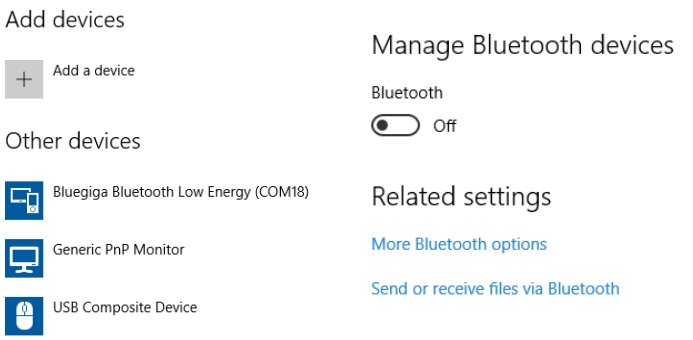
First steps in using the BBC Micro:bit for data-logging and modelling

Adrian Oldknow adrian@ccite.org 8th February 2017

This is the companion piece to "*First steps in using the BBC Micro:bit for control and physical computing*". The key difference between the two is that the earlier document concentrated on developing programs for the micro:bit in an external editor on a computer or mobile device which was then compiled into a hex file and transferred to the micro:bit which could then be disconnected and run as an autonomous device. Here we look at other Apps and programs which can access the built-in sensors of the micro:bit. This is possible because another feature provided in the £15 micro:bit is a Bluetooth Low Energy transmitter and receiver. If you have used the micro:bit editor systems on a mobile device like a Samsung smart phone or an Apple iPad you will found out how to pair your device so that you can transfer your hex file wirelessly to the micro:bit. We also saw in the earlier piece how two micro:bits could "talk to each other" using the `***Radio***' commands. Currently we can't use Bluetooth on micro:bits to talk each other because there is not enough memory for this. We can expect quite soon to see further developments of the micro:bit which will have more memory and other features, but for now we will be looking mainly at micro:bits sending data to computers via Bluetooth.

The first App we will explore is one developed by Clive Seager of Revolution Education in Bath. This is a free App which works on Windows, Apple and Linux systems. I will be illustrating its use with my Windows 10 laptop. Because the signal strength from the micro:bit is comparatively weak, I will use a £10 USB `dongle' for the communication.

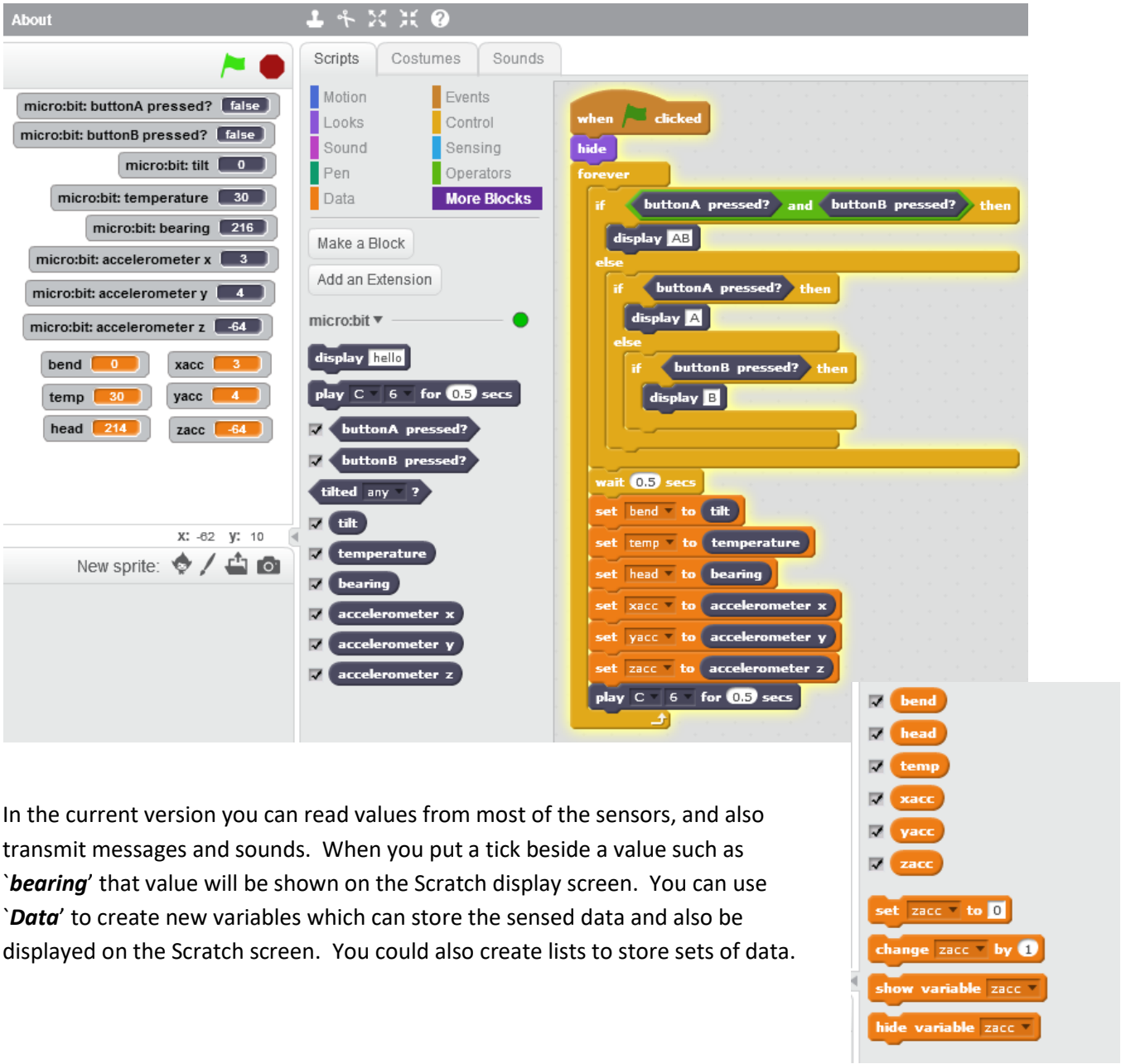
We don't want the communication to be blocked by the computer's more powerful in-built Bluetooth, so I have used `Settings' to confirm that the BLE dongle is activated and that the built-in Bluetooth is turned off.



Then you can install the correct version of the `S2Bot' App from the links at the end of the web-page. It is called "S2Bot" because it is designed to connect a variety of robot-like educational devices to MIT's `Scratch' programming language. I will be using the Scratch offline editor. If you follow the set-up instructions for the S2Bot micro:bit here you should be able to connect your micro:bit. The three parallel bars at the top right of the display contain the menu to help you do this. First you select `Program micro:bit'. This downloads the hex file `microbit-s2bot.hex' file which you store somewhere convenient on your computer. Then you plug your micro:bit into a USB support. After a short while it should pop-up an external memory device, e.g. D: Find your hex file and right-click on it. Select `Send to:' and then click on `microbit D:'. It will then download the file to your micro:bit. It will probably then ask you to "Draw a circle". You rotate the micro:bit so that the outer leds form a large "O". You should see a kind of happy

face. This is the process by which the micro:bit calibrates its magnetometers. Then you can disconnect the micro:bit and attach a battery box. Now you are set up. When you select “Scan for devices” your micro:bit’s name and number should appear. Click on it. It should turn yellow and then green, with a “C” displayed on the micro:bit. Close the dialog and you should see reading from the micro:bit’s sensors and buttons on the App display. Check that these change as you move the micro:bit about, or press its buttons. So we now have a monitor screen displaying remotely the state of the micro:bit’s on-board devices. So this is the first example of `telemetry’ I have come across for the micro:bit – i.e. giving a real-time feed of sensed data on a remote device wirelessly. Next we will use sensed values to control on-screen displays using the Scratch programming language. First we will just show Scratch sensing if a button is pressed on the micro:bit.

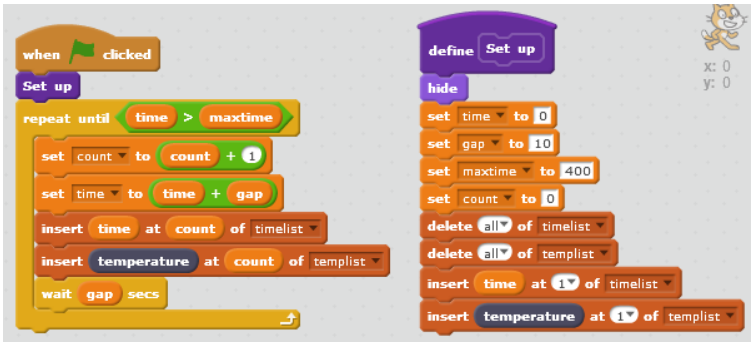
If you return to the `S2Bot’ menu, you can select 'New Scratch template'. Save the `microbit_template.sb2’ template to somewhere convenient. Now start a new Scratch project. Use `File’ and `Open’ to open the template file. That both connects Scratch to the micro:bit, and gives access to a new menu of tools in Scratch for working with the micro:bit. Use `***More Blocks***’ to see the extensions which S2Bot creates.

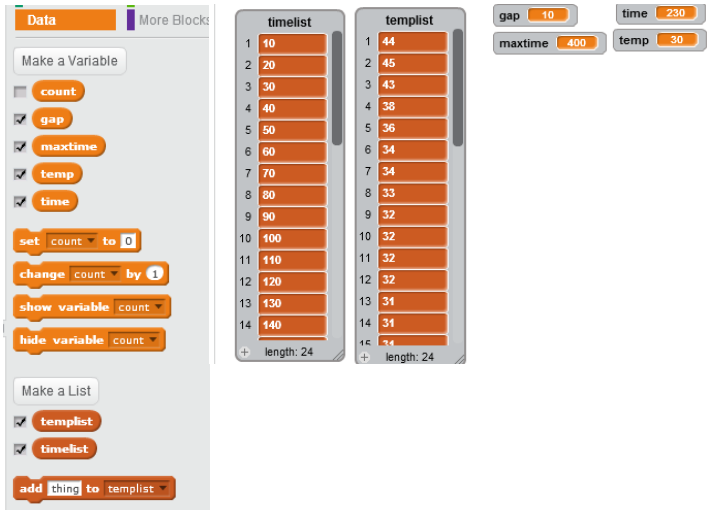


In the current version you can read values from most of the sensors, and also transmit messages and sounds. When you put a tick beside a value such as `***bearing***’ that value will be shown on the Scratch display screen. You can use `***Data***’ to create new variables which can store the sensed data and also be displayed on the Scratch screen. You could also create lists to store sets of data.

The example program just checks if a button is pressed and displays a character on the led display if it is. It also shows the current readings of the sensors on the Scratch graphics displays and outputs a beep on a buzzer attached to the micro:bit, such as the one the Tektronik M1 Battery board.

1. **Storing lists of time and temperature data**

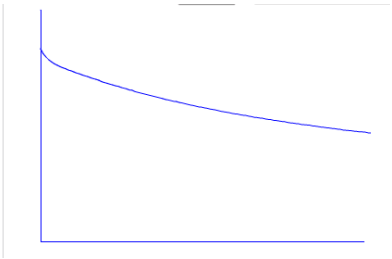


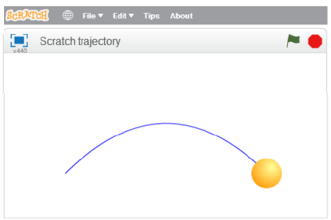


The program we will start to develop in Scratch will collect data from the micro:bit's temperature sensor . The experiment will consist of heating the micro:bit up by putting it on some card over a mug of hot water, and then logging data as it is moved away to cool. The experiment is controlled by the variables `***gap***' (time between readings) and `***maxtime***' (total of the experiment). We need to set up two lists to save the time and temperature data for the readings – these are `***timelist***' and `***templist***'. The program has a block called `Set up' which initialises the variables for the experiment. The main program is a `***while***' loop which updates the variables and takes the readings. If you right-click on either list you have the option to `***export***' the data. Unfortunately this is saved as a text file with no separators between the values. So there' a challenge for those with computing skills – can you convert the files into a common format, such as with a comma between each value. This format is known as CSV for `comma separated variable' file – which can be opened by most common spreadsheet programs.

But here is the **real challenge**. Can you adapt the program so that each pair of values from the lists defines the coordinates of a point (x,y) on the Scratch display. Can you plot the data, say, as a scattergram, or as a bar chart, or as a line graph? See what you can find out about "Newton's Law of Cooling"? Does your data fit the model which Newton discovered experimentally?

Better still, can you measure data from a micro:bit in motion? Maybe you could turn it into a pendulum, or hang it from a spring, or put in your pocket as you take some exercise, such as bouncing on a trampoline. In Scratch you could export the data, or graph it. But for real dynamism can you create an **animation** where the data controls the position of a Scratch icon? (This is called simulation and is the basis of most video games and animated cartoons.) You might find these sources useful: the CAS newsletter and the STEM Learning resource.





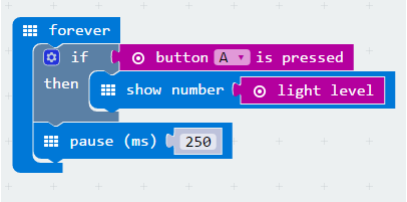
WHEN SCRATCH GOES BALLISTIC!
EXPLORING THE MATHS

Adrian Oldknow from the Cambridge Centre For Innovation in Technological Education (CCITE) looks at developing Scratch code to simulate an object's flight in real-time.

We will now leave Scratch and use a spreadsheet to display and model data collected from the micro:bit. The simplest way for some experiments is just to use the micro:bit as a measurement tool e.g. a thermometer, and record data manually in a spreadsheet. Here is a simple experiment,

2. How does light intensity vary with distance? Modelling data with Excel

I am using a micro:bit with the Tektronik M1 battery case for convenience. In the kitchen there is a bright downward pointing LED. It is evening and all other lights are switched off. A folding 1 metre rule is balanced against the wall.

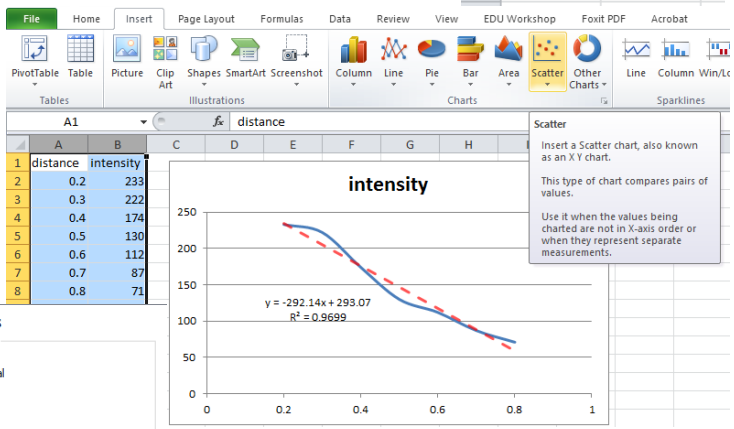


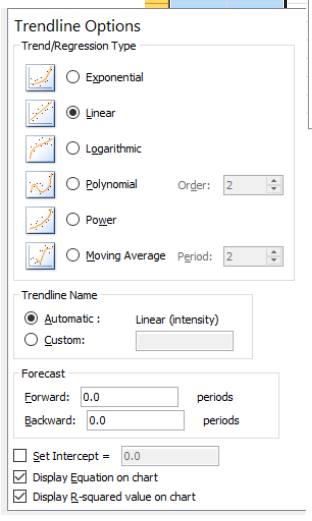
We want to be able to move the micro:bit to a given distance below the light and the press button A to display the corresponding light intensity. With a pen and paper you can record data in two columns for distance and intensity. When you have enough readings you can open an Excel, or similar, spreadsheet and enter the data by hand.

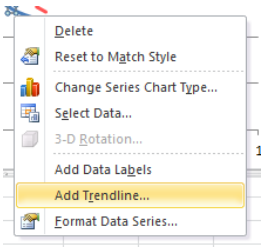
	A	B
1	distance	intensity
2	0.2	233
3	0.3	222
4	0.4	174
5	0.5	130
6	0.6	112

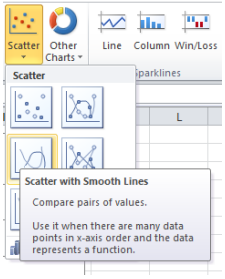
The units are in the micro:bit's own scale, where 255 is the maximum intensity. As we would expect, the further the micro:bit is from the LED the lower the light intensity. This is a relationship of "*the more…the less*" type. (Also known as negative correlation)

Can you imagine what a graph of the data will look like? If you click on the "A" and shift-click on the "B" you can select the two columns of data to graph. From the `Insert' tab drop down the `Scatter' menu and chose a type you fancy. I have chosen the second on the left `Scatter with Smooth Lines'. This produces the blue graph shown. I have deleted the caption. As well as having a table of data and a scatterplot, we can also fit a possible function which might be a close match to our data. If you right-click on the blue graph you can select `Add Trendline' and chose a type from the menu. I have selected `Linear', and also asked for the equation of the line to be shown, as well as the value called `R-squared'.





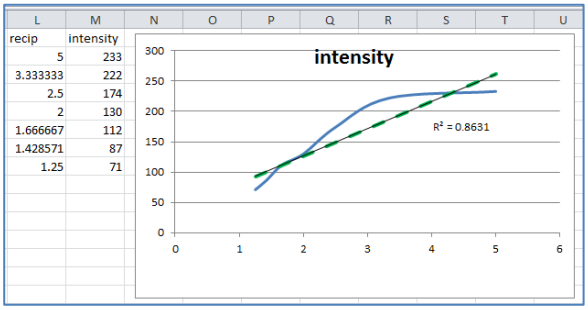




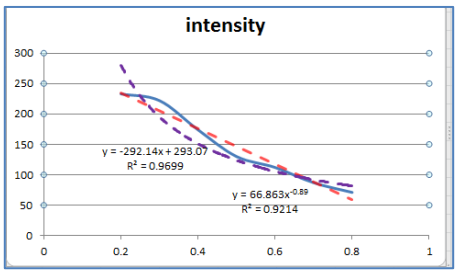
Behind the scenes, Excel does some smart computation using a technique called `least-squares curve fitting' to find the equation of the straight line which is a `best-fit' for the data in some sense. This involves calculating a number called R, which is the `correlation coefficient'. Excel displays the value R^2, which is 0.9699. The actual value of R is then -√0.97 i.e. R ≈ -0.96 . It is negative because we are in a "the more the distance the less the intensity" situation. The maximum possible value of R^2 is 1, so we have a pretty good fit for the small amount of data we collected. The measurement of distance was `by eye' and the room was not completely dark, so there was experimental error. But the red dashed line on the screen can be continued to cross the *x*-axis at around 1 metre. After that it will return a negative value for the light intensity – which is impossible! So we have a good `locally linear' fit. But the further the micro:bit is away from the light we would expect the intensity to be very small and then the graph would approach, but not cut, the *x*-axis.

You may have met a model used in electricity called `Ohm's Law', which connects voltage V, current i and resistance R by the formula V = iR. Imagine a simple circuit with a fixed voltage (using a battery, say), a variable resistor (using a potentiometer or a light dependent resistor) and a bulb or LED. Then as you increase the resistance, the bulb will be less bright – and vice versa. The more the resistance, the less the current. Congratulations – you have just performed a `mind experiment'! The formula connecting i and R is i = V/R. Again we have a negative correlation, but instead of using a straight line graph like $y = a.x + b$, to model the scatterplot data, we are could also explore a function like $y = a/x$.

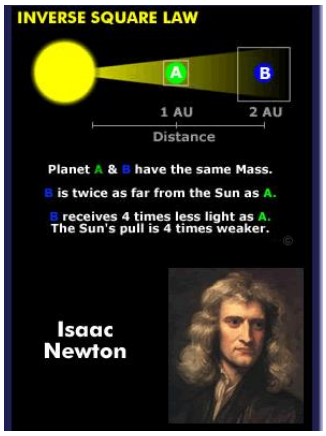
A standard technique in seeking good curve-fits to data before we all had access to computers was to transform the data in some way. We could see if a `reciprocal function' is a good fit for our data by plotting the light intensity against the reciprocal of the distance, and then seeing if we have a good linear fit. The value in cell L2 was calculated from the formula "=1/A2", and this is copied down as far as cell L8.



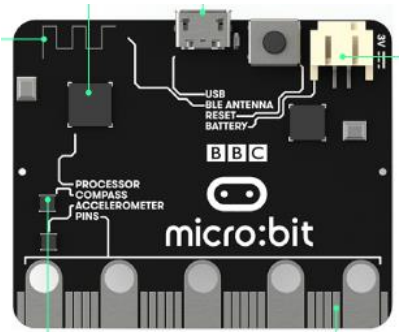
The data in column B has just been copied and pasted to column M. The scatterplot is shown as the blue curve, and the linear trendline is shown in green. So it's not as good a fit as we would like, but it does suggest we could try a different class of functions. In mathematics $1/x$ can also be written as x^{-1}. If you right-click on the scatterplot again you can now select another `Add Trendline'. This time choose the `Power' option. This gives a function like $y = 67\ x^{-0.9}$. Its graph is shown in purple, and the R^2 value is 0.9214 , also suggesting a close fit.



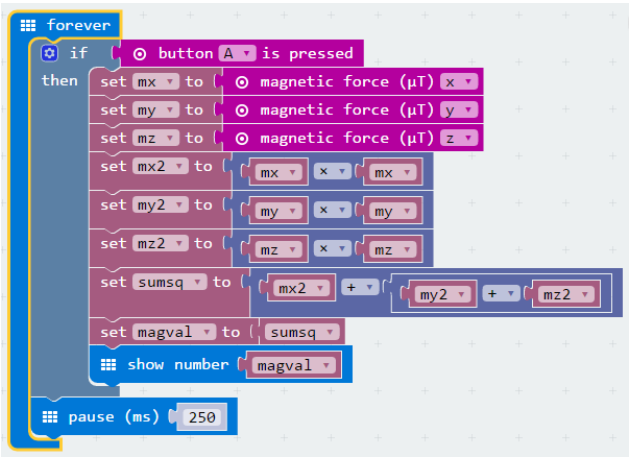
So we have taken a little excursion into the world of `mathematical modelling'. This uses statistical techniques to find functions which provide close approximations to experimental data. They don't explain anything about why the data might be connected in any given way. Explanation is the job of science – and many famous scientists have come up with explanations for formulae derived from experimental data. One of these was Sir Isaac Newton, and his `inverse square law'. Can you devise a more accurate experiment for the decay of light intensity with distance? Do your results agree with an inverse square law?



3. Finding out more about the built-in magnetometers – as well as how to do some more advanced maths in PXT!



This is a do-it-yourself exploration about the micro:bit's magnetometers which measure magnetic field strength in three directions. They are housed in the little black box marked `Compass'. The *x*-axis is across the micro:bit parallel to the line drawn above the 5 holes. The *y*-axis is perpendicular to this going up the micro:bit. The *z*-axis is perpendicular to both of these coming out from the front of the micro:bit. The units are called `micro Teslars' – see what you can find out about them. (You may have heard of Teslar in connection with electric cars?) The micro:bit uses these sensors to locate the direction of magnetic North and thus to calculate Compass bearings. When you start to use these sensors for the first time you will be asked to `Draw a Circle' – and you have to tilt the micro:bit to get a single flashing white LED to drop into one of the edges until you make a letter "O". The basic program to read the strength of the magnetic field uses a 3D version of Pythagoras's Theorem. If you wanted to find the value of a vector in two dimensions you just add the squares of the two components and take the square root – like finding the length of the diagonal of a rectangle. You do just the same in three dimensions by taking the square root of the sum of the squares of the three components – just like finding the diagonal of a brick. Here we run into a problem with the current version of the PXT block editor. There are none of the more advanced maths functions like square-root, sine, cosine, logarithm and exponential. But they are there buried in the JavaScript editor! So we will develop the main program to create a magnetic field measurer using blocks. Then convert it JavaScript, and then add in the few magic commands needed to calculate a square root.




```
6  let mz = 0
7  let my = 0
8  let mx = 0
9  basic.forever(() => {
10     if (input.buttonIsPressed(Button.A)) {
11         mx = input.magneticForce(Dimension.X)
12         my = input.magneticForce(Dimension.Y)
13         mz = input.magneticForce(Dimension.Z)
14         mx2 = mx * mx
15         my2 = my * my
16         mz2 = mz * mz
17         sumsq = mx2 + (my2 + mz2)
18         magval = sumsq
19         basic.showNumber(magval)
20     }
21     basic.pause(250)
22 })
```

The last of the `set' blocks ought read something like `set magval to sqrt(sumsq)' but the Math blocks do not include a square root function. The trick is to convert to JavaScript which has an editor which can access the full Math library which includes the `Math.sqrt(x)' function. But we can't just write: 21 magval = Math.sqrt(sumsq)

Instead we have to declare our own function `getRoot' which will ensure that the sqrt function is given a number to work with. So you need to enter a `function' definition in lines 9 to 11. Now you can use sine or tan etc.

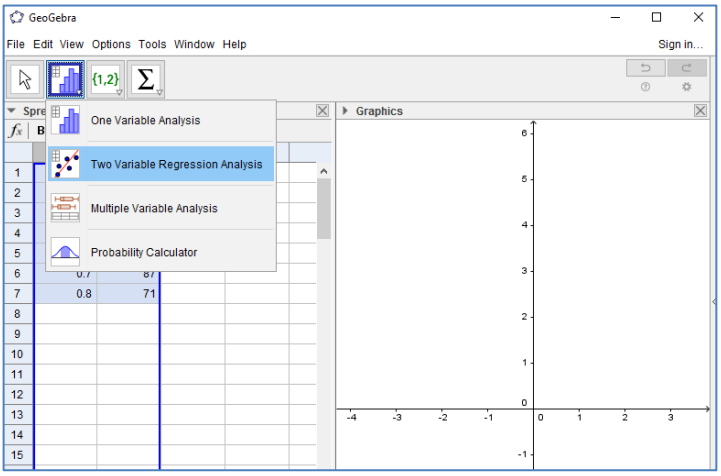
```
8  let mx = 0
9  function getRoot(x: number) {
10     return Math.sqrt(x)
11 }
12 basic.forever(() => {
13     if (input.buttonIsPressed(Button.A)) {
14         mx = input.magneticForce(Dimension.X)
15         my = input.magneticForce(Dimension.Y)
16         mz = input.magneticForce(Dimension.Z)
17         mx2 = mx * mx
18         my2 = my * my
19         mz2 = mz * mz
20         sumsq = mx2 + (my2 + mz2)
21         magval = getRoot(sumsq)
22         basic.showNumber(magval)
23     }
24     basic.pause(250)
25 })
```

Can you develop a program to read data from the micro:bit's three accelerometers? You will need to find out which units the micro:bit uses for acceleration. Try reading out the *x*-value when A is pressed, the *y*-value when B is pressed and the *z*-value when both A and B are pressed. You should find that they do not all show zero when the micro:bit is at rest! Can you adapt the program to read out just the magnitude of the acceleration in metres per second per second (ms^{-2})? These will be the square root of the sum of the three components after you have done a bit of tinkering with the raw values.

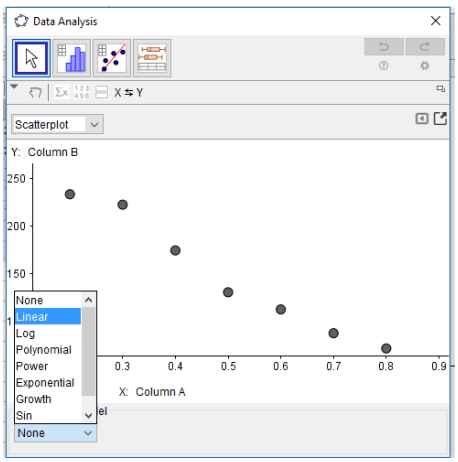
So you might like to use the PXT editor as a way to learn more about JavaScript programming. But here we more concerned with the science and maths than computing. Like all spreadsheets, Excel was really designed for accountancy and book-keeping, although it does have many smart features for statistics. If we want to some mathematics, especially drawing graphs, it would be much easier to use a purpose-built program like GeoGebra, as well as being much cheaper (well, free actually).

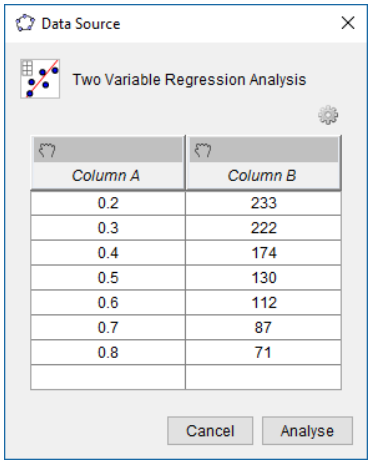
4. Modelling micro:bit data with GeoGebra

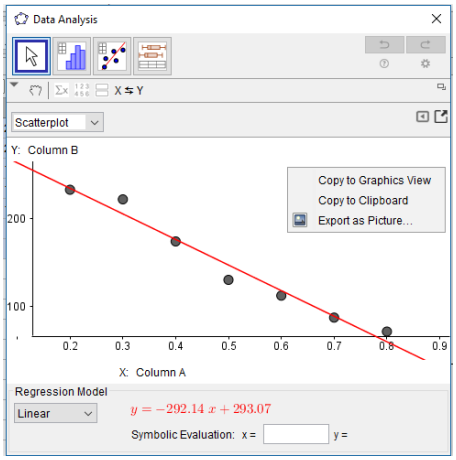
We will just use the data from the earlier light intensity experiment in section 2. You might like to try your hand using the techniques with data from some of your own experiments. I will use the current version of GeoGebra for my Windows 10 laptop.



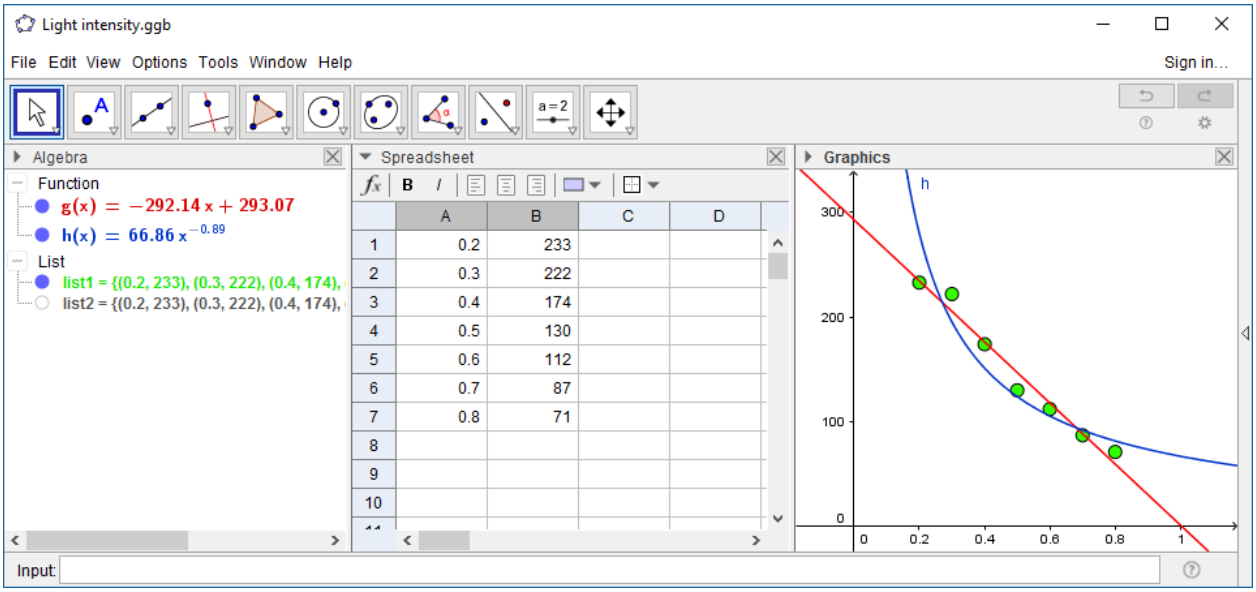
I have copied and pasted the data to columns A and B of the SpreadSheet View. Clicking on A and shift-clicking on B we can select the data to work with. There are just four icons for the various tool menus in this view. The drop-down menu for the second one includes the `Two Variable Regression Analysis' option. This is the one will use regularly for fitting functions to data. The stages are shown below. Use a right-click in the display and select `Copy to Graphics View'.





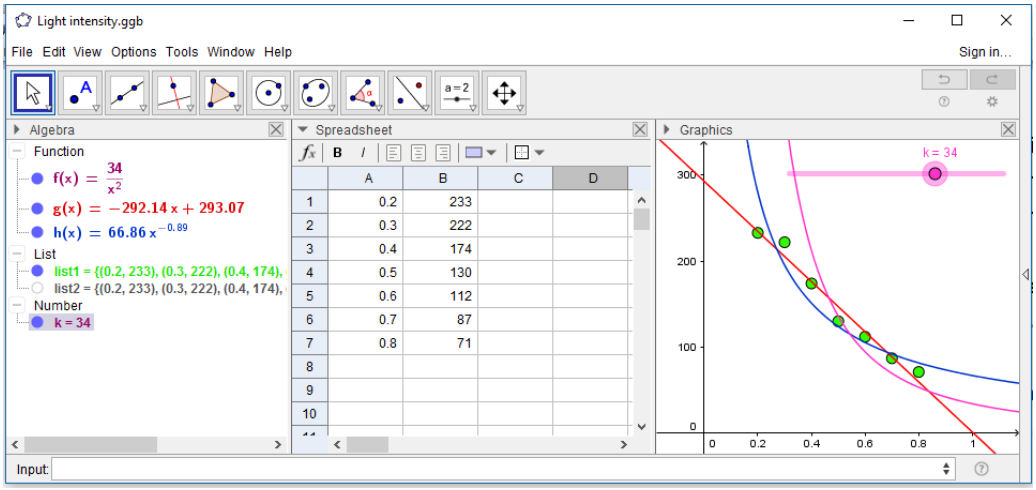


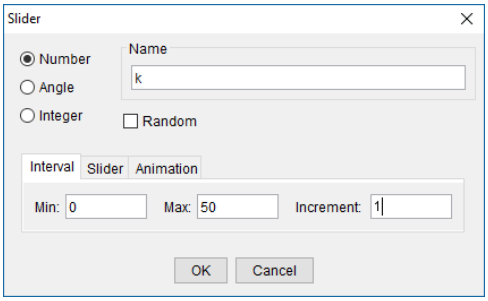
Now repeat the process, but this time select `Power' as the `Regression model'. The results are shown below with a little extra tidying up.



From the `View' menu I have added an `Algebra View' and also an `Input Bar'. You adjust the dimensions of any of the views by dragging. In the Graphics View you can use the mouse wheel to zoom in and out, as well use the mouse to drag the graph so that the axes and origin are visible. The linear function has been plotted in red, and now you can read its equation in the Algebra View. The data have been converted into a list `list1' of coordinates, and its scatterplot has shown in black. If you right-click on `list1' you can use `Set Properties' to change the colour of the scatterplot to green. You could also change the shape and size of the plotted points. When you do a second regression using the power model you get a new list, `list2' which is exactly the same as `list'. Just click in the little circle beside `list2' in the Algebra View to hide this. The equation of the power fit is also now in the Algebra View, and you can change its colour and other properties in the same way as before. You can use `File' and `Save As' to give your GeoGebra file a name and to save it.

You can experiment with your own models by defining functions. Type in your definition in the Input Bar and they will appear in the Algebra View. I am testing out an inverse square model using a `slider'. In the Graphics View we have eleven icons, and the tenth of these has an option to create a `slider'. Click in the Graphics View to position it, and a dialog box opens up. Type in a name for your variable, such as `k'. Change the Min, Max and Increment values as you like. Then click on OK. Now you can define your function as `f(x) = k/x^2' and it will appear in the Algebra View. Drag the slider to change the value of k.





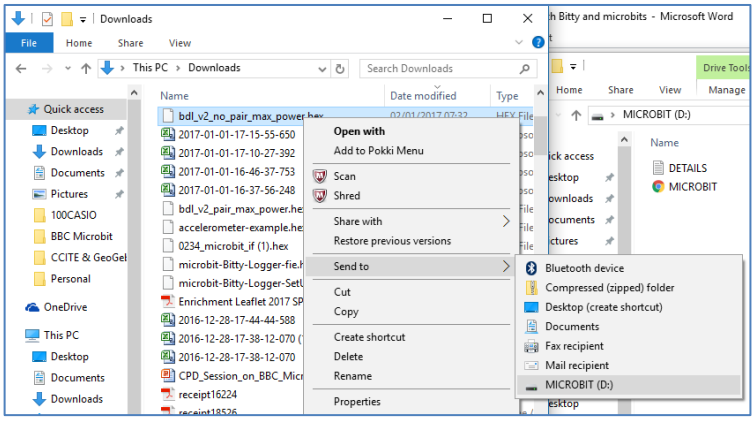
GeoGebra is a very powerful tool indeed and I hope you will find it friendly and fun to use. There are versions of it for most devices and it can be run on-line without needing to be installed. nnAt the moment there are not many options available for collecting data from micro:bits into file formats ready for use with spreadsheets. In fact I only know of one such tool, the `Bitty Data Logger'. With this we can use the micro:bit with its sensors and Bluetooth connection to send data wirelessly to a computer.

There is an article about `Data-capture, modelling and simulation' on the STEM learning site as well as other materials. There are some Teachers TV videos which illustrate interesting approaches to cross-curricular STEM, such as `Hard to teach: quadratics', `Bath bombs and rockets' and `Geometry from the playground'.

5. Logging data on a mobile device with a micro:bit and Bluetooth

The big leap forward for the micro:bit in data-logging has come from Martin Woolley with his free Bitty Data Logger App for Android and Apple. This enables you to stream live data to the App on your mobile device (in my case a Samsung Galaxy S6 Android smart phone and an Apple iPad Pro). The App graphs the data and saves it as a CSV file. This can be uploaded to a temporary file-store in the Cloud. The file can be downloaded to a laptop and be opened by a spreadsheet such as MS Excel. For more sophisticated modelling the data can then copied and pasted to other tools such as the free GeoGebra software – and merged with data captured from other sources, such as video data captured from the free Tracker software.

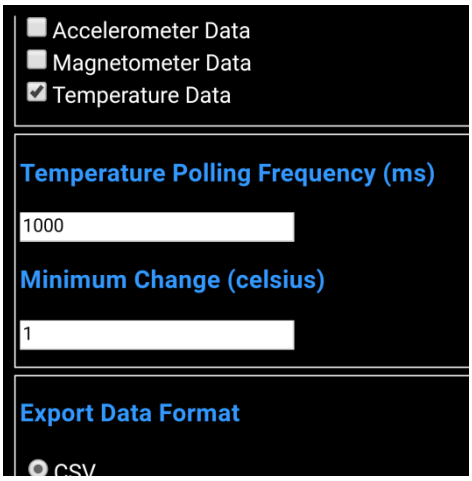
Currently this supports telemetry from the micro:bit's accelerometer, magnetometer and temperature sensors. I hope that soon this will extend to external sensors attached to the micro:bit's I/O pins. This opens up a wide range of possibilities to facilitate data-capture from experiments. These might be undertaken in a school's science labs, or by students capturing their own data during sporting and other activities, inside or outside school. Let's look first at a simple cooling model – where I will take the micro:bit from the warmth of the dining-room into the fridge to chill off, before returning it to the warmth again. The Bluetooth signal is strong enough to be received once the fridge door is shut!

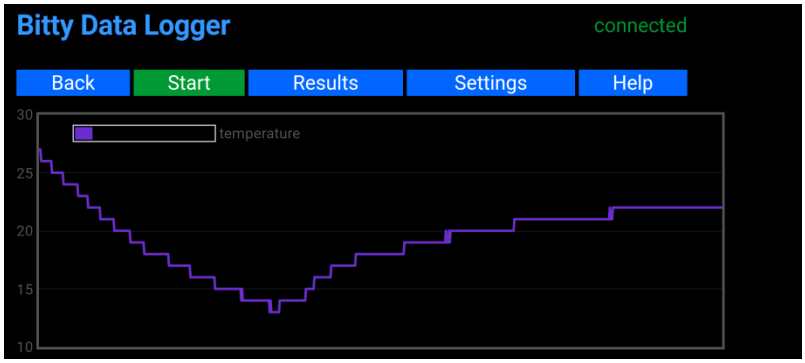


In order to get data transmitted from the micro:bit by Bluetooth you need to flash some code to it first. The Bitty hex file is here. I will be using a Windows 10 laptop for the modelling. The hex file is stored in the Downloads folder. Plug in a micro:bit using the USB cable. Find the hex file (bdl_v2_no_pair_max_power.hex) and right-click on it. Select `Send to' and then `MICROBIT (D:)'. When it has finished downing to the micro:bit you will be asked to make a circle using a single flashing LED. Once you have done this by twisting the micro:bit around, you will have calibrated the magnetometers and compass. Open the Bitty App on your mobile device. Here is a simple first experiment.

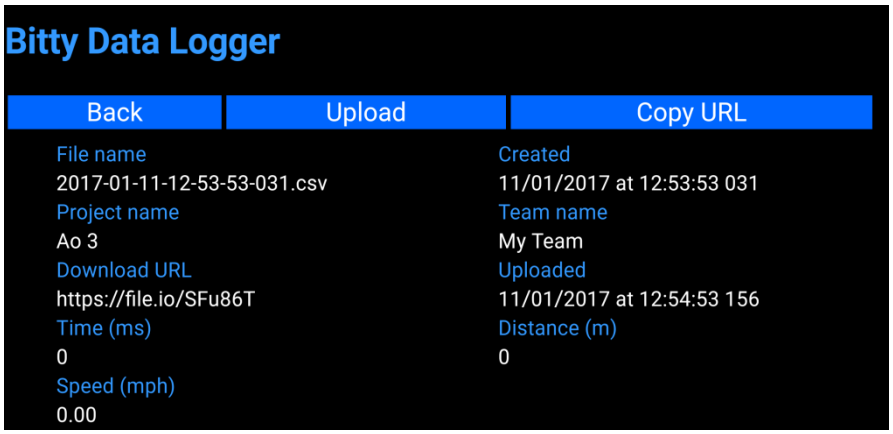
Experiment 1 **Sensing cooling and heating with micro:bit and Bitty Data Logger App**

Click on SCAN to see which micro:bits are listening. Click on the micro:bit you want to connect with. The AO bit's display will now show "C" to verify the connection. Then you can use OPTIONS to select which data you want to collect. I only want to display temperatures against time. So tick the Temperature Data box. I don't really understand the next two boxes, so I'll leave them with their default values. Tick the CSV box to save results in a format compatible with the Excel spreadsheet.





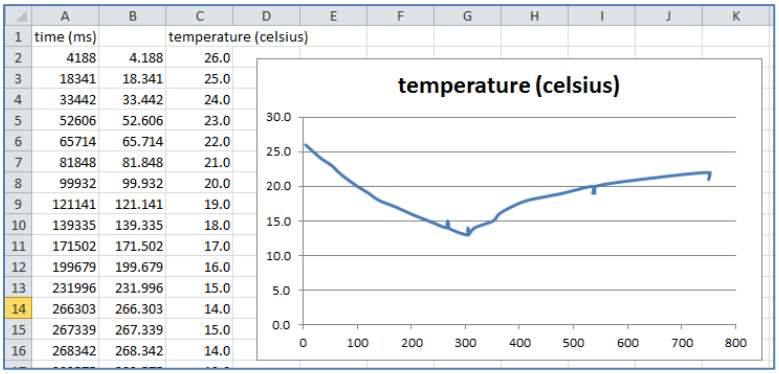
Now go back to main graph page and press the green START button, which turns red as data is being collected. Move to the fridge, open the door and pop your micro:bit in. When you think it's cool enough, take it out and return to room temperature. When the micro:bit is sufficiently warm, press the STOP button to finish data collection. Now press RESULTS to see information about the CSV data file you have created. Ignore the last few entries which are specific to the Bloodhound model rocket car competition. Use UPLOAD to send the file to the Cloud – which will give you a temporary link to a file store e.g. https://file.io/SFu86T.



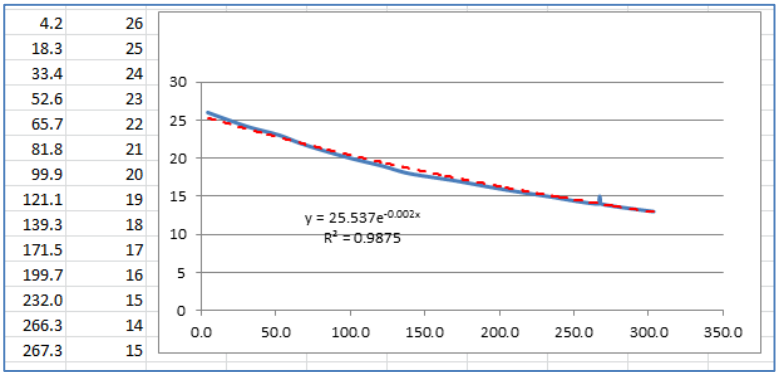
Now you can type that URL into your computer's web browser and download the CSV file. It will be saved in the Downloads folder. Right-click on its name and select "Open with" and chose MS Excel. You will find quite a few rows of title and accelerometer stuff before the rows containing temperature data. Delete these. I have inserted an additional column B, with the formula for B2 as "=B1/1000". Dragging this down converts times from column A in milliseconds to times in column B in seconds. Click on column B and shift-click on column C. Select `Insert' and `Scattergram' to make a graph of the data. Now we can see what the "temperature polling frequency" was all about. Instead of reading a temperature value at a fixed sampling rate, such as once per second, you set the temperature increment and it records the time taken to reach the next

increment. So we have asked it to read the thermometer just once every second and to record how long it takes to reach the next whole number.

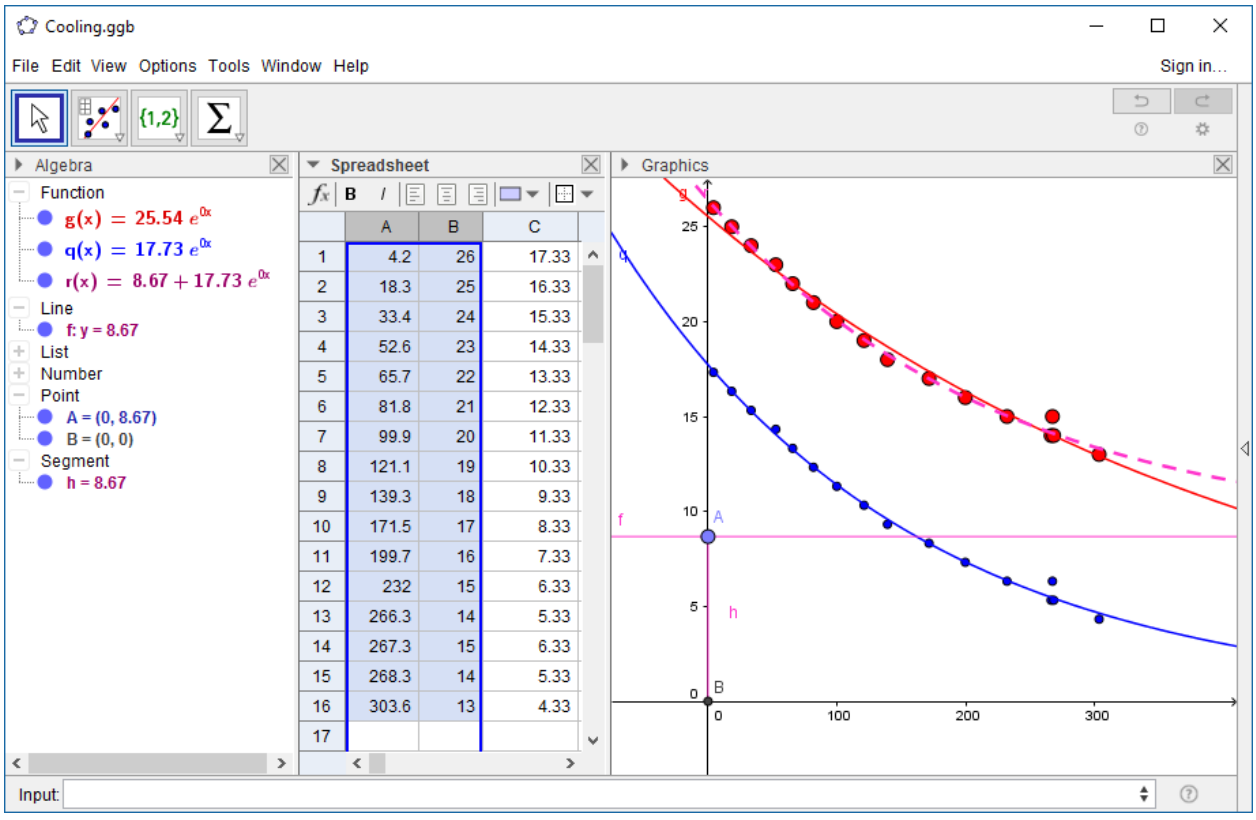
For a smoother set of results we could use a faster sample rate and a smaller threshold. Excel has the facility to compute a `trend-line' to fit the data. So we can extract the data from the cooling part of the experiment and see that an exponential curve gives a pretty good fit to the data. But Newton's `law of cooling' tells us that we need to take the ambient temperature of the fridge into consideration. So let's see how we can model this in GeoGebra using a slider.



	A	B	C
1	time (ms)		temperature (celsius)
2	4188	4.188	26.0
3	18341	18.341	25.0
4	33442	33.442	24.0
5	52606	52.606	23.0
6	65714	65.714	22.0
7	81848	81.848	21.0
8	99932	99.932	20.0
9	121141	121.141	19.0
10	139335	139.335	18.0
11	171502	171.502	17.0
12	199679	199.679	16.0
13	231996	231.996	15.0
14	266303	266.303	14.0
15	267339	267.339	15.0
16	268342	268.342	14.0



4.2	26	
18.3	25	
33.4	24	
52.6	23	
65.7	22	
81.8	21	
99.9	20	
121.1	19	
139.3	18	
171.5	17	
199.7	16	
232.0	15	
266.3	14	
267.3	15	



	A	B	C
1	4.2	26	17.33
2	18.3	25	16.33
3	33.4	24	15.33
4	52.6	23	14.33
5	65.7	22	13.33
6	81.8	21	12.33
7	99.9	20	11.33
8	121.1	19	10.33
9	139.3	18	9.33
10	171.5	17	8.33
11	199.7	16	7.33
12	232	15	6.33
13	266.3	14	5.33
14	267.3	15	6.33
15	268.3	14	5.33
16	303.6	13	4.33
17			

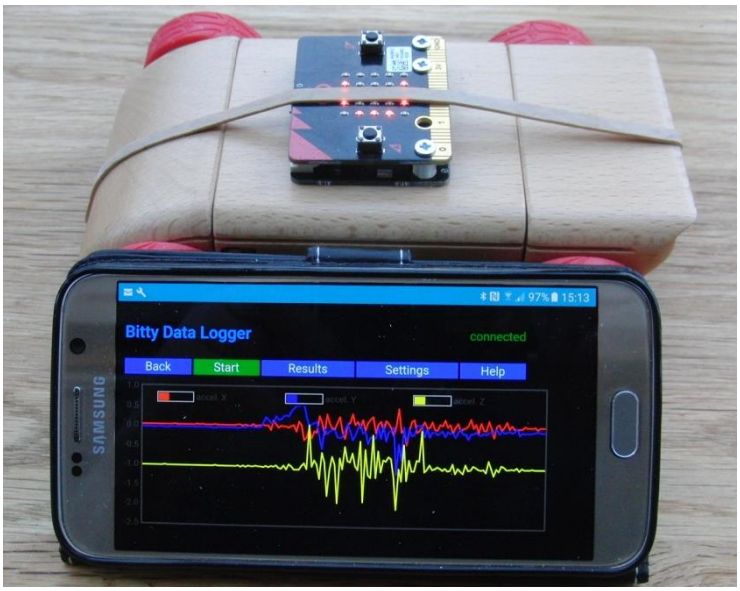
First copy the data from the time and temperature columns for the cooling phase. Open a new GeoGebra window. You will need 3 views: Algebra, Spreadsheet and Graphics – and also the Input Bar. Past the data into cell A1 of the Spreadsheet. Click on column A and shift-click on column B. From the second icon select the `2-variable regression analysis' option. This will open a pop-up window from which you can select the

regression model, e.g. exponential. Then right-click in the window and select “copy to graphics view”. This will create a scatterplot of your data together the graph of the curve fit function g(x) – both shown in red.

You can create the point A on the *y*-axis, and the line perpendicular to the y-axis at A. Create the origin B(0.0) and the segment BA. The value ***h*** shows the length of BA, which we will use for the fridge’s ambient temperature. Now select create column C using the formula “C1=B1-***h***” and copy it down. Click on column C and Ctrl-click on column A. Select “2-variable regression” and fit another scattergraph and its exponential regression q(x) – shown in blue. In the Input bar define the new function r(x) = ***h*** + q(x) – shown as the mauve dashed curve. See the effect of dragging A up and down the y-axis. So we have now used the micro:bit as a £15 digital thermometer for data-capture with the free Bitty Data Logger App, and anlaysis with the free GeoGebra multi-platform software. In order to log high temperatures we willl have to wait until we can connect an external heat sensor.

Of course the real fun comes when things are put into motion. But using the accelerometers alone in the micro:bit can produce some unexpected results as we shall see. The most revealing results are obtained when we merge data captured from the micro:bit accelerometers with that captured from video clips of the experiments using the free open-source Tracker software for video analysis. For a simple dynamical example here is micro:bit accelerometer data captured from a toy car rolling under friction.

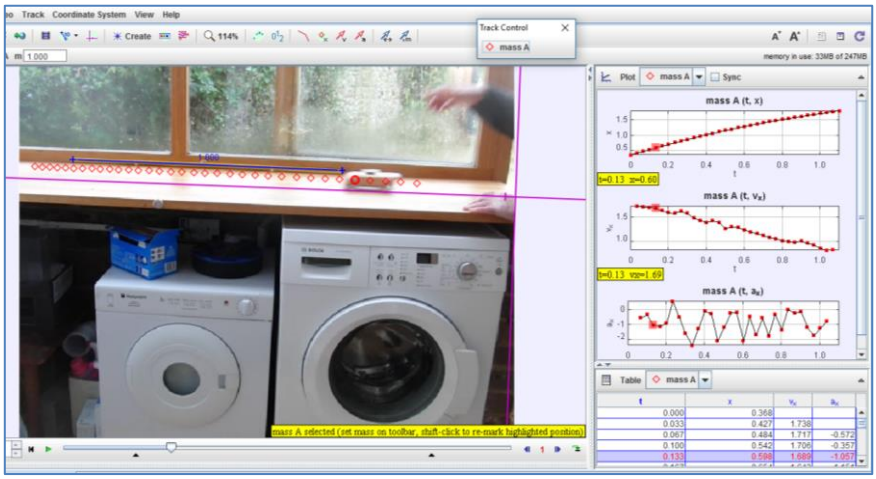
Experiment 2 Accelerations of a toy car rolling under friction using Bitty Data Logger



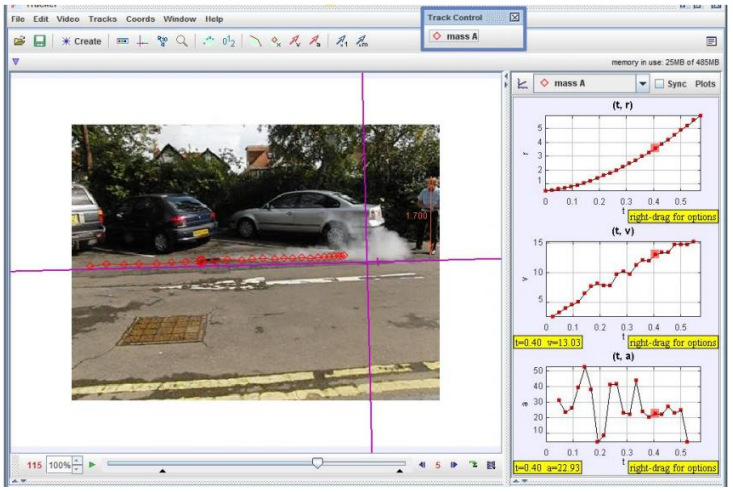
The micro:bit is set up so that the *z*-axis is vertical and the *y*-axis is along the line of travel. We would expect, under perfect conditions, that the sideways (*x*) and vertical (*z*) accelerations would be constant at zero ms^{-2} and that, once the car is give an initial thrust, its velocity will steadily decline until it comes to rest.

So there are several things to discuss about the graphs displayed by Bitty. The yellow (*z*) graph is not constant which shows it must have bumped up and down a bit during the run. Also its mean value is -1. So we can deduce that the units of acceleration are expressed as multiples of ***g***, rather than in ms^{-2}. Also that at rest the vertical `acceleration’ is shown as negative ***g***, even though there is no vertical motion. So it is really measuring force rather than change in velocity! The red (*x*) graph wiggles a small amount all over the place – again probably due to it not running quite smoothly. As we would expect, the blue (*y*) graph shows an initial positive spike representing the thrust in the ***y***-direction, and after that it wiggles about in mainly negative values. So the data really doesn’t tell us very much. For comparison we will now compare it the video data captured with Tracker using a 30 fps video clip.

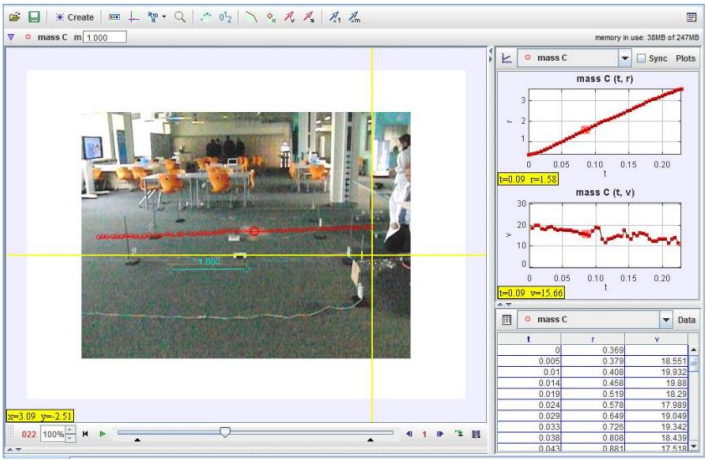
In the experiment there is a 1m yellow folding rule laid out as a frame of reference – shown marked in blue. The axes are arranged so that *x*-axis lies along the path of the car. Position data of the front wheel is captured in the table. The *y*-data values have been removed and numerical values for the velocities and acceleration computed. The three graphs displayed are for the displacement, velocity and acceleration against time. The first two plots conform to our intuition, but the accelerations are as devoid of meaningful information as those recorded by the micro:bit.



Here are some sources of information, tutorials and worked examples of Tracker used for video analysis of motion. The first is a screen shot taken from the report of a CDP session run by Ian Galloway about a model Bloodhound SCC rocket car running on a wire in a car park at Southampton University. The results are pretty similar to those with the unpowered friction car we've already seen.

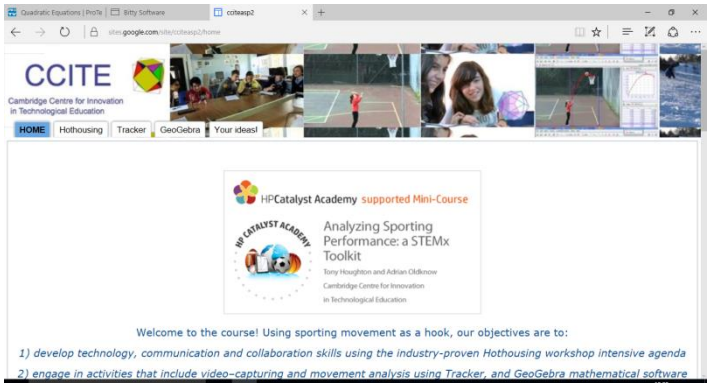


The second is from a report I wrote on the launch of the Manchester Bloodhound SSC Education Centre in Manchester where compressed air balloons are being run on strings to see the effect of shape on air resistance.



You could easily suspend a micro:bit from the balloon. Another simple experiment is to make different size parachutes to investigate terminal velocity. This would be an ideal exploration with both Bitty Data Logger and Tracker video analysis.

Dr. Tony Houghton and I developed an extensive MOOC on the use of Tracker and GeoGebra for Analysing Sporting Performance. This includes many video clips, Tracker files and GeoGebra files for you to practice with.



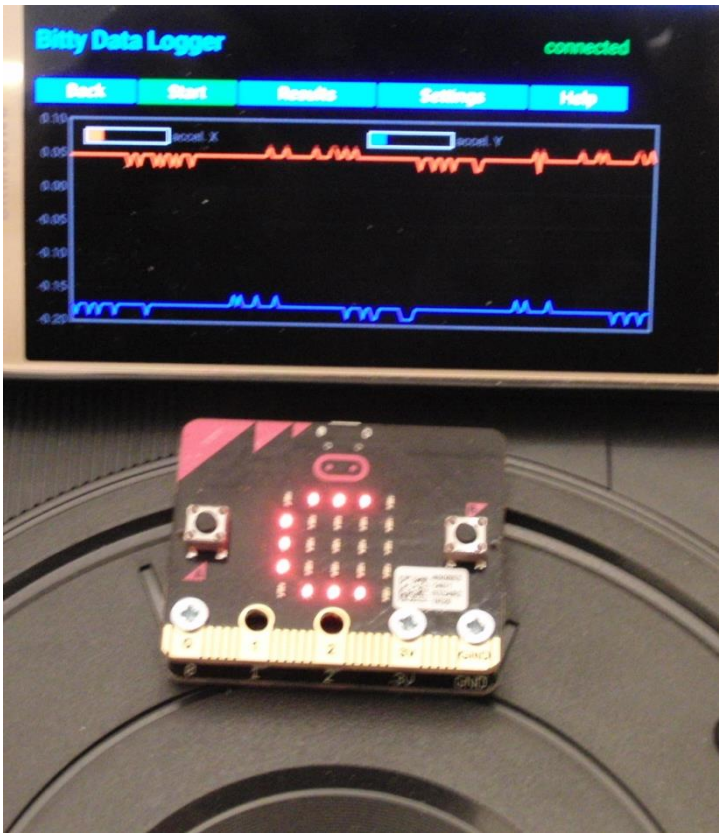
Some of these were taken during a Bloodhound model rocket car competition between some West Sussex secondary schools. Others were used at Bohunt School's STEM festival for families. Further examples are here.

There is an extensive collection of STEM material produced by Ian Galloway, Linda Tetlow and me such as the Maths In Motion book which contains many example of the use of Tracker which could easily be adapted for use with the Bitty Data Logger App and the GeoGebra software.

Our final worked example is for motion in a horizontal circle using a USB turntable connected to the laptop.

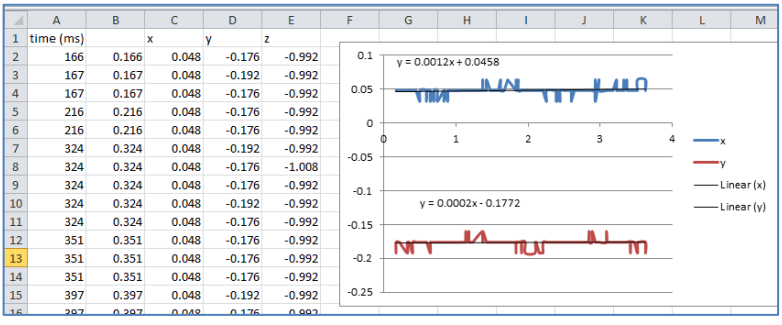
Experiment 3 Motion in a horizontal circle using Bitty Data Logger, Tracker and GeoGebra

Here we have set up Bitty to collect and graph data from the *x*- and *y*-accelerometers from a micro:bit mounted horizontally on a turntable rotating at 33 1/3 rpm. As expected the *z*-acceleration (not graphed) is constant at -1 (in units of ***g***). The *x*- and *y*-axes are fixed in the frame of reference of the rotating micro:bit – and appear to be constants. The red acceleration is in the positive *y*-direction which is the instantaneous direction of travel of the micro:bit and looks to have a constant value of about +0.05***g***, or around 0.5 ms^{-2}. The blue acceleration is about -0.18 in the y-direction – showing a constant acceleration towards the centre of about -1.8 ms^{-2}.



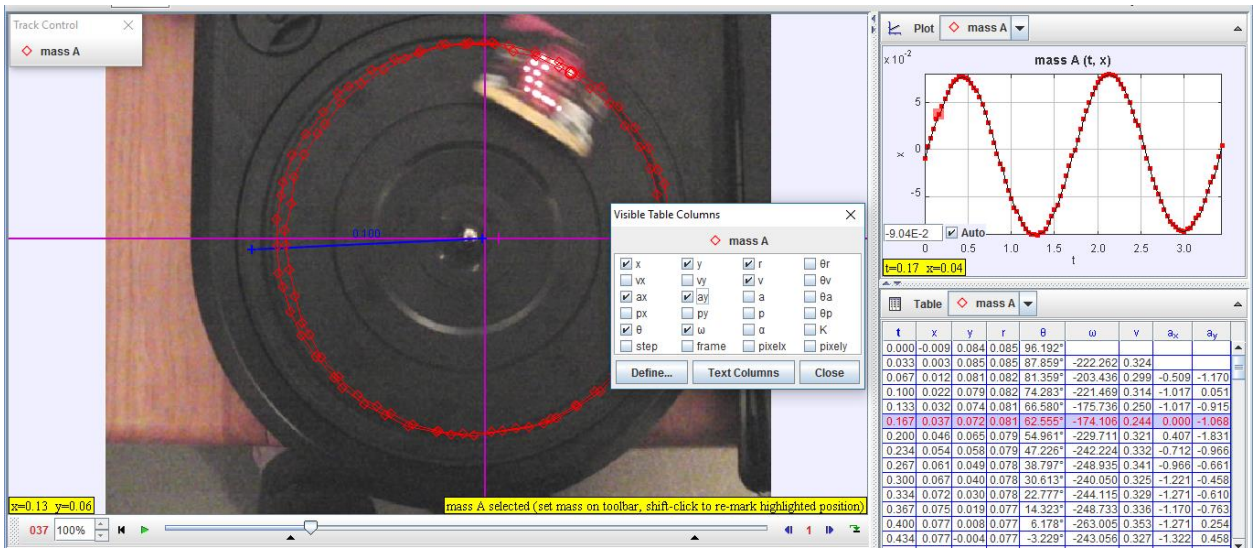
Uploading the Bitty data CSV file and then downloading from the https://file.io/xxxxxx URL we can open this in MS Excel. After a bit of tidying up we can plot the *y*- and *z*-accelerations against time in seconds.

Fitting linear trend-lines to the data gives values of +0.0458 and -0.1772 as the tangential and radial accelerations. The centre of the micro:bit is around 0.08 from the centre of rotation. The turntable is rotating at a constant rate of 33.33 revs per minute. Now would be a good time to read



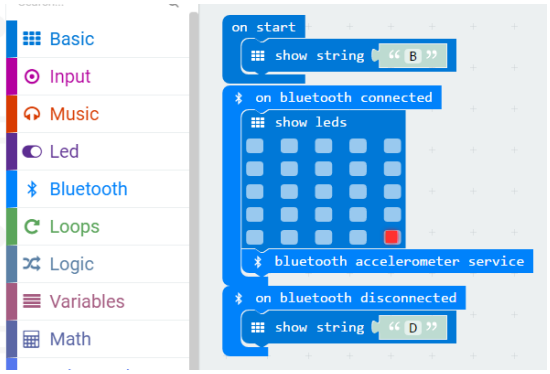
	A	B	C	D	E
1	time (ms)		x	y	z
2	166	0.166	0.048	-0.176	-0.992
3	167	0.167	0.048	-0.192	-0.992
4	167	0.167	0.048	-0.176	-0.992
5	216	0.216	0.048	-0.176	-0.992
6	216	0.216	0.048	-0.176	-0.992
7	324	0.324	0.048	-0.192	-0.992
8	324	0.324	0.048	-0.176	-1.008
9	324	0.324	0.048	-0.176	-0.992
10	324	0.324	0.048	-0.192	-0.992
11	324	0.324	0.048	-0.176	-0.992
12	351	0.351	0.048	-0.176	-0.992
13	351	0.351	0.048	-0.176	-0.992
14	351	0.351	0.048	-0.176	-0.992
15	397	0.397	0.048	-0.192	-0.992
16	397	0.397	0.048	-0.176	-0.992

up on the physics of circular motion and to see how our results match up to the theory. We can also easily capture the motion on a video clip and analyse it in Tracker.



Here the axes are set in relationship to the static turntable, and so the *x*- and *y*- directions are not aligned with the directions of the micro:bit's accelerometers. Maybe it is possible to have a revolving frame of reference in Tracker – but I have yet to discover how to do that! As well as creating a table of times and coordinates, Tracker will estimate a pretty wide variety of additional data based on these, I have collected quite a number above including ***r*** – the distance from one of the LEDs to the centre. This is, as expected, pretty well constant – the error coming from either camera shake or inaccuracies tracking the particular pixel I am concentrating on. ω is the angular velocity in degrees per second, which should also be a constant for our turntable spinning at 33.33 rpm. θ is the angle turned through in degrees. V is the value of the tangential velocity. a_x and a_y are the components of the acceleration in the *x*- and *y*-directions of Tracker's frame of reference. Knowing the angle θ we should be able to translate accelerations between the two different frames of reference. For more extensive analysis we can copy and paste from either the Excel or Tracker file (or both) into a GeoGebra spreadsheet.

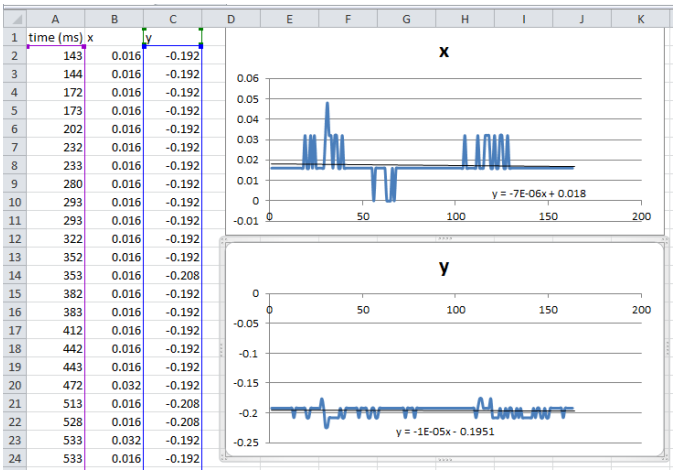
Some ideas to try.



1. It would be easier for Tracker to pick up position data if only the pixel nearest the accelerometers was illuminated. Here is some code to try to set up the micro:bit for Bitty with only the Bluetooth accelerometer services selected. Here is the code to compile and send from the PXT editor. You need to use `Add Package' to bring in the Bluetooth blocks in place of the Radio ones.

2. If you have a turntable then how about repeating a simple experiment to explore how the radial and tangential components depend upon the distance *r* from the centre to the accelerometer sensors? I am starting with r = 75cm and using a fixed 33 1/3 rpm angular speed. We can collect data with Bitty from the x- and y-accelerometers. They will be stored in a CSV file which we can upload the file to the cloud, and then download it from the given URL to the Downloads folder on a laptop.

Double-clicking on the file opens it in Excel (or your chosen spreadsheet). Here are the scatterplots of the *x*- and *y*- accelerations, which are a bit `jumpy'. Fitting a linear trendline to each we can estimate a constant value for 0.018 for the *x*-acceleration (tangential) and -0.195 for the *y*-acceleration (towards the centre).



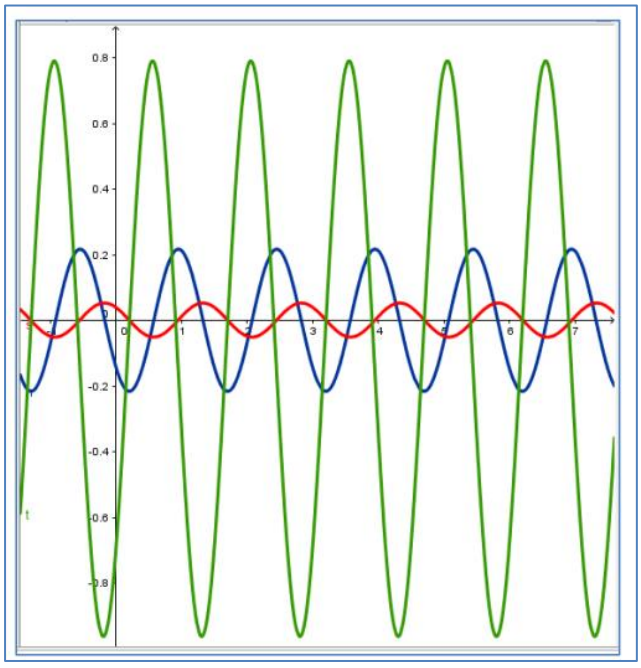
	A	B	C
1	time (ms)	x	y
2	143	0.016	-0.192
3	144	0.016	-0.192
4	172	0.016	-0.192
5	173	0.016	-0.192
6	202	0.016	-0.192
7	232	0.016	-0.192
8	233	0.016	-0.192
9	280	0.016	-0.192
10	293	0.016	-0.192
11	293	0.016	-0.192
12	322	0.016	-0.192
13	352	0.016	-0.192
14	353	0.016	-0.208
15	382	0.016	-0.192
16	383	0.016	-0.192
17	412	0.016	-0.192
18	442	0.016	-0.192
19	443	0.016	-0.192
20	472	0.032	-0.192
21	513	0.016	-0.208
22	528	0.016	-0.208
23	533	0.032	-0.192
24	533	0.016	-0.192

Repeating this experiment for *r*=65, 55, 45, 35 and 25cm will provide a data set from which you model both *x* against *r* and *y* against *r*.

Repeating the whole experiment for 45 rpm and 78 rpm will also allow you to get a feel for the effect of increasing and decreasing the speed of rotation. Maybe you could build your own turntable using a motor with a variable resistor so you can produce more data values. You could visit a playground which has a turntable with no motor and collect samples to bring home for modelling.

3. Try designing other experiments with things in motion. Inclined planes, swings and springs are all useful sources. For a worked example see the spring-mass (SHM) system analysed here.

Further ideas. If you are building a model rocket car for the Bloodhound "Race to the Line" challenge can you use the Bitty Data Logger to capture accelerations – and maybe also use video and Tracker. How about some other experiments, like dropping a bouncy ball with an micro:bit inside? Or motion in a vertical circle with a bicycle wheel? Or a simple pendulum with a micro:bit attached? Or putting a micro:bit in your pocket and running, jumping, bouncing etc. Or using Velcro to secure a micro:bit to apparatus like a sled, skate-board, bike etc.



Whatever you try, can you post some information on the STEM Learning group sites such as this (for Student Digital Ambassadors) or this (for the micro:bit). There is also now an excellent Wiki here.