# Engduino Libraries: The LEDs

This is a quick guide to some of the things you can do with the LEDs. The idea is that you can use this information to experiment and put together something interesting that you created for yourselves rather than something we designed for you. Try stuff out – see what happens and see what you can make.

The LEDs can be set individually, or all together, and they can be set to different colours and brightnesses by mixing together different amounts of red, green and blue. There are 16 LEDs – and if you look on the board you’ll see their numbers (D0 – D15) written in white next to them.

## Header

At the top of your program you must have the following line – just like in the example

#include <EngduinoLEDs.h>

## setup()

In the setup() function, you must have the following line – again, just like the example

EngduinoLEDs.begin();

## Basic functions

There are a number of functions that you can call to set the LEDs. The LEDs are numbered from 0 to 15 inclusive.

### Set the colour of All LEDs:

You can set the colour of all the LEDs at the same time, just as you did in the example you were given. So, to set all the LEDs to the colour BLUE, at maximum brightness, we use:

EngduinoLEDs.setAll(BLUE);

The full set of colours available to you is RED, GREEN, BLUE, YELLOW, MAGENTA, CYAN, WHITE and, although it's not strictly a colour, OFF.

### Controlling brightness

Maximum brightness is often a bit bright, so we can turn it down by selecting a number that ranges from 0 (= off) to 15 (=brightest). To set the LEDs to a fairly dim YELLOW, we use:

EngduinoLEDs.setAll(YELLOW, 2);

### SETTING A SINGLE LED

If we want to set the colour of a single LED, we can do that too. For example to set LED0 (which is just by the USB connector) to be RED at maximum brightness we type:

EngduinoLEDs.setLED(0, RED);

But you can also control the brightness – to set LED2 to GREEN at brightness 3, then we use:

EngduinoLEDs.setLED(2, GREEN, 3);

Try this - you should see that LED2 is a lot less bright than LED1. Brightness can vary between 0 (off) and 15 (brightest) - experiment....

## More advanced functions

The primary colours of light are red (R), green (G), and blue (B). If we mix these in different ways, we can get all sorts of shades of light. The LED libraries allow users to set LEDs by the specific mix of brightnesses of R, G, and B. Remembering that the brightness lies between 0 and 15:

Set all LEDs with red brightness = 2, green = 2; and blue = 0

EngduinoLEDs.setAll(2, 2, 0);

Red and green mixed together in equal proportions make yellow – so this is yellow and, because we chose low numbers, it’s not very bright. But what if we want to make orange – which isn’t on our original list? Well, orange is like yellow, but with a bit more red in it, so let’s make the red a bit brighter and keep the green the same:

EngduinoLEDs.setAll(5, 2, 0);

Voila, orange.

We can do the same thing for individual LEDs. So, to set LED4 to a salmon pink (red = 9, green = 6, blue = 2), we might use:

EngduinoLEDs.setLED(4, 9, 6, 2);

There are even more ways of setting the LEDs for advanced programmers. You can find these on our website <http://www.engduino.org/doxygen/> but you need to know quite a lot to understand this.

## Ideas for projects with the LEDs

* Make a flashing rear cycle light.
* Make an electronic candle where the top flickers, or any type of Christmas tree decoration.
* Make the LEDs spell out the letters UCL one after another.
* Make a single LED seem to run round the board.
* Make the LEDs fade in and out
* …. It’s up to you.

# Engduino Libraries: ThE button

There is a single user button on the Engduino and we can work out when it is pressed (or released) to help us control what is happening in the program.

## Header

At the top of your program you must have the following line – often in addition to the one for LEDs and other bits of the Engduino.

#include <EngduinoButton.h>

## setup()

At the top of your program you must have the following line – often in addition to the one for LEDs and other bits of the Engduino.

EngduinoButton.begin();

## Basic functions

The first functions allow you to wait until a button is pressed or released....

EngduinoButton.waitUntilPressed();

or

EngduinoButton.waitUntilReleased();

If you have these in your program, then it will do nothing until the button is pressed (or released). For example the following code turns your Engduino into a torch – press the button to switch it on and off:

...

void loop() {

EngduinoButton.waitUntilPressed();

EngduinoLEDs.setAll(WHITE);

EngduinoButton.waitUntilPressed();

EngduinoLEDs.setAll(OFF);

}

The second function needs you to know a little bit more about programming. In a program, there is a way in which we can choose to do one thing or another – it’s called the ‘if’ statement. It’s easiest to see by example with our next function.

void loop() {

if (EngduinoButton.isPressed())  
    EngduinoLEDs.setAll(RED);

else

EngduinoLEDs.setAll(OFF);

}

This switches the LEDs to red only when the button is pressed. As soon as you release the button, then it sets the LEDs off.

## More advanced functions

The final function is useful is you want to write code that occasionally checks whether the button has been pushed but then goes on to do something else. While the code is doing the something else, it might be the case that the user pushes the button and lets it go and, because you haven’t checked to see if it’s pressed then, you might miss it. Well, the following function will tell you whether a button has been pushed or released since you last called that function - while your code is doing something else. In this way you need never miss a button press again.

void loop() {

if (EngduinoButton.wasPressed())  
    EngduinoLEDs.setAll(BLUE);

else

EngduinoLEDs.setAll(OFF);

delay(1000);

}

What you’ll see is that the LEDs turn blue for a second if you press the button, and then they go off again. If you press the button while the program is delaying, then the LEDs don’t come on immediately, but they will come on when it next tests to see if the button has been pressed. You’ll see this more clearly if you make the delay bigger.

Now try using EngduinoButton.isPressed()instead of EngduinoButton.wasPressed(). What you will find is that if you press the button while the code is delaying the Engduino will never realised that you pressed it at all.

## Ideas for projects with the BUTTON

* Traffic lights that change when you press the button
* Games that need user input…

# Engduino Libraries: The thermistor

The thermistor is a very simple device that changes its resistance (think Physics lessons and Ohms law) depending on how warm it is. This means we can use it to measure temperature.

## Header

At the top of your program you must have the following line – often in addition to the one for LEDs and other bits of the Engduino.

#include <EngduinoThermistor.h>

## setup()

At the top of your program you must have the following line – often in addition to the one for LEDs and other bits of the Engduino.

EngduinoThermistor.begin();

## Basic functions

There is only one function for the thermistor – to get the temperature, which comes back in degrees Celsius by default. But let’s introduce another idea too – serial output, so you can read a numerical the value on the screen.

void loop() {

float f;

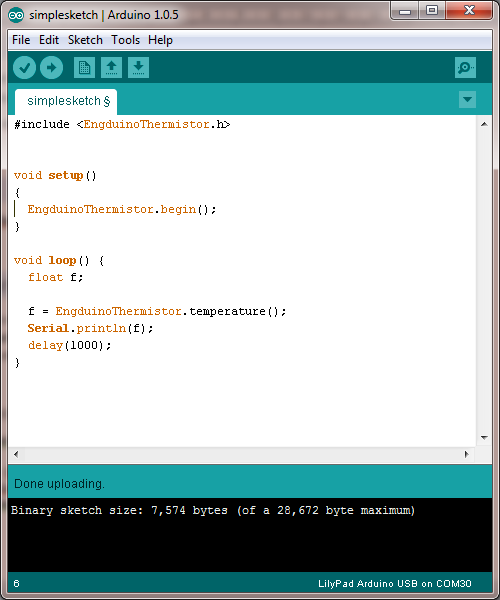
f = EngduinoThermistor.temperature();

Serial.println(f);

delay(1000);

}

This program reads the temperature and then prints a value out that you can read on the screen. It does this over the USB cable, so it isn’t going to do anything if that isn’t connected. Upload the program and then click on the button in the top right hand corner of the screen that looks like:



It should say ‘Serial Monitor’ next to it when you move the mouse pointer over it.

This will bring up another window and you will see that a temperature value appears every second. The thermistor is a tiny component at the top of the E with NTC written next to it in white (look on the Engduino diagram if you’re still unsure). Put your finger on it and, unless you’re programming this on a beach in Hawaii, you should see the temperature go up.

## More advanced functions

It is possible to specify the temperature scale you wish to use:

float fc = EngduinoThermistor.temperature(CELSIUS);  
float fk = EngduinoThermistor.temperature(KELVIN);  
float ff = EngduinoThermistor.temperature(FAHRENHEIT);

 It is also worth saying a bit more about the printing to the screen.

void loop() {

Serial.println("Hello World");

delay(1000);

}

Will print “Hello World” all on one line, delay a second and print it again on a new line. But what if we want to print several things on the same line? Try the following:

void loop() {

float f;

f = EngduinoThermistor.temperature();

Serial.print("Temperature is: ");

Serial.print(f);

Serial.println(" Celsius");

delay(1000);

}

This should print “Temperature is 25.71 Celsius” (or whatever your temperature is). Using Serial.print doesn’t move the output on to the next line – so we’ve used this for the first two parts of the line, and then we used Serial.println finally to end the line.

Note that you can only start the Serial Monitor after you have uploaded your program. So you might miss some of what it prints.

## Ideas for projects with the thermistor

* Make the LEDs change colour as temperature changes
* Find out how much the temperature changes in a room when you open a window – what do you think that tells you about heating energy wasted?

# Engduino libraries: The light sensor

The light sensor measures light flux – it returns a higher value when more light lands on the sensor.

## Header

#include <EngduinoLight.h>

## setup()

EngduinoLight.begin();

## Basic functions

There is only one function for the light sensor:

void loop() {

int v = EngduinoLight.lightLevel();

Serial.print(“Light value: “);

Serial.println(v);

}

The values returned range from 0 to 1023, inclusive – if there is more light, the number returned is higher. (For those interested in the technical details, the returned value is directly proportional to the illuminance in lux).

## Ideas for projects with the LIGHT SENSOR

* Make a bar graph out of the LEDs to show how much light is reaching the sensor. Can you think why this might not be 100% accurate? Hold a sheet of white paper flat on the top of the Engduino. Keep holding the part of the paper near the Engduino’s switch and gradually lift the end nearest the light sensor – what happens to your bar graph and why?
* See if you can make a night-light. It should come on when it becomes reasonably dark and go off in the daylight.
* Make a communication system.
  + Flash the LEDs on one Engduino and receive those flashes on another – print the value of the light level to the screen so you can see when the LEDs are on and when they are off… the numbers should be different.
  + Work out a way of using this to send the elements of morse code – print out whether you think the thing that was sent was a dot, dash or space. Do you always get it right?
  + How about sending just two values rather than the three in morse – this is binary code and we call the values 1s and 0s. Can you figure out a way of turning 1s and 0s into flashes of light and turning the flashes you receive back into 1s and 0s? If so, congratulations, you built a modern digital communication system.

# Engduino Libraries: The accelerometer

The accelerometer on the Engduino measures accelerations, as its name suggests. Again, the interface is straightforward. As it is set up, the accelerometer can measure accelerations of ±2g.

## Header

At the top of your program you must have the following line – often in addition to the one for LEDs and other bits of the Engduino.

#include <EngduinoAccelerometer.h>  
#include <Wire.h>

**NOTE the extra include.**

## setup()

At the top of your program you must have the following line – often in addition to the one for LEDs and other bits of the Engduino.

EngduinoAccelerometer.begin();

## Basic functions

Because the accelerometer reads three dimensions of acceleration, it needs to return three values and these are stored in an array. Ask your teacher about arrays, but for now, just use the following code. The values of acceleration are given in ‘g’ (where 1g is the acceleration due to gravity).

void loop() {

float accelerations[3];

EngduinoAccelerometer.xyz(accelerations);

float x = accelerations[0];

float y = accelerations[1];

float z = accelerations[2];

Serial.print("Acceleration: x = ");

Serial.print(x);

Serial.print("g y = ");

Serial.print(y);

Serial.print("g z = ");

Serial.print(z);

Serial.println("g");

delay(1000);

}

Upload this and open the serial monitor. Hold the Engduino flat with the LEDs uppermost. You should see that the X and Y accelerations are near to zero, and the Z acceleration is close to -1. This tells you that gravity is acting downwards relative to the Engduino. Flip the board over so the LEDs are nearest the floor. The Z value should become positive at +1g. Work out which axis represents X and which represents Y (hint: turn the Engduino on its side and on its end).

If you shake the Engduino vigorously enough, you’ll see that the accelerations go up to ±2g. That’s because this accelerometer is set to measure a maximum of ±2g: the true number might be higher than that.

## More advanced functions

There aren’t any for the accelerometer. But the 3D acceleration is what is known as a vector quantity (talk to your maths teacher) – it has a magnitude (size, length) and a direction. To get the overall magnitude, irrespective of orientation, we need to do a bit of maths. If we only had X and Y axes (i.e. we had a 2D accelerometer) the situation would be:

Overall acceleration

y

x

We can calculate the magnitude (length) of the resultant from Pythagoras’ rule:

BUT, we have a 3D accelerometer – and the same principle holds where we have X, Y and Z axes. So the overall magnitude of the resultant acceleration vector is:

Add the following to the code above:

float a = sqrt(x\*x + y\*y + z\*z);

Serial.print("Acceleration: = ");

Serial.println(a);

Now, if you keep the accelerometer still (put it on the desk), this will give an acceleration of about 1g, irrespective of what orientation you have the Engduino in – and it will be different to that as you move it about. Actually, the value will vary slightly even if you keep it still, because the accelerometer isn’t a perfect measuring device. Dealing with this is a process called calibration and is something we have to do when we need to know a quantity accurately.

## Ideas for projects with the ACCELEROMETER

* Make the LEDs light up when the magnitude of the acceleration is greater than 1g (or maybe a little bit more, say 1.1g).
* Change the colour of the LEDs depending on the direction of the acceleration
* Turn the Engduino into a spirit level

# Engduino Libraries: The infrared

The infrared transceiver allows two (or more) Engduinos to talk to each other over a distance of upto about 50-100cm. This is the most complex of the libraries in terms of basic functionality, but it is easy enough to use once you are a competent programmer.

## Header

#include <EngduinoIR.h>

## setup()

EngduinoIR.begin();

## Basic functions

To make use of the IR, you must understand the concept of arrays. Take some time to revise this if you need to before trying the exercises in here. Because the IR is about communication, you will need two Engduinos to do anything useful.

In terms of basic functions, you can send a single character (byte) at a time. So, for example, to send the character ‘c’ we do the following.

EngduinoIR.send('c');

And, to receive, you should execute the following code on the destination:

uint8\_t buf[IRBUFSZ];   // Declare a buffer in which to receive

// any message. It should be this size.

int len = EngduinoIR.recv(buf);   // Block until there is something to

// receive and then receive it.

// len contains the length of the

// received message

Write code for a sender, which sends a character every second. On a different Engduino, write the code for a receiver, which repeatedly executes a receive and prints the result.

## More advanced functions

Rather than sending a single byte at a time, you can send upto 12 bytes:

char buf[12]; // Declare a buffer of characters.

// [Could be declared as uint8\_t]  
  
*... put some values into buf*  
EngduinoIR.send(buf, 12);        // Send the buffer - you need to provide

// the length of the buffer as well as

// the contents

If you don't want to wait forever for an incoming message to arrive, you can set a timeout. When the function call returns, either (a) a message has been received or (b) the call has timed out. In the first case, the length of the message is returned, in the second, a negative number is returned:

uint8\_t buf[IRBUFSZ];   // Declare a buffer in which to receive

// any message. It should be this size.

int len = EngduinoIR.recv(buf, 500);   // Block until either we have a

// message to receive, or 500

// microseconds has elapsed  
                                  // len contains either the length of the

// received message or a negative number  
if (len >= 0)

    .... a message was received  
else

    .... the call timed out - there is no message

If you set the timeout to zero, the recv function will block forever - as it did when we omitted the timeout value.

## Ideas for projects with the IR

* Make an Engduino send the word “Hello” when a button is pressed, and print out any message it receives. Try this between two Engduinos and vary the distance between them. You will probably notice that at some point, the message isn’t received correctly.
* To deal with this, we need to know when a message has been received correctly. There are several ways of doing this: for example, you could use a parity bit or a cyclic redundancy check (CRC) algorithm. Search online for these terms and see whether you can implement them for the Engduino
* Write a simple communications protocol. Each Engduino should periodically send a message that tells other Engduinos that it is there. It should also listen for other Engduinos sending this message. When two Engduinos meet, one of them should flash the LEDs red, the other should flash them blue. It doesn’t matter which does what, so long as one is red and the other blue. You shouldn’t use the button at all – this should happen automatically when they are in range of each other.
* Note for Engduino v2+: because of the way in which the IR is wired in v2+, which was changed from v1 to allow additional interfacing capabilities, the IR communications will not be able to decode the signal sent by a TV remote. It may still be able to distinguish between some buttons, but that is beyond the scope of this document.

# Engduino Libraries: The Magnetometer

**Note that the magnetometer only exists on Enduino v2+**

The magnetometer measures magnetic field strength in each of three axes. It can be used to create a digital compass or to explore magnetic fields, such as those generated by a permanent magnet or those around a coil through which a current is running.

The interpretation of magnetic field strength is not easy. The driver for the magnetometer returns raw values. Each magnetometer is different and will require individual calibration to account for offsets in the raw numbers and distortions to the magnetic field introduced by what are known as hard and soft iron interference.

## Header

#include <EngduinoMagnetometer.h>

#include <Wire.h>

**NOTE the extra include.**

## setup()

EngduinoMagnetometer.begin();

## Basic functions

The interface to the magnetometer looks very much like the interface to the accelerometer:

void loop() {

float magneticField[3];

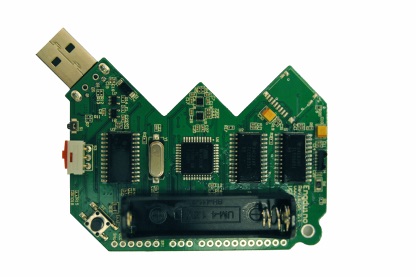
// Read magnetic field strength. The values range from -20000

// to +20000 counts

//

EngduinoMagnetometer.xyz(magneticField);

float x = magneticField[0];

float y = magneticField[1];

float z = magneticField[2];

Serial.print(x);

y

x

z into the page

Serial.print(", ");

Serial.print(y);

Serial.print(", ");

Serial.println(z);

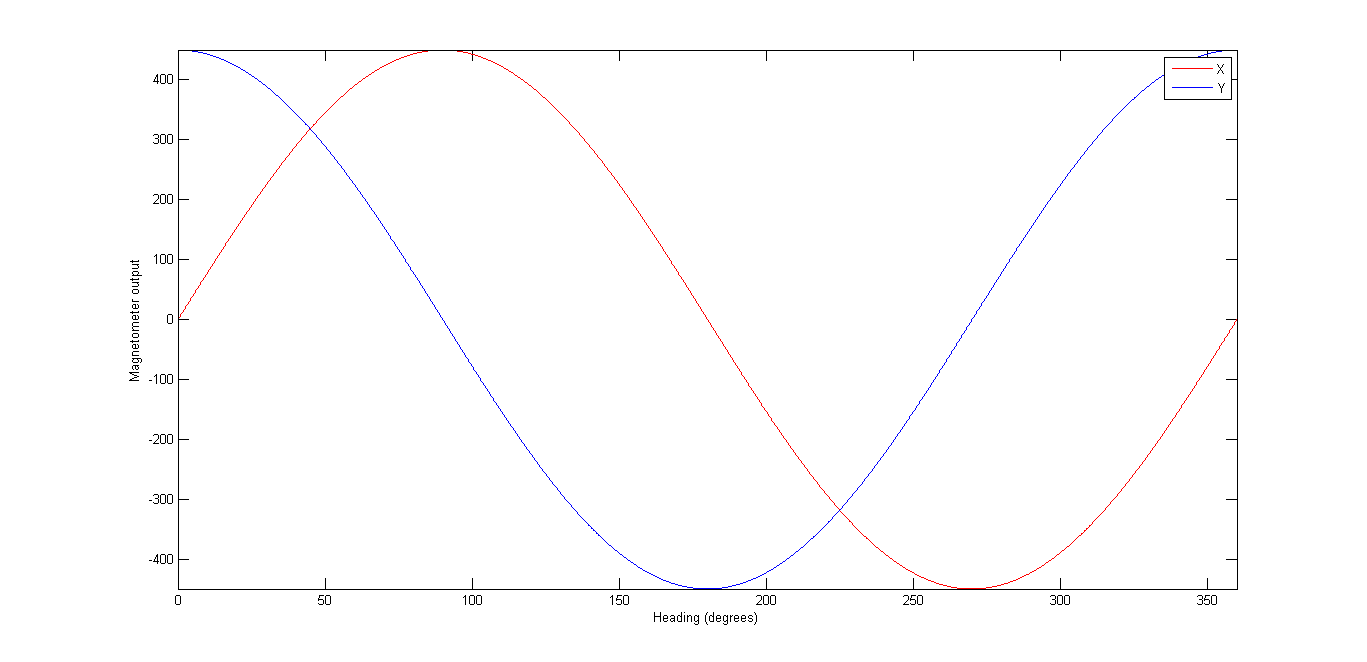
delay(200);

}

This reads the magnetic field in three dimensions and outputs comma separated values, which seems easy enough. However, interpreting the outputs of the magnetometer is quite a complicated matter.

Unfortunately, the values returned from the call are raw, uncalibrated, values. In practice this means that there might be different numbers returned from the x y and z axes to represent the same magnetic field strength. Moreover, the exact numbers and the relationship between axes will differ from Engduino to Engduino. So, to use an Engduino to measure magnetic field strength, we first need to calibrate the sensor[[1]](#footnote-1).

If the sensor was fully calibrated, we held the Engduino perfectly flat and rotated it, keeping the Engduino flat at all times, then we should see only the x and y values change as they point towards and away from north. In fact, they should change in this way:



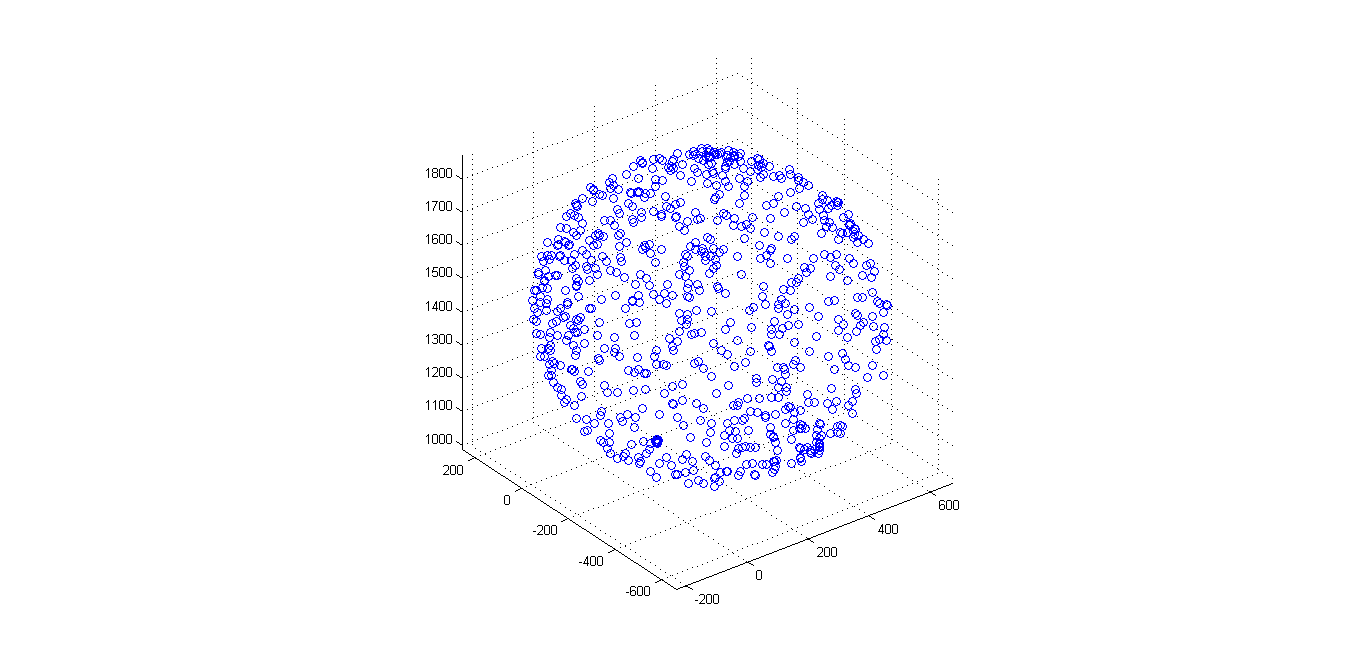
Let’s assume that we can go into an environment (e.g. on a playing field) in which we can be reasonably sure that the only magnetic field is that due to the earth (buildings have steel in them and that can have its own magnetic field). If this is the case, we know that what we are measuring is a constant – it will not change over the course of a set of measurements.

Try the following.

1. Face North (your phone might have a compass app if you don’t know where this is)
2. Plug the Engduino into a laptop on a USB extension lead.
3. Hold the Engduino as flat as you can.
4. Rotate the Engduino while keeping it flat. You will see the x and y values change and the z will stay about the same.
5. Find the minimum and maximum values for each of the x and y axes separately.

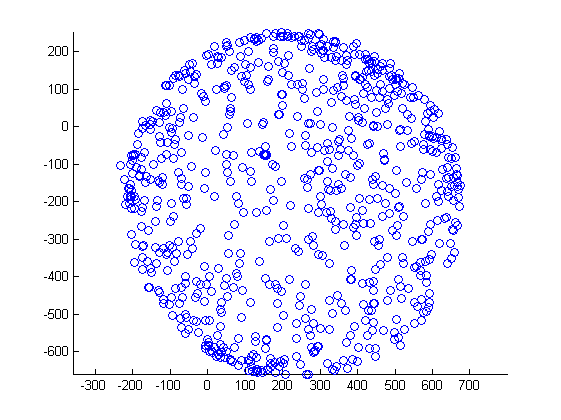
You will be very lucky to see exactly the readings we have above – the x and y values may not be about -450 to +450, but may instead be, say, +232 to 1138 and -660 to 253 respectively. So the zero is likely to be in the wrong place on each axis and the maximum minus the minimum might be slightly different.

If you move the Engduino about through all the angles you can manage – not even trying to keep it flat, your max/min values might change a little for x and y, and you will find a range for the z axis. If you plot the x,y,z values in a 3D scatter plot (unfortunately, Excel doesn’t have one of these, so I used Matlab) you might get something that looks like the following:



The values lie on (or close to the surface of) a sphere – which is good news, but the sphere isn’t centred on zero, which is not. Note that, in some cases of magnetic interference, it is possible for the sphere to distort into an ellipsoid (something that looks more like an egg), which makes life just a little bit more problematic.

Rather than worry about 3D at this point, let’s see what happens if we just plot the x,y values (for which you can use Excel), the picture we see is just as if we were looking down on the top of the sphere along the z axis:



This isn’t quite a circle but it’s fairly close (there is some distortion or the sensor is slightly wonky). What we would like to do is to move the centre of the circle to (0, 0), so that if you hold the Engduino flat, we can work out which direction you’re facing in. One simple way of estimating the centre is just to take the maximum value in each dimension, subtract the corresponding minimum value and divide by 2. We then subtract the coordinates of that centre point from any value we see – and the result should be a circle centred (roughly) at the origin.

Given we have this, we can calculate direction using simple trigonometry. Take a look at the heading graph above, and see if you can work out what the function should be. Hint: arctangent is in the right area, but it only gives a value between -90° and +90°. In many programs, you can use a function atan2 to give you a value between -180° and +180°.

## More advanced functions

The above approach is very simplistic. It will give the wrong result if you tip the Engduino out of horizontal and it takes no account of the case in which the points lie on an ellipsoid rather than a sphere.

Doing this right involves two things: (i) calibrating the sensor properly against an ellipsoid[[2]](#footnote-2); (ii) compensating for tilt in the Engduino by using the accelerometer to tell us what orientation the magnetometer is being held in[[3]](#footnote-3). This in turn involves some linear algebra – the details of which can be found in the references below.

## Ideas for projects with the MAGNETOMETER

* Calibrate your magnetometer. Find out whether the calibration stays (about) the same over a number of days and whether it is the same inside or outside a building or near something that has a lot of steel in it (e.g. a lift).
* Make the Engduino into a compass that illuminates the LED closest to where north lies. The Engduino is expected to be held flat to work. What happens when you tilt the Engduino?
* Now make the compass work even if the Engduino is held tilted at an angle.

Engduino Libraries: The SDCARD

**Note that the SD card only exists on Enduino v2+**

The Engduino can be used to log sensor readings and other text to an SD card in the SD card slot on the board. To write data to the SD card, this sequence of events must be followed:

The interpretation of magnetic field strength is not easy. The driver for the magnetometer returns raw values. Each magnetometer is different and will require individual calibration to account for offsets in the raw numbers and distortions to the magnetic field introduced by what are known as hard and soft iron interference.

## Header

#include <EngduinoSDCard.h>

## setup()

EngduinoSDCard.begin();

## Basic functions

The …

void loop() {

float magneticField[3];

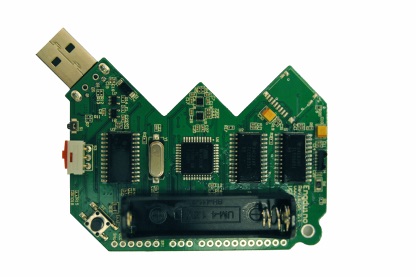
// Read magnetic field strength. The values range from -20000

// to +20000 counts

//

EngduinoMagnetometer.xyz(magneticField);

float x = magneticField[0];

float y = magneticField[1];

float z = magneticField[2];

Serial.print(x);

y

x

z into the page

Serial.print(", ");

Serial.print(y);

Serial.print(", ");

Serial.println(z);

delay(200);

}

# Control flow: IF statements

When programming the Engduino, it is useful to be able to change what the code is doing depending on the value of a variable, or on whether some event has happened. We saw this in the code for the button.

void loop() {

if (EngduinoButton.wasPressed())  
    EngduinoLEDs.setAll(BLUE);

else

EngduinoLEDs.setAll(OFF);

delay(1000);

}

Strictly, the if statement takes a Boolean value as its argument. A Boolean value is one that can only be either false or true. If it is true, then the statement following the if is executed, if the value is false, then the statement following the else is executed.

What if we don’t want an else part – we only want to do something extra if the condition is true. Well, we can just omit it:

void loop() {

if (EngduinoButton.wasPressed())  
    EngduinoLEDs.setAll(BLUE);

delay(1000);

EngduinoLEDs.setAll(OFF);

}

This code achieves the same thing as the previous code: if the button has been pressed, it switches the LEDs on for a second.

What if we want to execute more than a single statement if some condition is true. Well, we can do that too, by grouping statements inside curly braces. This isn’t like Python – indentation is nice for keeping your code neat, but it isn’t used to group statements together. For that you need braces.

void loop() {

if (EngduinoButton.wasPressed()) {  
    EngduinoLEDs.setAll(BLUE);

Serial.println("Button Pressed");

}

delay(1000);

EngduinoLEDs.setAll(OFF);}

# Control flow: FOR statements

If you have done the exercise with the traffic lights, you will have found that you need to write a lot of code to set half the LEDs to one colour and the other half to another – we have to set them individually, one at a time. Fortunately, there’s a shortcut called the for statement which is used to execute some code a fixed number of times.

If we want to make the first 8 LEDs flash RED the code would be:

void loop() {

int i;

for (i = 0; i < 8; i++)

EngduinoLEDs.setLED(i, RED);

delay(1000);

EngduinoLEDs.setAll(OFF);

delay(1000);

}

There are several parts to this. The first is the statement:

int i;

This declares a variable i. You can think of a variable as a box to put things into with a special name. Our box is called i, but you can call it what you like so long as it’s unique and you’re consistent.

The loop is then:

for (i = 0; i < 8; i++)

EngduinoLEDs.setLED(i, RED);

What this means is:

1. Set the value of i (the number in the box called i) to be 0
2. check to see if the value of i is less than 8. If it is not, then go to step f). If i is less than 8, then
3. set LED with the number we have in the box labelled i to be RED
4. add one to i
5. go back to step b) and carry on.
6. Carry on with the rest of the program - execute whatever is next in the code after this for statement. In this case that’s delay(1000)

Just as for if statements, it is possible to group more than one statement together:

for (i = 0; i < 8; i++) {

EngduinoLEDs.setLED(i, RED);

Serial.print(i);

Serial.println(" switched on");

}

# Control flow: WHILE statements

If we want to do something while a condition is true, then we can use a while statement. So let’s write a program that turns on more and more LEDs while the button is pressed.

void loop() {

int i = 0;

while (EngduinoButton.isPressed()) {

EngduinoLEDs.setLED(i, RED);

i = i + 1;

delay(200);

};

delay(1000);

EngduinoLEDs.setAll(OFF);

}

Like the for statement, we declare a variable, i, but we give it an initial value of 0 this time:

int i = 0;

Now comes the loop – with several statements grouped together inside the curly braces:

while (EngduinoButton.isPressed()) {

EngduinoLEDs.setLED(i, RED);

i = i + 1;

delay(200);

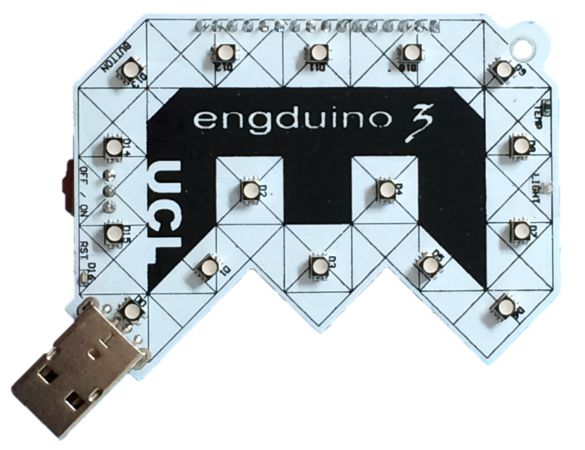
};

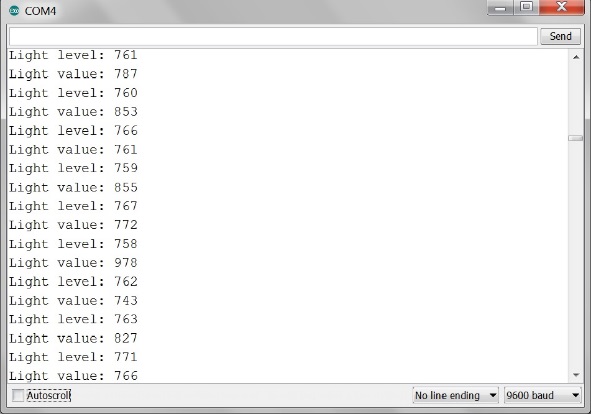
What this means is:

1. Check to see if the condition is true – in other words, if the button is pressed. If it is no longer pressed, go to step f).
2. set LED number i to be RED
3. add one to the value of i
4. delay a little (0.2s)
5. and go back to the step a)
6. As soon as we let the button go (the condition is false) we carry on with the rest of the program. This waits for a second (delay(1000);) and then sets all the LEDs so that they are off again (EngduinoLEDs.setAll(OFF);).

# Data visualisation: python

It’s useful to view sensor readings using the Serial Monitor by printing out the values using Serial.print() and Serial.println()functions. The Engduino sends the information to the serial interface which is actually the USB connector.





Everything that you print is output through the serial port. In this case the serial port is the USB connector.

Sometimes it is easier to understand the meaning of data if we can plot values on a graph. There are several ways of doing this with the Engduino and in this worksheet we are going to explain how to do it using Python. Some special libraries are needed, you should install:

* [SciPy](http://www.scipy.org/) which is a set of software for mathematics, science, and engineering.
* [drawnow](https://pypi.python.org/pypi/drawnow/) which provides a feature for redrawing a graph. You might need to install [Python pip](https://pypi.python.org/pypi/pip) first.

At the top of your Python program you must import the following Python modules:

**import serial**

**import numpy**

**import matplotlib.pyplot as plt**

**from drawnow import \***

The Python serialmodule allows you to read from the serial port; everything which is printed to the Engduino’s serial port can be used by your Python program. You have to create a serial object and to tell the Python program what the characteristics of the serial port are, like this:

**arduinoData = serial.Serial('com4', 9600)**

This says that the Engduino is plugged in to COM port number 4 and that we will communicate with it at a speed of 9600 bits per second. On a computer running Linux the port might be labelled /dev/ACM0 or similar. The line speed is sometimes described as the baud rate.

**Note**: this speed must match the speed specified in the setup loop in your Engduino sketch. Your Engduino sketch should have this line in the setup() function:

Serial.begin(9600);

Now let’s look at a simple Python 2.7 example that will plot the light values from the light sensor.

import serial

import numpy

import matplotlib.pyplot as plt

from drawnow import \*

arduinoData = serial.Serial("/dev/ttyACM0", 9600)

lightReadings **=** **[]**

count **=** 0

plt**.**ion**()** # use interactive mode to plot data

**def** doPlot**():**

plt**.**ylim**(**0**,**1023**)** # Set limits for y axis.

plt**.**title**(**‘Light level in classroom 4B at 9 am')

plt**.**grid**(True)** # Draw a grid

plt**.**ylabel**(**'Light '**)** # Set label for y axis

plt**.**plot**(**lightReadings**,** 'ro-'**,** label**=**'Light readings'**)**

plt**.**legend**(**loc**=**'upper left'**)** # Position of legend

**while** **True:**

**while** **(**arduinoData**.**inWaiting**()==** 0**):** # Wait until there is data

**pass**

arduinoString **=** arduinoData**.**readline**()** # Read from the serial port

temp **=** int**(** arduinoString **)** # Convert to an integer

lightReadings**.**append**(**temp**)** # Add to the array

drawnow**(**doPlot**)** # Update the graph

count **=** count **+** 1 # Delete the first element

**if(**count **>** 50**):** # so that the array remains

tempF**.**pop**(**0**)** # no larger than 50 items

Let’s take a closer look at the ‘while True:’ loop. What this means is:

1. Check to see if any data has been sent to the serial port. If no data has been received, go to the start of the loop again.
2. Read a line of data into a string.
3. Convert the string into a number (an integer) and store it in a variable called temp.
4. Store the temp value at the end of the array called lightReadings.
5. Re-draw the graph.
6. If the array lightReadings is more than 50 elements long, remove the first element.
7. Go back to step a).

1. Actually, this is something that is common to all sensors – for the thermistor, the light sensor or the accelerometer, we ideally need to know how the value returned from the sensor compares to the actual truth. [↑](#footnote-ref-1)
2. See “Calibrating an eCompass in the Presence of Hard and Soft-Iron Interference” AN4246 by Talat Ozyagcilar [↑](#footnote-ref-2)
3. See “Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors” AN4248 by Talat Ozyagcilar [↑](#footnote-ref-3)