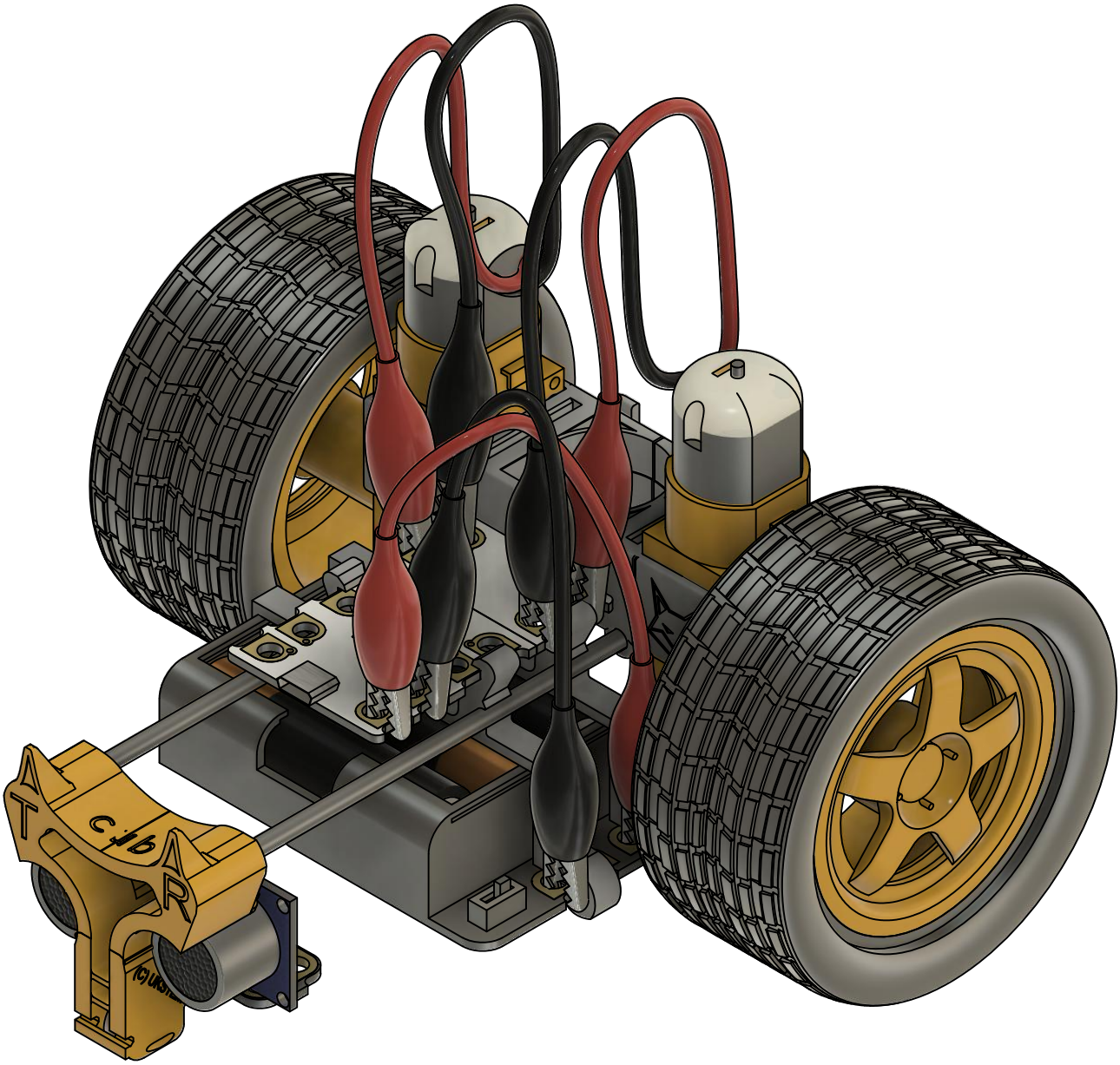
Redfern
electronics

Getting Started:



The Crumble Robotics Add-On Pack

Overview

This Crumble Robotics set has been designed to help you get more out of your Crumble Class pack. The additional components provided enable you to create 16 Crumble Universal Buggies (CUBs) as well as providing 16 Servo Motors to further explore the world of robotics.

This booklet will to guide you through the process of getting your CUB working, as well as using the various components that attach to your robot. This booklet does assume some prior knowledge/understanding. If this is your very first time using the Crumble, we would recommend taking a look at the various tutorials available on our website or, better still, watching our YouTube videos which take an in-depth look at how to get started with the Crumble Controller.

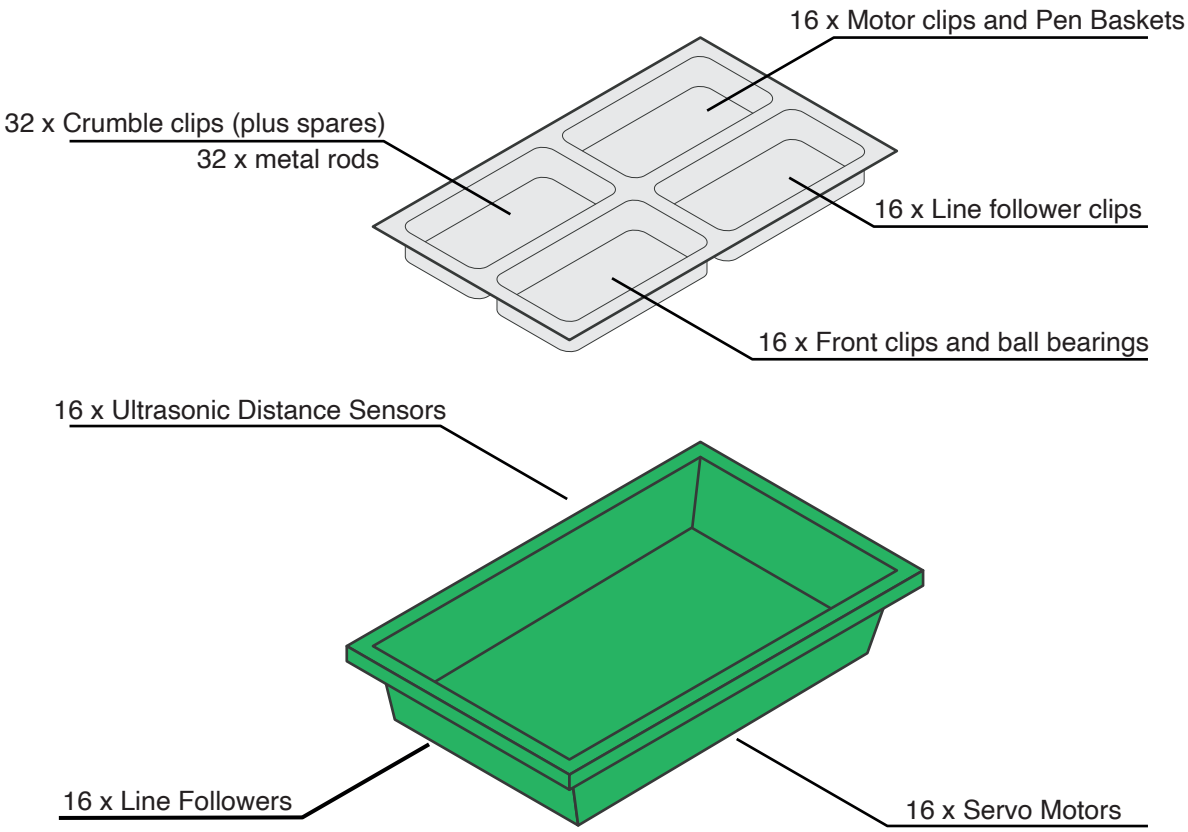
What's in my kit?

Inside your Gratnells storage tray, you will find 16 sets of:

- CUB chassis
- Ultrasonic Distance Sensors
- Line Followers
- Servo Motors

What else do I need?

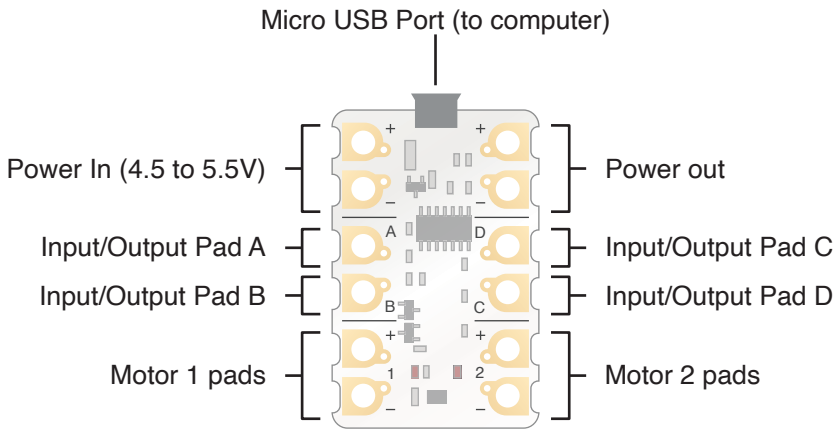
To make the CUBs you will also need a few components from your Crumble Class Pack. You will need 16 Crumbles, micro USB leads and bundles of croc leads, as well as 32 motors and wheels.



You will find that these guides are not overly prescriptive with what we provide. Like the Crumble, we try to keep projects as open-ended as possible. As such, the following provides you with curriculum links, a brief overview of possible activity progression as well as wiring diagrams and example code. We have suggested which year groups each of the activities are suitable for, but, given the broadness of the KS2 computing curriculum, it remains less of an issue as to what is completed and when, so long as you have confident pupils by the time they move onto KS3.

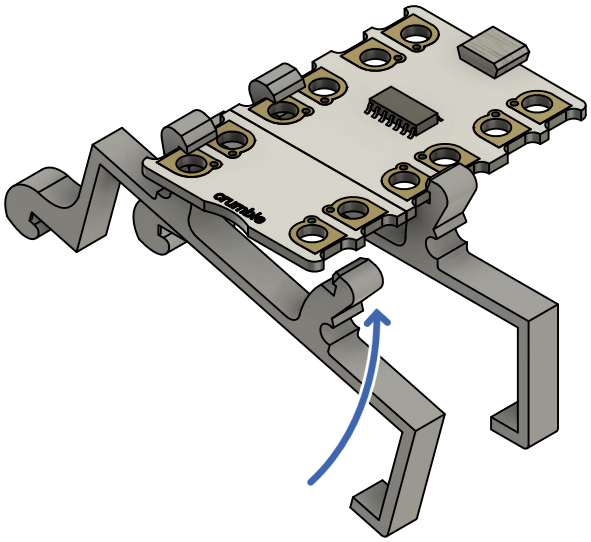
Session	Components	Curriculum Targets					
		Sequence	Selection	Repetition	Variables	Inputs	Outputs
Making it Move	Motors	■		■			■
Sparkles	Sparkles (and motors)	■		■			■
Servo	Servo	■		■	■		■
'Scaredy Cat'	Ultrasonic (and motors)	■	■	■	■	■	■
Following a Line	Line Follower (and motors)	■	■	■		■	■

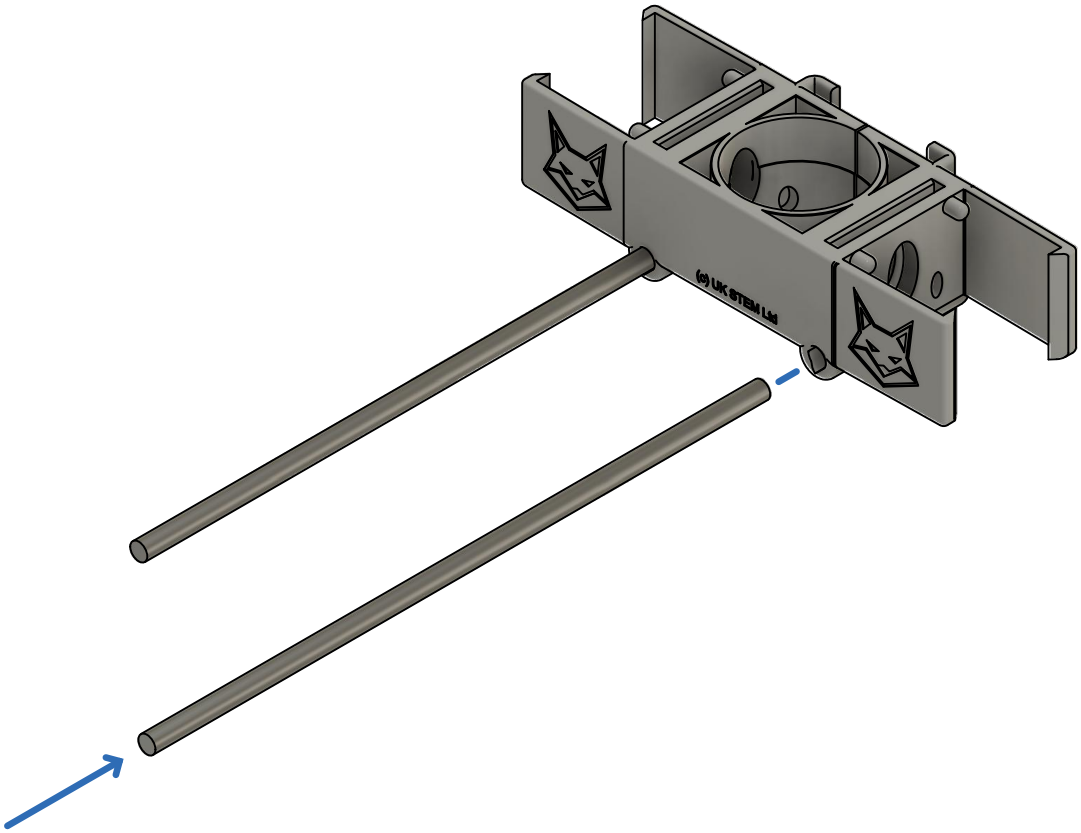
Throughout this booklet, we will refer to various parts of the Crumble Controller to assist in connecting the components together. The following diagram should provide some clarity around this language.

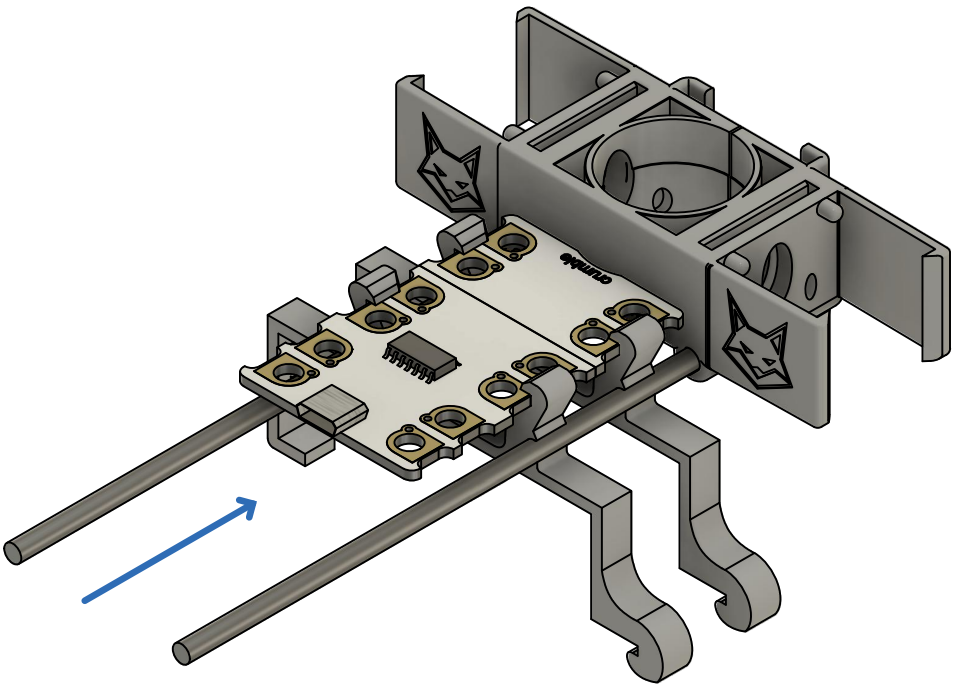


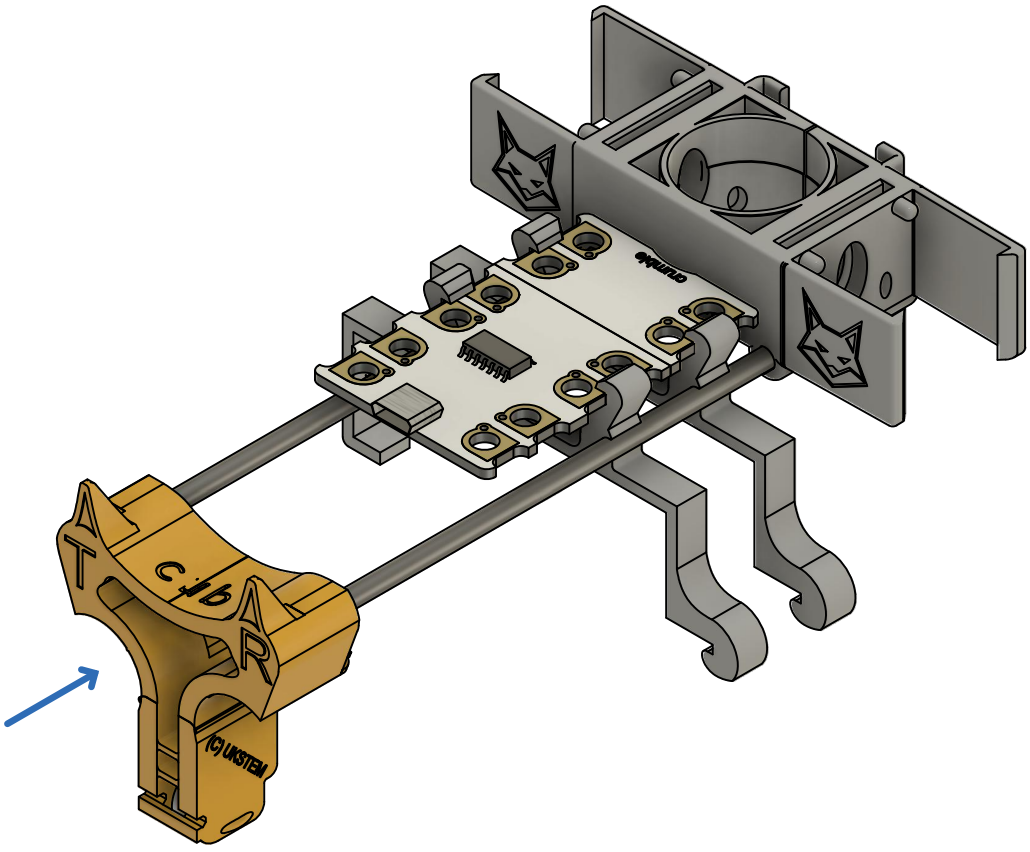
Building the CUB Chassis

Included as part of your Crumble Robotics add-on pack are the parts needed to make 16 CUBs. The following instructions give an overview of how to make the base version of the robot.

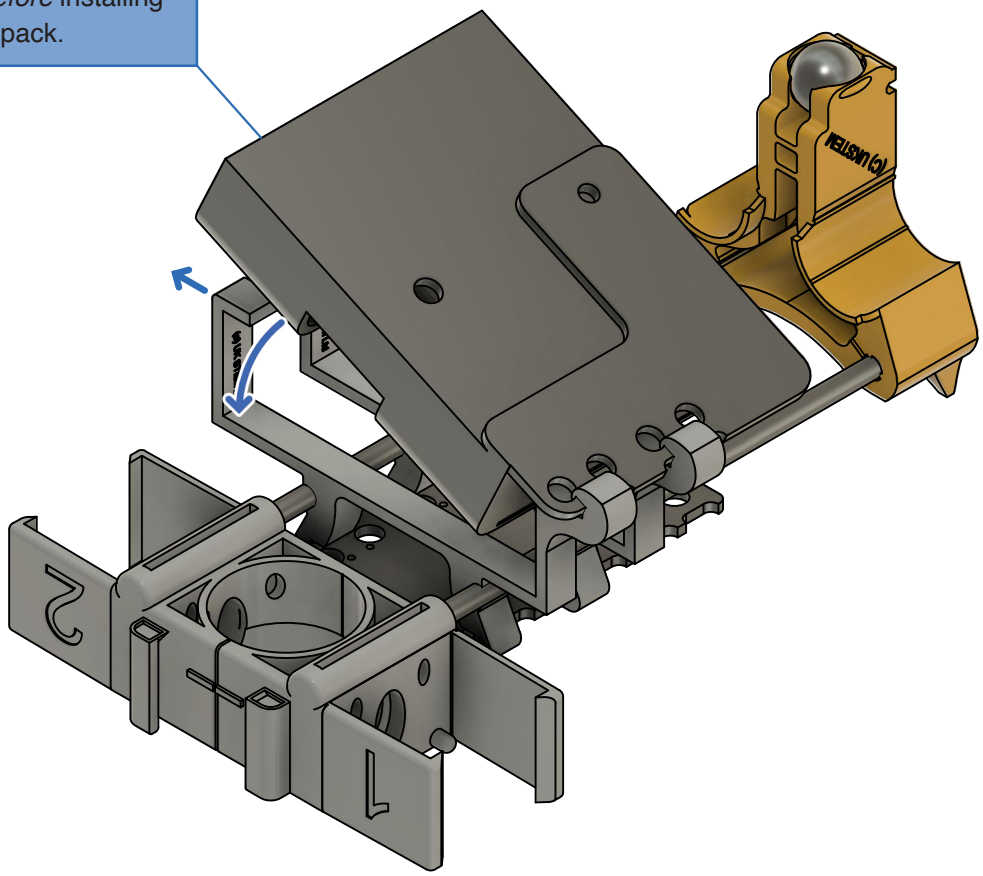


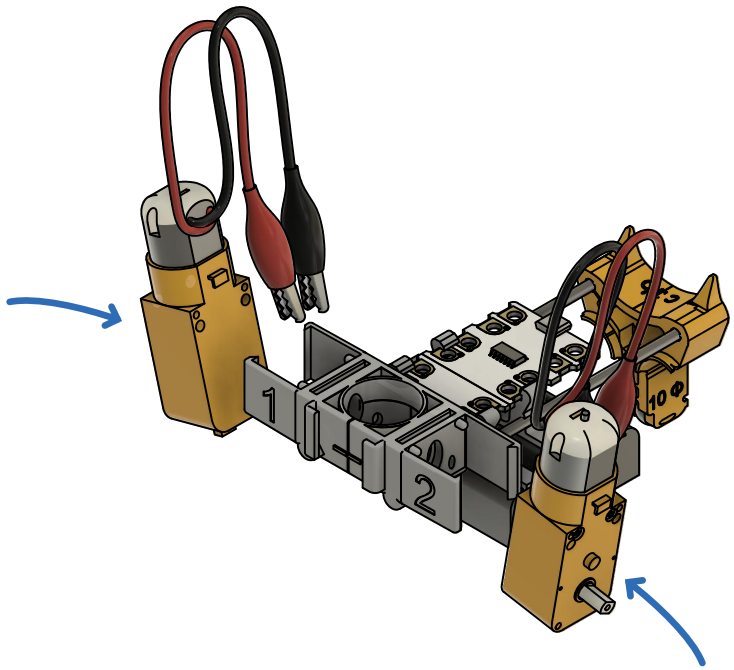


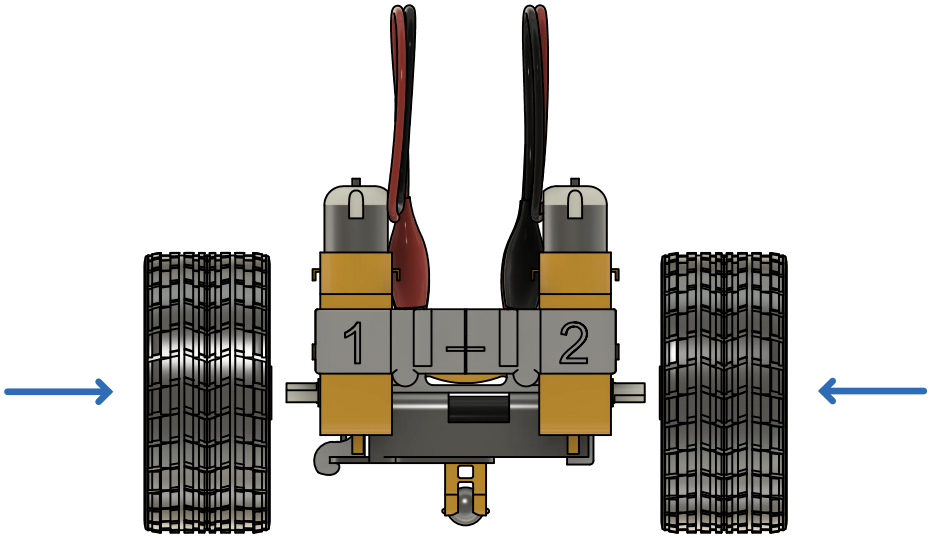


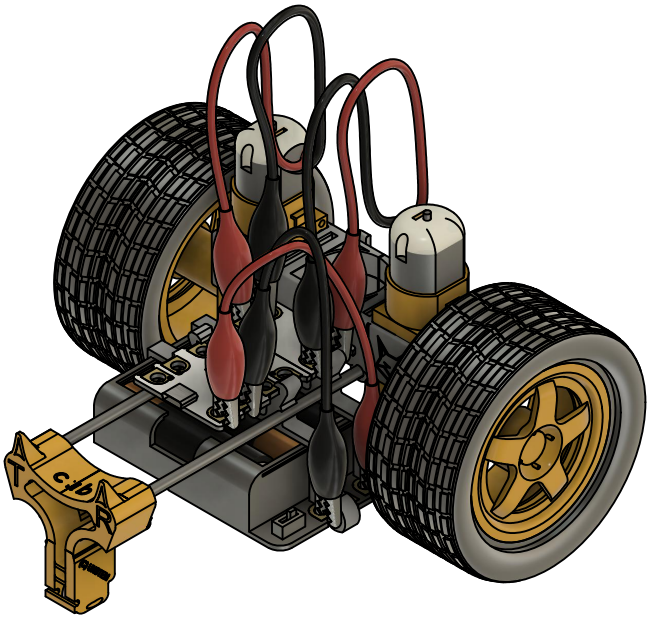

(C) UKSTEM

Don't forget to insert your batteries *before* installing the battery pack.









For wiring diagrams, see each activity.

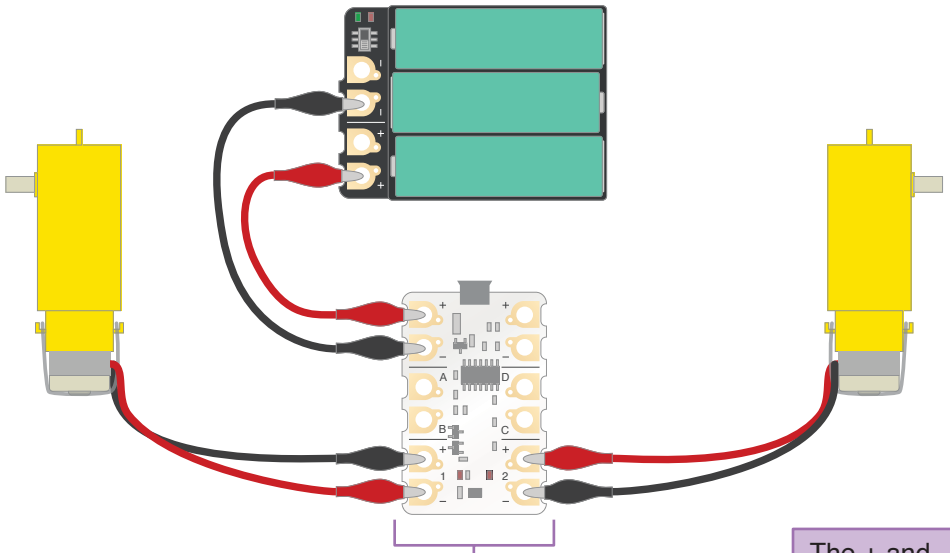
Making it Move

Recommended for Year 3+

Curriculum Objectives: To use sequence, repetition and outputs within a program

Progression: Basic movements → Drawing paths and shapes using sequential steps → Shapes and patterns using repetition

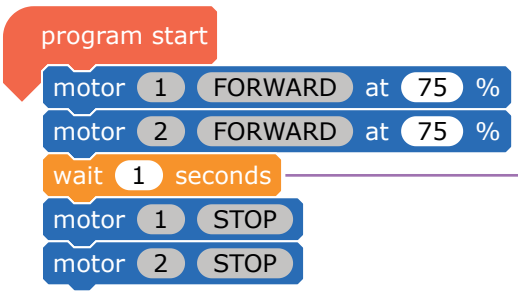
Using motors with the Crumble controller is simple to do, especially as the Crumble can directly control the speed and direction of motors. First of all, connect a motor to the Motor 2 pads, using a red lead for positive (+) and black lead for negative (–). The other motor, connected to Motor 1, needs to be wired in the opposite polarity, with a black lead to + and red to –. This is to ensure that both motors, which are arranged back-to-back, travel in the same direction when we program them to go forwards/backwards with the Crumble. For further explanation, take a look on our website.



The + and – Connections to Motor 1 and 2 are opposites

Basic Movements

Let's get our CUB to drive forwards for one second. Try running the following program:

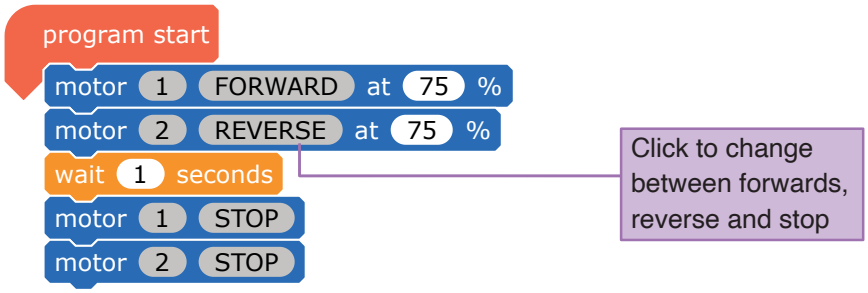


This **wait** block acts as a 'pause' between commands. Here it sets how long the motors are on for and therefore how far the robot travels.

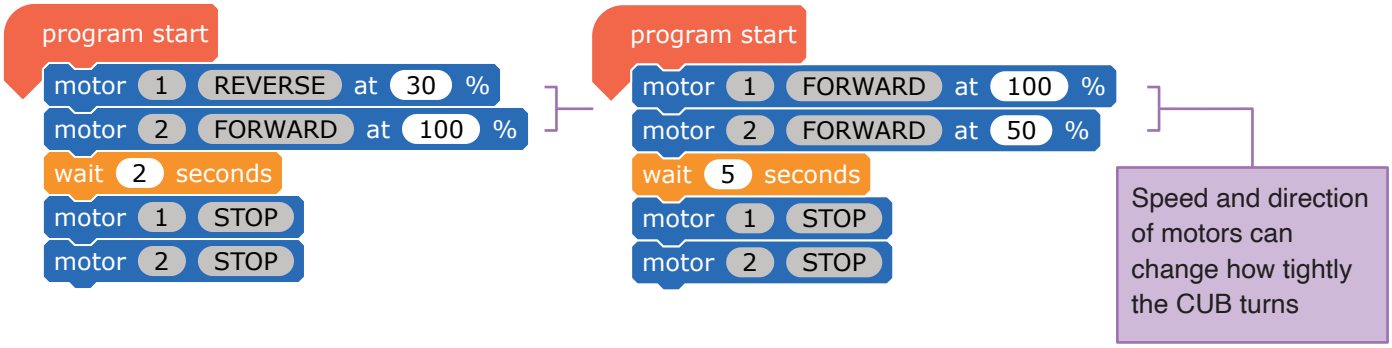
Remember to keep your battery pack *off* whilst programming your CUB. If not, your robot will try and drive off whilst still connected to the computer!

Travelling in reverse works much the same, we just have to change the direction of the motor block.

To make the CUB turn on the spot, we set the motors going in opposite directions. Similarly to before, we will use the **wait x seconds** block, but now it will determine the angle of the turn.

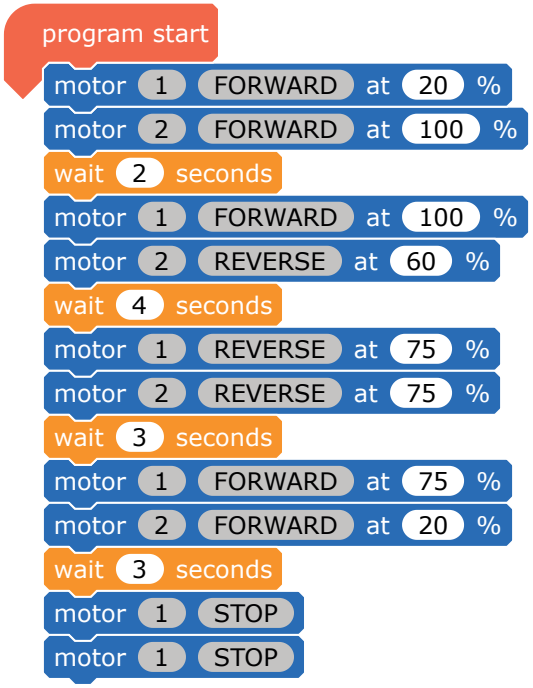


We aren't limited to on-the-spot turns either. By running each motor at a different speed, the robot will turn in an arc. Can you predict what each one will do?

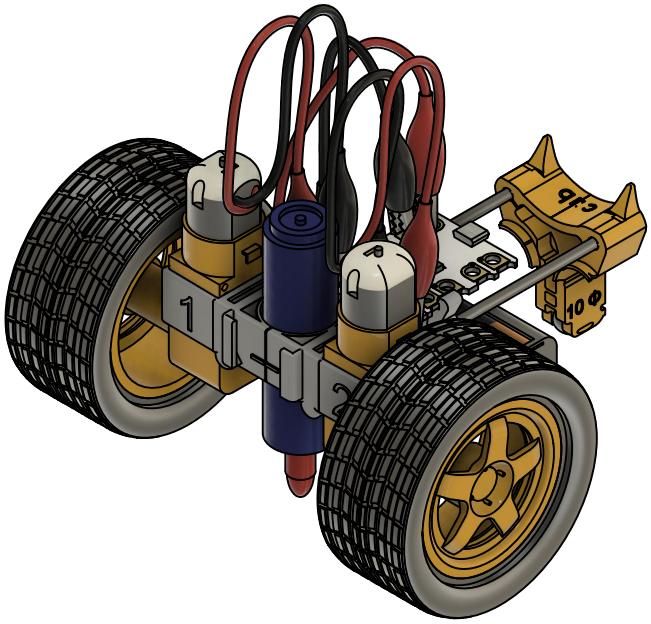


Drawing Paths and Shapes

By joining together different movements and turns, we can create all manner of different paths and patterns for our robot to travel.

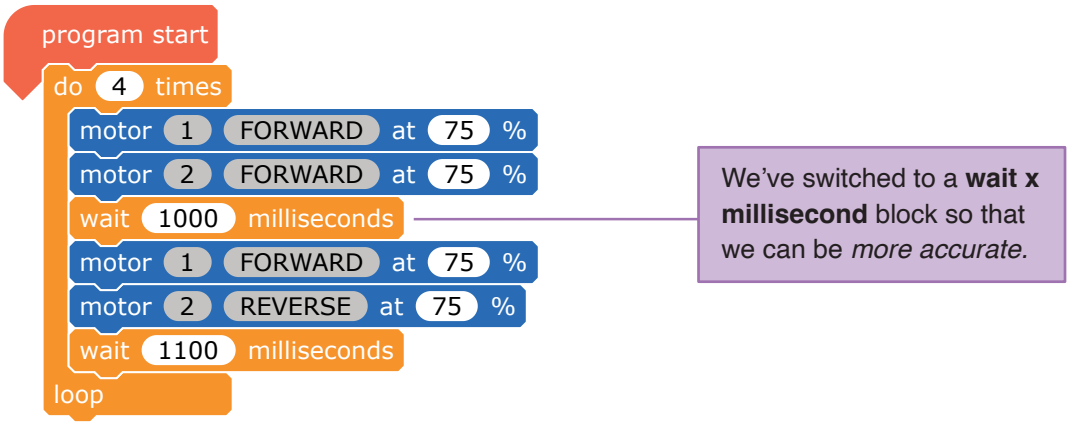


By adding a pen into back of the robot (use the basket for smaller pens), we are able to visually track and keep record of the path that the robot takes. Trust us when we say that this causes great delight in children and adults alike.



Shapes and Patterns using Repetition

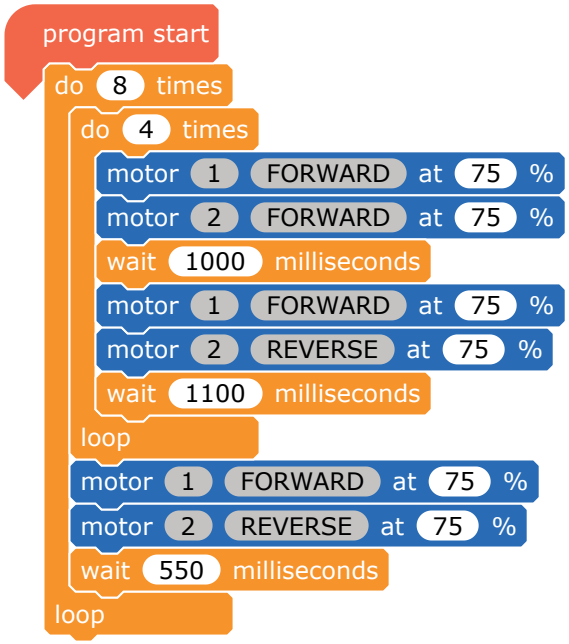
By using loops, we are able to repeat steps within a program and create more intricate and interesting drawings. We have used a **do 4 times** block in the program below. Think about what it will do, before trying it out.

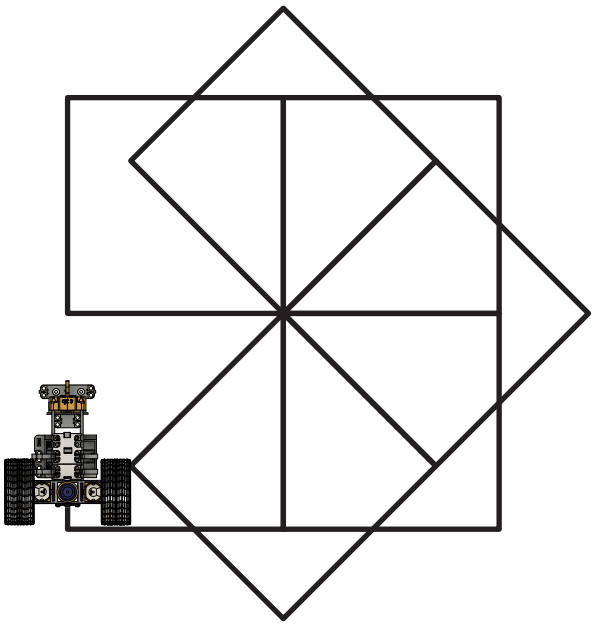


If you've got a square, fantastic. If not, then don't panic! Have a go at tweaking the wait time for the turn as this will affect the angle. Slight discrepancies are normal as the motors will vary slightly, as will the quality and charge of your batteries.

Hopefully your head is already buzzing with the cross-curricular links to maths, in particular geometry (shapes, interior/exterior angles etc).

If you want to increase the complexity of the patterns/shapes, then you can use 'nested' loops (a loop within a loop). Let's take the square program from before (you may need to substitute in your version). If we put this inside a do forever loop, we'll keep drawing a square over and over and over again. It will start to drift and erase itself – it won't look very nice at all. However if, after we complete the initial square, we turn a bit and redraw the square, we're going to get something *much* more interesting – think spirograph!





When it comes to line drawing robots, there are so many fun and exciting possibilities!

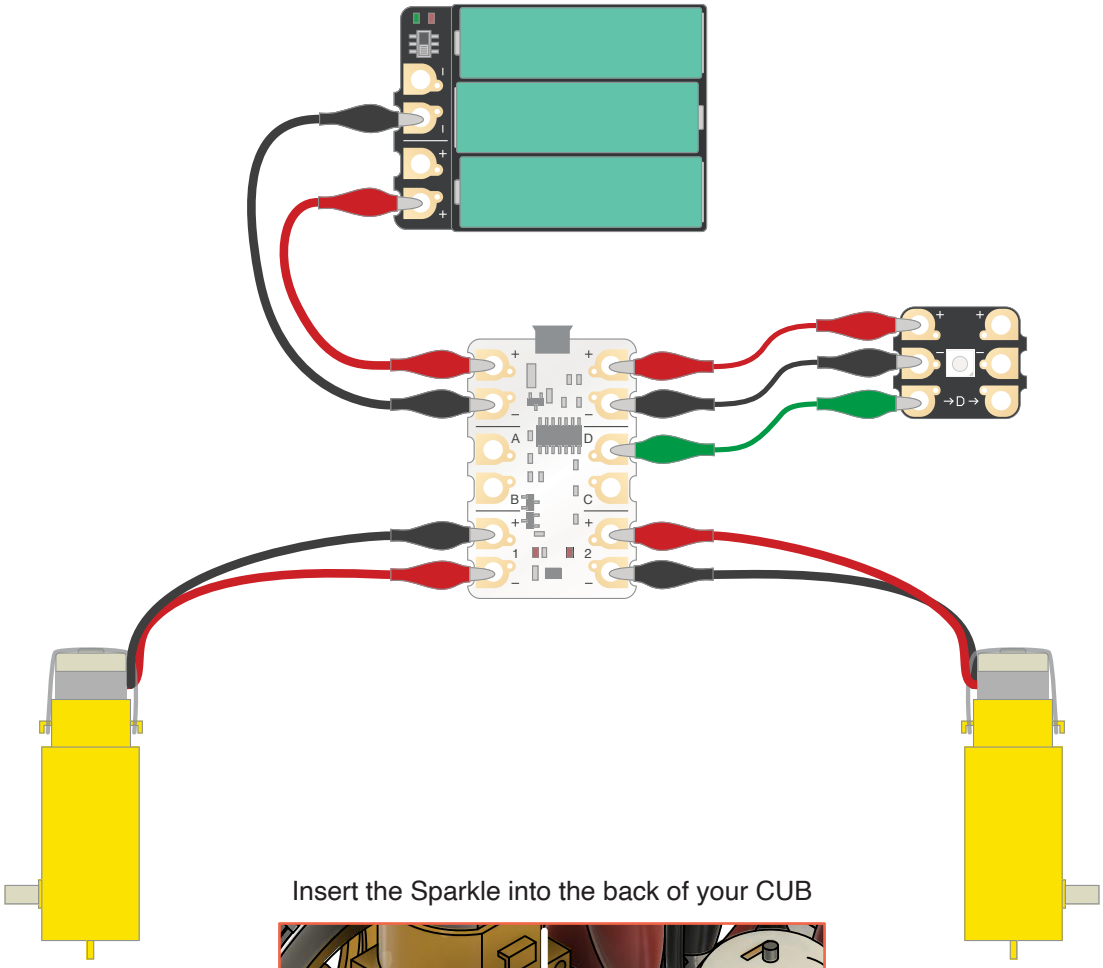
Sparkles

Recommended for Year 3+

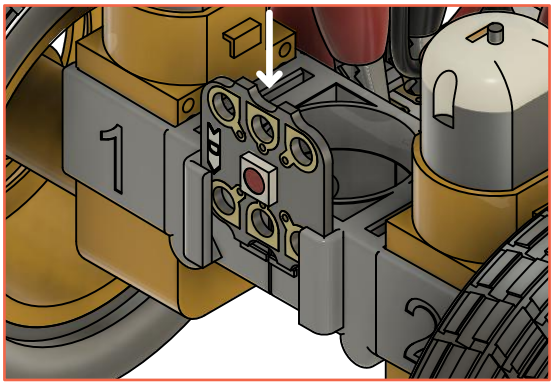
Curriculum Objectives: To use sequence, repetition and outputs in a program

Progression: Lighting it up → Changing colours → Emergency lights

The Sparkle, as you are likely already familiar with, is a special type of LED that we can control the colour of. We can also 'daisy chain' up to 32 Sparkles together and individually control each of these. To connect a Sparkle to the Crumble, you need to make three connections, one each for power (+ and –) and a third to D (which you can think of as the data connection). Unlike other components, you can only use D for connecting Sparkles. This is due to the special signal which controls the LEDs.



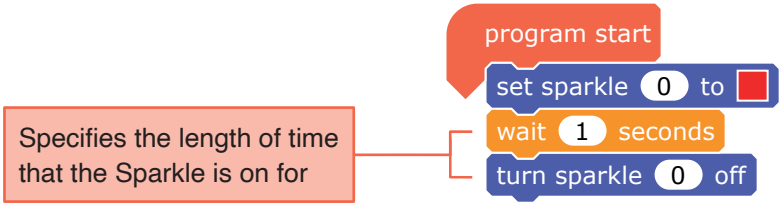
Insert the Sparkle into the back of your CUB



Lighting it up

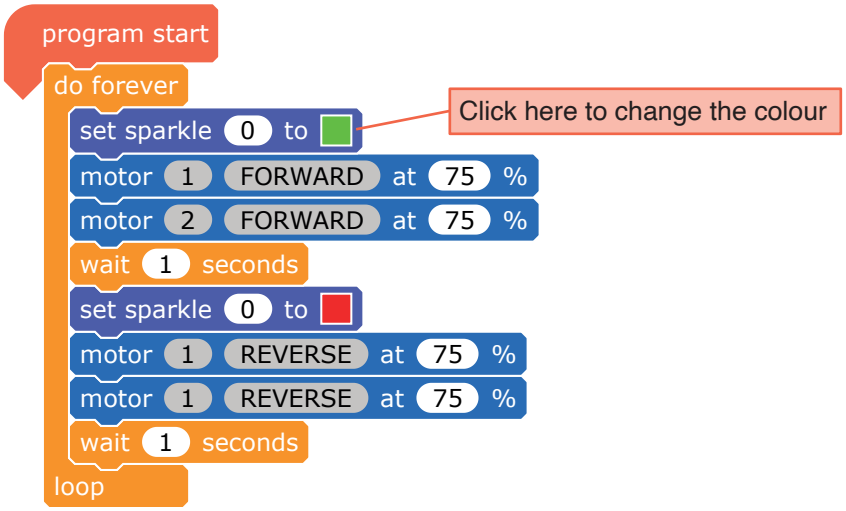
Let's start by turning the Sparkle on. When we run the following program, the Sparkle will light up red. It will only stop when we turn the power off, or stop the program.

By using a **wait x seconds** block, we are able to control how long we want to keep the Sparkle on for, or complete another action, like changing the colour etc.



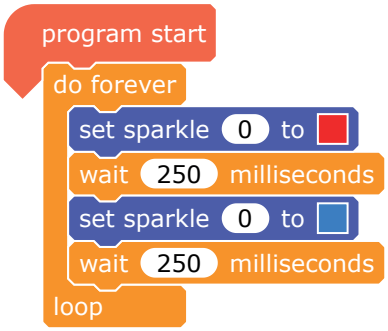
Changing Colours

Now let's consider how we can combine our Sparkle with the motors we have used previously. We are going to write a program that lights the Sparkle green when the CUB is driving forwards and red when it is driving backwards.

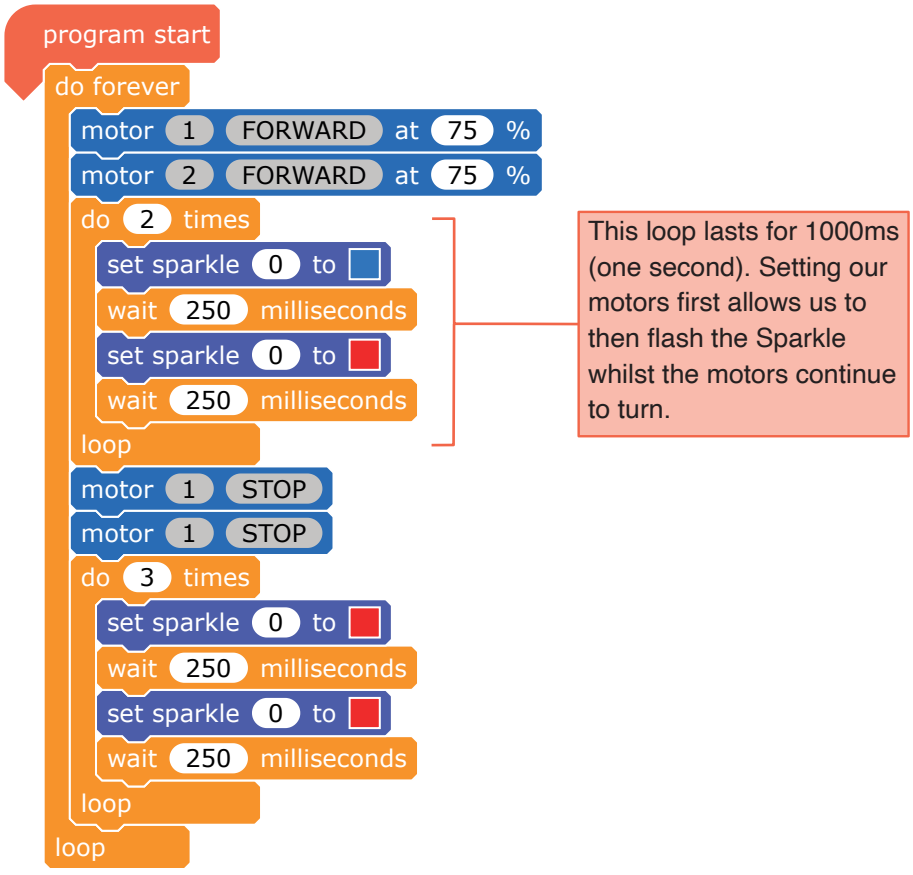


Emergency Lights

Next we're going to have a go at recreating some emergency lighting! Whilst a lot of emergency vehicles only make use of blue, we are going to alternate ours between blue and red, which makes it more effective when using a single Sparkle.



What can be more challenging is to combine a light sequence with the use of motors. The distance our CUB travels is determined by how long we leave the motors running for, in a certain direction. Similarly, the way in which the Sparkles flash/change colour is also determined by the use of strategic 'wait' statements. By combining the two, we should be able to use both the Sparkles and the motors.



This brief look at using Sparkles is only the beginning of what you can achieve with them. Try experimenting with multiple Sparkles chained together and see what you can create!

“The greatest benefit of the Crumble is its ability to help teachers and students integrate programming within the context of hands-on engineering design challenges.”

- Tyler S. Love & Abraham Bhatty
‘Technology and Engineering Teacher’, October 2019

Using the Servo

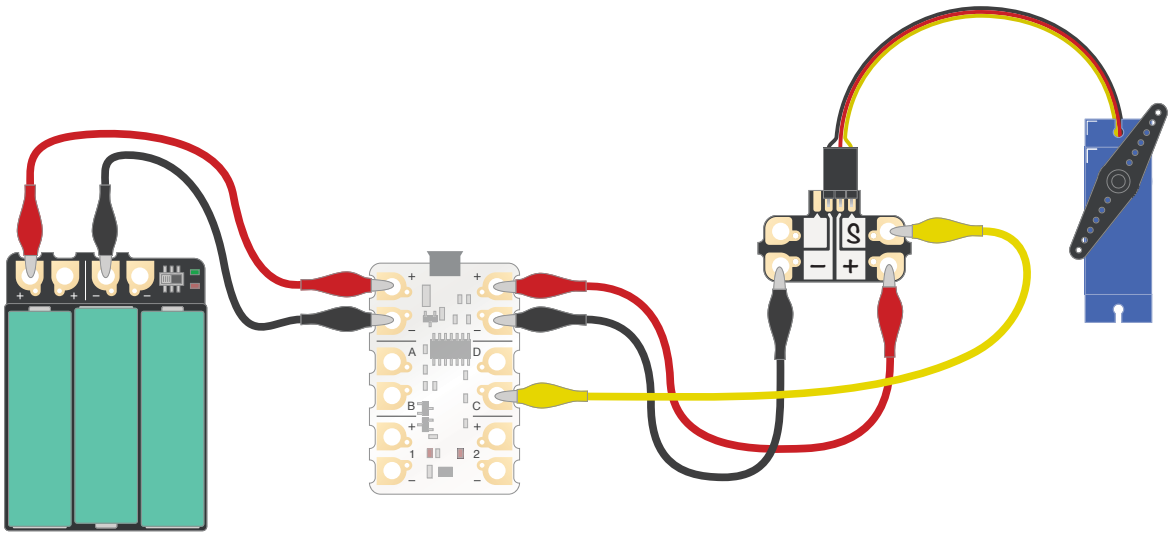
Recommended for Year 4+

Curriculum Objectives: To use sequence, repetition, outputs and variables within a program

Progression: Setting the position→ Gradual movements → Moving scenes

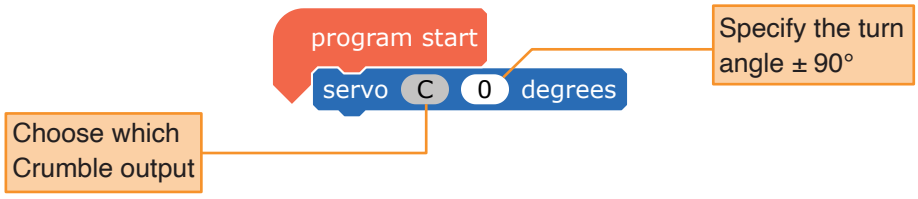
A servo motor, or servo, is a special type of motor which we can accurately control by setting it to rotate in a particular way. Whilst it doesn't much look like your average motor, believe it or not, hidden within the outer shell of the servo is a tiny motor and set of gears, as well as an equally small circuit board. Servos have a defined range of motion, with ours being 180° (or – 90° to + 90°). A repeating pulse of electricity tells the servo which angle to turn to. They are great for connecting to levers or linkages and adding motion to a project – they are commonly found in many items that require small and powerful movements e.g. robotics, rudders, grippers etc.

To begin with, we need to wire up our servo to the Crumble. It requires a + and – connection as well as a single signal (S) connection, which can be any one of A, B, C or D. We'll use C for our project.



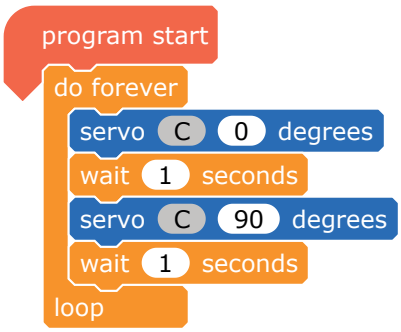
Setting the Position

To check that everything is working, we are simply going to set the servo to an angle. Try running the program. If the servo isn't already in the correct position, you should hear it move. Try changing the angle and rerun the program to see what happens. You may also want to connect a servo horn (the included black plastic pieces that fit onto the output shaft of the servo) to more easily see the movement.

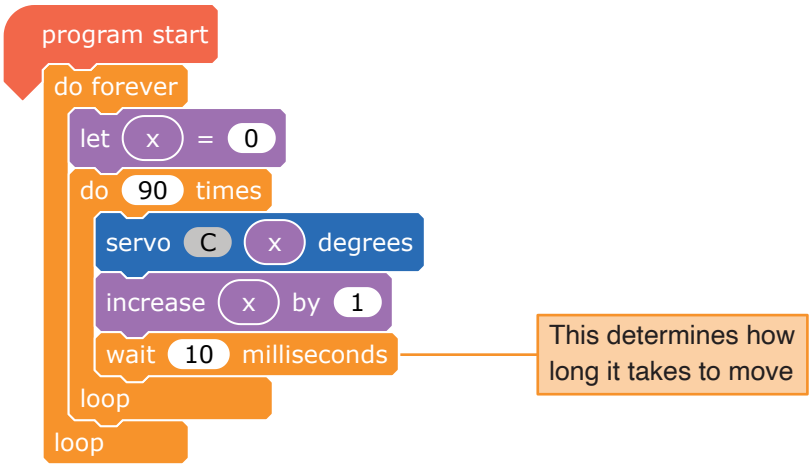


Gradual Movements

Try running the following program. Notice how quickly the servo moves between the angles. Why do you think this is the case?

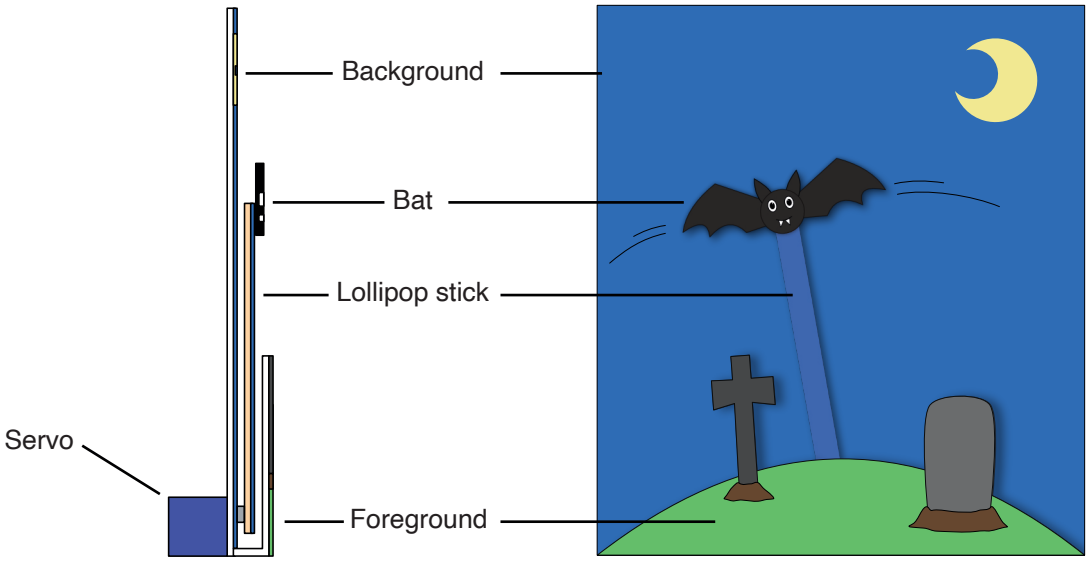


If we use a simple counted loop, we can control the speed at which the servo moves between angles. This is achieved by slowly increasing the servo angle within a loop, with a small delay between each angle. Whilst this program theoretically achieves the same as the previous one, notice how much more controlled the motion between 0° and 90° is. Can you finish off the program so that it smoothly goes back from 90° to 0°, too?

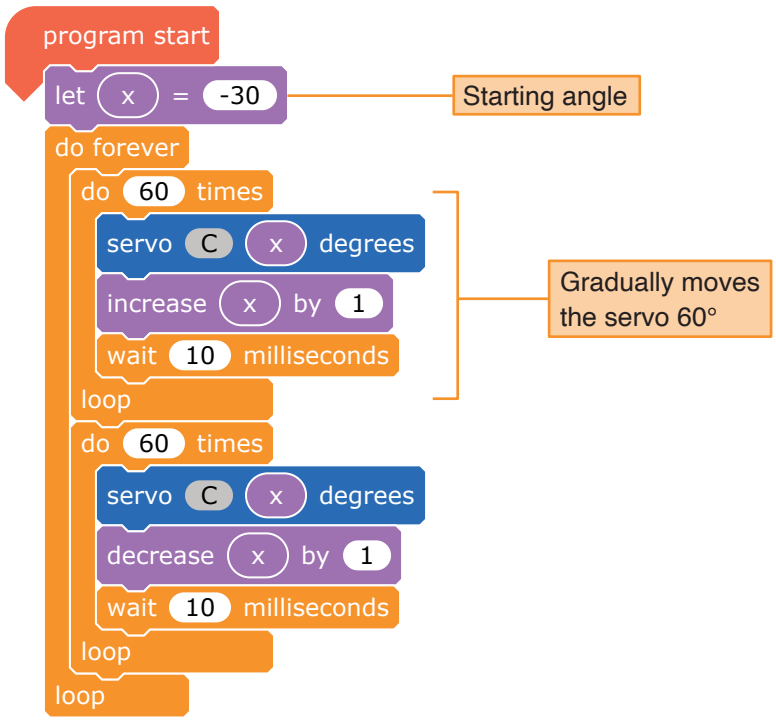


Moving Scenes

Let's now embed this within a project. We have found that one of the more popular ways of using the servo with the Crumble, is to create moving scenes, whether it be book covers or adverts. Part of the scene is attached to the servo arm and moved around, separate from the background.



A painted lollipop stick attached to a servo gives us a 'flying' bat. To control the bat, we are gradually moving the servo arm back and forth between our two set angles – in this case it's ± 30°.



The servo is an excellent tool for adding controlled movement to your projects, as well as providing some cross-curricular links with maths (angles) and D&T (levers and linkages, as well as controlling products). As always, take a look on our website for more inspiration.

“The most important skill for a computer scientist is problem-solving. Even if you don’t know all the details of the technology you are using, if you can solve the problem, you can figure out how to do it.”

- Bill Gates

'Scaredy Cat'

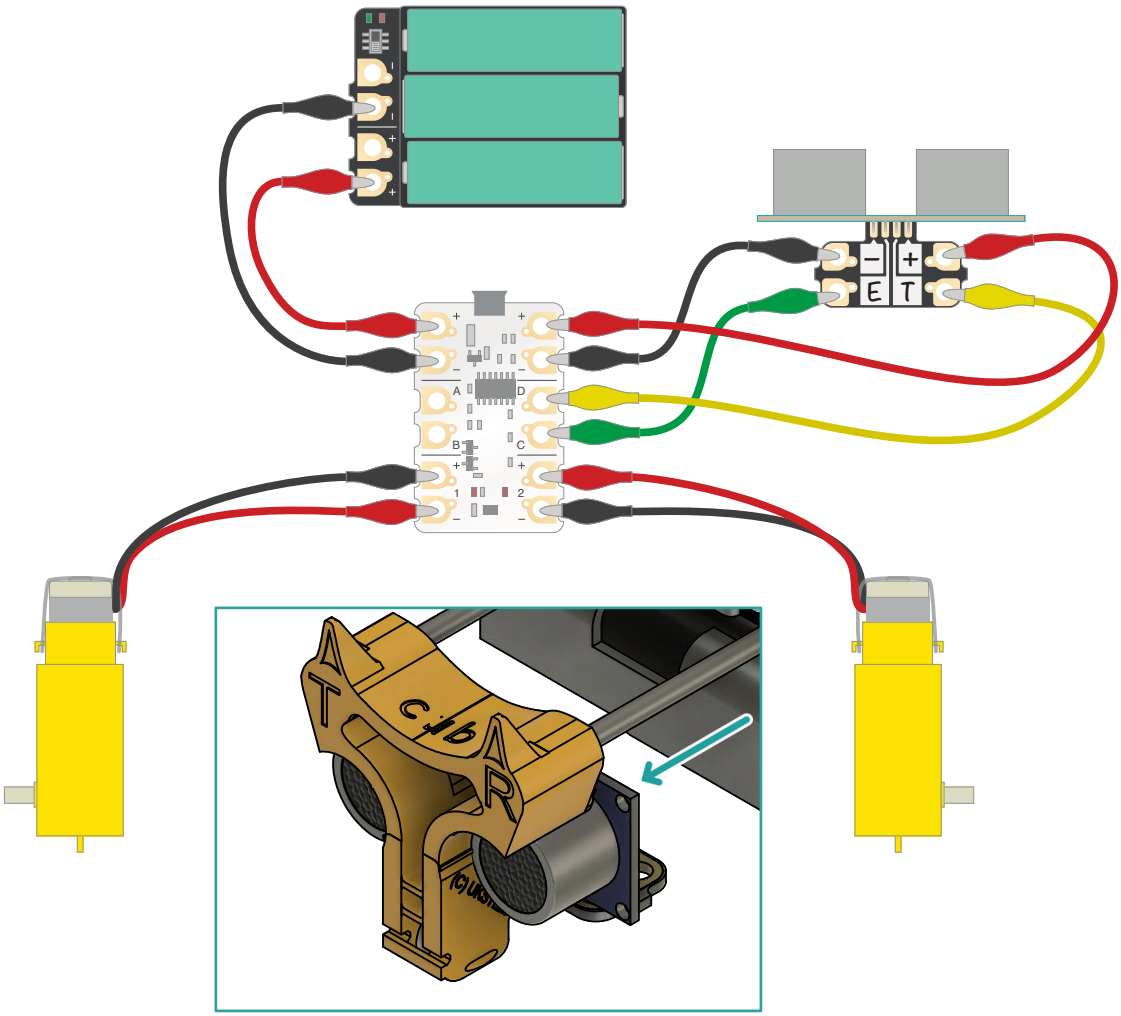
Recommended for Year 5+

Curriculum Objectives: To use sequence, selection, repetition, inputs, outputs and variables within a program

Progression: Naming variables and reading distances → Detecting a hand → Autonomous robot (object avoidance)

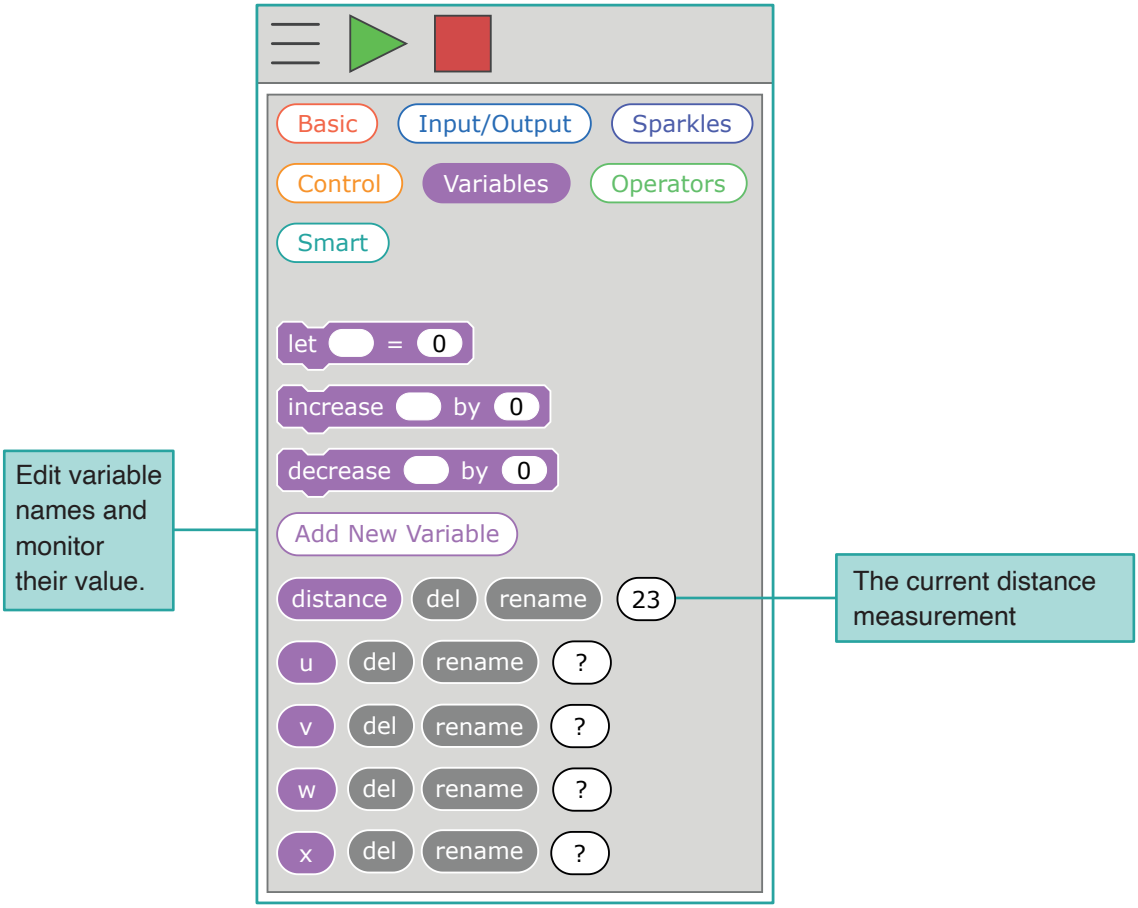
The Crumble's Ultrasonic Distance Sensor is an input device which allows the Crumble to 'see' objects and measure how far away they are. It works in a similar way to sonar or echo location, whereby a high frequency (inaudible) tone is sent out by the sensor. The sensor then measures how long it takes the echo to return. Using the speed of sound, the Crumble calculates how far an object is away within the sensor's 'line of sight'.

To begin with, we will wire up the distance sensor and use a variable to check everything is working correctly. First of all, connect + and – on the Ultrasonic Distance Sensor to + and – on the Crumble. There are two other connection points on the ultrasonic, Trigger (T) and Echo (E). Whilst these can be connected to any of A, B, C or D on the Crumble, we are going to use D for Trigger and C for Echo. We also have our motors connected as before.

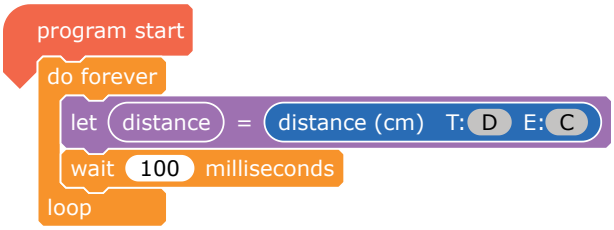


Naming Variables and Reading Distances

Let's make a simple program to store the distance measured within a variable, which we've called 'distance'. Navigate to the **Variables** tab on the software and you will be presented with a few different blocks to create, monitor and control your variables. Rename or create a new variable called 'distance'.



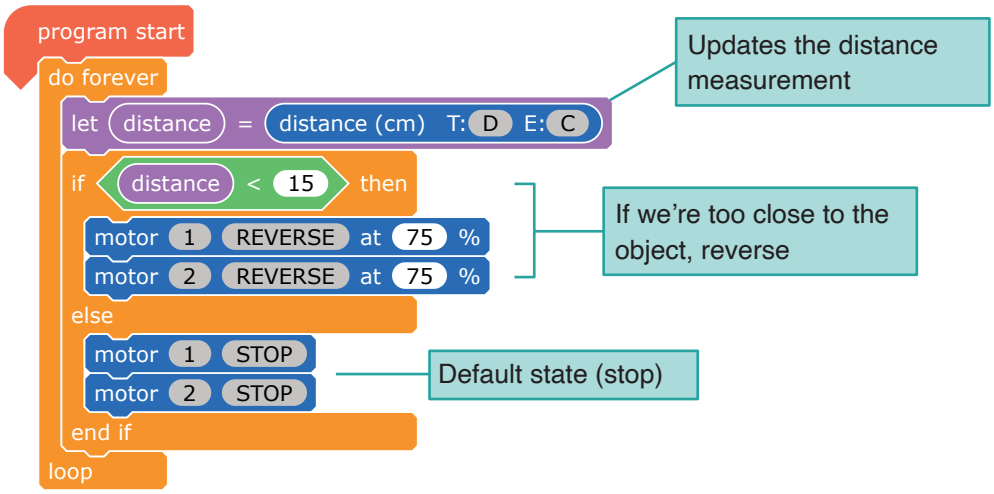
We will create a loop so that we can continually update the variable. Try putting your hand in front of the distance sensor, moving it back and forth. You should see the distance variable, in centimetres, increase and decrease as you move your hand.



Detecting a Hand

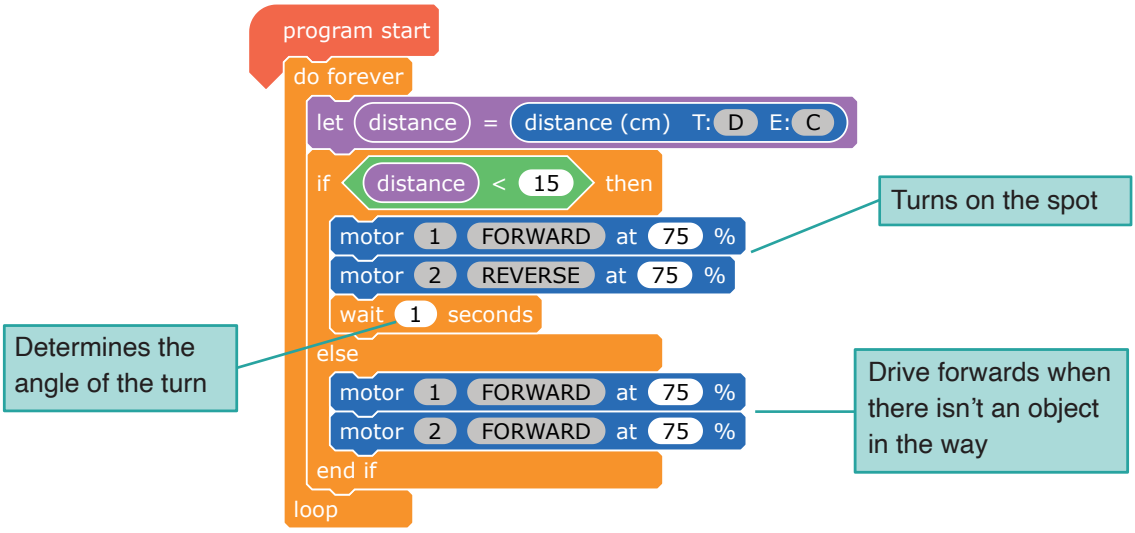
We can use this information to provide a condition for our program (selection) and make a 'Scaredy Cat' robot. This is a robot which, when you put your hand too close to it, will retreat as if it's scared of you. The program to do this is relatively simple. If something is closer than 15cm, then drive backwards, otherwise stay still. You can improve this program in a few different ways:

- Reverse for a set period of time
- Alternate the motor speeds so that the robot is more 'jittery' when reversing
- Only start reversing after something has triggered the distance condition (closer than 15cm) for more than 250ms – this helps prevent false triggers.



Autonomous Robot

Now we can start to create an autonomous robot. Our CUB robot will drive forwards until it detects an object closer than 15cm. At this point, it will turn away and continue driving.



As with all programs, further improvements can be made. In this instance, you could randomly turn either left or right, as well as for a random about of time, resulting in a different angle.

Following a Line

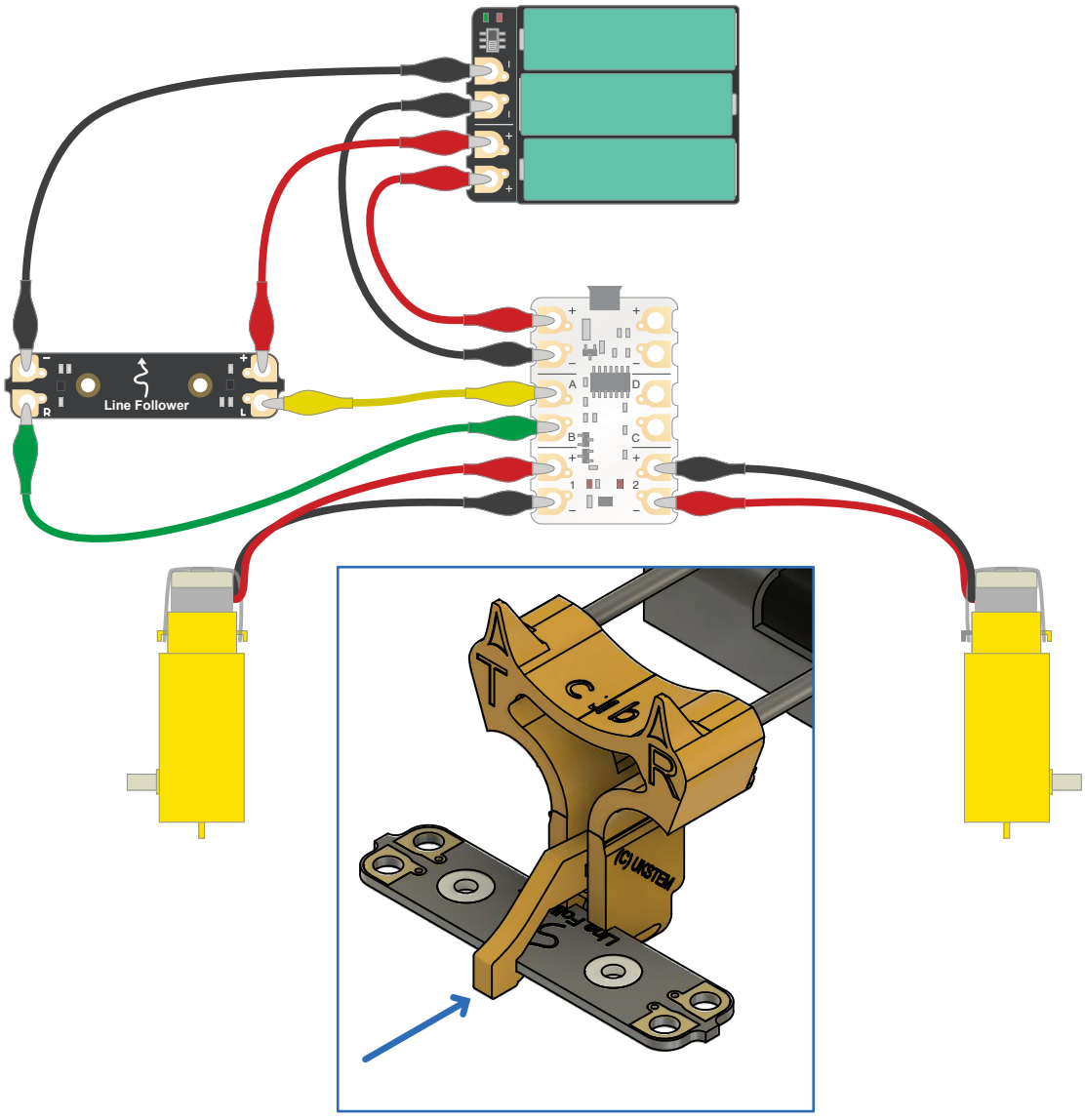
Recommended for Year 5+

Curriculum Objectives: To use sequence, selection, repetition, inputs and outputs within a program

Progression: Line detection → Creating a boundary → Algorithm to follow a line

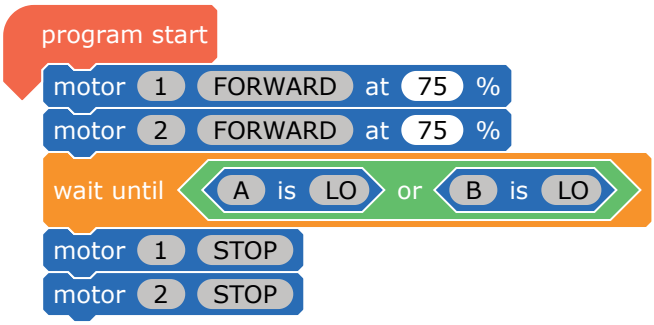
The Crumble Line Follower is a form of input device which uses phototransistors to detect the difference between a black and white surfaces. The Line Follower board outputs infrared light and uses two phototransistors to measure the light reflected back. A lighter surface will reflect more light than a darker surface. We can use this information to detect changes in surface colour and, therefore, create a robot which behaves in a particular way when it comes across these changes, for example follow a line.

First of all, clip the line follower into its holder on the front of the CUB robot and wire it to the Crumble as shown.



Line Detection

To start with, let's get our CUB to come to a complete stop if it detects a line with either sensor. The program below drives the robot forwards and then, when either the left or right sensor are LO (on a dark line), stops the motors.

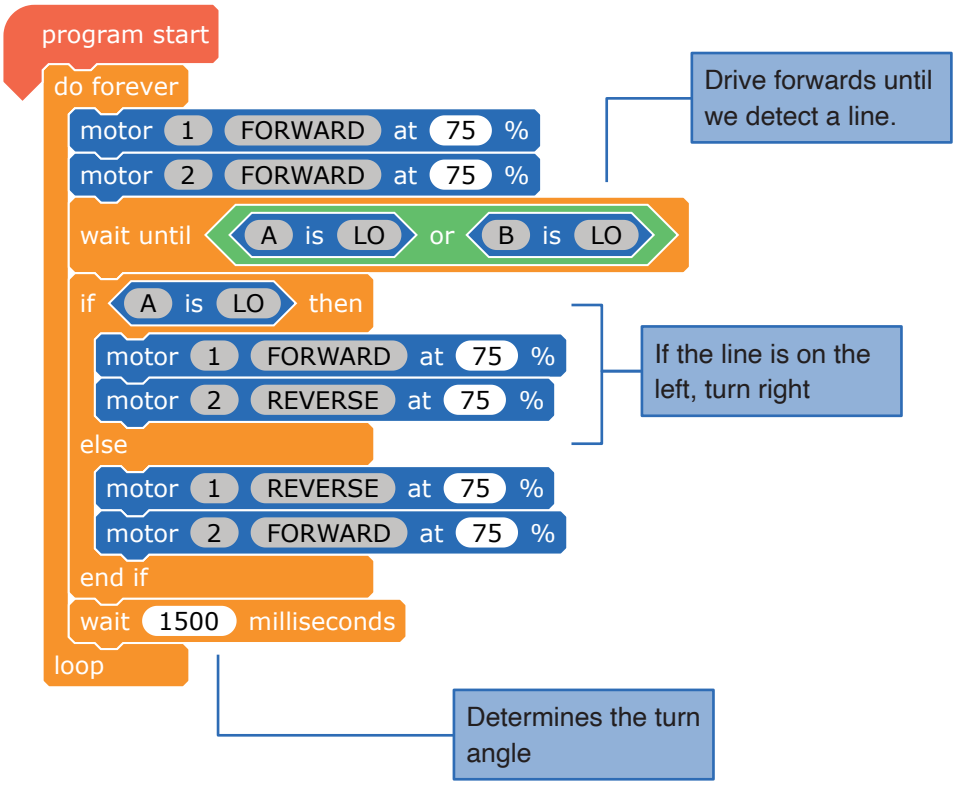


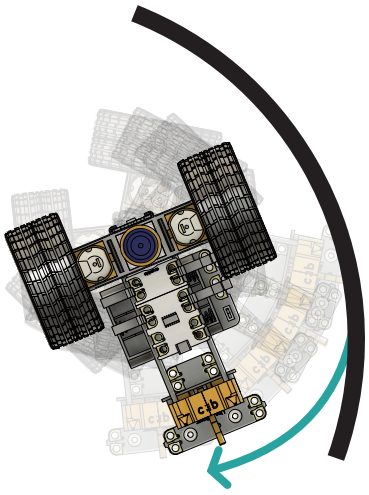
What are HI and LO?

These refer to how close the input voltage on the Crumble is to being at its maximum (**HI**) or being 'off' (**LO**).

Creating a Boundary

We can take the same principle and create a robot which will remain inside a given area or boundary (not dissimilar to robot vacuums or lawnmowers). This time, when we detect a line, we want to turn away from it. If we detect the line on the left side first, then we want to turn right. Conversely, if we detect it on the right, we want to turn left. This is the basis of how our program will work. Drive forwards until we detect a line, then turn away from it.





The CUB turns away from a border line

Algorithm to Follow a Line

Now let's consider how we would use the principle of the last project, to create a line following robot. Whilst there are many different algorithms to follow a line, we're going to stick with one that we feel helps you to grasp the how and the why. Let's begin with the robot placed perfectly over the line – each side of the line follower should be HI. If this is the case, we want to keep going forwards. When we get to the point where we are going to cross the line, either because it has curved, or because the robot isn't travelling perfectly straight, we need to turn to adjust, depending on whether the left or right side of the line follower has been set to LO (crossed into the black line). The following table should help break down what is happening.

Line Follower		Motors		Turn Direction
Left	Right	Motor 1	Motor 2	
HI	HI	On	On	Straight
LO	HI	Off	On	Turn left
HI	LO	On	Off	Turn right
LO	LO	Off	Off	Stop*

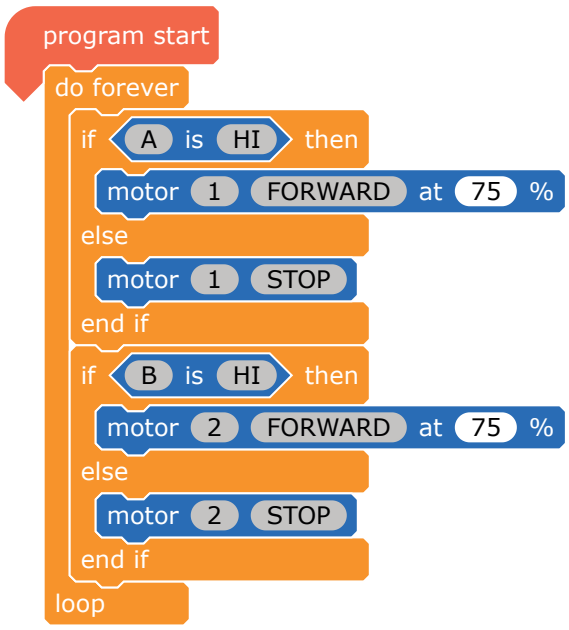
**This is when we detect a line on both sides of the follower. This could happen if you have a line that is too wide, or you have a bend that is too sharp. Instead of stopping you could keep going forwards, and hope that you pick up the line correctly.*

If we were to write a Crumble program mapping what we've 'learnt' from the table, we would end up with a lot of **if x then** statements. Fortunately, we can simplify it! Can you notice a pattern between what is happening with the 'Line Follower' section and the 'Motors' section?

Line Follower		Motors		Turn Direction
Left	Right	Motor 1	Motor 2	
HI	HI	On	On	Straight
LO	HI	Off	On	Turn left
HI	LO	On	Off	Turn right
LO	LO	Off	Off	Stop*

■ Over Black ☐ Over White ■ Motor On ■ Motor Off

Compare the 'Left' column of table to the 'Motor 1' column. Notice how **HI** corresponds to **On** and **LO** corresponds to **Off**. Similarly with the 'Right' column and 'Motor 2'. This simple relationship allows us to control Motor 1 (the left motor) with the left input of the Line Follower and Motor 2 with the right input. It may help to place your CUB over a line and follow the program, below, and work through each scenario.



Whilst we've suggested having one motor on and the other off, you may want to tweak your program so that the 'off' motor is slowly travelling forwards at say 15%. This is useful as you will get a wider turn and therefore continue to travel forwards a bit. This may help if you are completing a time trial run etc.

As already mentioned, this is only one of many different algorithms to create a line-following robot and it certainly has room for improvement. Can you come up with a different solution?

Hopefully the starting projects outlined throughout this booklet have given you the information needed to get started with robotics projects using the Crumble Controller. Take a look on our website for more inspiration or more detailed 'how-to' guides.

FAQ

Are you still stuck?
Get in touch!

Why isn't my robot following my instructions?

Make sure you've connected the USB cable to both the Crumble and the computer, click the green triangle and look out for the 'Programming successful' message.

Why is my program not having the desired outcome?

Particularly if you have a complicated program, try breaking it down into smaller parts (decomposition), checking that each of these has the desired effect before rebuilding the larger program.

Do I have to use the same coloured cables?

No of course you don't! However, it is best practice to stick to some form of convention e.g. using red for + and black for –. We also find that it makes following diagrams easier!

Does it matter if I connect Crumbs/accessories to the battery pack instead of the Crumble?

Other than the motors, it shouldn't make any difference where you get your power from.

My Sparkle isn't changing colour!

If you only have one Sparkle connected, make sure that you have got a '0' in the index box on any relevant Sparkle blocks.

Why is my Sparkle is only lighting up reddish hues?

The Sparkle contains three emitters – red, green and blue. These are set to varying brightnesses to make a whole spectrum of colours. The blue emitter in particular requires the highest voltage and as such, when your batteries start to become depleted you will find that the blue, followed by the green colouring start to fade. You will need new batteries.

My line follower isn't working as expected.

Check all connections are correct. Fresh batteries.

I can't think of any other line-following algorithms – help!

We have a blog on our website all about this! Head on over to find out more.

Why is my servo 'jittery'?

Fluctuations is power supply can sometimes cause a Servo Motor to behave in a slightly peculiar way, especially if you are trying to control motors/Sparkles etc all at once. Try staggering the time between using different outputs. Holding a fixed position can also amplify this. If it is an issue, and the servo isn't under strain, use the servo off block.

My ultrasonic isn't working.

Double check all of the wiring, as well as which pads you've specified you're connected to, in the Crumble software. It can also be down to battery charge.

Acknowledgements: Thank you to Mike Cargill at UKSTEM for developing the CUB chassis.