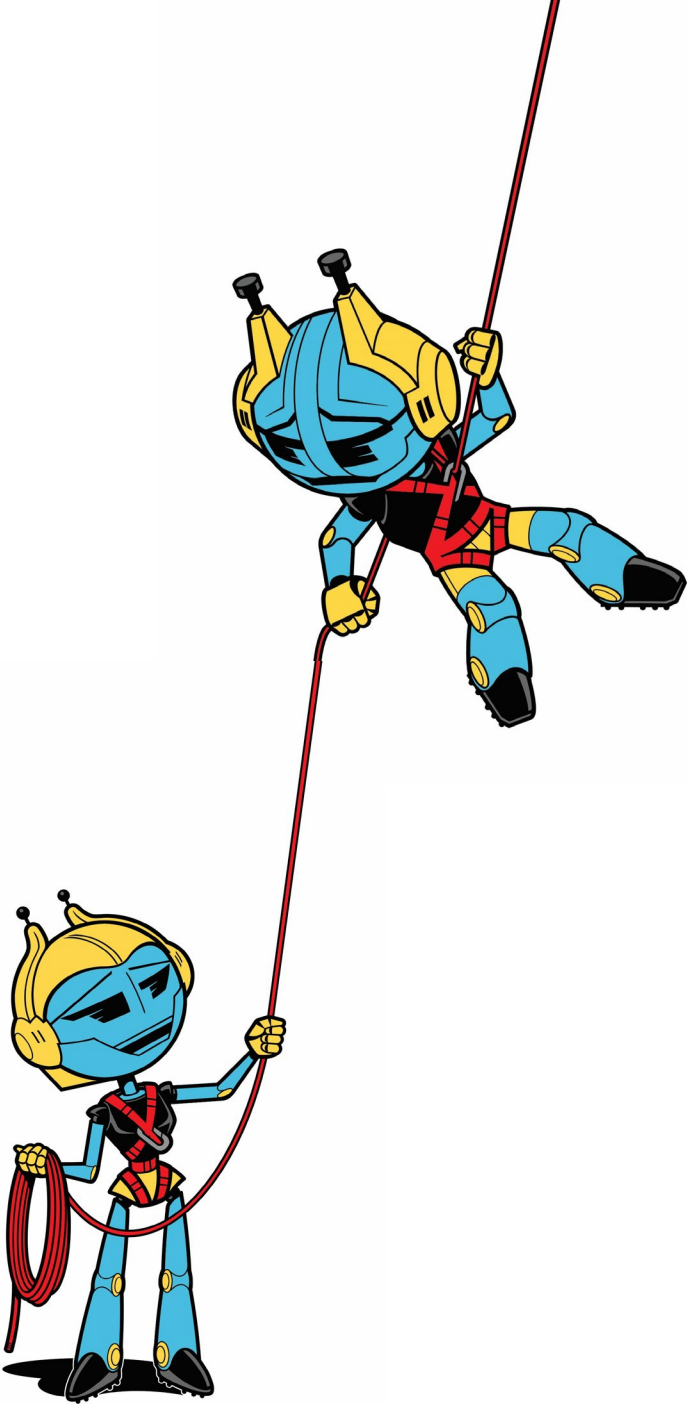




Riley Rover Handbook

A beginner's guide to getting started with the
RoboCup Junior Rescue Line competition

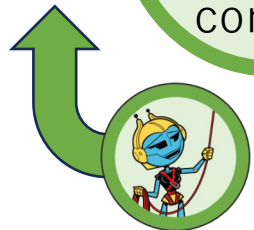


Hey there! This presentation will take you through all the basics of getting set up for Riley Rover Rescue. It makes no assumptions about your level of expertise, so you can move through it as you please.

The next slide is the index slide and has links to all the topics covered. Click on any one to take you to that topic. When I appear at the bottom with an arrow to the right, this means you can advance to the next slide, when ready. When you complete the section, you should be returned automatically to the list of topics. If you want to skip back to the topic list at any point. Just click on my friend on the lower left of your screen.

Slides do have animations. If viewing this as a pdf, you will miss the animations and automatic return to the index slide, but all hyperlinks will work.

Best of luck and remember you can always reach out to your local RCJA committee for help.



What is in this handbook?

Introduction to Rescue Line



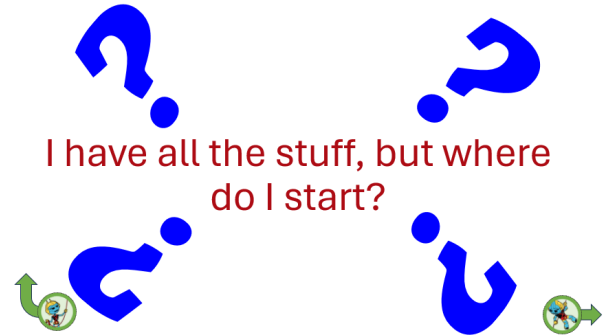
What is the Best Robotic System?

RoboCup Junior is a “platform agnostic” competition

- There is no specified robotic system.
- The rules state that:

“3.1.4 Any robot kit or building materials may be used, as long as the robot fits the specifications documented in these rules and as long as the design and construction are primarily and substantially the original work of the student(s). Commercial robot kits may be used but must be substantially modified by the student participants.”

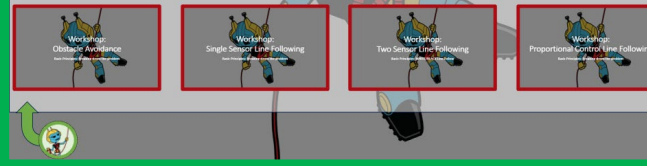
- The Lego Spike Prime system is a relatively easy entry point and can be adapted readily to other Rescue Line divisions and RCJA competition challenges.
- This is also true for the Lego EV3 robotic system, but this is no longer commercially available.



Building a Rescue Line Robot



Breaking Down the Problem and Coding the Robot Examples



Problem Solving: Some Simple Checks



Introduction to Rescue Line



The Scenario

Riley Rover Rescue

The Riley Rover Rescue division has been developed as an easier entry point for students and mentors new to RoboCup Junior.

It includes the key elements of all the Rescue Line divisions:

- Line following
- Object avoidance
- Rescue
- Movement up shallow inclines

Without the more difficult challenges:

- Intersection navigation
- Movement over uneven surfaces
- Movement across line gaps
- Movement up steep inclines

It is important that mentors and students have a good understanding of the [Rescue Line rules](#) as posted on the RCJA website.



A terrible disaster has hit the city and caused a large chemical storage unit to rupture spilling thousands of litres of toxic chemicals in the centre of the city.

There is a Victim trapped in a sinking Rescue Capsule in the chemical spill. Rescue crews are having trouble entering the city, with the amount of rubble around. Rescue from the air has also been ruled out due to the noxious gases rising from the toxic chemicals.

It has been decided that the best form of rescue is the deployment of an autonomous robot that can navigate to the chemical spill, rescue the victim, and exit the chemical spill.

Your team has been asked to design, make, and appraise a robot to complete this challenge.



What is the Best Robotic System?

RoboCup Junior is a “platform agnostic” competition

- There is no specified robotic system.
- The rules state that:

“3.1.4 Any robot kit or building materials may be used, as long as the robot fits the specifications documented in these rules and as long as the design and construction are primarily and substantially the original work of the student(s). Commercial robot kits may be used but must be substantially modified by the student participants.”

- The Lego Spike Prime system is a relatively easy entry point and can be adapted readily to other Rescue Line divisions and RCJA competition challenges.
- This is also true for the Lego EV3 robotic system, but this is no longer commercially available.



Introducing Spike Prime

- The basic Spike Prime set has everything you need to get started with Riley Rover Rescue
- It is also recommended that you get:
 - An extra colour sensor, either as a separate item or as part of the expansion set
 - A Rotacaster omniwheel or equivalent
 - Rescue tiles
- All items are available from [Modern Teaching Aids](#) (RCJA Platinum National Sponsor):
 - The extra colour sensor and omniwheel are not necessary for Riley Rover Rescue but will be a benefit, and are necessary for other Rescue Line divisions
 - The small Rescue tiles are available from MTA but can also be printed on A3 paper, cut to size and fixed to a solid support (e.g. a hard floor). Printable files are available on the [Rescue Line home page](#) on the RCJA website.



LEGO® Education SPIKE™ Prime Set (LEG45678)

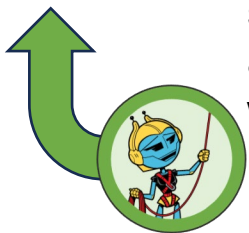


**Rotacaster
omniwheel
(ROT5435)**



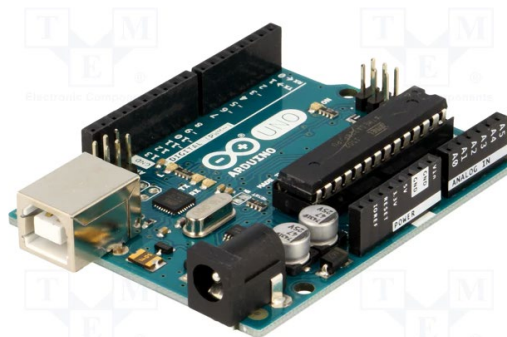
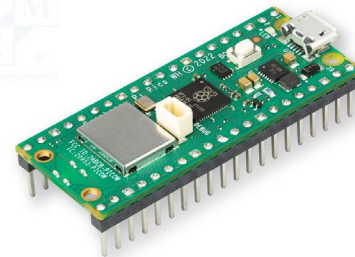
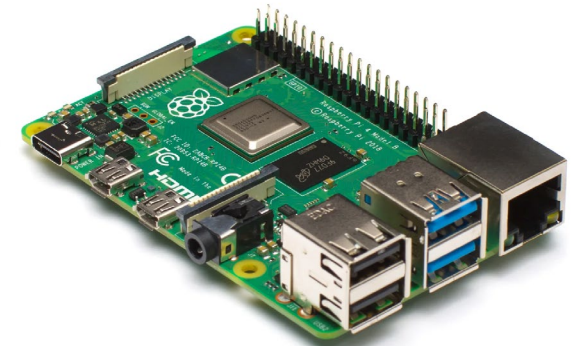
Practice tiles

- [Large tile vinyl mat \(ROB0009\)](#)
- [Large individual tiles \(RCJA\)](#)
- [Small individual tiles \(ROB016\)](#)

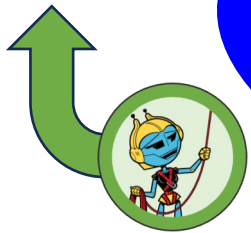


Custom Robots

- Another purpose of the Riley Rover Rescue division is to give students a place to try something new.
- This will not be covered in this handbook, but students who have competed before may use Riley Rover to try out new ideas using custom built robots based on other controllers, such as:
 - Arduino
 - Raspberry Pi
 - Raspberry Pi Pico



I have all the stuff, but where
do I start?

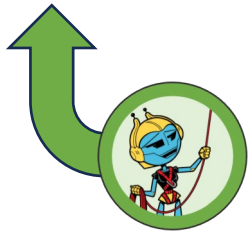




Hey there! In this presentation I'll take you through the process of breaking down the Rescue Challenge into manageable parts.

All you'll need to do is click to go to the next slide and I'll pop up along the way. You'll see me appear at the bottom with an arrow, this means you can advance to the next slide, when ready. If at any time you want to return to the index slide, click on my friend to the left.

Best of luck and remember you can always reach out to your local RCJA committee for help.

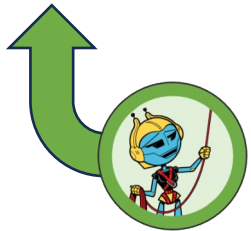


Approaching the Challenge



Let's have a look at the basic components of the challenge:

- Follow a line
- Go up/down an incline
- Detect an obstacle
- Go around an obstacle
- Detect the entrance to the spill tile
- Find the rescue capsule
- Move the rescue capsule out of the spill



Breaking It Down

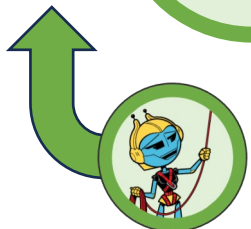


This may look like a lot to work through, but really it can be broken down into two simpler problems:

- **Problem 1** - Detecting and responding to changes in reflected light intensity/colour.
- **Problem 2** - Detecting and responding to an object.

With this we have the basis for creating two much simpler challenges, both of which will help develop skills for solving the more complex problem.

- **Challenge 1** – Program a robot to stay inside an area marked by a black line (electrical tape on the floor is good for this).
- **Challenge 2** - Program a robot to avoid objects placed randomly on the floor.



Breaking It Down



You may very well say at this point, “Those might seem like simpler problems, but I still am not sure how to even build the robot.”

Consider what you need your robot to do:

- Move and turn (two motors connected to two wheels with a third point of contact for balance)
- See black and white on a horizontal surface (light sensor pointing down)
- Detect object (ultrasonic sensor pointing forward)

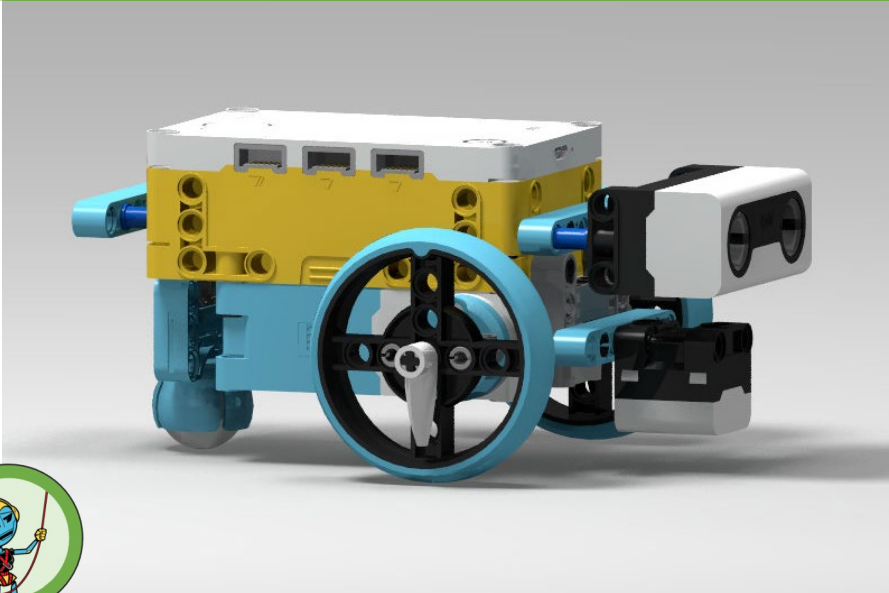
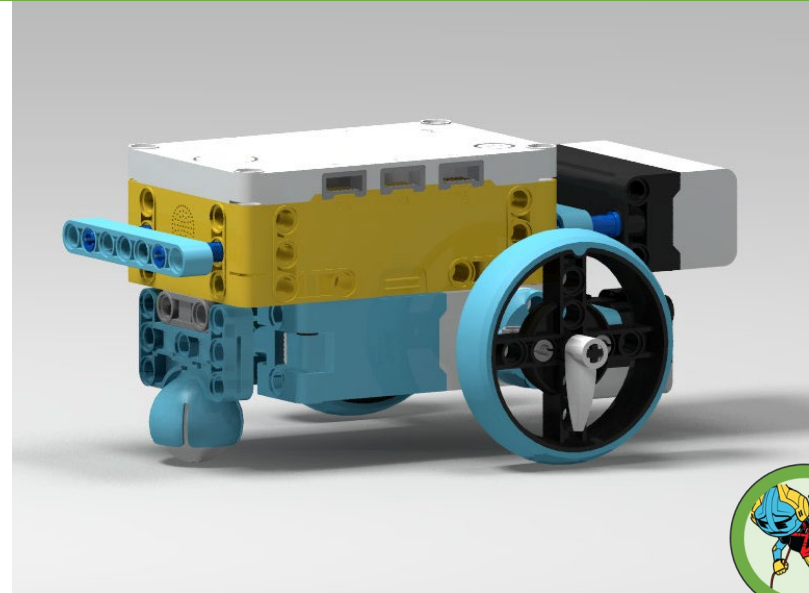


A Very Basic Robot



Let's build a robot to tackle the simple problems before moving on to the more complex. Here are images from two angles of a basic robot (MTA-Bot) with all the properties you are looking for.

[Full build instructions](#) can be found on the Modern Teaching Aids website.

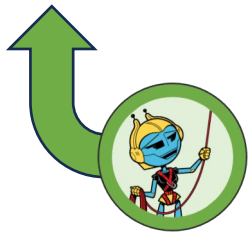


The Most Basic of Robots



The images are of a 'virtual' build using CAD software, so no cables are shown, but you probably can see how easy it is to put together a simple robot.

Will this be a perfect Riley Rover Rescue robot? Not likely, but there are more tips for that when you are ready for more complexity.



Solving a Simple Challenge

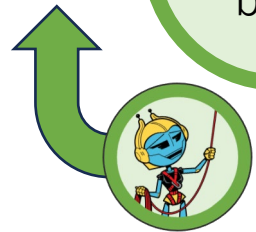
– stay inside an area defined by black tape



Now that we have a robot, let's get started on some code. For beginners, having some code on which to build is a very powerful way to start the learning journey.

For this exercise I will give you some simple code. This will need to be modified for your robot build.

Note the use of comments to explain the code and the parameters that can be modified.



```
when program starts
  set movement motors to B+F
  set movement speed to 30 %
  forever loop
    if A is colour black then
      move up for 2 rotations
      move right: 100 for 2 rotations
    else
      start moving down
```

Initialise movement motors. The port designation will depend on which ports the motors are attached to.

Initialise movement speed. This value can be any value up to 100. At very low speeds the motors may not be able to overcome friction.

Check if colour sensor sees black. If true, move back, then turn right. The user needs to check that colour sensor is seeing the border as black. The number of rotations required should be determined by the user.

If black is not detected, move forward. Note that the movement direction is determined by the orientation of the motors, not by the direction the arrow is pointing in the code. This needs to be tested by the user.



Solving a Simple Challenge

– avoiding objects



The second code is aimed to solve the problem of avoiding objects.
Note how similar the codes are.

The code blocks are as follows:

- when program starts** (yellow block)
- set movement motors to B+F** (pink block)
- set movement speed to 30 %** (pink block)
- forever** loop (orange block) containing:
 - if** block: **E** is **closer than** **10 cm** ? **then** (blue block)
 - move** **up** for **2** **rotations** (pink block)
 - move** **right: 100** for **2** **rotations** (pink block)
 - else** (orange block)
 - start moving** **down** (pink block)

Callout boxes provide additional context:

- Initialise movement motors.** The port designation will depend on which ports the motors are attached to.
- Initialise movement speed.** This value can be any value up to 100. At very low speeds the motors may not be able to overcome friction.
- Check that distance sensor detects an object within 10 cm.** If true, move back, then turn right. The number of rotations required should be determined by user.
- If no object is detected, move forward.** Note that the forward direction is determined by the orientation of the motors and needs to be tested by the user.



Extending the Simple Challenges



Once the simple codes are working, try modifying the code by changing movement and sensor parameters, or the build of the robot.

For Challenge 1, what happens if you use green tape instead of black?

For Challenge 2, what is the min and max distance detected by the distance sensor? Can you use the force (touch) sensor instead of the distance sensor?

Why not combine the challenges? Program a robot to avoid objects while staying inside the area marked by the black line.

Maybe modify the challenge by having the robot detect an object and push it out of the circle, while not exiting the circle itself.



Now That You Have Gotten a Start ...

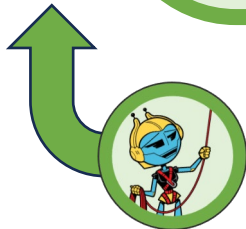


The main aim of these exercises were to get you started. You now have a very simple robot and basic code for detecting a black line and avoiding objects.

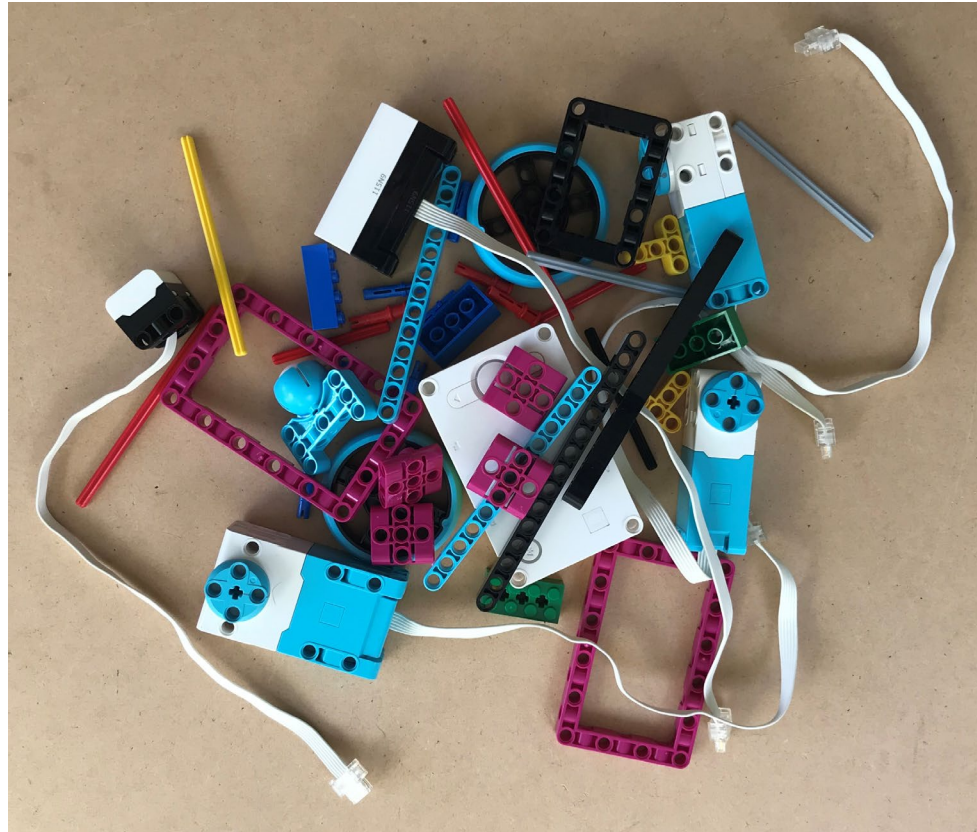
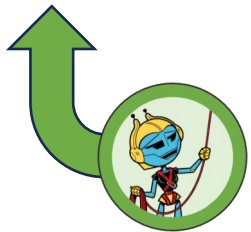
That isn't a rescue robot by any means, but it is a good start.

Later in this handbook you will be given some tips about things that need to be considered when building a rescue robot and guidance for developing code to manage the challenges that your robot will face on the Rescue course.

In a classroom or club setting, these simple challenges are good ways to get students motivated by experiencing success. Depending on the experience of your students, you may decide to give them more or less information to begin. Do encourage them to explore what happens when they modify the code or the build.



Building a Recue Line Robot

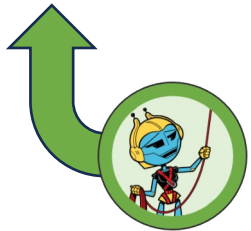




Hey there! In this presentation I'll take you through the process of breaking down the Rescue Challenge into manageable parts.

All you'll need to do is click to go to the next slide and I'll pop up along the way. You'll see me appear at the bottom with an arrow, this means you can advance to the next slide, when ready. If at any time you want to return to the index slide, click on my friend to the left.

Best of luck and remember you can always reach out to your local RCJA committee for help.



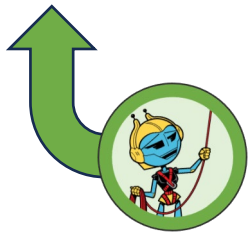


Let's go through some of the questions we need to ask when designing a Rescue robot:

- Is there a limit to robot size?
- What drives the robot forward, backward, right and left?
- How do we ensure the robot is stable?
- What inputs (sensors) will the robot require?
- Where will the sensors need to be placed?

Two important thing to remember from the very start:

1. Your first design is unlikely to be your last design.
2. Changes to design will very likely require changes to your code and *vice versa*.





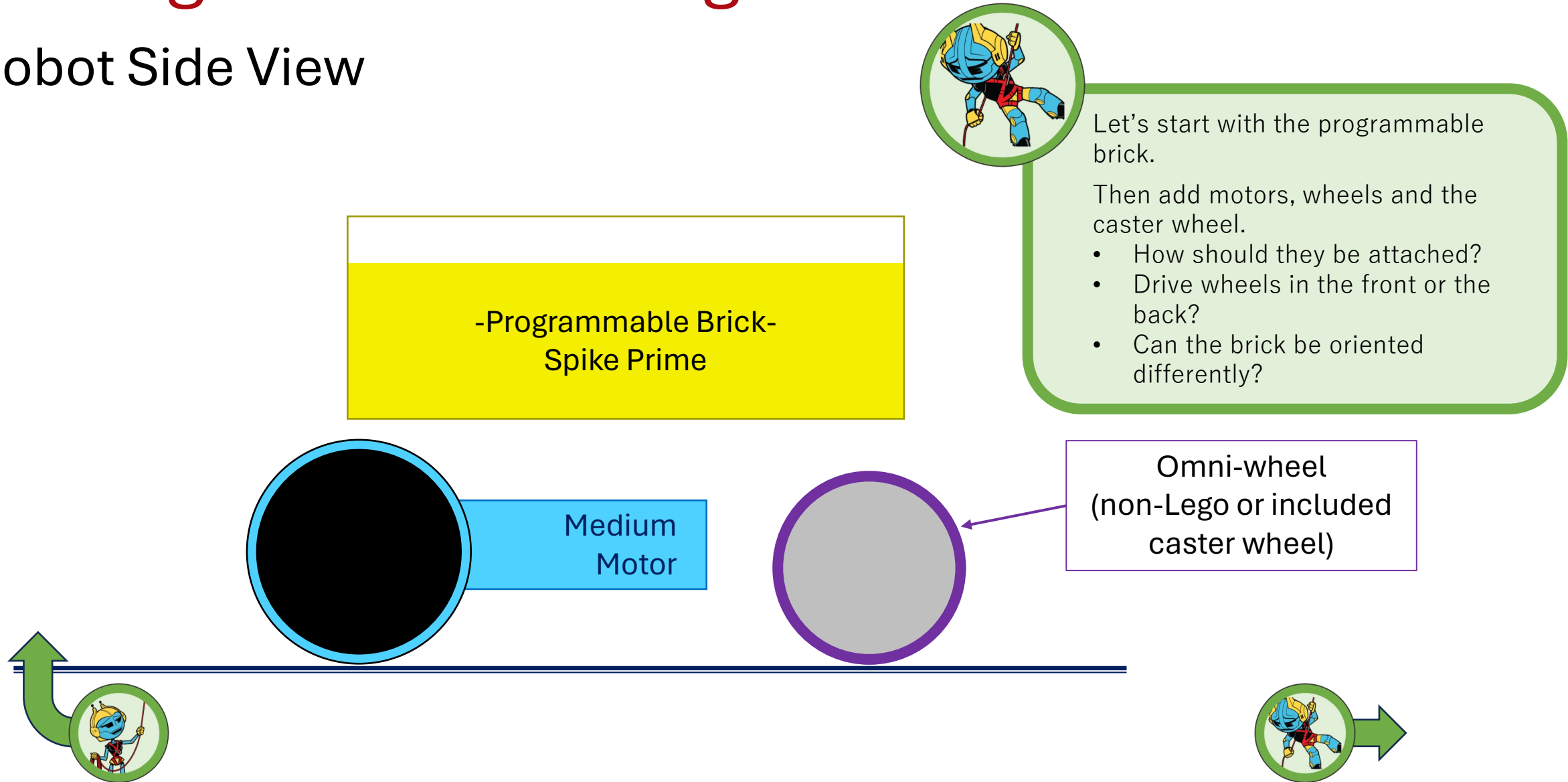
The next set of slides will go through design considerations using schematic diagrams.

This is NOT a blueprint for a robot design. The beauty of Lego is there are so many options to make your robot unique.



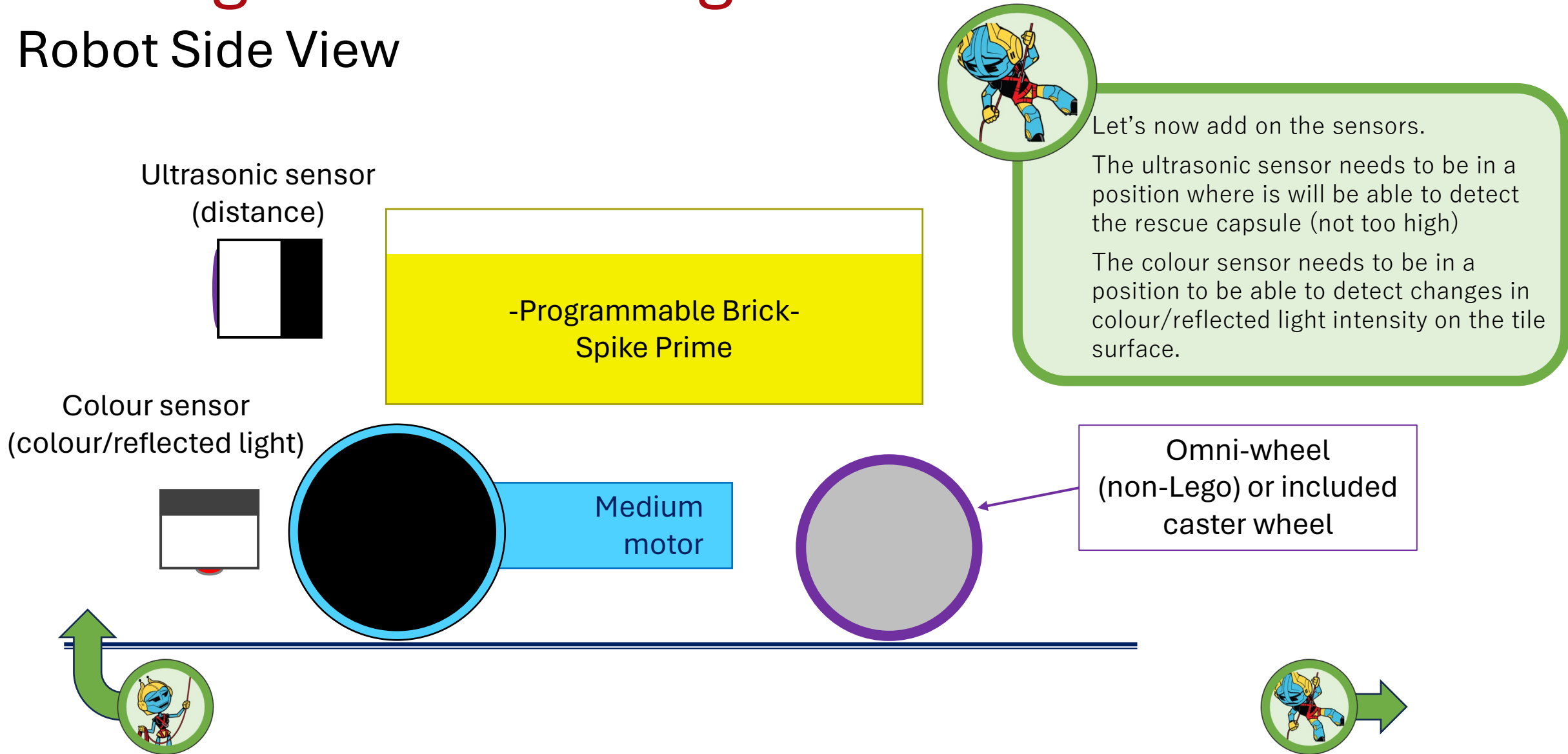
Building a Robot – Design Considerations

Robot Side View



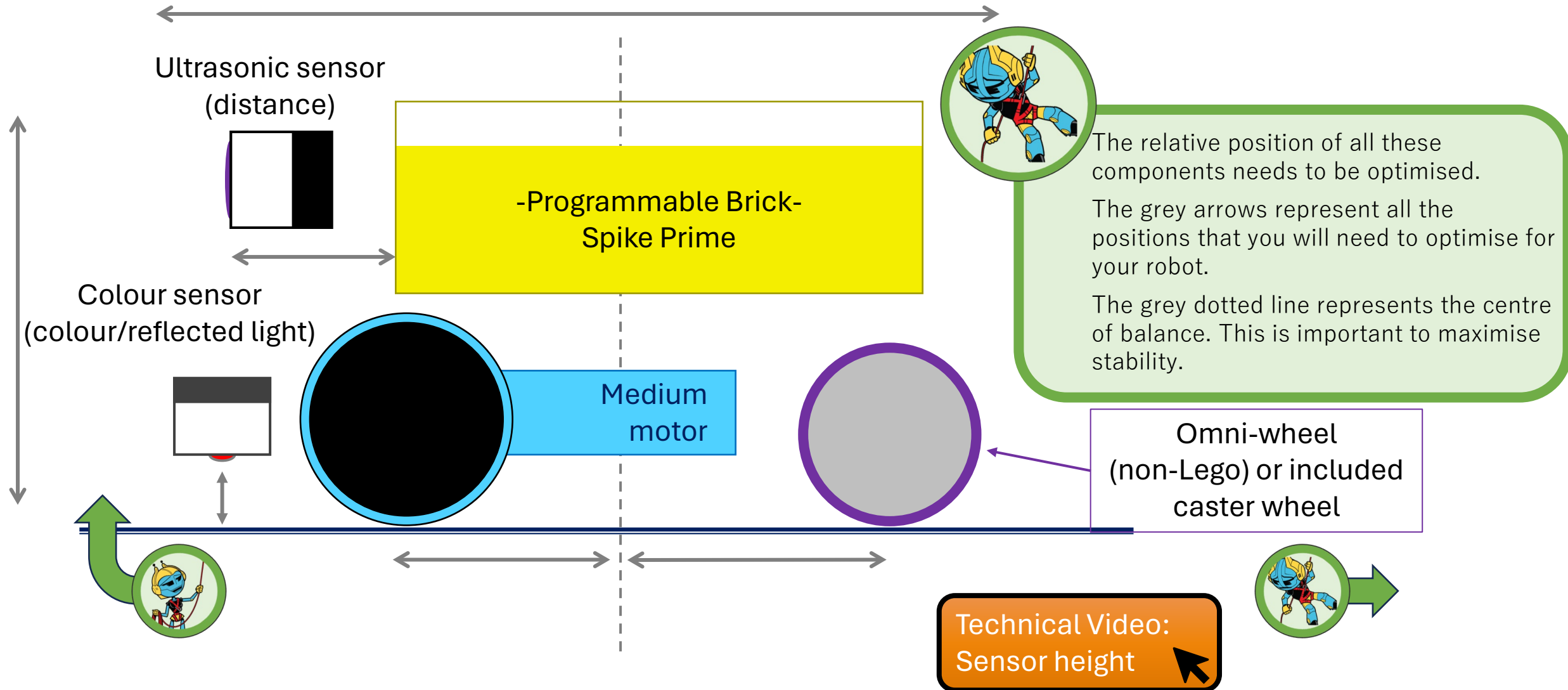
Building a robot – design considerations

Robot Side View

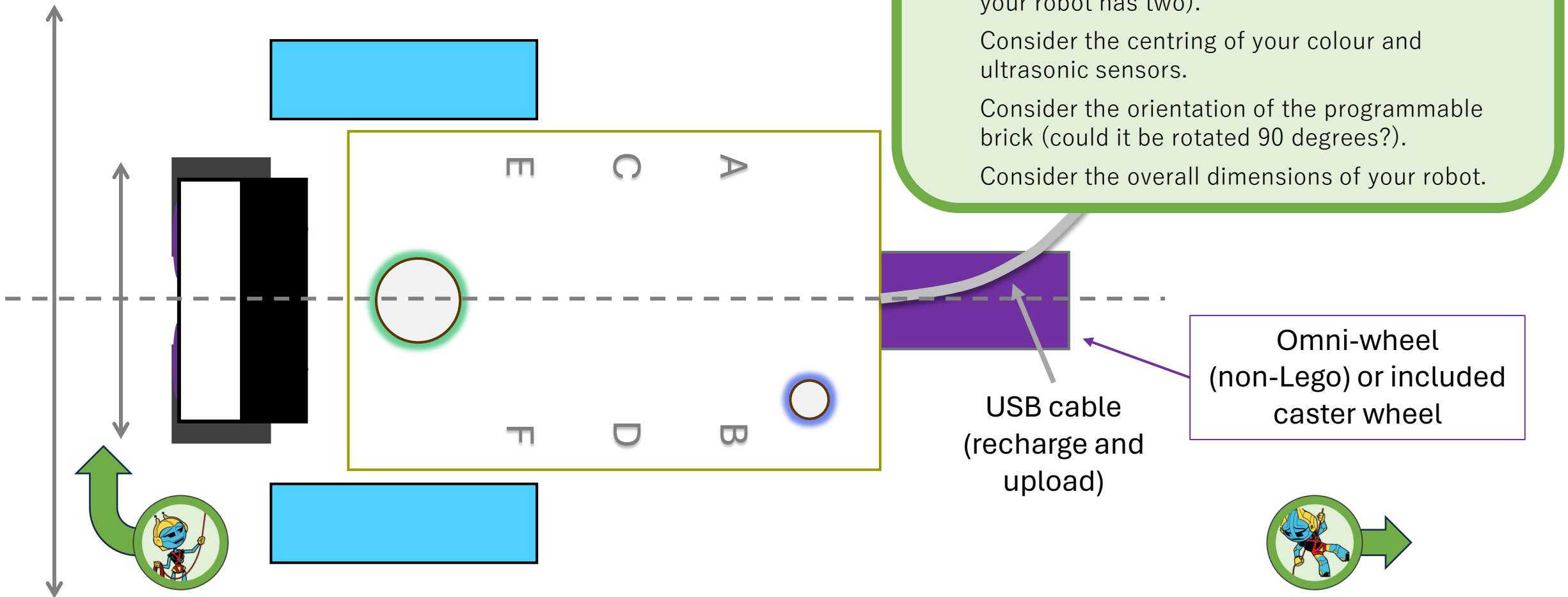


Building a robot – design considerations

Robot Side View

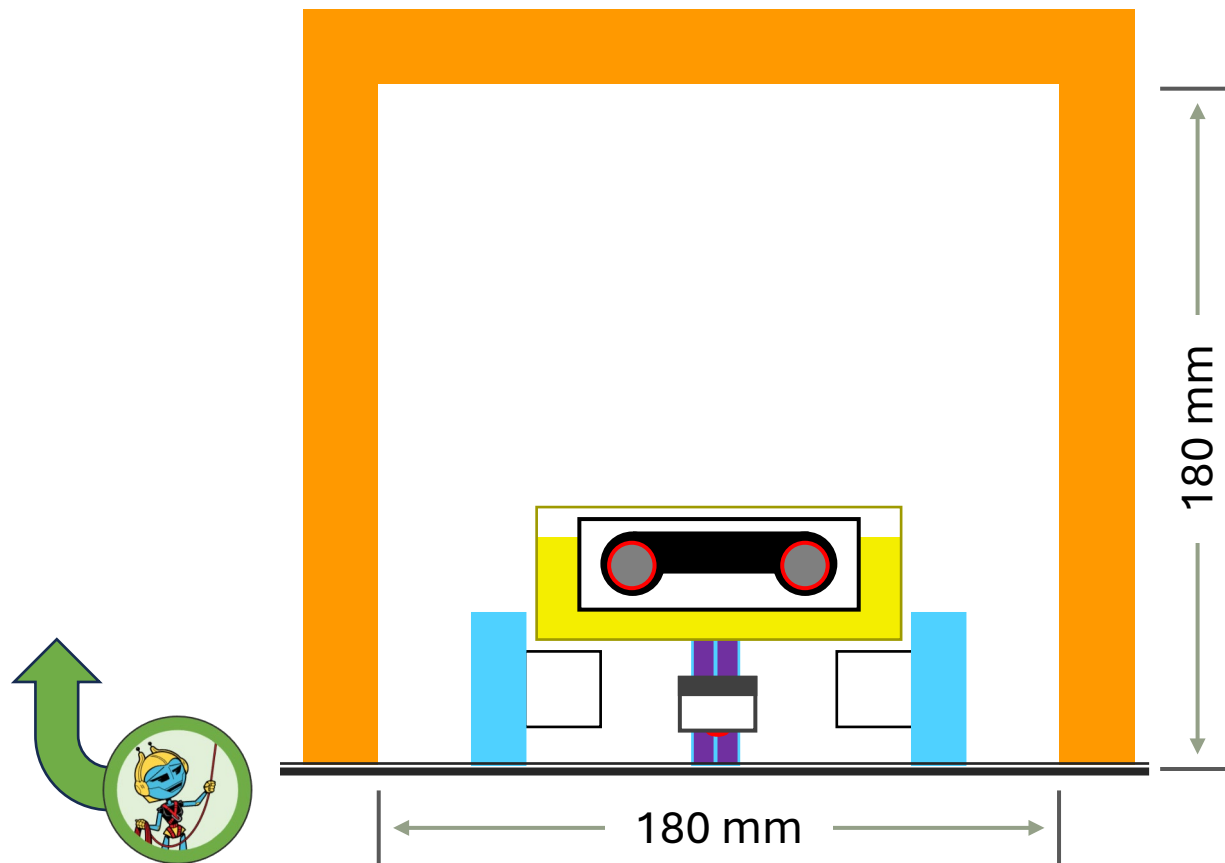


Robot Top View



Physical Challenges

Fitting through the gate



Your robot will face many challenges on the way to the rescue. What could possibly go wrong?

- Robot is too wide or moves off the line and knocks over the gate.

The rules state that:

If a doorway is present on the course, the robot must have passed through it before the score for the round will be recorded.



Physical Challenges

The Bridge and Elevation tiles

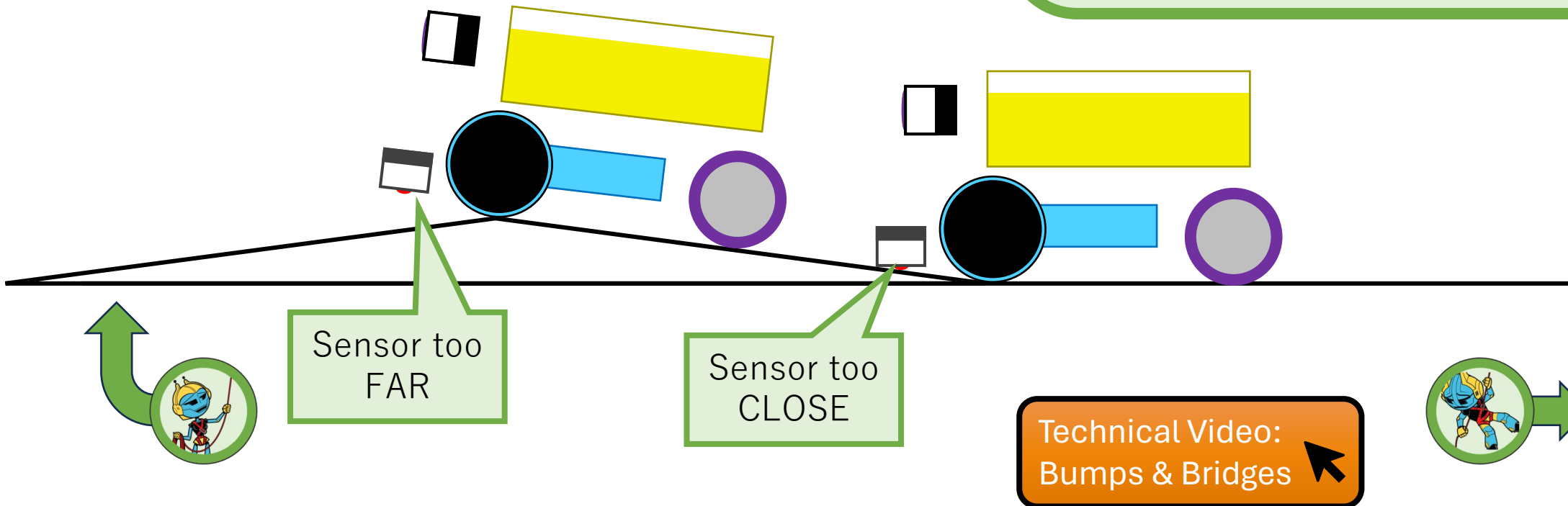


Your robot will face many challenges on the way to the rescue. What could possibly go wrong?

Fixed colour sensors may be too close to the surface when the robot approaches a raised surface.

This may impact robot movement if the sensors are in contact with the surface.

The distance from the surface may impact values read by the sensors.



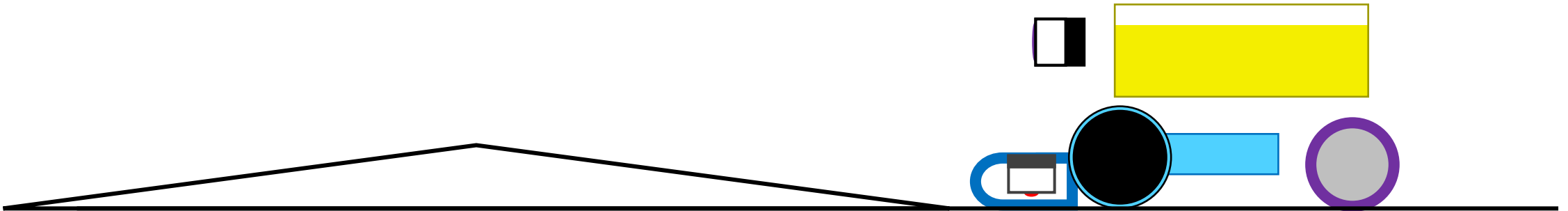
Floating Sensor



Maybe consider a “floating sensor”.

What is a floating sensor?

- A sensor that moves up and down with the contours of the course.



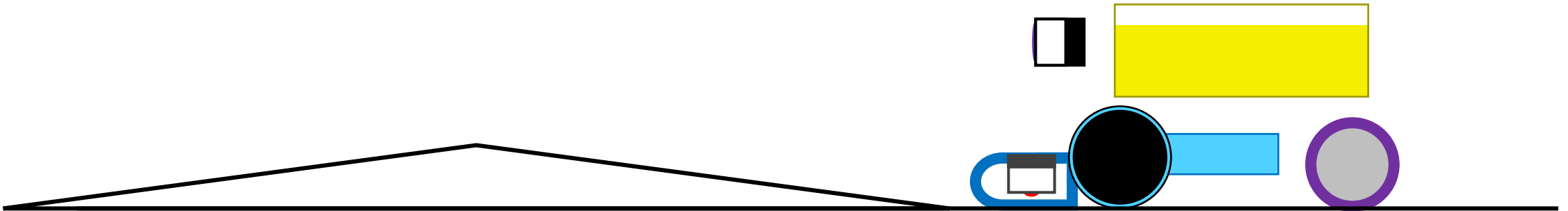
Floating Sensor



Maybe consider a “floating sensor”.

What is a floating sensor?

- A sensor that moves up and down with the contours of the course.



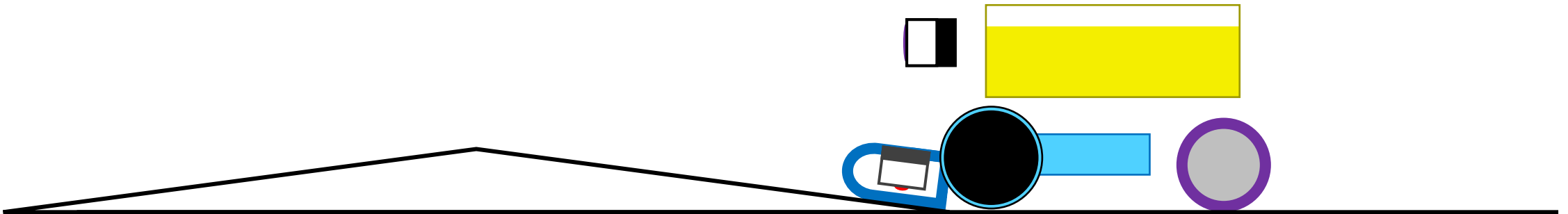
Floating Sensor



Maybe consider a “floating sensor”.

What is a floating sensor?

- A sensor that moves up and down with the contours of the course.



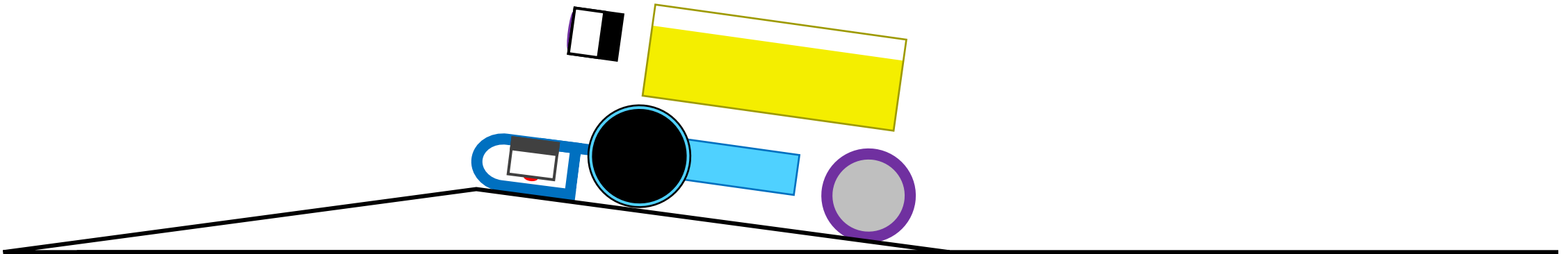
Floating Sensor



Maybe consider a “floating sensor”.

What is a floating sensor?

- A sensor that moves up and down with the contours of the course.



Floating Sensor



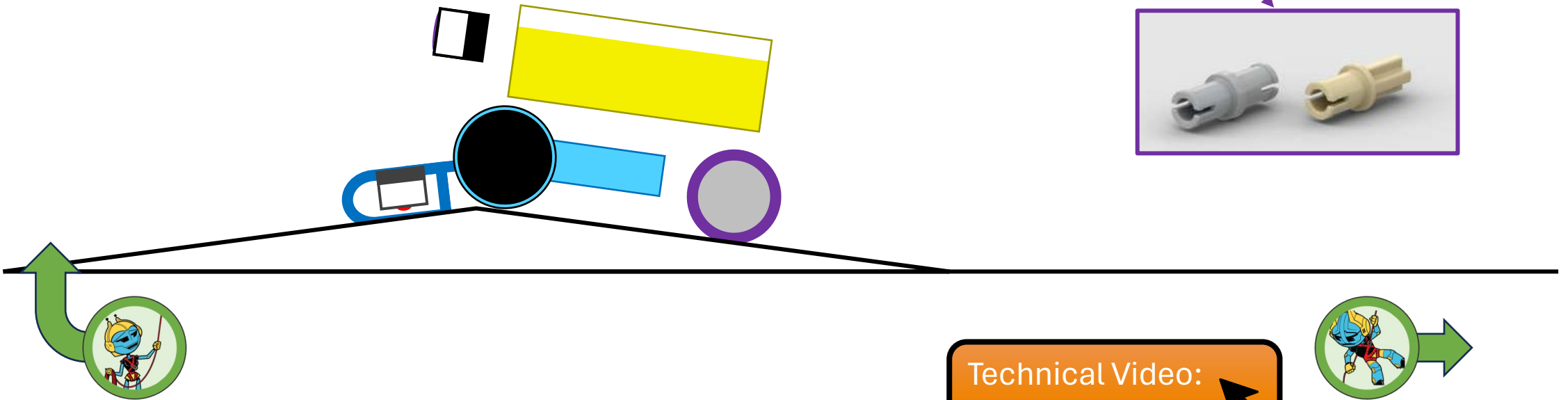
Maybe consider a “floating sensor”.

What is a floating sensor?

- A sensor that moves up and down with the contours of the course.

Many variations on construction including thin wheels without tyres, pieces with gentle curves (not standard in Spike sets).

The “frictionless” peg connectors (grey or tan) are very useful for creating pivot points.



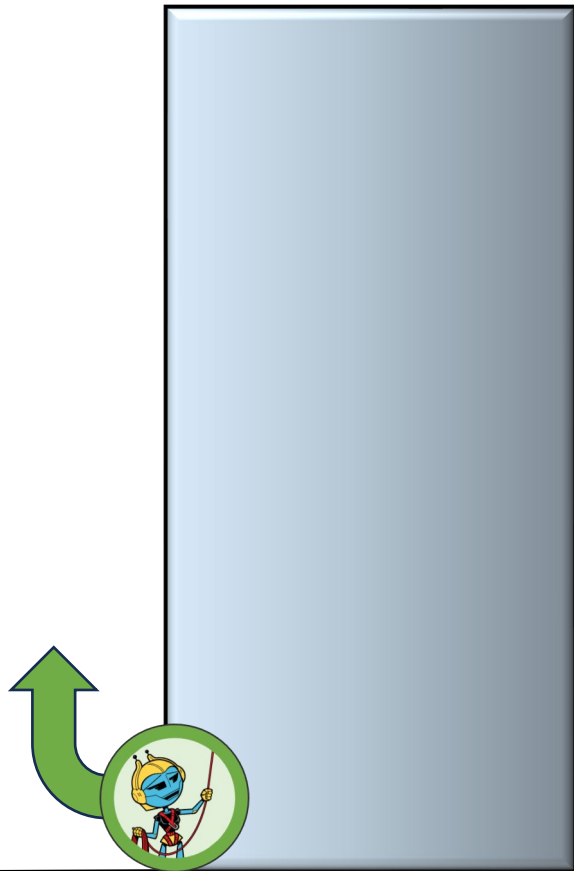
Technical Video:
Bumps & Bridges



Physical Challenges



The Water Tower



What about obstacles? Which is the best sensor to use?

I won't give you an answer, but I will give you some things to think about.

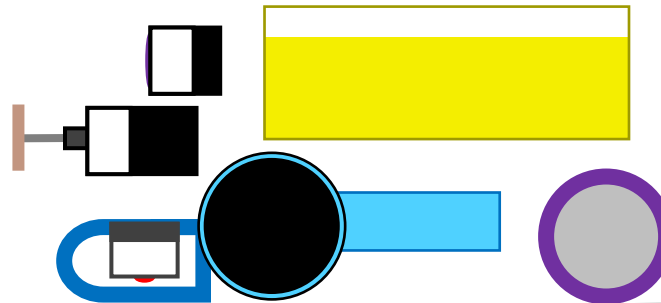
One option is the ultrasonic sensor:

- You can determine exactly how far your robot is from the obstacle.
- But might your robot mistake other objects (e.g., a riser) for the obstacle?

Another option is a force (touch) sensor:

- Your robot will need to make physical contact before detection.
- But is the touch sensor sensitive enough?

As you can see, there are many decisions to be made, all of which will impact the build of your robot.



Breaking Down the Problem and Coding the Robot Examples

Workshop: Obstacle Avoidance

Basic Principles: Breaking down the problem

Workshop: Single Sensor Line Following

Basic Principles: Breaking down the problem

Workshop: Two Sensor Line Following

Basic Principles: WHITE/BLACK Line Follow

Workshop: Proportional Control Line Following

Basic Principles: Breaking down the problem



A stylized cartoon robot with a blue body and yellow joints is climbing a red rope. The robot is positioned in the center of the frame, with its body angled towards the left. It has a round head with black eyes and a small antenna. Its arms are extended, gripping the rope, and its legs are also wrapped around it. The robot's feet are black with small white dots. The background is a solid grey color.

Workshop: Obstacle Avoidance

Basic Principles: Breaking down the problem



Hey there! In this presentation I'll take you through some basic concepts for avoiding an obstacle.

All you'll need to do is click to go to the next slide and I'll pop up along the way. You'll see me appear at the bottom with an arrow. This means you can advance to the next slide, when ready.

If at any point you want to go back to the "Code Examples" index slide, click on my friend on the left of the screen.

Best of luck and remember you can always reach out to your local RCJA committee for help.



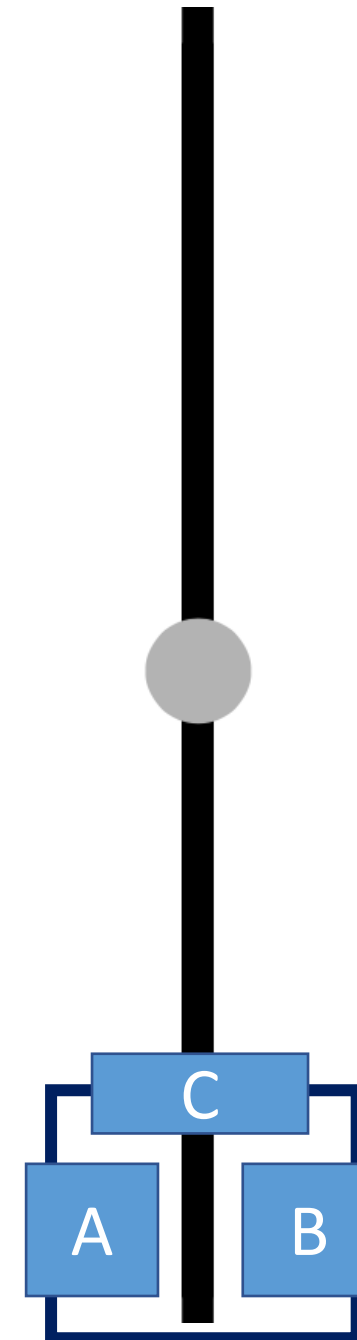
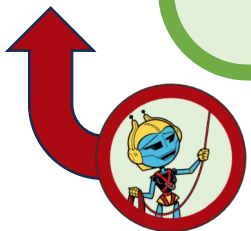
Robot Movement



First, we will take one of the tiles as an example – Water Tower.

We will also assume we have a standard two wheeled robot (**A** and **B**) with a forward pointing Ultrasonic sensor (**C**). For this you can use any range sensor, a push button will also work, you will just need to adjust the sensor blocks later in the presentation.

The mock robot is placed at the entrance of the tile.

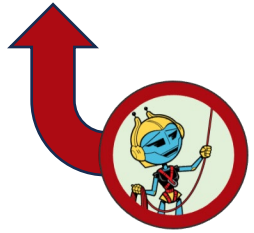
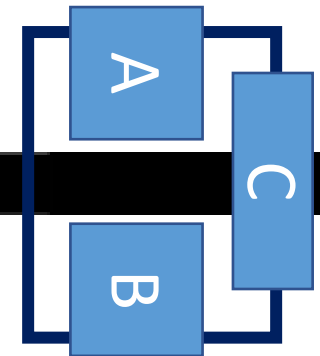




Next, you'll need a piece of paper and pen. Two different colors is best.

On the sheet of paper, in one of the colors, draw the Water Tower tile. We will use this to plan our route around the obstacle.

I have put mine horizontal with the Robot on the left. This will be the start of the tile.

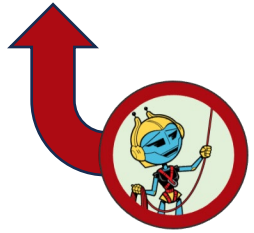
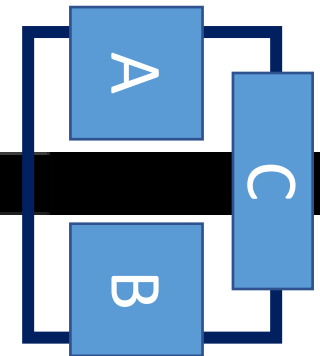




Now, we want to create a plan for our robot. We will do this by drawing arrows that show the robot's intended path.

We are working in a scenario where the robot is driving forwards, sees an obstacle, backs away and then goes around.

I'll start by drawing some of the path first.

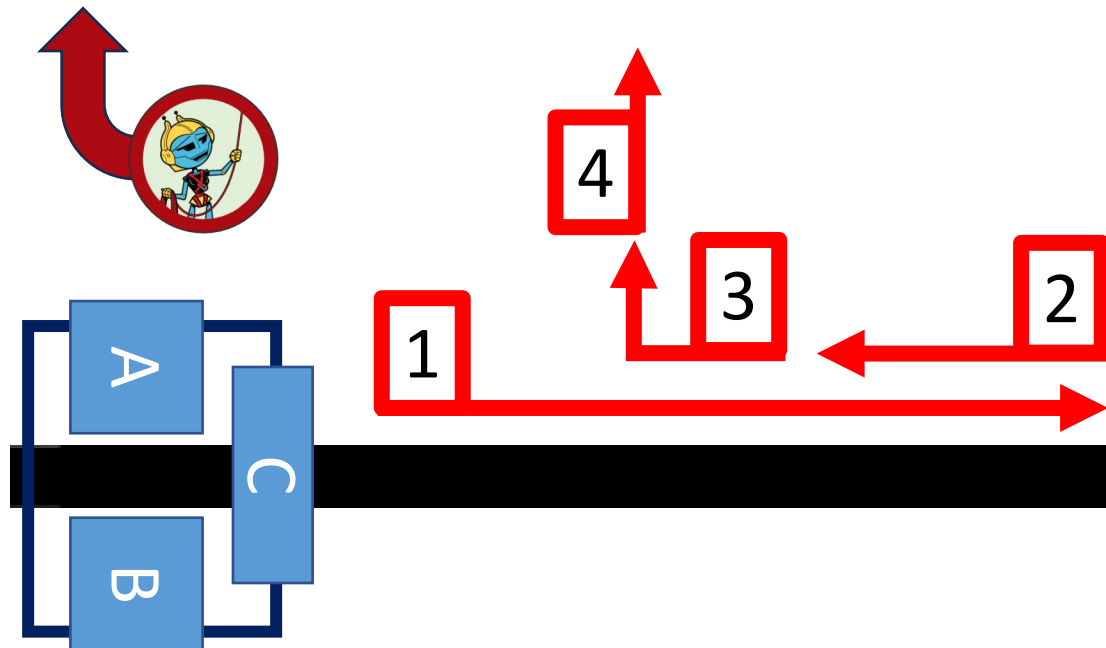




1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. ...

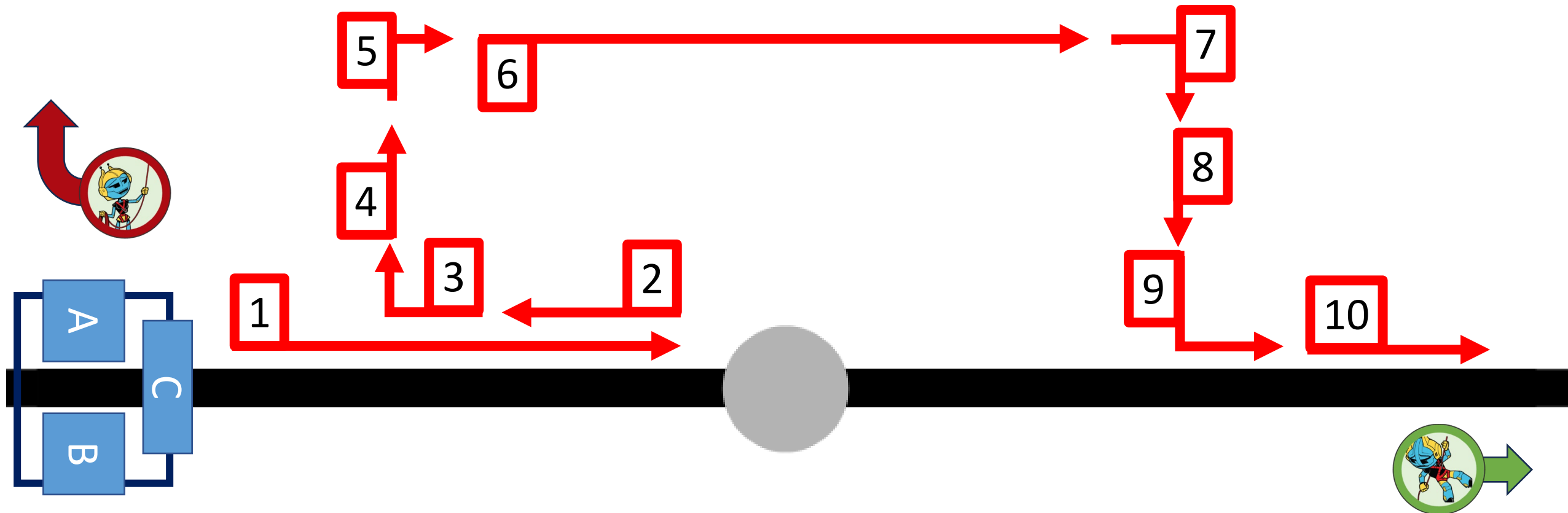


Have a go at finishing the path and continue building the list. I'll put mine on the next slide. Use a different pen to the one used to draw the tile.





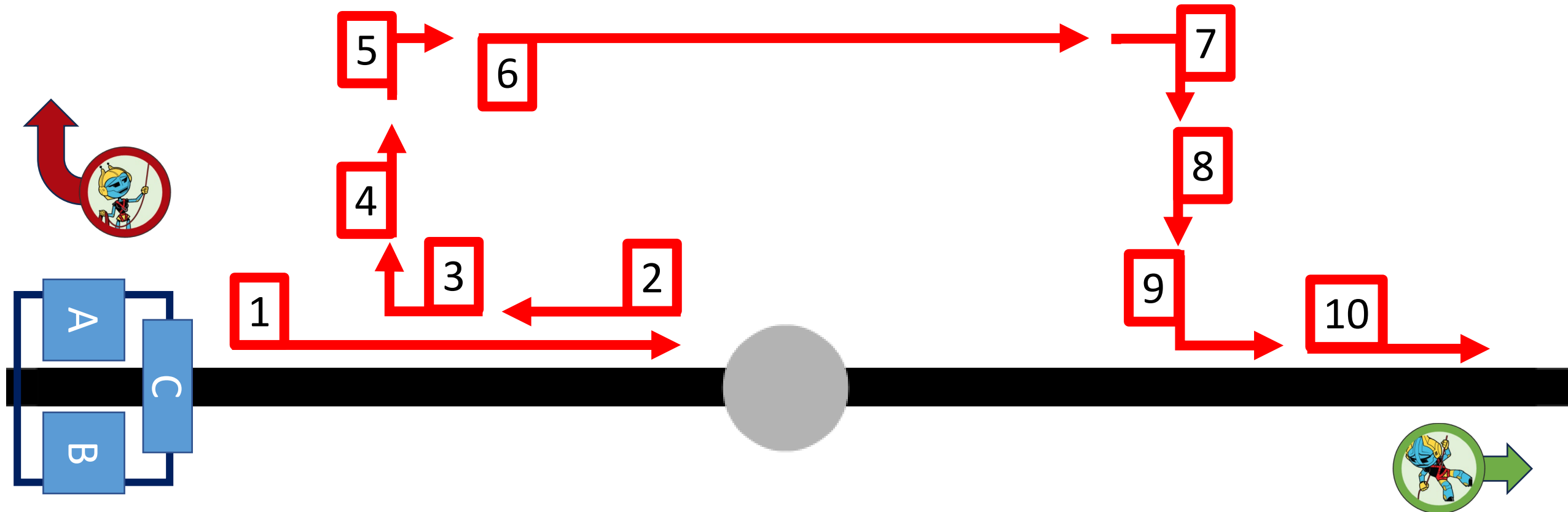
Your path might look something like this.





Finally, your list should look something like this

1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. Turn Right 90 degrees – parallel to line
6. Drive forward and pass obstacle
7. Turn right 90 degrees – face line
8. Drive towards line
9. Turn right 90 degrees – parallel to line
10. Continue along the line





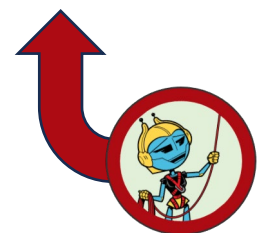
1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. Turn Right 90 degrees – parallel to line
6. Drive forward and pass obstacle
7. Turn right 90 degrees – face line
8. Drive towards line
9. Turn right 90 degrees – parallel to line
10. Continue along the line



Now in front of you there should be a drawn map of the robot's path around the obstacle and a list of instructions.

We are going to work through some of the steps in your list to give an idea of how to code this path.

I'll use blocks from the SPIKE Word Blocks and will also give some descriptions. The logic will translate to all coding languages.





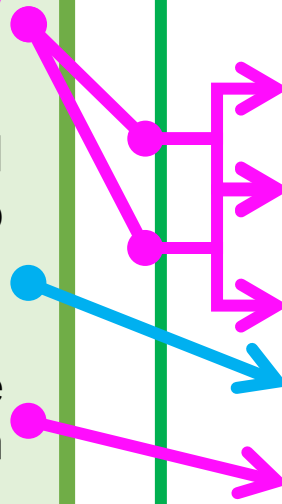
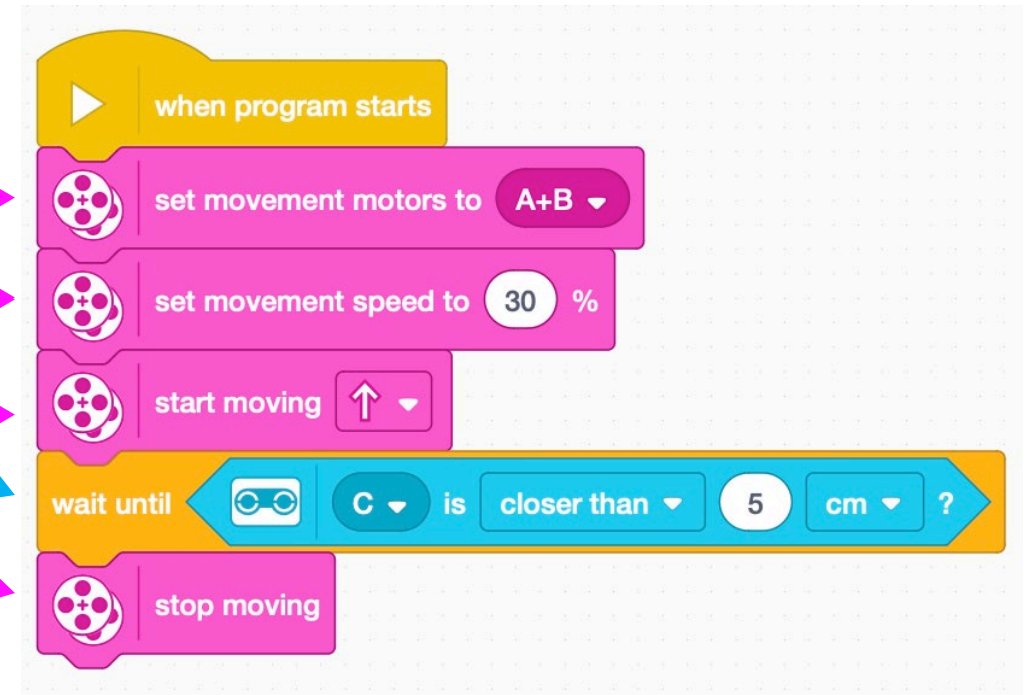
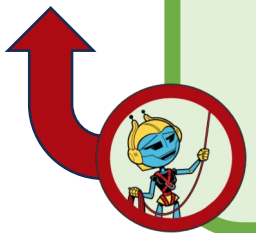
1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. Turn Right 90 degrees – parallel to line
6. Drive forward and pass obstacle
7. Turn right 90 degrees – face line
8. Drive towards line
9. Turn right 90 degrees – parallel to line
10. Continue along the line



We first set the **MOTORS** and **SPEED**, followed by the **MOVEMENT** direction.

Next, we use the **WAIT** block and **ULTRASONIC SENSOR INPUT** to keep checking the sensor.

This will cover **STEP 1**. We will have the robot **STOP** for now rather than completing step 2. Run the code; you may need to adjust the **CLOSER THAN** value, so the robot is truly 5 cm away.



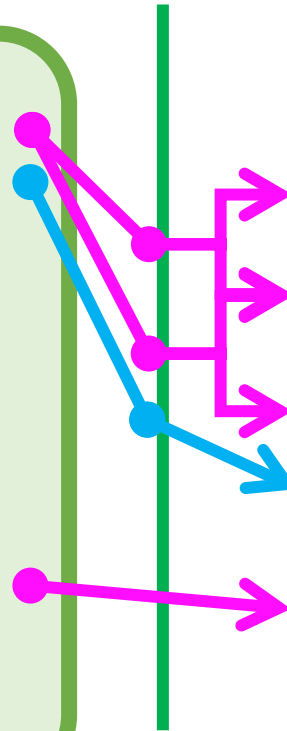
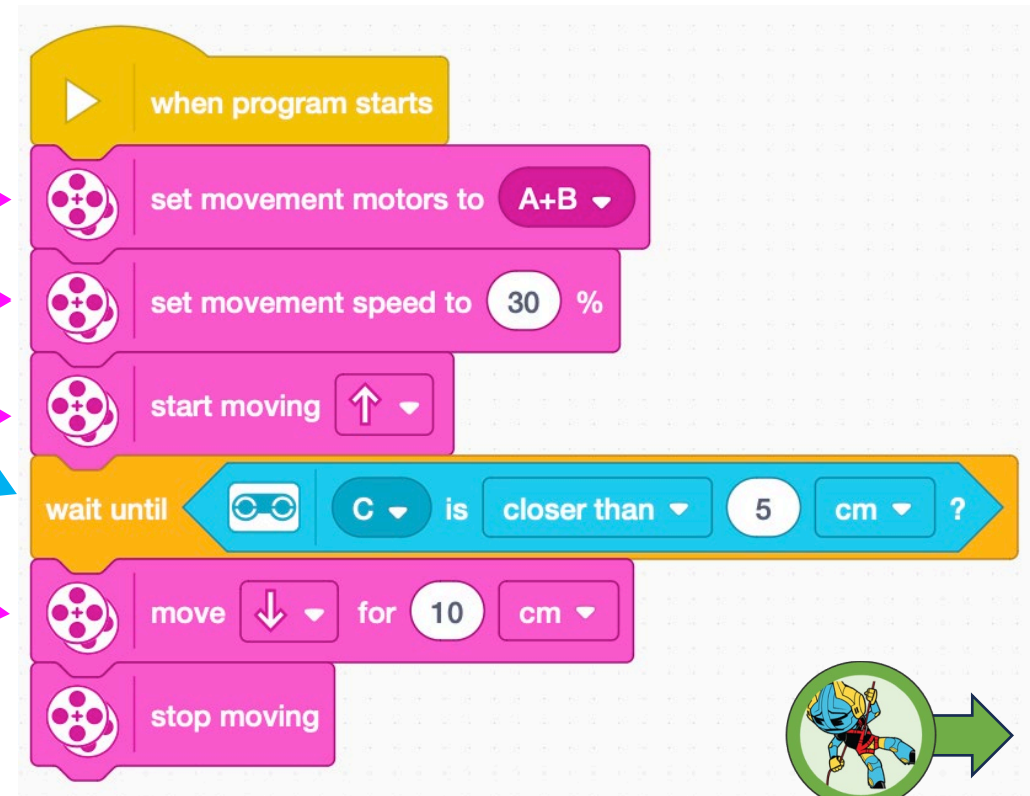


1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. Turn Right 90 degrees – parallel to line
6. Drive forward and pass obstacle
7. Turn right 90 degrees – face line
8. Drive towards line
9. Turn right 90 degrees – parallel to line
10. Continue along the line



Now we have **STEP 1** completed – **DRIVE FORWARD** and **DETECT OBJECT** - We can move onto **STEP 2**.

To do this we replace the **STOP** we added in the previous slide with a **MOVE BACKWARDS** for a distance block. I will select 10 cm, but you'll have to tune a distance, so your robot does not leave the tile. Let's see this all together.





1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. Turn Right 90 degrees – parallel to line

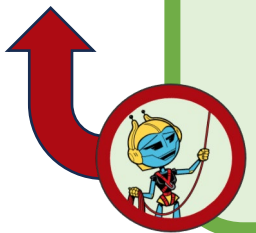
6. Drive forward and pass obstacle
7. Turn right 90 degrees – face line
8. Drive towards line
9. Turn right 90 degrees – parallel to line
10. Continue along the line



Now you can tick off **STEP 1** & **STEP 2**.

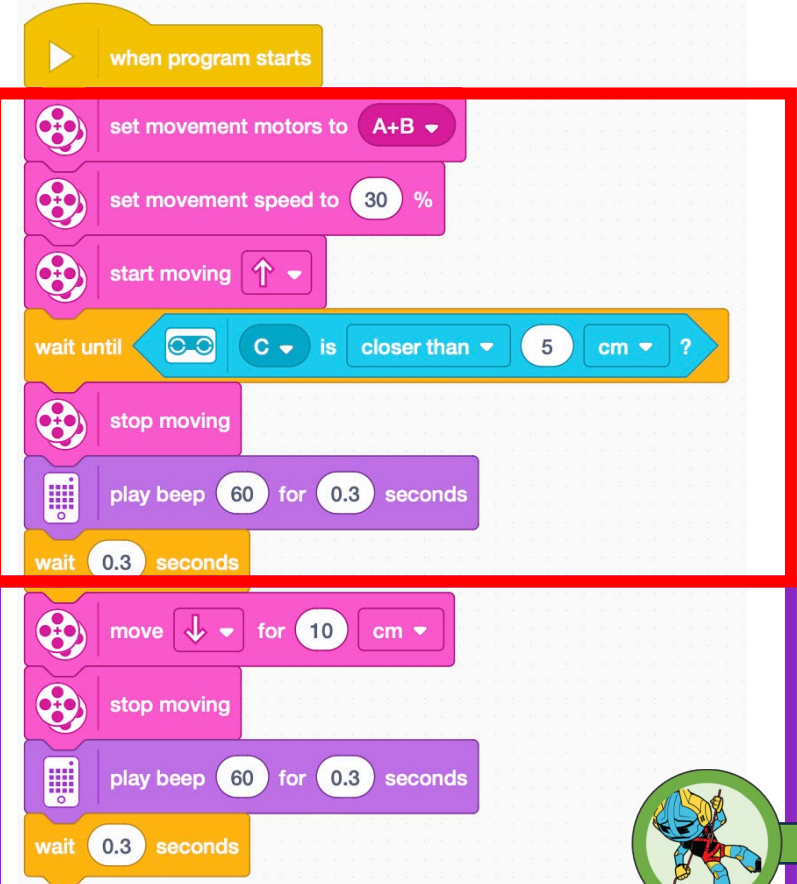
Lastly, I just want to show you some ways you can make the development easier. You'll see I've added some blocks to the code.

Knowing what step you are up to can be confusing, so I recommend adding some sounds and/or time delays with **STOPS**. This way you know where you are up to while the robot is running.



STEP 1

STEP 2





1. Drive forward and detect obstacle
2. Back away slightly from obstacle
3. Turn 90 degrees away from the line
4. Drive away from the line, clearing obstacle
5. Turn Right 90 degrees – parallel to line

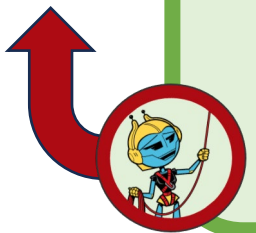
6. Drive forward and pass obstacle
7. Turn right 90 degrees – face line
8. Drive towards line
9. Turn right 90 degrees – parallel to line
10. Continue along the line



And there we go! That is the basic concepts behind breaking down and coding obstacle avoidance.

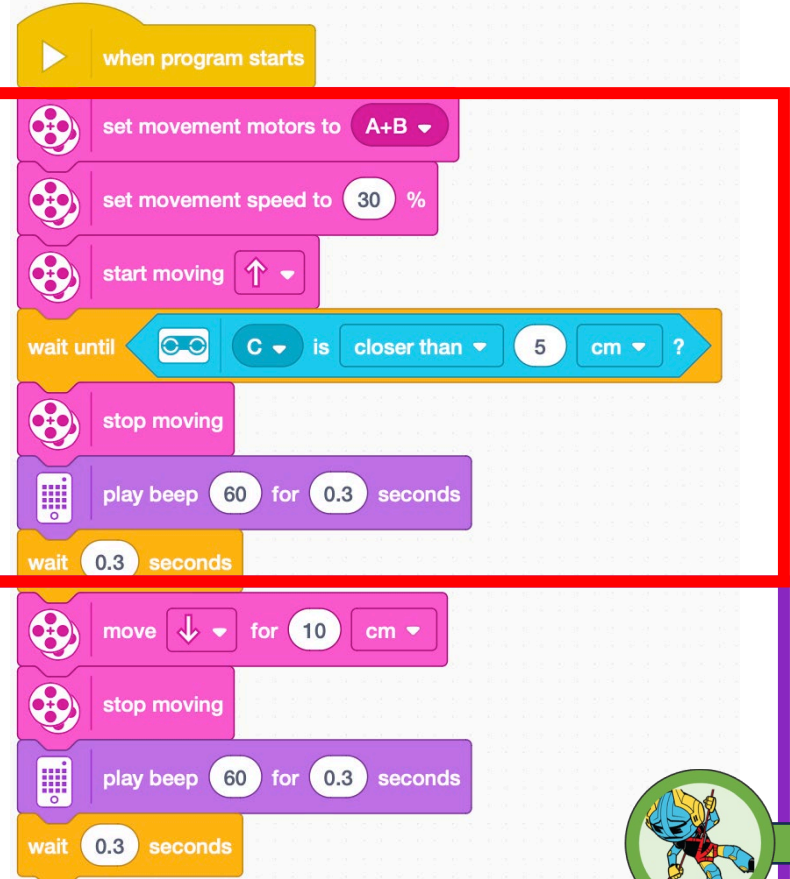
The remaining steps will use the same blocks as **STEP 1** and **STEP 2**, except with different movements and distances.

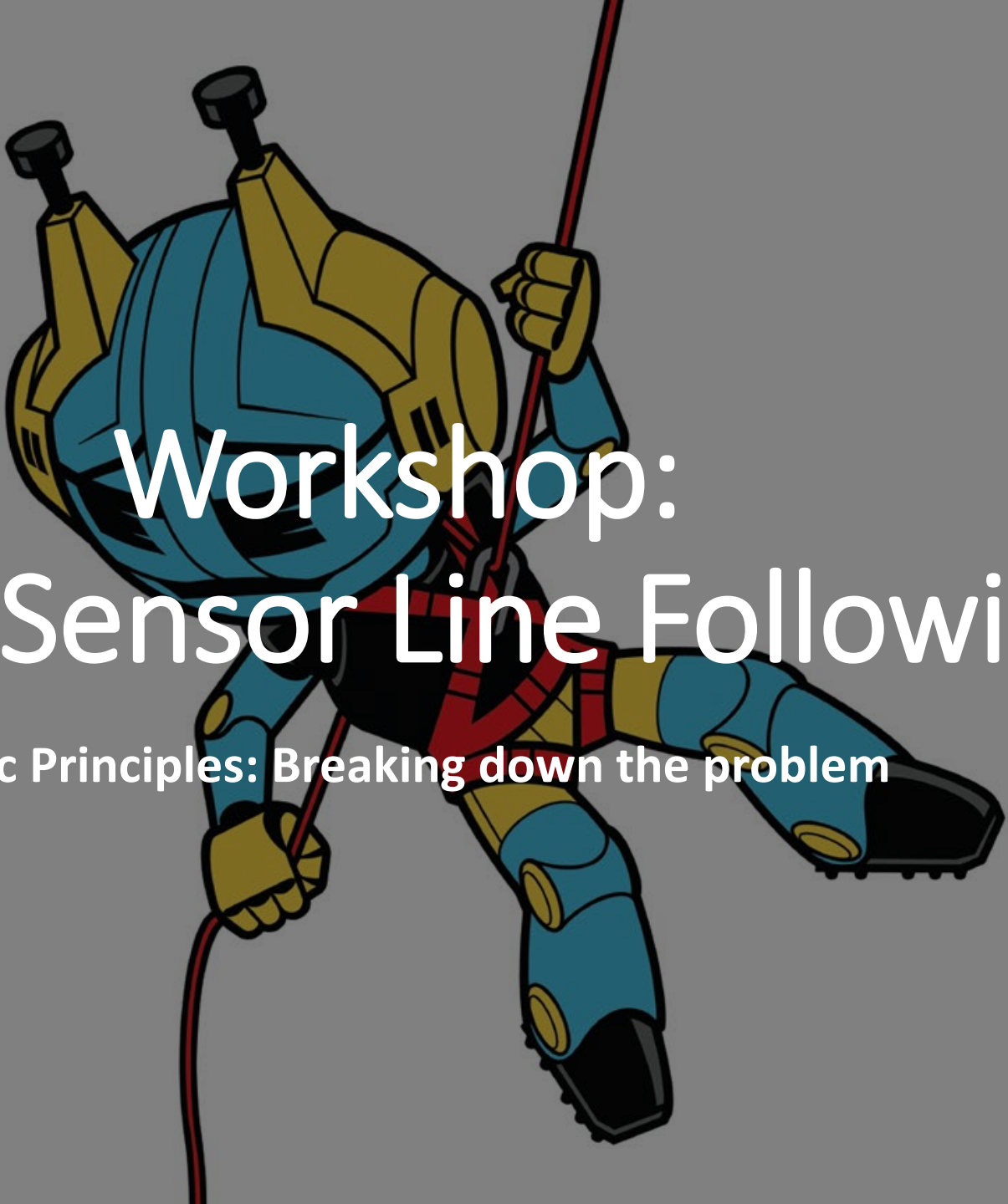
Use the list to make the rest of your code. GOOD LUCK 😊



STEP 1

STEP 2



A cartoon robot with a blue and yellow body is climbing a red rope. The robot has a round blue head with two black antennae, a blue torso with yellow joints, and blue legs with yellow joints. It is holding the red rope with its yellow-gloved hands. The background is a solid grey color.

Workshop: Single Sensor Line Following

Basic Principles: Breaking down the problem

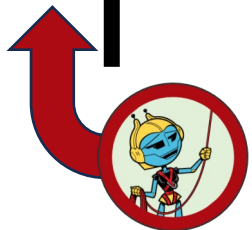


Hey there! In this presentation I'll take you through some basic concepts for avoiding an obstacle.

All you'll need to do is click to go to the next slide and I'll pop up along the way. You'll see me appear at the bottom with an arrow. This means you can advance to the next slide, when ready.

If at any point you want to go back to the "Code Examples" index slide, click on my friend on the left of the screen.

Best of luck and remember you can always reach out to your local RCJA committee for help.



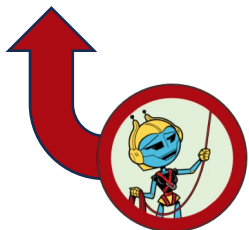
Robot Movement



First, we will take one of the tiles as an example – A Straight.

We will also assume we have a single light sensor robot, with its sensor placed to the Left of the line. Shown by the L square.

The mock robot is placed at the start of the tile.



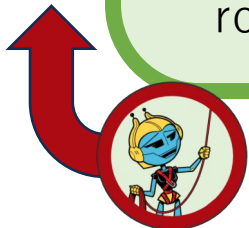
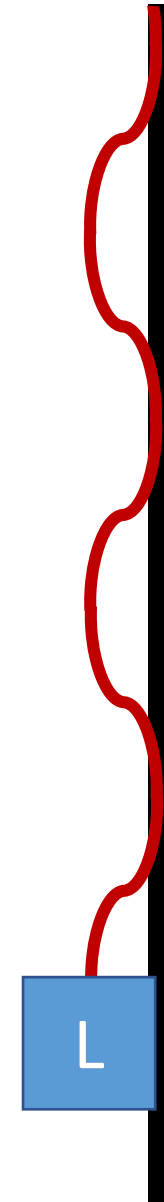
Robot Movement



Now to follow a line with line sensors, we need to think like a robot. Unlike a human, a robot with a light sensor can only see what is below it, not what is ahead. A human will ask, “Can I see the line ahead of me.” A robot will ask, “Am I on the line or not?” The robot is essentially following the edge of the path - if on the path, head away from it; if off the path, head towards it.

We do this with a movement that looks like a wobble.

I’ve drawn the **path** in red. You’ll notice how the robot wobbles towards and away from the line.



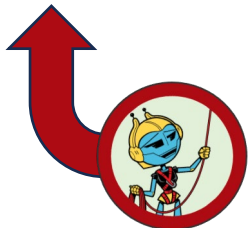
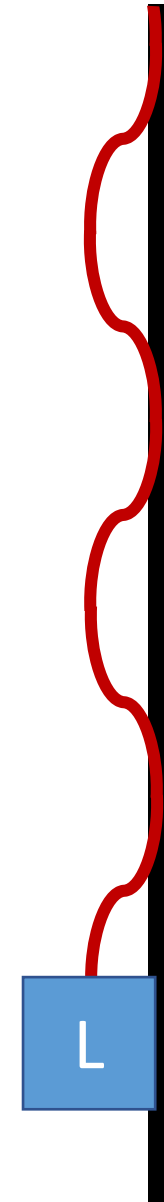
Robot Movement



Now we have a **path** we can start to break it down into simpler language.

When the robot is seeing WHITE then turn towards the line or if the robot is seeing BLACK turn away from the line.

Looking at this we have two conditions, this is the exact logic of an **IF** statement, let's explore.



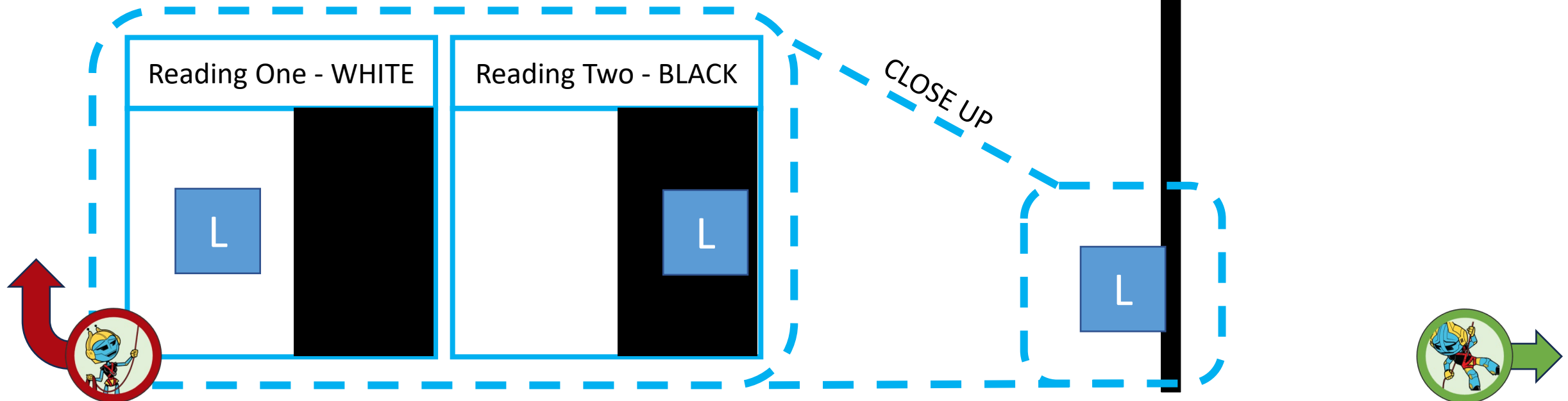
Robot Movement



To begin, we are going to collect two light values, one for WHITE and the other for BLACK.

Then you want to pick a number that is roughly in the middle of the two. For this workshop, it does not need to be exact.

We do this because the **IF** needs something to check against.





You should have your two measured values, WHITE and BLACK. For now, we just need to pick a value roughly in the middle of these two, we will call this the **chosen value**. You can also tune this number to increase robot performance.

You can also calculate this value using an average of the two readings but for now selecting a value will work out OK.

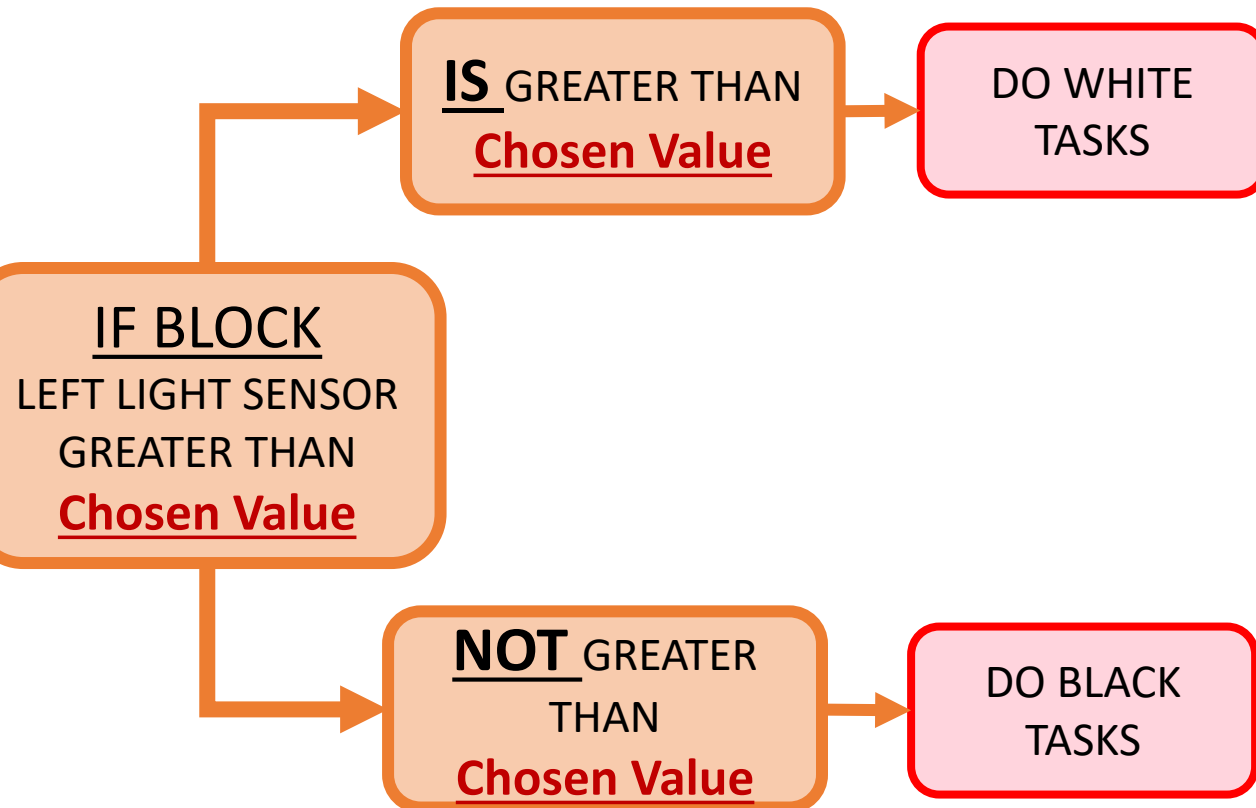
Typically, WHITE will tend towards 100 and BLACK towards 0. This means if you are having trouble with readings, you can start by just selecting 50.



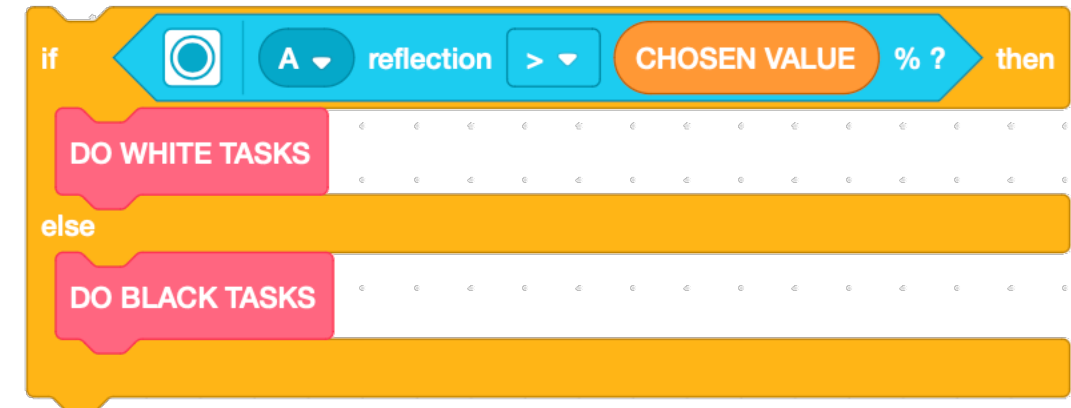


Let's have a look at how we can use our **Chosen Value** in code. For this example, I'll use blocks from the SPIKE Word Blocks and will also show a general diagram for a single sensor check. The logic will translate to all coding languages.

THEORY



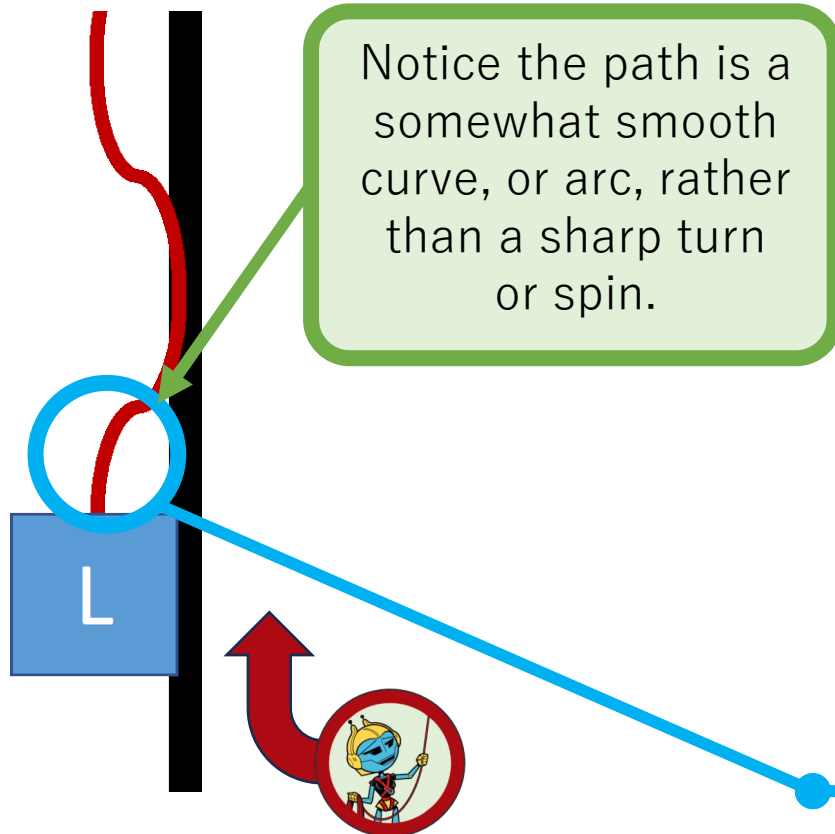
SPIKE Word Block



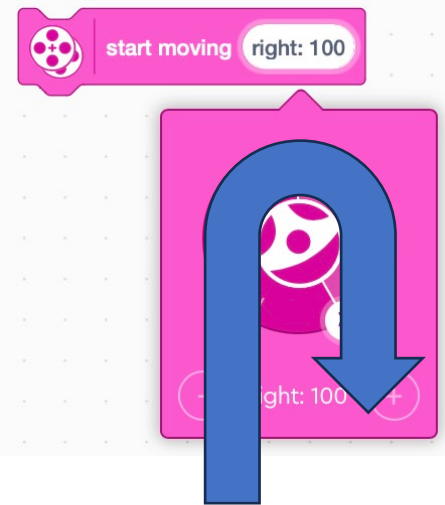


Lastly, something to remember is the wave pattern, or wobble, that we are trying to achieve. If we set our robots to turn all the way around or a perfect right angle, we won't move forward. So, we'll need to make sure the turn is an arc shape. These turns are something you will need to tune to get your robot working well.

THEORY



✗ SHARP TURN ✗



✓ GENTLE ARC ✓





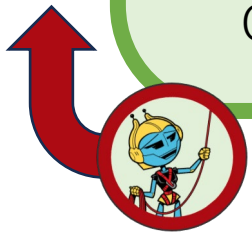
And there we go! That is the basics for single sensor line following.

Using the example on the right you can now add in your **movement blocks**, **chosen value**, and forever loops.

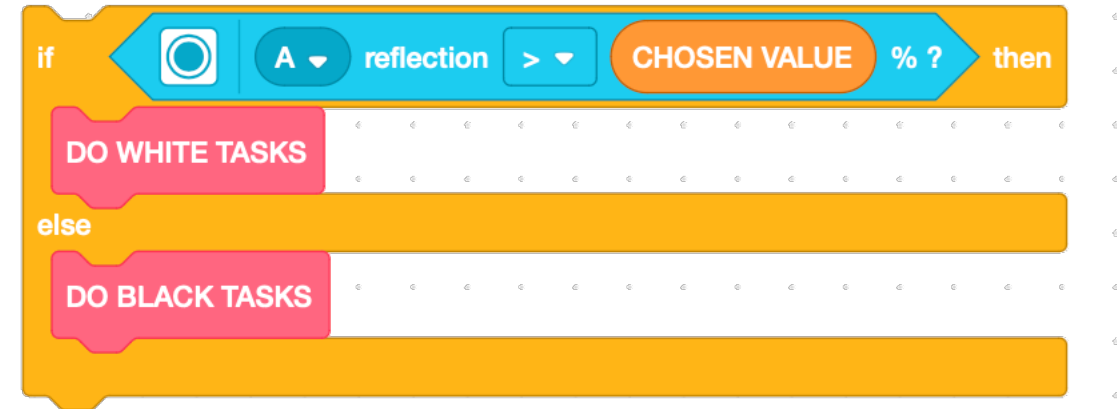
Some common things to check:

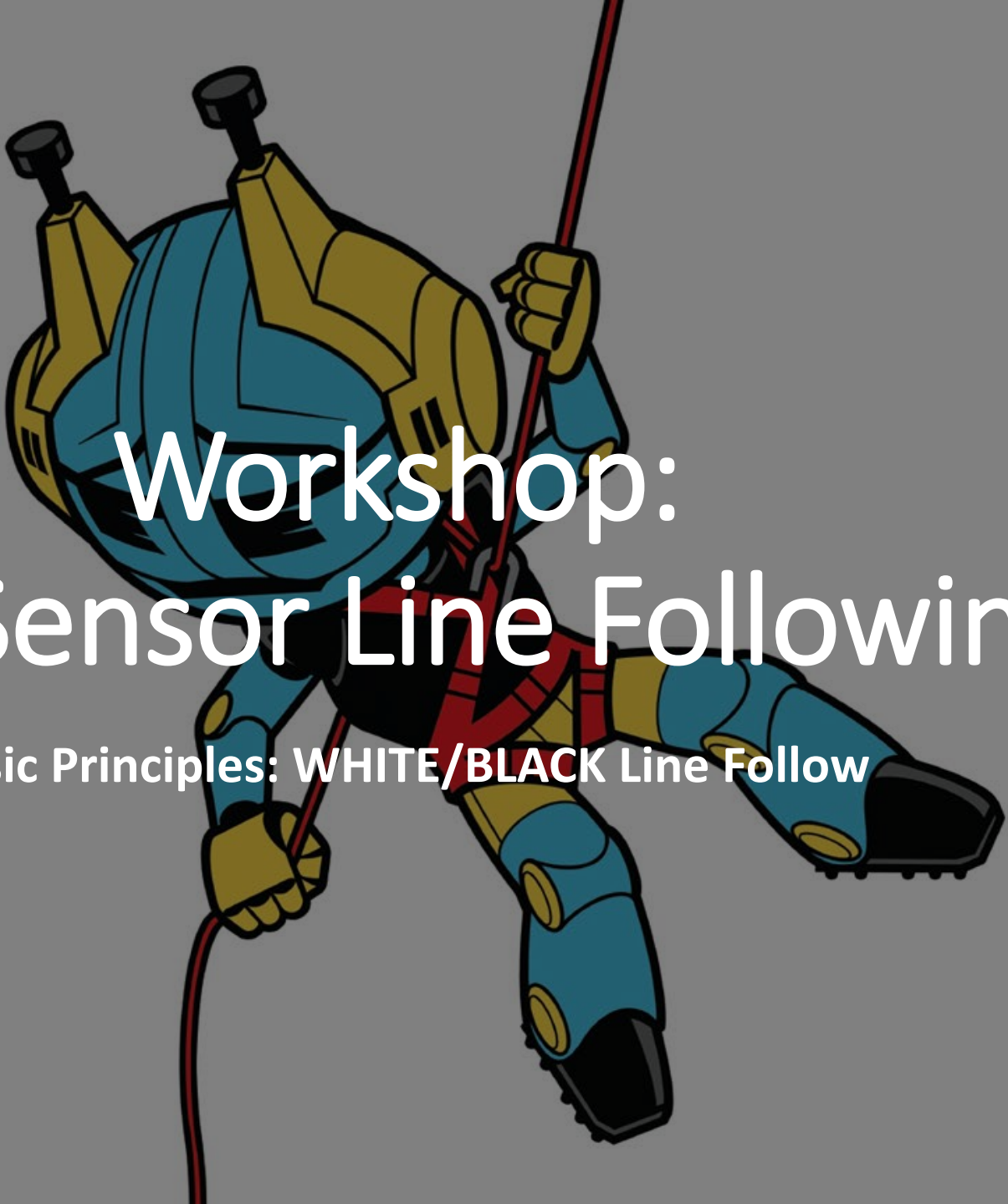
- Am I on the correct side of the line?
- Have I set the turns the correct way?
- Is my chosen value too high or low?
- Is my arc too sharp or shallow?
- Should I adjust the robot speed?

Good luck and happy coding!



SPIKE Word Block



A cartoon robot with a blue and yellow body is climbing a red rope. The robot has a blue head with two black sensors, yellow arms, and blue legs with yellow joints. It is holding the rope with both hands and has one foot planted on the rope. The background is a solid grey color.

Workshop: Two Sensor Line Following

Basic Principles: WHITE/BLACK Line Follow

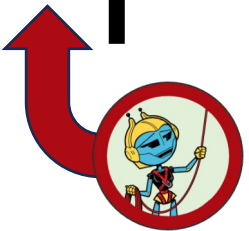


Hey there! In this presentation I'll take you through some basic concepts for avoiding an obstacle.

All you'll need to do is click to go to the next slide and I'll pop up along the way. You'll see me appear at the bottom with an arrow. This means you can advance to the next slide, when ready.

If at any point you want to go back to "Code Examples" index slide, click on my friend on the left of the screen.

Best of luck and remember you can always reach out to your local RCJA committee for help.



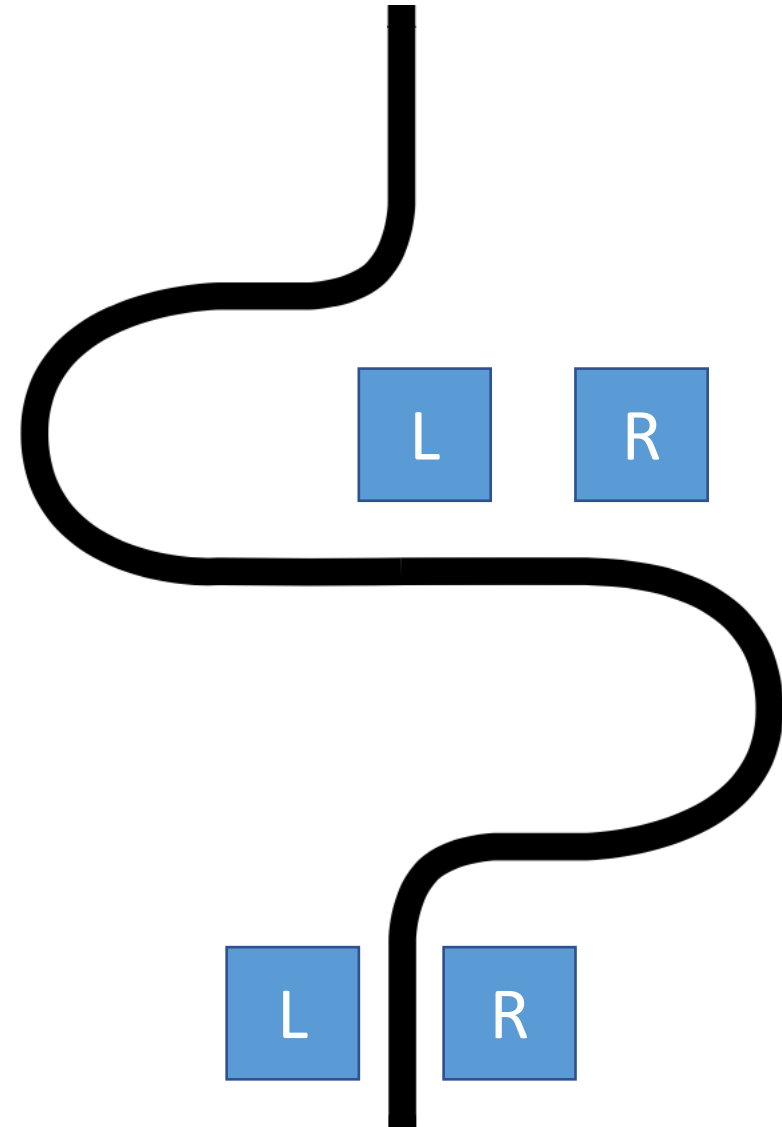
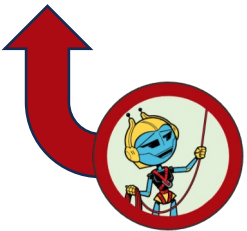
Robot Movement



First, we will take one of the tiles as an example – S Bend.

We will also assume we have two light sensors on your robot, these are shown by the **L** and **R** squares.

The mock robot is placed at the start of the tile, with a sensor either side of the black line.

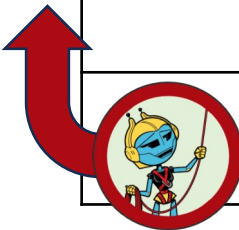
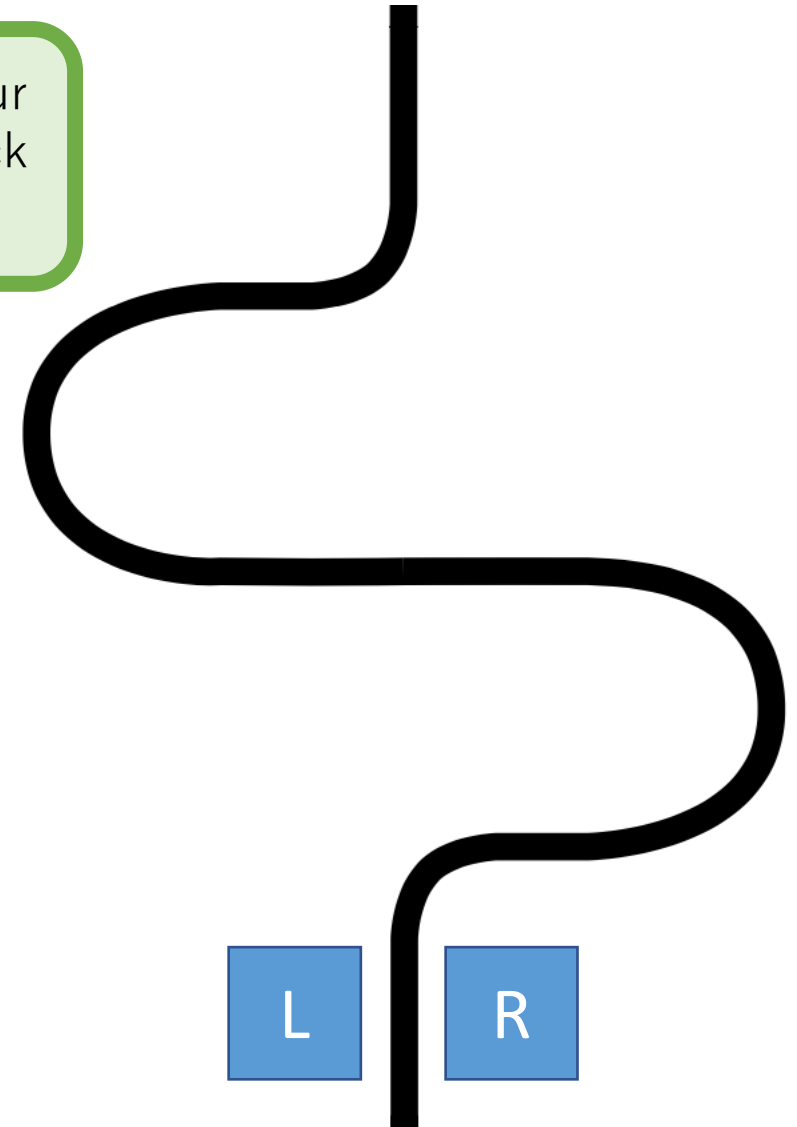


Robot Movement



Now, we want to first understand all the scenarios our robot may encounter along the line. To help keep track of this, we will create a table.

Left Sensor	Right Sensor	Movement

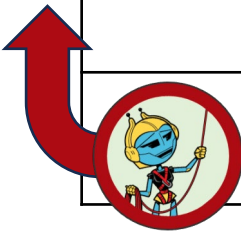
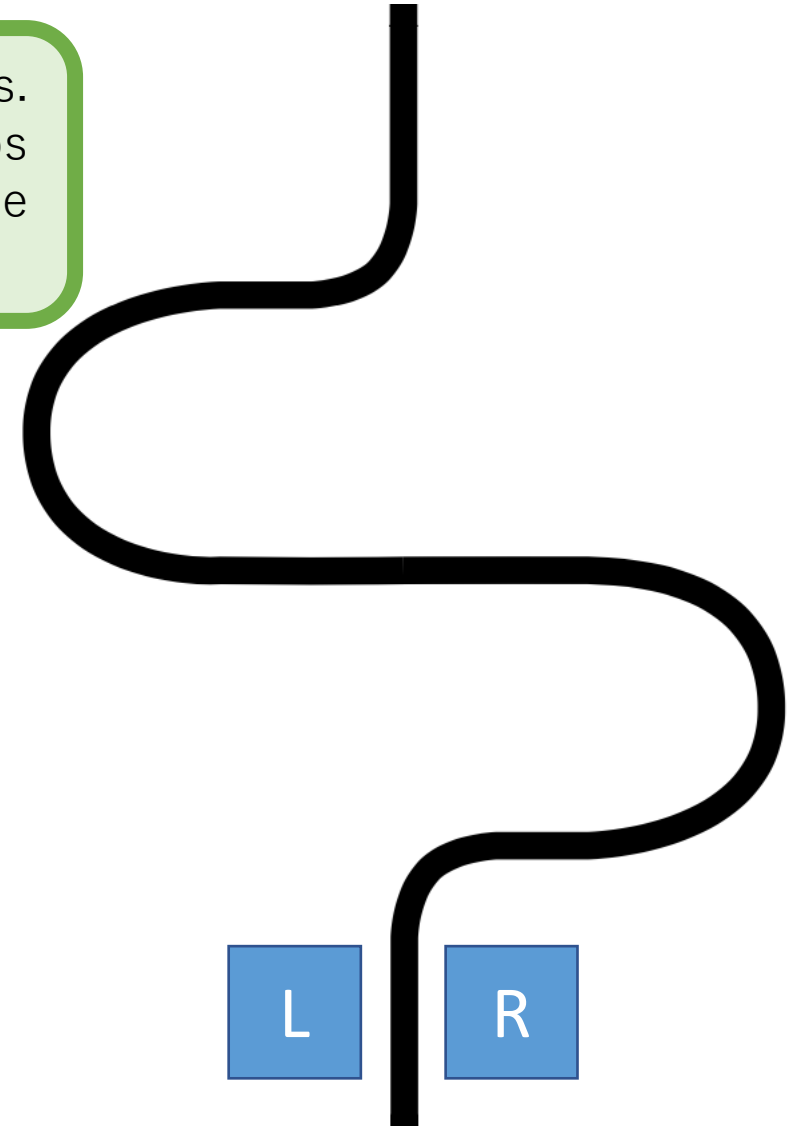


Robot Movement



You may see these, or similar, called Truth Tables. Now, we will step through, slide by slide, the scenarios we may see along the line. We will also fill out the table while doing this.

Left Sensor	Right Sensor	Movement

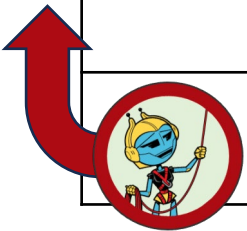
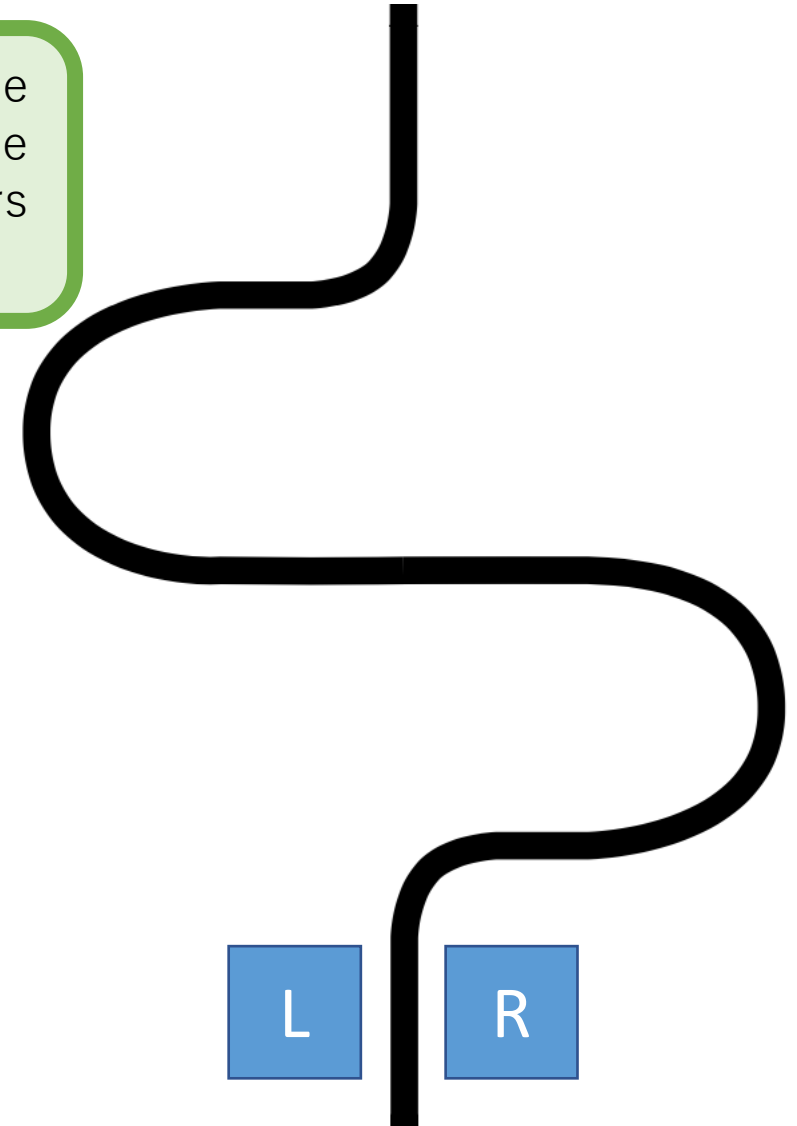


Robot Movement



In the first instance, we can see the robot is at the start of the tile. The LEFT sensor is on WHITE, and the RIGHT is also on WHITE. This indicates each sensors is clear of the line, so we can go FORWARD.

Left Sensor	Right Sensor	Movement
WHITE	WHITE	FORWARD

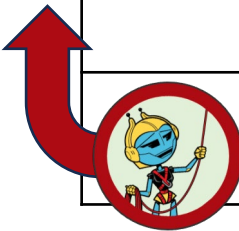
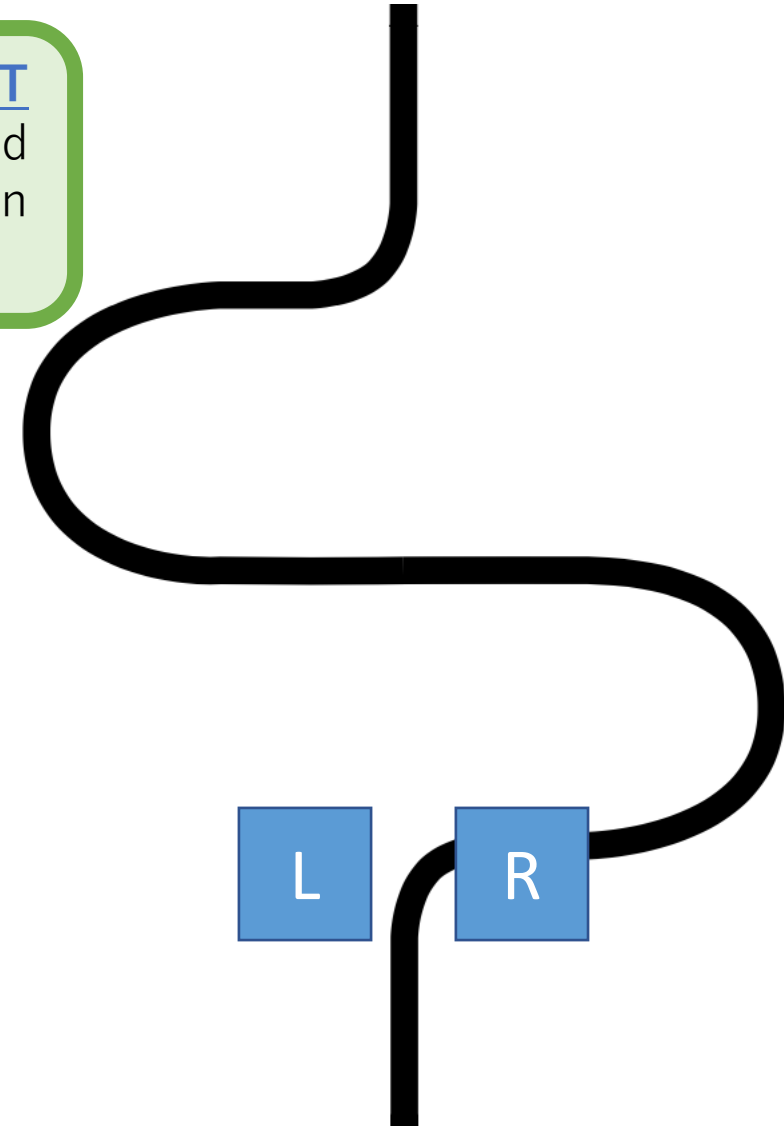


Robot Movement



Next, we can see the robot has progressed. The LEFT sensor is on WHITE, and the RIGHT has encountered BLACK. This indicates we have a change in direction and want to follow, turn RIGHT.

Left Sensor	Right Sensor	Movement
WHITE	WHITE	FORWARD
WHITE	BLACK	RIGHT TURN

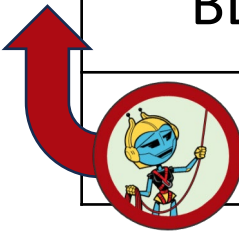
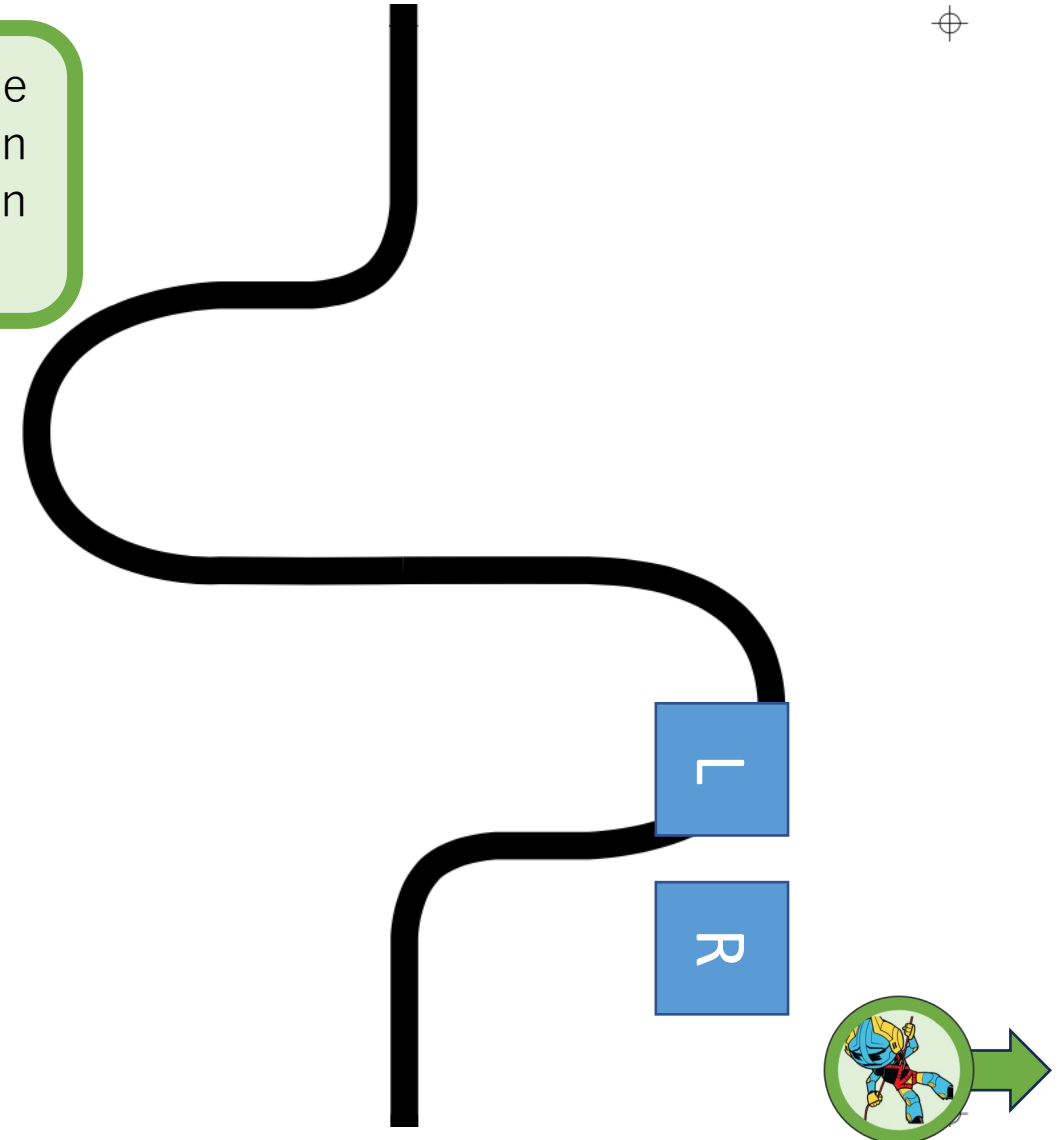


Robot Movement



Next, we can see the robot has progressed further. The LEFT sensor is now on BLACK, and the RIGHT is on WHITE. This indicates we have a change in direction and want to follow, turn LEFT.

Left Sensor	Right Sensor	Movement
WHITE	WHITE	FORWARD
WHITE	BLACK	RIGHT TURN
BLACK	WHITE	LEFT TURN



Robot Movement



Now we have the truth table we can convert it into more robot language. We can imagine each row of the table as an **IF** statement – meaning **IF** something is TRUE then do a task OR **IF** it is FALSE then do something else (we will focus on the true here).

Left Sensor	Right Sensor	Movement
WHITE	WHITE	FORWARD
WHITE	BLACK	RIGHT TURN
BLACK	WHITE	LEFT TURN
BLACK	BLACK	DO NOTHING

TO
IFs

1. **If** LEFT SENSOR and RIGHT SENSOR see WHITE, then move forward
2. **If** LEFT SENSOR sees WHITE and RIGHT SENSOR see BLACK, then turn right
3. **If** LEFT SENSOR sees BLACK and RIGHT SENSOR see WHITE, then turn left
4. **If** LEFT SENSOR and RIGHT SENSOR see BLACK, then do nothing

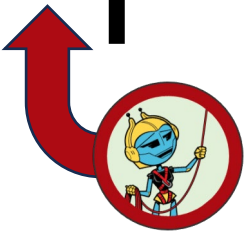




Now we have a table that describes the robot's motion and some robot language we can use to code. But before we do that, we need to think about what the sensors are "seeing."

We could hard code every scenario, e.g., if two sensors see white then go forward OR if the left is on black and the right on white then turn left, for every possible shade of BLACK and WHITE but this can make the code very large and not very reliable.

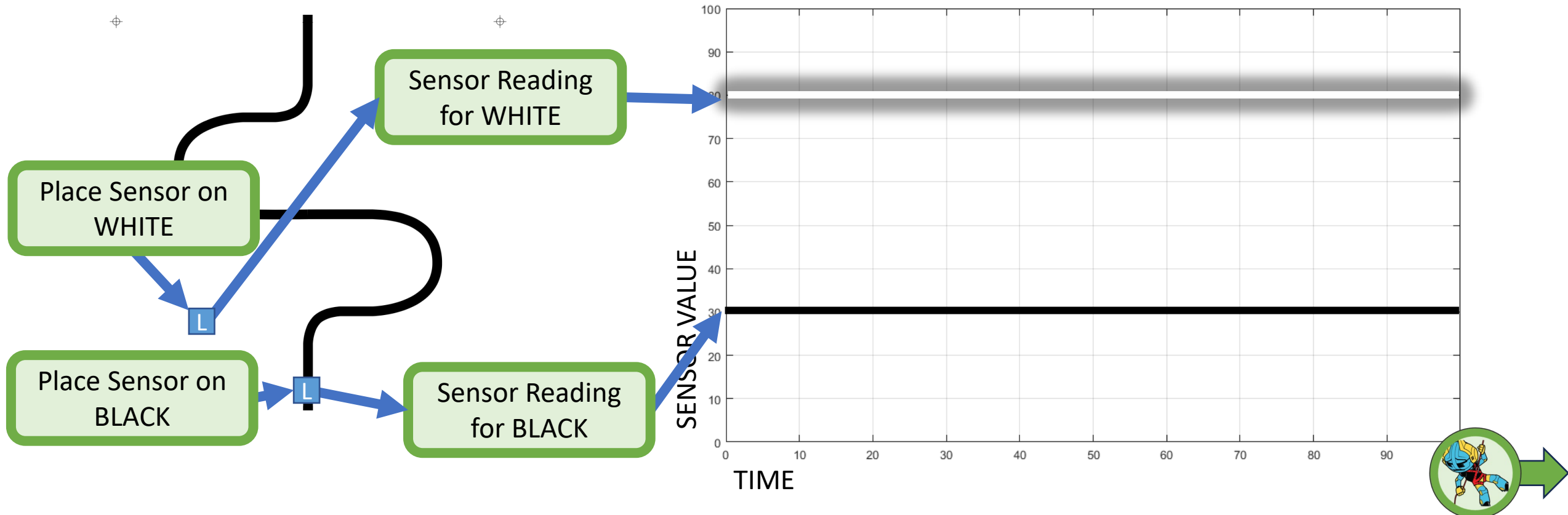
Let's explore this.



Light Readings



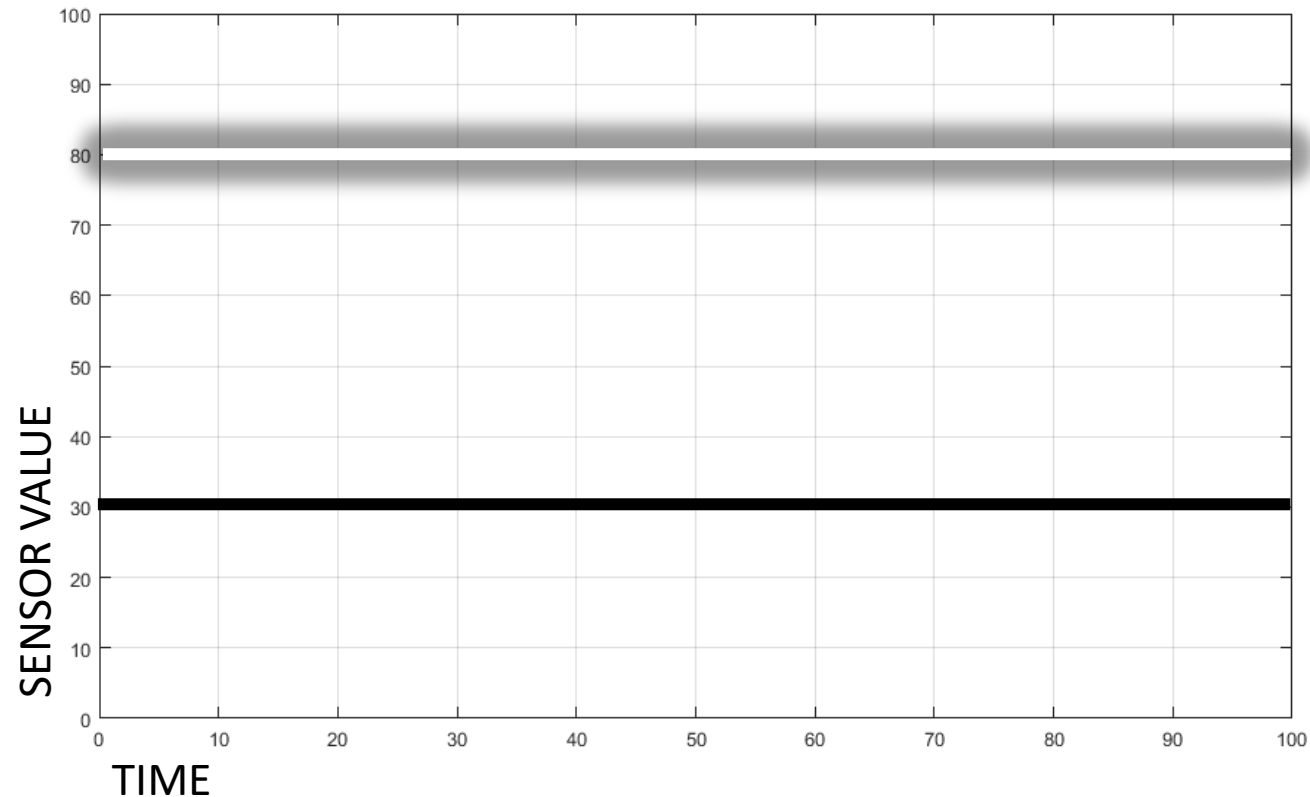
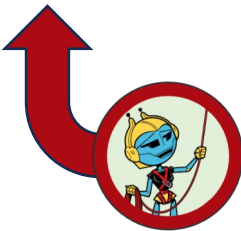
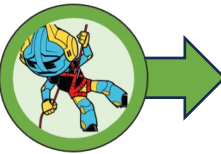
Let's explore, specific sensor readings and how that can apply to code. So, the graph on the right will show different sensor readings and the tile below will show where the sensor is. We will just look at the **LEFT** sensor for this example, in an ideal case.



Light Readings



In this ideal case, we have two distinct values for black and white. We could code two distinct areas. One part says **IF** a sensors sees WHITE do something and something else **IF** it sees BLACK. In this case it could work but in practice white and black will vary across a tile.



Light Readings



Now, initially the robot sees a nice WHITE tile. But in the real world different **LIGHT SOURCES** change how different areas of the tile, or track, are lit up – you can see this with the overlaid gradient changing the darkness across the tile.

Place Sensor on
WHITE

L

L

L

Place Sensor on
BLACK

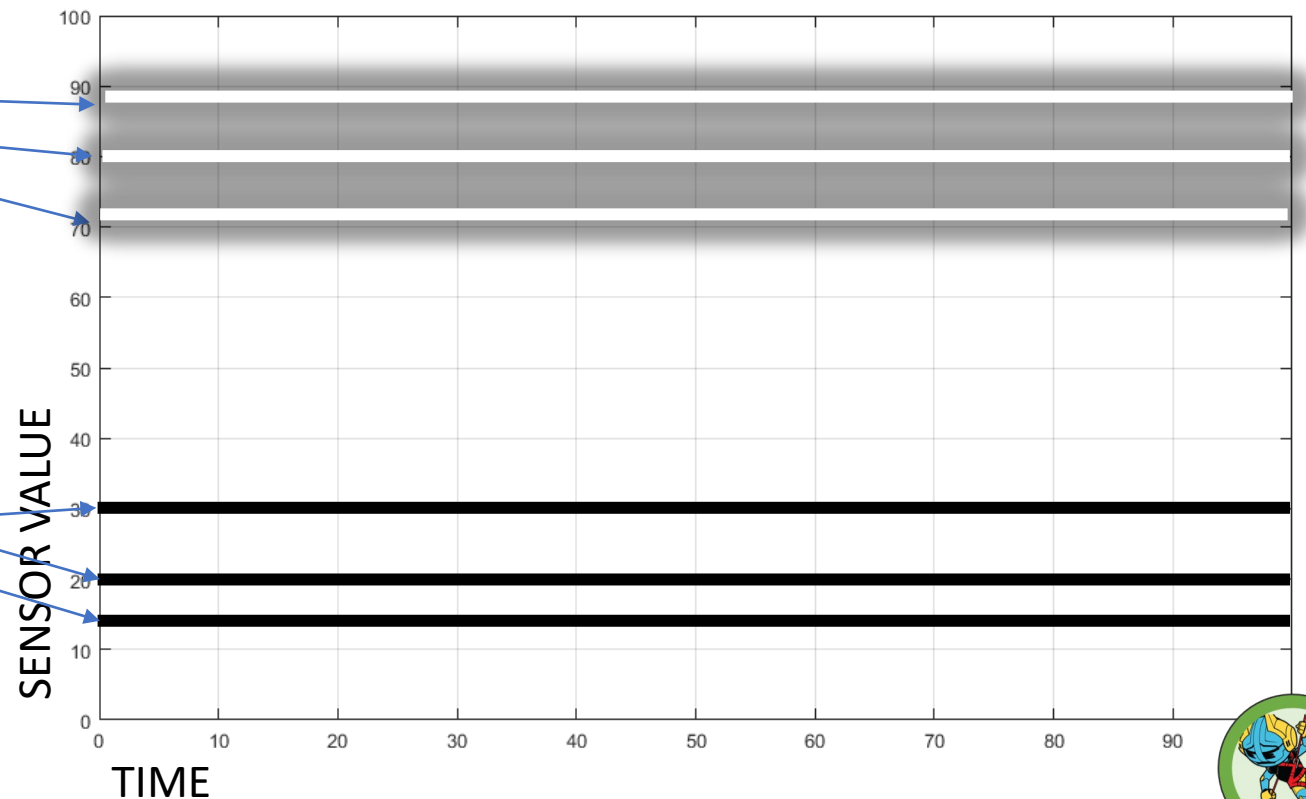
L

L

Sensor Reading for
WHITE

Sensor Reading for
BLACK

LIGHT



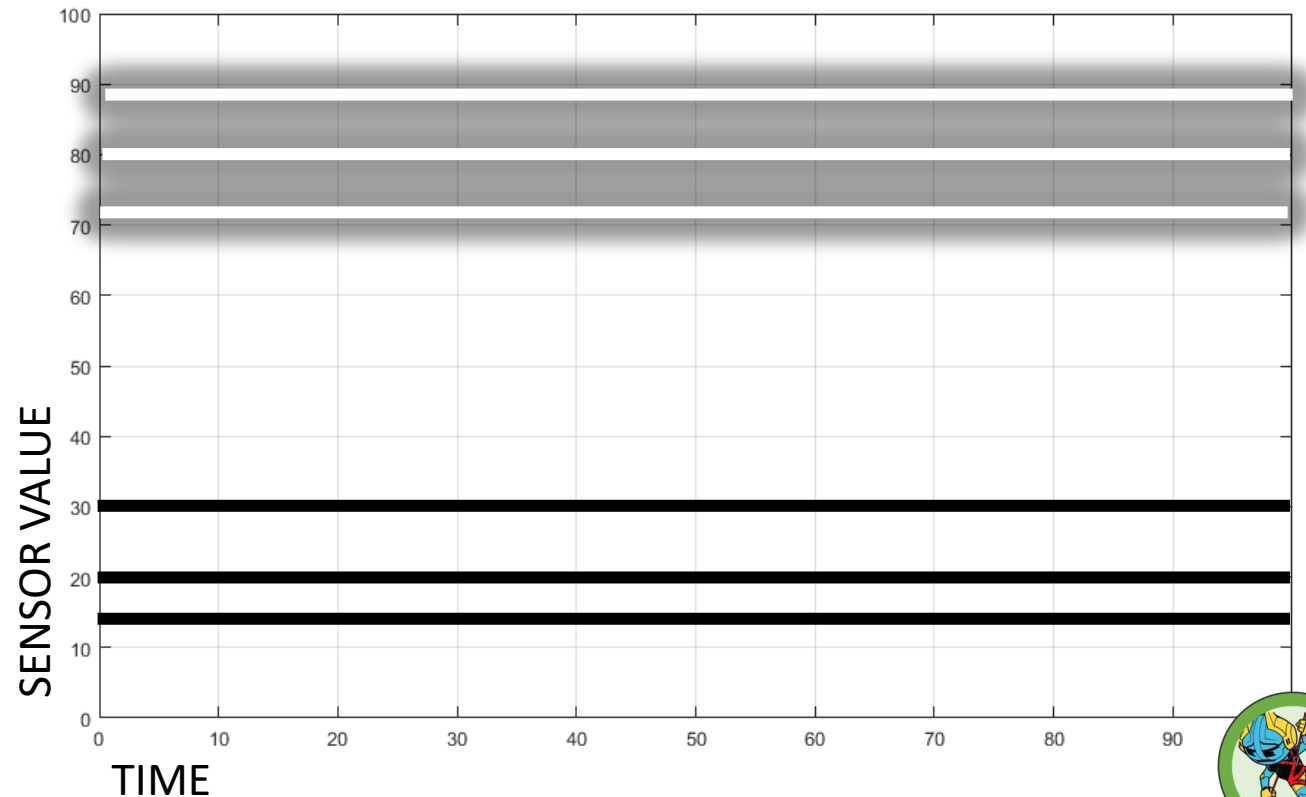
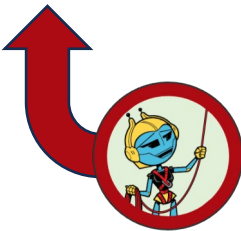
Light Readings



With all these readings (6) and possible movements (3 or 4), we would need many if statements to handle all the exact scenarios – and this would increase with longer tracks. BUT! There is a pattern that appears, and we can use this.



Looking at the graph we can see two groupings. The brighter WHITES tend towards 100, while the darker BLACKS tend towards 0.



Light Readings



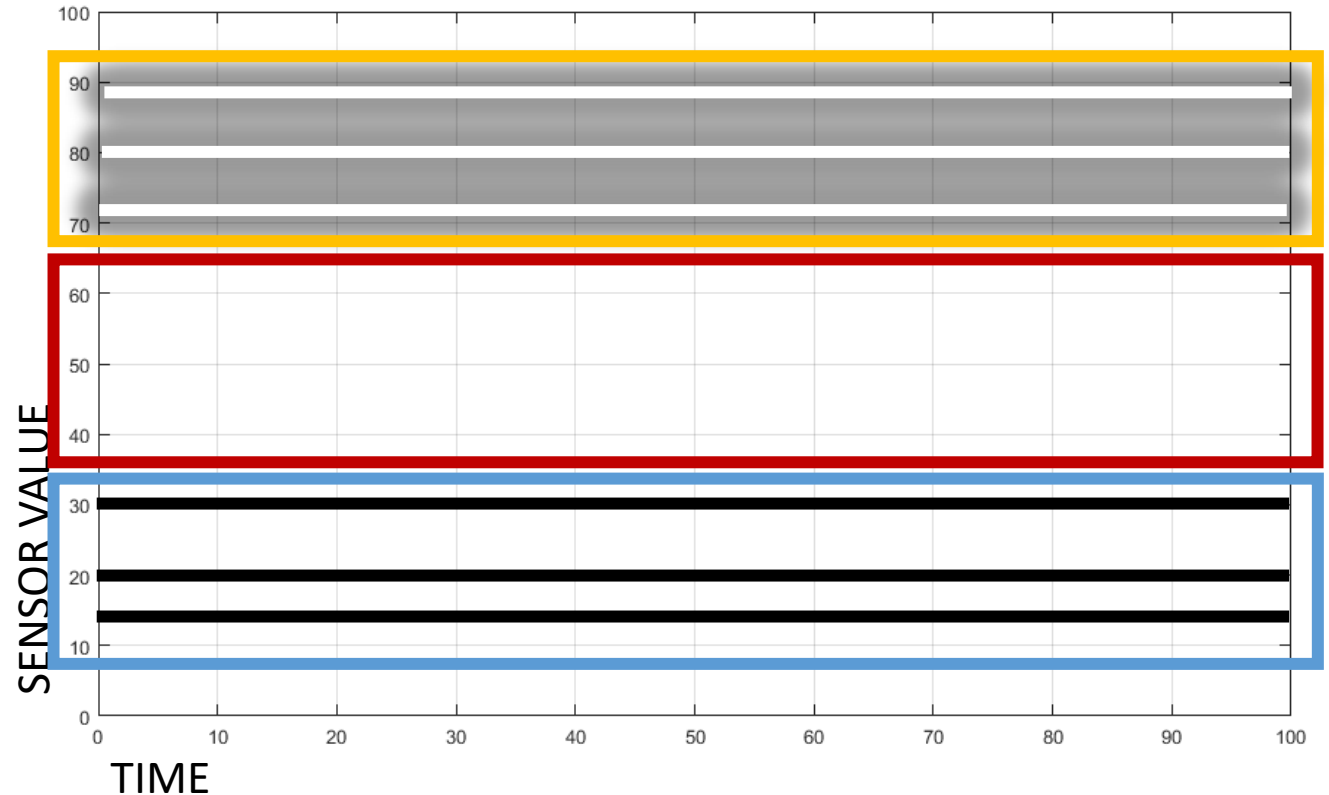
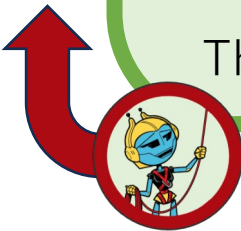
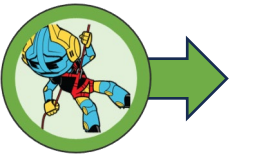
Let's isolate the WHITE readings, we will do this with a **yellow** box.

Now, let's isolate the BLACK reading, we will do this with **blue** box.

By isolating each set of readings, we can now see an area between, where above is WHITE and below is BLACK, we will mark this in **red**.

So, if we were to pick a value somewhere in the **red**, we now have a single number that separates WHITE and BLACK.

This is the same logic for an **IF** 😊



Light Readings

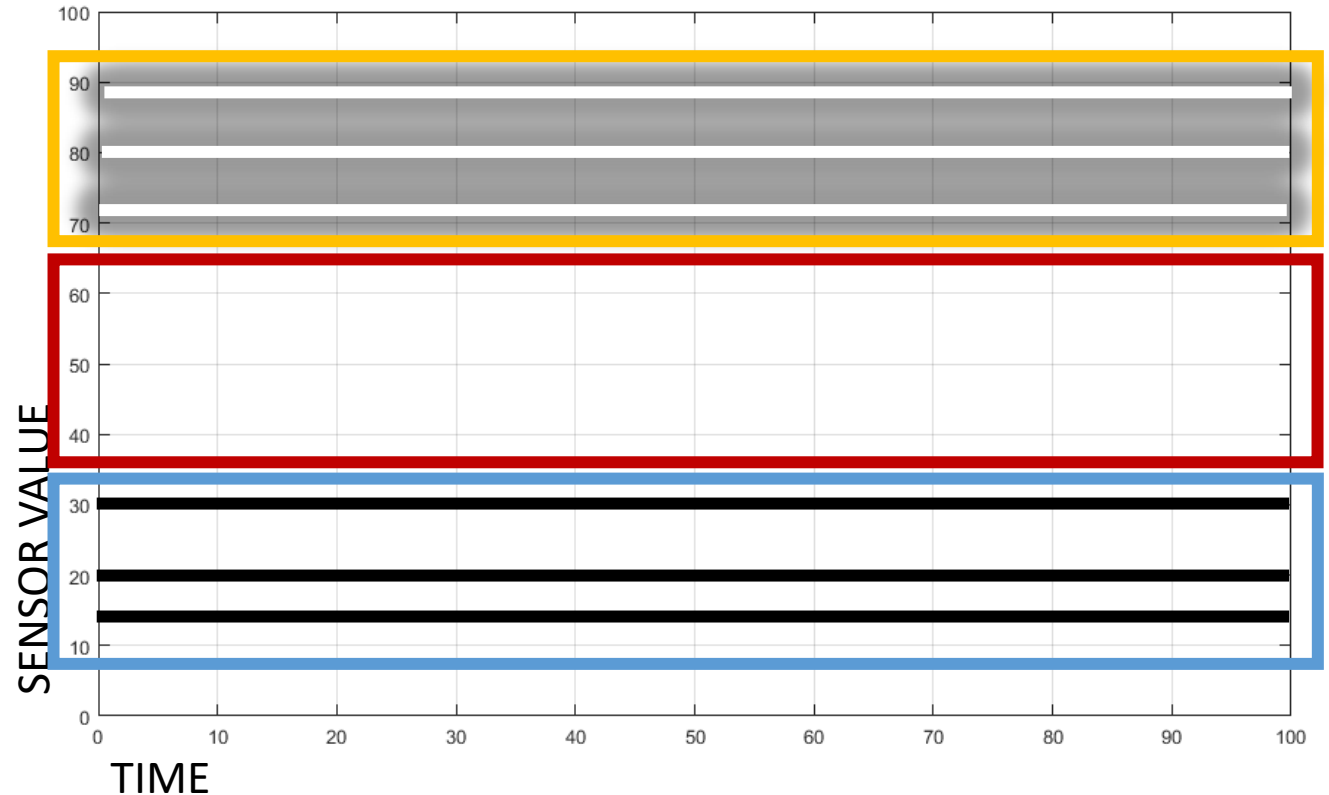
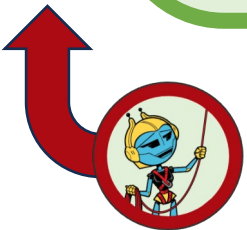


We are now going to look at what using a value from the red zone does for our code logic.

From here on we will be calling this number the **Chosen Value**.

Note: depending on yours or your student's level of coding and maths, this **Chosen Value** can be calculated using an average of BLACK and WHITE light readings.

OK, back to it, for now I'll be fading out the colored boxes.



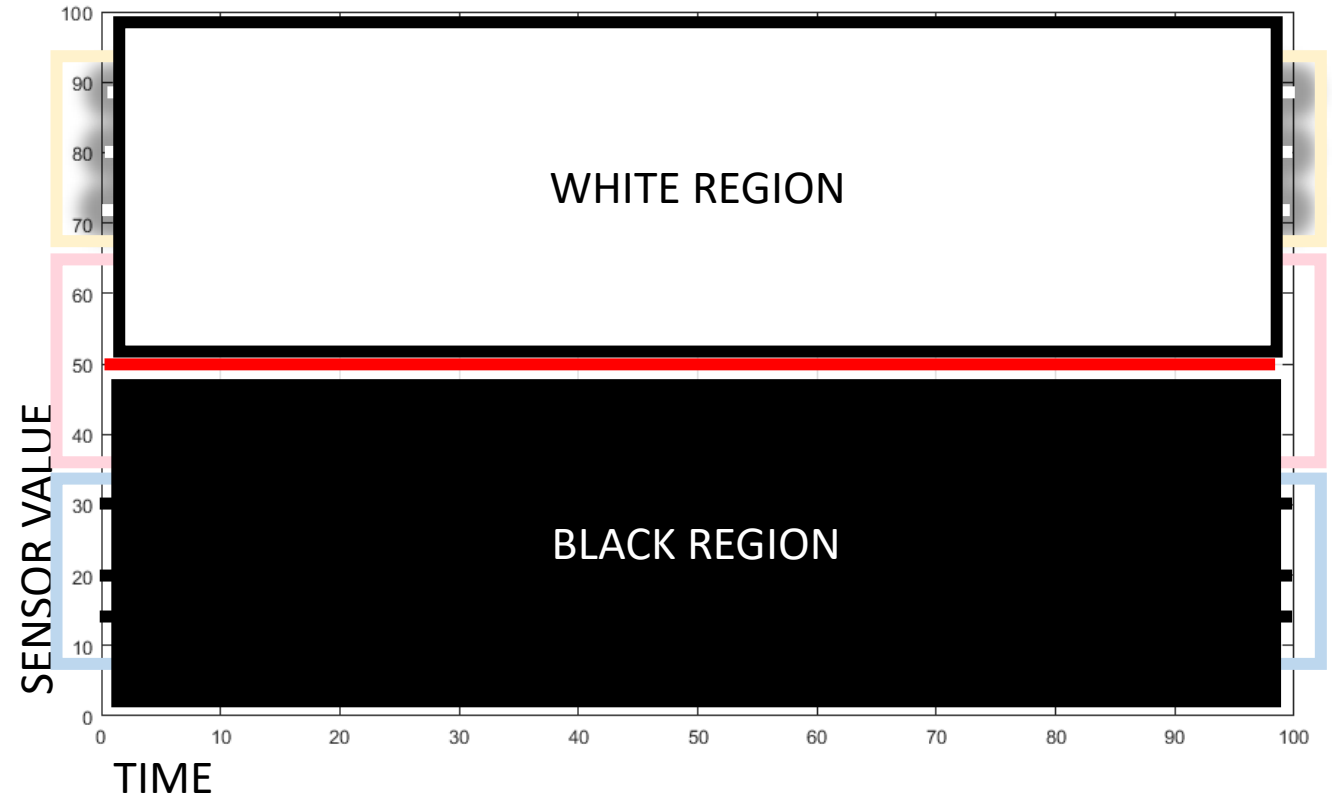
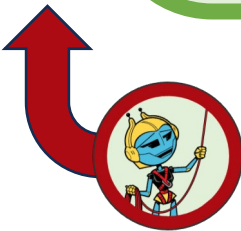
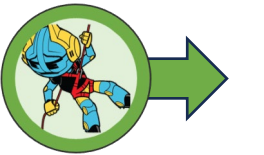
Light Readings



To start off with, we need to pick a number in the red zone that somewhat splits the two areas apart. I am going to go with **50** as our **Chosen Value**.

With this line added, I'll also show the WHITE and BLACK Regions, rather than the values.

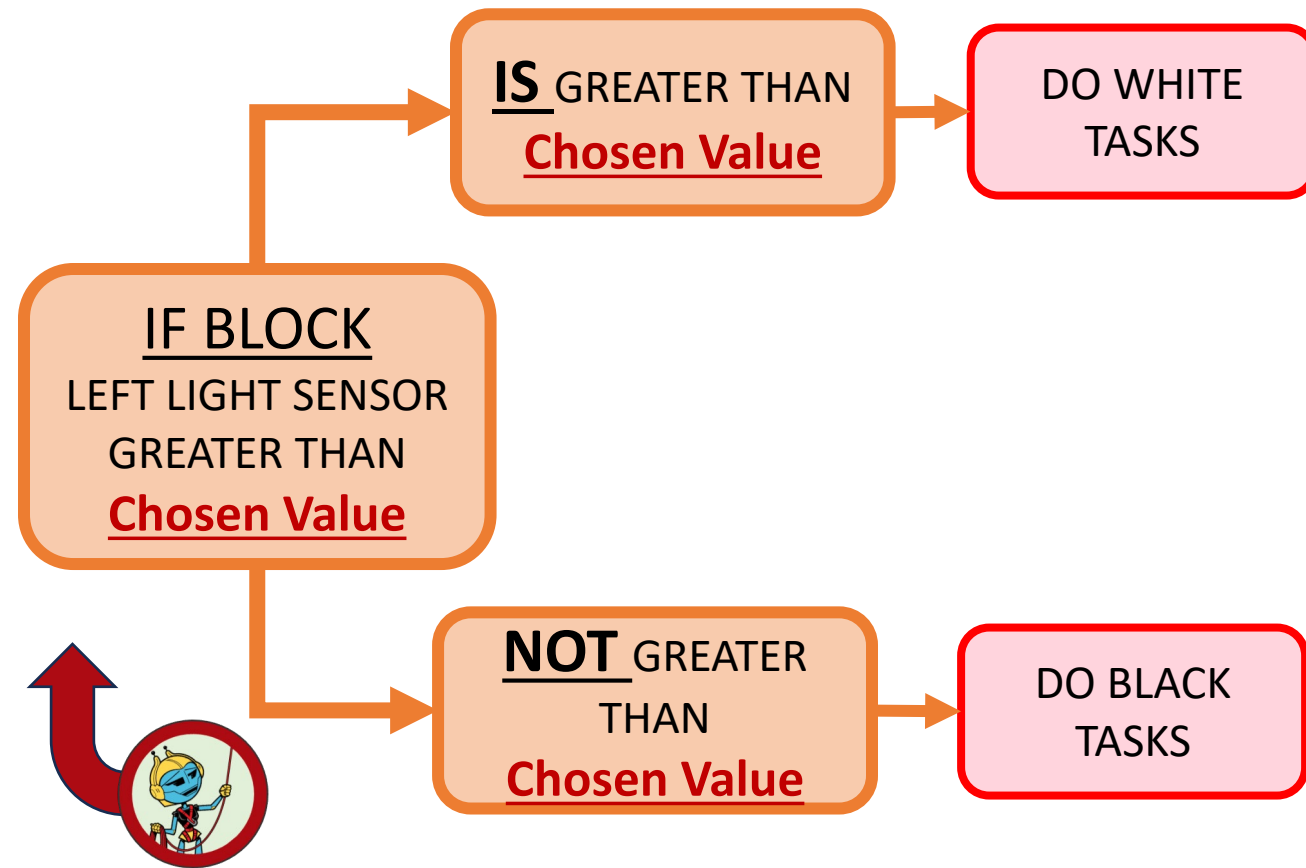
Now we can see two areas that **IF** greater than the **Chosen Value** we are seeing WHITE and **IF** less than the **Chosen Value** we are seeing BLACK.



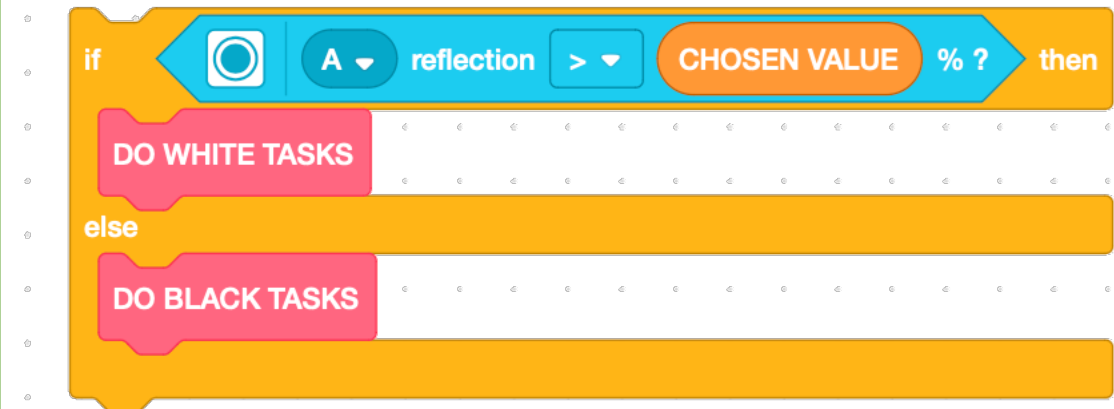


Let's have a look at how we can use our **Chosen Value** in code. For this example, I'll use blocks from the SPIKE Word Blocks and will also show a general diagram for a single sensor check. The logic will translate to all coding languages.

THEORY



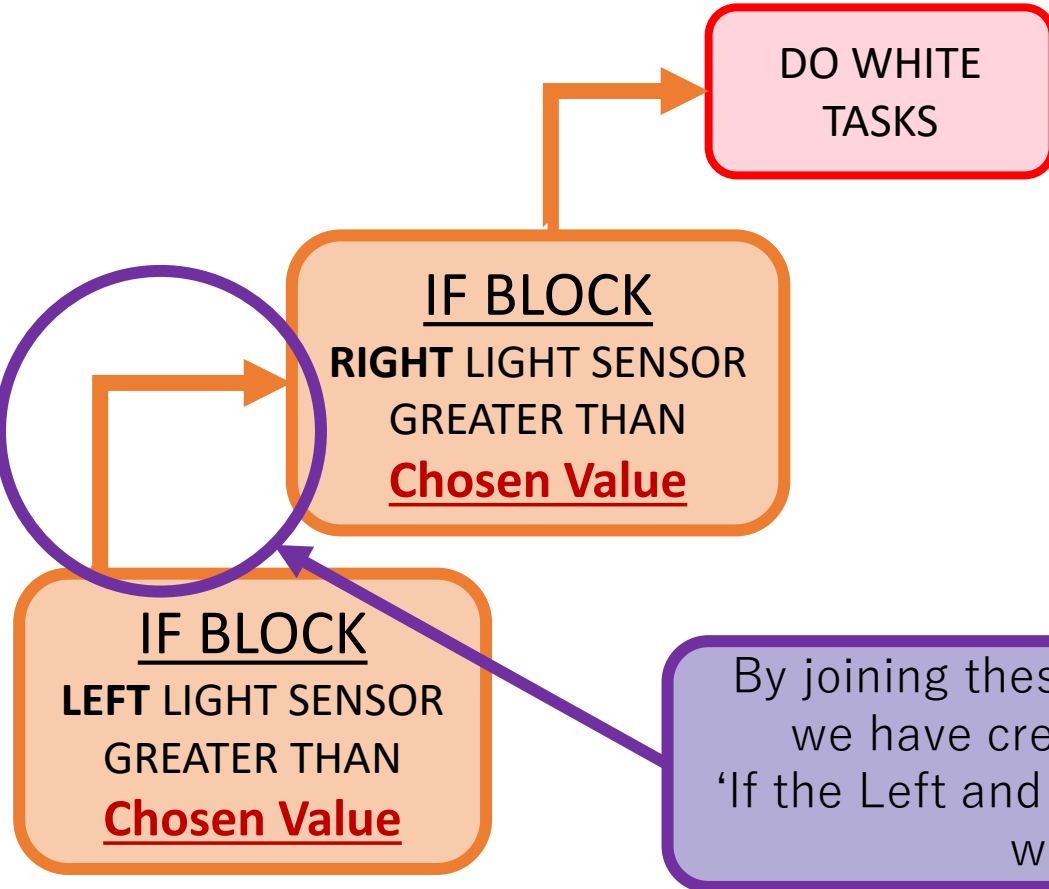
SPIKE Word Block





We're almost there! The last bit I want to show is how to incorporate the second sensor, so we can use our Truth Table and get the robot going! For this we are going to focus on both sensors seeing WHITE, where the orange arrows now show greater than **Chosen Value**. You can also repeat the methods for **Chosen Value** for the second sensor.

THEORY



What we are doing here is called nesting, this means we have an **IF** statement within an **IF** statement. This allows us to implement AND conditions, rather than having separate statements for separate movements. You can also use the AND logic block inside an **IF**.

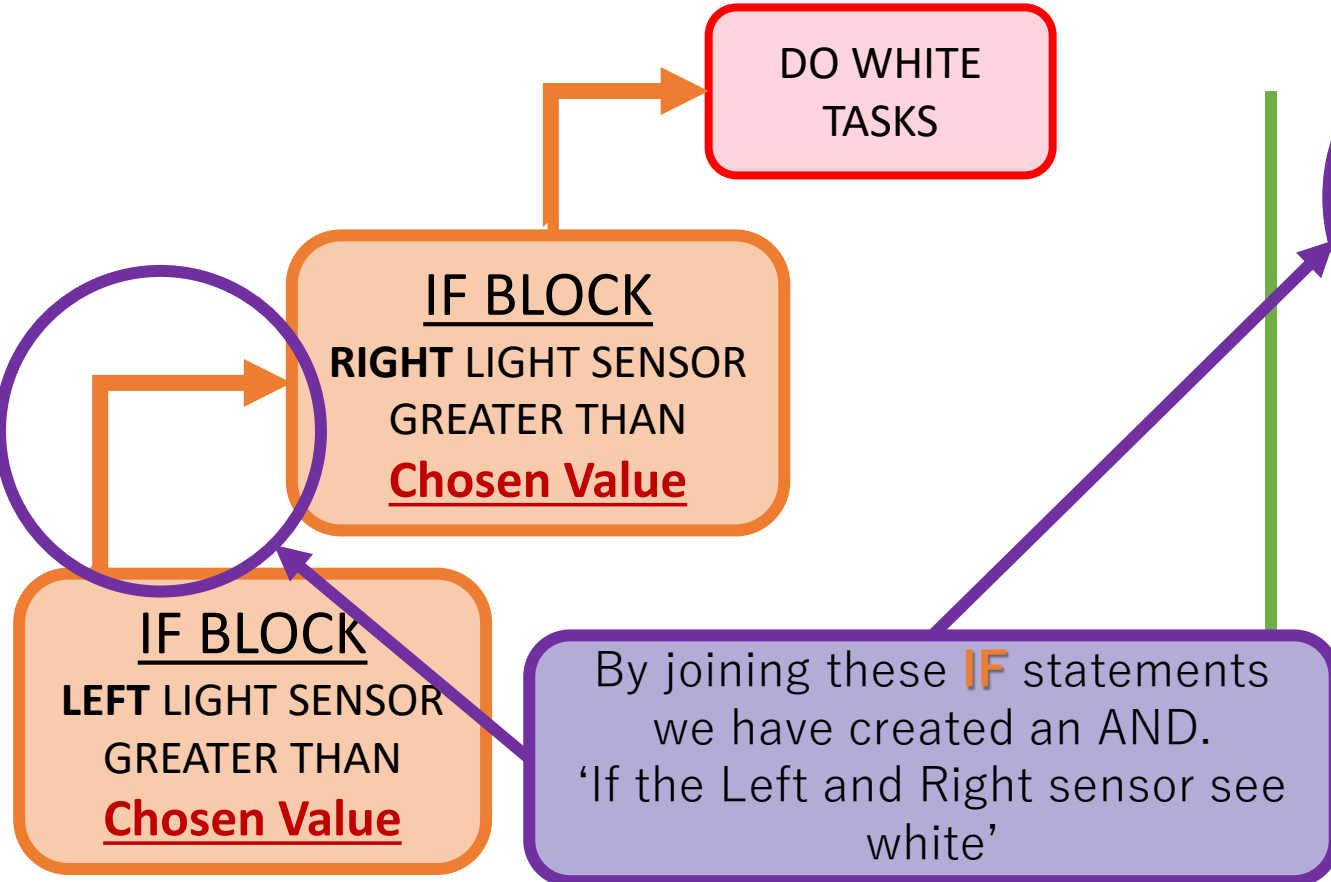
By joining these **IF** statements we have created an AND.
'If the Left and Right sensor see white



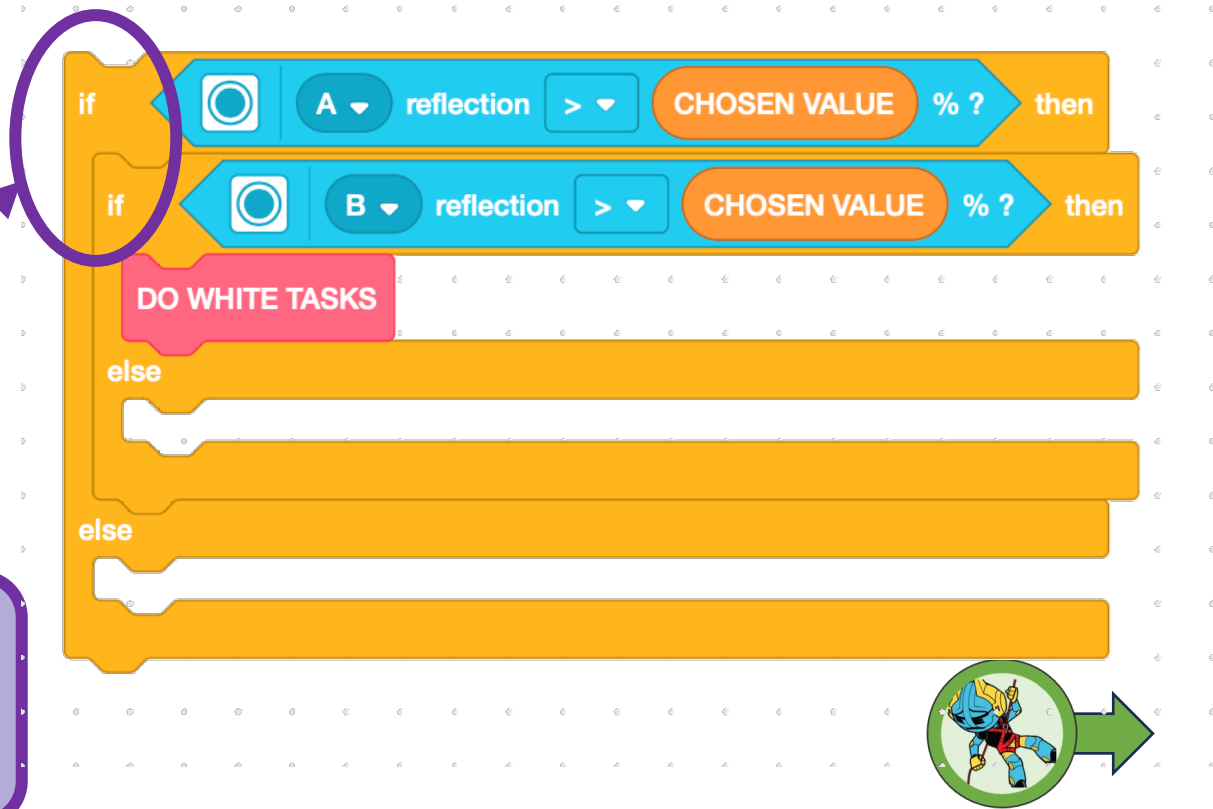


Now we will look at how it comes together. The structure is the same as our previous example, but now we are adding our nested **IF** (AND). Remember the Spike code will read from out to in top to bottom, so the outside **IF** statement will be checked first.

THEORY



SPIKE Word Block





And there we go! That is the basic concepts behind a two-sensor line follow using nested **IFs**.

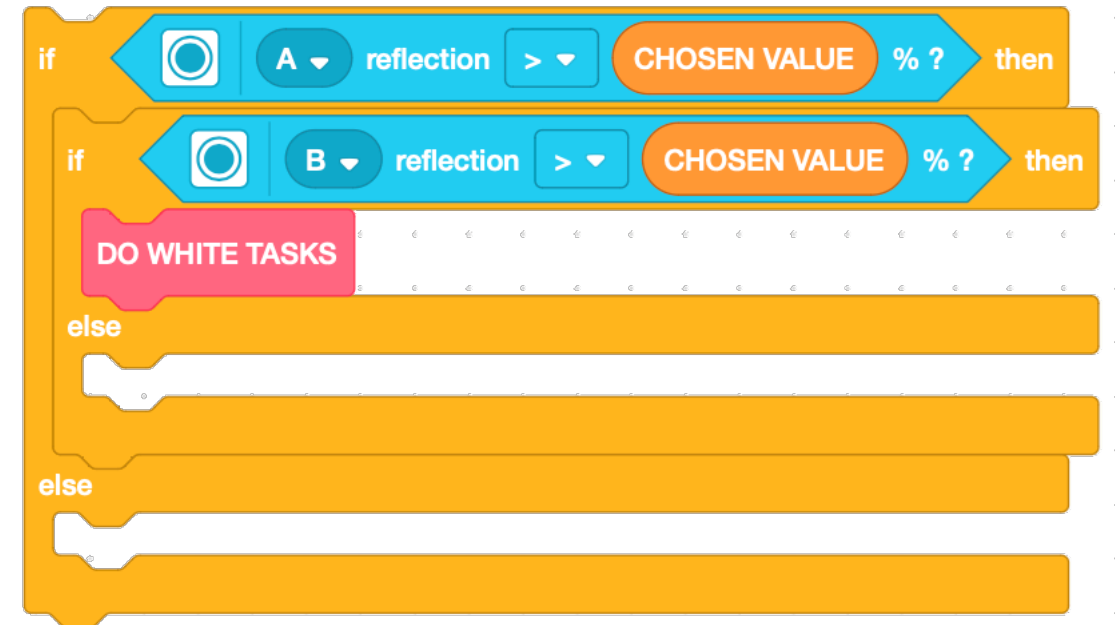
At this stage we have covered all the theory and most of the framework to construct your code.

Using the example on the right, you can now add in another nested **IF**, in the ELSE of the first **IF** statement, and the relevant motor movement blocks to get your code going. Use the truth table to help you do this.

Remember to write your **Chosen Value** where it says 'Chosen Value' in the code example.



SPIKE Word Block



A blue and yellow robot is shown climbing a red rope. The robot has a blue body with yellow joints and a yellow head. It is holding the rope with its hands and feet. The background is a solid grey color.

Workshop: Proportional Control Line Following

Basic Principles: Breaking down the problem

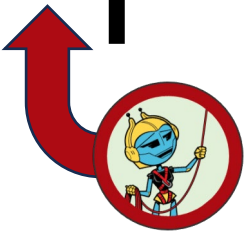


Hey there! In this presentation I'll take you through some basic concepts for avoiding an obstacle.

All you'll need to do is click to go to the next slide and I'll pop up along the way. You'll see me appear at the bottom with an arrow. This means you can advance to the next slide, when ready.

If at any point you want to go back to the "Code Examples" index slide, click on my friend on the left of the screen.

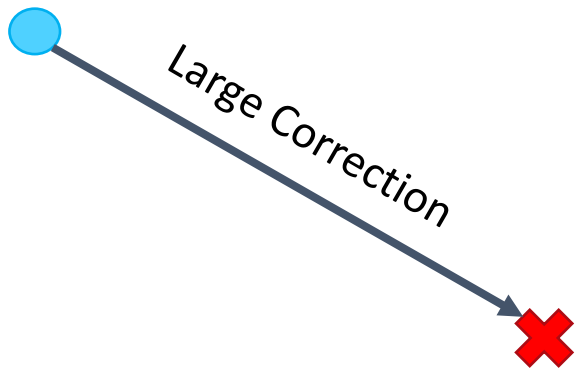
Best of luck and remember you can always reach out to your local RCJA committee for help.



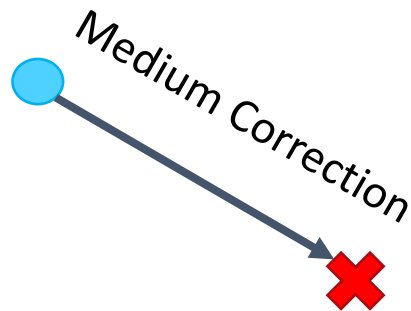


Proportional line following is a way to have a robot change its turn, speed or any set value (correction), based on how far away it is from a target. Basically, if you are far away from where you want to be then you need a bigger correction. In the image below, the **circle** is where we are, and the **X** is where we want to be.

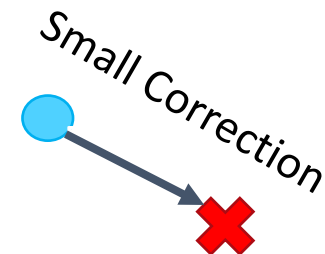
Far away



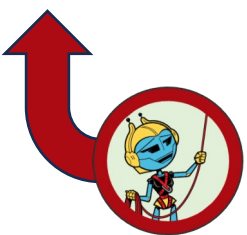
Some what away



Close



What we need now is a response that takes into consideration how far away we are from where we need to be – hence the basis for proportional control.





Now we have a trend, we can start to build an equation. Let's look at the same examples but this time on a graph. The **circle** is where we are, and the **X** is where we want to be.

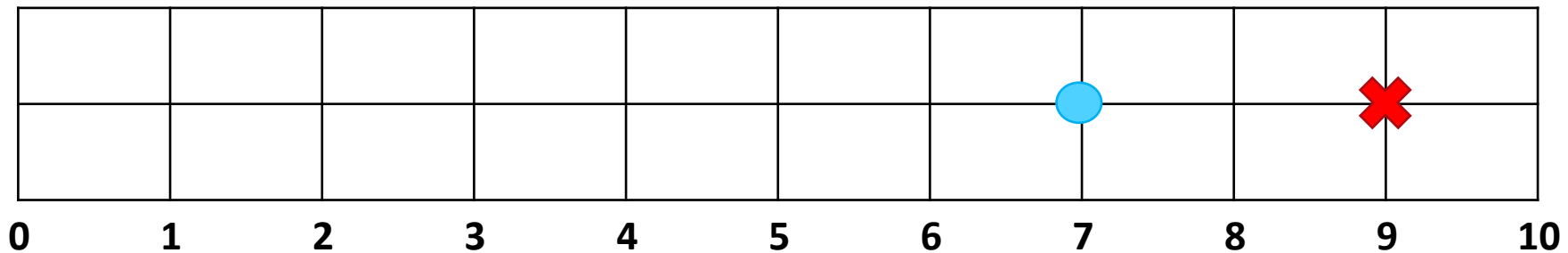
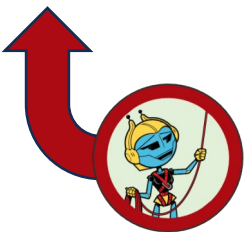
F
A
R



M
I
D



N
E
A
R

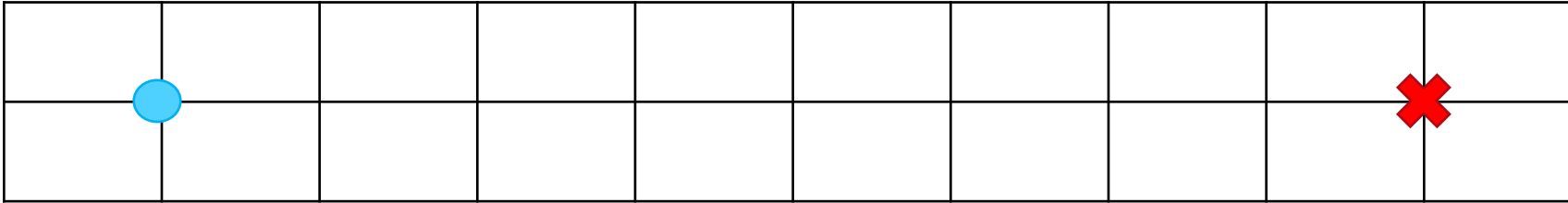




What we need now is to calculate the difference between **where we are** and **where we want to be**. We can do this by subtracting the **X** location from the **circle** – known as error.

EQUATIONS

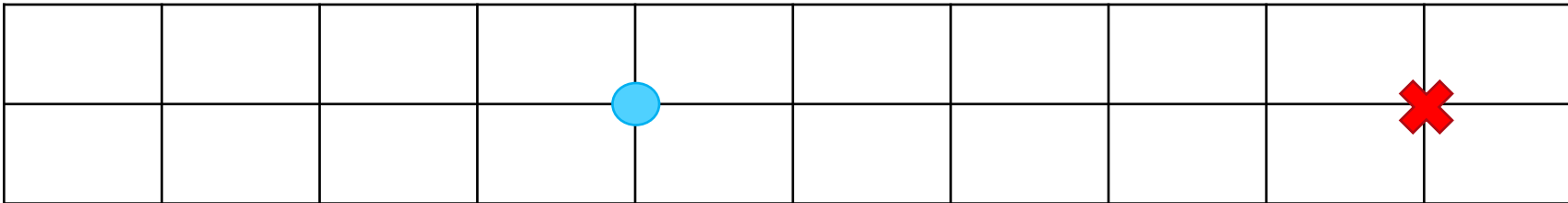
F
A
R



$$9 - 1 = 8$$

We are 8 units away
from our target

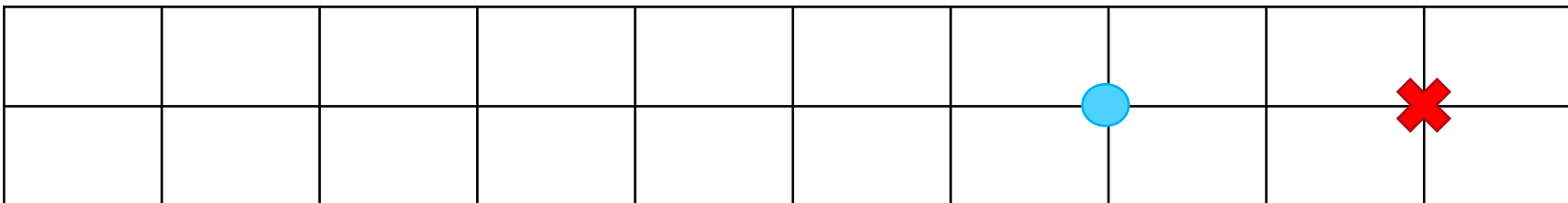
M
I
D



$$9 - 4 = 5$$

We are 5 units away
from our target

N
E
A
R



$$9 - 7 = 2$$

We are 2 units away
from our target

0 1 2 3 4 5 6 7 8 9 10

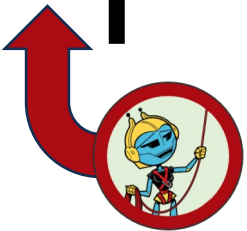




We now have a way of looking at our position and then finding out a number that represents our response.

In these examples this was a unitless response, and just by it being bigger meant we had a bigger response.

But now we want to explore some more specific Robocup scenarios. For this we will look at following a side of the line with a single sensor, that way we have white on our left and black on our right.

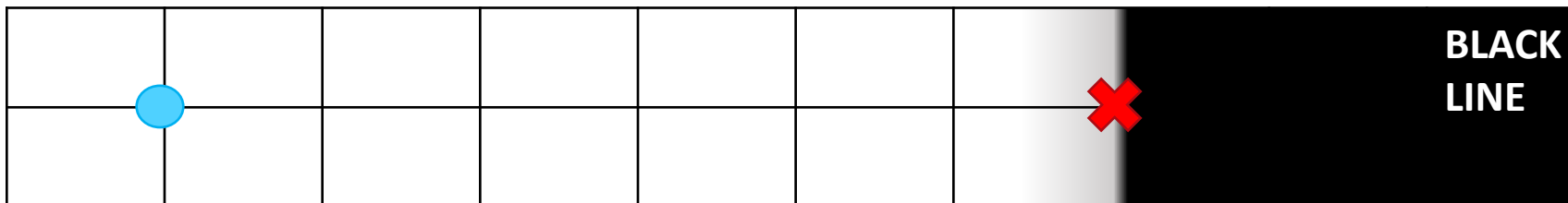




We will use similar graphs but this time we will add a black line and light readings on the X-axis. This line has an exaggerated transition from white to black. The target value is the value midway between the black and white values, essentially the edge of the line.

EQUATIONS

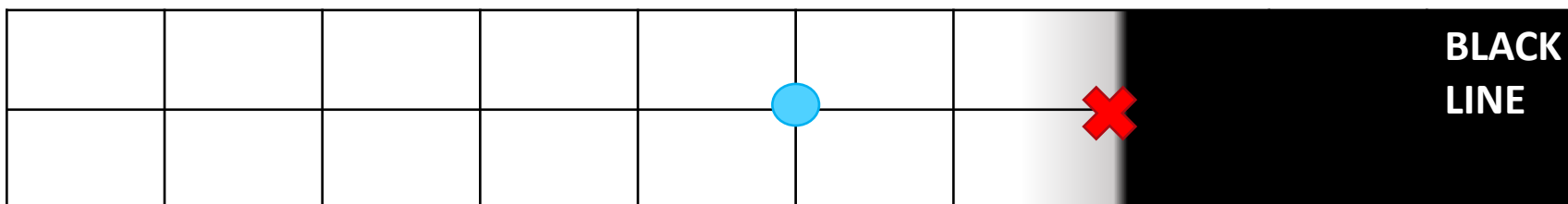
F
A
R



$$54 - 74 = -20$$

We are -20 units away from our target

N
E
A
R



$$54 - 64 = -10$$

We are -10 units away from our target

P
A
S
T



$$54 - 30 = 24$$

We are 24 units away from our target

75 74 72 69 68 64 58 54 36 30 30

LIGHT READINGS (%)





Using our numbers from the calculations, we can start to think about controlling the robot. I am going to keep the equations on the side for now and talk about robot control.

EQUATIONS

F
A
R

$$54 - 74 = -20$$

We are **-20** units away from our target

N
E
A
R

$$54 - 64 = -10$$

We are **-10** units away from our target

P
A
S
T

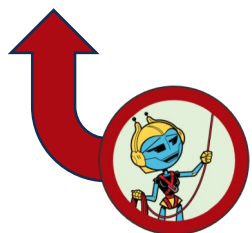
$$54 - 30 = 24$$

We are **24** units away from our target

A typical robot will use the light sensors to check the track and then motors will react.

When proportional control isn't used, the robots have set turn angles and speed, but in proportional control we want to adjust the turn angle based on a position relative to the target.

We could just plug how far away we are from the target into a turn block, but this can have issues, let's explore.





Let's investigate the range of a light sensor compared to motor. See below:

EQUATIONS

$$54 - 74 = -20$$

We are **-20** units away from our target

$$54 - 64 = -10$$

We are **-10** units away from our target

$$54 - 30 = 24$$

We are **24** units away from our target

SPIKE Light Sensor

RANGE 0 to 100%



WORKING 30 to 75%

SPIKE Motor

-180° to 180°



-180° to 180°

Now there is one thing we need to consider. The light sensor has a range of 0 to 100% but often in line following we use a much smaller **working range**. I'll add an example, but yours may differ.





Now we will look at our **WORKING** range and our motors. Let's see what happens if we were to use the values calculated in our equations from before.

EQUATIONS

$$54 - 74 = -20$$

We are **-20** units away from our target

$$54 - 64 = -10$$

We are **-10** units away from our target

$$54 - 30 = 24$$

We are **24** units away from our target

SPIKE Light Sensor

WORKING 30 to 75%

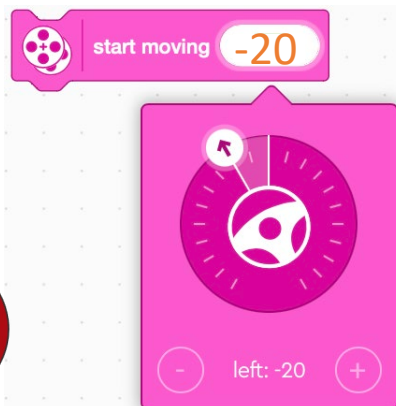


SPIKE Motor

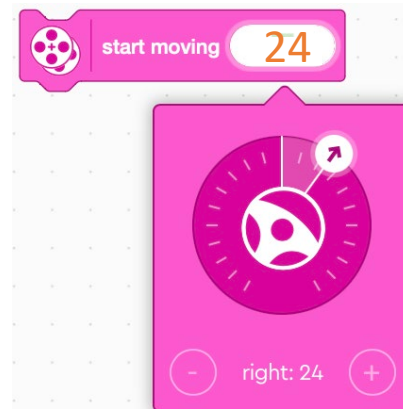
-180° to 180°



FAR



PAST



If we take any of the **calculated values** and plug them into the motors, we will get small turns. This is a problem. The corrections required for our extreme cases are much larger.





We now better understand looking at our position and then finding a number that represents our response.

However, we now know that while we can calculate a difference, this value is not in the same range needed. That is, light sensors have a smaller range of values compared to a motor. So even in our far case we are still calculating a small number – not great for sharp turns.

The way around this is to use a multiplier, or gain. This constant value is multiplied by the calculated value to move them into a similar range to the motors. Let's investigate.





Let's see what happens now. We will take the turning values we had before but this time we will include a multiplier of 3.2. Note, multiplier is value that needs to be tuned.

EQUATIONS

F
A
R

$$54 - 74 = -20$$

We are -20 units away from our target

N
E
A
R

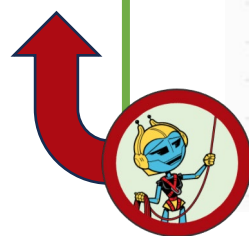
$$54 - 64 = -10$$

We are -10 units away from our target

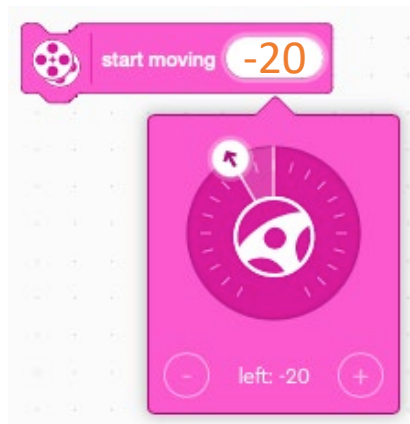
P
A
S
T

$$54 - 30 = 24$$

We are 24 units away from our target



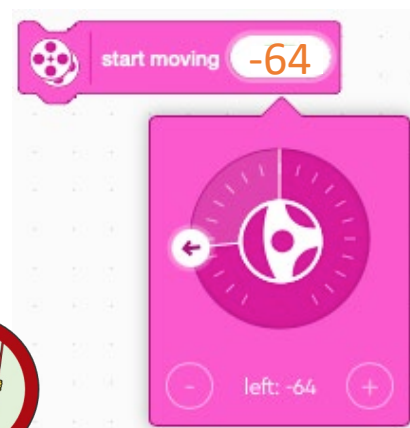
FAR



PAST



× 3.2





By using the multiplier of 3.2 we have now made a proportionally larger output (motor movement) from our input (sensor readings).

EQUATIONS

F
A
R

$$54 - 74 = -20$$

We are -20 units away from our target

N
E
A
R

$$54 - 64 = -10$$

We are -10 units away from our target

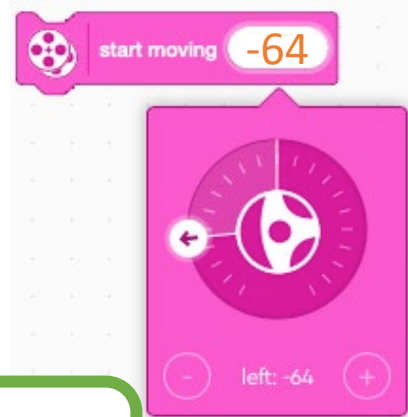
P
A
S
T

$$54 - 30 = 24$$

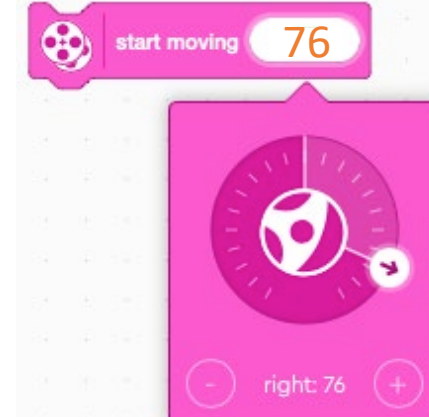
We are 24 units away from our target

× 3.2

FAR

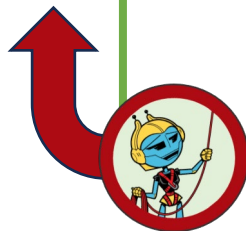


PAST



Logically, we are now saying if the robot moves away from our target, the edge of the line, then increase the turn so the robot turns towards towards the edge of the line proportional to how far away it is.

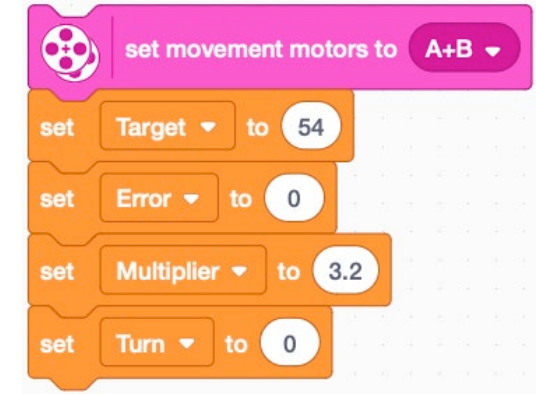
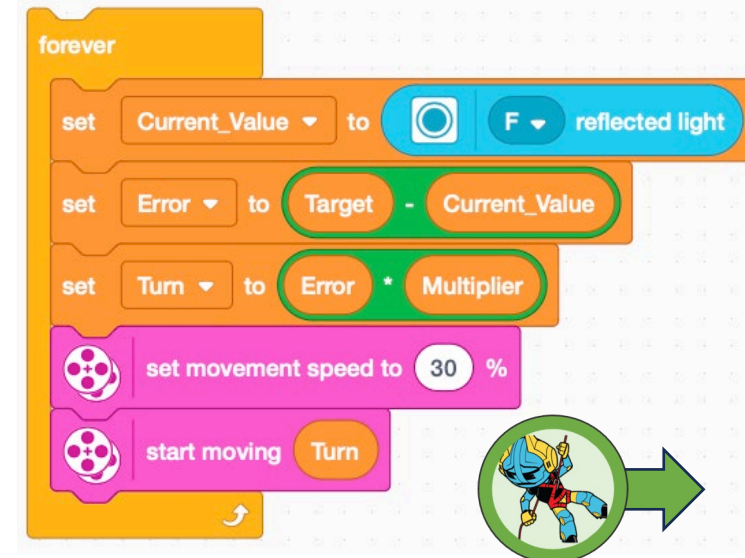
This logic will work because we have WHITE and BLACK on either side of the robot. Meaning our equation will swap from negative to positive, changing our turn direction - you can see this in the FAR and PAST example.





Now we want to look at the code for this. We will set this up in two blocks and they can be joined when you're ready to run this on your robot.

1. We first want to create some variables: Target, Error, Multiplier, Turn and Current_Value. And set our motors, for this example we will use A and B
2. Set:
 - **Target** – To the light reading when your robot is on the edge of the line. In the example this was 54.
 - **Error** – To 0 at the start of the code. This gives it an initial value.
 - **Multiplier** – To a multiplier you know works; we can start at 3.2.
 - **Turn** – To 0 at the start of the code. This gives it an initial value.
3. Now in a forever loop we want to calculate the error and use this and the multiplier to determine the turn value.
 - **Current_Value** – Set by the current light sensor reading, port F in this example
 - **Error** – Set to **Target** minus **Current_Value**; equation: **Target – Current_Value**
 - **Turn** – Set to **Error** multiplied by the **Multiplier**; equation: **Error * Multiplier**
4. Finally, we just need to add the motor movement, speed and direction to our forever loop. **Speed** can be set to any speed, for now we will select 30. The motors' direction will be set to our calculated **Turn** value.





The previous slides gives you the basics to get proportional control for a single sensor line following going. But there is some tuning needed.

There is no perfect way to tune this type of line following, and it is something that will take time to get right. You are trying to find the balance between all your variables

In the next example we will discuss your target, multiplier and speed, note all of these affect each other. So, any change will have an effect on the robot and require tuning of the others.





Here is some understanding and what to look out for when tuning your proportional control line following.

MULTIPLIER

Multiplier adjusts the response of the motor proportional to the change in light sensor reading.

Try a few different values for Multiplier and see what happens.

If your robot goes crazy and its turns are very harsh or it keeps spinning, then your Multiplier is likely too large.

If the multiplier is too low, then your robot won't make it around corners or will not find the line quick enough.

If your robot turns the wrong way, try a negative multiplier

SPEED

Speed is the max speed of your robot; Spike Prime will make adjustment when turning as needed.

Try a few different values for speed and see what happens.

Now, if you adjust speed, you will also need to adjust Multiplier. Alternatively, if you are happy with your speed, then you can just adjust Multiplier.

TARGET

Target is the value you are trying to maintain.

Try taking your target value from a few different locations along the line and see what happens.

If your robot is, for example, better at responding to white than black, then it means your target might be too close to black. This is because error isn't becoming larger enough to create a sharp turn.



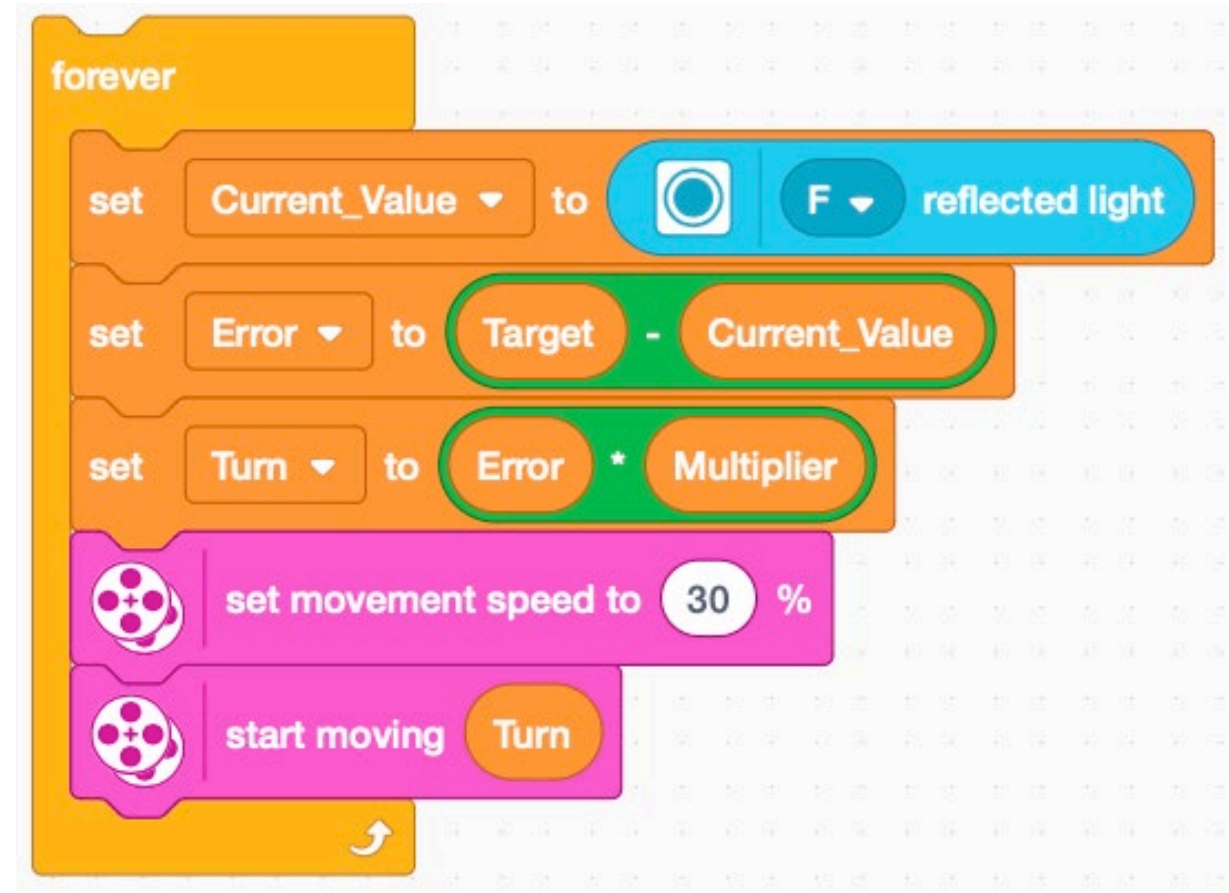


What you now have is the basis to build and tune a single sensor proportional control line following robot. You also have a list of points to look out for when tuning your system.

But what you also have is a very powerful piece of code within, see on the Right.

This is the backbone of proportional control, and this paired with tuning can be adapted for any scenario or sensor.

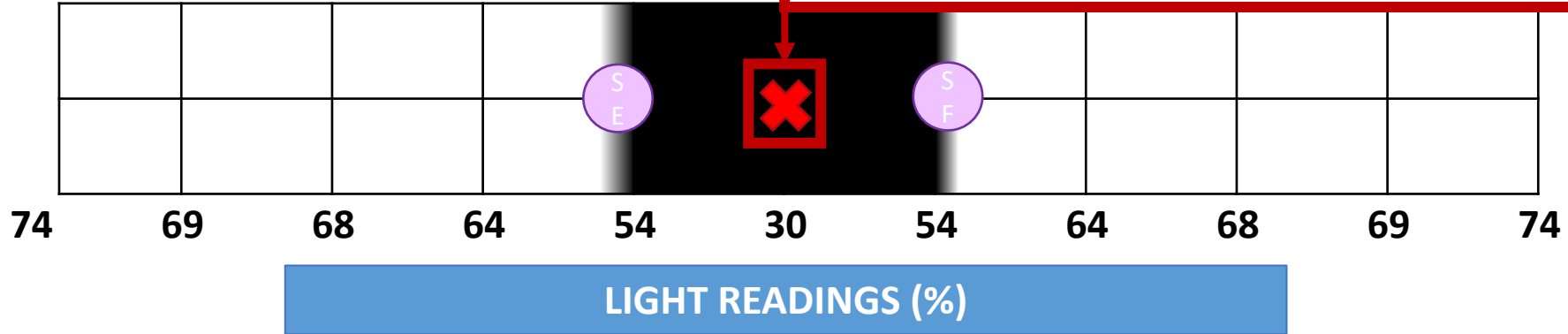
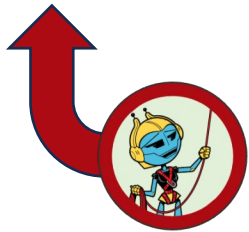
Let's investigate this for two sensor line following.





If we take the same line as before but this time we have a sensor on either side, **Sensor E (SE)** and **Sensor F (SF)**.

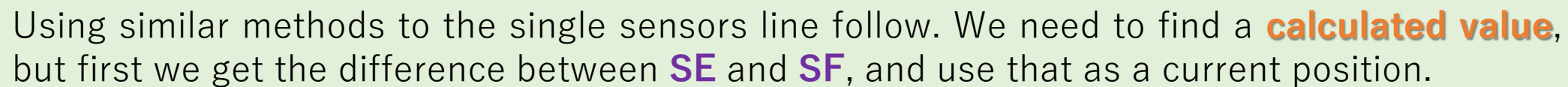
Target: $SE - SF$, When sensors either side of line



The logic for this is almost the exact same as what we already have, except we need a way to express **SE** and **SF** as a single number. We do this by subtracting **SE** from **SF**. This now gives us a way to compare a single number to a target.

In an ideal scenario, **SE** and **SF** would read the same value. So, if the robot is facing straight, then the difference in **SE** and **SF** would be zero. Hence, we can also say we want our **Target to be zero**. You will need to check your own sensors and pick an appropriate Target (not all sensors are perfectly matched).





A diagram illustrating a 2D environment with a black barrier. The environment is represented by a grid of white squares. A black barrier is positioned in the center, separating the left and right halves. On the left side, a purple circle labeled 'S' and 'E' is located. On the right side, a purple circle labeled 'S' and 'F' is located. A blue circle is positioned in the middle, between the barrier and the 'S' 'F' circle. A red 'X' is located between the blue circle and the 'S' 'F' circle.

The diagram shows a 2D grid with a black obstacle region in the center. A blue circle is at the bottom center, a red 'X' is above it, and two purple circles labeled 'S' and 'F' are on either side of the obstacle.

A diagram showing a 2D grid with a central black barrier. To the left of the barrier, there is a purple circle labeled 'S' above 'E' and a red 'X' mark. To the right of the barrier, there is a purple circle labeled 'S' above 'F' and a blue circle.

79 69 68 64 54 30 54 64 68 69 74

LIGHT READINGS (%)

EQUATIONS

$$1. SE - SF = 62 - 30 = 32$$

$$2.0 - 32 = -37$$

1. $SE - SF = 54 - 54 = 0$

$$2. \textcolor{red}{0} - \textcolor{blue}{0} = \textcolor{brown}{0}$$

$$1. SE - SF = 30 - 62 = -32$$

$$2.0 - -32 = 32$$

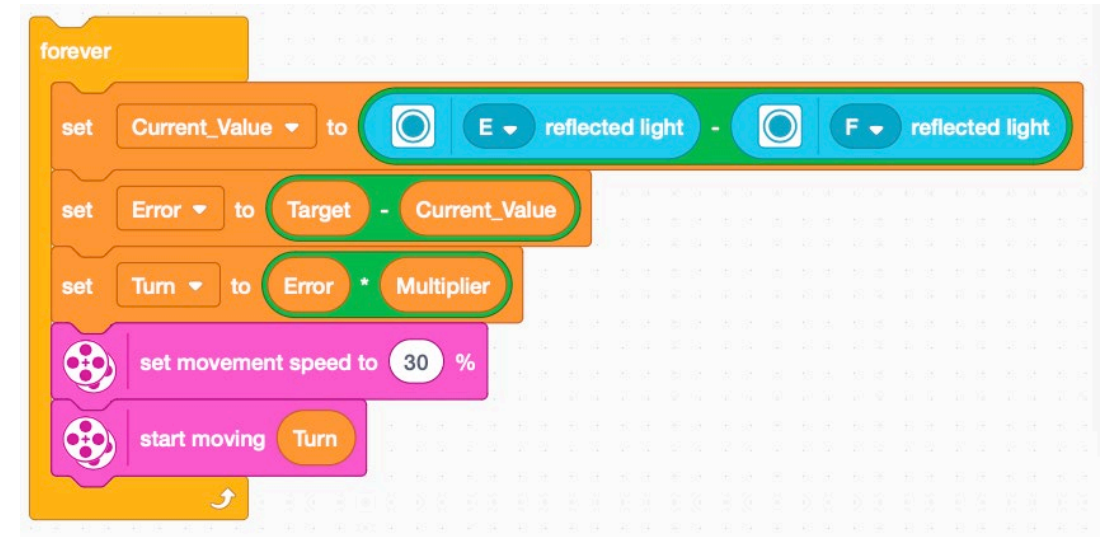




This Proportional algorithm is very powerful. If you can find a way to summarise multiple sensors as a single value, like we did for two sensor proportional line following, you can use the same code for many scenarios.

You now have the basics for getting started with proportional line following, for both single and two sensor line following.

Good luck and happy coding!





YouTube Series

Riley Rover using similar methods

Video series



There are many ways to solve Riley Rover and we have given you some examples so far. To further help you along your way we also have some links to a web series that solves the same problems we have gone over but in a slightly different way.

You are welcome to go with any solution that works best for you.

The next slide is the last slide in this section. On this slide I have included buttons with links to the web series videos.

Happy watching and coding!



Technical Video Links

Technical Video:
Line Following



Technical Video:
Bumps & Bridges



Technical Video:
Avoid Obstacles



Technical Video:
Sensor Height





Problem Solving: Some Simple Checks



Robots are temperamental, and always seem to play up right when you don't want them to. But it is important to know, this is completely normal, and we can normally work out what's wrong pretty quickly.

I have consolidated some initial checks you can do when things don't quite go right.

Always remember to keep calm and take your time. Tip: photos of this problem solving is great for your logbook.

Good luck! 😊



Some Problem-solving checks

General:

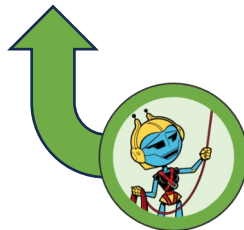
- ☐ Are all my clips and beams pressed in?
- ☐ Did my code download and was it the right code?
- ☐ Have I selected the right ports?
- ☐ Are all my ports plugged in?
- ☐ Is my battery going flat?

Line following

- ☐ Am I on the right side of the line? [Single Sensor]
- ☐ Are my sensors too low or too high?
- ☐ Have I checked my sensors readings?
- ☐ Have I put my left and right turns in the correct spot?
- ☐ Is my multiplier too high or low?
- ☐ Is my target correct?
- ☐ Does it work better if I slow down?

Obstacles, Bridges and Bumps:

- ☐ Am I getting too close to the bottle?
- ☐ Do my wheels have enough traction?
- ☐ Have I coded all the moves I planned?
- ☐ Is my robot too low or high?



Remember when something goes wrong to always:

1. Pause and take a breath
2. Put your robot back in the position where things went wrong.
3. Think about your robot and what the problem could be.
4. Try changing something small in your code or build and change one thing at a time.
5. If that doesn't work, try something else.
6. If the problem keeps on happening, then ask for help 😊