using a Max31855 Thermocouple Amplifier, check out the next page for some excellent linearization code developed by some members of the Adafruit Forum.

https://adafru.it/fcK

https://adafru.it/fcK

But if you are working with a home-brew DIY sensor, you may need to do some characterization to determine the characteristic curve and derive a linearization formula for your sensor.

Excel and similar spreadsheet type programs have some built-in tools to assist with curve fitting. For a good tutorial on Advanced Regression with Excel, see the following link:

https://adafru.it/C-u

https://adafru.it/C-u

Maxim 31855 Thermocouple Linearization

Forum members heypete (https://adafru.it/fcM) and jh421797 (https://adafru.it/fcM) have implemented the NIST K-type thermocouple equations for the Max31855 thermocouple amplifier.

The code is posted below:

You can check out the original forum thread here:

https://adafru.it/aP1

https://adafru.it/aP1

Code from jh421797

```
// corrected temperature reading for a K-type thermocouple
// allowing accurate readings over an extended range
// http://forums.adafruit.com/viewtopic.php?f=19&t=32086&p=372992#p372992
// assuming global: Adafruit_MAX31855 thermocouple(CLK, CS, DO);
float correctedCelsius(){

   // MAX31855 thermocouple voltage reading in mV
   float thermocoupleVoltage = (thermocouple.readCelsius() - thermocouple.readInternal()) * 0.041276;

   // MAX31855 cold junction voltage reading in mV
   float coldJunctionTemperature = thermocouple.readInternal();
   float coldJunctionVoltage = -0.176004136860E-01 +
      0.389212049750E-01  * coldJunctionTemperature +
      0.185587700320E-04  * pow(coldJunctionTemperature, 2.0) +
      -0.994575928740E-07 * pow(coldJunctionTemperature, 3.0) +
      0.318409457190E-09  * pow(coldJunctionTemperature, 4.0) +
      -0.560728448890E-12 * pow(coldJunctionTemperature, 5.0) +
      0.560750590590E-15  * pow(coldJunctionTemperature, 6.0) +
      -0.320207200030E-18 * pow(coldJunctionTemperature, 7.0) +
      0.971511471520E-22  * pow(coldJunctionTemperature, 8.0) +
      -0.121047212750E-25 * pow(coldJunctionTemperature, 9.0) +
      0.118597600000E+00  * exp(-0.118343200000E-03 *
                            pow((coldJunctionTemperature-0.126968600000E+03), 2.0)
                         );


   // cold junction voltage + thermocouple voltage
   float voltageSum = thermocoupleVoltage + coldJunctionVoltage;

   // calculate corrected temperature reading based on coefficients for 3 different ranges
   float b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
   if(thermocoupleVoltage < 0){
      b0 = 0.0000000E+00;
      b1 = 2.5173462E+01;
      b2 = -1.1662878E+00;
      b3 = -1.0833638E+00;
      b4 = -8.9773540E-01;
      b5 = -3.7342377E-01;
      b6 = -8.6632643E-02;
      b7 = -1.0450598E-02;
      b8 = -5.1920577E-04;
```

```
      b9 = 0.0000000E+00;
   }

   else if(thermocoupleVoltage < 20.644){
      b0 = 0.000000E+00;
      b1 = 2.508355E+01;
      b2 = 7.860106E-02;
      b3 = -2.503131E-01;
      b4 = 8.315270E-02;
      b5 = -1.228034E-02;
      b6 = 9.804036E-04;
      b7 = -4.413030E-05;
      b8 = 1.057734E-06;
      b9 = -1.052755E-08;
   }

   else if(thermocoupleVoltage < 54.886){
      b0 = -1.318058E+02;
      b1 = 4.830222E+01;
      b2 = -1.646031E+00;
      b3 = 5.464731E-02;
      b4 = -9.650715E-04;
      b5 = 8.802193E-06;
      b6 = -3.110810E-08;
      b7 = 0.000000E+00;
      b8 = 0.000000E+00;
      b9 = 0.000000E+00;
   }

   else {
      // TODO: handle error - out of range
      return 0;
   }

   return b0 +
      b1 * voltageSum +
      b2 * pow(voltageSum, 2.0) +
      b3 * pow(voltageSum, 3.0) +
      b4 * pow(voltageSum, 4.0) +
      b5 * pow(voltageSum, 5.0) +
      b6 * pow(voltageSum, 6.0) +
      b7 * pow(voltageSum, 7.0) +
      b8 * pow(voltageSum, 8.0) +
      b9 * pow(voltageSum, 9.0);
}
```

Code from 'heypete'

For the latest updates, please see heypete's github repo:

https://adafru.it/qiD

https://adafru.it/qiD

```
#include <SPI.h>
    #include "Adafruit_MAX31855.h"
```

```
    #define DO   12
    #define CS   11
    #define CLK  10
    Adafruit_MAX31855 thermocouple(CLK, CS, DO);

    void setup() {
       Serial.begin(9600);
       Serial.println("MAX31855 test");
       // wait for MAX chip to stabilize
       delay(500);
    }
    void loop() {
       // Initialize variables.
       int i = 0; // Counter for arrays
       double internalTemp = thermocouple.readInternal(); // Read the internal temperature of the
MAX31855.
       double rawTemp = thermocouple.readCelsius(); // Read the temperature of the thermocouple. This
temp is compensated for cold junction temperature.
       double thermocoupleVoltage= 0;
       double internalVoltage = 0;
       double correctedTemp = 0;

        // Check to make sure thermocouple is working correctly.
       if (isnan(rawTemp)) {
        Serial.println("Something wrong with thermocouple!");
      }
       else {
          // Steps 1 & 2. Subtract cold junction temperature from the raw thermocouple temperature.
          thermocoupleVoltage = (rawTemp - internalTemp)*0.041276;  // C * mv/C = mV

          // Step 3. Calculate the cold junction equivalent thermocouple voltage.

          if (internalTemp >= 0) { // For positive temperatures use appropriate NIST coefficients
             // Coefficients and equations available from
http://srdata.nist.gov/its90/download/type_k.tab

             double c[] = {-0.176004136860E-01,  0.389212049750E-01,  0.185587700320E-04, -
0.994575928740E-07,  0.318409457190E-09, -0.560728448890E-12,  0.560750590590E-15, -0.320207200030E-18,
0.971511471520E-22, -0.121047212750E-25};

             // Count the the number of coefficients. There are 10 coefficients for positive
temperatures (plus three exponential coefficients),
             // but there are 11 coefficients for negative temperatures.
             int cLength = sizeof(c) / sizeof(c[0]);

             // Exponential coefficients. Only used for positive temperatures.
             double a0 =  0.118597600000E+00;
             double a1 = -0.118343200000E-03;
             double a2 =  0.126968600000E+03;


             // From NIST: E = sum(i=0 to n) c_i t^i + a0 exp(a1 (t - a2)^2), where E is the
thermocouple voltage in mV and t is the temperature in degrees C.
             // In this case, E is the cold junction equivalent thermocouple voltage.
             // Alternative form: C0 + C1*internalTemp + C2*internalTemp^2 + C3*internalTemp^3 + ... +
C10*internaltemp^10 + A0*e^(A1*(internalTemp - A2)^2)
             // This loop sums up the c_i t^i components.
             for (i = 0; i < cLength; i++) {
                internalVoltage += c[i] * pow(internalTemp, i);
```

```
            }
               // This section adds the a0 exp(a1 (t - a2)^2) components.
               internalVoltage += a0 * exp(a1 * pow((internalTemp - a2), 2));
         }
         else if (internalTemp < 0) { // for negative temperatures
            double c[] = {0.000000000000E+00,  0.394501280250E-01,  0.236223735980E-04, -
0.328589067840E-06, -0.499048287770E-08, -0.675090591730E-10, -0.574103274280E-12, -0.310888728940E-14,
-0.104516093650E-16, -0.198892668780E-19, -0.163226974860E-22};
            // Count the number of coefficients.
            int cLength = sizeof(c) / sizeof(c[0]);

            // Below 0 degrees Celsius, the NIST formula is simpler and has no exponential components:
E = sum(i=0 to n) c_i t^i
            for (i = 0; i < cLength; i++) {
               internalVoltage += c[i] * pow(internalTemp, i) ;
            }
         }

         // Step 4. Add the cold junction equivalent thermocouple voltage calculated in step 3 to the
thermocouple voltage calculated in step 2.
         double totalVoltage = thermocoupleVoltage + internalVoltage;

         // Step 5. Use the result of step 4 and the NIST voltage-to-temperature (inverse)
coefficients to calculate the cold junction compensated, linearized temperature value.
         // The equation is in the form correctedTemp = d_0 + d_1*E + d_2*E^2 + ... + d_n*E^n, where E
is the totalVoltage in mV and correctedTemp is in degrees C.
         // NIST uses different coefficients for different temperature subranges: (-200 to 0C), (0 to
500C) and (500 to 1372C).
         if (totalVoltage < 0) { // Temperature is between -200 and 0C.
            double d[] = {0.0000000E+00, 2.5173462E+01, -1.1662878E+00, -1.0833638E+00, -8.9773540E-
01, -3.7342377E-01, -8.6632643E-02, -1.0450598E-02, -5.1920577E-04, 0.0000000E+00};

            int dLength = sizeof(d) / sizeof(d[0]);
            for (i = 0; i < dLength; i++) {
               correctedTemp += d[i] * pow(totalVoltage, i);
            }
         }
         else if (totalVoltage < 20.644) { // Temperature is between 0C and 500C.
            double d[] = {0.000000E+00, 2.508355E+01, 7.860106E-02, -2.503131E-01, 8.315270E-02, -
1.228034E-02, 9.804036E-04, -4.413030E-05, 1.057734E-06, -1.052755E-08};
            int dLength = sizeof(d) / sizeof(d[0]);
            for (i = 0; i < dLength; i++) {
               correctedTemp += d[i] * pow(totalVoltage, i);
            }
         }
         else if (totalVoltage < 54.886 ) { // Temperature is between 500C and 1372C.
            double d[] = {-1.318058E+02, 4.830222E+01, -1.646031E+00, 5.464731E-02, -9.650715E-04,
8.802193E-06, -3.110810E-08, 0.000000E+00, 0.000000E+00, 0.000000E+00};
            int dLength = sizeof(d) / sizeof(d[0]);
            for (i = 0; i < dLength; i++) {
               correctedTemp += d[i] * pow(totalVoltage, i);
            }
         } else { // NIST only has data for K-type thermocouples from -200C to +1372C. If the
temperature is not in that range, set temp to impossible value.
            // Error handling should be improved.
            Serial.print("Temperature is out of range. This should never happen.");
            correctedTemp = NAN;
         }

         Serial.print("Corrected Temp = ");
```

```
        Serial.println(correctedTemp, 5);
        Serial.println("");

    }

    delay(1000);

}
```