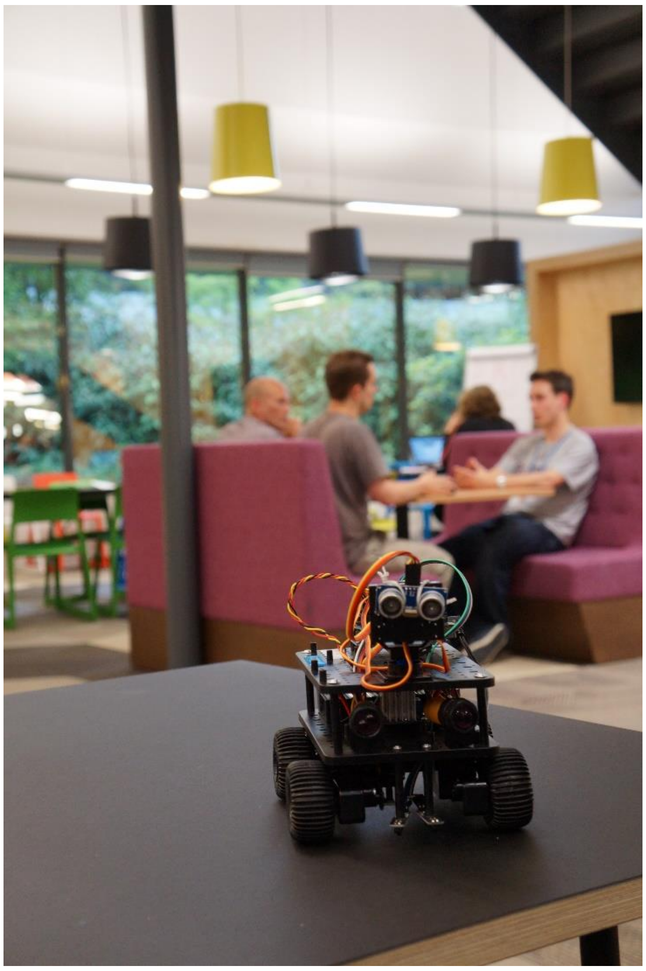


BARCLAYS ROBOTICS CHALLENGE

MENTOR SURVIVAL GUIDE

Table of Contents

Introduction

Welcome to the Barclays Robotics challenge and thank you for being a mentor.

The purpose of the Robotics Challenge is to teach pupils (and adults) the basics of computer programming using mobile robotics.

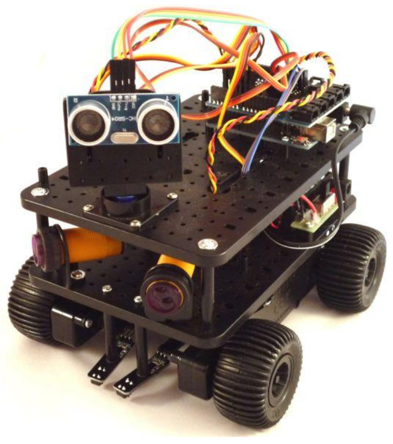
Programming is a fairly simple idea. You have a limited number of words that you can use to make a computer application. You use computer code to express a real world physical problem. Once you have learned the words and the grammar (the syntax) – it's the understanding of modelling the problem that is where the real challenge lies.

In the Robotics Challenge, we do exactly that – we give a number of real world problems that need to be solved by computer programming using robotics. That is the essence of the challenge - to teach how computer code can be used to solve problems. Using robotics gives a real world feel to the learning. If the robot doesn't go the way you want it to – then it must be something to do with the code. You can see it and you can feel it and of course you can fix it.

You need to have a reasonably good grasp of basic programming concepts. The best way to learn is to do the challenges yourself with a robot and a laptop. There are plenty of resources on the internet to teach you how to program – but learning how to model problems with code is mostly about doing it.

We will be using the Arduino http://arduino.cc/ programming IDE (Interactive Developer Environment) to write code and then send the code to the robot.

The robot is a 4tronix - 4 wheel, twin motor device which uses an Arduino board to control the motors and read the sensors. There is plenty of information on the internet about this – so please – Google is your friend.



There are 5 sensors on the robot – an Ultrasonic sensor that is able to measure distance of objects between 100cm and 0cm. There are two infrared sensors that are able to detect a surface ahead – roughly between 5cm and 20cm. And there are two infrared line following sensors that are able to see black and white and can be used to follow a line.

There are two motors which drive left and right wheels – each set of wheels is geared from one motor. So it's possible to drive the wheels with variable speed forward and backwards – meaning you can go forward and reverse and also turn the robot either by differential speed (each wheel moving at a different speed) or spin on its own axis by spinning each set of wheels in opposite directions.

Each laptop is set up with the Arduino software already installed and 4 primary sketches. There are also a number of other sketches that can be loaded – they don't all work – so if you want to use them – you will need to understand them and load them. The idea is that we give a minimal program – which is called a sketch – and from there you can write code and complete the challenges that will be set.

We have written some back end code that means you can use natural name functions like forward and reverse to drive the robot. At the backend – the code is actually just switching motors on or off or reading digital pins on the Arduino board.

How the Functions work

These functions are based on the idea of teaching sequential programming. Do one thing then do the next thing and then the next thing and then repeat. If something is of a particular value do this, otherwise do that.

It’s all about using the limited functions to achieve the challenge set.

There are six basic move functions:

forward, reverse, spinLeft, spinRight, halt and delay.

forward and reverse functions take three values – duration, left speed and right speed. The left and right motors can be set at different speeds and to make the robot turn.

spinLeft and spinRight take two values – duration and speed.

halt and delay only take one value – duration in milliseconds.

You can also use spinLeft and spinRight to do a fast turn “on a sixpence”

This example sketch will make the robot go forward for one second, stop for 1 second and then go backwards for 2 seconds:

```
forward(1000,255,255);

reverse(2000,255,255);
```

After the delay time – the motor will stop. So the behaviour of the robot will be to start moving for say 3 seconds and then stop. The code then moves onto the next command.

If you use the delay(time); function you can wait between calls for example

```
forward(1000,255,255);

delay(5000);

reverse(2000,255,255);
```

Would move the robot forward for 1 second – stop and wait for 5 seconds then go back for 2.

Reference Functions

forward(milliseconds,speedLeft,speedRight);

Move forward with left motor at “speedLeft” and right motor at “speedRight”. The function will wait for the specified number of milliseconds before returning and moving to the next command. The motors will stop after the number of milliseconds.

reverse(milliseconds,speedLeft,speedRight);

Move backwards with left motor at “speedLeft” and right motor at “speedRight”. The function will wait for the specified number of milliseconds before returning and moving to the next command. The motors will stop after the number of milliseconds.

leftSpin(milliseconds, Speed);

Spin left with motors at speed “Speed”. The function will wait for the specified number of milliseconds before returning and moving to the next command. The motors will stop after the number of milliseconds.

rightSpin(milliseconds, Speed);

Spin right with motors speeds “Speed”. The function will wait for the specified number of milliseconds before returning and moving to the next command. The motors will stop after the number of milliseconds.

halt(milliseconds);

Stop the motors and wait “milliseconds”. This is not required for the move commands.

delay(milliseconds);

Wait “milliseconds” and then return and continue to the next statement.

leftObstacleSensor() == true | false

Returns true if left Infrared sensor is active (has detected something) and false if nothing is detected.

rightObstacleSensor() == true | false

Returns true if right Infrared sensor is active (has detected something) and false if nothing is detected.

leftLineSensor() == BLACK | WHITE

Returns a value of BLACK or WHITE

rightLineSensor() == BLACK | WHITE

Returns a value of BLACK or WHITE

tiltCentre() ;

Causes the tilt and pan sensor to be centred. This is the tilt action

pointCentre() ;

Causes the tilt and pan sensor to be centred. This is the point action.

tiltUp() ;

The tilt and pan sensor will be tilted upwards fully.

tiltDown() ;

The tilt and pan sensor will be tilted downwards fully.

pointLeft() ;

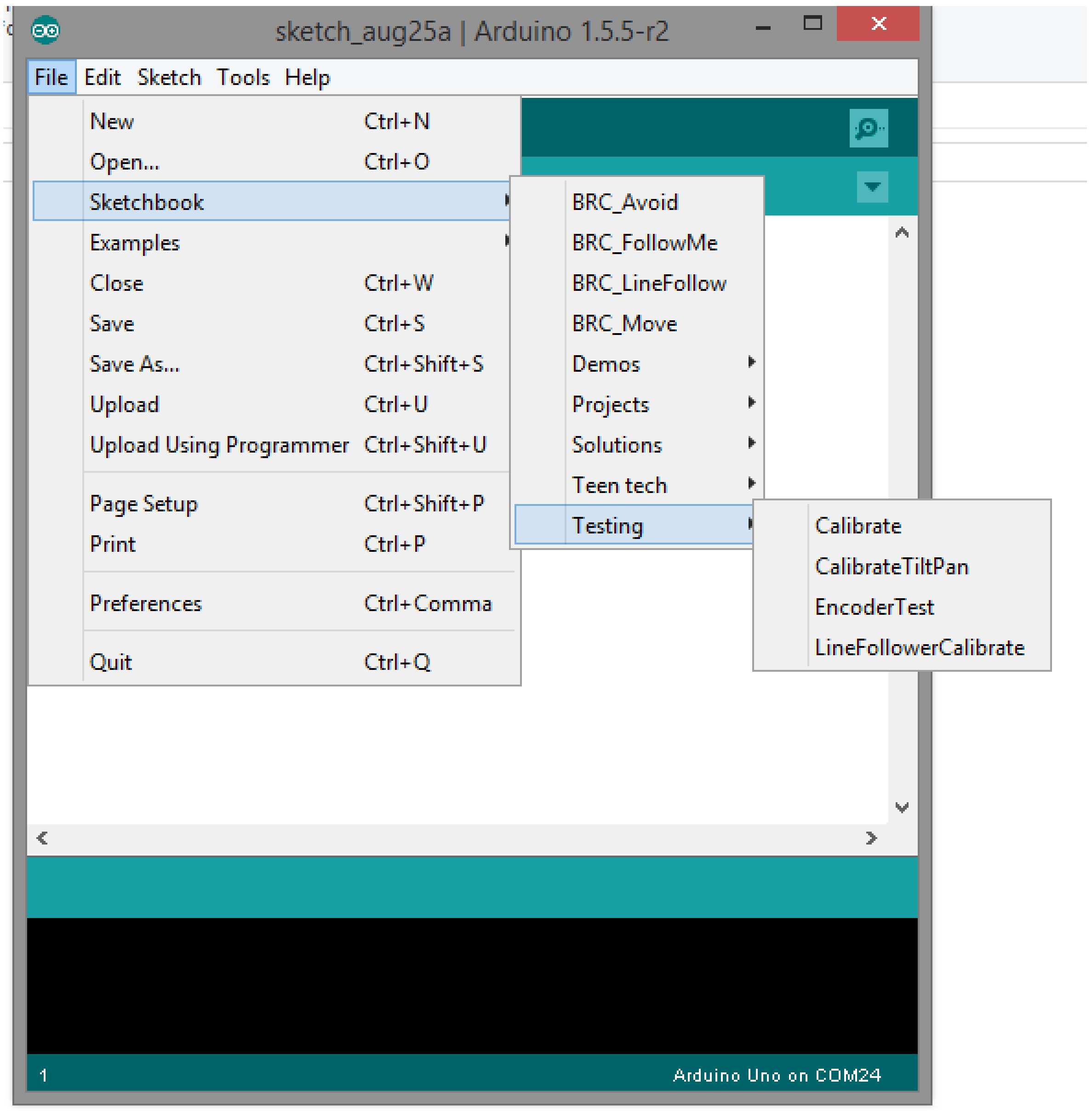
The tilt and pan sensor will turn and fully point left.

pointRight() ;

The tilt and pan sensor will turn and fully point right.

Arduino Setup

Make sure you have the correct set up. When you run the Arduino you must see these sketches and folders. If you don't – the laptop is wrongly configured. Seek assistance.



There are only 4 sketches available

Move – Simple Move – back and forward and a spin

Goal – A more complex sketch to be use for advanced challenges – employs the use of the Ultrasonic sensor.

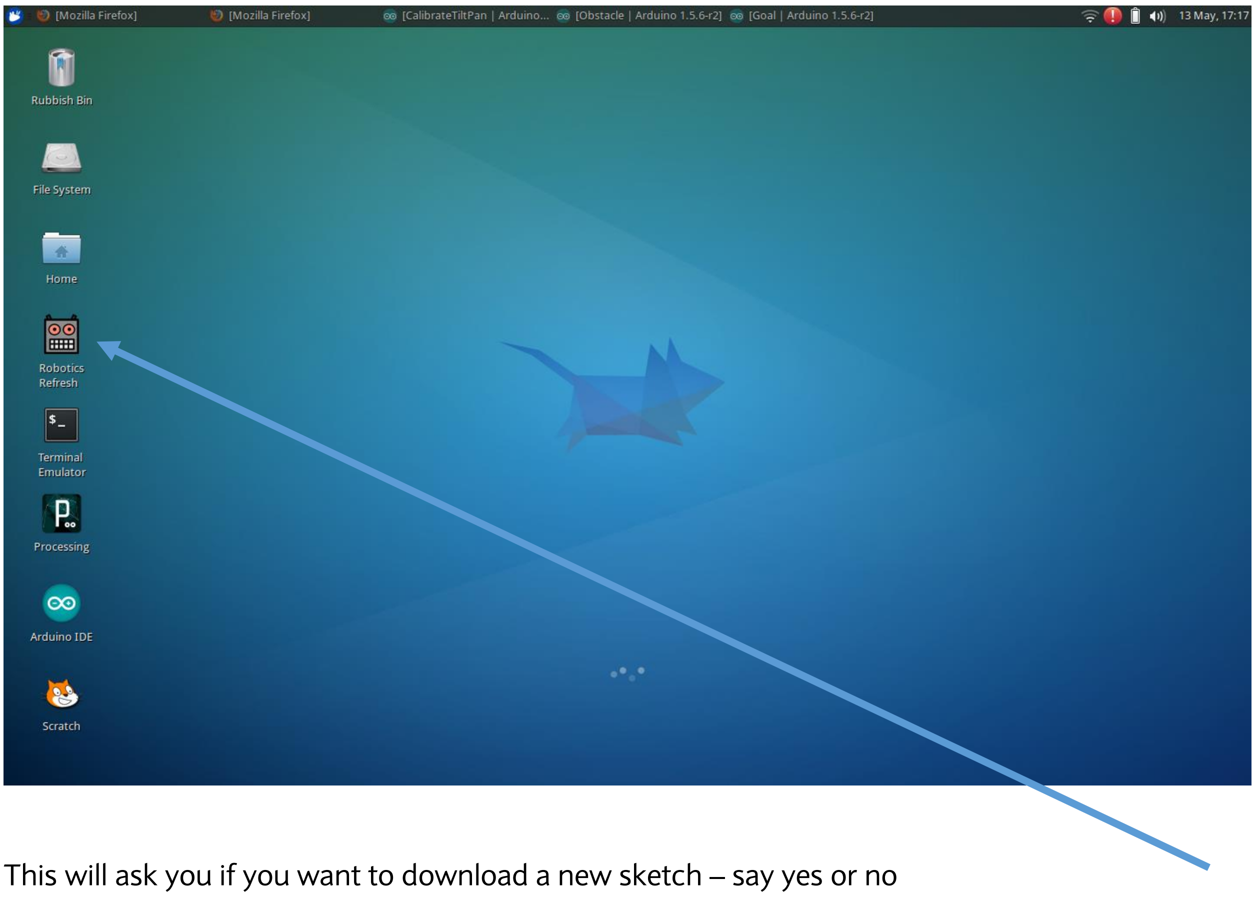
LineFollow – A simple line follower

Obstacle – A simple obstacle avoider

There are also other sketches to test the components – however just use the File -> Open and navigate to the RoboticsChallenge -> sketches folder – there you can download previous sketches and some testing sketches.

Refreshing the sketches between events

To refresh the laptop if sketches have been saved, go to the desktop and click on the icon "Robotics Refresh"



This will ask you if you want to download a new sketch – say yes or no

You will then be asked if you want a local or remote refresh. Remote means that you can download from the master location

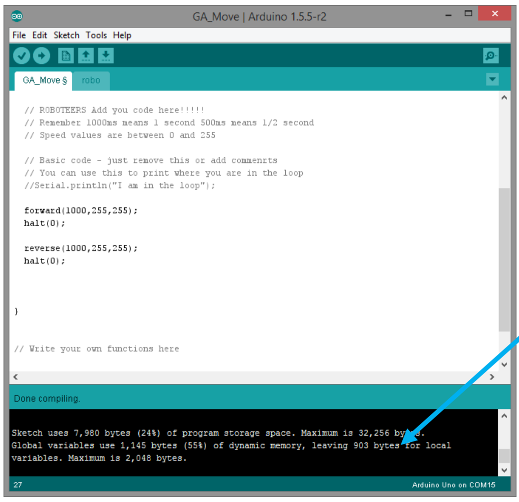
https://github.com/daverobertson63/RoboticsChallenge

If you say local – it will refresh from a master copy. Local is used after a challenge event.

Syntax Errors

We want the students to actually write code. This is an important aspect of the challenge. We don't want them getting bogged down in syntax problems so it's important to be able to explain the basics and point out errors.

If you compile a sketch and there are no errors – then you will see white text with success in the bottom on the Arduino window:

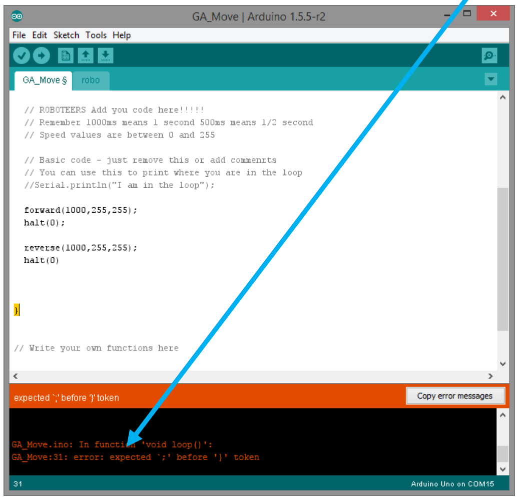


```
  // ROBOTEERS Add you code here!!!!!
  // Remember 1000ms means 1 second 500ms means 1/2 second
  // Speed values are between 0 and 255

  // Basic code - just remove this or add comenrts
  // You can use this to print where you are in the loop
  //Serial.println("I am in the loop");

  forward(1000,255,255);
  halt(0);

  reverse(1000,255,255);
  halt(0);



}


// Write your own functions here
```

Done compiling.

```
Sketch uses 7,980 bytes (24%) of program storage space. Maximum is 32,256 bytes.
Global variables use 1,145 bytes (55%) of dynamic memory, leaving 903 bytes for local
variables. Maximum is 2,048 bytes.
```

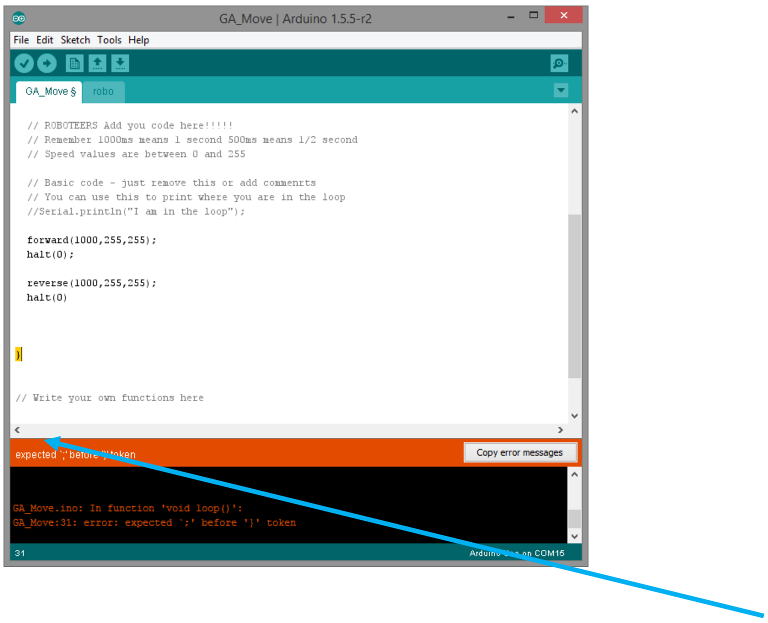
If there are any errors – you will get red text. This means it won't upload and it won't work. You need to fix the error.



```
  // ROBOTEERS Add you code here!!!!!
  // Remember 1000ms means 1 second 500ms means 1/2 second
  // Speed values are between 0 and 255

  // Basic code - just remove this or add comenrts
  // You can use this to print where you are in the loop
  //Serial.println("I am in the loop");

  forward(1000,255,255);
  halt(0);

  reverse(1000,255,255);
  halt(0)



}


// Write your own functions here
```

expected `;' before '}' token

```
GA_Move.ino: In function 'void loop()':
GA_Move:31: error: expected `;' before '}' token
```



```
// ROBOTEERS Add you code here!!!!!
// Remember 1000ms means 1 second 500ms means 1/2 second
// Speed values are between 0 and 255

// Basic code - just remove this or add commenrts
// You can use this to print where you are in the loop
//Serial.println("I am in the loop");

forward(1000,255,255);
halt(0);

reverse(1000,255,255);
halt(0)



}


// Write your own functions here
```

```
expected `;' before '}' token
```

```
GA_Move.ino: In function 'void loop()':
GA_Move:31: error: expected `;' before '}' token
```

Note the line number is on the bottom left of the window (31) this is where your cursor is

In this case – the error message says:

BRC_Move.ino: In function 'void loop()':
BRC_Move:31: error: expected `;' before '}' token

At line 31 there is a missing semicolon ;

Troubleshooting Common Syntax Errors

There are a number of common errors that have come up in coding so please check these first.

1. Missing Semicolons. The semicolon is an end of line marker. You must have one. Always check your semicolons. A semicolon is **not** required at the end of an if statement – that's a case where it will wont give an error but then it wont work

   `forward(1000,255,255);` is correct

   `forward(1000,255,255)` will generate an error

   ```
   if (leftLineSensor()== WHITE )
   {
          forward(1000,255,255);
   }
   ```

   is correct

   ```
   if (leftLineSensor()== WHITE );
   {
          forward(1000,255,255);
   }
   ```

   is incorrect and will not work – it will always go forward

2. **Function Names Spelt Wrong:** Probably the most common error.

   `forward(1000,255,255);` is correct

   `Forward(1000,255,255);` will generate an error

   The error would say something like:

   'Forward' was not declared in this scope

3. **Bracket Matching.** The curly brackets must be equal in all cases. The simple set up is in loop

   ```
   void loop()
   {
   }
   ```

   Where you must have an open and closed bracket. Sometimes the pupils will accidently delete the last bracket. So make sure your bracket count is equal. There is a nice little feature where if you put the cursor over a bracket – the equivalent one, either open or closed will highlight. This can help when you have if statements which can have missing brackets.

   A common error may be something like:

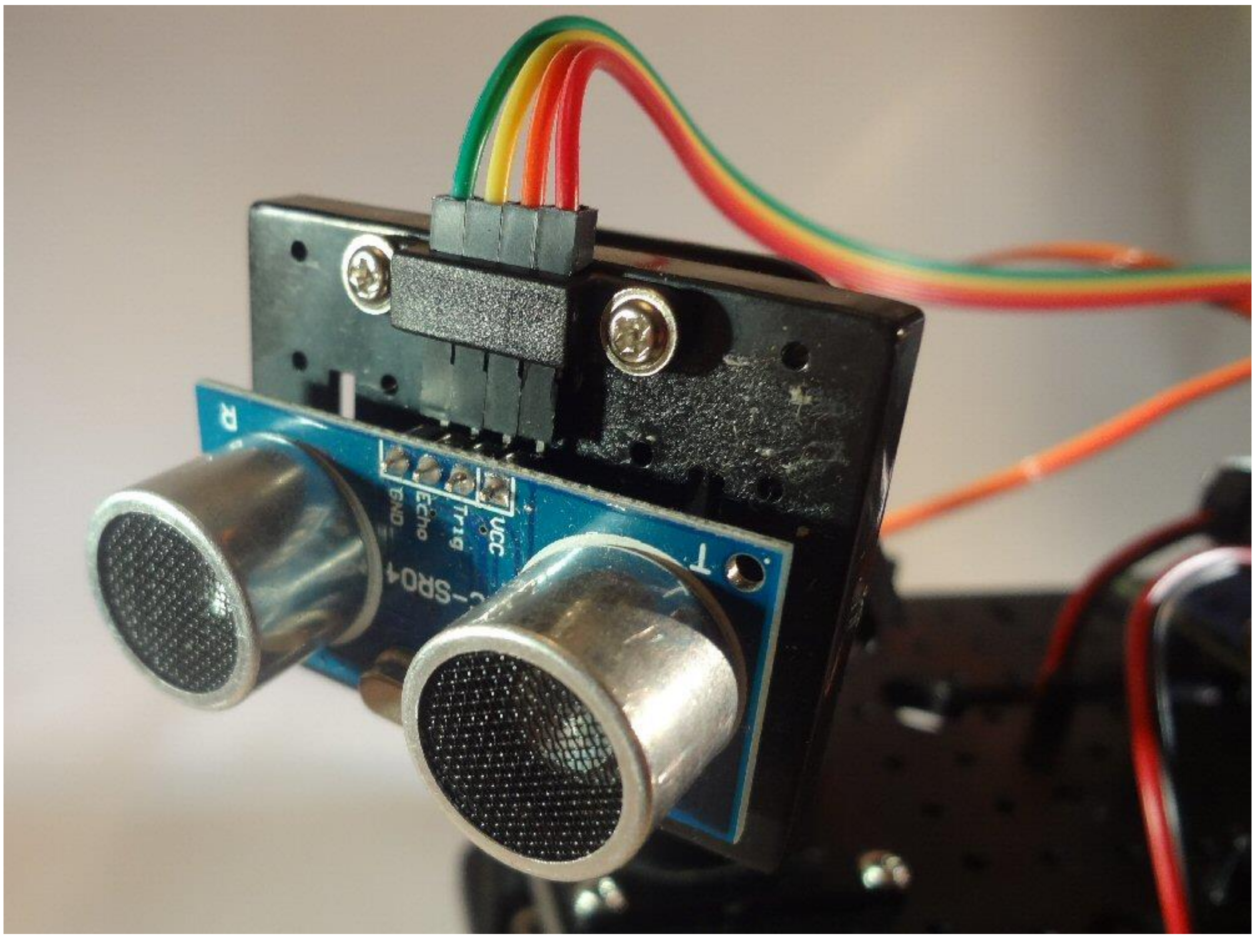
   Expected } at the end of Input or an error that mentions the curly bracket { or }

Tilt and Pan

The tilt and pan device contains the Ultrasonic sensor. It is most useful to find empty space or to look for an object. The Goal challenge has good code on what you can do with it

If you combine this with the "Goal" challenge you can use this mechanism to move the ultrasonic sensor around. Its also handy to test code without moving the actual robot.

You can also just move it – say to nod – which gives it a fun appearance.



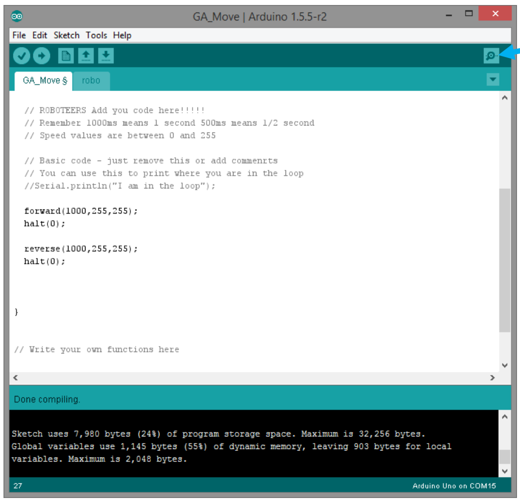
Debugging

Even though Arduino is an 8 bit computer, we can display "live" values of what it's doing on the screen when the robot is connected to the laptop. This is one of the reasons Arduino is a very popular platform.

This allows us to see what the code is doing when it's running (debugging). It is useful for the students to know this early on

To display something in the debug window, you need to use a print command in the code. Most of the built in functions like forward already have a print statement in them so you can see what's going on.

To access the debugger click on the serial monitor icon at the top left of the Arduino IDE window.

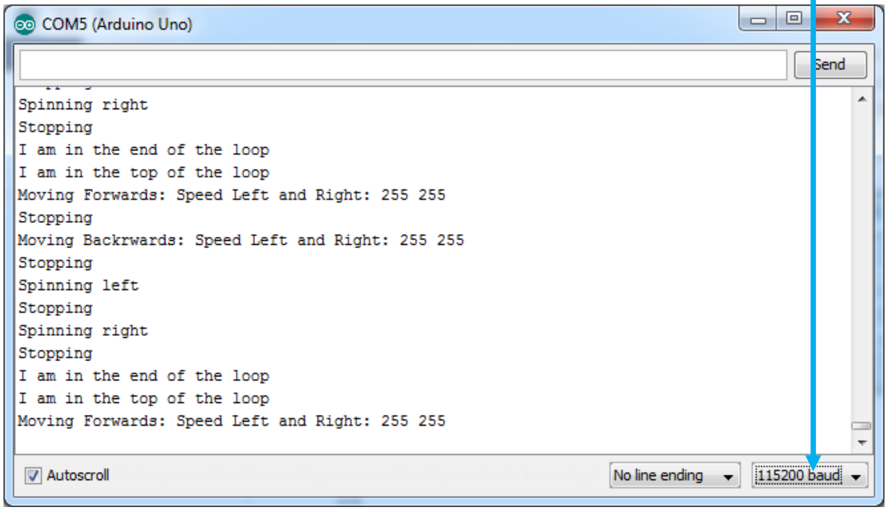


```
  // ROBOTEERS Add you code here!!!!!
  // Remember 1000ms means 1 second 500ms means 1/2 second
  // Speed values are between 0 and 255

  // Basic code - just remove this or add comenrts
  // You can use this to print where you are in the loop
  //Serial.println("I am in the loop");

  forward(1000,255,255);
  halt(0);

  reverse(1000,255,255);
  halt(0);



}


// Write your own functions here
```

```
Sketch uses 7,980 bytes (24%) of program storage space. Maximum is 32,256 bytes.
Global variables use 1,145 bytes (55%) of dynamic memory, leaving 903 bytes for local
variables. Maximum is 2,048 bytes.
```

This will display a debugger window. Set the baud rate to 115200 if it's not already set and when you run a sketch you will see the code running live.

Each sketch already contains a Serial.println("Example Text");



```
Spinning right
Stopping
I am in the end of the loop
I am in the top of the loop
Moving Forwards: Speed Left and Right: 255 255
Stopping
Moving Backrwards: Speed Left and Right: 255 255
Stopping
Spinning left
Stopping
Spinning right
Stopping
I am in the end of the loop
I am in the top of the loop
Moving Forwards: Speed Left and Right: 255 255
```

This is the debug window showing the code as it executing!

Challenge 1 - Move the robot

Time – 20 minutes

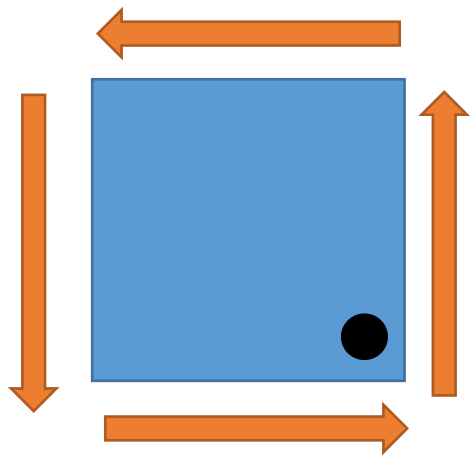
1 sheet flip chart paper, robot, laptop, Move sketch

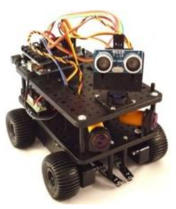
Objectives

1. To understand sequential code and how to use the basic functions.
2. To use the minimum number of lines of code for the job by understanding the concept of a loop.

Setup

Using a flip chart pad sheet – create a square. Fold or cut the paper to form a square. Put this on the ground. Mark one corner as the start. Using the "Move" sketch – make the robot go round the outside of the paper. Back wheels start at the corner marker.





Code Challenge:

Code provided uses four functions – forward, reverse, left spin and right spin. Each function can make the robot move by left and right motor speeds for a time in milliseconds. The maximum speed is 255 and the minimum is about 120. Halt is used to stop the motors.

```
forward(delay,left,right);
```

So;

```
forward(1000,255,255);
```

Will make the robot go forward for 1 second at maximum speed.

Use all the functions to perform the move in the shortest possible time.

Do it in reverse. How many lines of code did you use ? Can you do it in just two lines ?

Extra Challenges:

- Make the robot do a circle. Hint, you can use the different motor speeds. Try this on a smooth surface.
- Make the robot do a figure of eight.
- Write your own function such as a full square, by combining commands and using

  ```
  void yourFunction() { }
  ```

  which is included at the end of each sketch.

Challenge 2 – Make the Robot Avoid Obstacles

Time: 40 – 60 minutes

Resources: Giant Jenga blocks – robot, laptop, ‘Obstacle’ Sketch

Objectives

1. Understand the concept of the “if” statement and use that in code to make the robot make a decision when it sees an obstacle.
2. Test the effect of code changes and practice changing the code to get a better result

Setup

The setup must include an obstacle course which will be the Giant Jenga blocks.

Load the **Obstacle** sketch to start.

In this challenge we want to make the robot avoid obstacles using the infrared sensors. There are two sensors; left and right and this can help us decide which direction to move in order to avoid an obstacle.

Code Challenge:

1. The code given is an empty if block which has both obstacle sensors set to false. A simple forward inserted will allow the robot to move when nothing is in front. The first challenge is to cut and paste the existing block or change the values to true – to allow the sensors to work when something is in front. Code for making the robot go forward could be inserted into the main loop as well – rather than within an if block.
2. The next challenge to make the robot avoid a vertical Jenga brick. Both sensors must be used. If the left sensor is activated then the robot must take action to avoid the obstacle (move right or reverse, etc.). If the right sensor is activated then robot must take slightly different evasive action.
3. The final obstacle challenge is to make the robot go through an obstacle course. The obstacle course will be set up beforehand with Jenga bricks (see obstacle course setup sheet).
4. Use && for AND, || for OR operations. The “if” statements can be combined or just left as single if commands. The following is a single if statement for just one sensor value:

```
if ( leftObstacleSensor() == true )
{
     reverse(250,255,255);
}
```

Or if you want to check the value of both sensors at once:

```
if ( leftObstacleSensor() == false && rightObstacleSensor() == true )
{
     leftSpin(100,255;
}
```

Extra Challenges:

- The corner:
  For students who want to take it further the corner is a great challenge. They will need to use a variable to count how many times the robot gets stuck.

Setup a counter as an int variable in the global area to record how many times both sensors are activated. After say 5 times, the robot must execute a reverse out. The challenge is to make the robot get out of the corner and go back the way it came!

There will be a corner set up for students who want to tackle this challenge. (See obstacle course setup sheet.)

Challenge 3 – Goal – score a goal

Time: 40 – 60 Minutes

Blow up goal posts and ball. Robot, laptop, Goal

Objectives

1. Understanding of the use of variables and taking action based on the value of the variable generated by the Ultrasonic sensor.

Setup:

The idea is to get the ball in the net. It's not feasible with the single sensor to be able to find the ball – but if you start from a known position you can use the ultrasonic sensor to find the ball – and then take action – shoot.

Code Challenge:

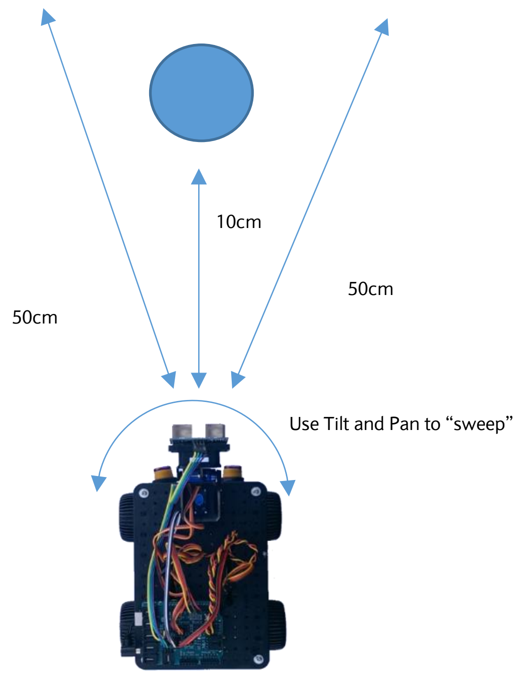
1. The initial code is quite simple although there are some additional functions that need to be moved around. The given if block needs to be modified but it shows how you could get three different distances and compare them to make a decision.

```
if ( leftDistance <= midDistance && leftDistance <= rightDistance )
  {
    //Shoot!

  }
```

   The code also shows you how to get a value from the ultrasonic sensor and store it in a variable.

```
pointLeft();
leftDistance = Ultrasonic();
```

2. Now you can look at the different values you can get from the Ultrasonic sensors. So if you set the student a starting position and have the ball in say a forward position – you can program the robot to move forward – point to the left and right and see if it can detect an object. As you can see from the diagram – the distance from three different readings will show where the ball is



So if the code has 3 reads and sets each variable for example:

```
pointLeft();
leftDistance = Ultrasonic();
pointCentre();
midDistance = Ultrasonic();
pointRight();
rightDistance = Ultrasonic();
```

It's a case of setting up the goal and getting the students to think about how they can solve the problem.

You can also make this into a simple "Follow Me" sketch – by using the same idea of sweeping the sensor to look for objects. When it finds something it can turn the robot to move to the smallest distance.

Extras: Writing your own functions

We have supplied an additional function called “stupidCelebration” This shows how you can combine commands and create your own function.

This is more of an extended programming effort – use this for students who clearly want to understand programming more.

Challenge 4 – Line Follower

Time: 40 – 60 Minutes

Line follower tracks – robot, laptop, LineFollow

Objectives

1. Understand how to use “if”
2. Optimise turns using spin or forward with different left and right values.

Setup:

The line follower is a popular challenge and can become quite competitive - we will be making it so! The line follower tracks will be already laid out and ready to use. Open the LineFollow sketch and proceed to the code challenge.

Code Challenge:

The code supplied in the sketch is quite simple – the supplied if block is for when the two sensors are seeing a black value. Insert a forward statement to make the robot go forward – this will be the first step.

1. Compile the sketch and make it work on the track to check that everything is working. If both sensors are on the black line – after the insertion of forward. A value of 100ms is a good starting value.
2. Now modify the code to be able to move the robot back onto the black line if it goes off. Hint – work out the number of different scenarios you could have for the sensors. Both black is one scenario. Write them down if it helps and work out how the robot should move in the other scenarios.
3. Hint! You can use `leftSpin(100,200)` or you could use `forward(100,200,255)` say to move the robot when it comes off the line. `leftSpin` does a spin turn by doing a full reversal – this is called a slip turn. `forward` does a differential turn – like a tank.   It’s up to you – experiment…
4. Now, time your robot round the track – we are keeping a top gear style “fastest Barclays roboteer in a reasonably priced robot”. The leaderboard awaits!
5. The values of the time can be varied to optimised the speed of the robot round the track – and you can encourage the students to think about that.

Extras:

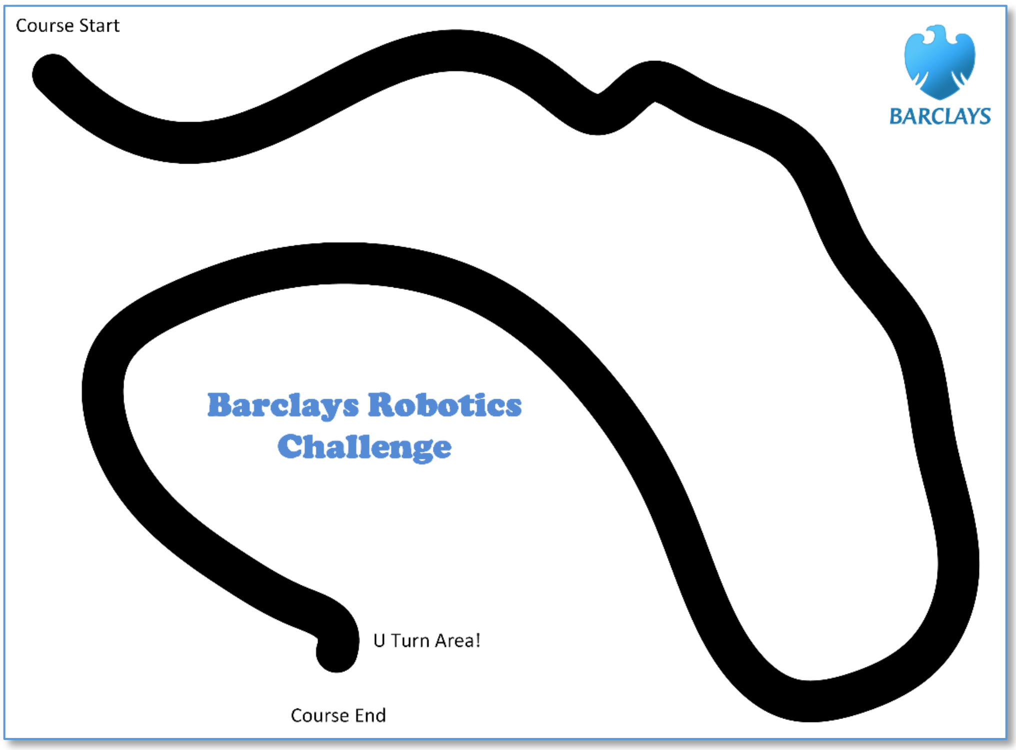
Line following is a great robotics challenge and there are lots of opportunities to optimise the way the robot will follow the line so this could well take up the whole challenge.   If you have a student who wants that extra challenge then look at going back round the track, starting with a U turn. The track has it marked.

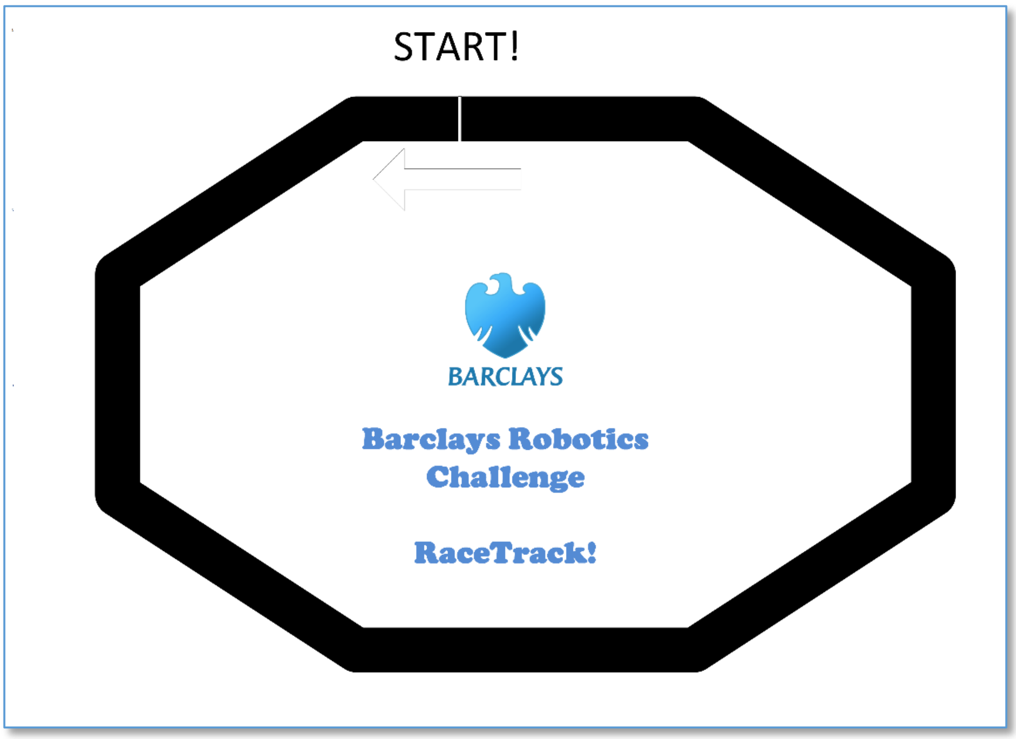
Use a simple counter to detect when the robot finds white space say, 5 times in a row (both sensors detect white, robot reverses, both find black, robot goes forward, both find white… etc.). There is a counter employed in the supplied code as a hint!

Line Follower Tracks

There are two tracks available. There is a leaderboard available here.

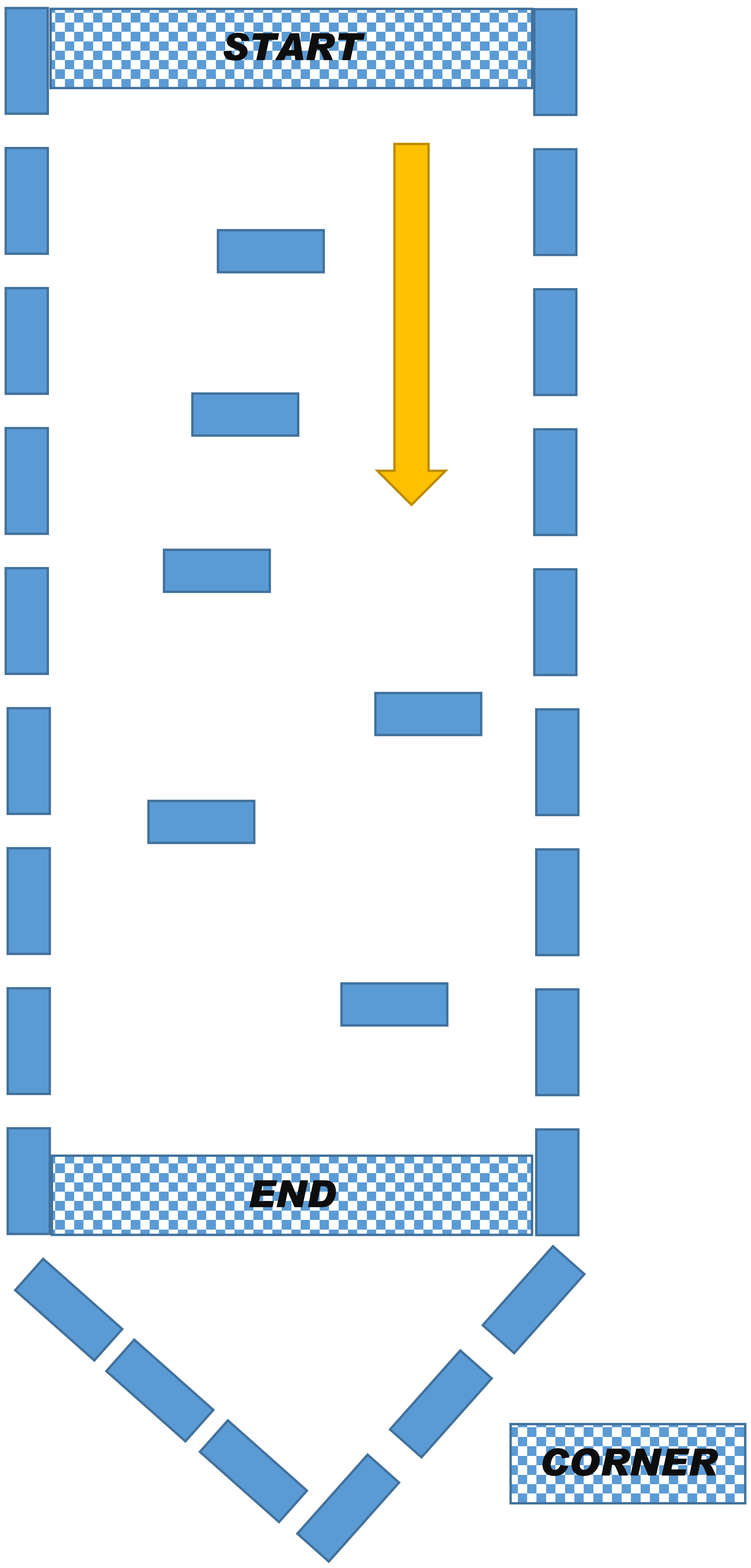
http://barclaysapps.wordpress.com/2014/08/24/fastest-barclays-roboteer-in-a-reasonably-priced-robot/





Obstacle Course Setup

The arena format allows robots to start at one end and work their way through. Mark the start and end areas with tape. The robot has completed the challenge when it goes through the course. Multiple robots will work as well and add to the fun!


START
END
CORNER