Using a Thermistor to Measure Temperature

Overview of a Thermistor

Thermistors are resistors that vary their resistance according to temperature.
The change in resistance is roughly proportional to the change in temperature. But the resistance is not the same as the temperature, so a tiny bit of maths is required to turn that resistance into a temperature reading.
The resistance is not completely linear across the whole range of temperatures. This means that for 1 degree change in temperature, the change in resistance might not be the same at every possible temperature increase. So, at very low temperatures or very high temperatures, the reading can become inaccurate.
PTC Thermistors are 'positive temperature coefficient', which means that as the temperature goes up, the resistance goes up. NTC Thermistors are 'negative temperature coefficient', which means that as the temperature goes, up, their resistance goes down.

Converting Resistance to Temperature

Manufacturers provide data sheets that contain key parameters of each type of thermistor, and it is possible to calculate the temperature from the resistance using the information in these data sheets. However, the maths associated with those calculations is quite accurate, but can be quite complex to understand.

A much easier method is to perform a 'two point calibration'. This means that you measure two known temperatures with your thermistor, and work out what reading you get for those two temperatures. Then if you assume that the change in resistance for each degree in temperature is roughly the same across that whole range, you can perform a simple calculation to convert that reading into a real temperature reading. If you plot a graph of temperature against resistance and if that looks like a straight line rather than a curve, it is called 'linear' (it is 'like a line').

Analogue and Digital

The measurement of temperature, resistance, light level, and other forms of measurement are analogue – this means that there is a continuous range of numbers that can be used to represent them. Compare this against digital, which is either on or off.
Computers are digital devices and can only measure on and off. Many computers have analogue input pins that do this conversion from an analogue reading to a digital value for you. If they do not have such a pin, you can connect an analogue to digital converter chip (ADC) to the computer which does the conversion for you.

Measuring Resistance From a Computer

Because we are going to connect our thermistor to an analogue input pin on a computer, we need a little bit of extra circuitry to make this work. The analogue input pin on a computer measures voltage, and it will measure that voltage somewhere been 0 Volts and the supply voltage of the computer CPU (for most modern computers this will be around 3 Volts).

The way to turn a resistance into a voltage reading, is to build a potential divider. This consists of two resistors, one wired from the pin to the positive 3 Volts supply, and one wired from the same pin to the 0 Volts. As either of the resistances are varied, this causes the potential between the 3 Volt supply and the 0 Volt to vary somewhere between 0 and 3 Volts where it connects to the pin, so the resistors 'divide the potential voltage'.

Making it More Like a Straight Line

A useful 'engineers trick' to improving how linear the response of a thermistor is, consists of putting a resistor in series with it. A general rule of thumb is that you get a good amount of 'linearisation' (turning the response into a straight line) around 25 degrees Celsius, by using the same value series resistor as that of the thermistor. So, if your thermistor is a 10K thermistor, using a 10K series resistor is a good bet.

Which Way Does the Line Go – Up or Down?
If you are using a PTC thermistor (positive temperature coefficient), put the thermistor from the pin to the 0 Volts, and the series resistor from the pin to the 3 Volts. This way, when the resistance of the thermistor increases due to an increase in temperature, the potential divider will have a bigger resistance at the bottom, and the voltage at the pin will go up.

If you are using a NTC thermistor (negative temperature coefficient), put the thermistor from the pin to the 3 Volts, and the series resistor from the pin to the 0 Volts. This way, when the resistance of the thermistor decreases due to an increase in temperature, the potential divider will have a smaller resistance at the top, and the voltage at the pin will go up. This will just make the numbers in your program code go the right way (up when the temperature rises, down when it falls), which will make your program code easier to understand.

Readings from an Analogue to Digital Converter
An analogue to digital converter (ADC) will convert a voltage reading into a number. The ADC has a range of numbers it can work with, and this is its resolution. A 10 bit ADC has 10 binary digits. With 10 binary digits you can represent 2^10 (2 to the power of 10) different numbers, which works out to be from 0 to 1023. A zero voltage on the analogue input pin will give a reading of 0 when you read the ADC from your program code. Assuming the supply to your CPU is 3 Volts, then 3 Volts applied to the analogue input pin will give you a reading of 1023. Voltages in between will give other numbers. Remember that your potential divider has converted the resistance of your thermistor into a voltage.

The Whole Chain of Conversions from Temperature
It is worth remembering that there are lots of conversions taking place here. The temperature in the real world is being converted to a resistance by the thermistor. This is then being converted to a voltage by the potential divider (and the addition of the extra resistor makes its response more like a line) and you set the direction of this line depending on which way round you put the resistors.

The ADC (analogue to digital converter) on the computer turns this voltage into a number e.g. in the range of 0 to 1023. Finally your program code has to convert this ADC reading into a real temperature reading again. How do you make sense of all those conversions to get this 0 to 1023 number back to a temperature again?!

Two Point Calibration Technique
Fortunately for us, we don't have to do lots and lots of maths for each of those conversions. All we have to do is to understand how the number range 0 to 1023 represents a temperature range in degrees Celsius. Because we have arranged for all of these conversions to be roughly linear (a straight line if we draw a graph), we can use a technique called Two Point Calibration. If we have two known temperature readings roughly at opposite ends of the number scale, we can make the thermistor measure the first temperature and then record the ADC reading at that temperature. Then we make the thermistor measure the second temperature and record the ADC reading at that temperature. Finally, we use the equation of a straight line ($y = mx + c$) to turn any ADC reading into its appropriate temperature. Our computer then knows how hot or cold it is in the outside world!

Choosing Temperatures for Two Point Calibration
There is a classical physics experiment where you calibrate an unmarked thermometer tube by first putting the thermometer in ice (0 degrees Celsius), marking a line on the tube, then putting the thermometer in steam from a kettle (100 degrees Celsius) and marking a second line. You then divide the gaps between the two lines into 10 equal segments, and divide each of those segments into 10 equal marks, and you have a thermometer! This works because a thermometer tube is roughly linear (a straight line response) across its whole length. Ice is quite easy to come by from a freezer. Steam from a kettle is a little dangerous to use (especially for small children). An easier way to choose the upper temperature point is to use room temperature (roughly 20 degrees Celsius) or body temperature (roughly 37 degrees Celsius) and measure that with a second accurate thermometer (e.g. a digital thermometer from the science cupboard).

Doing a Two Point Calibration

1. Using an accurate second thermometer, note down the room temperature. This will probably be about 20 Celsius. Alternatively hold the thermistor and accurate thermometer between your fingers and measure body temperature, this will probably be 37 Celsius. Note down the reading on the accurate digital thermometer and the value that the ADC on your computer reports for this temperature, as the 'upper reading'.
2. Get an ice cube. Put the thermistor on the ice cube for a few seconds, and read out the digital value that the ADC on your computer reports for this temperature. Write that down as 'lower reading'.

Write those both down as ‘upper readings’ like this:

Reading	Temperature	ADC Reading
Lower	0 (ice)	66
Upper	20 (room temperature)	345

Calculating 'm' and 'c' (y=mx+c equation for your line)

The next step is to calculate 'm' (the gradient) and 'c' (the intersection of the y axis) for your temperature line.
'm', the gradient, is just 'rise over tread' – i.e. how much does the line rise in the y axis, for a certain tread in the x axis?
You can do this visually by drawing the graph on graph paper. Or you can do it using a bit of maths:

rise = ADChigher – ADClower
tread = TEMPhigher – TEMPlower

In our example:
rise = 345 – 66 = 279
tread = 20 – 0 = 20
m (gradient) = rise/tread = 13.95

'c' the intersection, is where the line cuts the y axis. In this case, because your lower calibration point was at zero, you already know this value – at a temperature of 0, the ADC reading is 66, so the line crosses the y axis at 66.

If you choose a different lower temperature calibration point, you can draw the graph on graph paper or use a bit of maths to calculate where the line crosses the y axis. But we used ice, so our maths here is much simpler!

Checking your Maths

The last thing to do is to check that your maths is correct. You can easily do this by putting in some known values of x(temperature) and checking that the y value (ADC reading) is what you expect. Fortunately we have two known values of x and y – the values you used for your two point calibration!

m = 13.95
c = 66
x = 0

y = mx + c
y = (13.95 * 0) + 66
y = 66

m = 13.95
c = 66
x = 20

y = mx + c
y = (13.95 * 20) + 66
y = 345

Converting ADC Values to Temperatures
But hang on a minute – this works if we know the temperature and want the ADC reading – our program will have an ADC reading and need to convert it to a temperature, so it's all back to front? Fortunately, we just run the y = mx + c equation backwards by rearranging it. Take 'c' off of both sides, then divide both sides by 'm', and you get this backwards equation that turns an ADC reading into a temperature:

x = (y – c) / m

Test the maths again to make sure your equation is correct:

m = 13.95
c = 66
y = 66

x = (y – c) / m
x = (66 – 66) / 13.95
x = 0

m = 13.95
c = 66
y = 345

x = (y – c) / m
x = (345 – 66) / 13.95
x = 20

Writing your Thermometer Program
Now that you know how to convert ADC readings into temperatures for your chosen circuit, you can write the program.
Note that many tiny computers do not allow you to use decimal numbers like 13.95, but only allow whole numbers like 13. Because we only want to display temperatures in whole numbers, the best thing you can do here is to round the 'm' value – so if the decimal is below 0.5, use the whole number only, if the decimal is 0.5 or greater, round it up to the next number. Because our 'm' is 13.95, we round it up to 14.

The computer will generally do the same, i.e. it will round the result. So in the example below using an ADC reading of 345, 345-66 = 279, then 279/14 = 19.92, which the computer will round up to 20.

For the micro:bit

1. Use this program to read out the ADCValue for a given temperature first, by doing your two point calibration:

- • **JavaScript Blocks Editor:** microbit-ADCreading-jsb.hex
- • **Python Editor:** ADCReading.py

2. Calculate 'm' and 'c' as above and put the numbers into the variables in these programs, to create a thermometer:

- • **JavaScript Blocks Editor:** microbit-temperature-jsb.hex
- • **Python Editor:** temperature.py