

Making a line follower with the BBC micro:bit

Minos April 2017 –

by David Hannaford –

Email - david@davidhannaford.com



TOPICS:

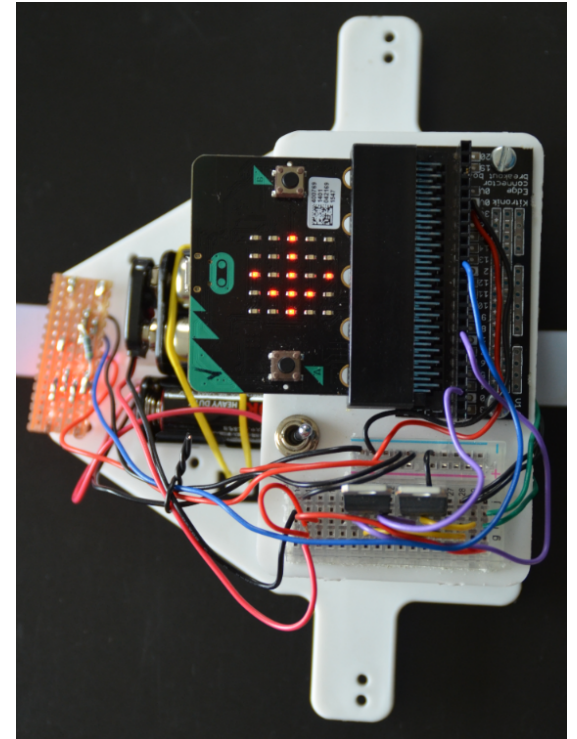
What is the BBC micro:bit

What is on it?

How can you program it?

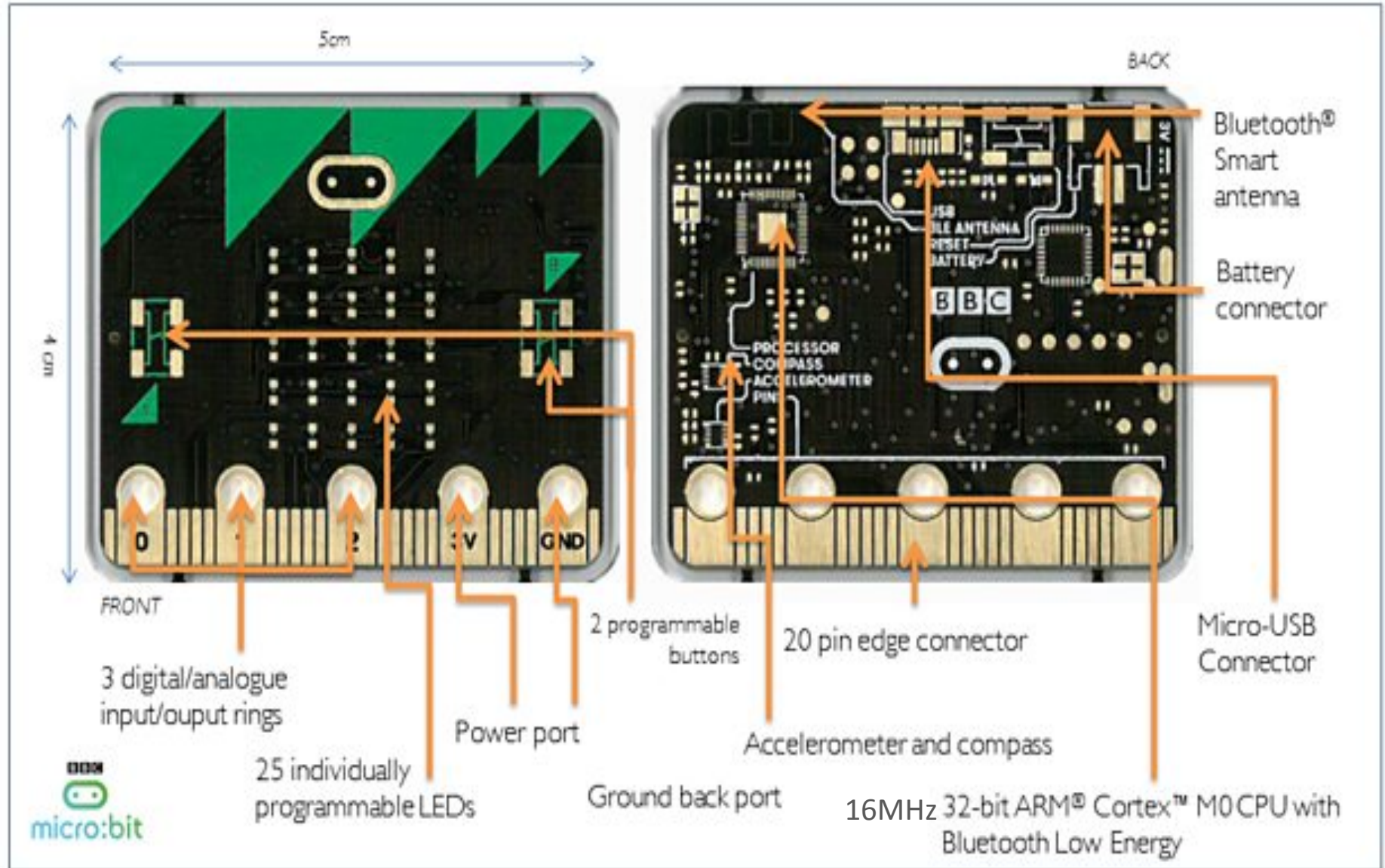
Making a simple line follower

Making a remote control robot



What is the micro:bit ?

A 5 x 4 cm microcontroller on a board



What is on it?

- Processor 32 bit ARM Cortex M0 running at 16Mhz
- Inputs - 2 buttons plus analogue and digital input pins
- Outputs – PWM and digital output pins
25 LEDs in 5 x 5 matrix
- Sensors – accelerometer and compass
- Communication – Bluetooth low energy
 - Simple radio (between micro:bits)
 - USB link to PC
- Running from a 3 volt battery or USB lead to PC

How to program it

- Online environments available to code it on a PC using:
 - JavaScript Blocks editor either using a Blocks visual programming language or directly in JavaScript or a combination of both
 - Python editor
 - At <http://microbit.org/code/>
 - Programs are compiled online and saved to a file on the PC then moved using the USB cable to the Micro:bit which is seen as an external drive
- You can also use an Android or IOD app to do the coding and then download the code via Bluetooth
- Previous editors are still available if you still want to use them

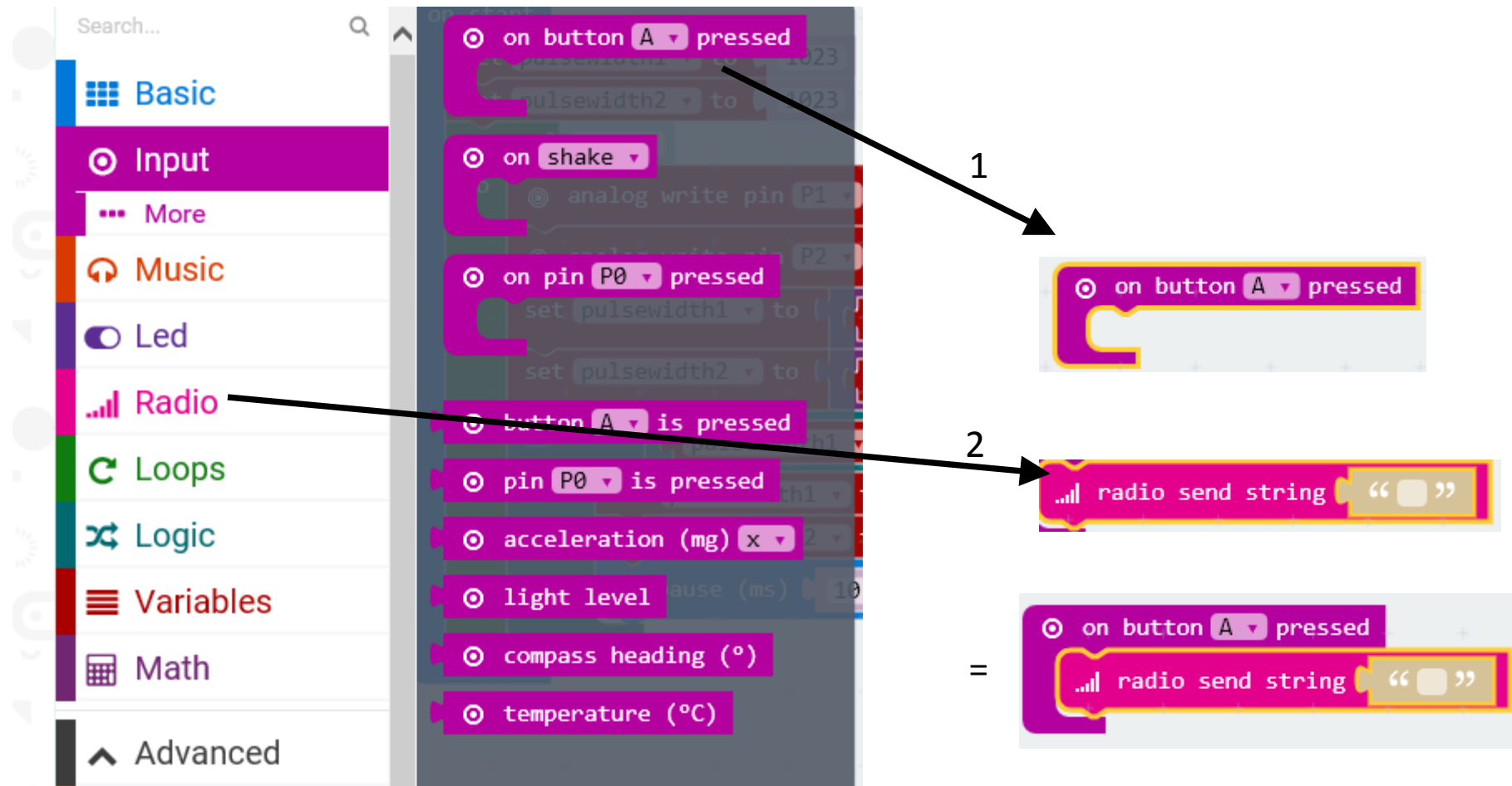
The blocks editor

The screenshot shows the Microsoft MakeCode Blocks editor for the micro:bit. The interface is divided into several sections:

- Simulator:** A visual representation of the micro:bit board is shown on the left side of the editor.
- Program component menus:** A central menu lists various code blocks categorized by function: Basic, Input, Music, Led, Radio, Loops, Logic, Variables, Math, Advanced, Text, Game, Images, Pins, Serial, Control, and Add Package.
- Code area:** The right side of the editor displays a script for an "on start" event. The script includes:
 - Two "set" blocks for "pulsewidth1" and "pulsewidth2" both set to 1023.
 - A "while" loop set to "true" with a "do" block containing:
 - "analog write pin P1" and "analog write pin P2" blocks, both using the "pulsewidth" variables.
 - "set" blocks for "pulsewidth1" and "pulsewidth2" each decreased by 1.
 - An "if" block checking if "pulsewidth1" is less than 50. If true, it resets both "pulsewidth1" and "pulsewidth2" to 1023.
 - A "pause (ms)" block set to 10.
- Compile & run buttons:** At the bottom left, there are buttons for "Download" (a download icon) and "Run" (a play icon).

Red text labels are overlaid on the image to identify these key components: "Simulator", "Program component menus", "Code area", and "Compile & run buttons".

Drag program components from the menus to build the program



Program now built

Basic

Input

Music

Led

Radio

Loops

Logic

Variables

Math

Advanced

Text

Game

Images

Pins

Serial

Control

Add Package

on start

set pulsewidth1 to 1023

set pulsewidth2 to 1023

while true

do

analog write pin P1 to pulsewidth1

analog write pin P2 to pulsewidth2

set pulsewidth1 to pulsewidth1 - 1

set pulsewidth2 to pulsewidth2 - 1

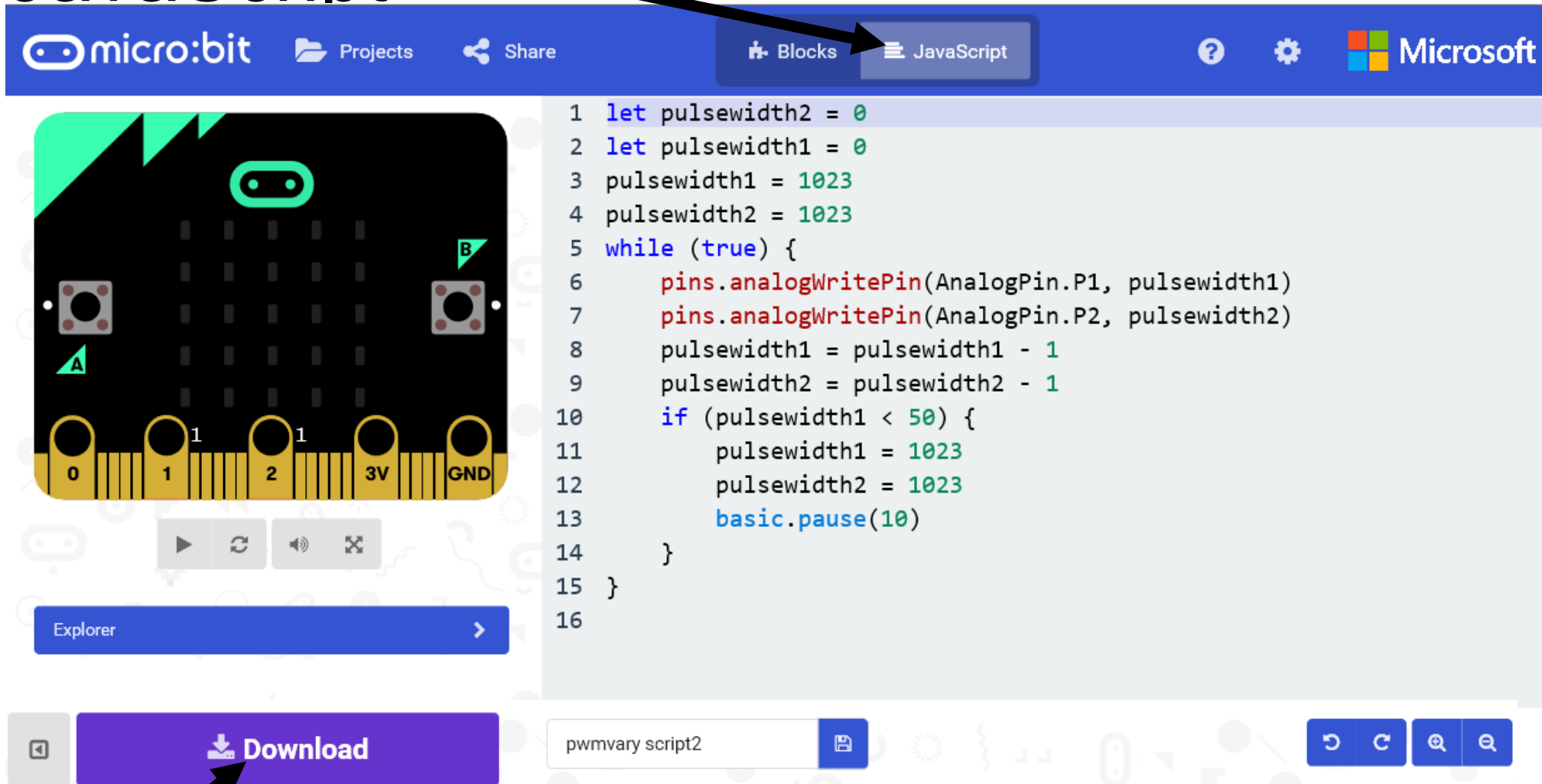
if pulsewidth1 < 50

then set pulsewidth1 to 1023

set pulsewidth2 to 1023

pause (ms) 10

Press this button and it becomes JavaScript



The screenshot shows the micro:bit online editor interface. At the top, there's a blue header bar with the micro:bit logo, 'Projects', 'Share', 'Blocks', and 'JavaScript' tabs. An arrow points from the text 'Press this button and it becomes JavaScript' to the 'JavaScript' tab. Below the header, on the left, is a visual representation of the micro:bit board with pins labeled 0, 1, 2, 3V, and GND. To the right of the board is a code editor with the following JavaScript code:

```
1 let pulsewidth2 = 0
2 let pulsewidth1 = 0
3 pulsewidth1 = 1023
4 pulsewidth2 = 1023
5 while (true) {
6     pins.analogWritePin(AnalogPin.P1, pulsewidth1)
7     pins.analogWritePin(AnalogPin.P2, pulsewidth2)
8     pulsewidth1 = pulsewidth1 - 1
9     pulsewidth2 = pulsewidth2 - 1
10    if (pulsewidth1 < 50) {
11        pulsewidth1 = 1023
12        pulsewidth2 = 1023
13        basic.pause(10)
14    }
15 }
16
```

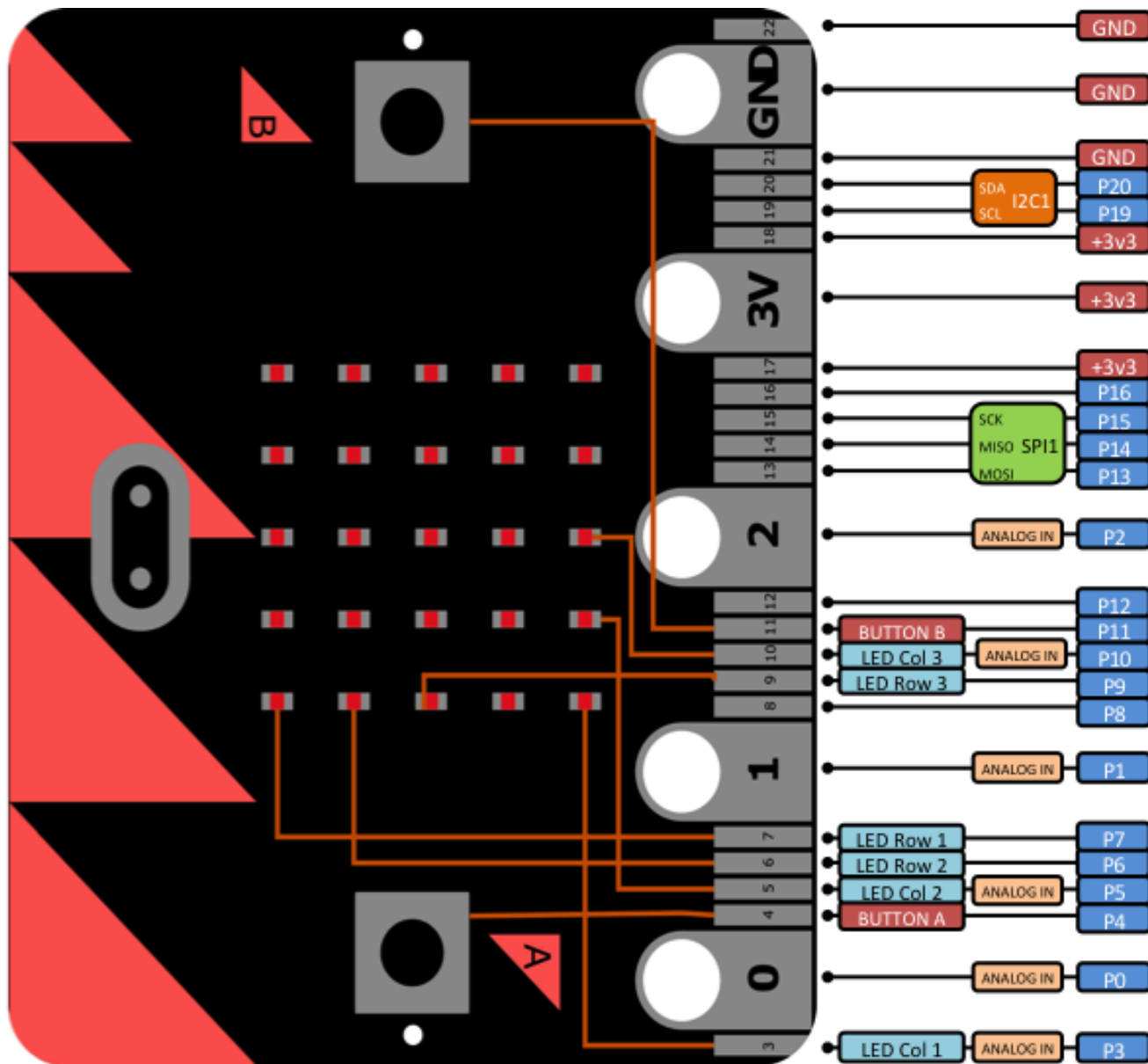
At the bottom left, there's a blue 'Download' button with a download icon. An arrow points from the text 'Click here to compile the program and save it to a file on your PC' to this button. To the right of the 'Download' button is a text input field containing 'pwmvary script2' and a save icon. On the far right, there are icons for undo, redo, search, and zoom.

Click here to compile the program and save it to a file on your PC
Then move the file to your micro:bit to load the program.

Using micro:bit for a line follower robot

- We need
 - Two PWM controllable outputs to drive the motors We suggest using pins 0 and 1
 - One or two analogue inputs on pins 1 and 12 for inputs from phototransistor line sensors
 - A & B Press buttons for program selection or starting use pins 4 and 11
 - LED matrix uses pins 3,5,6,7,9,10
- So whilst the micro:bit has all the functions we need we need to be careful on the use of the pins as they are not reconfigurable and many pins have duplicate functions
- Diagram on next slide shows what all the pins can be used for and what ones have duplicate functions

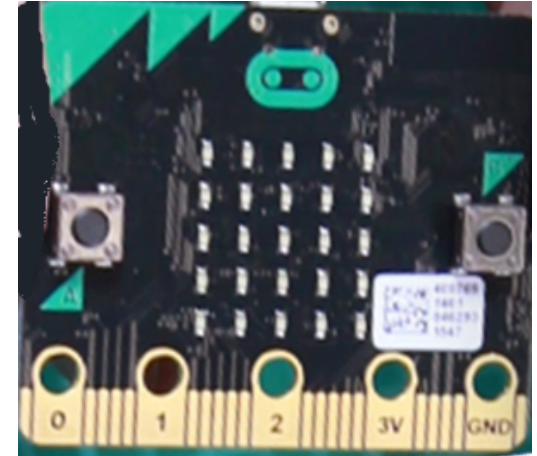
BBC micro:bit pin connections



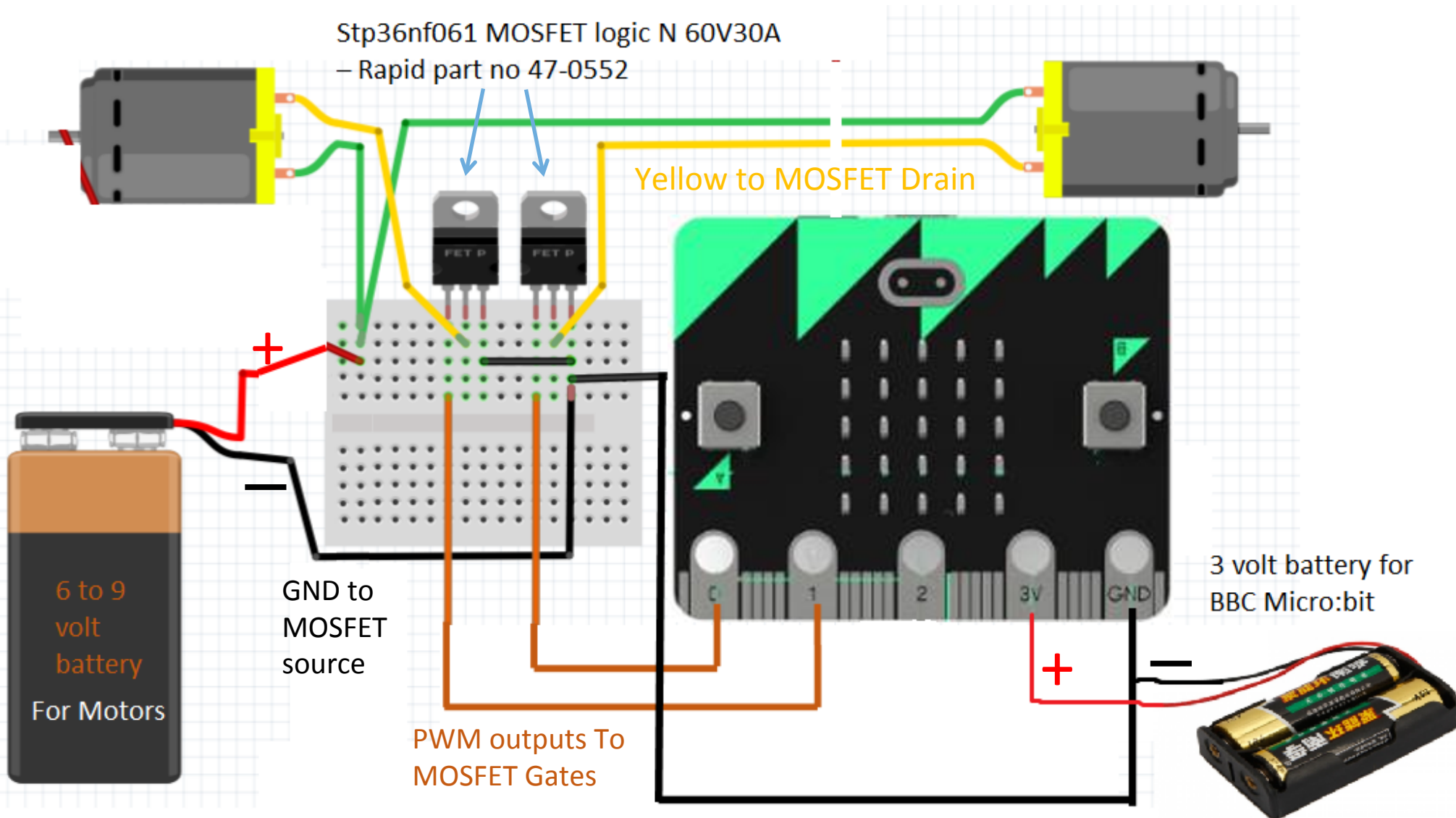
- 6 General purpose input output (GPIO) on Pins 6,7,8,9,12,16
- 2 press buttons on Pins 4,11
- 6 Analogue in on Pins 0,1,2,3,5,10 (but 3 of the same pins also used for PWM outputs)
- 3 reconfigurable PWM outputs on pins 0,1,2 Use with command analog write pulsewidth to pin
- I2C on Pins 19,20
- SPI on Pins 13,14,15

To access all the pins

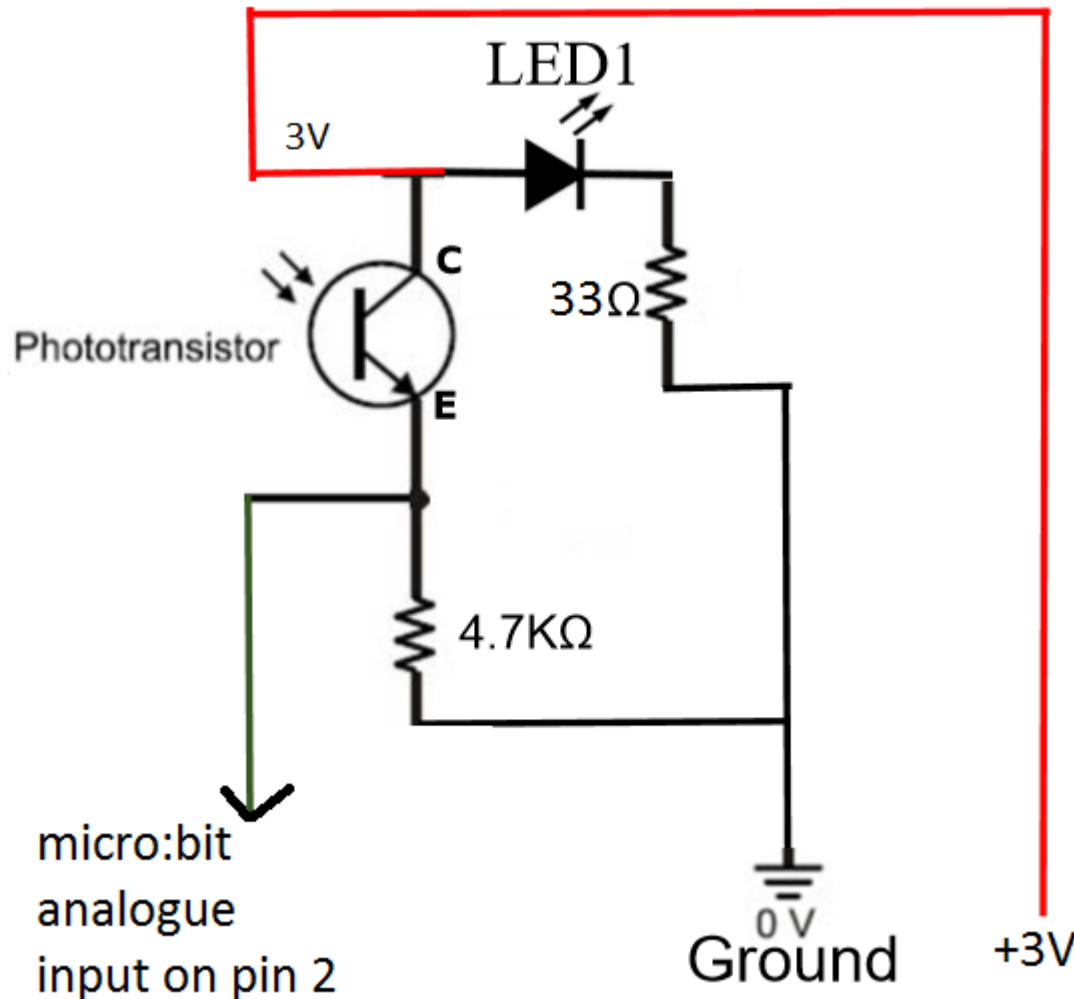
- Pins 0,1,2, 3v and Gnd can be connected by a wire or an M4 or smaller bolt through the holes or with small crock clips
- For a mouse with 2 PWM controlled motors and one phototransistor input for line detection you can get away with just using the easily accessed pins 0,1 and 2
- To use any of the pins other than 0,1, or 2 you really need a connector.
- Kitronic do this one at £3 as a kit or £5 pre built.



A circuit for driving the motors just using 2 MOSFETs



A one photo transistor and LED circuit for detecting the line



Details of recommended LED and phototransistor components on next slide

When powered by 3 volts, this circuit should give about 1.25 volts when over a black surface and about 2.5 volts over a white one, equating to analogue input values of around 450 (black) to 900 (white) with a midpoint of around 675.

Veroboard 1 LED sensor design for line following

Viewed from component side of board. Solder parts to tracks on back of board

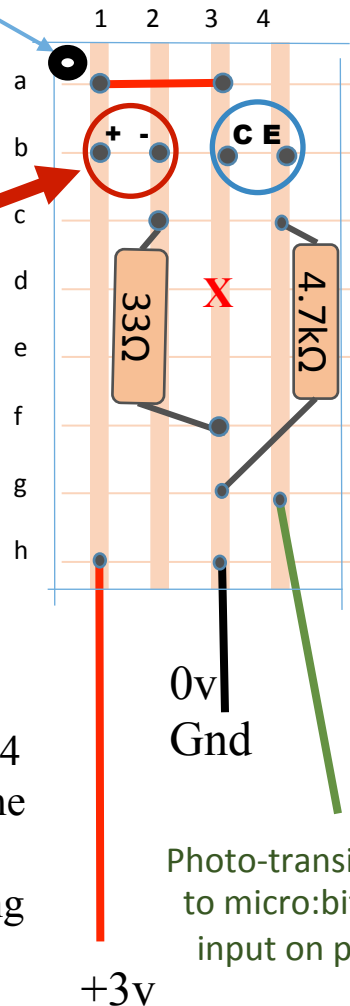
Mark the board here so you keep it correctly oriented while putting in the components

LED long lead is +

LED

e.g. Rapid part no 55-0066.
3mm red LED
3000mcd, max
50mA

Veroboard size needed is 8 x 4 useable holes. So cut along the 9 x 5 row of holes
Make sure the tracks are going vertical when you cut it out



IMPORTANT: Make sure you put the LEDs and phototransistors in the right way round. See below.

Photo-transistor

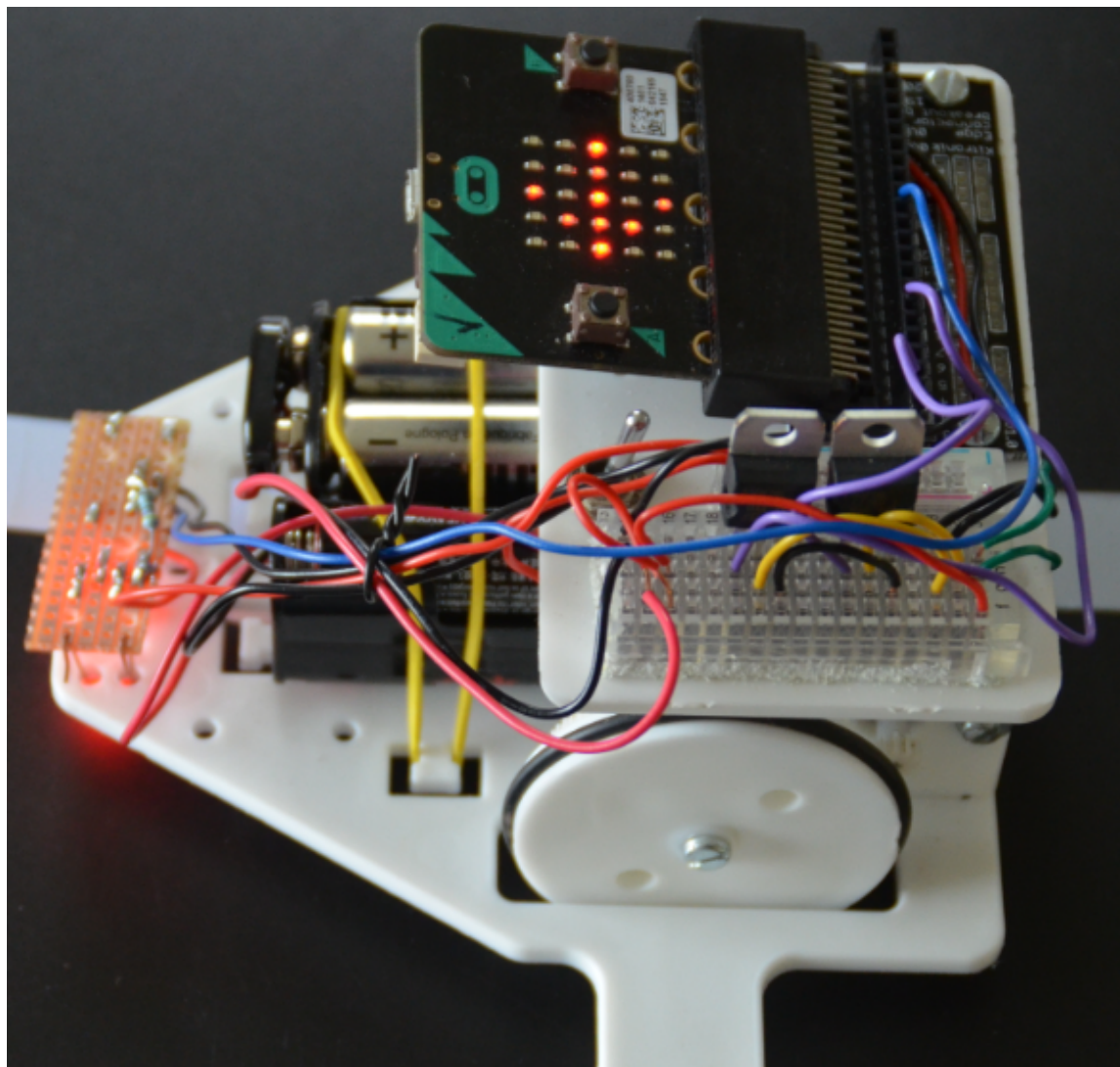
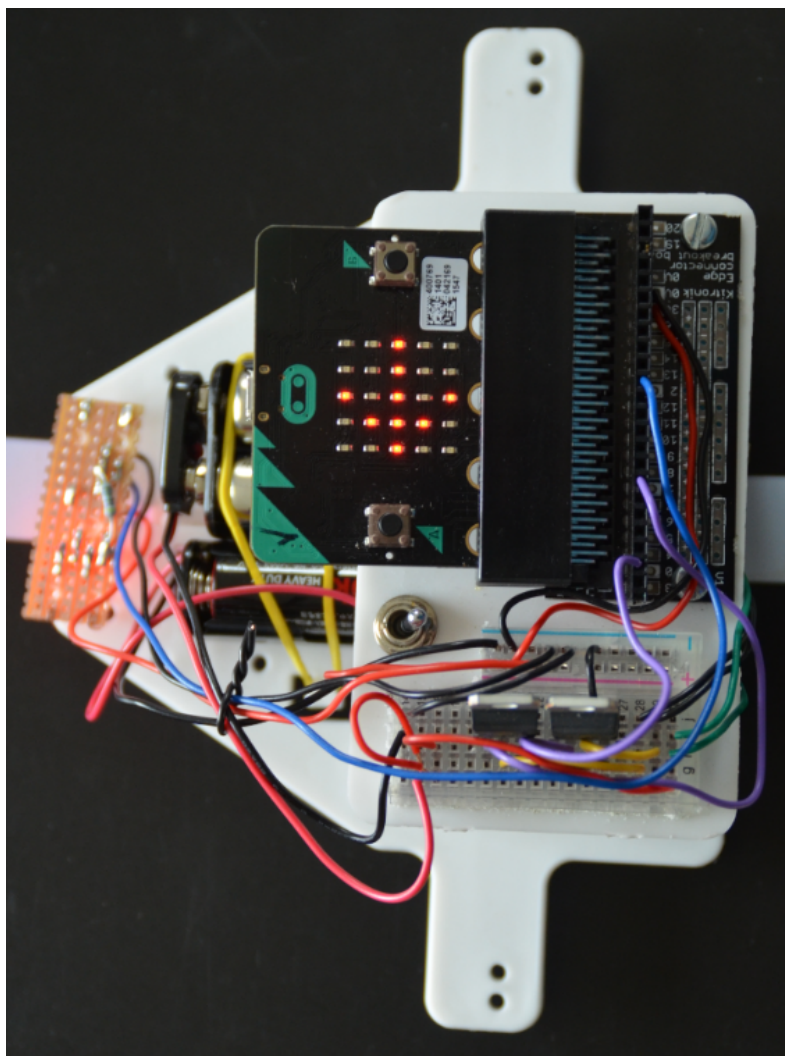
e.g. Farnell part no 104-5380
Vishay BPW85 photo-transistor

Short lead is collector (**C**)

Check datasheet for which lead is Collector (**C**) and Emitter (**E**) if using a different part

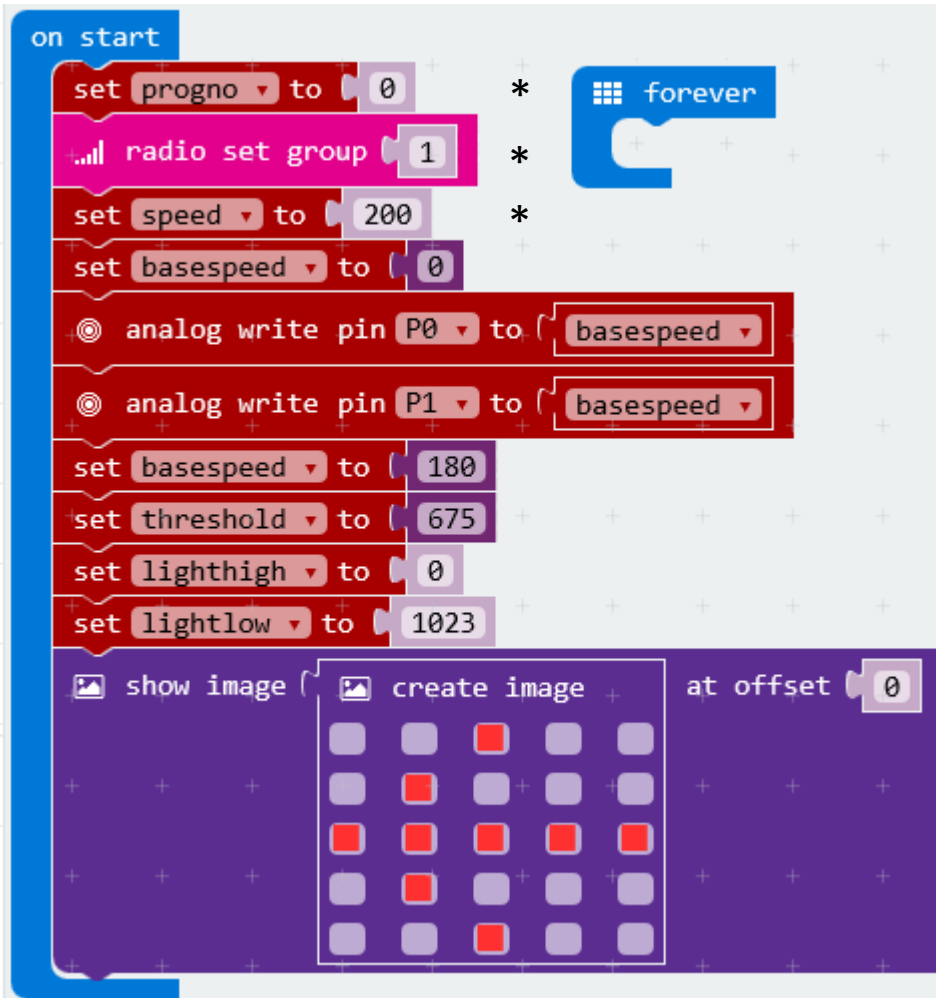
1. Cut track at point marked with **X** (d3)
2. Solder in the LEDs in row b, 5mm above board
3. Solder in photo transistor in row b at same height
4. Add red link wire (a1 to a3)
6. Add 33Ω LED series resistor (c2 to f3)
7. Add 4.7kΩ photo transistor resistor (c4 to g3)
8. Add connecting wires for 3V (h1) and Ground (h3)
- 9 Add green wire to go to micro processor (g4)
10. Fix board to chassis with hot glue gun

What it looks like assembled on a BCU chassis



Line following code

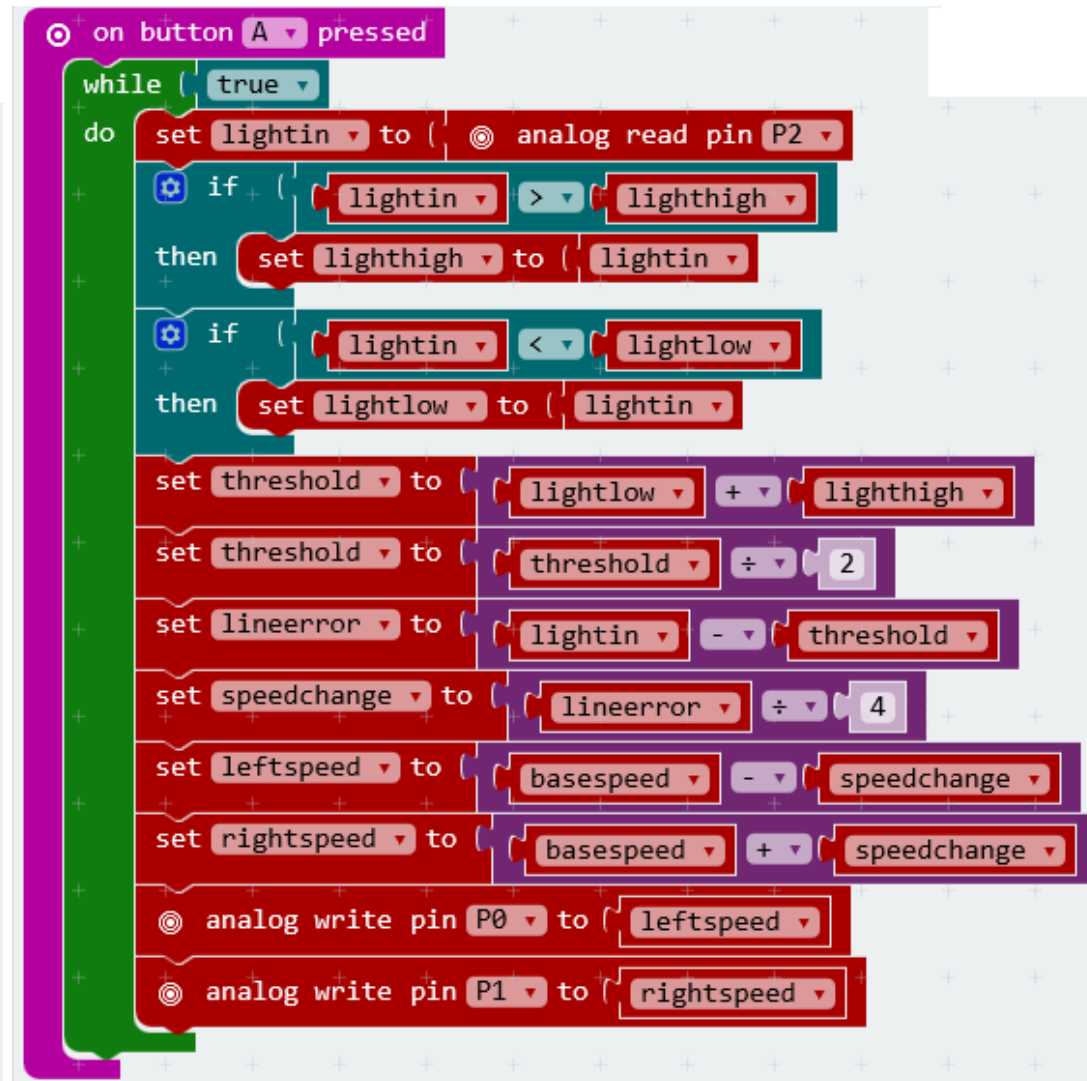
Setup code – The 3 lines marked * not required for line following



```
on start
  set progno to 0 *
  radio set group 1 *
  set speed to 200 *
  set basespeed to 0
  analog write pin P0 to basespeed
  analog write pin P1 to basespeed
  set basespeed to 180
  set threshold to 675
  set lighthigh to 0
  set lightlow to 1023
  show image (create image) at offset 0
```

The code is written in a Scratch-like block-based language. It starts with an 'on start' block. Inside, there are several 'set' blocks: 'set progno to 0', 'set speed to 200', 'set basespeed to 0', 'set basespeed to 180', 'set threshold to 675', 'set lighthigh to 0', and 'set lightlow to 1023'. There are also two 'analog write' blocks: 'analog write pin P0 to basespeed' and 'analog write pin P1 to basespeed'. At the bottom, there is a 'show image' block with a 'create image' sub-block and 'at offset 0'. Three lines are marked with an asterisk (*): 'set progno to 0', 'radio set group 1', and 'set speed to 200'. A blue 'forever' loop block is also present, but it is empty.

Line following main loop. First half of code automatically adjusts the black/white midpoint threshold



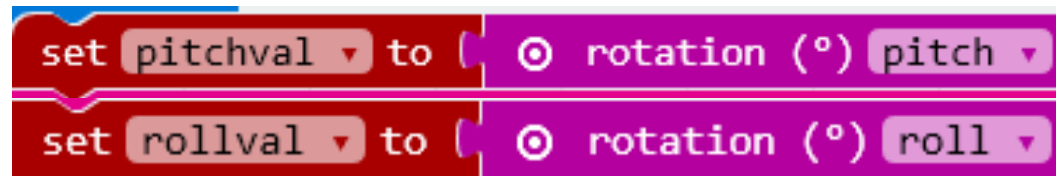
```
on button A pressed
  while (true)
    do
      set lightin to (analog read pin P2)
      if (lightin > lighthigh)
        then set lighthigh to lightin
      if (lightin < lightlow)
        then set lightlow to lightin
      set threshold to (lightlow + lighthigh)
      set threshold to threshold ÷ 2
      set lineerror to (lightin - threshold)
      set speedchange to (lineerror ÷ 4)
      set leftspeed to (basespeed - speedchange)
      set rightspeed to (basespeed + speedchange)
      analog write pin P0 to leftspeed
      analog write pin P1 to rightspeed
```

The code is written in a Scratch-like block-based language. It starts with an 'on button A pressed' block. Inside, there is a 'while (true)' loop. The loop contains several blocks: 'set lightin to (analog read pin P2)', 'if (lightin > lighthigh) then set lighthigh to lightin', 'if (lightin < lightlow) then set lightlow to lightin', 'set threshold to (lightlow + lighthigh)', 'set threshold to threshold ÷ 2', 'set lineerror to (lightin - threshold)', 'set speedchange to (lineerror ÷ 4)', 'set leftspeed to (basespeed - speedchange)', 'set rightspeed to (basespeed + speedchange)', 'analog write pin P0 to leftspeed', and 'analog write pin P1 to rightspeed'.

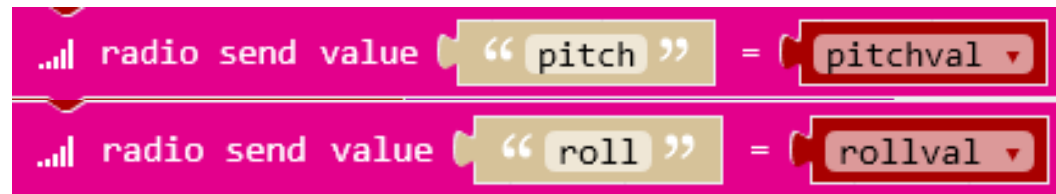
Second half of code proportionally adjusts the left and right motor speeds to keep sensor light value at midpoint

Other control methods for the robot

- Using the Accelerometer



- Pitch and roll give values -180 to +180
- Using the Radio



Using these 2 features you can use a second micro:bit as a remote control for speed and direction

- Using the compass (gives values 0 to 360)



Microbit Radio

The processor chip contains a built-in 2.4GHz radio module. This radio can be configured in a number of ways, and is primarily designed to run the Bluetooth Low Energy (BLE) protocol. However, it can also be placed into a much simpler mode of operation based that allows simple, direct micro:bit to micro:bit communication.

Capability	Brief Description
Frequency	1MHz narrowband, typically 2.407 GHz. Use configurable in the 2.400 GHz - 2.499 GHz band.
Channel Rate	1Mbps.
Maximum Transfer Unit	Typically 32 bytes, but reconfigurable in code up to 1024 bytes.
Addressing	All devices share the same address to guarantee user privacy.
Encryption	None. User level encryption (or BLE) should be considered if secure channels are required.
Meshing	None. (yet!)
Error Detection	16 bit hardware CRC.
Transmisson Power	Eight user configurable settings from 0 (-30dbm) to 7 (+4dbm).
Transmisson Range	Approx. 20m at 0dbm.

Code to transmit pitch and roll values to another micro:bit
and display roll value on LEDs in with "5" as level

Read and send out pitch & roll values by radio
And display normalised roll value

```
on radio received name value
  if (name == "pitch")
    then
      set speed to (value x 2)
      set speed to (speed + 200)
      show leds
  if (name == "roll")
    then
      set dispval to (value ÷ 18)
      set dispval to (dispval + 5)
      show number dispval
  analog write pin P0 to (speed + value)
  analog write pin P1 to (speed - value)
```

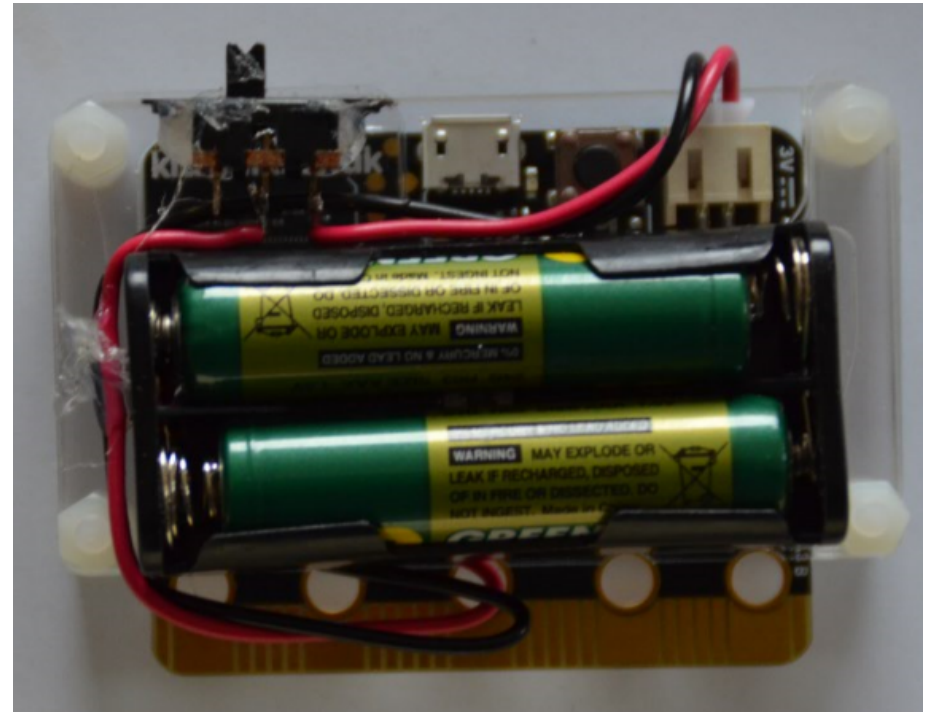
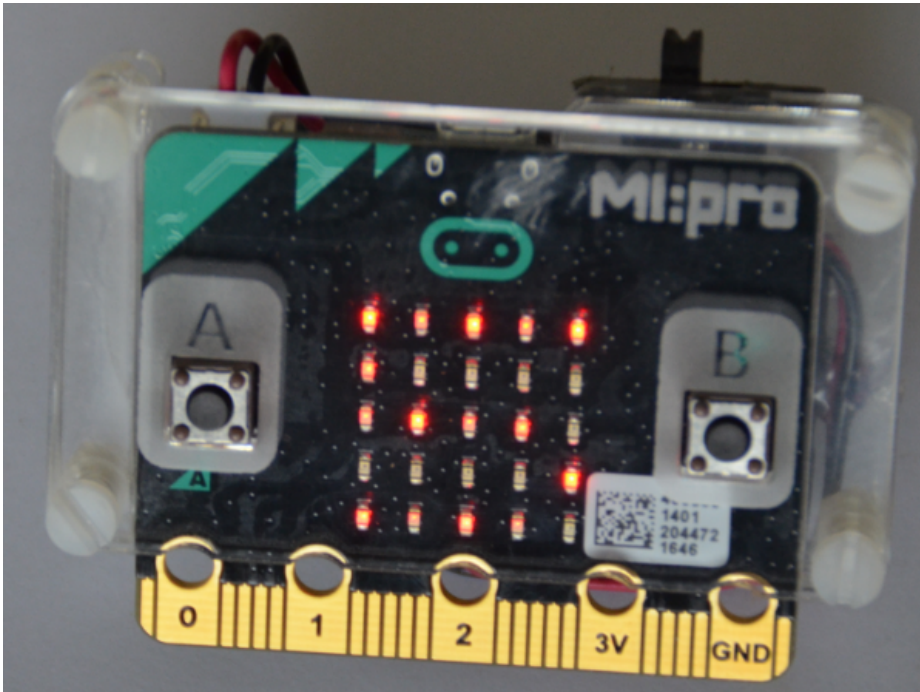
The code is written in a block-based language. It starts with an 'on radio received' block with 'name' and 'value' dropdowns. An 'if' block checks if 'name' is 'pitch'. If true, it sets 'speed' to 'value' multiplied by 2, then adds 200 to 'speed', and shows the LED matrix. Another 'if' block checks if 'name' is 'roll'. If true, it sets 'dispval' to 'value' divided by 18, then adds 5 to 'dispval', and shows the number 'dispval'. Finally, two 'analog write pin' blocks are shown: one for P0 with 'speed + value' and one for P1 with 'speed - value'.

Receive pitch & roll values by radio and
display normalised roll value
Use values to adjust speed and turning

```
on start
  radio set group 1
  set progno to 0
on radio received name value
  if (name == "pitch")
    then
      set speed to (value x 2)
      set speed to (speed + 200)
      show leds
  if (name == "roll")
    then
      set dispval to (value ÷ 18)
      set dispval to (dispval + 5)
      show number dispval
  analog write pin P0 to (speed + value)
  analog write pin P1 to (speed - value)
```

This code is similar to the first one but includes an 'on start' block. The 'on start' block contains a 'radio set group' block set to 1 and a 'set progno' block set to 0. The rest of the code is identical to the first code block, handling 'pitch' and 'roll' radio messages and adjusting speed and turning.

Remote control using MI:pro case



Forward and back adjusts speed
Tilt side to side controls direction

Any questions?