

Some ideas for stimulating cross-curricular STEM through dynamics experiments with BBC micro:bits

Adrian Oldknow

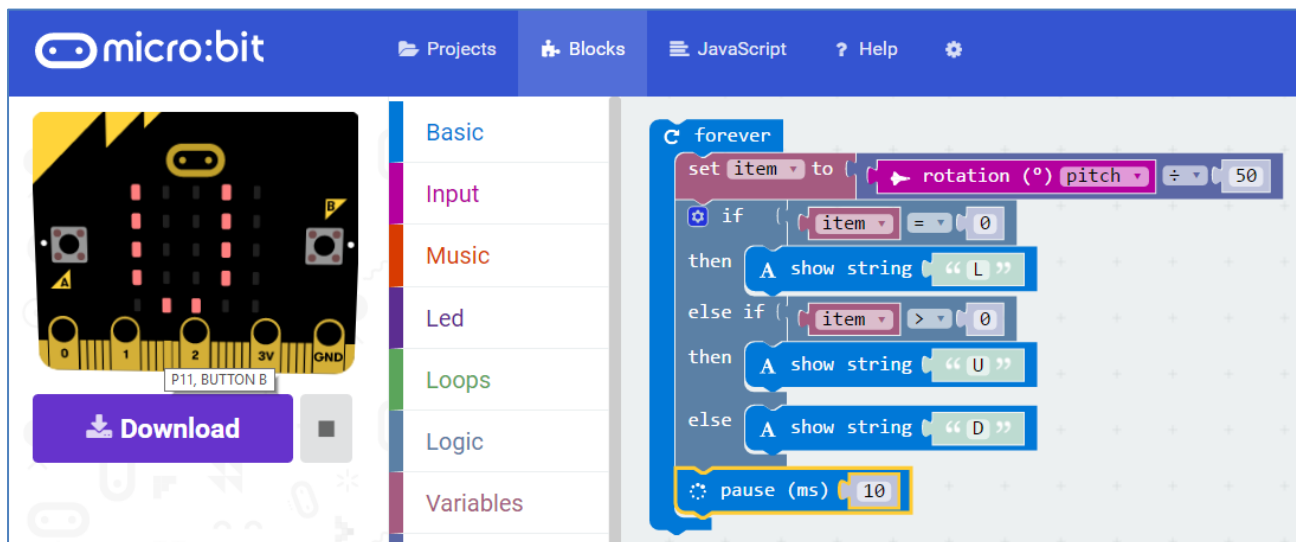
adrian@ccite.org

9th January 2017

The [BBC micro:bit](#) has been distributed free to 1 million 11-year olds in the UK, and is now on sale, together with battery pack and USB cable, for c£15. Most of the educational resources for the micro:bit concentrate on programming the device using blocks (e.g. Microsoft's PXT editor), Python, Javascript or C++. This article contains some simple examples to help interested readers (students, teachers, parents and others) to make a start with the micro:bit (or m:b for short) for what is sometimes called 'physical computing'. It also considers ways in which the m:b can be used to collect data and transmit for scientific experiments. We will start with a simple (free) example which doesn't even need a real m:b!

Experiment 1: turning the micro:bit into a spirit-level

Enter the Microsoft PXT blocks editor from this link: <https://pxt.microbit.org/>. Create a new Project.



The '**forever**' and '**show string**' commands are in Basic blocks menu. '**set to**' and '**item**' are found in the Variables menu. '**if..then..else if..then..else**' is found in the Logic menu. '**rotation**' is found in the Input More menu. The **divide** block is found in the Math menu, and the **compare** block in the Logic menu. So how does the program work? It continuously checks the angle being returned by the '**pitch**' sensor. Dividing this angle by 50 returns a value for the variable '**item**' which you could read using the '**show number**' command from the Basic block. This is a single digit signed number between -4 and +4. We are just going to test whether this is a positive, negative or zero number. If it's zero we display the letter L for Level, if it's negative we display D for Down, and if it's positive we display U for Up. Then we have a slight pause before doing the job again. You test this with on screen emulator by clicking the mouse somewhere inside the image of the m:b at the top left. Check that you can get it to display each one of the 3 letters. So this is a way to use the m:b's built-in sensors to control an output. It simulates the way a smart-phone or tablet senses the orientation of the way in which you are holding the device – and so is able always to display text in the right direction for you to read clearly. If you click on the '**JavaScript**' icon you can see the text equivalent of the Block program, shown below. Click on the '**Blocks**' icon again to see the graphical version. Click on '**Projects**' again to save your project with a sensible name, like '**Spirit Level**'.

In order to test the program 'live' you now need to connect a m:b to the computer with the USB cable. The computer will recognise the m:b as an external memory device and give it a drive name such as 'D:'.

When the hex file has finished downloading you can right-click on it and select '**Show in folder**'. In my '**Downloads**' folder I find the new file called "microbit-Spirit-Level.hex". Now right-click on the file name and select '**Send to**', and then 'MICROBIT (D:)'. The led under the m:b should flash as the code is sent to your m:b. If this is the first time you have done this – congratulations!

If you connect a battery box, you can disconnect your m:b from the computer. You now have a program running forever (maybe) in your own £15 autonomous device. If you press the Reset button it will start running again. If you disconnect the battery the program will be saved, and when you reconnect the battery your program will start running again.

All sorts of objects have microprocessors built into them to carry out such monitoring and other tasks. We now tend to use the word "*smart*" to indicate that a device (phone, TV, fridge, house, car engine ...) has an in-built computer which can sense conditions and make actions depending on them. The technical phrase for such devices is "*embedded systems*". Embedded systems which can communicate with others via the web are known as the "*Internet of Things*" or **IoT**.

The BBC has worked with its partners such as Microsoft, ARM, Lancaster University, the IET and Kitronik to develop a versatile, powerful and robust little device which allows us to create our own electronic solutions to problems and to design our own smart devices. This brains of the m:b is an ARM mbed processor which is at the heart of many smart systems in current use. The m:b has also been designed to connect to external electronic devices such as sensors and inputs so you can extend its capabilities.

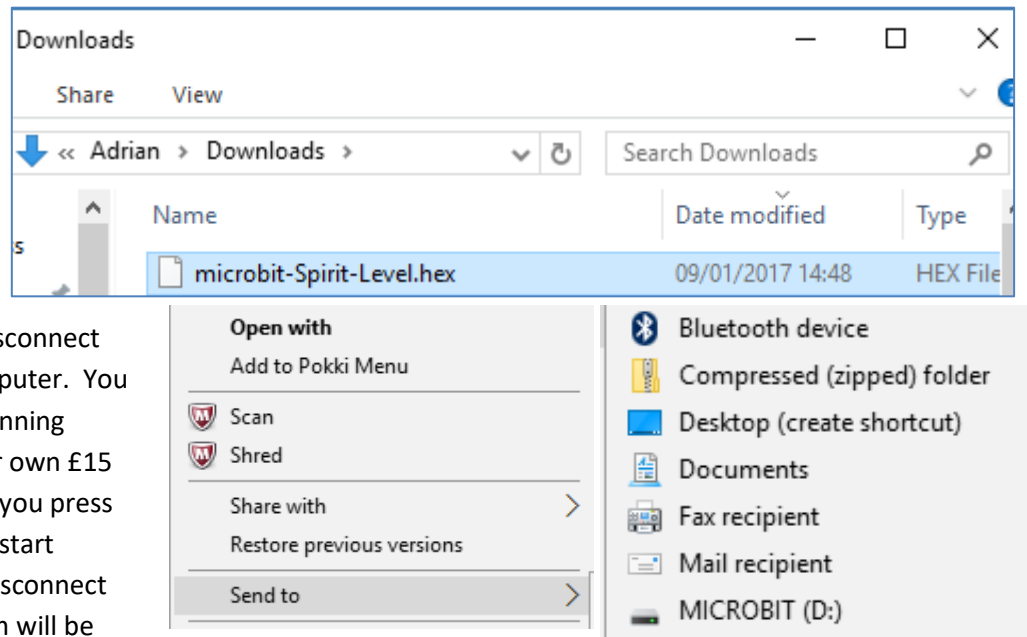
For the next experiment we will build a physical electronics circuit to simulate room lighting with an on/off and dimmer switch. You may already have suitable supplies of resistors, LEDs and other electronic kit. I am going to follow one of the worked example projects in the Kitronik '[Inventor's Kit for the BBC micro:bit](#)'. This costs £25 and provides a very neat electronic workbench based around an edge connector and break-out pins for the m:b, a large capacity 'breadboard' and a variety of gadgets and connectors.

Experiment 2 Dimming an LED using a potentiometer

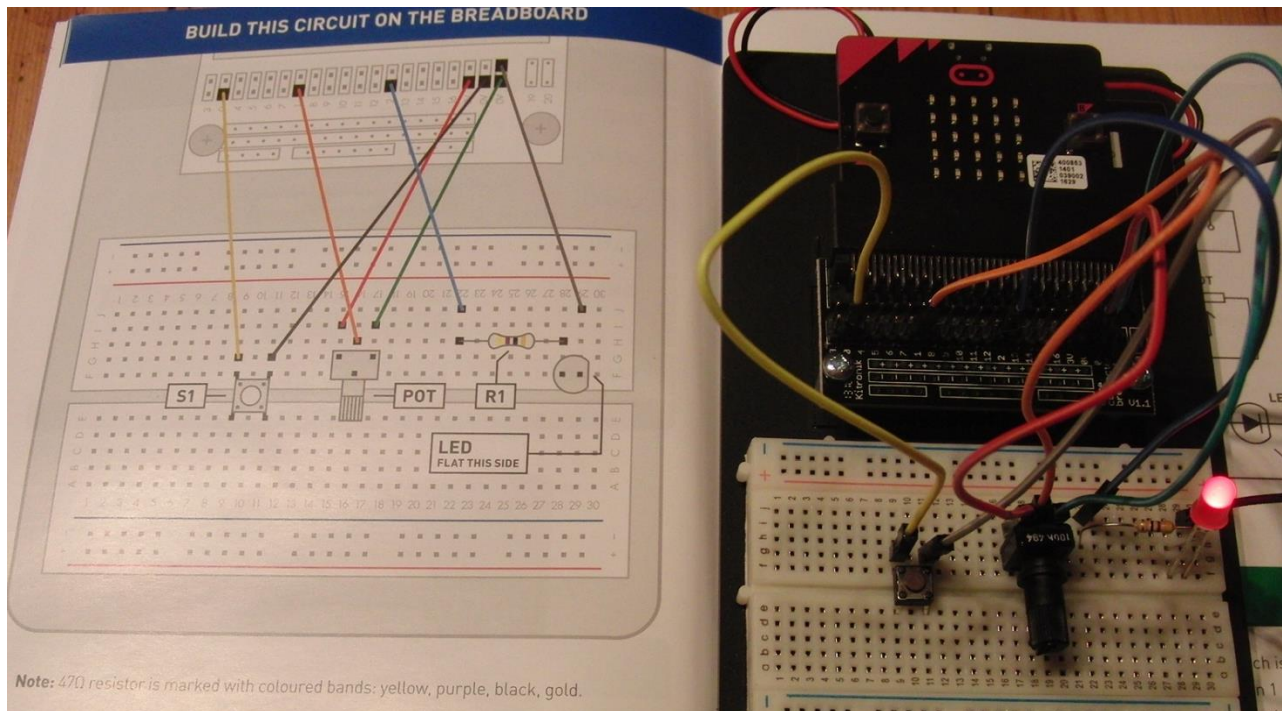
```

1 let item = 0
2 basic.forever(() => {
3   item = input.rotation(Rotation.Pitch) / 50
4   if (item == 0) {
5     basic.showString("L")
6   } else if (item > 0) {
7     basic.showString("U")
8   } else {
9     basic.showString("D")
10  }
11  basic.pause(10)
12 })
13

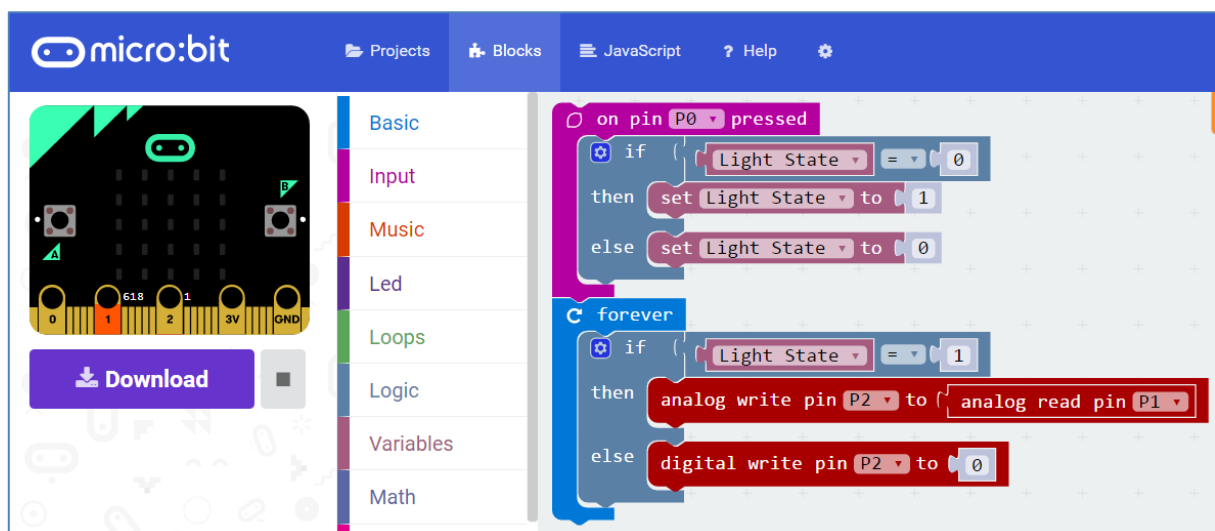
```



The photograph below shows page 24 of the tutorial with the circuit diagram we need to build. On the right is the assembled Inventor's kit with the programmed micro:bit plugged into the edge connector, with the battery box tucked away underneath. The coloured leads slip over the I/O pins connected to the m:b and plug into holes in the bread board. We are using three I/O pins, 0, 1 and 2. Also the GND and +3V power pins. The components are a push switch, a potentiometer, a red LED and a 47Ω ohm resistor.



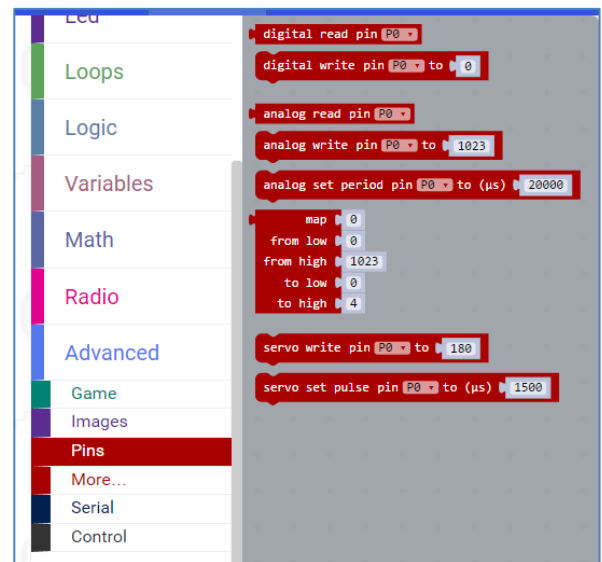
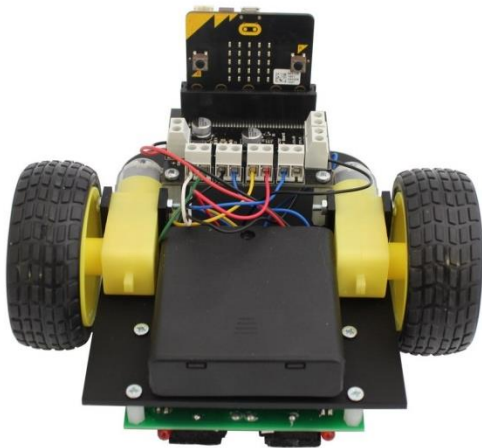
Here is the program written in the current Microsoft PXT editor. I have called it 'Dimmer switch'. We use a variable called '**light state**' to tell whether the LED is switched on (1) or off (0). So the first bit of code just tells the m:b to use the push switch attached to pin P0 as a flip-flop to change the state of the LED.



The second block uses the potentiometer attached to the analog pin P1 to control the brightness of the LED connected to the analog pin P2. To access the red commands use the Advanced blocks menu and then select the Pins menu. When you have flashed the program to the m:b you can use the push button to switch the LED on and off, and turn the spindle attached to the potentiometer to control the brightness of the LED.

The 10 well described experiments give a pretty good feel for how to design and control your own circuits and devices. There is now an increasing number of other fun projects for use with micro:bits on the market. Here are a few examples:

Kitronik's '[Line-following buggy](#)' at £26



[DIMM](#) and [UFO](#) from Binary Bots (m:b included, £40 each)



Cheap sensors and accessories are available from '[microbit accessories](#)', such as [head-phone adaptors](#), a natty [plant watering project](#) and the [chick-bot robot](#). Using the head-phone adaptor you can also attach speakers to the m:b. I also recommend two other devices.

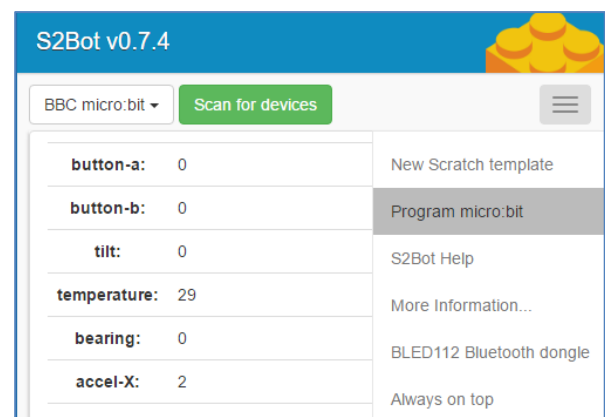
The Kitronik [MI:power board](#) costs £5 and provides a robust casing for the m:b as well as compact power from a coin battery.

To connect an m:b to a laptop via its Bluetooth Low Energy (BLE) radio I recommend getting a [plug-in USB](#) BLE dongle from Picaxe at £12.

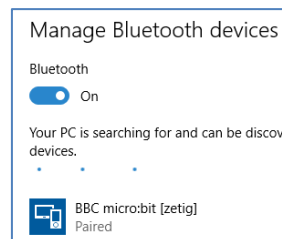
Using this we can read the m:b's sensors directly through Clive Seager's free [S2Bot App](#) which supports many Bluetooth enabled devices including m:b.

Experiment 3 Reading m:b sensors on a laptop using the S2Bot App.

The instruction for using S2Bot with a m:b are [here](#). From the drop down menu select the device you want to connect to – in our case the BBC micro:bit. Then attach your m:b to the computer with the USB cable. Click on the menu icon at the top right and select 'Program m:b'.



This will flash the BLE hex code to your m:b. You may be asked to calibrate some of the sensors by twisting the m:b about to create a large letter 'O' with the LEDs. When you have done this you will see a happy face, and you can detach your m:b and connect it to batteries. Now you can click on 'Scan for devices' – which will give a list of connectable devices which should include the m:b you want to use. When the connection symbol turns from red to green you will see that readings from many of the m:b's on board sensors are now available. I am in my warm dining room at 29° C and the micro:bit is sitting horizontally on the table pointing North. The horizontal (x- and y-) accelerations are virtually zero (showing that the m:b isn't exactly horizontal, but very nearly!). The vertical (z-) acceleration is -64 which corresponds to acceleration due to gravity (9.81 ms^{-2}) downwards. So we now know how to calibrate accelerations by multiplying by a factor of $9.81/64 \approx 0.153$. Can you arrange the m:b so that the y-acceleration is +64, or -64?



BBC micro:bit	Scan for devices
button-a:	0
button-b:	0
tilt:	0
temperature:	29
bearing:	0
accel-X:	2
accel-Y:	-5
accel-Z:	-64

Clive Seager has produced the S2Bot App to enable devices like the m:b to be recognised in [MIT's Scratch](#) (version 2) programming language. If you run the Scratch editor you should see that the second red light on the App now turns green to show that Scratch is connected.

Experiment 4 Collecting data from the m:b's sensors in Scratch

From the S2Bot App menu select the 'New Scratch template' option, and save the file 'microbit_template' to a suitable folder. In Scratch use File and open the 'microbit_template'. Select the 'More Blocks' menu and you will see the m:b specific blocks now available to you. Ticking any of the black sensor blocks displays the current readings on the Scratch display. My demonstration program uses some variables to control the data-logging, and stores readings into lists. The data collected are times in seconds and raw accelerations read from the z-accelerometer. All I am doing is shaking the m:b up and down a bit.

Can you think of interesting things to do in this environment? Could you plot graphs from the data? Can you use the m:b to control Scratch animations. There are some ideas in my 'Switched On' (the Computing At School Group's termly newsletter) article 'Scratch Goes Ballistic' on pp 12-13 of the [Spring 2016](#) edition, as well as on the [STEM Learning site](#).

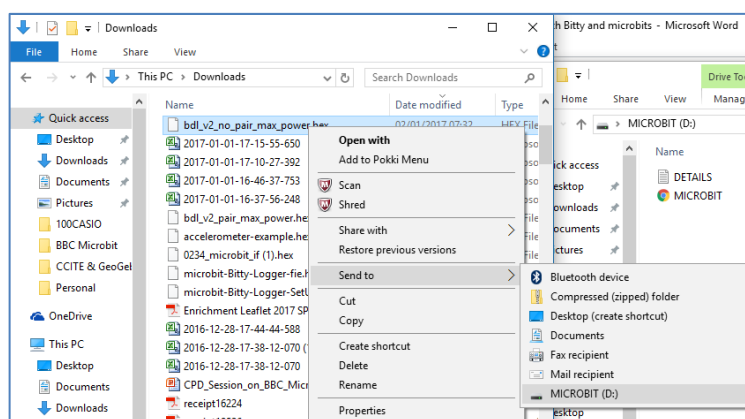
So we can use m:b with its sensors and Bluetooth connection to send data wirelessly to a computer.

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](#)'.

The big leap forward for the m:b in data-capture 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.

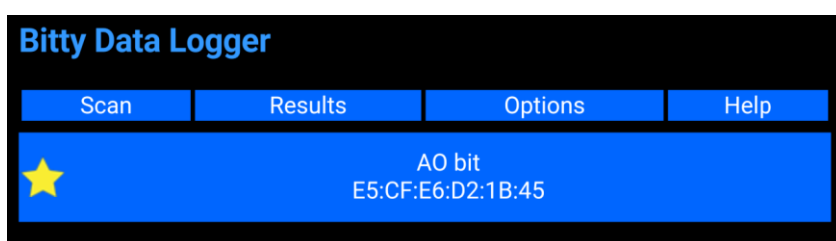
Currently this supports telemetry from the m:b's accelerometer, magnetometer and temperature sensors. Soon this will extend to external sensors attached to the m:b's I/O pins. This opens up a wide range of possibilities to facilitate data-capture from experiments. These might be undertaken in a school's 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 m:b 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 m:b 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 downloaded into the Downloads folder. Plug in a m:b 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 downloading to the m:b you will be asked to make a circle using a single flashing LED. Once you have done this by twisting the m:b around, you will have calibrated the magnetometers and compass. Open the Bitty App on your mobile device. Here is a simple first experiment.

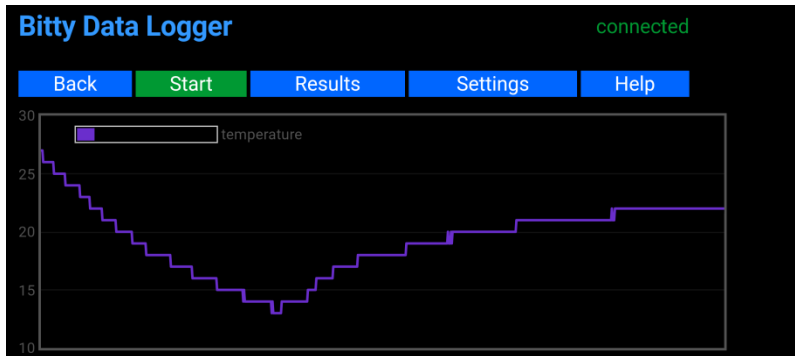


Experiment 5 Sensing cooling and heating with m:b and Bitty Data Logger App

Click on SCAN to see which micro:bits are listening. Click on the m:b 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 the results in a format compatible with e.g. MS Excel spreadsheet.



☐ Accelerometer Data
☐ Magnetometer Data
☒ Temperature Data

Temperature Polling Frequency (ms)
 1000

Minimum Change (celsius)
 1

Export Data Format
☒ CSV

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 m:b in. When you think it's cool enough, take it out and return to room temperature. When the m:b 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>.

Bitty Data Logger

Back Upload Copy URL

File name
 2017-01-11-12-53-53-031.csv

Project name
 Ao 3

Download URL
<https://file.io/SFu86T>

Time (ms)
 0

Speed (mph)
 0.00

Created
 11/01/2017 at 12:53:53 031

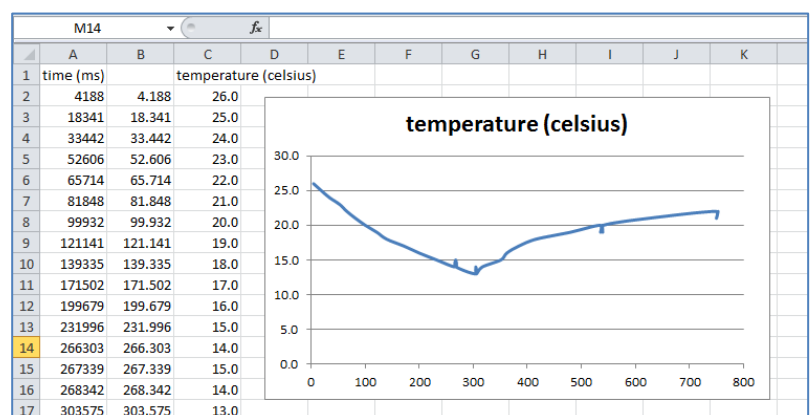
Team name
 My Team

Uploaded
 11/01/2017 at 12:54:53 156

Distance (m)
 0

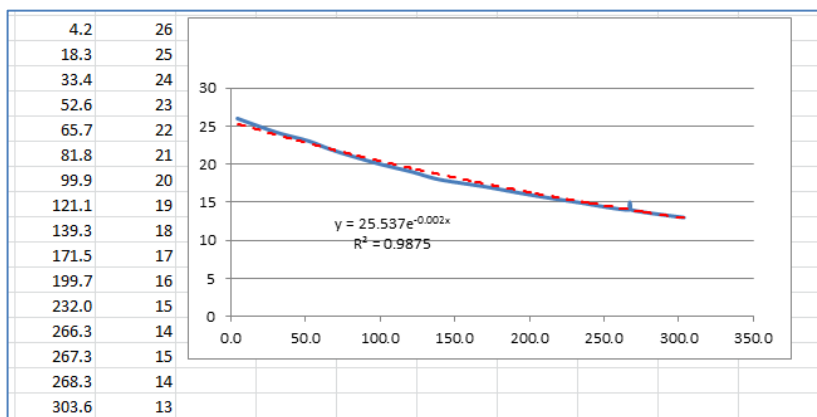
Now you can type that URL into your computer's web browser and download the CSV file. It will be saved in the Downloads folder. Right-click on it's name and select "Open with" and chose MS Excel. You will find quite a few rows of title and accelerometer stuff before the rows containing temperature data. Delete these. I have inserted an additional column B, with the formula for B2 as $=B1/1000$. Dragging this

down converts time from column A in milliseconds to time 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

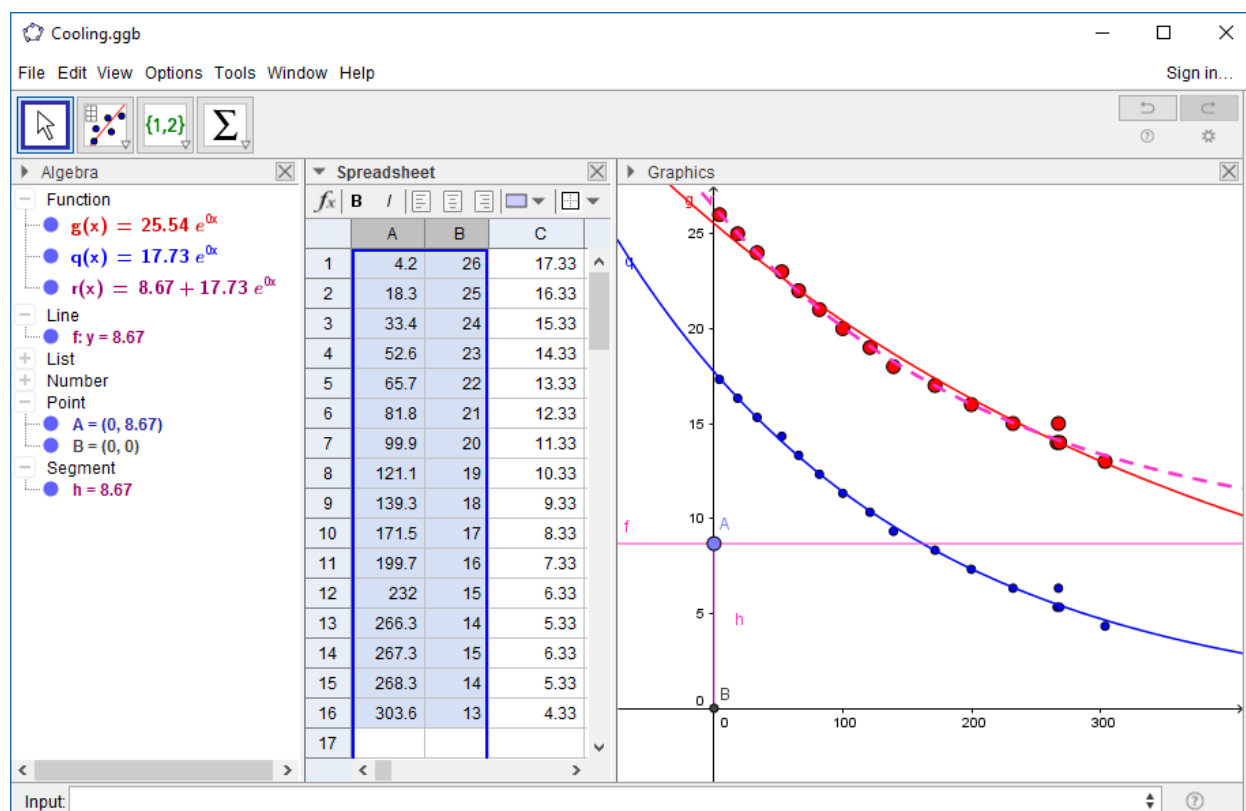


reach the next increment. So we have asked it to read the thermometer just once every second and to record how long it takes to reach the next whole number.

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. 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.



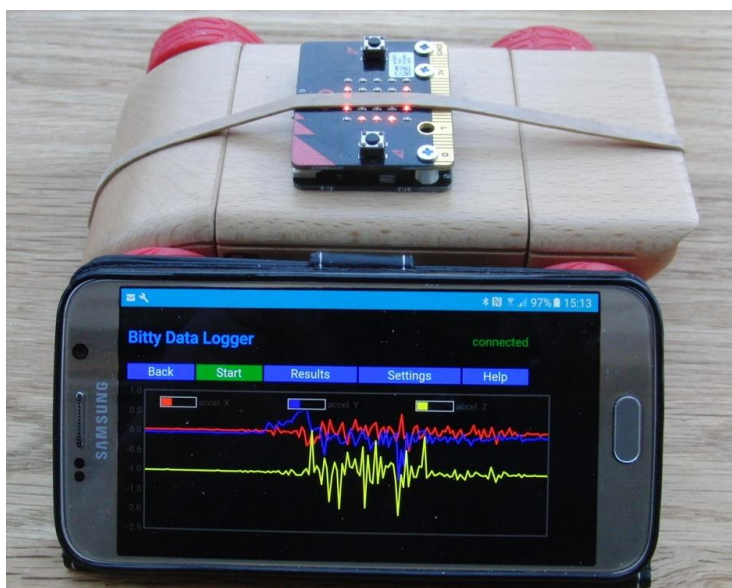
You can create the point A on the y-axis, and the line perpendicular to the y-axis at A. Create the origin B(0.0) and the segment BA. The value h shows the length of BA, which we will use for the fridge's ambient temperature. Now select create column C using the formula " $C1=B1-h$ " and copy it down. 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 analysis with the free GeoGebra multi-platform software.

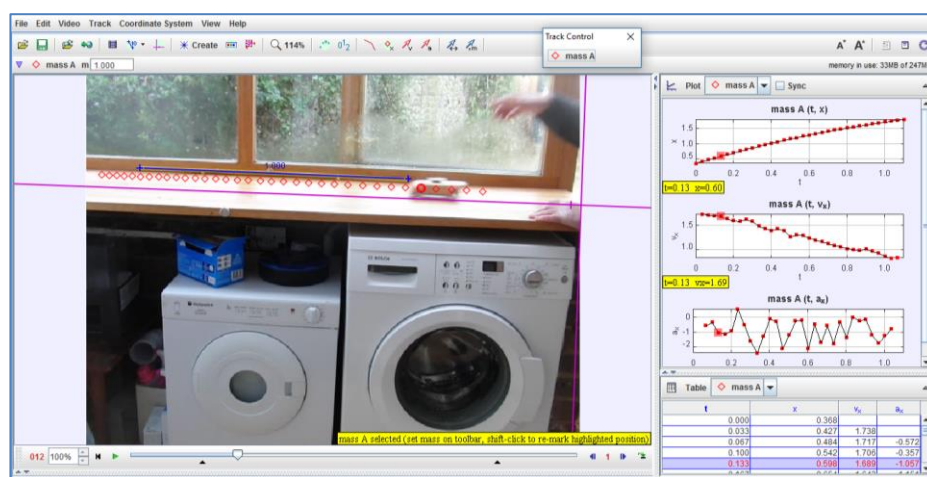
Of course the real fun comes when things are put into motion. But using the accelerometers alone in the m:b can produce some unexpected results as we shall see. The most revealing results are obtained when we merge data captured from the m:b accelerometers with that captured from video clips of the experiments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

Experiment 6 Accelerations of a toy car rolling under friction using Bitty Data Logger

The m:b 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 given 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 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 graph wiggles a small amount all over the place – again probably due to it not running quite smoothly. As we would expect, the blue 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 as very much. For comparison here is 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 – marked in blue. The axes are arranged so that x-axis lays 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.

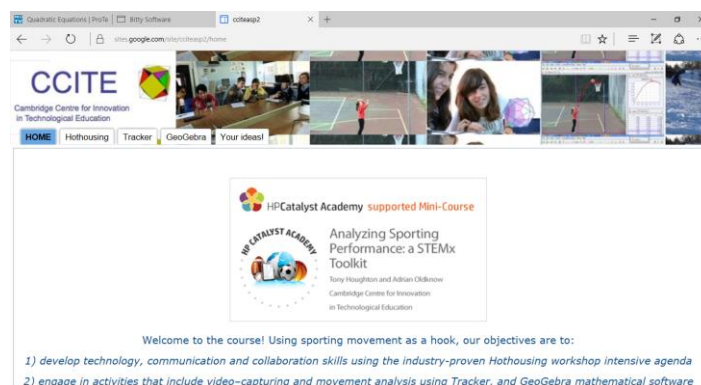
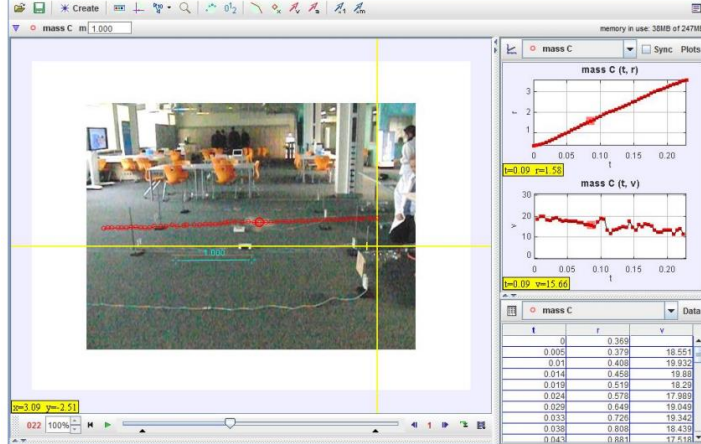
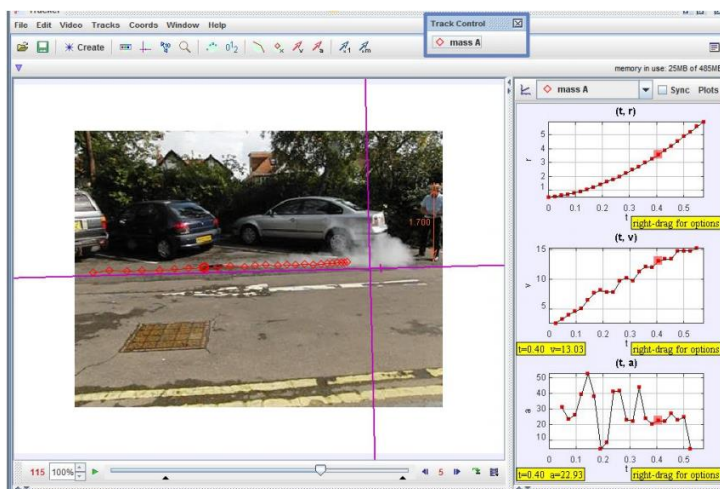
The second is from a [report I wrote](#) on the launch of the Manchester Bloodhound SSC Education Centre in Manchester where compressed air balloons are being run on strings to see the effect of shape on air resistance.

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 school 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.

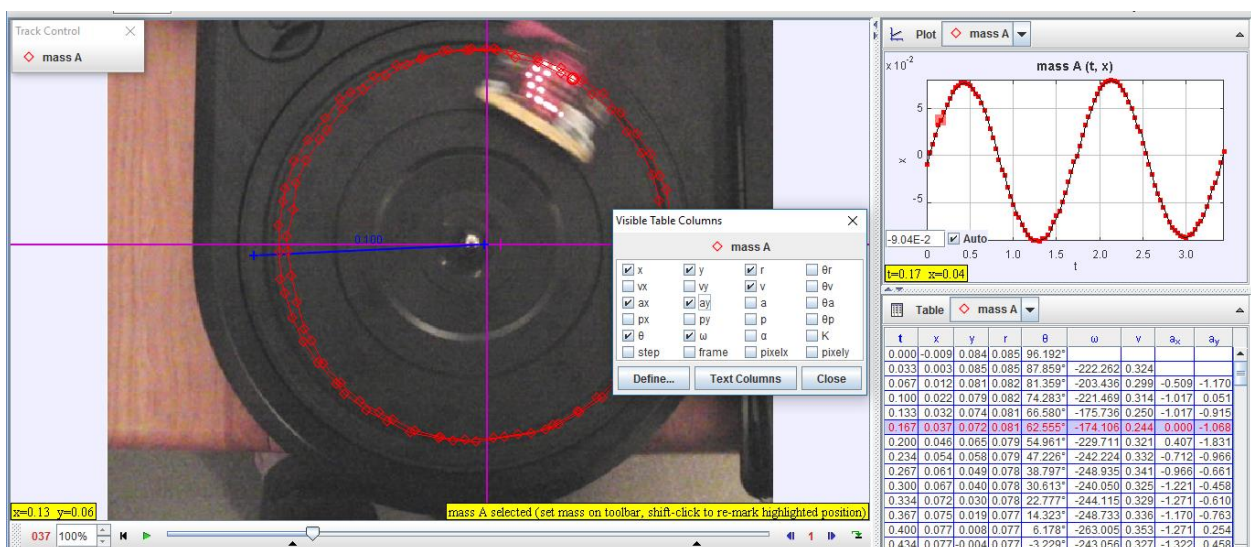
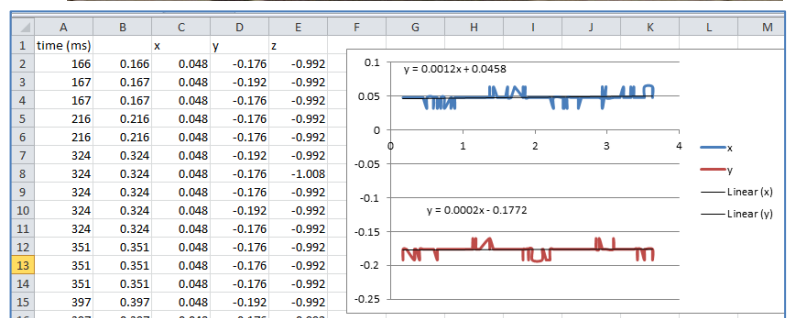
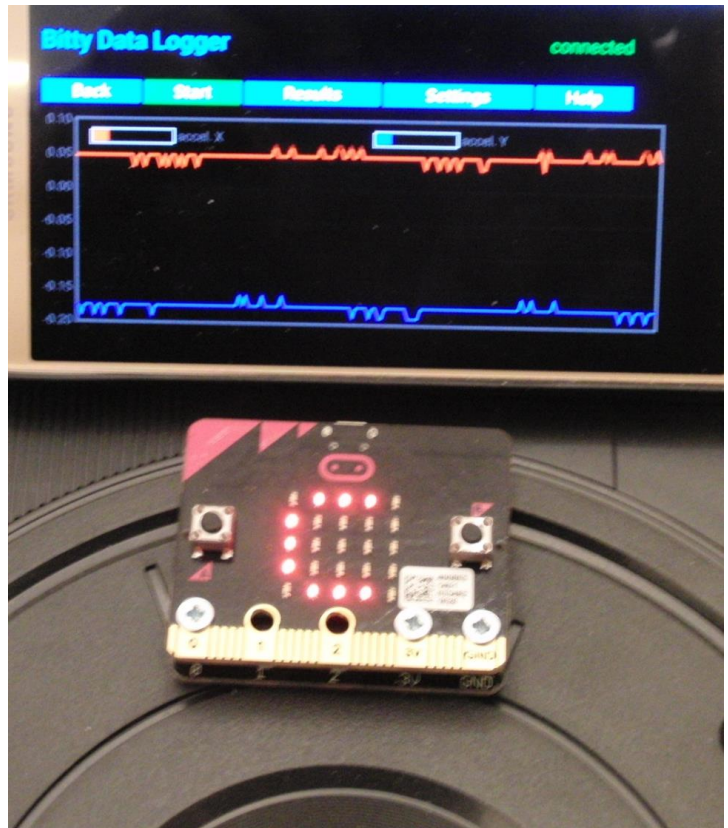
Experiment 7 Motion in a horizontal circle using Bitty Data Logger, Tracker and GeoGebra

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 m:b – and appear to be constants. The red acceleration is in the positive y-direction which is the instantaneous direction of travel of the m:b and looks to have a constant value of about +0.05g, 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 m: b 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 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.



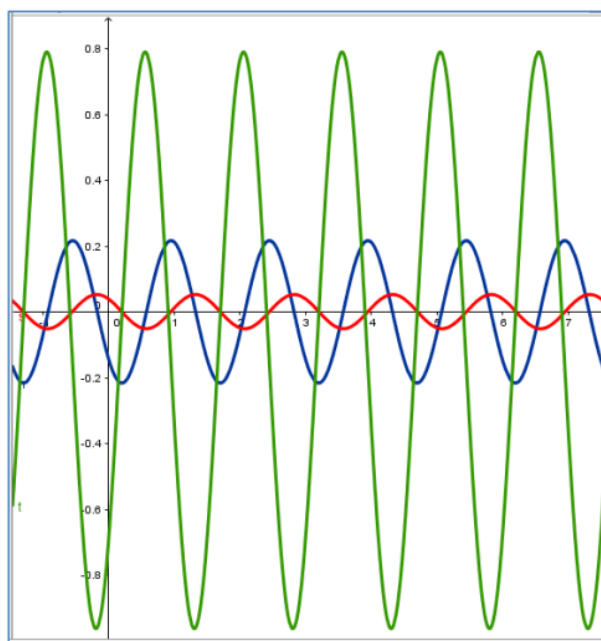
Here the axes are set in relationship to the static turntable, and so the x- and y- directions are not aligned with the directions of the m:b's accelerometer. Maybe it is possible to have a revolving frame of reference in Tracker – but I have yet to discover how to do that! As well as creating a table 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.

For a worked example see the spring-mass (SHM) system analysed [here](#)

have multiplied F values by $-g$ rather than $+g$, since the acceleration due to gravity is actually -9.81 ms^{-2} .

The final relationships between the models for micro:bit acceleration data in green, the Tracker displacement data in red and the Tracker estimates for velocity in blue are shown clearly in the GeoGebra graphs of Ellie's oscillations.

I hope this has given something of the flavour of how scientific and engineering experiments can be easily measured using micro:bit's on-board sensors, possibly enhanced with other devices. And how powerful free software tools such as Tracker and GeoGebra can be used to analyse and model the data mathematically. It is quite amazing that about £15 not only buys you an impressive array of sensors, but also the whole micro:bit with its ARM mBed processor and Bluetooth Low Energy transmission.



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 a 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 m:b in your pocket and running, jumping, bouncing etc. Or Velcro-ing m:b to apparatus like a sled, skate-board, bike etc.

Whatever you try, can you post some information on a site such as [this](#).