

Encounter
Edu

Code Smart

Computing | Ages 11 - 14

Teacher Book



Makeblock

OXBOTICA



A resource by Encounter Edu

Digital Explorer CIC T/A Encounter Edu
Pill Box G04
115 Coventry Road
London, E2 6GG

Text © Digital Explorer 2018

These resources may be reproduced for educational purposes only.

Written by Helen Steer
Edited by Jenny Zhou
Layout and design by Daniel Metson and Riccardo Guido

Acknowledgements

Encounter Edu would like to thank Chip Cunliffe, Ashley Stockwell, and Richard Jinks from AXA XL; Prof Paul Newman, Prof Ingmar Posner, Dirk Gorissen, and Joonas Melin from Oxbotica; Kenny Wang from Makeblock; Helen Steer and Rehana Al-Soltane from Do It Kits; and Nat Hunter from Machines Room.



Sponsored by

AXA XL^[1]

AXA XL¹, the property & casualty and specialty risk division of AXA, provides insurance and risk management products and services for mid-sized companies through to large multinationals, and reinsurance solutions to insurance companies globally. We partner with those who move the world forward. To learn more, visit www.axaxl.com

¹AXA XL is a division of AXA Group providing products and services through four business groups: AXA XL Insurance, AXA XL Reinsurance, AXA XL Art & Lifestyle and AXA XL Risk Consulting.

With thanks to

Oxbotica

Oxbotica, a leading driverless software startup based in Oxford, is developing the brain behind self-driving cars. Our autonomous operating system, Selenium, is platform agnostic and navigates autonomously around cities, warehouses and off-road environments, using data from lasers and cameras placed on vehicles. The software learns from both its own experiences, and those of other vehicles.

MakeBlock

MakeBlock Co., Ltd, founded in 2013, is a leading STEAM education solution provider. Targeting the STEAM education and entertainment markets for schools, educational institutions and families, Makeblock provides the most complete hardware, software, content solutions, and top-notch robotics competitions, with the aim of achieving deep integration of technology and education.

Contents

Foreword	page 1
Overview	page 2
Scheme of work	page 3
Resource guidance	page 6
Lessons	
Lesson 1: Robot cars and smart cities	section 1
Lesson 2: Controlling cars with code	section 2
Lesson 3: Where am I?	section 3
Lesson 4: What's around me?	section 4
Lesson 5: Safety and signalling	section 5
Lesson 6: What do I do next?	section 6
Lesson 7: Designing our smart city pt. 1	section 7
Lesson 8: Designing our smart city pt. 2	section 8
Lesson 9: Designing our smart city pt. 3	section 9
Subject Updates	section 10

Welcome to Code Smart



AXA XL has partnered with Oxbotica in the development of risk solutions related to autonomous vehicle software. Our notion is simple: as a global (re)insurer, building future focused risk solutions is part of our mission.

We're proud to play a role in helping to support and promote autonomous vehicle software and robotics development that will aid our understanding of the risks of the future and help business and society adapt to the ever changing technologies evolving our world.

Educational outreach is a natural extension of AXA XL's commitment. Working with our partners, we believe that an important legacy is the delivery of high quality curriculum-led education programs, that can in turn empower others to contribute.

This Code Smart Teacher Book introduces our Code Smart resources for ages 11 to 14, with the aim of raising literacy in coding and smart cities and inspiring a new generation.

We hope you enjoy the materials in these pages.

Chip Cunliffe
Director, Sustainable Development
AXA XL

OXBOTICA

At Oxbotica we are excited to develop the next generation of autonomous vehicles, creating the software that makes them go. Our world is constantly changing and as engineers we believe that technology has a significant role to play in ensuring that it changes for the better.

Delivering this vision requires individuals with understanding and appreciation of computing and digital literacy. Individuals who inspire and who are inspired. People like you. We are proud to support educational resources that foster and encourage the development of these skills so that, together, we are ready for an ever brighter future.

We are delighted to have you join us in thinking about the future - today.

Ingmar Posner
Co-founder
Oxbotica

Paul Newman
Co-founder
Oxbotica

About Code Smart



What is Code Smart?

Code Smart is a computing and robotics education programme based on driverless car technologies. Students will develop their ability to code as well as their design thinking as they tackle smart city challenges.

This Teacher Book provides a step-by-step guide to using Code Smart in the classroom. Code Smart 11-14 is based on nine lessons, taking students on a journey from complete beginners to code creators. Each lesson provides real-world links and opens students to a range of possible STEM careers.

How do I use Code Smart?

Code Smart works with the mBot STEM robot and the mBlock scratch-based coding platform from Makeblock. Over the following pages, you will find detailed teacher guidance, student sheets and answer sheets, as well as background information in the form of Subject Updates. Further supporting resources such as videos and slideshows are available to download from the Code Smart website.

Is Code Smart curriculum aligned?

Code Smart has been written to align with the Computing National Curriculum in England and the ISTE Standards for the USA. Further details of curriculum and standards alignment are available from the Code Smart website.

Applicable standards

The national curriculum in England

KS3 Computing	Lessons								
	1	2	3	4	5	6	7	8	9
Element of the curriculum									
• Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems	✓	✓	✓	✓	✓	✓			
• Use a programming language to solve a variety of computational problems		✓	✓	✓	✓	✓		✓	
• Use computational abstractions that model the behaviour of real-world problems and physical systems		✓	✓	✓	✓	✓			
• Understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming			✓	✓	✓	✓			
• Understand how data of various types can be represented and manipulated digitally			✓	✓	✓	✓			
• Undertake creative projects that involve selecting, using, and combining multiple applications to achieve challenging goals, including meeting the needs of known users								✓	✓
• Create, re-use, revise and re-purpose digital artefacts for a given audience, with attention to trustworthiness and design									✓

KS3 Design & Technology	Lessons								
	1	2	3	4	5	6	7	8	9
Element of the curriculum									
• Use research and exploration to identify and understand user needs								✓	✓
• Develop specifications to inform the design of innovative, functional, appealing products that respond to needs in a variety of situations								✓	
• Use a variety of approaches to generate creative ideas and avoid stereotypical responses								✓	
• Develop and communicate design ideas using annotated sketches, oral and digital presentations and computer-based tools								✓	✓
• Select from and use a wider range of materials and components according to their functional properties and aesthetic qualities								✓	
• Test, evaluate and refine ideas and products against a specification, taking into account the views of intended users and other interested groups								✓	✓
• Apply computing and use electronics to embed intelligence in products that respond to inputs, and control outputs, using programmable components								✓	✓

Applicable standards

International Society for Technology in Education (ISTE) Standards for Students

Element of the curriculum	Lessons									
	1	2	3	4	5	6	7	8	9	
1c. Students use technology to demonstrate their learning in a variety of ways.								✓	✓	✓
1d. Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3a. Students plan and employ effective research strategies to locate information and other resources for their intellectual or creative pursuits.								✓	✓	✓
3c. Students curate information from digital resources using a variety of tools and methods to create collections of artifacts that demonstrate meaningful connections or conclusions.										✓
3d. Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4a. Students know and use a deliberate design process for generating ideas, testing theories, creating innovative artifacts or solving authentic problems.								✓	✓	✓
4c. Students develop, test and refine prototypes as part of a cyclical design processes.								✓	✓	✓
4d. Students exhibit a tolerance for ambiguity, perseverance and the capacity to work with open-ended problems.								✓	✓	✓
5a. Students formulate problem definitions suited for technology-assisted methods such as abstract models and algorithmic thinking in exploring and finding solutions.								✓	✓	
5c. Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.								✓	✓	
5d. Students understand how automation works and use algorithmic thinking to develop a sequence of steps to create and test automated solutions.		✓	✓	✓	✓	✓	✓	✓	✓	
6a. Students choose the appropriate platforms and tools for meeting the desired objectives of their creation or communication.										✓
6b. Students create original works or responsibly repurpose or remix digital resources into new creations.										✓
6d. Students publish or present content that customizes the message and medium for their intended audiences.										✓
7c. Students contribute constructively to project teams, assuming various roles and responsibilities to work effectively toward a common goal.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7d. Students explore local and global issues and use collaborative technologies to work with others to investigate solutions.	✓							✓	✓	✓

Lesson 1: Robot cars and smart cities

Overview

In the first lesson of this unit of work, we will introduce your class to the concept of robotics and autonomous cars. Your students will discuss what cities in the future will look like, and think about the role that robots and autonomous vehicles will play. Next, your class will work in groups to complete the main challenge of the first lesson: building the mBot, then getting it moving around the classroom with the remote control.

Learning outcomes

- Name a benefit of smart cities
- Describe characteristics of an autonomous vehicle
- Compare characteristics of robots and humans
- Construct a robot
- Name the basic parts of the robot
- Control a robot using a simple remote
- Identify and share problems encountered with solutions

Resources



Slideshow 1:
Robot cars and smart cities



Student Sheet 1a:
What are the differences between robots and humans?



Video:
Building your robot



360 Video:
Look! Driving with no hands!



Image:
Comparing my mBot to a driverless car interactive

Lesson 2: Controlling cars with code

Overview

In the second lesson of this unit of work, we will introduce your class to the concept of using code to control cars. Your students will discuss how programming a car compares to programming a standard computer. Next, your class will work in groups to complete the main challenge of the second lesson: programming the robot to drive in different shapes around the classroom.

Learning outcomes

- Understand the difference between hardware and software and how they relate to each other
- Describe functions carried out by software and hardware
- Get the robot moving with code
- Learn about loops (repetition)
- Debug programs
- Identify and share problems encountered with solutions

Resources



Slideshow 2:
Controlling cars with code



Student Sheet 2a:
Electric motors explainer

Student Sheet 2b:
Controlling cars with code



Answer Sheet 2b:
Controlling cars with code



Video:
What is code?

Video:
Uploading code to your robot

Lesson 3: Where am I?

Overview

In the third lesson of this unit of work, your class will learn about the role of mapping in autonomous vehicles. They will then be challenged to make their robot vehicle follow a path using new hardware and software. Students will learn about how the line follower sensor on the mBot works, then use the sensor data and logic-based code to autonomously control their robots. Finally, they will relate this activity to real world driverless vehicles and discuss the other sensors needed to make driving efficient and safe.

Learning outcomes

- Understand why mapping is important for programming autonomous cars
- Describe functions carried out by software and hardware
- Describe how a line follower functions
- Apply logic to get a robot to follow a defined path
- Debug programs
- Identify and discuss ways of solving the same problem under different conditions

Resources



Slideshow 3:
Where am I?



Student Sheet 3a:
Explaining the line follower

Student Sheet 3b:
Coding the line follower



Answer Sheet 3b:
Coding the line follower



Video:
Where am I?

Lesson 4: What's around me?

Overview

In the fourth lesson of this unit of work, your class will learn about obstacles and sensors. They will discuss the risks that a smart city presents, focusing on the challenges that an autonomous vehicle faces while navigating in the real world. They will then learn about the ultrasonic sensor on-board the mBot and how to use it to avoid obstacles. Finally, they will think about how driving speed can influence a vehicle's ability to react to obstacles.

Learning outcomes

- Describe what smart cities are and give examples of smart city initiatives
- Describe risks for autonomous vehicles in smart cities
- Describe how an ultrasonic sensor functions
- Apply code and sensor output to navigate around an obstacle
- Debug programs
- Identify and share ways of solving problems

Resources



Slideshow 4:
What's around me?



Student Sheet 4a:
Explaining the ultrasonic sensor

Student Sheet 4b:
Coding the ultrasonic sensor



Answer Sheet 4b:
Coding the ultrasonic sensor



Video:
What's around me?

Lesson 5: Safety and signalling

Overview

In the fifth lesson of this unit of work, your class will learn about how technology and people interact. They will learn about signalling movement and giving warnings with light and sound. Students will use code to control their robot car's LEDs and buzzer to produce lights and sounds for a variety of different scenarios. They will then combine an input – the ultrasonic sensor – and an output – the LEDs and buzzer – to create a proximity sensor. Finally, they will discuss the other sensors and signals they think would be useful.

Learning outcomes

- Understand what we mean by signalling and why it is needed to keep people safe
- Discuss the best ways for robots to signal to humans
- Understand, describe and control LEDs and buzzers
- Integrate inputs (ultrasonic sensor) to outputs (LEDs or buzzer)
- Debug programs
- Identify and share problems encountered with solutions

Resources



Slideshow 5:
Safety and signalling



Student Sheet 5a:
Safety and signalling



Answer Sheet 5a:
Safety and signalling



Video:
Safety and autonomous vehicles



Image:
Comparing my mBot to a driverless car interactive

Lesson 6: What do I do next?

Overview

In the sixth lesson of this unit of work, your class will learn about some of the ways in which technology has failed in the past, and how engineers have worked to overcome those problems. They will then look back over the hardware and software they have been using in the past five lessons and combine these skills to complete challenges. This lesson should be used as an opportunity to consolidate learning, revisit any shaky territory and experiment with combinations of inputs, outputs and different ways of coding the mBot.

Learning outcomes

- Understand that failure is an important part of technology development
- Describe how failure and persistence can help them learn
- Link a variety of inputs to a variety of outputs using code
- Debug programs
- Share learning
- Identify and share problems encountered with solutions

Resources



Slideshow 6:
What do I do next?



Student Sheet 6a:
Hardware and function cards

Student Sheet 6b:
Smart city challenges



Answer Sheet 6b:
Smart city challenges



Video:
What do I do next?

Video:
Failing is good

Lesson 7: Designing our smart city pt. 1

Overview

In the last section of this unit of work, your class will take part in a Design Thinking Workshop that can be delivered as three one hour sessions or combined as a half day activity.

In part one of the workshop, your class will use personas to empathise with different types of people. They will then use these insights to brainstorm ways that robots and autonomous vehicles can improve lives or solve problems.

Learning outcomes

- Understand that design is a process
- Name at least one job associated with design
- Describe basic design thinking techniques
- Understand that issues affect people in different ways
- Empathise with different people and describe how they might see the world
- Think creatively to generate solutions to problems
- Share and evaluate their own ideas

Resources



Slideshow 7:
Designing our smart city pt. 1



Student Sheet 7a:
User profiles

Student Sheet 7b:
Empathy map



Video:
Design thinking

Lesson 8: Designing our smart city pt. 2

Overview

Part two of the workshop sees your class use an ideas funnel to select and refine ideas from the brainstorming activity in part one. Each group will then prototype one of the ideas using the hardware and software skills they have learned with the mBot in lessons 1-6.

Learning outcomes

- Understand what prototyping is and why it is used
- Describe a number of prototyping methods
- Evaluate and refine own ideas and the ideas of others
- Combine hardware, software and crafting skills to make a prototype
- Share learning with the class

Resources



Slideshow 8:
Designing our smart city pt. 2



Student Sheet 8a:
Ideas funnel



Video:
Prototyping

Lesson 9: Designing our smart city pt. 3

Overview

In part three of the workshop each group will discuss different ways of sharing ideas then create articles, posters, videos, photo galleries or reports to persuade their audience that their prototypes are worth taking forward. Then each group will present their prototypes and demonstrate their ideas in action using the mBot as part of a working display.

Outcomes

- Understand why sharing ideas is important
- Name at least one job associated with communication
- Identify a variety of different media and describe when each might be used
- Plan and prepare a group presentation
- Select and create an appropriate way of sharing a project
- Present and explain a group project
- Share learning

Resources



Slideshow 9:
Designing our smart city pt. 3



Student Sheet 9a:
Communicating your ideas



Video:
Communications and marketing

Teacher guidance

The Teacher Guidance for each lesson uses a set of icons to provide visual cues to support teachers:

Lesson activities

**Explain**

teacher exposition using slides or script to support

**Demonstration / watch**

students watch a demonstration or video

**Student activity**

activity for students to complete individually such as questions on a Student Sheet

**Group work**

activity for students to complete in pairs or small groups

**Whole class discussion**

teacher conducts a whole class discussion on a topic or as a plenary review

Teacher ideas and guidance

**Assessment and feedback**

guidance to get the most from AfL (Assessment for Learning)

**Guidance**

further information on how to run an activity or learning step

**Idea**

optional idea to extend or differentiate an activity or learning step

**Information**

background or further information to guide an activity or explanation

**Technical**

specific ICT or practical hints and tips

Health and safety

**Health and safety**

health and safety information on a specific activity

Lesson 1: Robot cars and smart cities

In the first lesson of this unit of work, we will introduce your class to the concept of robotics and autonomous cars. Your students will discuss what cities in the future will look like, and think about the role that robots and autonomous vehicles will play. Next, your class will work in groups to complete the main challenge of the first lesson: building the mBot, then getting it moving around the classroom with the remote control.

Resources in this book:

-  Lesson Overview 1
-  Teacher Guidance 1
-  Student Sheet 1a: What are the differences between robots and humans?
-  Subject Update: What is a smart city?
 - Subject Update: How can autonomous vehicles be useful?
 - Subject Update: Troubleshooting guide
 - Subject Update: mBot's default program

Resources available online:

-  Slideshow 1: Robots cars and smart cities
-  Video: Building your robot
-  360 Video: Look! Driving with no hands!
-  Image: Comparing my mBot to a driverless car interactive

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

Robot cars and smart cities



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems

Resources



Slideshow 1:

Robot cars and smart cities



Student Sheet 1a:

What are the differences between robots and humans?



Video:

Building your robot



360 Video:

Look! Driving with no hands!



Image:

Comparing my mBot to a driverless car interactive



Subject Updates:

- What is a smart city?
- How can autonomous vehicles be useful?
- Troubleshooting guide
- mBot's default program

Kit (per group)

- mBot kit
- Note batteries are **not included** in mBot kits:
 - mBot requires either 4 x 1.5V AA batteries or a 3.7V DC lithium battery
 - Makeblock remote control requires a CR2025 coin cell battery

Lesson overview

In the first lesson of this unit of work, we will introduce your class to the concept of robotics and autonomous cars. Your students will discuss what cities in the future will look like, and think about the role that robots and autonomous vehicles will play. Next, your class will work in groups to complete the main challenge of the first lesson: building the mBot, then getting it moving around the classroom with the remote control.

Lesson steps

1. 360 video opener (5 mins)

Introduction to smart cities and autonomous vehicles. Students get 360 insight to a driverless car to provide context for this lesson's challenge: to build a robot vehicle.

2. Classroom discussion (10 mins)

Students will discuss the human responses to the three questions of autonomy. Then they will discuss additional differences between robots and humans.

3. Make (30 mins)

Build the mBot as per the instruction manual and guidance video.

4. Test (10 mins)

Test the mBot works with the remote control.

5. Reflect (5 mins)

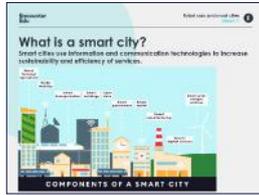
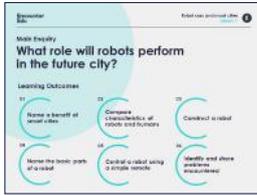
Students will reflect on their learning, including problems they had and how they solved them.

Learning outcomes

- Name a benefit of smart cities
- Describe characteristics of an autonomous vehicle
- Compare characteristics of robots and humans
- Construct a robot
- Name the basic parts of the robot
- Control a robot using a simple remote
- Identify and share problems encountered with solutions

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. More advanced outcomes are formatted in italics and can be deleted. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.



Use **slide 3-4** to introduce students to the concepts of smart cities and autonomous vehicles, or self-driving cars. A smart city uses information and communication technologies to increase sustainability and efficiency of services. An autonomous vehicle is capable of navigating without human input. You may find the **Subject Updates** on smart cities and autonomous vehicles helpful for additional background.



Have your class watch the 360 video **Look! Driving with no hands!**

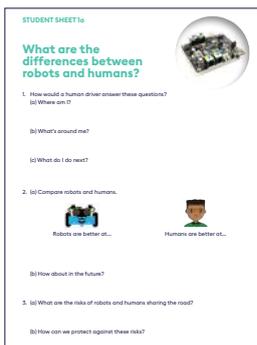


If you need further guidance about using 360VR in the classroom, see the Subject Update in the video description.



Additional interactive multimedia is available online, including the **Comparing my mBot to a driverless car** interactive image, which can be used for differentiation, extension or home learning.

2
10
mins



This section guides students to think about the differences between robots and humans.



Use the questions on **slide 5** and **Student Sheet 1a** to guide a discussion around vehicle autonomy.

First think about how human drivers would answer the three questions:

Where am I?

- They would use their senses.
- They would use their eyes to look around.
- They might use their ears to listen to their surroundings.
- For example, if you accidentally fall asleep on the train and wake up uncertain where you are, you might look at your surroundings. If you don't recognise the surroundings you might listen for an announcement on the train that tells you where you are. Or you might do both.

What's around me?

- They would use their senses.
- They would also look and listen to see what is around them.
- They might also use touch. For example, if they are unsure of what an object is they might feel it.

Step

What do I do next?

- They might follow directions to the destination based on what they know already or based on a map.
- They will also follow the rules of the road. For example, if you're driving then you shouldn't drive through a red light.
- They will also be aware of other road users and predict what they might do based on signalling, so they don't crash.



Continue the discussion of the differences between robots and humans using the questions on **Student Sheet 1a**.

What are robots better at than humans now?

- Anything that involves building products quickly and consistently. For example, making cars. Cars are produced on production lines by robots. As long as these robots are working correctly then every car will be produced quickly and exactly the same.
- Robots can stay on and work for long periods of time. Humans get tired and lose concentration so sometimes products they produce first thing in the morning or at the end of the day might be of a poorer quality.

How about in the future?

- This can only be guessed but as robots advance the number of tasks they will beat humans at will grow.

What are humans better at than robots now?

- Humans are better at socialising.
- Humans are generally better at completing tasks that require you to think. For examples, look at occupations such as teaching or medicine. NB we're thinking only about robots not computers in general as some computers can outthink humans and beat them at games like Chess and Go.

How about in the future?

- This can only be guessed but humans will likely continue to outperform robots at things like socialising and thinking. Robots may become better at some things like moving around quickly.

What are the risks of robots and humans sharing the road?

- The biggest risk is the unpredictability of human drivers. Robots will follow the rules and won't be affected by things like tiredness.

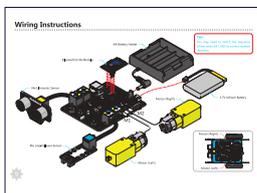
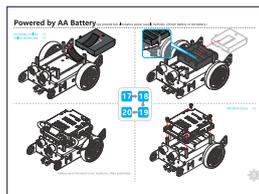
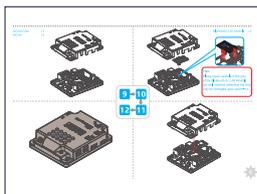
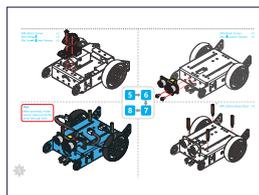
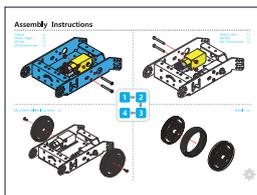
How can we protect against these risks?

- Make the robots aware that not all road users will follow the rules.

Step

3

30
mins



Use **slide 6** to introduce the lesson challenge: to build the mBot and control it with a remote.



Watch the video **Building your robot** once all the way through. Point out the different stages of making the mBot.



Hand out mBot kits, batteries and manual to each group. Ask students to unbox the components.



The first stage of making is to connect the motors and wheels to the chassis. This is supported by **Section 1 of the video Building your robot, slide 7 and page 4 of the mBot instruction manual.**



The second stage of making is to attach the sensors. This is supported by **Section 2 of the video Building your robot, slide 8 and page 5 of the mBot instruction manual.**



The third stage is to prepare the mCore. This is supported by **Section 3 of the video Building your robot, slides 9 and 10 and pages 6 and 8 of the mBot instruction manual.**



The fourth stage of making is to connect all the cables correctly. This is supported by **Section 4 of the video Building your robot, slides 11 and 10 and pages 9 and 8 of the mBot instruction manual.**



If you are using 3.7V DC batteries there are separate instructions on **page 7 of the mBot instruction manual.**



Problems students might encounter:

- Screwdriver works with 2 different types of screws. To change type you can pull the metal part from the handle and insert it back into the handle the opposite way around.
- The motors might be wired the wrong way around. This will be clear when controlling it with the remote control as it will drive the opposite way to the button that is pressed (pressing the forward/up button will make the robot drive backward, etc.)
- Make sure the sensors are connected to the correct ports. This doesn't matter as much at this stage but it will cause problems in later exercises that use the sensors.



Additional troubleshooting tips are available in the subject update **Troubleshooting guide.**

Step

4

10
mins

Encounter Edu Robot cars and smart cities Lesson 1

Prepare a 2-minute demonstration and presentation of the car moving

- What advice would you give someone else building this robot?
- What was the best bit of the robot building?
- Did you face any problems?
- How did you solve your problems?



Students should test that the mBot works with the remote control. The mBot comes loaded with a default program, which allows you to use the arrow buttons on the remote to drive the robot forward and backward as well as turn left and right.



Additional information about the mBot's default program is available in the subject update **mBot's default program**.



Each group should do a 2-minute demonstration and presentation showing the car moving, answering the following questions (**slide 12**):

- What advice would you give someone else building this robot?
- Did you have any problems?
- How did you solve your problems?
- What was the best bit of the robot building?

5

5
mins

Encounter Edu Robot cars and smart cities Lesson 1

Is your robot car an autonomous vehicle?

- Where am I?
- What do I do next?
- What's around me?



Conduct a whole class discussion around whether the mBot that students have just made is truly an autonomous vehicle. Remind students that for their robot car to be autonomous, it would have to be able to answer three questions (**slide 13**):

Where am I?

- Does this robot know where it is? No, not yet.
- What would we have to add to this robot for it to know where it is? Sensors to act as its eyes and ears.

What's around me?

- Does this robot know what's around it? No, not yet.
- What would we have to add to this robot for it to know what's around it? Sensors to act as its eyes and ears and if possible to touch/feel.

What do I do next?

- Does this robot know what to do next? No, not yet.
- What would we have to add to this robot for it to know what to do next? We would have to program it (give it a set of instructions).



In the real world what challenges might this robot have to overcome? Learning the rules of the road, being aware of other road users, learning how to adapt to different road conditions like rain or snow, taking different routes due to closed roads or busy traffic, etc.

What are the differences between robots and humans?



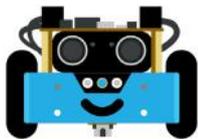
1. How would a human driver answer these questions?

(a) Where am I?

(b) What's around me?

(c) What do I do next?

2. (a) Compare robots and humans.



Robots are better at...



Humans are better at...

(b) How about in the future?

3. (a) What are the risks of robots and humans sharing the road?

(b) How can we protect against these risks?

Lesson 2: Controlling cars with code

In the second lesson of this unit of work, we will introduce your class to the concept of using code to control cars. Your students will discuss how programming a car compares to programming a standard computer. Next, your class will work in groups to complete the main challenge of the second lesson: programming the robot to drive in different shapes around the classroom.

Resources in this book:



Lesson Overview 2



Teacher Guidance 2



Student Sheet 2a: Electric motors explainer

Student Sheet 2b: Controlling cars with code



Answer Sheet 2b: Controlling cars with code



Subject Update: How to set up mBlock

Subject Update: Tips for teaching coding

Subject Update: Troubleshooting guide

Subject Update: mBlock glossary

Resources available online:



Slideshow 2: Controlling cars with code



Video: What is code?

Video: Uploading code to your robot

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

LESSON 2

Controlling cars with code



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems
- Use a programming language to solve a variety of computational problems

Resources



Slideshow 2:

Controlling cars with code



Student Sheet 2a:

Electric motors explainer

Student Sheet 2b:

Controlling cars with code



Answer Sheet 2b:

Controlling cars with code



Video:

What is code?

Video:

Uploading code to your robot



Subject Updates:

- How to set up mBlock
- Tips for teaching coding
- mBlock glossary

Kit (per group)

- mBot created in previous lesson
- Laptop or tablet with mBlock software installed (see How to set up mBlock Subject Update)

Lesson overview

In the second lesson of this unit of work, we will introduce your class to the concept of using code to control cars. Your students will discuss how programming a car compares to programming a standard computer. Next, your class will work in groups to complete the main challenge of the second lesson: programming the robot to drive in different shapes around the classroom.

Lesson steps

1. Video opener (5 mins)

Introduction to code and its relationship to hardware. This video will set this lesson's challenge: to control your robot car with code.

2. Classroom discussion (10 mins)

Students will discuss the video and think about the hardware and software needed to control a real world autonomous vehicle.

3. Make (25 mins)

Introduction to the most common types of motor. Use the mBlock coding environment for the first time. Learn how to control the DC motors to get the robot moving in simple shapes.

4. Test (10 mins)

Test the robot moves as expected.

5. Reflect (10 mins)

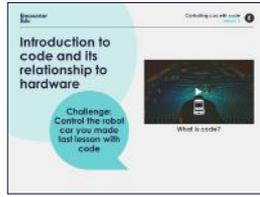
Students will reflect on their learning, including problems they had and how they solved them. The class will discuss how the robot car would behave differently on different surfaces and how they would compensate for this.

Learning outcomes

- Understand the difference between hardware and software and how they relate to each other
- Describe functions carried out by software and hardware
- Get the robot moving with code
- Learn about loops (repetition)
- Debug programs
- Identify and share problems encountered with solutions

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

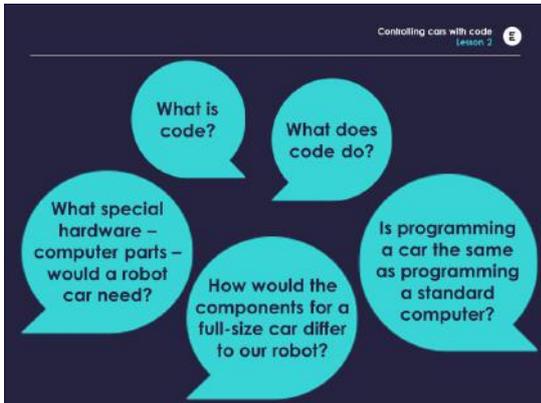


Using **slide 3** introduce students to the lesson where they are going to use code to control the car that they built in the first lesson. They are going to start by watching a video that shows the relationship between software (code) and hardware. Today's challenge is to control the robot car your pupils made last lesson with code.



Show your class the video **What is code?**

2
10
mins



This section of the lesson supports learning about the difference between software and hardware, the functions they perform and how they relate to each other.



Use the questions on **slide 4** to conduct a whole class discussion:

1. What is code?
2. What does code do?
3. Is programming a car the same as programming a standard computer?
 - The students might think it would be completely different, but we can actually use the same or similar programming languages, using similar logic.
4. What special hardware – computer parts – would a robot car need?
 - Encourage pupils to think back to the last lesson. What does a robot car need to know and do? Are there any special parts that would help it do a good job? Pupils might identify sensors that already exist in cars like parking sensors which notify drivers when they are close to other cars when parking. See if they can identify anything else that might be useful. Students do not have to describe the sensor types themselves, but could describe their functions, e.g. detecting the lines on the road or reacting to traffic lights.
5. How would the components for a full-size car differ to our robot?
 - They would have similar functions but be more advanced as safety is a lot more important when transporting people compared to a toy robot.

Step

3
25
mins



Students should have access to their connected device at this stage. When you show them **slide 7**, this is the screen that they should see on their devices.



Using **slides 5-8**, introduce the lesson challenges and explain the mBlock programming language to students. mBlock is based on Scratch - a language your class may already be familiar with. The blocks are split into different types by colour (slide 6). For our robot the types we will be using are control, operators and robots. For more advanced programs we might use data & blocks. The other types of blocks won't have an effect on the robot.



So students don't accidentally use the wrong blocks, click on the 'Edit' menu and select 'Arduino mode'.



Students should see the screen in **slide 7**. The block types students can use with the robot are highlighted. The middle rectangle is where we place our program and the right-hand rectangle is the same code in Arduino C – the language the robot understands. This is the code that is loaded onto the robot. Show your students the video **Uploading code to your robot**.



Using **slide 8**, explain motors. To get the robot moving we need to program the motors. The three main types of motors are DC motors, servo motors and stepper motors.

- **DC motors** are good for wheels that need to spin fast so are perfect for powering wheels to get a robot moving.
- **Servo motors** are fast, powerful and good for accurate rotation but can only move through 180 degrees. Good for robotic arms.
- **Stepper motors** are good at slow, accurate rotation with better positional control than servos and can move through 360 degrees. Good for systems where position is very important, such as 3D printers.

All these motors make use of something called an H bridge to be able to drive the motor backwards and forwards. The servo is the only motor to have it built in. The mBot has a DC motor and the board we will program to control the motors has an H bridge.



This may seem like a lengthy explanation and potentially over-complicated, but it is important that students understand that they are going to be coding the motors today and that without hardware, their code would have no physical 'impact'. **Student Sheet 2a** is an optional reference document that you may provide to your students.



Check that students understand the mBlock interface and that they are using code to control the motors.

Step



Hand out a copy of **Student Sheet 2b** to each group. If needed, read through the sheet as a class. Students should work in their groups to complete the activities. Challenge activities have been added to stretch students. Students will be prompted to drive the robot in a square using two different methods: 1) adjusting the motion block to 'turn' and 2) setting the robot's two motors to different speeds. This will lead to additional challenges to drive in different shapes.

4

10
mins

Encounter Edu Controlling cars with code Lesson 2

Did the robot do what you thought it would?

- Check code
- Check connection
- Check settings
- Check power



Showing **slide 9**, ask the class to raise their hands if their robot drove correctly for the given shape. For all the other groups do they know why it didn't? If they don't know can any other group tell them what might be going wrong?



One of the most common problems is turning too little or too much, students will need to adjust the time to fix this. Another thing to look for is testing the code on different surfaces.



Extend the discussion into asking students how the mBot may have performed differently on different surfaces.

- Does the program work the same on different surfaces?
- Why is this?

The reason is that on different surfaces there are different levels of friction. To handle this, we have to adjust the amount of time the robot turns for.

5

10
mins

Encounter Edu Controlling cars with code Lesson 2

Would the program you made work in all weathers?

The program I developed today would / would not work in all weathers because ...



Would the program the students designed work in all weather conditions? Using **slide 10**, ask students to write one or two sentences explaining why it would or would not.

You can support students' thinking by asking questions:

- Have the cars today behaved the same way on all surfaces?
- How might the weather affect a robot car on a real road?
- How else might you want to change the program according to the weather?

Step



Review students' ideas. If students need further support, prompt towards the following answers:

- What happened when we put the robot on different surfaces? It behaves differently. Are there any weather conditions that might have a similar effect? Snow or rain would mean there would be less friction, so we would have to act differently.



Students should understand that autonomous cars don't just have to get from a to b – they have to react to conditions on the road in real time.

- How would the vehicle detect these changes? Recall back to the start of the lesson when you discussed special hardware such as sensors.
- How should we drive in rain or snow? Slower and more carefully.

Electric motors explainer



To drive your robot car you will be using code to control the motors. The type of motor you will be controlling is a DC motor, but it's good to know there are other types of motors that you may encounter.



DC motors are good for wheels that need to spin fast so are perfect for powering wheels to get a robot moving.



Servo motors are fast, powerful and good for accurate rotation but can only move through 180 degrees. They are good for robotic arms.



Stepper motors are good at slow, accurate rotation with better positional control than servo motors and can move through 360 degrees. They are good for systems where position is very important, such as 3D printers.

All these motors make use of something called an H bridge to be able to drive the motor backwards and forwards. The servo is the only motor to have it built in. The mBot has a DC motor and the board we will program to control the motors has an H bridge.

Controlling cars with code



Today we are going to learn how to code a car to perform basic movements. **Let's get moving!**

1. In the robots blocks under mBot drag out the block and put it at the top of your program. All our mBot programs must start with this block.



If you don't see this block click on the Boards menu and make sure mBot (mCore) is selected.

2. To make the mBot move forward, connect the movement block and change the speed to 100.



Use the dropdown menu to change the speed.

3. You might think our program is now finished but if we load that on to the robot it will never turn the engine off and will keep running forward until we turn it off or the batteries run out. We also need to add in the same block with speed set to 0.

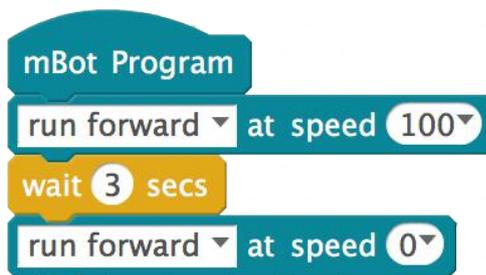
Do you think this will work as it is? This will cause the robot to turn the motor on then off again straight away, so the robot won't move.

4. To make the robot wait before starting the next command, use a wait block. Change the time to 3 secs.



This block is gold so is in the control type blocks.

5. Your final program should look like this...



STUDENT SHEET 2b

Load the code!

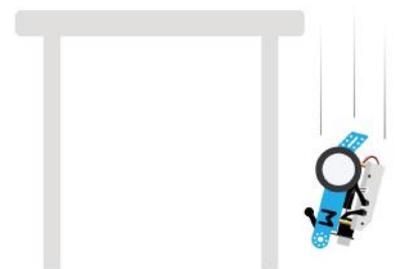
- To load the program onto the robot connect it to the computer using the USB cable – flat end into the computer and square end into the port marked USB on the robot. Turn on the robot and then put it upside down.
- We next need to tell the computer what port the robot is connected to. Click the Connect menu -> Serial Port and select the last port.
- Next click on “Upload to Arduino”.
- Once the program is loaded onto the robot it will run straight away. Disconnect the robot and place it on the floor.
- To restart the program either press the reset button on the mBot or turn the power switch off and on.



 If you have more than one option it will always be the bottom one.

 Make sure your power switch on the mBot is set to on and the power supply is connected.

 Be careful it doesn't run off the desk or table.



More challenge

- Can you edit the last block and add 2 extra blocks to make the robot drive forward then reverse back to where it started?
- Can you make the robot drive in a square? For this you'll need a new block. This is a repeat **loop** which will repeat the code inside it a set number of times.



 See what other options you have under the dropdown menu.



 To drive in a square we need to repeat the same blocks a set number of times. What other blocks will you need? Is your robot turning enough to make a square?

STUDENT SHEET 2b

More challenge (continued)

13. Try using this block to control each motor individually. You will need two of these blocks. M1 is the left engine (when looking from the back) and M2 is the right engine. Can you make the robot drive in a square by controlling each motor individually?



 How did the wheels move when driving in a square before? Did only one wheel turn or both?

14. Can you make the robot drive in a circle?

 Driving in a circle is kind of like driving a wide turn.

15. Can you make the robot drive in a hexagon?

 Go back to how you drove the car in a square using a loop. Can you adjust the code? How many times do you need to repeat the loop? What about the angle of the turn?

16. Can you make the robot drive in a pentagon?

Controlling cars with code

Forwards and backwards

Solution to drive the motor forward and then reverse back to same position.

The code consists of six numbered lines:

- mBot Program**
- run forward** at speed **100**
- wait 3 secs**
- run backward** at speed **100**
- wait 3 secs**
- run forward** at speed **0**

Explanations for each line:

- Line 1: This block is needed at the start of all mBot programs.
- Line 2: Move forward at speed 100.
- Line 3: Wait for 3 seconds before moving onto the next command. This means the robot will drive forward for 3 seconds.
- Line 4: Drive backward at speed 100.
- Line 5: Wait for 3 seconds to complete driving backward.
- Line 6: Stop by setting the motor speed to 0.

Drive in a square (method 1)

Solution to make the robot drive in a square.

The code consists of seven numbered lines:

- mBot Program**
- repeat 4**
- run forward** at speed **100**
- wait 3 secs**
- turn right** at speed **100**
- wait 1.75 secs**
- run forward** at speed **0**

Explanations for each line:

- Line 2: Repeat the set of blocks inside this repeat block, in order, 4 times.
- Line 3-7: Draws 1 line of the square and turns to be at the beginning of the next line.
- Line 5-6: The amount of time to turn right for will vary depending on the surface the robot is being tested on.

Note that the angle of the turn is dependent on the speed of the engine and the time it is on for. Also note that the same program might not work the same on all surfaces due to friction.

Drive in a square (method 2)

Drive in a square by controlling each motor individually. This is essentially the same as the last program but we need a block to control each motor. M1 is the left engine (when looking from the back) and M2 the right.

When turning, one wheel moves forward and the other wheel moves backward at the same speed to turn on the spot. For example, to turn right the left wheel moves forward and the right wheel moves backward. It is possible to turn right by just moving the left wheel but this would make it a much wider turn and impossible to drive in a square.

1 mBot Program

2 repeat 4

3 set motor M1 speed 100

4 set motor M2 speed 100

5 wait 3 secs

6 set motor M1 speed 100

7 set motor M2 speed -100

8 wait 1.75 secs

9 set motor M1 speed 0

10 set motor M2 speed 0

Line 3-4: set both motors to speed 100.

Line 6-7: set motor 1 to go forward at speed 100 and motor 2 to drive in reverse at speed 100. This will cause the robot to turn right on the spot.

Drive in a circle

To drive the robot in a circle we just need to set the motor speeds to be different. The difference between each of them will affect the size of the circle. To drive clockwise we make the left wheel drive faster than the right.

1 mBot Program

2 set motor M1 speed 150

3 set motor M2 speed 70

Line 2-3: set the motors to different speeds.

Drive in a hexagon

Driving in a hexagon is similar to driving in a square, but we have to draw 6 sides and adjust the angle.

1 mBot Program

2 repeat 6

3 run forward at speed 100

4 wait 3 secs

5 turn right at speed 100

6 wait 1 secs

7 run forward at speed 0

Line 2: Repeat the set of blocks inside this repeat block, in order, 6 times.

Line 3-7: Draws 1 line of the hexagon and turns to be at the beginning of the next line.

Line 5-6: The amount of time to turn right for will vary depending on the surface the robot is being tested on. It should be less time than for driving in a square.

Drive in a pentagon

Driving in a pentagon is similar to driving in a square, but we have to draw 5 sides and adjust the angle.

1 mBot Program

2 repeat 5

3 run forward at speed 100

4 wait 3 secs

5 turn right at speed 100

6 wait 1.25 secs

7 run forward at speed 0

Line 2: Repeat the set of blocks inside this repeat block, in order, 5 times.

Line 3-7: Draws 1 line of the pentagon and turns to be at the beginning of the next line.

Line 5-6: The amount of time to turn right for will vary depending on the surface the robot is being tested on. It should be less time than driving in a square but more than in a hexagon.

Lesson 3: Where am I?

In the third lesson of this unit of work, your class will learn about the role of mapping in autonomous vehicles. They will then be challenged to make their robot vehicle follow a path using new hardware and software. Students will learn about how the line follower sensor on the mBot works, then use the sensor data and logic based code to autonomously control their robots. Finally, they will relate this activity to real world driverless vehicles and discuss the other sensors needed to make driving efficient and safe.

Resources in this book:



Lesson Overview 3



Teacher Guidance 3



Student Sheet 3a: Explaining the line follower

Student Sheet 3b: Coding the line follower



Answer Sheet 3b: Coding the line follower



Subject Update: When can a car be considered autonomous?

Subject Update: Tips for teaching coding

Subject Update: Troubleshooting guide

Subject Update: About Oxbotica

Resources available online:



Slideshow 3: Where am I?



Video: Where am I?

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

LESSON 3

Where am I?



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems
- Understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming
- Use a programming language to solve a variety of computational problems

Resources



Slideshow 3:
Where am I?



Student Sheet 3a:
Explaining the line follower

Student Sheet 3b:
Coding the line follower



Answer Sheet 3b:
Coding the line follower



Video:
Where am I?



Subject Updates:

- When can a car be considered autonomous?
- Tips for teaching coding

Kit (per group)

- mBot
- Laptop or tablet with mBlock
- Black tape e.g. insulating tape
- Access to a light / white surface for practice

Lesson overview

In the third lesson of this unit of work, your class will learn about the role of mapping in autonomous vehicles. They will then be challenged to make their robot vehicle follow a path using new hardware and software. Students will learn about how the line follower sensor on the mBot works, then use the sensor data and logic-based code to autonomously control their robots. Finally, they will relate this activity to real world driverless vehicles and discuss the other sensors needed to make driving efficient and safe.

Lesson steps

1. Video opener (5 mins)

An introduction to mapping, the video will set this lesson's challenge: to make the robot follow a path.

2. Classroom discussion (10 mins)

Students will discuss the video. How do humans know which way to go? How do we know where we are? How do we teach robots to know which way to go?

3. Make (25 mins)

Introduction to the line follower hardware and how to control it with code. Define and follow 'rules of the road' using table-based code.

4. Test (10 mins)

Test the robot moves as expected.

5. Reflect (10 mins)

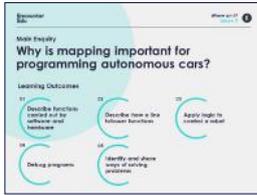
Students will reflect on their learning, including problems they had and how they solved them. They will then discuss how different road conditions might affect the robot and how this might affect the code.

Learning outcomes

- Understand why mapping is important for programming autonomous cars
- Describe functions carried out by software and hardware
- Describe how a line follower functions
- Apply logic to get a robot to follow a defined path
- Debug programs
- Identify and discuss ways of solving the same problem under different conditions

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

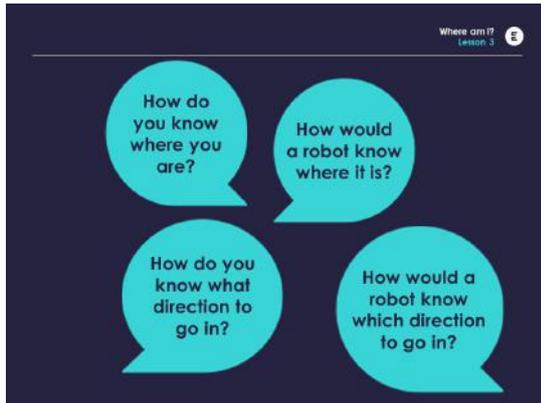


Using **slide 3** introduce students to the lesson where they are going to start to code their robots to behave autonomously. They are going to start by watching a video that describes the importance of mapping for programming autonomous cars. Today's challenge is to code the robot to follow a path.



Show your class the video **Where am I?**

2
10
mins



Manage a whole class discussion using the questions on **slide 4**. To add a little more structure, you can use a think-pair-share format as described below.



Give students two minutes to write down all the different ways that they can use to know where they are.



Students then share their lists in pairs for one minute comparing the different answers they have written down.



Ask student pairs to share some of their answers. Students should include:

- You look around.
- You read signs.
- You listen for different sounds.
- You feel the surface you are on.

Extend the discussion by asking students how robot would know where it is? Robots have to use sensors to simulate these senses: camera for eyes, microphone for sounds, it could tell the surface it is on for how easy it is to move (for example, it might be harder to move in mucky ground in a park compared to a smooth road).



Give students another two minutes to list all the different ways that they can use to know which direction to go in on a journey.



Give students an additional minute to discuss this in pairs. Students may find this more of a challenge than the previous questions.

Step



Select students to share their thoughts. Students may come up with:

- Using a map.
- Asking for directions.
- Knowing the way already.

How would a robot know what direction to go in? In a similar way to you, it might have access to a map or the directions might already have been given to it (just like the directions already being in your head).

3
25
mins

Your challenge is to program the robot to follow a black line.

- Define the "rules of the road"
- Decide how you want your robot to react?
- Program the robot using a simple logic
- Program the robot using a complex logic

The line follower is an input module on the robot.

How the line follower works

Coding the line follower

line_follower (Port2)

Sensor 1 (left)	Sensor 2 (right)	Value
Black	Black	0
Black	White	1
White	Black	2
White	White	3

STUDENT SHEET 3a

Explaining the line follower

How the line follower works

The line follower is on the bottom of the robot at the front. The line follower has 2 sensors which can detect whether a surface is black or white within a range of 1-2cm. If you look at each sensor you can see the LED (dark bulb) on the back and the photodiode (clear bulb) on the front.

The LED light emitting diode shows IR (infrared) light. This light beam is reflected off a surface is received by the photodiode. A white surface shows here, reflects a lot of light.

A dark surface only reflects a little light. Also, if the LED is far from a surface, only a little light will be reflected.

STUDENT SHEET 3b

Coding the line follower

Today you are going to learn how to code a car to follow a black line. **Let's get moving!**

- In the robot blocks under mBot drag out the block and put it in the top of your program. All our mBot programs must start with this block.
- To make the mBot follow the line continuously, we are going to use the forward loop block.
- We are now going to add the first of our logic blocks to use the output from the line follower sensors to control the car.
- In the hexagonal block, we need to add an operator block. We want the block that tells if the line follower output equals something. Remember our line follower report!



During this section, students will learn how to use the line follower to code their mBots to follow a line. There are two Student Sheets to support students' learning: **Student Sheet 3a** provides a background to the line follower, how it works and the principles behind coding it; **Student Sheet 3b** provides step-by-step guidance on writing the code.



Students will need somewhere to test their mBots and it is best to set this up before they start to code. Either create a circuit yourself using black tape on a white or light surface, or allow students to create their own. It would be best if this surface were on the ground and smooth.



Hand out **Student Sheet 3a** if using, to act as a guide for the information you are about to present.



Introduce the challenge using the information on **slide 5** and then explain how the line follower functions using the information on **slides 6 and 7**. The line follower is on the bottom of the mBot at the front. The line follower has two sensors which can detect a white surface (within the range of 1-2cm). It works by emitting IR (infrared) light and recording how much is reflected back. If a lot is reflected back, it is close to a white surface. If a little is reflected back, the surface is black, or the sensor is not near a surface. If you look at each sensor you can see the LED (dark bulb) at the back and the photodiode (clear bulb) at the front.



Then explain how to code the line follower. The line follower block shown on **slide 8** is used to read the sensor. It shows the port that the sensor is connected to.



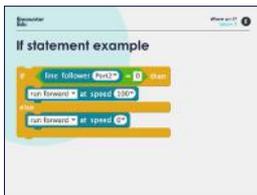
Ask students to check that the line follower sensor is connected to port 2. Looking at the top of the mBot, the ports are numbered 1-4 and are labeled with colours. If you look at the sensors they also have coloured labels. A sensor can technically be connected to any port with the matching colour. However, for this task the line follower should be connected to port 2.

Step

Coding the line follower

Line follower Part 2

Sensor 1 (left)	Sensor 2 (right)	Value
Black	Black	0
Black	White	1
White	Black	2
White	White	3



Use the table on **slide 8** to explain how the line follower returns a number depending on what it detects. If both sensors detect black, then 0 is returned. All the cases can be seen in the table.



Students are going to use some new code blocks (**slide 9**). The forever loop will just run the code inside it as long as there is power going to the robot. An if-else block, which is known as a conditional statement, runs code in the first part, if the if condition is true, or in the second part, if the condition is false. The example on **slide 10** tells the robot to move forward if both sensors detect black (remember this would be 0 from the table above), otherwise stop.



Hand out **Student Sheet 3b**, one per group. Student groups should work through the challenge, and then test their mBots.

4

10 mins

Does the robot manage to follow the line?



Using **slide 11**, ask the class to raise their hands if their robot managed to follow the line. For all the other groups, why did it go wrong? If they managed to fix the problems, how did they do it? If they didn't, can any other group tell them what might be going wrong?



The most common problem is driving too fast, causing the car to lose the line completely.



If there is spare time, student groups should try again to see if they can get their mBot to follow a line.



How would you change your code if it was on a slippery surface like ice? Drive slower.

5

10 mins

How could we use a line follower in driverless cars?

We could use a line follower in driverless cars to...
However, we might have problems with...



This final section can be conducted as a discussion or short written answer exercise.



Optionally, ask students to write an answer to the question on **slide 12**.

Step



Discuss responses as a class.

How could we use a line follower in driverless cars?

- We could use it to obey the current rules of the roads based on lines, e.g. lines we shouldn't cross or stop junctions.

Is this sensor enough?

- No.

Can you think of any problems you might come across?

- We wouldn't be able to just use it to follow a line to our destination, unless we completely changed existing roads. Additionally, there are so many other things we have to pay attention to on the road like traffic lights, other cars, and pedestrians.

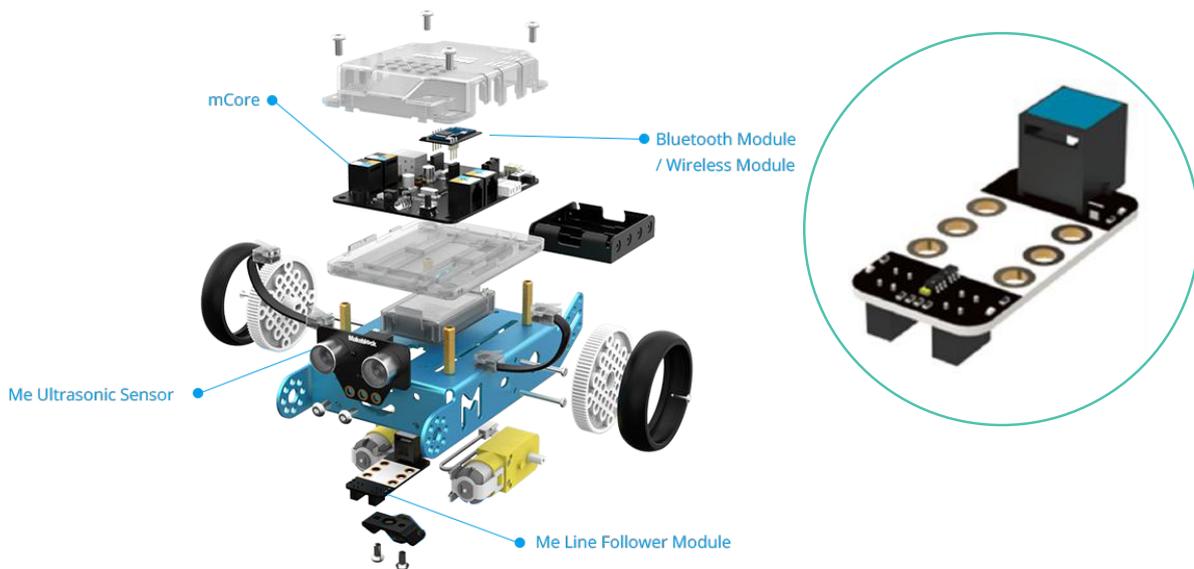
What other things would we have to sense?

- We would have to sense other cars, people and obstacles. We should also listen out for sirens to know when to give way to the emergency services. Feeling different road surfaces, would allow us to adjust our speed to drive safely.

Explaining the line follower



How the line follower works



The line follower is on the bottom of the mBot at the front. The line follower has 2 sensors which can detect whether a surface is black or white within a range of 1-2cm. If you look at each sensor you can see the LED (dark bulb) at the back and the photodiode (clear bulb) at the front.



The LED (light emitting diode) shines IR (infrared) light. Any light that is reflected off a surface is received by the photodiode. A white, surface shown here, reflects a lot of light.

A dark surface only reflects a little light. Also, if the LED is far from a surface, only a little light will be reflected.

STUDENT SHEET 3a

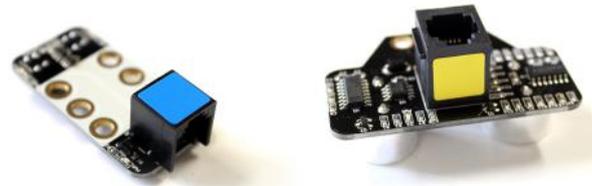
Reading the line follower

So that we can use the line follower to help control the robot, we need to know what information is coming from the sensor.

First, we should check our sensors. If you look at the top of the mBot you can see there are 4 ports, numbered 1 - 4.



Each port is labelled with multiple colours. If you look at the sensors they also have a coloured label. A sensor can be connected to any port labelled with its colour.



The line follower should be connected to **port 2**. If you look at the line follower you can see it's labelled blue so could be connected to any of the ports but for this task leave it connected to port 2.



We use the line follower block to read the sensor. The block returns a number depending on what the sensors detect. All the cases can be seen in the table below.



Sensor 1 (left)	Sensor 2 (right)	Line Follower Output
Black	Black	0
Black	White	1
White	Black	2
White	White	3

STUDENT SHEET 3a

Introducing the coding blocks

Now let's turn what we know into code to make the robot follow a black line. We need some new blocks for this. We need a **forever loop** which will just run the code inside it as long as there is power going to the robot.



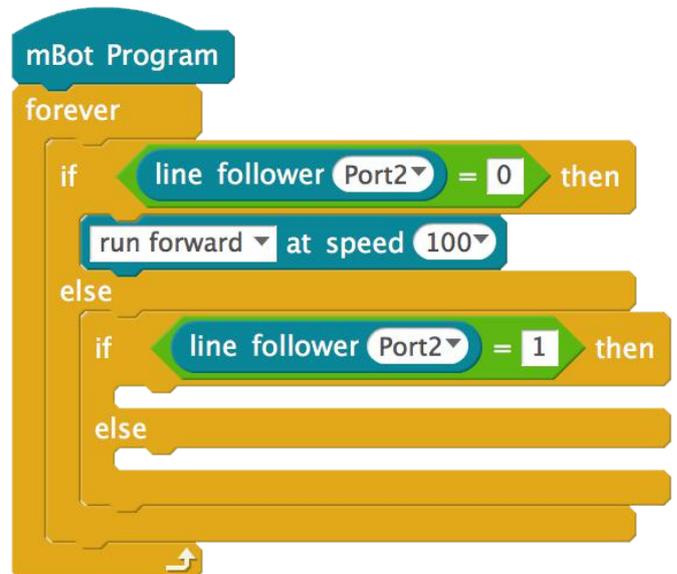
We also need an **if-else** block which is known as a **conditional statement**. It runs code in the first part if the if condition is true, or in the second part if the condition is false.



This code would tell the robot to move forward if both sensors detected black (remember this would be 0 from the table above), or otherwise stop.



Based on this, the start of our program for following a line would look something like this. Use the instructions on Student Sheet 3b to build this code and then complete the program.



Coding the line follower



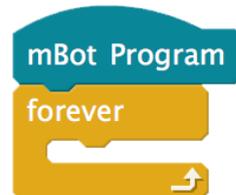
Today we are going to learn how to code a car to follow a black line.
Let's get moving!

1. In the robots blocks under mBot drag out the block and put it at the top of your program. All our mBot programs must start with this block.



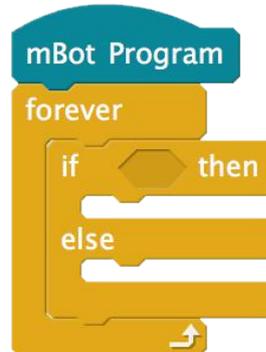
If you don't see this block click on the Boards menu and make sure mBot (mCore) is selected.

2. To make the mBot follow the line continuously, we are going to use the forever loop block.



This block is gold so it's in the control type blocks.

3. We are now going to add the first of our if-else blocks to use the output from the line follower sensor to control the car.



4. In the hexagonal space, we need to add an operator block. We want the block that tells if the line follower output equals something. Remember our line follower table!

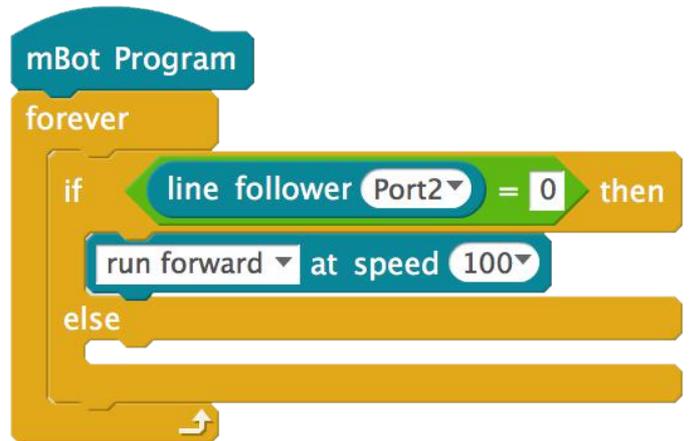


The equals block is green so it's in the operator type blocks.

To change the number in the equals block, just click your cursor in this space and type.

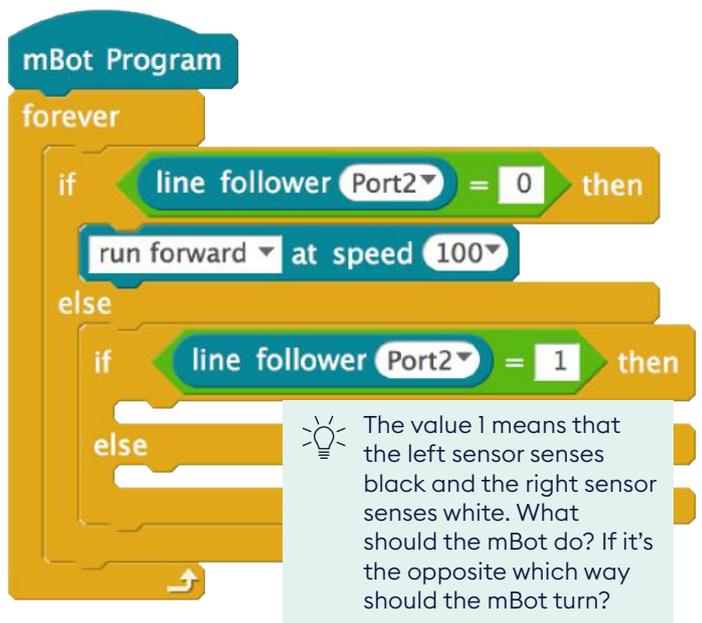
STUDENT SHEET 3b

5. Adding the line follower block and the movement block, means that the code now reads **if the line follower output = 0** [i.e. both sensors sense black] **then run forward at speed 100.**



```
mBot Program
forever
  if line follower Port2 = 0 then
    run forward at speed 100
  else
```

6. This will mean that the mBot will move forward if it senses the black line, but what happens if the mBot drives away from the line?



```
mBot Program
forever
  if line follower Port2 = 0 then
    run forward at speed 100
  else
    if line follower Port2 = 1 then
    else
```

 The value 1 means that the left sensor senses black and the right sensor senses white. What should the mBot do? If it's the opposite which way should the mBot turn?

7. Complete your code using the information in the table to help you.

Load the code!

8. To load the program onto the robot connect it to the computer using the USB cable – flat end into the computer and square end into the port marked USB on the robot.



9. We next need to tell the computer what port the robot is connected to. Click the Connect menu -> Serial Port and select the last port.

 If you have more than one option it will always be the bottom one.

10. Next click on “Upload to Arduino”.

 Make sure your power switch on the mBot is set to on and the power supply is connected.

STUDENT SHEET 3b

- Once the program is loaded onto the robot it will run straight away so place it on the floor.
- To restart the program either press the reset button on the mBot or turn the power switch off and on.



Be careful it doesn't run off the desk or table.



Let's use the logic!

- There are multiple ways to check the sensors. Let's try using Boolean logic. You will need this **and** block. This will return true only if both parts are true.



Remove both of the equals operator blocks. You'll only need one **and** block to check both sensors.

- Use this line follower block instead. It checks whether the sensor senses black. Remember you'll need to check two sensors.



You will need a block like this for the left side and one for the right side.

- Complete your code by thinking back to what you had mBot do if only one side sensed black.

Coding the line follower

Solution to make the robot follow a line

1 mBot Program

2 forever

3 if line follower Port2 = 0 then

4 run forward at speed 100

5 else

6 if line follower Port2 = 1 then

7 turn left at speed 100

8 else

9 turn right at speed 100

Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.

Line 3-4: If-else block. If both sensors can see the black line then drive forward.

Line 5-9: Else another if-else block which adjusts the robot's direction so both sensors are on the line again. If only sensor 1 can see the black line then we have to turn slightly to the left till both sensors can see the black line again. Otherwise, we are too far to the left and have to turn right to fix the robot's direction.

Solution to make the robot follow a line using Boolean logic

This is more or less the same as the last program but uses a different method of checking the sensors.

1 mBot Program

2 forever

3 if line follower Port2 leftSide is black and line follower Port2 rightSide is black then

4 run forward at speed 100

5 else

6 if line follower Port2 leftSide is black then

7 turn left at speed 100

8 else

9 turn right at speed 100

Line 3: This is true if both conditions are true, i.e, if condition 1 AND condition 2 are true. So this is only true when both sensors see black. This is same as saying line follower (port2) = 1.

Line 6: If only left sensor see black.

Line 8: Otherwise only right sensor must see black.

Lesson 4: What's around me?

In the fourth lesson of this unit of work, your class will learn about obstacles and sensors. They will discuss the risks that a smart city presents, focusing on the challenges that an autonomous vehicle faces while navigating in the real world. They will then learn about the ultrasonic sensor onboard the mBot and how to use it to avoid obstacles. Finally, they will think about how driving speed can influence a vehicle's ability to react to obstacles.

Resources in this book:



Lesson Overview 4



Teacher Guidance 4



Student Sheet 4a: Explaining the ultrasonic sensor

Student Sheet 4b: Coding the ultrasonic sensor



Answer Sheet 4b: Coding the ultrasonic sensor



Subject Update: When can a car be considered autonomous?

Subject Update: Tips for teaching coding

Subject Update: Troubleshooting guide

Subject Update: About Oxbotica

Resources available online:



Slideshow 4: What's around me?



Video: What's around me?

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

What's around me?



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems
- Understand simple Boolean logic and some of its uses in circuits and programming
- Use a programming language to solve a variety of computational problems

Resources



Slideshow 4:
What's around me?



Student Sheet 4a:
Explaining the ultrasonic sensor

Student Sheet 4b:
Coding the ultrasonic sensor



Answer Sheet 4b:
Coding the ultrasonic sensor



Video:
What's around me?



Subject Updates:

- Tips for teaching coding
- About Oxbotica

Kit (per group)

- mBot
- Laptop or tablet with mBlock
- Boxes or other materials to use as obstacles, e.g. buildings, fallen trees or gates

Lesson overview

In the fourth lesson of this unit of work, your class will learn about obstacles and sensors. They will discuss the risks that a smart city presents, focusing on the challenges that an autonomous vehicle faces while navigating in the real world. They will then learn about the ultrasonic sensor on board the mBot and how to use it to avoid obstacles. Finally, they will think about how driving speed can influence a vehicle's ability to react to obstacles.

Lesson steps

1. Video opener (5 mins)

An introduction to smart cities and risk. The video will set this lesson's challenge: to get the robot to notice and react to obstacles.

2. Classroom discussion (10 mins)

Students will discuss the video. What risks can they think of that their autonomous car will need to notice and avoid?

3. Make (25 mins)

Introduction to the ultrasonic sensor hardware and how to control it with code. Students should code the robot to stop when it detects an obstacle, then react and manoeuvre around obstacles of known and unknown sizes.

4. Test (10 mins)

Test the robot moves as expected. Does it react in time or crash into the obstacle?

5. Reflect (10 mins)

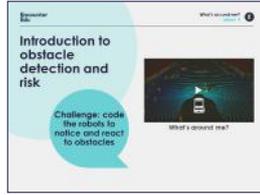
Students will reflect on their learning, including problems they had and how they solved them. Does how quickly your robot drive have an effect on how good it is at this challenge?

Learning outcomes

- Describe what smart cities are and give examples of smart city initiatives
- Describe risks for autonomous vehicles in smart cities
- Describe how an ultrasonic sensor functions
- Apply code and sensor output to navigate around an obstacle
- Debug programs
- Identify and share ways of solving problems

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.



Using **slide 3** introduce students to the lesson where they are going to start to code their robots to react to their environment. They are going to start by watching a video that introduces the concept of the smart city and how autonomous cars might work within them. Today's challenge is to code the robot to avoid an obstacle.



Show your class the video **What's around me?**

2
10
mins



Ask students if they can think of any risks in a smart city. They may be able to recall some of these from the video. How can we handle these risks? How can we avoid them?



Some of the risks might include:

- Security: What happens if the smart systems are hacked? Security has to be a top priority.
- Random behaviour: Smart systems don't behave randomly but people do. So, in situations when there's a mix of smart systems and people-controlled systems we need to be extra careful, e.g. a road with autonomous cars and cars still driven by people.
- Psychology: Some people will struggle to accept things like driverless cars. We can handle these risks by educating them and showing how safe they are. Driverless cars are a lot safer as they just complete the task at hand, whereas, people can get distracted by things like sales in shop windows or seeing someone they know.
- Law: Who is to blame if a car crashes? Is it the driver or is it the people who programmed it or the people who built the car?



This is an opportunity to discuss difficult ethics decisions.

- What do we do if a crash can't be avoided? The cars should look to cause minimal damage, but what happens if that decision is the choice between killing several pedestrians or the driver?

Step

3
25
mins

Your challenge is to program the robot to detect and avoid obstacles.

- 1. Code the robot to stop when it detects an obstacle.
- 2. Make the robot avoid an obstacle when it detects one.

Use one obstacle, Events, and Turn.

The ultrasonic sensor is our robot's eyes.

How the ultrasonic sensor works

STUDENT SHEET 4a

Explaining the ultrasonic sensor

How the ultrasonic sensor works

The ultrasonic sensor measures distance. It is on the front of the robot and looks like a set of eyes. We can use it to detect obstacles.

One of the 'eyes' transmits a sound, and the other waits for the echo of the sound to return. From the time this takes, the distance of the object from the sensor can be calculated. The ultrasonic sensor has a range of 3-400cm.

STUDENT SHEET 4b

Coding the ultrasonic sensor

Today we are going to learn how to code a car to avoid obstacles. Let's get started!

- In the robot blocks under mBot drop out the block and put it at the top of your program. All our mBot programs must start with this block.
- To ensure that the robot will always avoid obstacles, we are going to use the reverse block.
- We are now going to add the first of our if-else blocks to use the output from the ultrasonic sensor to control the car.
- In the hexagonal space, we need to add an operator block. We want the less than block, so we can set the distance of an obstacle that is too close. In the example on the right, we have typed 20 for 20 cm, but you can set your own distance.

Coding the ultrasonic sensor

ultrasonic sensor > distance

Loop and if statement

If statement example

Random number generator

During this section, students will learn how to use the ultrasonic sensor to code their mBots to detect obstacles. There are two **Student Sheets** to support students' learning: **Student Sheet 4a** provides a background to the ultrasonic sensor, how it works and the principles behind coding it; **Student Sheet 4b** provides step-by-step guidance on writing the code.

Students will need somewhere to test their mBots and it is best to set this up before they start to code. Decide what kind of obstacle you are going to use. Students may enjoy creating one or bringing a toy in from home to act as an obstacle. This could be anything from a 'fallen tree' to a building or another vehicle.

Hand out **Student Sheet 4a** if using, to act as a guide for the information you are about to present.

Introduce the challenge using the information on **slide 5** and then explain how the ultrasonic sensor functions using the information on **slides 6 and 7**. The ultrasonic sensor measures distance. It is on the front of the mBot and looks like a set of eyes. We can use it to detect obstacles. One of the "eyes" transmits a sound, and the other waits for the echo of the sound to return. From the time this process takes, the distance of the object from the sensor can be calculated. The ultrasonic sensor has a range of 3-400cm. If an object is outside this range, the sensor will return a value of 400. The line follower block shown on **slide 7** is used to read the sensor. It shows the port that the sensor is connected to.

Ask students to check that the ultrasonic sensor is connected properly. Also, take this opportunity to explain how the sensor is connected to the processor. If students look at the top of the mBot, they can see there are 4 ports, numbered 1-4. The line follower should be connected to port 3. Each port is labelled with multiple colours. If you look at the sensors they also have a coloured label. A sensor can be connected to any port labelled with its colour. If you look at the line follower you can see it is labelled blue, so could be connected to any of the ports but for this task leave it connected to port 3.

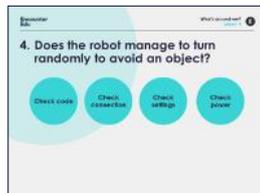
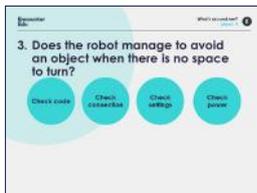
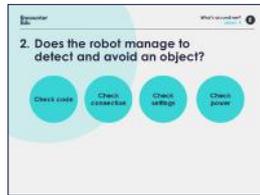
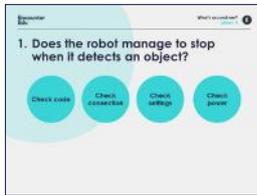
Hand out **Student Sheet 4b**, one per group. Student groups should work through the challenge, and then test their mBots.

Slides 8-11 are optional supporting slides to help show your students some of the critical coding blocks for this lesson.

Step

4

10 mins



Using **slide 12-15**, ask the class to raise their hands if their robot stopped in time / turned in time? Were they also able to get the robots to avoid obstacles that were too close and also to turn randomly? For all the other groups do they know why it didn't? If they don't know can any other group tell them what might be going wrong?



The most common problems are driving too fast or making the distance being checked by the ultrasonic sensor too small. If the robot has no space to turn, students need to stop and reverse the robot first.

For random turns make sure students are using the correct range.



If there is spare time, student groups may have time to try again to see if they can get their mBot to avoid an obstacle.

5

10 mins



This final section can be conducted as a discussion or short written answer exercise.



Optionally, students answer the question on **slide 16**. A whole class discussion can follow.



Discuss responses as a class.

How could we use this sensor in driverless cars?

- Similar to our robot we could use it to react and avoid other obstacles like people or cars.

Is this sensor enough?

- No.

Can you think of any problems you might come across?

- You might have noticed that this sensor can only see what is directly in front of it meaning it will miss anything to the side or behind it. This means it might not be able to see all obstacles which would result in an accident. Additionally, there are so many other things we have to pay attention to on the road like traffic lights, line markings, etc.

What other things would we have to sense?

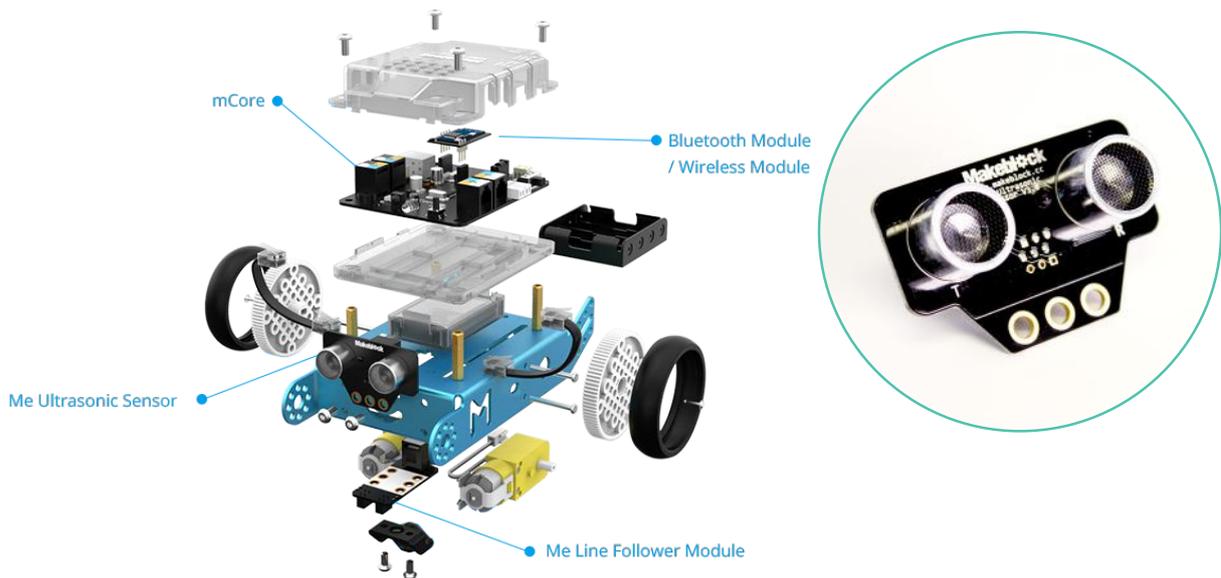
- Having more sensors all around the car might help. Also, another sensor like a camera to obey traffic signals would help to avoid collisions. The driverless car might also need to detect sirens to give way to the emergency services. Driverless cars would also be safer if they were to feel different road surfaces and adjust their speed to drive safely.

Explaining the ultrasonic sensor

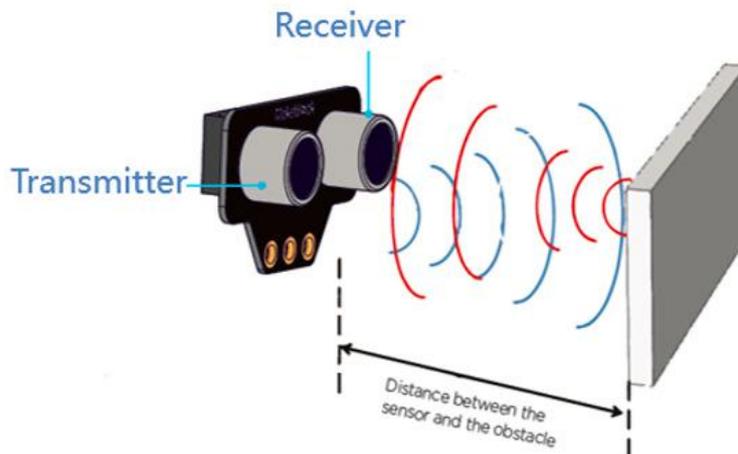


How the ultrasonic sensor works

The ultrasonic sensor measures distance. It is on the front of the mBot and looks like a set of eyes. We can use it to detect obstacles.



One of the 'eyes' transmits a sound, and the other waits for the echo of the sound to return. From the time this takes, the distance of the object from the sensor can be calculated. The ultrasonic sensor has a range of 3-400cm.



STUDENT SHEET 4a

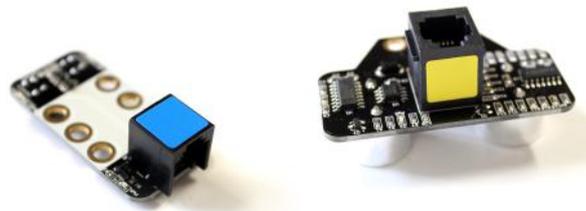
Reading the ultrasonic sensor

In order to control the robot when there is an obstacle, we need to know what information is coming from the ultrasonic sensor.

First, we should check our sensors. If you look at the top of the mBot you can see there are 4 ports, numbered 1 - 4.



Each port is labelled with multiple colours. If you look at the sensors they also have a coloured label. A sensor can be connected to any port labelled with its colour.



The ultrasonic sensor should be connected to **port 3**. If you look at the ultrasonic sensor you can see it's labelled yellow so could be connected to any of the ports but for this task leave it connected to port 3.



We use the ultrasonic sensor block to read the sensor. The block returns a number depending on the distance between the robot and the obstacle. For example, if the sensor detects an obstacle 20cm away, the block returns the value 20.

ultrasonic sensor Port3 distance

Note that at distances less than 3cm and more than 400cm the sensor is not accurate.

STUDENT SHEET 4a

Introducing the coding blocks

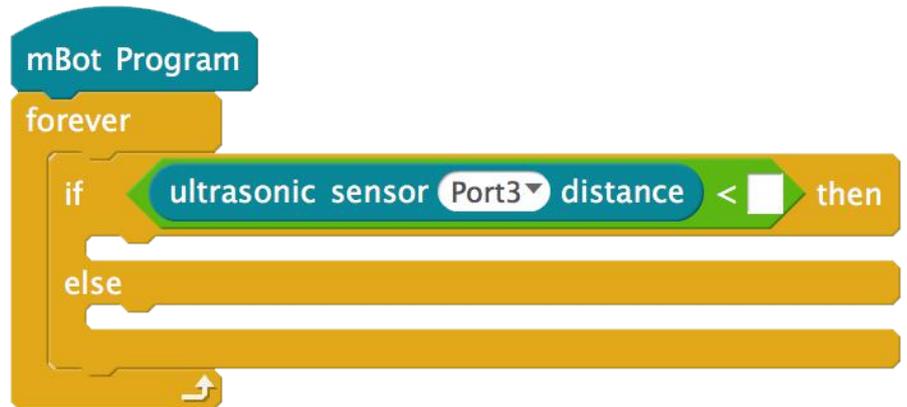
Coding the ultrasonic sensor is similar to coding the line follower. We need a **forever loop** which will just run the code inside it as long as there is power going to the robot.



We also need an **if-else** block which is known as a **conditional statement**.



Based on this, the start of our program for detecting an obstacle would look something like this. Use the instructions on **Student Sheet 4b** to build this code and then complete the program.



Coding the ultrasonic sensor



Today we are going to learn how to code a car to avoid obstacles.
Let's get moving!

1. In the robots blocks under mBot drag out the block and put it at the top of your program. All our mBot programs must start with this block.



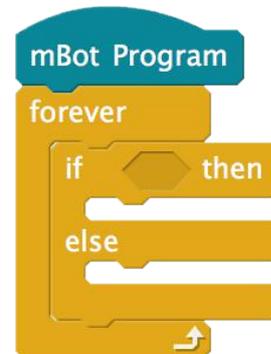
 If you don't see this block click on the Boards menu and make sure mBot (mCore) is selected.

2. To ensure that the mBot will always avoid obstacles, we are going to use the forever block.

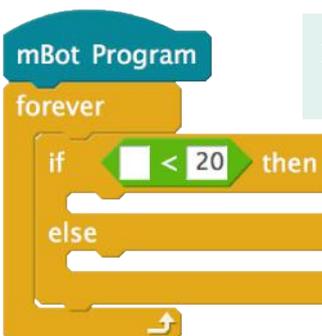


 This block is gold so is in the control type blocks.

3. We are now going to add the first of our if-else blocks to use the output from the ultrasonic sensor to control the car.



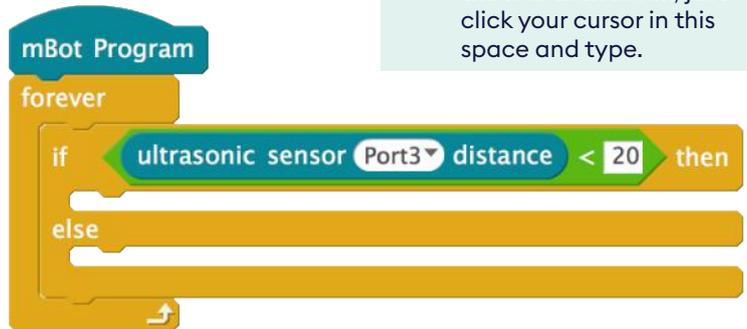
4. In the hexagonal space, we need to add an operator block. We want the **less than** block, so we can set the distance of an obstacle that is too close. In the example on the right, we have typed '20' for 20 cm, but you can set your own distance.



 Why do we use the < block instead of the = block?

STUDENT SHEET 4b

5. Adding the ultrasonic sensor block checks if the ultrasonic detects anything less than the distance we set.



⚙️ To change the number in the less than block, just click your cursor in this space and type.

6. Complete your code to make the robot react to avoid the obstacle.

💡 How could you do this? What options do you have?

Load the code!

7. To load the program onto the robot connect it to the computer using the USB cable – flat end into the computer and square end into the port marked USB on the robot.



8. We next need to tell the computer what port the robot is connected to. Click the Connect menu -> Serial Port and select the last port.

⚙️ If you have more than one option it will always be the bottom one.

9. Next click on “Upload to Arduino”.

⚙️ Make sure your power switch on the mBot is set to on and the power supply is connected.

10. Once the program is loaded onto the robot it will run straight away so place it on the floor.

⚠️ Be careful it doesn't run off the desk or table.

11. To restart the program either press the reset button on the mBot or turn the power switch off and on.



STUDENT SHEET 4b

More challenge

12. Can you write the code for another way to avoid the obstacle? Try to come up with as many as you can.

💡 How could you do this? What options do you have?

13. What happens if an obstacle is placed in front of the robot and the robot doesn't have enough time to turn? What else can your robot do besides turn? Write a program to solve this problem.

14. Sometimes it's helpful to turn in a random direction. To do this we'll need another if-else block, and also a new block called a **random number generator**.

```
pick random 1 to 10
```

⚙️ When choosing a direction to turn, there are really only two options: left or right. You should change the block to read **pick random 0 to 1**. This will make sure you are only given two options: 0 or 1.

15. We'll also need an equals block to check if we are given 0 or 1.

16. This is how your code might start. Complete the code to tell your robot what to do if the number equals 0 and what to do if it doesn't equal 0.

⚙️ Don't forget! You may need the wait block to make sure your robot has enough time to complete your instructions.

```
wait 1 secs
```

```
mBot Program
forever
  if ultrasonic sensor Port3 distance < 20 then
    if pick random 0 to 1 = 0 then
      [ ]
    else
      [ ]
  else
    [ ]
```

Coding the ultrasonic sensor

Making the robot stop

Solution to make the robot stop when it detects an obstacle.

1 mBot Program

2 forever

3 if ultrasonic sensor Port3 distance < 20 then

4 run forward at speed 0

5 else

6 run forward at speed 100

Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.

Line 3-4: If-else block. If an obstacle is detected less than 20cm away, stop.

Line 5-6: Else, drive forward.

Why do we use the < block instead of the = block?

If we use = the ultrasonic sensor has to detect this distance exactly. This distance might never be recorded as it's constantly sending out signals measuring the distance to obstacles but it might miss the exact value and the robot could end up crashing into an obstacle.

Make the robot react to avoid obstacle

There are multiple ways to avoid an obstacle.

Method 1: Turn one direction whenever the robot detects an obstacle

The robot could turn left or right when it encounters an obstacle. The simplest solution is to just turn left every time or just turn right every time.

1 mBot Program

2 forever

3 if ultrasonic sensor Port3 distance < 20 then

4 turn left at speed 100

5 else

6 run forward at speed 100

Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.

Line 3-4: If-else block. If an obstacle is detected less than 20cm away (you may want to make this larger than the check in the first program so the robot has enough time to turn), turn left. This will keep happening until the obstacle is no longer detected as the if condition will constantly be checked as it's in a forever loop.

Line 5-6: Else, drive forward.

Method 2: Reverse then turn one direction whenever the robot detects an obstacle

If an obstacle is too close or appears suddenly (if we place an obstacle in front of the robot while it's driving), it may be necessary to have the robot reverse and then turn later.

Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.

Line 3-7: If-else block. If an obstacle is detected less than 20cm away (this is bigger than the check in the first program so the robot has enough time to turn), reverse backward and then turn left. This is to give the robot enough space to turn and avoid crashing. This will keep happening until the obstacle is no longer detected as the if condition will constantly be checked as it's in a forever loop.

Line 8-9: Else, drive forward.

Method 3: Turn a random direction whenever the robot detects an obstacle

A more complex solution would see the robot picking at random whether to turn left or right each time it encountered an obstacle.

Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.

Line 3-5: If-else block. If an obstacle is detected less than 20cm away, pick a number between 0 and 1 randomly. If the number is 0 turn left.

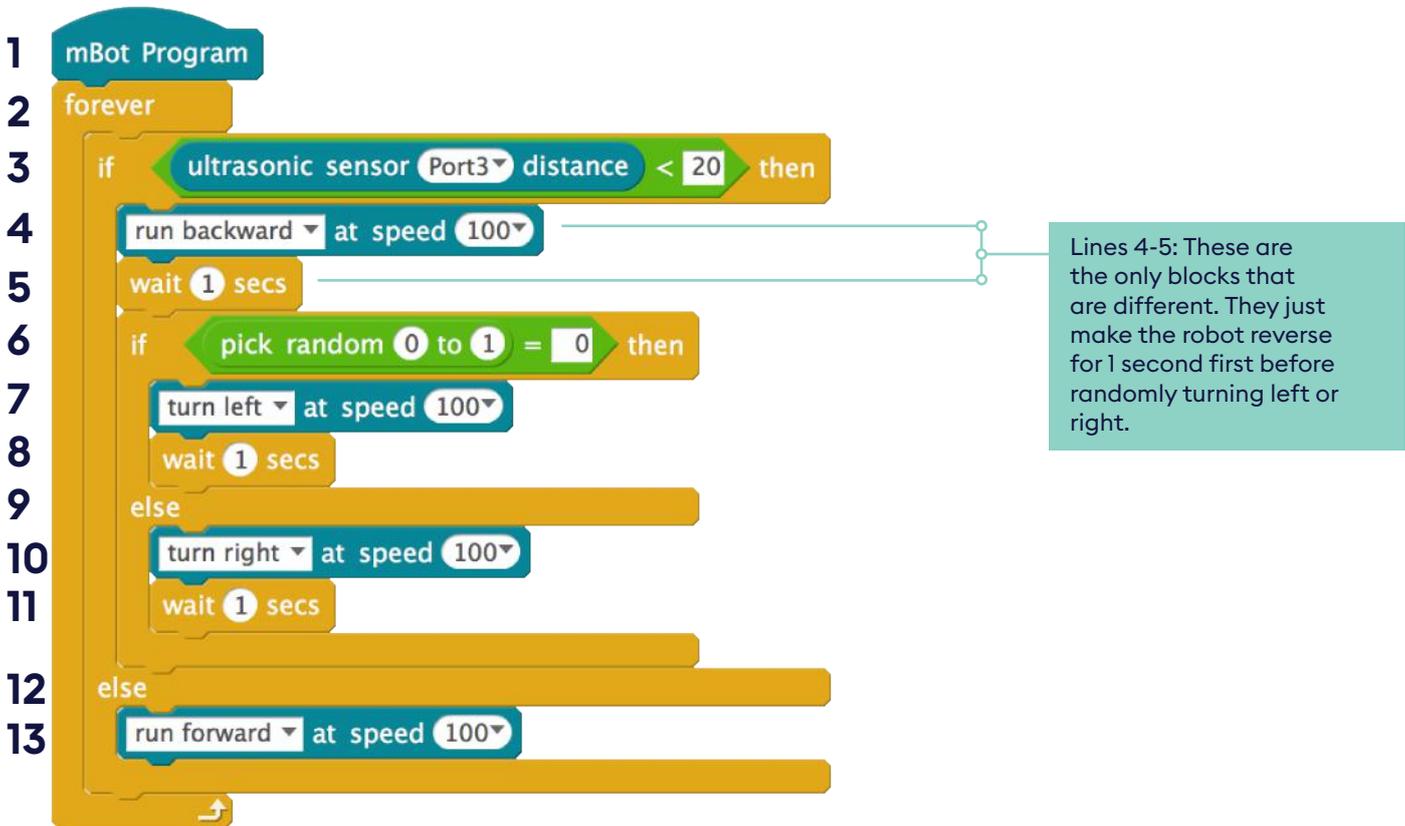
Line 6: Without the wait block, the robot will never make a full turn unless it randomly generates the same number repeatedly, which wouldn't make it a very good random number generator.

Line 7-9: If it's equal to 1 turn right.

Line 10-11: If there are no obstacles, then just drive forward.

Method 4: Reverse then turn a random direction whenever the robot detects an obstacle

This last solution is more complex and makes the robot reverse slightly before performing a random turn.



```

1 mBot Program
2 forever
3   if ultrasonic sensor Port3 distance < 20 then
4     run backward at speed 100
5     wait 1 secs
6     if pick random 0 to 1 = 0 then
7       turn left at speed 100
8       wait 1 secs
9     else
10      turn right at speed 100
11      wait 1 secs
12   else
13    run forward at speed 100
  
```

Lines 4-5: These are the only blocks that are different. They just make the robot reverse for 1 second first before randomly turning left or right.

More challenge notes

If an obstacle is too close or appears suddenly, an appropriate solution would require reversing before turning. See method 2 or 4 above.

If we want the robot to turn randomly rather than just one direction, then a random number generator is necessary. See method 3 and 4 above for solutions that make the robot turn randomly.

Lesson 5: Safety and signalling

In the fifth lesson of this unit of work, your class will learn about how technology and people interact. They will learn about signalling movement and giving warnings with light and sound. Students will use code to control their robot car's LEDs and buzzer to produce lights and sounds for a variety of different scenarios. They will then combine an input – the ultrasonic sensor – and an output – the LEDs and buzzer – to create a proximity sensor. Finally, they will discuss the other sensors and signals they think would be useful for their robots and an autonomous vehicle in real life.

Resources in this book:



Lesson Overview 5



Teacher Guidance 5



Student Sheet 5a: Safety and signalling



Answer Sheet 5a: Safety and signalling



Subject Update: Tips for teaching coding

Subject Update: Coding tips: writing elegant code

Subject Update: Coding tips: failing and debugging

Subject Update: Coding tips: commenting

Subject Update: Troubleshooting guide

Resources available online:



Slideshow 5: Safety and signalling



Video: Safety and autonomous vehicles



Image: Comparing my mBot to a driverless car interactive

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

Safety and signalling



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems
- Understand simple Boolean logic and some of its uses in circuits and programming
- Use a programming language to solve a variety of computational problems

Resources



Slideshow 5:
Safety and signalling



Student Sheet 5a:
Safety and signalling



Answer Sheet 5a:
Safety and signalling



Video:
Safety and autonomous vehicles



Image:
Comparing my mBot to a driverless car interactive



Subject updates:

- Tips for teaching coding
- Coding tips: writing elegant code
- Coding tips: failing and debugging

Kit (per group)

- mBot with remote
- Laptop or tablet with mBlock
- Materials to serve as obstacles

Lesson overview

In the fifth lesson of this unit of work, your class will learn about how technology and people interact. They will learn about signalling movement and giving warnings with light and sound. Students will use code to control their robot car's LEDs and buzzer to produce lights and sounds for a variety of different scenarios. They will then combine an input – the ultrasonic sensor – and an output – the LEDs and buzzer – to create a proximity sensor. Finally, they will discuss the other sensors and signals they think would be useful for their robots and autonomous vehicles in real life.

Lesson steps

1. Video opener (5 mins)

Introduction to safe cities and driverless cars. Today's challenge is to signal your robot cars' movements and point out hazards using lights and sound.

2. Classroom discussion (10 mins)

Students will discuss the video and talk about the variety of ways computers and humans 'talk' to each other.

3. Make (25 mins)

Introduction to the LEDs and buzzer hardware and how to control them with code. They should be able to signal movements with light and noise, then create a proximity warning using the ultrasonic sensor.

4. Test (10 mins)

Test the robot works as expected. What have you learned from the other groups' approach to signalling?

5. Reflect (10 mins)

Students will reflect on their learning, including problems they had and how they solved them.

Learning outcomes

- Understand what we mean by signalling and why it is needed to keep people safe
- Discuss the best ways for robots to signal to humans
- Understand, describe and control LEDs and buzzers
- Integrate inputs (ultrasonic sensor) to outputs (LEDs or buzzer)
- Debug programs
- Identify and share problems encountered with solutions

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

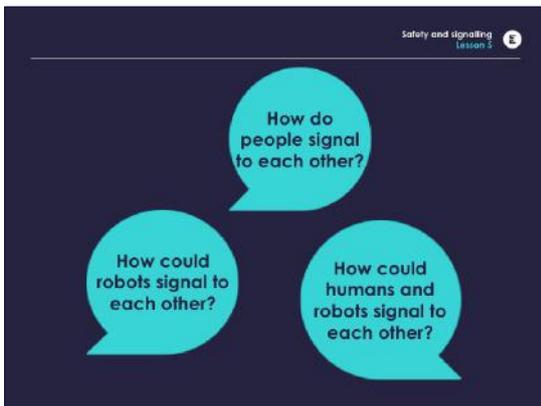


Using **slide 3** introduce students to the lesson where they are going to learn how to code the remote control, buzzer and LEDs for their robots. They are going to start by watching a video that discusses some of the safety considerations for driverless cars and how they interact with their environment. Today's challenge is to code the robot to signal to pedestrians what it is doing.



Show your class the video **Safety and autonomous vehicles**.

2
10
mins



Consider using a think-pair-share structure for these three questions as well, with students listing possible answers on their own for four minutes before sharing in pairs for two minutes. The whole class discussion element would then take a further four minutes.



Manage a whole class discussion using the questions on **slide 4**.

How do people signal to each other?

- They make noise, e.g. clapping, talking or shouting.
- They use hand gestures like pointing.

How do robots signal to each other?

- They can signal by making noise or flashing lights.
- They can also send signals or messages to each other through things like Bluetooth and Wi-Fi. When using the remote control, it sends a signal using infrared light. The robots can also send signals to each other using infrared light, but they have to be able to see each other, just like you have to point your remote control at the TV for it to work.

How could humans and robots signal to each other?

- They already do. When cars signal, other drivers see it and react. When we have driverless cars on the road, they will still signal in the same way to alert human drivers what they are going to do. When all the cars are driverless, we might not need to signal because the cars should just automatically communicate and react to other vehicles around them. For example, if the car in front slows down to turn, your driverless car will need to simply detect that the car is too close and slow down to prevent crashing.

Step

3
25
mins

Your challenge is to program the robot to signal.

- Program the robot to respond to the remote control.
- Make LEDs act as signals, red, green and blue.
- Make buzzer act as horn and produce warning sound.
- Make a warning signal respond to obstacle proximity.

Remote control

If remote **A** pressed

RGB LEDs

Set led on board **R** red **G** green **B** blue

Using RGB LEDs

Red	Green	Blue	Result
255	0	0	Red
255	255	0	Yellow
255	255	255	White

Buzzer

Play tone on note **C2** beat **D8**

STUDENT SHEET 5a

Safety and signalling

In this lesson, you will code the robot to respond to remote control steering. You will also operate the LED lights and buzzer based on the steering. Then you'll also create a proximity sensor by combining the input from the ultrasonic sensor and the output from the LEDs and buzzer.

Let's get moving!

Part 1. Using the remote control to steer

This code makes the robot drive forward when the up arrow is pressed.

Part 2. Using the LEDs to add signals to the movements

Each set of LEDs has three separate colours. We can control the overall colour shown by combining the strength of each colour, going from 0 for no light to 255 for full power. The table below shows how to mix the lights to make red, yellow and white.

Red	Green	Blue	Result
255	0	0	Red
255	255	0	Yellow
255	255	255	White



During this section, you will introduce students to three additional elements for the mBot: the remote control, LEDs and buzzer. Hand out a copy of **Student Sheet 5a** to each group.



Use **slide 5** to share this lesson's challenge with the class.

Introduce the remote control using **slide 6**. Remind students that they used this to control the robot in Lesson 1. Students are going to write their own version of the program the robot was running. We can program the robot to respond to any of the buttons on the remote control using the remote control block. We just need to change "A" to whatever button we want it to respond to.

Using **slide 7** introduce students to the RGB LEDs and how to program them. The mBot has 2 RGB LEDs. These are Light Emitting Diodes or simply lights which can be programmed to display a variety of different colours by mixing different amounts of red, green and blue light.

We can use these to make the mBot signal. These lights are programmed using the LED block. We can set all the lights to the same colour or control the left and right light individually. To signal we need red, yellow and white lights.

Explain how to make different colours using **slide 8**. Each colour can take the value 0 to 255 where 0 means none of that colour and 255 means the maximum brightness of that colour. We want to turn on both lights to red to signal braking. We can do this with the previous block and red set to 255 and the others set to 0. To turn the light yellow we need to set the LED block to red 255, green 255 and blue 0. By putting all colours to maximum brightness they cancel each other out to produce white.

Use **slide 9** to explain the buzzer and how to program it. The mBot also has a buzzer on it. We can use this to add a horn and reverse warning sounds. For this we need the play tone block. We can change the note the block plays, C2 is the lowest note it can play and D8 the highest, and the length (beat) of the note.



Students work through the challenges on **Student Sheet 5a** and then test their mBots.



If students have extra time, they can compare ways their robot car is like a driverless car by exploring the **Comparing my mBot to a driverless car** interactive image.

Step

4

10
mins

A presentation slide titled "Does the robot behave as expected? What have you learned from other groups?". The slide features four teal circles arranged horizontally, each containing a checkmark and a label: "Check code", "Check connection", "Check settings", and "Check power". The slide also includes the "Encounter Edu" logo and "Safety and signalling Lesson 3" text.



Using **slide 10**, ask the class to raise their hands if their robot behaved as expected? For all the other groups do they know why it didn't? If they don't know can any other group tell them what might be going wrong?



The most common problems are:

- For the remote control, see the notes in the **Answer Sheet 5a**.
- For the RGB LEDs, check to see that the lights turn off after you've finished signaling. While moving forward the LEDs should be turned off.
- For the buzzer, it will be hard to hear if the note is too high and for too short a time.
- For the proximity sensor, check that the extra if statement for the proximity sensor is placed inside the forever loop but outside of other if statements. Check that appropriate distance checks were used (10cm, 20cm & 30cm). See additional notes in **Answer Sheet 5a**.



What have you learned from the other groups' approach to signaling? Is it clear? Is it too early or too late? Any other issues?

5

10
mins

A presentation slide titled "Does the robot signal like a driver does?". The slide contains a text box with the following text: "The robot does / does not signal like a driver does, because... This could present problems, because...". The slide also includes the "Encounter Edu" logo and "Safety and signalling Lesson 3" text.



This final section can be conducted as a discussion or short written answer exercise.



Optionally, students answer the question on **slide 11**. A whole class discussion can follow.



Does the robot signal like a car does? If not, what do you need to change? Some people might say it indicates too late as it only does this when you're turning. We could fix this in a few ways but some ideas they might come up with are:

- Have a separate button to indicate. This would work but it would be up to the driver to remember to use it so not a great solution.
- When a button to turn is pressed, the robot could indicate for a set amount of time first before turning. This is a better solution.



Can you think of any other ways you might use the sensors you now know in a driverless car? The line following sensor could be used to make sure you don't cross lines in the road. We could add more ultrasonic sensors to check we're not going to collide with anything as currently the robot only checks for things in front of it. What other kind of sensors might be useful? A colour sensor would be useful to see things like traffic lights or a camera could also be used for this and for seeing other things like signs.

Safety and signalling



In this lesson, you will code the robot to respond to remote control steering. You will also operate the LED lights and buzzer based on the steering. Then you'll also create a proximity sensor by combining the input from the ultrasonic sensor and the output from the LEDs and buzzer.

Let's get moving!

Part 1. Using the remote control to steer

This code makes the robot drive forward when the up arrow is pressed.

1. Can you complete the program to make it respond to the left, right and back arrows?

```

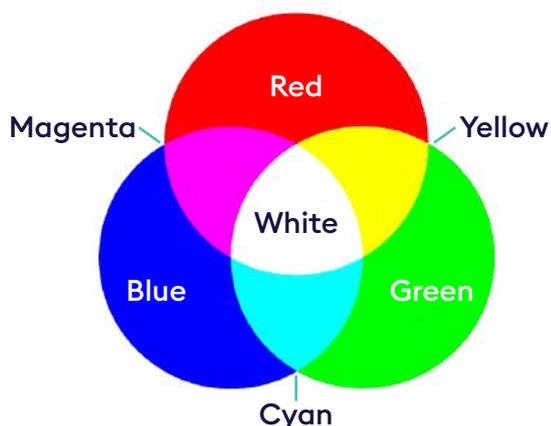
mBot Program
forever
  if ir remote ↑ pressed then
    run forward at speed 255
  else

```

We are using a forever loop here because we want the robot to always respond to our remote control commands.

Part 2. Using the LEDs to add signals to the movements

Each set of LEDs has three separate colours. We can control the overall colour shown by combining the strength of each colour, going from 0 for no light, to 255 for full power. The table below shows how to mix the lights to make red, yellow and white.



Red	Green	Blue	Result
255	0	0	Red
255	255	0	Yellow
255	255	255	White

STUDENT SHEET 5a

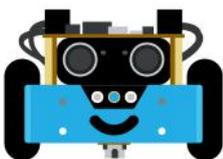
We can use the LED block to control which LEDs and which colours are operated.

2. If you adjust the red to 255, this block makes all the LEDs turn red. We can use it to signal braking. Since the command to brake is the same as the command not to move we should put this block with the block telling the robot to move at speed 0.

3. Next let's add in blocks to signal indicating left and right turns. Add blocks to your program to make the mBot turn the appropriate LED yellow while turning.

4. Finally, let's add reverse lights to the mBot. Add a block to make the robot turn all lights white while reversing.

5. Test your mBot.



```
set led on board all red 0 green 0 blue 0
```

```
run forward at speed 0  
set led on board all red 255 green 0 blue 0
```



Remember we only want the left or right light to come on to signal the turn, not both at the same time.



Do we need to add any more blocks to appropriately signal the robot's movements? What should the lights do when the robot is moving forward?

Part 3. Using the buzzer to add a reverse warning and a horn

6. Let's add in a block to play a reverse warning sound when the robot is moving backwards.

```
play tone on note C4 beat Half
```



Choose whatever note and beat you want. Have a play around with it to see what you like best.

7. Let's also add a horn. Add an extra piece of code to your program to sound the horn when you press the A button on the remote control.



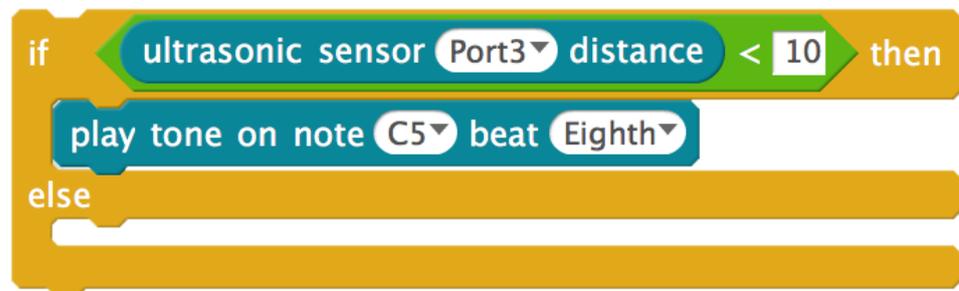
Again, you can pick the note and beat.

STUDENT SHEET 5a

Part 4. Using the ultrasonic sensor and buzzer to create a proximity sensor

Next we are going to use the ultrasonic sensor to add a proximity sensor to your robot. This will act like parking sensors to notify you if there's an obstacle and tell you roughly how close you are to it, using different notes and beats.

8. Add in the following code underneath the code for your horn. This will play a C5 note for an eighth of a beat when there is an obstacle less than 10cm away.



```
if ultrasonic sensor Port3 distance < 10 then
  play tone on note C5 beat Eighth
else
```

9. Add other tones for when an obstacle is 20cm and 30cm away. Make sure to adjust the beat of the tone so you can tell roughly how close you are to the obstacle, even with your eyes closed.

Safety and signalling

Part 1. Using the remote control to steer

There are a few ways to code remote control steering, but we have provided starter code for the “best” way to do it: nested if-else statements, or if-else statements within each other.

However, let’s look at 3 seemingly logical examples to code steering and some of their problems because some of your students might try these methods.

Example 1

If the code uses several separate if statements and looks like this...

The code consists of the following blocks:

- Line 1: mBot Program
- Line 2: forever loop
- Line 3: if ir remote ↑ pressed then
- Line 4: run forward at speed 255
- Line 5: if ir remote ↓ pressed then
- Line 6: run backward at speed 255
- Line 7: if ir remote ← pressed then
- Line 8: turn left at speed 255
- Line 9: if ir remote → pressed then
- Line 10: turn right at speed 255
- Line 11: run forward at speed 0

Callout boxes provide the following explanations:

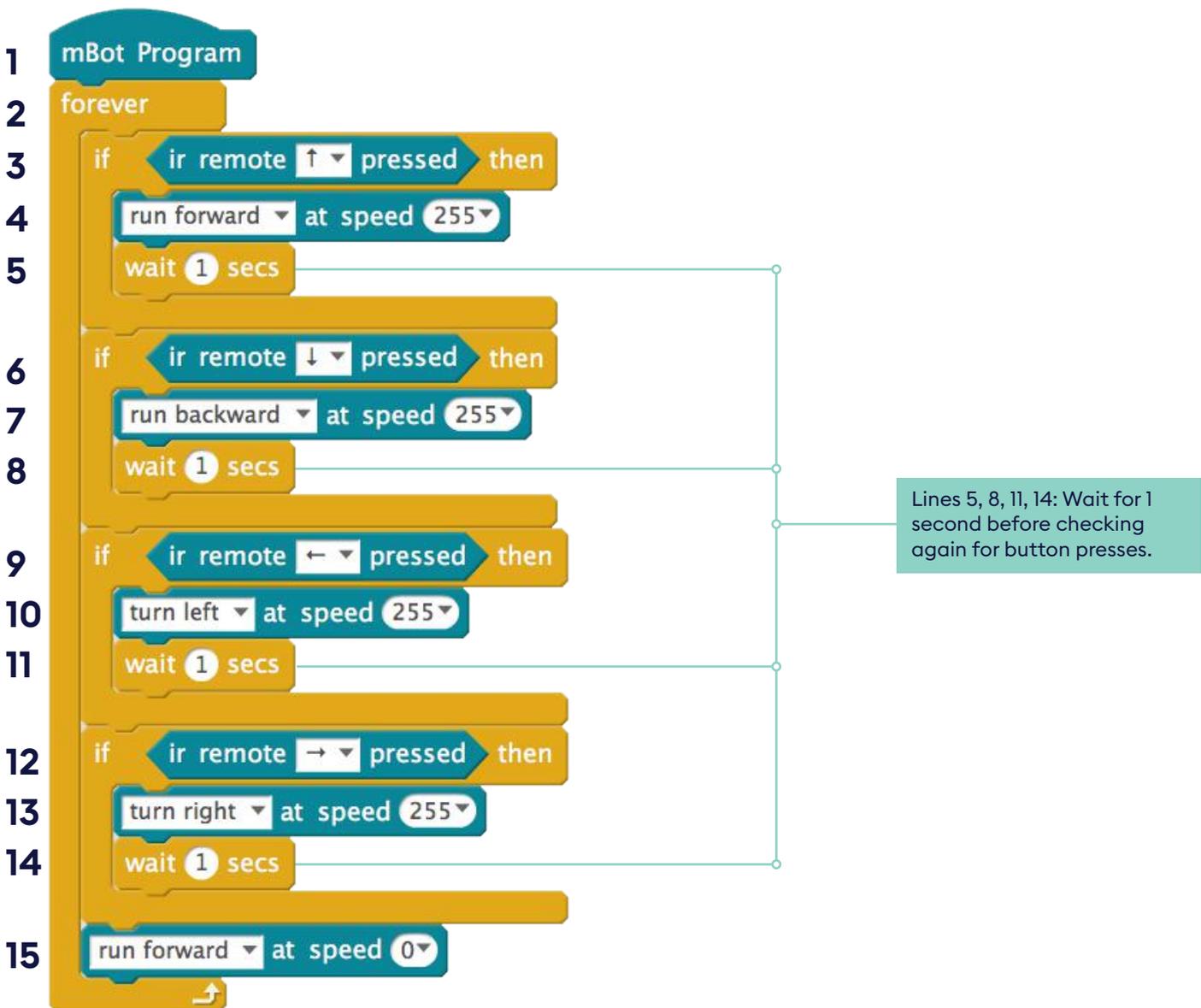
- Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.
- Lines 3-4: If the button with the up arrow is pressed then drive forward at full speed.
- Lines 5-6: If the button with the down arrow is pressed then drive backwards at full speed.
- Lines 7-8: If the button with the left arrow is pressed then turn left at full speed.
- Lines 9-10: If the button with the right arrow is pressed then turn right at full speed.
- Lines 11: If none of the buttons are pressed then stop the robot (turn the motors off).

Then the if statements blocks will compete with the stop block. This will cause the robot to behave strangely.

Part 1. Using the remote control to steer (continued)

Example 2

There are some different approaches we can take to solve this such as waiting for 1 second after each direction button pressed (see below). However, this means we have to drive in that direction for a whole second. We may not want to drive in that direction for a whole second but rather only for a fraction of a second. For example, to stop driving off a table. So this program works better than the previous one but still isn't great.



Part 1. Using the remote control to steer (continued)

Example 3

So what about adding an extra button to stop? This would work but makes the robot harder to control.

```
1 mBot Program
2 forever
3   if ir remote ↑ pressed then
4     run forward at speed 255
5   if ir remote ↓ pressed then
6     run backward at speed 255
7   if ir remote ← pressed then
8     turn left at speed 255
9   if ir remote → pressed then
10    turn right at speed 255
11  if ir remote B pressed then
12    run forward at speed 0
```

Lines 11-12: If the B button is pressed then stop the robot. Otherwise it will just keep doing what it was told to do by the last button press. For example, if you pressed forward the robot will keep moving forward until another direction button is pressed or you press the B button. This didn't happen for the first program as the robot would always be made to stop after each button press, as the command wasn't in an if statement.

Part 1. Using the remote control to steer (continued)

The “best” solution

The best solution uses nested if-else statements, or if-elses within each other. This stops the race condition we saw in the first program and gives us the best control over the robot. This is because in the first one it stopped the motors after every check meaning sometimes the robot got hardly anytime at all to turn its motors. In this one we only stop the motors after we check no other buttons have been pressed.

The code is as follows:

```

1 mBot Program
2 forever
3   if ir remote ↑ pressed then
4     run forward at speed 255
5   else
6     if ir remote ← pressed then
7       turn left at speed 255
8     else
9       if ir remote → pressed then
10        turn right at speed 255
11      else
12        if ir remote ↓ pressed then
13          run backward at speed 255
14        else
15          run forward at speed 0
  
```

Line 2: Forever loop. Keep running the code in this loop as long as the robot has power.

Line 3-4: If the button with the up arrow is pressed then drive forward at full speed.

Line 5-7: Else, if the button with the left arrow is pressed then turn left at full speed.

Line 8-10: Else, if the button with the right arrow is pressed then turn right at full speed.

Line 11-13: Else, if the button with the down arrow is pressed then drive backward at full speed.

Line 15: If none of the buttons are pressed then stop the robot (turn the motors off).

Part 2. Using the LEDs to add signals

For this program just make sure the light commands are in the correct if statement. It doesn't matter if the LED command is before or after the movement block just as long as it's in the correct part of the if statement. You need to add the red command after the motor has been turned to 0, the yellow command for the left light in the left arrow if statement, the yellow command for the right light in the right arrow if statement and the white command in the reverse if statement. A block the students will probably forget is to turn the lights off when the forward button is pressed. The only time when no lights should be on is when the robot is moving forward.

The code is as follows:

```

1  mBot Program
2  forever
3  if ir remote ↑ pressed then
4  set led on board all red 0 green 0 blue 0
5  run forward at speed 255
6  else
7  if ir remote ← pressed then
8  turn left at speed 255
9  set led on board led left red 255 green 255 blue 0
10 else
11 if ir remote → pressed then
12 turn right at speed 255
13 set led on board led right red 255 green 255 blue 0
14 else
15 if ir remote ↓ pressed then
16 run backward at speed 255
17 set led on board all red 255 green 255 blue 255
18 else
19 run forward at speed 0
20 set led on board all red 255 green 0 blue 0

```

Annotations:

- Line 4: Turn off all lights. When the car is moving forward no lights should be on but if you don't turn them off here then they might still be on showing the previous colour. For example, if you turned left just before moving forward then the left LED would still be yellow.
- Line 9: Turn on left indicator (left LED yellow).
- Line 13: Turn on right indicator (right LED yellow).
- Line 17: Turn on reverse lights (set all LEDs to white).
- Line 20: If no buttons are pressed then turn all the lights red and stop the robot.

Part 3. Using the buzzer to add a reverse warning and a horn

For the reverse warning, we need to add the tone block to the reverse if statement. For the horn, we need to add an extra if statement. It doesn't matter if we add this at the start or end of the loop. However, it should be an extra if statement and not added to the existing one, as the condition for sounding a horn doesn't rely on or affect what the motors are currently doing.

```

1  mBot Program
2  forever
3  if ir remote ↑ pressed then
4  set led on board all red 0 green 0 blue 0
5  run forward at speed 255
6  else
7  if ir remote ← pressed then
8  turn left at speed 255
9  set led on board led left red 255 green 255 blue 0
10 else
11 if ir remote → pressed then
12 turn right at speed 255
13 set led on board led right red 255 green 255 blue 0
14 else
15 if ir remote ↓ pressed then
16 run backward at speed 255
17 set led on board all red 255 green 255 blue 255
18 play tone on note D8 beat Eighth
19 else
20 run forward at speed 0
21 set led on board all red 255 green 0 blue 0
22 if ir remote A pressed then
23 play tone on note F7 beat Half
    
```

Line 18: Play the reverse warning sound.

Line 22-23: If the A button is pressed sound the horn.

Part 4. Using the ultrasonic sensor and buzzer to create a proximity sensor

To make the proximity sensor, we need to add an additional if-else statement. We have added it at the end of the loop. The students can use the same or different notes for each distance. The important part is to make the beat shorter the closer the robot gets to the obstacle so that the proximity can be roughly guessed just from the sounds the robot is making. In the sample solution, the beat has been halved each time from 30cm to 20cm to 10cm so we can tell how close we are to the obstacle by how fast the robot is beeping.

1 mBot Program

2 forever

3 if ir remote ↑ pressed then

4 set led on board all red 0 green 0 blue 0

5 run forward at speed 255

6 else

7 if ir remote ← pressed then

8 set led on board led left red 255 green 255 blue 0

9 turn left at speed 255

...

23 play tone on note F7 beat Half

24 if ultrasonic sensor Port3 distance < 10 then

25 play tone on note C5 beat Eighth

26 else

27 if ultrasonic sensor Port3 distance < 20 then

28 play tone on note B4 beat Quarter

29 else

30 if ultrasonic sensor Port3 distance < 30 then

31 play tone on note A4 beat Half

Lines 24 - 31: This is the additional code for the proximity sensor.

Lines 24-25: If the distance from the obstacle is less than 10cm, play C5 for an eighth of a note.

Lines 26-28: Else if the distance is less than 20cm, then play B4 for a quarter of a note.

Lines 29-31: Else if the distance is less than 30cm, then play A4 for half of a note.

Lesson 6:

What do I do next?

In the sixth lesson of this unit of work, your class will learn about some of the ways in which technology has failed in the past, and how engineers have worked to overcome those problems. They will then look back over the hardware and software they have been using in the past five lessons and combine these skills to complete challenges. This lesson should be used as an opportunity to consolidate learning, revisit any shaky territory and experiment with combinations of inputs, outputs and different ways of coding the mBot.

Resources in this book:



Lesson Overview 6



Teacher Guidance 6



Student Sheet 6a: Hardware and function cards

Student Sheet 6b: Smart city challenges



Answer Sheet 6b: Smart city challenges



Subject Update: Tips for teaching code

Subject Update: Coding tips: failing and debugging

Subject Update: Troubleshooting guide

Resources available online:



Slideshow 6: What do I do next?



Video: What do I do next?

Video: Failing is good



Image: Comparing my mBot to a driverless car interactive



360 Image: Look inside our self-driving car interactive

All resources can be downloaded from:

encounteredu.com/teachers/units/code-smart-11-14

What do I do next?



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Understand the hardware and software components that make up computer systems, and how they communicate with one another and with other systems
- Understand simple Boolean logic and some of its uses in circuits and programming
- Use a programming language to solve a variety of computational problems

Resources



Slideshow 6:
What do I do next?



Student Sheet 6a:
Hardware and function cards

Student Sheet 6b:
Smart city challenges



Answer Sheet 6b:
Smart city challenges



Video:
What do I do next?

Video:
Failing is good



Student Updates:

- Coding Tips: failing and debugging
- Troubleshooting guide

Kit (per group)

- mBot with remote
- Laptop or tablet with mBlock
- Black tape and a light surface
- Materials to serve as obstacles

Lesson overview

In the sixth lesson of this unit of work, your class will learn about some of the ways in which technology has failed in the past, and how engineers have worked to overcome those problems. They will then look back over the hardware and software they have been using in the past five lessons and combine these skills to complete challenges. This lesson should be used as an opportunity to consolidate learning, revisit any shaky territory and experiment with combinations of inputs, outputs and different ways of coding the mBot.

Lesson steps

1. Video opener (5 mins)

Introduction to learning through failure. Today's challenge is to look back at the skills the class have learned and combine them to complete one or more of the challenges.

2. Classroom discussion (5 mins)

Students will discuss the video and what they have learned by 'getting it wrong' the first time. Students will then choose one or more of the challenges to complete.

3. Make (30 mins)

Students will revisit the learning from the past five lessons and combine hardware and software skills to solve at least one of the challenges.

4. Test (10 mins)

Test the robot works as expected. Each group should share what challenge they chose and complete a brief demonstration.

5. Reflect (10 mins)

Students will reflect on their learning, including problems they had and how they solved them.

Learning outcomes

- Understand that failure is an important part of technology development
- Describe how failure and persistence can help them learn
- Link a variety of inputs to a variety of outputs using code
- Debug programs
- Share learning
- Identify and share problems encountered with solutions

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

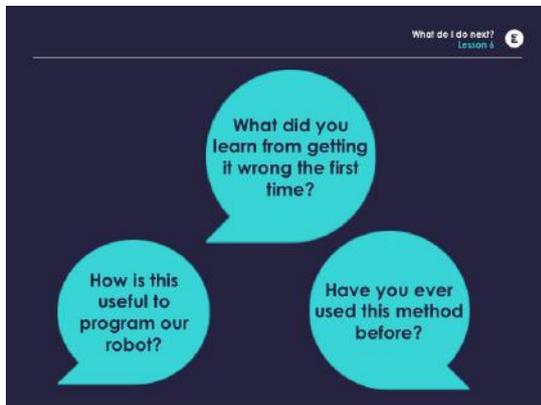


Using **slide 3** introduce students to the lesson where they are going to learn how to combine the skills and knowledge they have developed over the past five lessons and take on some Smarty City challenges.



Show your class the video **What do I do next?**

2
5
mins



Consider using a think-pair-share structure for these three questions on **slide 4**, with students listing possible answers on their own for two minutes before sharing in pairs for one minute. The whole class discussion element would then take a further two minutes.



What do you learn from getting things wrong the first time you try them?

- By trying you might get it wrong but you will learn from this and make a much better design the next time, based on what you learned.

How is this useful to program our robot?

- You will generally never get a program correct the first time you write it. Don't worry about getting it wrong.
- If you're unsure if something will work, test it before writing more of the code. Sometimes you won't be able to see an error just from looking at the code alone. The only way to know if the program works is to test it.

Have you ever used this method (trial and error) before?

- The class might tell you stories of where they have. If some students in the class have used trial and error before, ask them how they used it.



To emphasise that failing is the **First Attempt In Learning** show the video **Failing is good** (optional).

Step

3
30
mins

STUDENT SHEET 6a: Hardware and Function cards

Use a function card
Use the function card to identify the components of the mBot. Write down the name of the component and its function.

Make a function card
Write down the name of the component and its function. Use the function card to identify the components of the mBot.

Make a function card
Write down the name of the component and its function. Use the function card to identify the components of the mBot.

Make a function card
Write down the name of the component and its function. Use the function card to identify the components of the mBot.

STUDENT SHEET 6b: Smart city challenges

Navigation and Obstacle
Write down the name of the component and its function. Use the function card to identify the components of the mBot.

Navigation and Obstacle
Write down the name of the component and its function. Use the function card to identify the components of the mBot.

Work through as many of the challenges as you have time for.

- Make and solve an obstacle course.
- Add challenges to the course from work out how to get around them.
- Find the way out of a dead end.
- Navigate around a fallen tree.
- Navigate around a fallen tree blocking a path.



Hand out copies of **Student Sheets 6a** and **6b**, one per group. Student groups should also have access to laptops, mBots and other materials at this stage.



Go through the revision cards contained within **Student Sheet 6a** which can alternatively be used for home learning.



The class should use black tape to create an obstacle course for the robot to drive round or try to navigate out of a dead end. Get the students to pick one challenge to start. If there is time to spare get them to try another.



Explain the challenges using **Student Sheet 6b** and **slide 5**.



The teams should work through one challenge at a time. See **Answer Sheet 6b** for solutions.



If students have extra time, they can explore how what they have learned about their robot car compares to the development of larger autonomous vehicles using the **Comparing my mBot to a driverless car** interactive image, the **Look inside our self-driving car** interactive 360 image, or the **Look! Driving with no hands!** 360 video available online.

4
10
mins

Does the robot behave as expected? If not, what do you need to do to fix it?

- Check code
- Check connection
- Check settings
- Check power



Using **slide 6**, ask the class to raise their hands if their robot behaved as expected? For all the other groups do they know why it didn't? If they don't know, can any other group tell them what might be going wrong?



Most common problems are:

City tour - The black lines too close together or too narrow. Bad lighting can also confuse the sensor.

Obstacles - Not putting the if statements in the correct order. The obstacle must always be checked for first. See answer sheet.

Dead end - See answer sheet. Main problem is not turning for long enough.

Fallen tree - See answer sheet. Main problem is not calculating the length of the shape properly.

Fallen tree path - See answer sheet. Main problem is not calculating the length of the shape properly.

Step

5
10
mins



This is an opportunity for the groups to do a brief presentation. Use **slide 7** to help guide students.



Each group should explain to the class what challenge was chosen and give a short demonstration. Also discuss the problems the group had and how the group solved them.

STUDENT SHEET 6a: Hardware and function cards

Motor

The mBot has two motors, each controlling one of the large rear wheels. By controlling the motors, we can control how the mBot moves.



We use the movement block to control the motors:

```
run forward at speed 0
```

Here's an example of some code to move forward for 3 seconds and then stop:

```
mBot Program
run forward at speed 100
wait 3 secs
run forward at speed 0
```

What other types of motors might you use in robotics?

Is run forward the only option for the movement block? If not, what other choices do you have?

If you need any more help with using the motor, have a look at the Lesson 2 activities again.

Line follower

The mBot has a line follower, which uses two LEDs and two receivers to sense between black and white. The line follower can be used to code the mBot to follow a set route.



We use the line follower block to read the line follower sensor:

```
line follower Port2
```

Here's an example of some code to tell the robot to move forward if both sensors detected black:

```
if line follower Port2 = 0 then
  run forward at speed 100
else
  run forward at speed 0
```

How might you use the line follower for a smart city scenario?

Can you remember the different values for the outputs, e.g. black & black = 0?

If you need any more help with using the line follower, have a look at the Lesson 3 activities again.

STUDENT SHEET 6a: Hardware and function cards

Ultrasonic sensor

The ultrasonic sensor measures distance. It transmits a sound and waits for the echo of the sound to return. From the time this takes, the distance of the object from the sensor can be calculated.



We use the ultrasonic sensor block to read the ultrasonic sensor:

ultrasonic sensor Port3 distance

Here's an example of some code to tell the robot to stop if it detects an obstacle:

```
if ultrasonic sensor Port3 distance < 10 then
  run forward at speed 0
else
  run forward at speed 100
```

How might you use the ultrasonic sensor for a smart city scenario?

Can you code the mBot to do something else instead of stopping for an obstacle?

If you need any more help with using the ultrasonic sensor, have a look at the Lesson 4 activities again.

Remote control

The mBot can be controlled using a remote control. We can code each of the buttons to tell the robot to do different things.



We use the IR remote block to code the remote control:

ir remote A pressed

Here's an example of some code to tell the mBot to move forward if the up arrow is pressed:

```
mBot Program
forever
  if ir remote ↑ pressed then
    run forward at speed 255
  else
    →
```

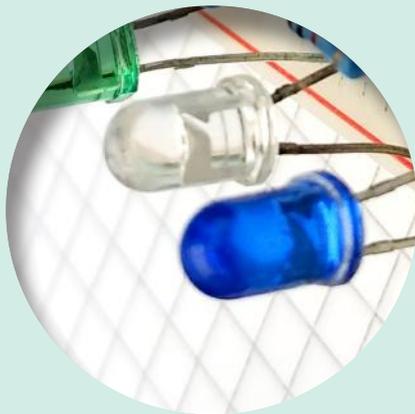
Can you think of other ways in which you would like to use the remote control?

If you need any more help with using the remote control, have a look at the Lesson 5 activities again.

STUDENT SHEET 6a: Hardware and function cards

LEDs

The mBot has LEDs, and each set has three separate colours. This means that the overall colour of the LEDs can be controlled by combining the strength of each colour.



We use the led block to code the RGB LEDs:

```
set led on board all red 0 green 0 blue 0
```

Here's an example of some code to tell the mBot to show red lights when the mBot is not moving:

```
mBot Program
forever
  if ir remote pressed then
    run forward at speed 255
  else
    run forward at speed 0
    set led on board all red 255 green 0 blue 0
```

Can you think of other ways to use the LEDs?

What kind of smart city scenarios would need you to use lights?

If you need any more help with using the LEDs, have a look at the Lesson 5 activities again.

Buzzer

The mBot has a buzzer which can be coded to make different sounds for different events.



We use the play tone block to code the buzzer:

```
play tone on note C4 beat Half
```

Here's an example of some code to tell the mBot to play a sound when the A button on the remote control is pressed:

```
if ir remote A pressed then
  play tone on note F7 beat Half
```

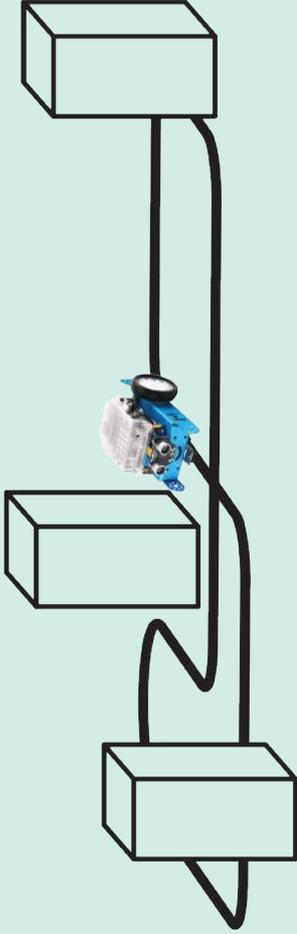
Can you think of other ways to use the buzzer?

What kind of smart city scenarios would need you to use the buzzer?

If you need any more help with using the buzzer, have a look at the Lesson 5 activities again.

STUDENT SHEET 6b: Smart city challenges

Take a city tour



Using black tape, create a course for your mBot to complete. Try not to put the lines too close together and if you have a cross roads or T-junction think about how you might be able to code the mBot.

You could also decorate your city by creating model landmarks and buildings along the route.

What sensor will you use?

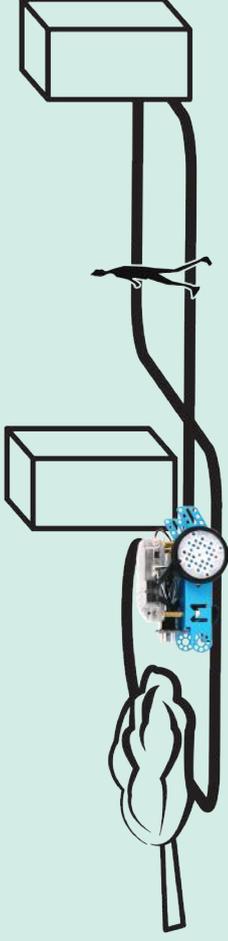
.....

What blocks will you need to use?

.....

.....

Watch out and keep safe



Use a course you have already created or make a new one using black tape. Place some obstacles on your course such as pedestrians, road works or fallen trees. Make the mBot stop when it encounters an obstacle, and then start moving again once they are removed.

What sensor will you use?

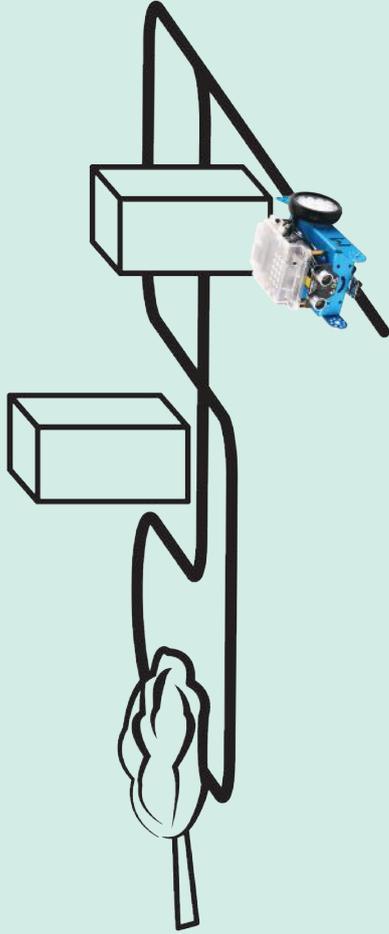
.....

What blocks will you need to use?

.....

.....

Escape from the dead end



Set up some obstacles to create a dead end like in the picture above. Program the mBot to find its way of the dead end after driving in.

This could be part of the course made for a city tour or a separate problem. Consider what the mBot should do after it has found its way out of the dead end.

What sensor will you use?

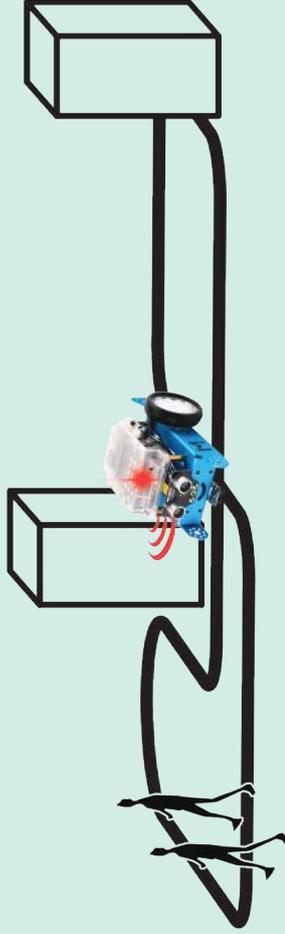
.....

What blocks will you need to use?

.....

.....

Make the city wondrous



This challenge brings an element of fun to the smart city. Program the LEDs and buzzers in a fun way. Maybe they respond to inputs from one of the sensors. Maybe they are on a timer.

This could be part of the course made for a city tour or a separate problem.

What sensor will you use?

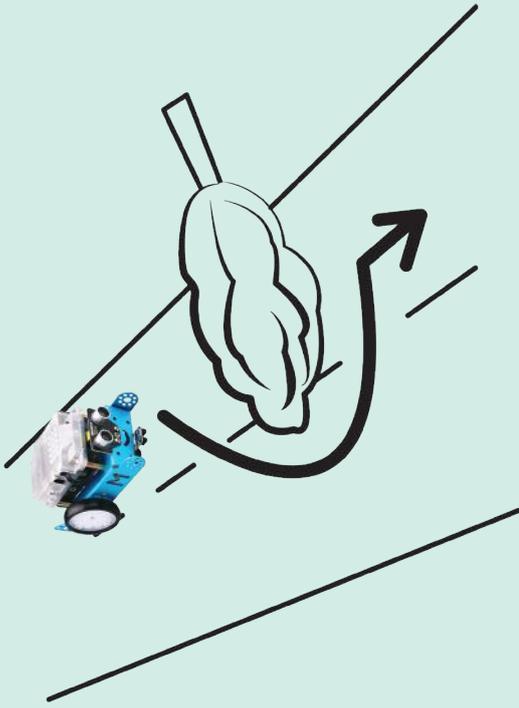
.....

What blocks will you need to use?

.....

.....

Drive around a fallen tree



Find an obstacle to be a fallen tree. Measure how long it takes the robot to drive length of the fallen tree. Make the mBot drive around the obstacle.

What sensor will you use?

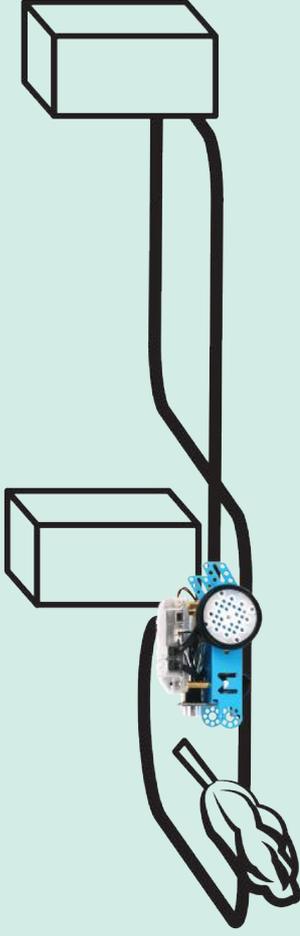
.....

What blocks will you need to use?

.....

.....

Drive around a fallen tree and continue touring



Use a course you have already created or make a new one using black tape. Place an obstacle on your course such as a fallen tree. Make the mBot drive around the obstacle and find the black tape again to continue following the path.

What sensor will you use?

.....

What blocks will you need to use?

.....

.....

Smart city challenges

Take a city tour on a path

This program is just the same as the first one in Lesson 3. When students create the course just make sure the black tape has clear white sections between it so the robot doesn't get confused.

The image shows a Scratch code block for an mBot program. The code is as follows:

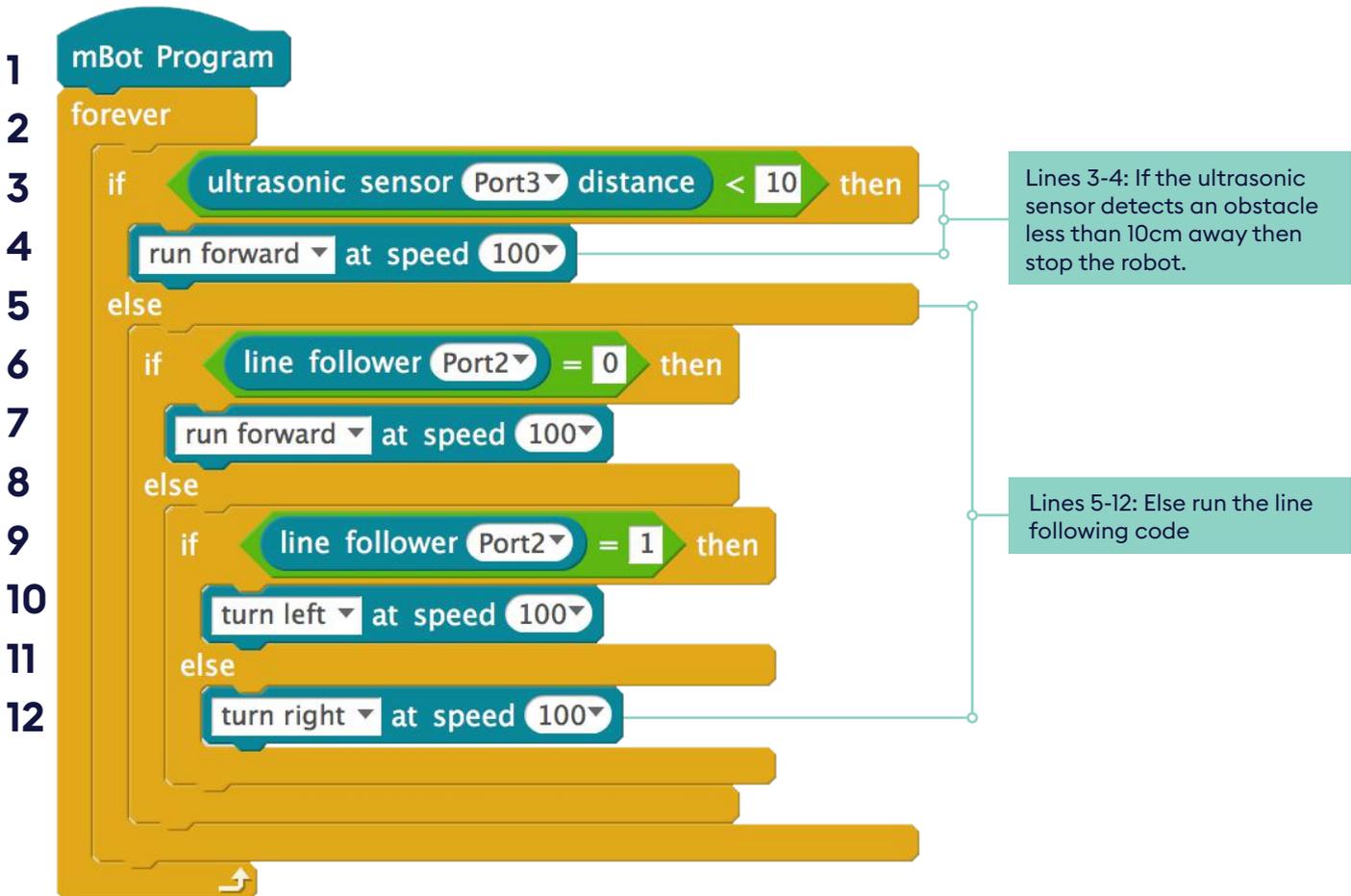
```
1 mBot Program
2 forever
3   if line follower Port2 = 0 then
4     run forward at speed 100
5   else
6     if line follower Port2 = 1 then
7       turn left at speed 100
8     else
9       turn right at speed 100
```

Three callout boxes provide explanations for specific parts of the code:

- Line 2:** Forever loop. Keep running the code in this loop as long as the robot has power.
- Line 3-4:** If-else block. If both sensors can see the black line then drive forward.
- Line 5-9:** Else another if-else block which adjusts the robot's direction so both sensors are on the line again. If only sensor 1 can see the black line then we have to turn slightly to the left till both sensors can see the black line again. Otherwise, we are too far to the left and have to turn right to fix the robot's direction.

Watch out and keep safe

This is just the line following program with an extra condition added in for the ultrasonic sensor telling it to stop when an obstacle is detected. As soon as the obstacle is removed the robot will start moving again.



Escape from the dead end

This program isn't as complex as it sounds. It's just the same as the program from lesson 4 for making the robot react to avoid an obstacle. The only difference is the wait block after the turn. This is because the ultrasonic sensor can only detect obstacles directly in front of it and this tells the robot to continue to turn for another second. This ensures that the robot completes the turn to avoid the obstacle, as otherwise it will start driving forward as soon as the obstacle clears the sensor. Make sure the students don't cheat by just making the robot turn before it drives into the dead end.

```

1  mBot Program
2  forever
3  if ultrasonic sensor Port3 distance < 10 then
4  turn right at speed 100
5  wait 1 secs
6  else
7  run forward at speed 100
  
```

Line 5: Make the robot turn for longer so it manages to get out the dead end

Make the city wondrous

This challenge is a very open-ended opportunity to add lights and sounds. Students might build programs like those from lesson 5 in which lights and sounds are produced when certain remote buttons are pushed. Below is a simple example of adding lights and music when the A button is pushed. Students can add LED and buzzer blocks to any of the other programs they've written.

```

1  mBot Program
2  forever
3  if ir remote A pressed then
4  set led on board all red 255 green 0 blue 0
5  wait 1 secs
6  set led on board all red 0 green 0 blue 255
7  wait 1 secs
8  play tone on note C4 beat Whole
9  play tone on note C6 beat Eighth
  
```

Line 4-7: Make the robot LEDs turn red and then blue

Line 8-9: Make the robot produce music

Drive around a fallen tree

For this example program answer, the robot took 3 seconds at engine speed 100, to drive the length of the obstacle. The students will have to determine the time it takes for the robot to drive the length and width of the obstacle they use. The program assumes the robot drives towards middle of the obstacle so to navigate around it, we turn left and drive forward for 2 seconds to be sure we clear the length of the obstacle. Then we turn right and drive forward for 3 seconds to ensure we clear the width of the end (you can measure this as well). Finally, we turn right and drive forward for 2 seconds and turn left again. We should be roughly on the path we would be on if the obstacle weren't in the way. Remember the amount of time you turn for depends on the floor surface and engine speed, and you will have to experiment with this. This program makes the robot go around the left side of the obstacle but it would also work to drive around the right side of the obstacle.

The code is as follows:

```

1  mBot Program
2  forever
3  run forward at speed 100
4  if ultrasonic sensor Port3 distance < 10 then
5    turn left at speed 100
6    wait 1.25 secs
7    run forward at speed 100
8    wait 2 secs
9    turn right at speed 100
10   wait 1.25 secs
11   run forward at speed 100
12   wait 3 secs
13   turn right at speed 100
14   wait 1.25 secs
15   run forward at speed 100
16   wait 2 secs
17   turn left at speed 100
18   wait 1.25 secs
  
```

Line 4-18: If the ultrasonic sensor detects a fallen tree less than 10 cm away, then drive around the tree.

Line 3: Drive forward until mBot detects an obstacle.

Lines 5-6: Turn left so the robot is parallel with the tree.

Lines 7-8: Drive the length of the tree. (In this example, it takes 3 seconds to drive the whole length of the fallen tree. If the mBot approaches the middle, 1.5 seconds and a bit of extra time will be needed to clear the length.)

Lines 9-12: Turn right to drive around the left end of the tree. Drive 3 seconds to clear the width (this can also be measure).

Lines 13-16: Turn right and drive to roughly the other side of the tree from where the robot detected the tree.

Lines 17-18: Turn left to be in position to keep driving along the original route.

Drive around a fallen tree and continue touring on a path

This program combines the program to drive around a known obstacle, the fallen tree example above, with the line following program. There are only a couple of differences: 1) we remove the first block to turn on the engine as the line following section of the program will do this and 2) the final drive forward doesn't need a wait block as it will keep running forward until it finds the black line again.

1 mBot Program

2 forever

3 if ultrasonic sensor Port3 distance < 10 then

4 turn left at speed 100

5 wait 1.25 secs

6 run forward at speed 100

7 wait 2 secs

8 turn right at speed 100

9 wait 1.25 secs

10 run forward at speed 100

11 wait 3 secs

12 turn right at speed 100

13 wait 1.25 secs

14 run forward at speed 100

15 wait until line follower Port2 < 3

16 else

17 if line follower Port2 = 0 then

18 run forward at speed 100

19 else

20 if line follower Port2 = 1 then

21 turn left at speed 100

22 else

23 turn right at speed 100

Line 3-16: If the ultrasonic sensor detects a fallen tree less than 10 cm away, then drive around the tree.

Line 15: Keep driving until the robot finds the path again.

Line 17-18: If-else block. If both sensors can see the black line then drive forward.

Line 19-23: Else another if-else block which adjusts the robot's direction so both sensors are on the line again. If only sensor 1 can see the black line then we have to turn slightly to the left till both sensors can see the black line again. Otherwise, we are too far to the left and have to turn right to fix the robot's direction.

Lesson 7: Designing our smart city pt. 1

In the last section of this unit of work, your class will take part in a Design Thinking Workshop that can be delivered as three one hour sessions or combined as a half day activity.

In part one of the workshop, your class will use personas to empathise with different types of people. They will then use these insights to brainstorm ways that robots and autonomous vehicles can improve lives or solve problems.

Parts two and three of the workshop see student groups select, refine and prototype ideas before presenting and demonstrating their proposals.

Resources in this book:



Lesson Overview 7



Teacher Guidance 7



Student Sheet 7a: User profiles

Student Sheet 7b: Empathy map



Subject Update: How can autonomous vehicles be useful?

Subject Update: Futures Thinking

Resources available online:



Slideshow 7: Designing our smart city pt. 1



Video: Design thinking

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

Designing our smart city pt. 1



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Design & Technology

- Use research and exploration to identify and understand user needs
- Develop specifications to inform the design of innovative, functional, appealing products that respond to needs in a variety of situations
- Use a variety of approaches to generate creative ideas and avoid stereotypical responses

Resources



Slideshow 7:

Designing our smart city pt. 1



Student Sheet 7a:

User profiles

Student Sheet 7b:

Empathy map



Video:

Design thinking



Subject Updates:

- How can autonomous vehicles be useful?
- Futures Thinking

Kit (per group)

- No additional kit required

Lesson overview

In the last section of this unit of work, your class will take part in a Design Thinking Workshop that can be delivered as three one hour sessions or combined as a half day activity.

In part one of the workshop, your class will use personas to empathise with different types of people. They will then use these insights to brainstorm ways that robots and autonomous vehicles can improve lives or solve problems.

Parts two and three of the workshop see student groups select, refine and prototype ideas before presenting and demonstrating their proposals.

Lesson steps

1. Video opener (5 mins)

Introduction to design thinking. This workshop's challenge is to use design to solve the problems of citizens living in a city of the future.

2. Classroom discussion (10 mins)

Students will discuss the video and share ways they've solved problems in that past.

3. Design thinking: empathise (15 mins)

Students will use character personas to empathise with different people and their travel related problems.

4. Design thinking: ideate (20 mins)

Students will complete a class brainstorming activity then work in groups to brainstorm ideas to make life better for the character personas.

5. Reflect (10 mins)

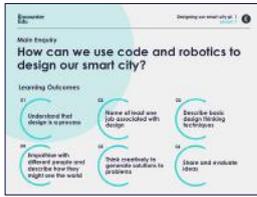
Students will reflect on their brainstorming activity then share their best and worst ideas.

Learning outcomes

- Understand that design is a process
- Name at least one job associated with design
- Describe basic design thinking techniques
- Understand that issues affect people in different ways
- Empathise with different people and describe how they might see the world
- Think creatively to generate solutions to problems
- Share and evaluate their own ideas

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

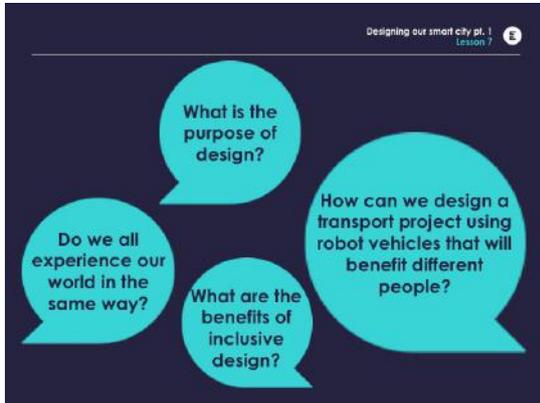


Using **slide 3** introduce students to the lesson where they are going to embark on their design challenge and use what they have learned to address societal needs.



Show your class the video **Design thinking**.

2
10
mins



Consider using a think-pair-share structure for these three questions as well, with students listing possible answers on their own for four minutes before sharing in pairs for two minutes. The whole class discussion element would then take a further four minutes.



Manage a whole class discussion using the questions on **slide 4**.

What is the purpose of design?

- Design is not just to make things look nice – it helps us make the world better and solve problems.

Do we all experience our world in the same way?

- One person may see something as good and another may see the same thing as bad.
- Empathy is very important in design. When designing something we must make sure we consider how different people will react to it. Something that is positive for one set of people may be negative for another – we should always aim to find a balance.

Highlight the importance of inclusive design.

- Designing for people with disabilities means that more people can participate and contribute to society.
- Many times inclusive design makes things more accessible for people with disabilities and also easier for people without the disability as well. Think about how helpful a ramp can be for someone in a wheel chair and how helpful it can be when you're rolling a heavy suitcase.
- Have they ever noticed any ways that a space or object is designed for someone with a disability?

How can we design a transport project using robot vehicles to benefit people who may use it?

- Start your class thinking about how what they've learned could be applied to real life scenarios.

Step

3

15 mins

Encounter Edu Designing our smart city pt. 1 Lesson 7

Think carefully about how each user...

- Thinks and feels
- What they might say and do
- What they could hear and see
- What their pains and gains are

STUDENT SHEET 7a

User profiles

Name: Rowen Barry
Age: 52

Occupation: Headline coach. Rowen is also the primary carer for her young children.

Where does he live? On the outskirts of town where rent is cheap.

How does he currently travel? Mostly by car. It's expensive to keep a car but the bus is infrequent and inconvenient.

How often does he travel? Their journey is to and from school to drop off the kids, then to and then work.

What is most important to him about his mode of transport? Convenience, speed and cost.

Name: Rosemary Sullivan
Age: 72

Occupation: Retired dentist.

Where does she live? In the centre of town.

How does she currently travel? Mostly by bus. It's free with her Senior Citizen Card and she likes looking out the window. She worries about getting hit by cars crossing the road as her weight can jostle the road.

How often does she travel? Rosemary takes around five journeys a week.

What is most important to her about her mode of transport? Cost and enjoyment.

STUDENT SHEET 7b

Empathy map

Pain: What would make this person unhappy?

Think and feel: What is important to this person? What do they worry about?

Gain: What would make this person happy?

Say and do: How would this person like to behave and how does that measure up against their actions on a day-to-day basis?

Hear and see: What is this person's environment like? Who influences their decisions?



Hand out copies of **Student Sheets 7a and 7b**. Provide each group with a couple of different user profiles and the same number of empathy maps.



Explain that the user profiles and empathy maps are design tools that help service and product designers work out the motivations, worries and habits of different people. Assign small groups one or two user profiles each and ask them to fill in empathy maps for each profile. You can use **slide 5** to highlight some of the things they might think about. You can also use the blank profile for pupils to make up their own users.



Students will work through the student sheets.



This is a design tool called an empathy map. It helps service and product designers work out the motivations, worries and habits of different people. Students should use it in conjunction with one or more user profiles, so they can empathise with how different types of people would use or be affected by any robot vehicle project they might design.

4

20 mins

Designing our smart city pt. 1 Lesson 7

How can we make people's lives better using our robot vehicles?

- Don't interrupt
- Keep your end users in mind
- Stay focused
- Wild ideas are encouraged
- Let everyone have a say
- Work together and build on other people's ideas
- Quantity over quality
- Make notes and diagrams so you don't forget great ideas



Explain that using their learning from the user empathy maps, your class will now think about problems people have with transport. In groups, they will now brainstorm ideas to make people's lives better using robot vehicles. Use the brainstorming guidelines on **slide 6** to keep them on track.



Students will probably come up with all sorts of answers, but make sure they are considering a range of issues from different viewpoints. The major takeaway from this exercise should be that we should think carefully about users when designing a feature, product or service. Just being cool, new and shiny doesn't equal good design!

Step

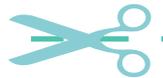
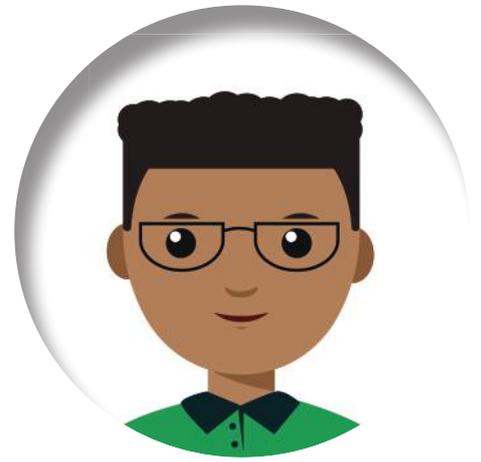
5

10
mins

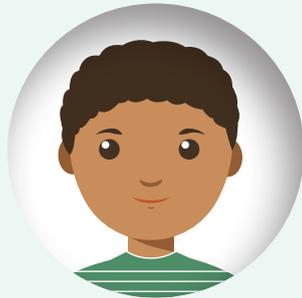


Using **slide 7**, ask students to reflect on the brainstorming exercise, sharing their best, worst and funniest ideas.

User profiles



Name:
Rowan Berry



Age:
36

Occupation:
Part-time cashier. Rowan is also the primary carer for two young children.

Where does he live?
On the outskirts of town where rent is cheap.

How does he currently travel?
Mostly by car. It's expensive to keep a car but the bus is infrequent and inconvenient.

How often does he travel?
Four journeys a day: to and from school to drop off the kids, then to and from work.

What is most important to him about his mode of transport?
Convenience, speed and cost.

Name:
Rosemary Sultana
Age:
72



Occupation:
Retired chemist
Where does she live?
In the centre of town.

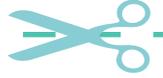
How does she currently travel?
Mostly by bus. It's free with her Senior Citizen Card and she likes looking out the window. She worries about getting hit by cars crossing the road as her eyesight isn't perfect any more.

How often does she travel?
Rosemary takes around five journeys a week.

What is most important to her about her mode of transport?
Cost and enjoyment.



User profiles



Name:
Ash Quince



Age:
19

Occupation:
Full-time student

Where does he live?
On campus at university, near the town centre.

How does he currently travel?
Ash is blind. He mainly travels on foot with his guide dog. His parents sometimes drive him but he'd prefer to be self-sufficient.

How often does he travel?
Several short journeys a day, plus a long journey once a month.

What is most important to him about his mode of transport?
Cost, safety and independence.

Name:
Holly Apple



Age:
49

Occupation:
Marketing manager

Where does she live?
In the suburbs, but she works in the town centre.

How does she currently travel?
By car at least twice a day on work days. She feels guilty about the environmental cost and hates traffic but she can't find a good alternative.

How often does she travel?
At least twice a day on work days plus she likes to drive in the country at the weekend.

What is most important to her about her mode of transport?
Convenience and comfort.



User profiles



Name:
Laurel Plum



Age:
27

Occupation:
Delivery driver

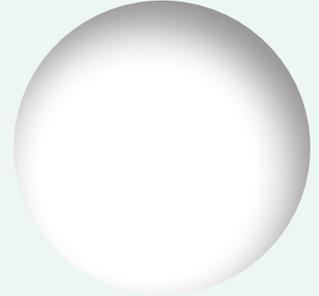
Where does she live?
In a flatshare near the town centre.

How does she currently travel?
She travels by a company-owned van. It often breaks down, which annoys her.

How often does she travel?
All day every day for work. She likes to go for cycle rides on her days off.

What is most important to her about her mode of transport?
Reliability, comfort and speed.

Name:



Age:

Occupation:

Where does this person live?

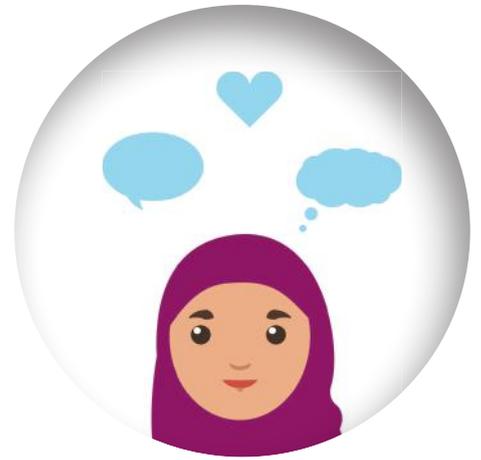
How do they currently travel?

How often do they travel?

What is most important to them about their mode of transport?



Empathy map



Pain

What would make this person unhappy?

Think and feel

What is important to this person? What do they worry about?

Gain

What would make this person happy?



Say and do

How would this person like to behave and how does that measure up against their actions on a day-to-day basis?

Hear and see

What is this person's environment like? Who influences their decisions?

Lesson 8: Designing our smart city pt. 2

Part two of the workshop sees your class use an ideas funnel to select and refine ideas from the brainstorming activity in part one. Each group will then prototype one of the ideas using the hardware and software skills they have learned with the mBot in lessons 1-6.

Resources in this book:



Lesson Overview 8



Teacher Guidance 8



Student Sheet 8a: Ideas funnel



Subject Update: How can autonomous vehicles be useful?

Subject Update: Futures Thinking

Resources available online:



Slideshow 8: Designing our smart city pt. 2



Video: Prototyping

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

LESSON 8

Designing our smart city pt. 2



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Computing

- Use a programming language to solve a variety of computational problems

Design & Technology

- Test, evaluate and refine ideas and products against a specification, taking into account the views of intended users and other interested groups
- Develop and communicate design ideas using annotated sketches, oral and digital presentations and computer-based tools
- Apply computing and use electronics to embed intelligence in products that respond to inputs, and control outputs, using programmable components

Resources



Slideshow 8:

Designing our smart city pt. 2



Student Sheet 8a:

Ideas funnel



Video:

Prototyping



Subject Updates:

- Futures thinking

Kit (per group)

- mBot with remote
- Laptop or tablet with mBlock
- Prototyping materials, such as clay, coloured paper and tape

Lesson overview

Part two of the workshop sees your class use an ideas funnel to select and refine ideas from the brainstorming activity in part one. Each group will then prototype one of the ideas using the hardware and software skills they have learned with the mBot in lessons 1-6.

Lesson steps

1. Video opener (5 mins)

Introduction to prototyping, including examples of different ways people prototype and why it is useful.

2. Classroom discussion (10 mins)

The class will discuss the video and think about how they can use prototyping to take some of their ideas forward.

3. Design thinking: prototype (35 mins)

Groups will use an ideas funnel to choose one idea from the brainstorming session and use the mBot and crafting skills to create a working demonstration that they will be able to present to the class in the next part of the workshop.

4. Reflect (10 mins)

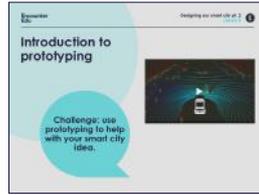
Students will reflect on the prototyping activity and share their learning.

Learning outcomes

- Understand what prototyping is and why it is used
- Describe a number of prototyping methods
- Evaluate and refine own ideas and the ideas of others
- Combine hardware, software and crafting skills to make a prototype
- Share learning with the class

Step

1
5
mins



If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

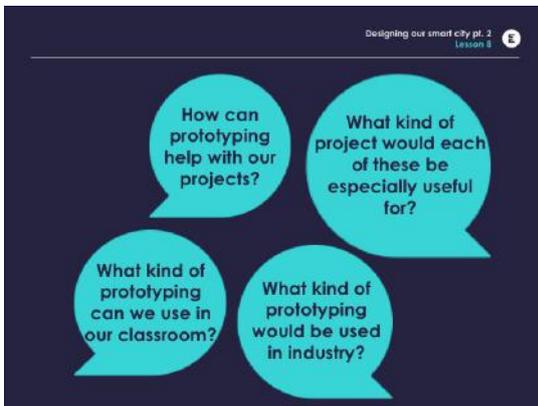


Using **slide 3** introduce students to the lesson where they are going to embark on their design challenge and use what they have learned to address societal needs.



Show your class the video **Prototyping** with examples of different types of prototype and why it is a useful process.

2
10
mins



Consider using a think-pair-share structure for these three questions as well, with students listing possible answers on their own for four minutes before sharing in pairs for two minutes. The whole class discussion element would then take a further four minutes.



Manage a whole class discussion using the questions on **slide 4**.

How can prototyping help us with our projects?

- It can give a quick, cheap way to see if something works.
- Prototypes are very useful for getting feedback from people before you spend a lot of time and money making something, especially with unusual or new products.

What kind of prototyping would be used in industry?

- Prototyping in a professional setting can use technologies we probably don't have in the classroom such 3D printing, laser cutting, electronics or special model making tools.

What kinds of prototyping can we use in our classroom?

- We can use paper prototyping, role-playing, cardboard, crafting and our mBot robot to prototype products or services in the real world.

3
35
mins



Get the students to bring out their notes from the brainstorm session. Hand out **Student Sheet 8a** Ideas funnel.

Step

Evaluate and refine using an ideas funnel

- Does it help your target users?
- Can you prototype it?
- Does it help your target users?

Make a prototype

1. Focus on how things should work. Make your ideas do what you want it to do.
2. Show how things should look. Make your ideas look the way you want it to look and use your model.
3. Make paper and get the scene to show off your prototype to other models.

STUDENT SHEET 8a

Ideas funnel

Ideas

Ideas

Ideas

Final idea chosen

Why?



Remind students that they have brainstormed many different solutions. (Slides 5-6 can optionally be used as a refresher). Then explain that they must evaluate their ideas using an ideas funnel (Student Sheet 8a) to determine the final idea they want to prototype and present. The funnel is a visual representation of “good, better, best” and should also record the chosen project and the reasons for the choice.



As a guideline, the top part of the funnel should have 8-10 ideas, the middle part 4-6 and the bottom 2 or 3.



In their groups, students should narrow down the ideas they had during the brainstorm by using the ideas funnel and the three questions on slide 7.



Once students have chosen their final idea, they should move on to starting their prototypes. Remind them that the point of prototyping is to make something quickly that tests their idea, so they shouldn't worry too much about details. Given the limited amount of time in this session, groups may wish to divide up tasks, choosing either how things should 'work' or how things should 'look' (Slide 8). If there is time, they can make props and set the scene for their demonstration.



You could also choose to get the class to vote for their single favourite idea and work on different aspects of the prototype as a single big design team with different responsibilities: code, testing, body, scenery, props, role play scripts and so on.

4
10
mins

Encounter Edu Designing our smart city pt. 2 Lesson 8

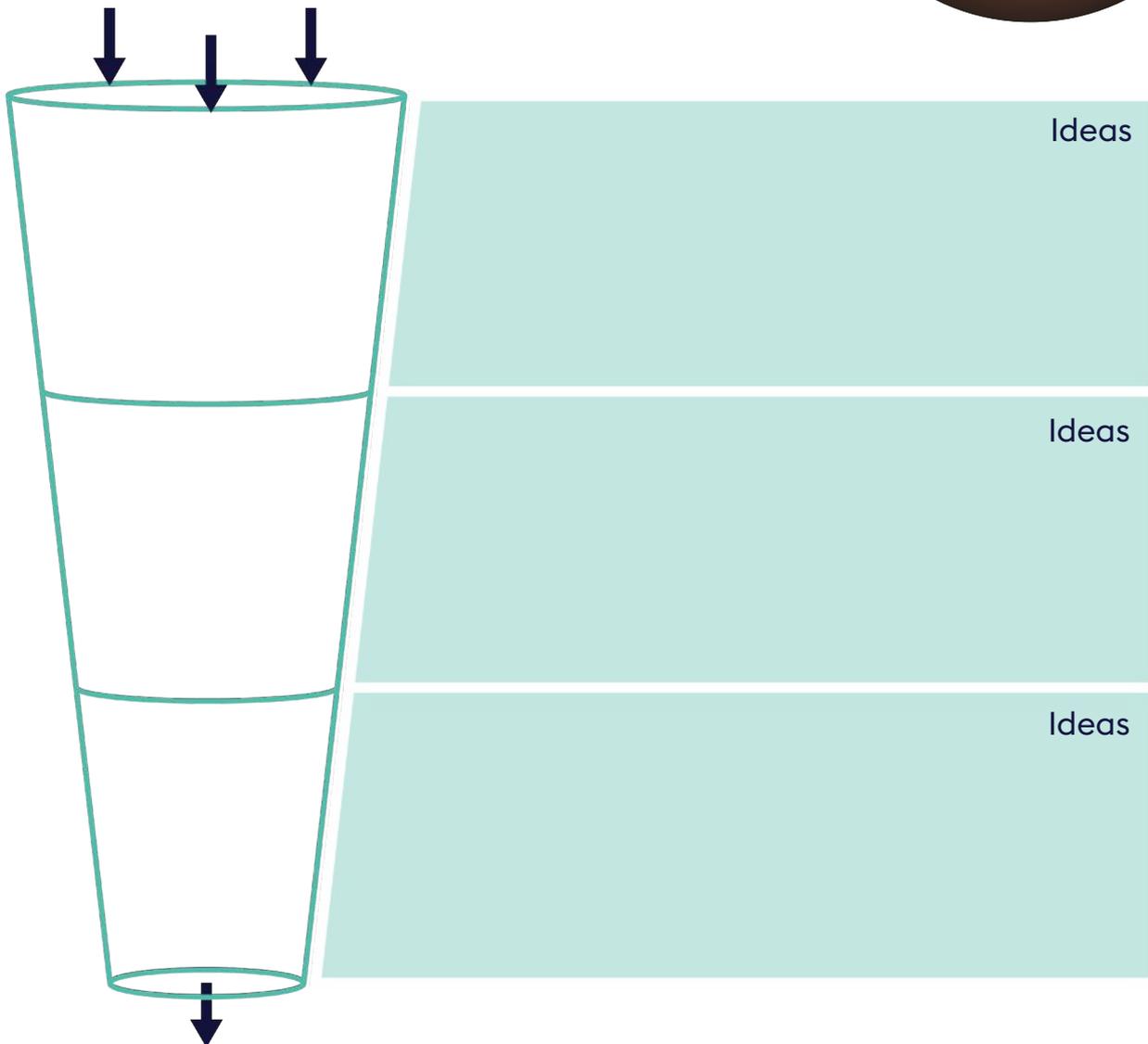
Reflect on your work

- What went well?
- What didn't go so well?
- What did you learn?



Using slide 9, ask your class to reflect on what went well, what didn't go so well and what they learned.

Ideas funnel



Final idea chosen:

Why?

Lesson 9:

Designing our smart city pt. 3

In part three of the workshop each group will discuss different ways of sharing ideas then create articles, posters, videos, photo galleries or reports to persuade their audience that their prototypes are worth taking forward. Then each group will present their prototypes and demonstrate their ideas in action using the mBot as part of a working display.

Resources in this book:



Lesson Overview 9



Teacher Guidance 9



Student Sheet 9a: Communicating your ideas

Resources available online:



Slideshow 9: Designing our smart city pt. 3



Video: Communications and marketing

All resources can be downloaded from:

encounteredu.com/teachers/units/code-smart-11-14

Designing our smart city pt. 3



Age 11-14
(Key Stage 3)



60 minutes

Curriculum links

Design & Technology

- Use research and exploration to identify and understand user needs
- Test, evaluate and refine ideas and products against a specification, taking into account the views of intended users and other interested groups
- Develop and communicate design ideas using annotated sketches, oral and digital presentations and computer-based tools
- Apply computing and use electronics to embed intelligence in products that respond to inputs, and control outputs, using programmable components

Resources



Slideshow 9:
Designing our smart city pt. 3



Student Sheet 9a:
Communicating your ideas



Video:
Communications and marketing



Subject Updates:

- Futures thinking
- About Oxbotica

Lesson overview

In part three of the workshop each group will discuss different ways of sharing ideas then create articles, posters, videos, photo galleries or reports to persuade their audience that their prototypes are worth taking forward. Then each group will present their prototypes and demonstrate their ideas in action using the mBot as part of a working display.

Lesson steps

1. Video opener (5 mins)

Communicating ideas, introducing the ideas of different audiences and different methods of communication.

2. Classroom discussion (10 mins)

The class will discuss the video and link different methods of communication with the audiences they might reach.

3. Design thinking: share (20 mins)

Each group will prepare their prototype project for a 'show and tell', including an explanation of the problem the prototype is solving, a demonstration and one example of how they would share their project beyond the class.

4. Present and reflect (25 mins)

Each group will give a short 'show and tell' of their prototype to the class, including the thought process behind it. The class will then reflect on their learning and achievements across the whole unit of work.

Learning outcomes

- Understand why sharing ideas is important
- Name at least one job associated with communication
- Identify a variety of different media and describe when each might be used
- Plan and prepare a group presentation
- Select and create an appropriate way of sharing a project
- Present and explain a group project
- Share learning

Kit (per group)

- Student mBot prototype
- Laptop or tablet with mBlock

Step

1
5
mins

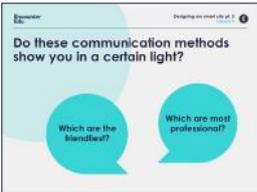


If you use formal learning outcomes with your class every lesson, the list on **slide 2** has been formulated to structure learning for this lesson. All learning outcomes are composed using the SWBAT (Students Will Be Able To) format.

Using **slide 3** introduce students to the lesson where they are going to embark on their design challenge and use what they have learned to address societal needs.

Show your class the video **Communication and marketing**, which includes different audiences and different methods of communication

2
10
mins



Guide your class in a discussion of the three questions presented on slides 4-6. Consider using a think-pair-share structure for these three questions as well, with students listing possible answers on their own for four minutes before sharing in pairs for two minutes. The whole class discussion element would then take a further four minutes.

Manage a whole class discussion using the questions on **Slide 4-6**.

What are some different ways of telling a story or sending a message? (**Slide 4**)

- If needed, prompt with social media platforms, email, TV, radio, text, in person and adverts.
- Ensure the students understand the difference between digital and traditional media.

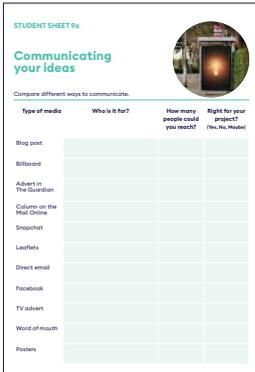
Are these communication methods best suited to certain people? (**Slide 5**)

- Consider different characteristics of people and the message, such as demographics and location.
- Encourage the class to come up with examples of typical people who might see a message in different places, such as mums visiting mumsnet, younger people using Snapchat or local people seeing a poster in a shop.

Do these communication methods show you in a certain light? What is appropriate for their project? (**Slide 6**)

- Ask your class to think about tone and how choosing to use different types of media will portray them.
- Highlight the difference between communicating with users of a service and enablers of a service, such as someone in local government who can approve it or a funder.

Step



 Hand out **Student Sheet 9a** Communicating your ideas.

 Explain that different forms of media serve different purposes. Your class should consider audience (who it reaches), reach (how many people it reaches), tone of different methods of communication, then think about the best ways of communicating with their target audience.

 Student groups will decide on the best methods for reaching their target audience.

3
20 mins



 Tell each group that they are going to use what they now know about communications to prepare their prototype project for a final 'show and tell'. They should include an explanation of the problem the prototype is solving, a description of who they are helping, a simple demonstration of their prototype and at least one example of how they would communicate their project to a target group.

 Students should use the information on **slide 7** to guide the preparation of their show and tell presentation. Each group should give a short 'show and tell' of their prototype to the class, including the thought process behind it. If your class worked together as one team, ask them to present it to you as a character: perhaps a local resident or the mayor.

4
25 min



 Finally, ask your class to reflect on their learning and achievements across the whole unit of work. You can use **slide 8** to guide the discussion.

Communicating your ideas



Compare different ways to communicate.

Type of media	Who is it for?	How many people could you reach?	Right for your project? (Yes, No, Maybe)
Blog post			
Billboard			
Advert in The Guardian			
Column on the Mail Online			
Snapchat			
Leaflets			
Direct email			
Facebook			
TV advert			
Word of mouth			
Posters			

Subject Updates

The subject updates are designed to give you, the teacher, the confidence you need to deliver some of the newer content.

Resources in this book:



Subject Update: What is a smart city?

Subject Update: How can autonomous vehicles be useful?

Subject Update: What questions does an autonomous vehicle need to answer?

Subject Update: When can a car be considered autonomous?

Subject Update: mBot's default program

Subject Update: How to set up mBlock

Subject Update: mBlock app

Subject Update: Tips for teaching coding

Subject Update: Coding tips: writing elegant code

Subject Update: Coding tips: failing and debugging

Subject Update: Coding tips: commenting

Subject Update: Troubleshooting guide

Subject Update: mBlock glossary

Subject Update: Futures Thinking

All resources can be downloaded from:
encounteredu.com/teachers/units/code-smart-11-14

What is a smart city?

Did you know?

Today, over half of the world's population lives in urban areas, and this number will increase to about two thirds of the world's population by 2050.

The Code Smart education programme challenges students to design solutions for the smart city. The smart city is a concept rooted in innovation, information and infrastructure. It points to issues of sustainability and how an increasingly urban population can enjoy a decent future.

The smart city

Today, over half of the world's population lives in urban areas, and this number will increase to about two thirds of the world's population by 2050. That means that today, 3.9 billion people are living in cities, and this number will continue to rise exponentially.

A successful and sustainable future will need successful and sustainable cities. This is the heart of **UN Sustainable Development Goal 11**: to make cities and human settlements inclusive, safe, resilient and sustainable. Smart cities are a way of making life better for the billions of city dwellers around the world.

Learn more about SDG 11



Link to resource:
<http://sustainabledevelopment.un.org/sdg11>

The ideal city would be cleaner, quieter, safer, more accessible and healthier. Cleaner cities would have less trash and less pollution than current systems. Quieter cities would have less noise from cars and a more organised sense of the urban chaos that city dwellers love. Safer cities would be well lit, well patrolled, and have a strong sense of community. Accessible cities would make public transportation the cheapest and easiest option for travel for all people, negating the need for cars and the congestion and pollution they bring. Bike access and priority is also a cornerstone of the ideal sustainable city. Finally, the ideal city makes health of individuals and the city a priority. This means people have easy access to fresh food, recreation, and healthcare and there are top-notch sanitation systems, like great sewer and wastes services.

Smart cities incorporate digital technologies such as sensors, transmitters, internet of things, and artificial intelligence to collect data to produce improvements in the way a city functions for its inhabitants and businesses. Smart city initiatives can be applied to any area of urban services and management and already people have started focusing on transport, buildings, administration, waste, and energy.

Transport

A classic example is the smart parking meter that uses an app to help drivers find available parking spaces without prolonged circling of crowded city blocks. The smart meter also enables digital payment, so there's no risk of coming up short of coins for the meter.

Did you know?

Energy conservation and efficiency are major focuses of smart cities. Using smart sensors, smart street lights dim when there aren't cars or pedestrians.

Transport (continued)

Also in the transportation arena, smart traffic management is used to monitor and analyse traffic flows to prevent roads from becoming too congested. Such smart traffic management systems would be easier to integrate with autonomous vehicles. Smart public transit is able to coordinate services and fulfil traveller needs in real time, improving efficiency and satisfaction. Ride-sharing and bike-sharing are also common services in a smart city.

Energy

Energy conservation and efficiency are major focuses of smart cities. Using smart sensors, smart street lights dim when there aren't cars or pedestrians. Smart grid technology can be used to improve operations, maintenance and planning more generally to supply power on demand and monitor energy outages. Smart city energy initiatives also aim to monitor and address environmental concerns such as air pollution and climate change that result from energy consumption.

Waste

Sanitation can also be improved with smart technology. One example is using internet-connected bins and a connected fleet of rubbish lorries to prioritise areas of need rather than driving on a pre-defined schedule. Sensor technology could also be used to test the quality of drinking water or spot potential sewage blockages before they become an issue.

Safety

Smart city technology is increasingly being used to improve public safety, from monitoring areas of high crime to improving emergency preparedness with sensors. For example, smart sensors can be critical components of an early warning system before droughts, floods, landslides or hurricanes.

Buildings

Smart buildings are also often part of a smart city project. Old buildings can be retrofitted, and new buildings constructed with sensors to provide real-time building management and ensure public safety. Attaching sensors to buildings and other structures can detect wear and tear and notify officials when repairs are needed. Citizens can also help in this matter, notifying officials through a smart city app when repairs are needed in buildings and public infrastructure, such as potholes in roads. Sensors can also be used to detect leaks in water mains and other pipe systems, helping reduce costs and improve efficiency of public workers.

How can autonomous vehicles be useful?

Did you know?

Researchers estimated that there are at least 700 million parking spaces in the US, which is more than 6,000 square miles

Re-imagine parking



Excess parking spaces can be transformed.

Though self-driving cars used to be science fiction fantasies, they are now likely to be part of our not-too-distant future and have the potential to re-shape our society similar to the impact of agriculture, the industrial revolution, or the internet. Autonomous vehicles could change how we approach ownership and reshape our cities, improve welfare and accessibility and increase safety.

Shift from individual car ownership to shared ownership

Despite existing car sharing solutions, such as taxis, carpooling, and car rentals, many adult individuals currently own a car. In 2015, there were an estimated 263 million cars in the United States. That's almost 1 car for every adult. Additionally, in 2017 Americans spent nearly \$2 trillion on car related costs (including fuel, maintenance, and insurance), more than they spent on food that year.

Unfortunately, the average vehicle is only used 4% of the time and remains parked for the other 96%. Autonomous cars, enable vehicles to be used more efficiently, by allowing them to be continuously used over the course of a day to transport multiple groups of people. For example, in the morning an autonomous vehicle could be used to take students to school and adults to work. Afterwards, that autonomous vehicle could be used to drive tourists to landmarks, take the elderly to the doctors, and help people run errands. Then in the evening the autonomous car can help students and adults return home. Because these vehicles will be in constant use, there will be less use for parking spaces. In the US there is an estimated 700 million parking spaces, which take up more than 6,000 square miles, that could be transformed into parks, shops, and housing.

More efficient use of vehicles will also decrease the total number of cars needed. Fewer vehicles also mean less traffic, especially since autonomous vehicles can communicate with each other. This enables them to driver faster and closer together, which also allows people to get to their destinations faster.

Since these vehicles are shared, it is likely that most autonomous vehicles, will not be owned by people, but instead owned by business who rent to individuals on a pay-as-you-go basis. Experts estimate that this will still be cheaper for most individuals since they no longer have to buy, fuel or repair their own cars.

Improved welfare and accessibility

A major benefit of autonomous cars is that commuting hours can be used for other tasks such as leisure or work. Instead of the stress associated with being stuck in traffic, drivers could read, talk with friends / family, surf the internet or even sleep.

Autonomous vehicles will also increase accessibility of travel for people who otherwise could not drive, such as people with epilepsy, people with poor vision, or others with impairments.

Did you know?

According to the World Health Organization, there were approximately 1.25 million road traffic deaths in 2013.

Increased safety

Currently, a leading cause of preventable deaths is road accidents. According to the World Health Organization, approximately 1.25 million pedestrian, cyclists, motorcyclists, and car occupants were killed by traffic deaths in 2013. Driverless vehicles can minimize deaths and injuries due to human errors such as distracted driving and aggressive driving, saving millions of lives and billions of dollars in damages and healthcare costs annually.

So is it all good?

It is likely that many people will lose their jobs as autonomous vehicles become more mainstream. This includes drivers as well as people involved in the production of cars. In the US, this is an estimated 4 million driver jobs and another 3 million jobs in the production and sales of cars.

In the past, whenever technology has displaced jobs, new jobs and industries have also been created, such as designing and maintaining the autonomous vehicles, both hardware and software, but the transition for workers who lose their jobs can be difficult.

Timing

Multiple companies, including major technology companies and major car manufacturers, are working on autonomous driving technology, but the road to full adoption of autonomous vehicles still has some barriers.

Currently, the cost of equipment to make autonomous cars truly functional is still in the hundreds of thousands of dollars and the technology is not fool proof, especially in difficult driving conditions like at night or in rain. There are also still barriers to broad adoption including individual preferences, ethical debates and political regulation.

However, many experts expect autonomous vehicles to start appearing on the roads in the 2020s.

What questions does an autonomous vehicle need to answer?

Did you know?

Autonomous vehicles will use a combination of different sensors, such as cameras, lasers and sonar, to precisely pinpoint where they are on a map

Mobile Autonomy



Autonomous vehicles need to be aware of surroundings including pedestrians.

In order to have an autonomous vehicle, there must be an intelligent software system that can answer 3 important questions: 'Where am I?', 'What's around me?' and 'What do I do next?' Currently, most vehicles rely on humans to provide answers to these questions, but soon sensors will help provide intelligent software systems with data about the environment around the vehicle so that the intelligent software systems can make decisions about the best actions to take.

Where am I?

The world is always changing and we're constantly moving. Humans sense cues from their environment to help determine where they are. You might use information like seeing a signpost using your eyes or hearing sirens with your ears to help conclude that you are near the hospital. Autonomous vehicles also need a way of knowing where they are. You may think of satellite navigation systems which many vehicles and mobile phones have now, but these systems are often unreliable depending on location and weather conditions. The intelligent systems of autonomous vehicles will use a combination of different sensors, such as cameras, lasers and sonar, to precisely pinpoint where they are on a map, no matter the weather or lighting conditions.

What's around me?

Before deciding what to do, an autonomous car will have to determine what's around it. Just like you wouldn't want to walk straight into a wall or trip over a rock, autonomous vehicles will need a way to identify and track obstacles in their environment. This will mean a system of sensors and algorithms must be in place to recognise pedestrians, other vehicles, buildings, and more. This is especially important for keeping everyone safe, inside and outside of the vehicles.

What do I do next?

With all that information coming in, it can be challenging to decide what to do next. Humans make decisions on how to reach their goals based on different priorities. Autonomous vehicle will need to determine what to do next in order to transport you to your destination based on various algorithms. The intelligent software systems within autonomous vehicles will use the information about where they are and what's around them and calculate a safe and efficient route to transport people to their destinations.

When can a car be considered autonomous?

Did you know?

Many cars already have some driver assistance, such as cruise control or parking assistance.

Hands-off driving



Hands off driving is partial automation.

The terms autonomous car, driverless car, self-driving car and robot car have entered our language and are often used interchangeably, but when can a car actually be considered autonomous? Broadly, an autonomous car is capable of sensing its environment and navigating without human input.

As the scope and interest in autonomous cars has grown, a group of engineers and technical experts has produced a guide to the levels of automation in vehicles of the future: SAE International's J3016 standard, also known as the Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. The guide describes six levels of autonomy, from no autonomy at Level 0, to full autonomy at Level 5.

Level 0: No Automation

Exactly as described. The car is not automated in any way and relies on a human for all of the tasks. Even if warning systems flash, a human is still in full control.

Level 1: Driver Assistance

The human driver is still expected to do the majority of the work, but is assisted by automated systems. For example, cruise control sets the speed of the vehicle, while the human driver steers. Parking assistance can automate steering, with the human driver controlling the speed. The driver must be ready to retake full control at any time.

Level 2: Partial Automation

Level 2 is often described as "hands-off" driving. The automated system takes full control of the vehicle (accelerating, braking, steering) using information about the driving environment. The driver must monitor the driving and be ready to take control. So, even if this level is described as "hands-off", the driver's hands should be pretty close to the steering wheel to intervene at any time.

Level 3: Conditional Automation

Also known as “eyes-off”, at level 3, the driver does not need to monitor the driving environment, but can turn their attention to other activities like sending an email or watching a film. The vehicle will handle safety-critical functions like emergency braking, but can still call on the driver to take control.

Vehicles in level 3 and above are considered “automated driving systems”. The substantial difference here is that the vehicles are able to monitor the driving environment around them. Crucially, these types of vehicles make decisions themselves. For instance, a level 3 car will be capable of seeing a slower moving vehicle in front of it before making the decision to overtake. The human is on hand, mostly, to intervene if things go wrong.

Level 4: High automation

At Level 4, the vehicle is fully autonomous within certain driving scenarios. This means that the driver could go to sleep or leave the driver’s seat and the vehicle is able to handle all aspects of the driving task. However, this functionality is limited to what is known as the ‘operational drive domain’ of the vehicle. For example, a vehicle might only be fully autonomous in traffic jams or within a limited geographic area.

Level 5: Full Automation

Full automation means full-time performance by an automated driving system for all aspects of driving and under all the conditions that you would expect of a human driver. No human intervention is ever required and so there would be no pedals, steering wheel or controls for a driver to take control of. A good example would be a fully robotic taxi.

Did you know?

Full automation means full-time performance by an automated driving system for all aspects of driving and under all the conditions

mBot's default program

Did you know?

You can always return the mBot to the default program, even after you have uploaded other programs.

The mBot comes with a default program uploaded so you can start playing with your mBot as soon as it is built. This default program has three modes, which allow you to explore different functionalities.

The three default modes

Mode A – Remote control driving You can use the arrow buttons on the remote to drive the robot forward and backward as well as turn left or right. The number buttons correspond to different sounds the mBot can make using its buzzer. In mode A, the mBot's LEDs are white.

Mode B – Obstacle avoidance The mBot drives forward on its own until it detects an obstacle, such as a wall. If it detects an obstacle, it turns then continues driving forward. In mode B, the mBot's LEDs are green.

Mode C – Line following The mBot will drive forward while following a black line. In mode C, the mBot's LEDs are blue.

It is easy to get started



Students can drive the mBot as soon as it is built.

Switching between the modes

The mBot starts off in mode A, and there are two ways to switch between the modes:

1. Select the mode by pressing the A, B, or C button on the remote.
2. Press the onboard button on the mBot (located at the front end of the mCore).

Returning mBot to the default program

You can always return the mBot to the default program, even after you have uploaded other programs. This may be helpful if you want to use the mBot with a new group of students or if you want to test that the mBot is working.

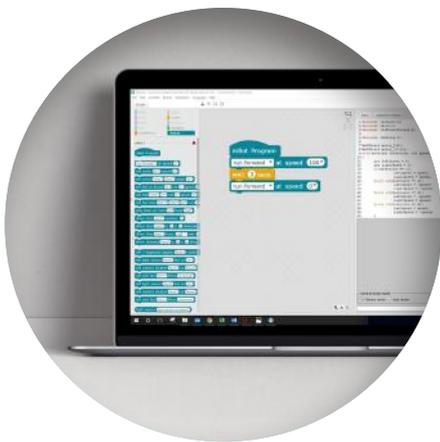
1. Connect the mBot to the computer using the USB cable and turn on power to the mBot.
2. Open mBlock and connect to the mBot. Click the Connect menu -> Serial Port and select the last port.
3. Next, click the Connect menu -> Reset Default Program. The default program will upload to the mBot.
4. When the upload is finished, you can disconnect the mBot and use the modes of the default program.

How to set up mBlock

Did you know?

The mBlock language is easy to learn and allows you to code your mBot and other Arduino based hardware quickly.

mBlock



mBlock can be downloaded for Windows, Mac OS, Chrome OS and Linux.

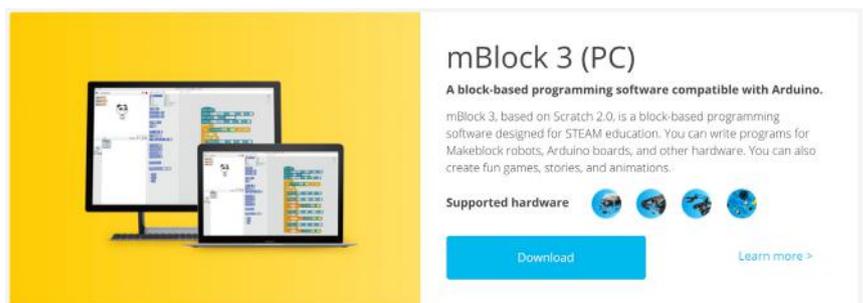
In order to program your mBot, you will need mBlock on your computer. mBlock is a graphical programming environment developed by Makeblock based on Scratch 2.0 Open Source Code. The mBlock language is easy to learn and allows you to code your mBot and other Arduino based hardware quickly.

This Subject Update provides you with the instructions to get mBlock set up on your computer before your first coding session in Code Smart Lesson 2.

Download and install mBlock 3

Note: These instructions will help you to get started with mBlock on your computer or laptop device. There is a different set of instructions for tablets and mobile devices. You may need to contact your school's network manager to help you if your school system requires permission to install outside software. They may also be helpful for getting the software on all of the computers your students will be using.

1. Download **mBlock 3** for your operating system from <http://www.mblock.cc/software/mblock/> Follow the instructions to install the downloaded software onto your computer



2. You may also need to download the **Arduino Software (IDE)** for your operating system from <https://www.arduino.cc/en/Guide/HomePage>, especially if you are using macOS.

SUBJECT UPDATE

Did you know?

mBlock is based on Scratch. Both are graphical coding languages which make it easier to build programs.

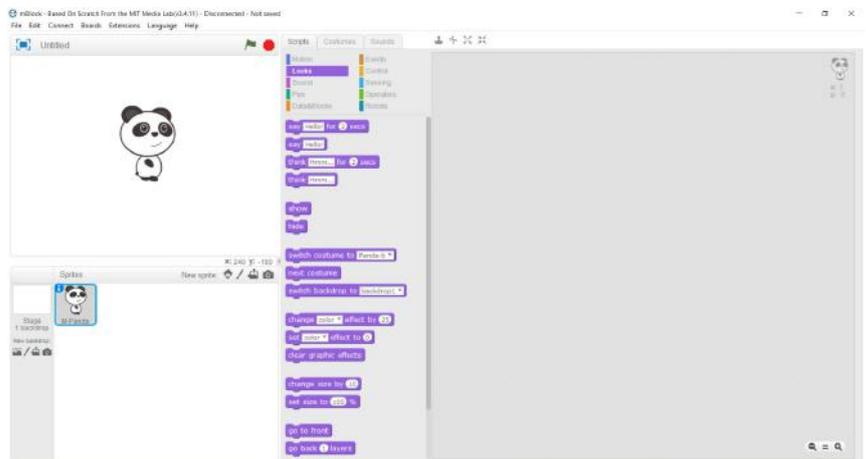
More mBlock support



mBlock FAQ, go to <http://mblock.cc/faq/>

Download and install mBlock 3 (continued)

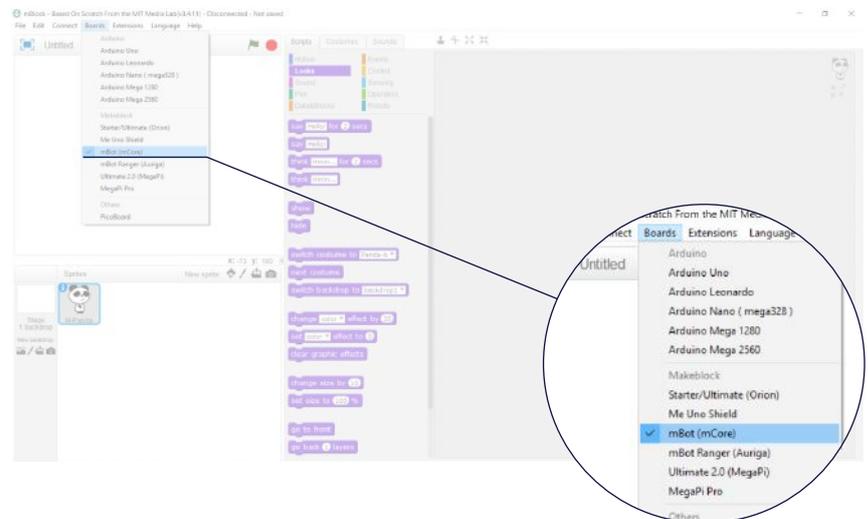
3. When you open mBlock, you should see a screen similar to this:



Check mBlock settings

4. There are a couple of settings that you should check for in order to be ready to code your mBot.

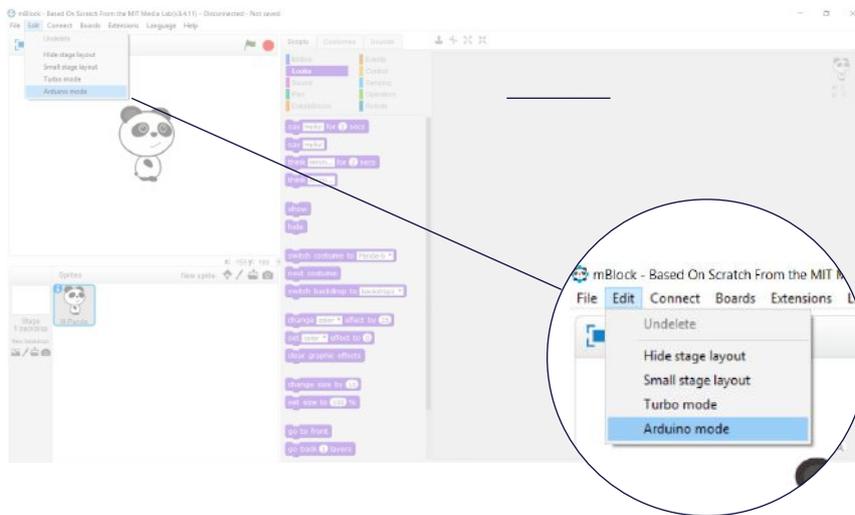
Go to the **Boards menu** and check that **mBot (mCore)** is selected. It may have a check mark already next to it. If not, click mBot (mCore) to select.



SUBJECT UPDATE

Check mBlock settings (continued)

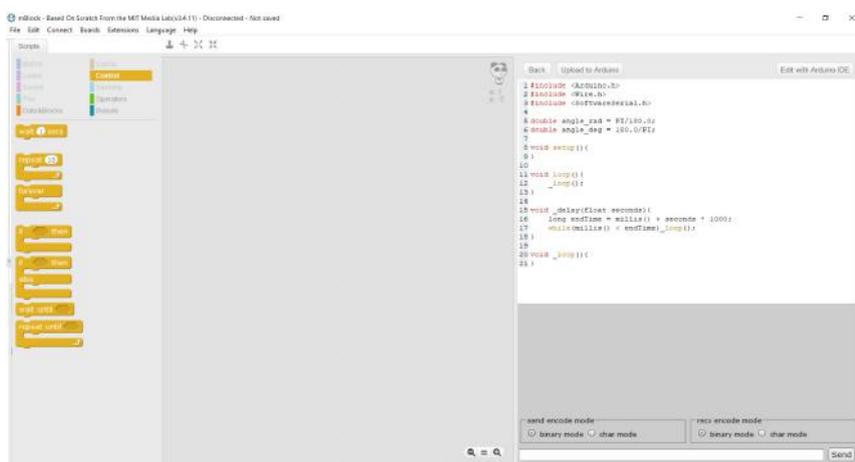
Then go to the **Edit** menu and select **Arduino mode**.



Did you know?

You can use mBlock to control other Arduino based hardware.

This will make your screen look like this:

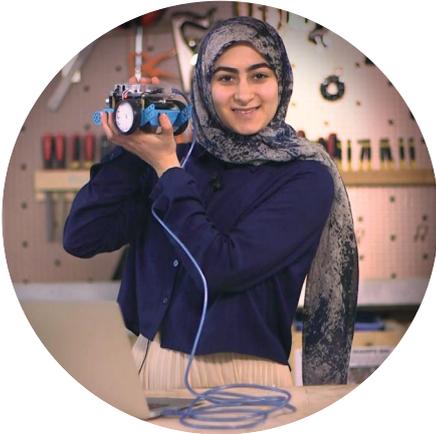


SUBJECT UPDATE

Did you know?

mBlock scripts are in bit-sized blocks that are easy to build together.

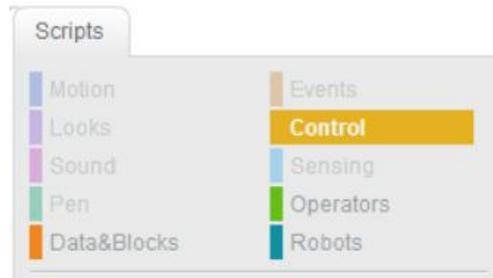
Uploading code



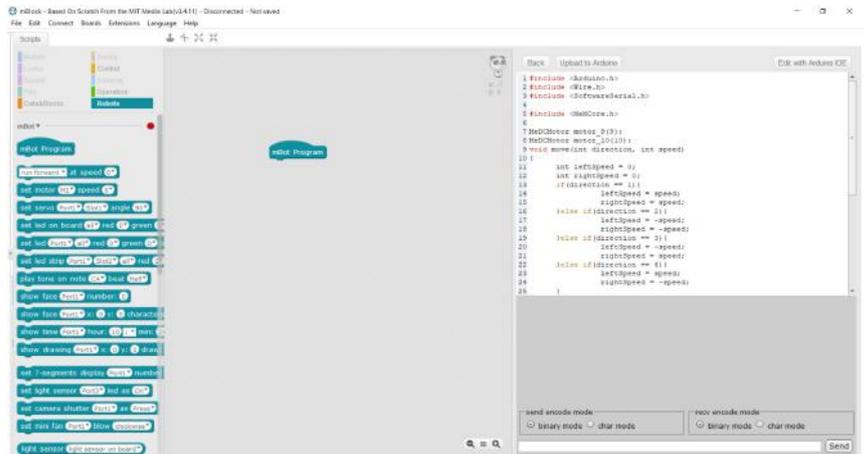
Make sure your mBot is turned on when you upload your code.

Quick start guide to coding

5. On the left side of your screen you should see all of the different 'Scripts' that you can use. You'll be mostly using the blue **Robots** blocks, gold **Control** blocks, and green **Operators** blocks.

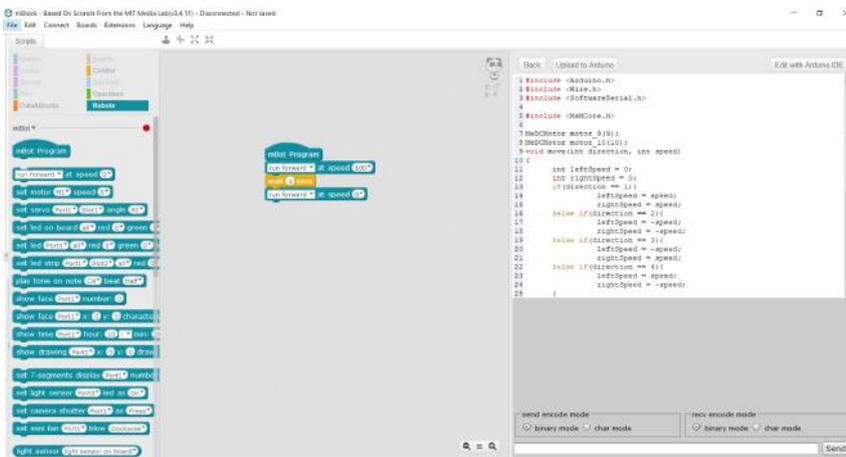


6. When you are ready to write your code, all you have to do is click a block, drag and drop it to the script area in the middle of your screen. Note that the Arduino code in the right panel automatically updates, which can be helpful for demonstrating what typical typed code might look like.



Quick start guide to coding (continued)

- As you add more blocks to your code, make sure they snap into place. You can also rearrange blocks as much as you like. To delete a block, just drag it back into the script blocks section. **Note:** many blocks also have dropdown menus within them to allow you to adjust specific commands. Please refer to the **mBlock glossary** Subject Update for brief descriptions of the blocks used in Code Smart.



Did you know?

We recommend putting the mBot upside down while programs upload because we wouldn't want the mBot to run off the desk or table!

Quick start guide to uploading code to the mBot

- Connect the mBot to your computer with the USB cable provided.



- Then switch ON the power to the mBot and set it upside down. We recommend putting the mBot upside down because it immediately starts running programs once they are uploaded and we wouldn't want the mBot to run off the desk or table!

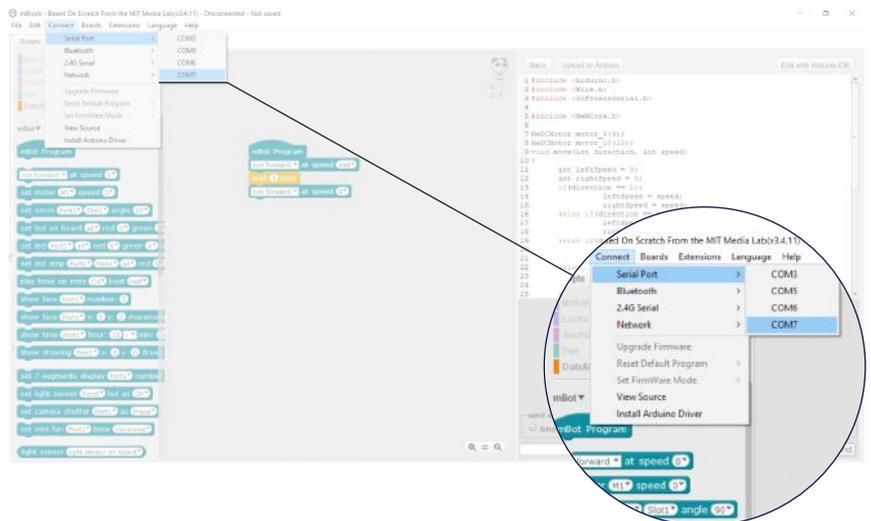


Did you know?

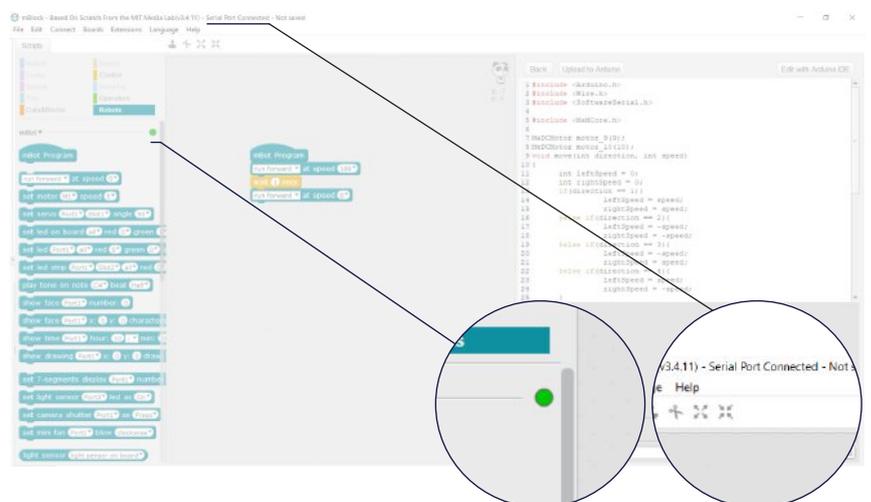
While the code uploads, check out the typed code in the Arduino column (right side of your screen). This is what more typical typed code looks like.

Quick start guide to uploading code to the mBot (continued)

10. Go to the **Connect** menu and enter the **Serial Port** submenu. Select the *last option*. The last option should be something like 'COM#', where # is the largest number in the list.

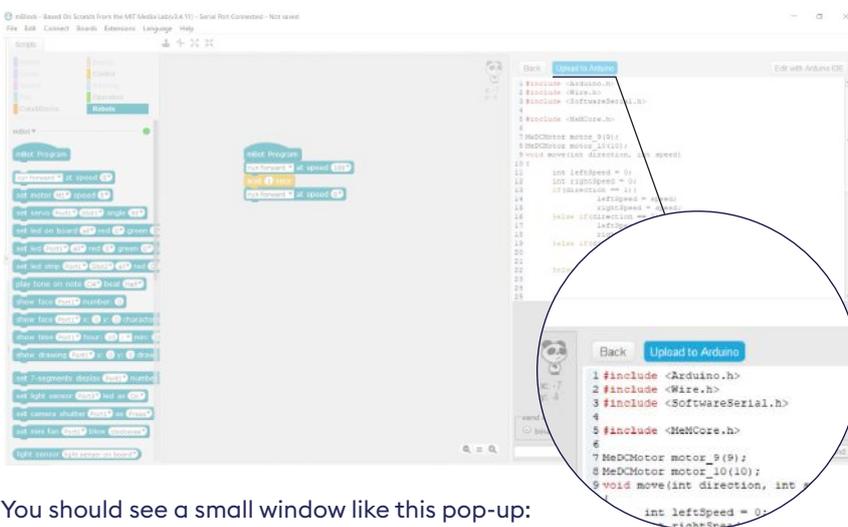


11. You should notice that your mBlock environment should update to say 'Serial Port Connected'. Additionally, you should notice a green status dot in the Scripts column next to mBot if you have the Robots block type selected.



Quick start guide to uploading code to the mBot (continued)

12. To upload your code to your mBot, click **Upload to Arduino**.



13. You should see a small window like this pop-up:



Don't close this window as it will give you a status update when uploading is finished. (Note if the green dot next to mBot in the Scripts column turns red that is okay.)

14. When the upload is finished you should see the small window update to look like this:



15. Success! Your mBot should have your program uploaded. You may even notice your mBot moving as soon as the upload is done. To let the mBot move according to your program, disconnect the mBot, set it on the ground, and toggle the power switch OFF and then ON again. Your mBot should do what your program says.

Did you know?

To let the mBot move according to your program, disconnect the mBot, set it on the ground and toggle the power switch

mBlock app

Did you know?

MakeBlock has released an app for iOS and Android so that you can use tablets and mobile devices to control the mBot.

mBlock Blockly app

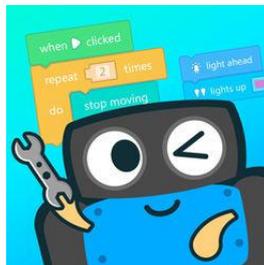


mBlock can be downloaded for iOS and Android.

In addition to the desktop mBlock software for personal computers and laptops, MakeBlock has released an app for iOS and Android so that you can use tablets and mobile devices to control the mBot. The mBlock app is a game-based programming app which challenges students to write progressively more difficult code. There is also the option to create code in a block-based programming environment right in the app and then to upload the program to the mBot via Bluetooth.

Downloading the mBlock Blockly app

You can download the mBlock Blockly app for iOS and Android from the Makeblock website: <https://www.makeblock.com/software/mblock-app/downloads> or from the App Store or Google Play store.



Upgrading Firmware

As additional features are added to Makeblock software and hardware, you may need to upgrade firmware.

1. Connect the mBot to the computer using the USB cable and turn on power to the mBot.
2. Open mBlock and connect to the mBot. Click the Connect menu -> Serial Port and select the last port.
3. Next, click the Connect menu -> Upgrade Firmware. The firmware upgrade will start automatically.
4. When the upgrade is finished, you can disconnect the mBot.

Connecting to the mBot via Bluetooth

The mBlock app communicates with the mBot using Bluetooth. To connect your device to your mBot:

1. Make sure Bluetooth is enabled on your device and that the mBot is powered on.

SUBJECT UPDATE

Connecting to the mBot via Bluetooth (continued)

- The app may prompt you to connect or you may notice an 'unlinked' icon in the top right corner of the app (if so, click on the icon).



- The app will prompt you to bring the device close to the mBot. The device and the mBot should link automatically.

If the device and the mBot do not link automatically, check that the mBot is turned on and that Bluetooth has been enabled on the device. If this does not work, then try upgrading the firmware.

Using the app

There are two primary features of the app that may be of interest:

- game-based coding challenges
- programming environment

1) Game-based coding challenges

If you click on the mBot option in the app, you are given a series of guided coding challenges.



Did you know?

You can use the mBlock Blockly app even without internet connection.

Connecting mBot



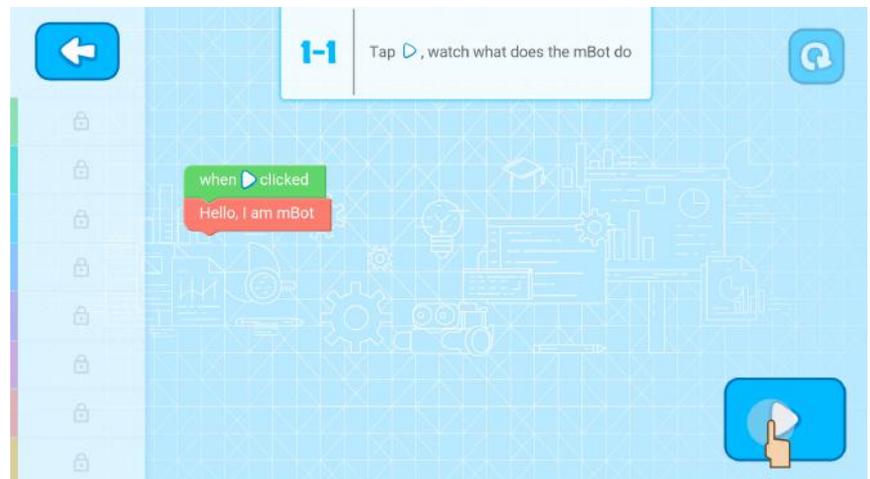
If the device and the mBot do not link automatically, check that the mBot is turned on and that Bluetooth has been enabled on the device.

Did you know?

You can code on your mobile device in the mBlock Blockly programming environment.

Using the app (continued)

There are step-by-step instructions that will guide you through progressively more difficult challenges and will show you more features of what the mBot can do.

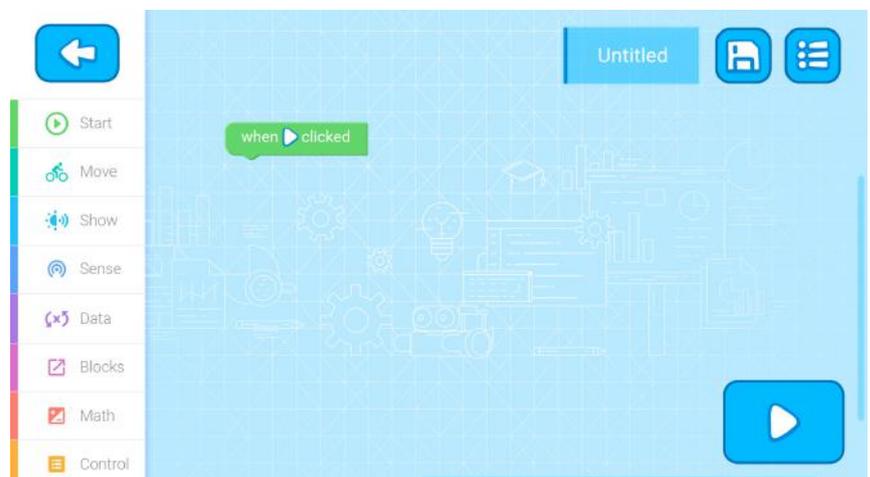


2) Programming environment

To access the in-app programming environment, click the 'create' icon.



The programming environment in the app should look familiar if you've used mBlock or Scratch. However, be aware that it is a little different. Because it is block-based coding, you'll likely be able to jump in by reading carefully and learning through trial and error. The game-based coding challenges are a good way to learn more about this in-app programming environment.



Tips for teaching coding

Did you know?

The most important skill you will need as a teacher is the ability to help your class solve their own problems.

You don't have to be an expert programmer to teach robotics and coding. The most important skill you will need as a teacher is the ability to help your class solve their own problems.

However, it is helpful to have some tips for good practice that will help guide you and your students. In this Subject Update, you'll find tips about writing, editing and sharing code. You may also find the **Troubleshooting guide Subject Update** useful. It has solutions to the most common errors you'll come across in your robotics education journey. If you don't find the solution to your problem in these Subject Updates, there is also a helpful forum of mBot users online <http://forum.makeblock.com/>.

Makeblock forum



Forums are great for asking questions and helping others. Check out <http://forum.makeblock.com/>.

Multiple Solutions

In programming there are often multiple ways to solve a problem. For example, if you want to make your mBot keep driving forward if there is not an obstacle right in front of it, but stop when there is one, you might write a piece of code like this:

```
mBot Program
forever
  if ultrasonic sensor Port3 distance < 10 then
    run forward at speed 0
  else
    run forward at speed 100
```

If there is an obstacle less than 10cm away then stop. Else keep driving forward.

However, this piece of code works too:

```
mBot Program
forever
  if ultrasonic sensor Port3 distance > 10 then
    run forward at speed 100
  else
    run forward at speed 0
```

If there is an obstacle more than 10cm away then keep driving forward. Else stop right away.

Did you know?

Elegant code should be clean, concise and easy to understand.

Multiple Solutions (continued)

The two programs both solve the same problem, but approach it slightly differently. In that case, both solutions are fairly similar in logic and length. However, sometimes there is a 'better' solution. For example, when making the car drive in a square, you might repeat steps to drive and turn multiple times like this:

```
mBot Program
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 0
```

Or like this:

```
mBot Program
repeat 4
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 0
```

Both solutions work, but the second solution would be considered 'more elegant'

Writing 'elegant code'

Some coders like to talk about writing 'elegant code'. But what does that mean? When you think of elegance in the literary world, you may think of long, beautiful sentences and descriptive paragraphs. In coding, it's different. It's more important to be clear and concise, like a straightforward recipe. Elegant code should be **clean**, **concise** and **easy to understand**.

Find out more in the **Coding tips: writing elegant code Subject Update**.

SUBJECT UPDATE

Failing & debugging

It is rare to write a working program the first time you try. Do not worry about failing. To fail is to make your **F**irst **A**ttempt **I**n **L**earning. Mistakes in your code show that you are trying.

Debugging is a systematic way for your students to iron out their mistakes. Instead of giving them the answers, you should encourage your students to work through potential problems step-by-step, starting from basic issues and working up to more complex issues.

Find out more in the **Coding tips: failing and debugging Subject Update**.

Comments

Programs can include comments which are ignored by the computer but are meant for humans to read to help them understand the program. When you write code it is helpful to add comments to it to let other people know what it does or remind you what it does when you've not looked at in a while.

Find out more in the **Coding tips: commenting Subject Update**.

Sharing code

Code should be shared so we can learn from others and so we can work together to write better code. Code is written to solve a problem, and this is very rarely only a problem for one person. By sharing your code, you are helping others solve their problems, and, by letting them see the actual code, they can understand how it works and apply it to solve other similar problems.

By working together in this way, we can solve bigger programs rather than having everyone individually solve the same smaller problems repeatedly and making no progress. Computer software where the code is available for anyone to use and see is called open source. The largest open source community in the world can be found on github.com.

Did you know?

Code is written to solve a problem, and this is very rarely a problem for only one person.

Coding tips: writing elegant code

Did you know?

When you write code for your computer, it's like writing a recipe and you need to be very clear.

Clear instructions are important



If your code isn't elegant, the mBot may still work but may not be as efficient.

Some coders like to talk about writing 'elegant code'. But what does that mean? When you think of elegance in the literary world, you may think of long, beautiful sentences and descriptive paragraphs. In coding, it's different. It's more important to be clear and concise, like a straightforward recipe. Elegant code should be **clean, concise and easy to understand**.

What is elegant code?

Remember that code is a set of instructions for your computer. When you write code for your computer, it's like writing a recipe and you need to be very clear. If you had to follow the instructions on a messy, long and confusing recipe for making a cake, you might miss an important step, need extra time, or even give up on completing the cake!

Elegant code should be clean, concise and easy to understand. It's not about showing how clever you are or how complicated you can make your code, it's about finding the simplest way possible of making your idea work. As the famous writer Antoine de Saint-Exupery said, "perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."

Look at the two examples of code to make a car drive in a square on the next page. Both programs make the car do the exact same thing, but which solution would be easier to type out and faster to explain?

SUBJECT UPDATE

First way

```
mBot Program
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 100
wait 3 secs
turn right at speed 100
wait 1.75 secs
run forward at speed 0
```

Second way

```
mBot Program
repeat 4
  run forward at speed 100
  wait 3 secs
  turn right at speed 100
  wait 1.75 secs
run forward at speed 0
```

Did you know?

You might not think efficiency is a big deal, but when considering programs for driverless cars, every second can count.

The second piece of code is easier to type and would be quicker to explain. Also, when you need to drive in a pentagon or a hexagon in the future, it will be much easier to modify the code in the second example.

There are many cases, where multiple solutions might achieve the same goal, but one program may be a lot more efficient. For example, if a driverless car is using a camera to read the traffic lights one program might constantly scan the whole screen for traffic lights. However, another program might only scan at street junctions and only scan the sides of the screen because that is where we know traffic lights will be. Both programs would have the same result, but the second one is more efficient and can save time, memory, and energy. You might not think efficiency is a big deal, but when considering programs for driverless cars, every second can count. Having an efficient program could be the difference between stopping in time and crashing.

Coding tips: failing and debugging

Remember!

If we make no mistakes, we may not learn about all the different ways to tackle a problem and may actually miss a better solution.

Robot fail



Professionally-designed robots fail too.

Do not be scared to test code at any time. We cannot be sure our code will work until we test it. Often it is easy to miss something that might happen when it runs from just reading the code. If something unexpected happens, it's an opportunity to think about what went wrong and how to fix it.

Failing

It is rare to write a working program the first time you try. Do not worry about failing. To fail is to make your **F**irst **A**ttempt **I**n **L**earning. Mistakes in your code show that you are trying.

When we fail, we have an opportunity to learn from fixing the problem and this helps us to understand how things went wrong in the first place. If we make no mistakes, we may not learn about all the different ways to tackle a problem and may actually miss a better solution.

If you or your students are feeling a little deflated about code not working, remember to have a look at the Robot fail video.

Debugging

Debugging is a systematic way for your students to iron out their mistakes. It is highly likely that your students will get something wrong and their code may not work the first time. At this stage, it might be tempting to leap in and try to solve the problem. Instead of giving them the answers, you should encourage your students to work through potential problems step-by-step, starting from basic issues and working up to more complex issues.

Here's a top tip: before checking code, always check hardware:

- Are the batteries working?
- Is everything connected properly?
- Did the code upload correctly?

Once you have established these basic things are in place, you can start looking at the code line by line:

- Are there any statements not inside the correct bracket?
- Are there any sets of instructions that contradict another set of instructions?

Remember to tell your students not to get disheartened by code not working the first time – in the coding world, failing is good!

Coding tips: commenting

Did you know?

In the real world, programmers work as part of teams. In teams, it's important to share thoughts, ideas and processes quickly and easily.

Documentation is important



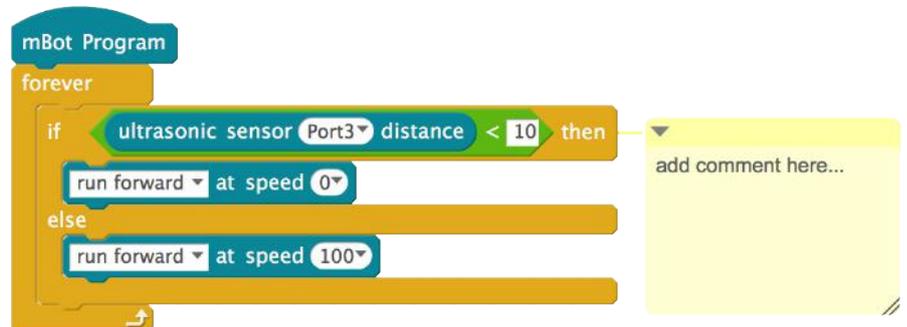
It may seem like extra work at first, but documentation can save time in the long run.

An important aspect of good code is documentation. This is part of making your code easy to understand. In the real world, programmers work as part of teams. In teams, it's important to share thoughts, ideas and processes quickly and easily. That's where comments come in.

Why comments are important

Programs can include comments, which are ignored by the computer but are meant for humans to read. When you write code, it is helpful to add comments to let other people know what the code does or to remind yourself what it does when you've not looked at it in a while.

In the mBlock language you can add a comment to any block by right clicking on it and clicking on, add comment, for example:



Comments are very important for sharing code, so that anyone reading, editing or debugging the original code in the future is able to understand the ideas and intent of the original author.

Troubleshooting guide

Remember!

Tell your students not to get disheartened by code not working the first time – in the coding world, failing is good!

mBot has many pieces



How many pieces do you think are in a full-sized car?

This Subject Update provides the solutions to common problems you might encounter on your robotics journey with the mBot.

This troubleshooting guide is organized into two sections. The first section 'General errors getting started' includes common issues building the robot, uploading code, and working with sensors. The second section 'Difficulties with lesson challenges' includes some tips if any of the lesson challenges aren't running as expected.

General errors getting started

The screwdriver doesn't fit the screw.

The screwdriver can work with two different screw types. To change the screwdriver head between Philips head and Allen key, pull the metal part from the handle and insert it back into the handle the opposite way around.

The robot drives opposite of what is expected, e.g. backward when you press the forward / up button.

The motors might be wired the wrong way around. To fix this, disconnect and reconnect the motor wiring in the correct arrangement.

The robot does not respond correctly to the remote control (without additional coding).

Check what mode the mBot is in. It should start off in mode A manual remote control driving. To get back into mode A, press the A button and check that the mBot's LEDs are white. If the robot does not respond at all, check the batteries and power. Finally you can try re-uploading the default program to the mBot (see the Subject Update mBot's default program).

I can't see the Upload to Arduino button.

In mBlock, make sure you have selected the following menu option:
Edit > Arduino mode

The program won't upload to the mBot

1. First check the cable is connected correctly between the mBot and the computer.
2. Check that the mBot is turned on.
3. Using mBlock, make sure you have selected the following menu options:
 - Boards -> mBot (mCore)
 - Connect -> Serial Port -> last option (usually COM#)

If it still doesn't work after these steps,

1. Try restarting mBlock and the mBot.
2. Then try swapping the USB port the cable is connected to on your computer (some USB ports are different).
3. Swap the USB cable for a different one. Cables can become faulty, so try one that is working for a different group.

SUBJECT UPDATE

General errors getting started (continued)

Not all my blocks of code are being run.

Make sure they are all connected together in mBlock. Sometimes code blocks can be near each other but not actually connected.

The sensors aren't working.

Make sure they are fully connected to the correct ports.

Difficulties with lesson challenges

The robot doesn't stop in time when using the ultrasonic sensor.

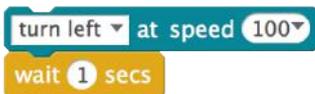
Make sure the distance that you have chosen in the ultrasonic sensor block isn't too small:



This will ensure you give the robot time to stop. The distance you need to set will be affected by the speed. If the robot is driving fast, you may need to make the distance larger.

The robot isn't turning the correct amount.

Turning is affected by the speed of the mBot, how much time it is turning for, and the characteristics of the surface the mBot is on.



If the mBot needs to turn more (a sharper turn), try increasing the amount of time it is turning for first and then consider increasing the speed. Especially on rougher surfaces, like carpet, you may need the mBot to turn for a longer time or more quickly than you need on a smoother surface, like tile.

Robot keeps losing the black line when working with the line follower.

Black lines might be too close together or too narrow. Bad lighting can also confuse the sensor.

The robot doesn't have space to turn.

Try adding code blocks to reverse the mBot first.

The robot doesn't respond as expected to the remote control when the students program it.

Refer to **Answer Sheet 5** from Code Smart to help you.

Did you know?

Bad lighting can confuse the line follower sensor.

Robot fail



Professionally-designed robots fail too.

Remember!

Make sure to tell the robot to check for obstacles first, before telling it the line follower code.

Difficulties with lesson challenges (continued)

The robot lights stay on after I've finished with them.

If you do not turn them off the mBot lights will stay on as long as the robot has power. You should include a block like the following to turn off and reset the lights when mBot is driving forward.



I can't hear the buzzer.

Check the note isn't for too short a time or too high. Many people have trouble hearing high frequency notes, which end in larger numbers.

I can't get the robot to act as a proximity sensor.

Common issues include putting the if statements in the wrong place and using the wrong distance checks for the ultrasonic sensor. Use **Answer Sheet 5** from Code Smart for further support.

The robot crashes into obstacles when following a black line.

Check that the if statements are in the correct order. Make sure to tell the robot to check for obstacles first, before telling the line follower code. Use **Answer Sheet 6** from Code Smart for further support.

The robot can't get out of a dead end.

Make sure the mBot is turning for long enough time. Alternatively, if there isn't enough space, you might have to make mBot reverse first.

The robot can't navigate around an object.

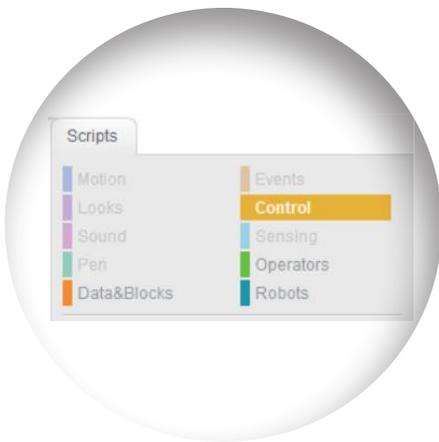
The main problem is not calculating the length of the shape properly. Use **Answer Sheet 6** from Code Smart for further support.

mBlock glossary

Did you know?

Learning to program a robot includes coding and is a bit like learning how to speak the language of the robot.

mBlock script types



mBlock scripts are organized into types. Code Smart focuses on 3 types.

To help you on your journey, this glossary contains mBlock script blocks that are used in Code Smart as well as some common words used in the coding world. Soon enough you'll be comfortable with all the new lingo!

How to use this glossary

This glossary is divided into two primary sections: 1) mBlock script blocks and 2) common words used by coders.

The mBlocks script blocks section is organized by block type and the blocks appear in the same order as in the mBlock environment. There is an image and description of the blocks used. Additionally, extra notes like what is in the drop-down menus of each block are included for your convenience. The drop-down menus in each block will be referred to by number, indicating its position within the block left to right.

The common words used by coders is organized alphabetically like a typical glossary.

mBlock glossary

mBot script blocks

Robots These blocks either gather data from the robot as inputs or result in actions of the robot as outputs.

<p>mBot Program</p> 	<p>Description</p> <p>Start script; this needs to be at the top of your program</p>
<p>Movement</p>  <ul style="list-style-type: none">- Drop-down 1: run forward, run backward, turn right, turn left- Drop-down 2: speeds of 255, 100, 50, 0, -50, -100, -255 (negatives indicate movement in opposite direction)	<p>Description</p> <p>Tells robot to move or turn in a certain direction and speed</p>
<p>Motor</p>  <ul style="list-style-type: none">- Drop-down 1: M1 and M2; referring to left and right motors- Drop-down 2: speeds of 255 100, 50, 0, -50, -100, -255 (negatives indicate movement in opposite direction)	<p>Description</p> <p>Adjusts the individual motor speeds; M1 should be the left motor and M2 should be the right motor</p>
<p>LED</p>  <ul style="list-style-type: none">- Drop-down 1: all, led left, led right- Drop-down 2, 3, 4: LED intensity of 0, 20, 60, 150, 255	<p>Description</p> <p>Adjusts the LED lights on mBot</p>
<p>Tone (buzzer)</p>  <ul style="list-style-type: none">- Drop-down 1: various sound frequencies from C2 to D8- Drop-down 2: length of note including half, quarter, eighth, whole, and double	<p>Description</p> <p>Plays a tone; can be used to make buzzer or horn sounds</p>

mBlock glossary

Ultrasonic sensor

ultrasonic sensor Port3 distance

- Drop-down: Port 1, Port 2, Port 3, Port 4
- According to the building instructions the ultrasonic sensor should be connected to **Port 3**.

Description

Is the distance from the ultrasonic sensor to nearest obstacle

Line follower

line follower Port2

- Drop-down: Port 1, Port 2, Port 3, Port 4;
- According to the building instructions the ultrasonic sensor should be connected to **Port 2**.

Description

Represents the line follower robot component, which is composed of a left and a right sensor; see Student Sheet 3a for more information on the meaning of line follower output values

Line follower

line follower Port2 leftSide is black

- Drop-down 1: Port 1, Port 2, Port 3, Port 4;
- According to the building instructions the ultrasonic sensor should be connected to **Port 2**.
- Drop-down 2: leftSide, rightSide
- Drop-down 3: black, white

Description

Indicates whether a sensor (leftSide or rightSide) of the line follower robot component is sensing black or white

IR remote

ir remote A pressed

- Drop-down: all of the buttons on the mBot remote.
- Note the #s on the remote appear as R# and the gear on your room is 'Setting'

Description

Indicates a button on the IR remote

These blocks can be used as inputs into other blocks, and are especially helpful condition values.

mBlock glossary

mBot script blocks

Control These blocks control the code, including when and how long it runs.

Wait

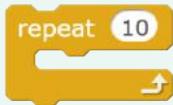


can be changed by typing

Description

Tells your robot to wait a # of seconds before moving on to the next line of code

Repeat



can be changed by typing

Description

Makes whatever is inside it repeat the indicated # of times

Notice the little arrow indicating everything inside will be run from the top again. These are called loops.

Forever



Description

Makes whatever is inside it repeat continuously forever

If



You can put other blocks into the hexagon in the first line to establish a condition to check for

Description

Makes whatever is inside it happen IF the specific condition is met

These blocks allow you to write conditional statements. Whether the instructions inside the block will run depends on whether the condition in the first line is met.

If – else



You can put other blocks into the hexagon in the first line to establish a condition to check for

Description

Makes whatever is inside the first section happen IF the condition is met; ELSE whatever is in the second section will happen

mBlock glossary

mBot script blocks

Operators These blocks are used to do mathematics and construct conditions used in Boolean logic, such as 'and', 'or' and 'not'.

Block - random number  You can set the boundaries of the number range by typing in numbers	Description Choose a random number
--	--

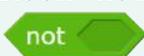
Block - less than  The values can be other blocks or numbers that you type	Description Checks if the 1st value is greater than the 2nd value
--	---

Block - equals  The values can be other blocks or numbers that you type	Description Checks if two values are equal
---	--

Block - greater than  The values can be other blocks or numbers that you type	Description Checks if the 1st value is greater than the 2nd value
---	---

Block - and 	Description Checks if both conditions are true
---	--

Block - or 	Description Checks if either condition or both conditions are true
--	--

Block - not 	Description Checks if a condition is NOT true
---	---

These blocks allow you to fill in conditional statements.

mBlock glossary

Did you know?

Algorithms is a term often used in computing, but you'll also hear doctors use it to describe sets of rules that structure the way they think.

Try to use the vocabulary



You'll probably spend some time debugging your programs.

Common words used by coders

Algorithm – a set of rules to be followed, especially by a computer

Boolean logic – a way of breaking decisions down into simple True/False questions; Boolean expressions include: =, <, >, AND, OR, NOT

Bug – an error/mistake in code which stops a program from running properly

Code – instructions written in computer language

Coding – writing instructions for a computer

Computer – a machine designed to follow instructions and process data

Computer logic – the basic rules that a computer follows

Condition – something a computer must consider before making an action; in computing, this is usually a statement that is either determined to be true or false.

Conditional statements (conditionals) – instructions which tell the computer to react differently depending on a condition

Constant – a piece of data that is fixed, or not variable

Data – information used by a computer

Debugging – fixing code to remove bugs, or errors

If else – a conditional instruction which tells the computer what to do if a specific condition is met and what to do if it is not met

If then – a conditional instruction which tells the computer what to do if a specific condition is met

Input – information or instructions which you put into a computer

Loop – a section of code which repeats

Output – the result you get from a computer

Nested loop – a loop inside of another loop

Program – a set of instructions in computer language which tells a computer what to do

Programming language – a language designed for computers

Run – to set a program going

Script – an instruction or a set of instructions in code

Syntax – a way of writing and arranging code so a computer will be able to understand

Variable – an item/characteristic/trait that might change

Futures Thinking

Did you know?

Futures Thinking is a cross-disciplinary approach to considering potential futures through the exploration of trends

OECD Futures Thinking in Action



Learn more about Schooling for Tomorrow: <http://www.oecd.org/education/school/schoolingfortomorrow-thestarterpackfuturesthinkinginaction.htm/>

The future will not be the same as the present and the future is not fixed or confirmed. However, we have the ability to shape what the future will be like through our decision making and actions now. As the saying goes, 'children are our future', and education is all about preparing them for the future.

What is Futures Thinking?

Futures Thinking is a cross-disciplinary approach to considering potential futures through the exploration of trends and drivers for change that may lead to different future scenarios. This includes evaluating what scenarios are possible, probable and preferable futures. This is not about predicting the future, but rather critically considering the future, so that we can better make decisions and take actions in the present.

Futures Thinking enables you to consider the major changes in the next 5, 10, 20 or more years in all areas of life, including social interaction, education and technology. While the future cannot be reliably predicted, we can critically consider the future, so that we, as individuals and groups, can be more deliberate with actions, decisions and policies that may help promote desirable futures and help prevent undesirable ones.

Why is it important?

It is especially important that everyone consider the future to offset short-term thinking often driven by short-term gains, such as profits or re-election, and to ensure that there is diverse input shaping a future that is preferable to all people. Futures Thinking has been applied to the technology and policy sectors such as energy, environment and transport. However, other sectors, such as education, have been less challenged or only recently challenged. The OECD has been leading the way on Futures Thinking about education, so that rapidly transforming societies are able to prepare the youth of today for the challenges of tomorrow. More information on this can be found on the OECD website <http://www.oecd.org/education/school/schoolingfortomorrow-thestarterpackfuturesthinkinginaction.htm>. Only by considering the future can we deliberately help shape it.

Did you know?

Only by considering the future can we deliberately help shape it.

How to incorporate Futures Thinking into your classroom

Futures Thinking is not limited to policy makers and corporate executives. It can apply to all areas of life and be done by anyone. Thus, we encourage you to engage in Futures Thinking with your classroom. In order to engage with Futures Thinking, there are a few things that you must consider:

- **Existing situation** – What is happening now and why? Who benefits and who loses?
- **Trends** – How does the existing situation compare to the past? Are there patterns in the changes?
- **Drivers** – What is causing the changes? The causes might be specific community perceptions, beliefs, values or attitudes. It might be that other changes have caused ripple effects, such as demographic changes, environmental damage, developments in science and technology or changes in political policy.
- **Possible futures** – What might happen in the future?
- **Probable futures** – What is most likely to happen in the future? Which trends and drivers are likely to persist?
- **Preferable futures** – What do you want to happen in the future? Why? Who benefits and who loses?

You can incorporate these questions into discussions, journal entries, research projects and presentations.

Tips for incorporating Futures Thinking in your classroom

1. To structure Futures Thinking discussions, it is often helpful to choose one sector or topic, like what transportation might be like in the future.
2. Establish a couple of discussion rules with your students that will allow your students to share openly and critically discuss without making disagreements personal. It is often helpful to remind students that all ideas are worth considering and that the purpose of this process is to consider which ideas are possible, which are probable, and which are preferable. All of history's greatest thinkers have had both 'good' and 'bad' ideas.
3. Encourage students to consider multiple perspectives. Students should consider their personal perspectives and be prepared to share with each other. Draw in outside perspectives using articles, videos, or even communications with people outside of the classroom. This is an opportunity to incorporate research using secondary resources, and even an opportunity to do primary research through chats with other classes for example.
4. Consider the questions from different levels: individual, local, national, and global. If something is good for the individual, is it automatically good for everyone in the world (or even the Earth itself)?

About Oxbotica

Did you know?

Computers are better suited to receive large amounts of data and don't get distracted. Autonomous vehicles can help make roads safer

Oxbotica is developing the next generation of autonomous vehicles. An important part of this is the artificial intelligence software that allows vehicles to interpret their environment and respond appropriately. Using the latest in computer vision and machine learning, the systems learn from their environment and share experiences with each other, so that they're getting smarter all the time.

The intelligent system within the autonomous vehicle

Currently, vehicles on the road rely on the intelligence of humans to decide what to do. So in order to have autonomous vehicles of the future, there must be an intelligent system that decides what the car should do. Oxbotica is developing this.

Oxbotica has developed a system called Selenium that uses patented algorithms to interpret the environment around the vehicle and to navigate the vehicle safely and efficiently. Different sensors, that are mounted onto the vehicles, take in data about their environment. Selenium then uses this information to make decisions about where to go next for example turning right, slowing down, or stopping. What makes Oxbotica's software solution special is its ability to learn. Data from different vehicles is shared within Selenium, so that the system actually becomes more intelligent over time.

Oxbotica Geni



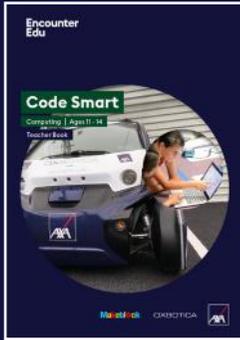
The Geni vehicle uses a software system called Selenium.

What are the applications that are being explored?

Computers are better suited to receive large amounts of data and don't get distracted. Autonomous vehicles can help make roads safer and reduce the amount of wasted time in traffic. They can also increase accessibility of transportation for people who can't currently drive because of physical impairments like poor eyesight or epilepsy. More efficient driving enabled by intelligent systems will also reduce carbon emissions and road wear and tear, both beneficial for the environment.

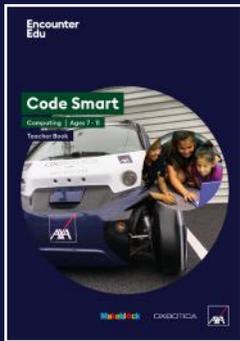
Oxbotica is partnering with other companies to trial applications in different environments, for example at airports, in mines, warehouses, and even in space. Airports require ground vehicles to transport cargo and supplies to and from airplanes; however, many of these vehicles are used inefficiently. Oxbotica CEO Dr. Graeme Smith said, "Airports offer an incredibly interesting domain for our autonomous driving software. There is a huge diversity of vehicles, each with a very specific mission." If the trial goes well this could lead to reduced fleet size at the airports and a whole system of different autonomous vehicles at the airport from push back tugs, passenger load bridges, baggage vehicles, and more.

This book and associated resources can be accessed from encounteredu.com/teachers/units/code-smart-11-14



-  Videos and interactive diagrams
-  Individual lesson and resource downloads
-  Live broadcasts with coders and robotics scientists
-  Subject Updates and training courses

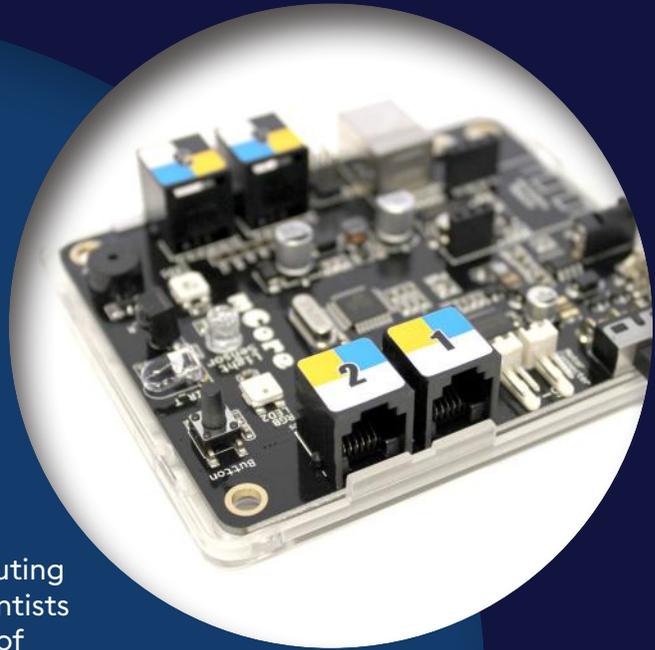
Other books in this series



Code Smart 7-11

Photo credits

Cover	Autonomous vehicle: Oxbotica Students: Start Dream Big
Student sheet 2a	mBot: Makeblock
Student sheet 2b	'DC motor' by Dcaldero8983 on Wikimedia Commons is used under a Creative Commons Attribution-ShareAlike 3.0 Unported license < https://creativecommons.org/licenses/by-sa/3.0/deed.en > 'Servo motor' by oomlout on Wikimedia Commons is used under a Creative Commons Attribution-Share Alike 2.0 Generic license < https://creativecommons.org/licenses/by-sa/2.0/deed.en > 'Stepper motor' by oomlout on Wikimedia is used under a Creative Commons Attribution-Share Alike 2.0 Generic license < https://creativecommons.org/licenses/by-sa/2.0/deed.en >
Student sheet 3a	mBot exploded view diagram: Makeblock
Student Sheet 4a	Lidar viewer: Oxbotica mBot exploded view diagram: Makeblock
Student Sheet 6a	LEDs: Pixabay, halejandropmartz
Student Sheet 6b	mBot: Makeblock
Student Sheet 9a	Bus shelter ad is a derivative of 'Bus shelter' by Albert Bridge on Geograph.ie used under a Creative Commons Attribution-ShareAlike 2.0 Generic license < https://creativecommons.org/licenses/by-sa/2.0/deed.en >
SU How can autonomous vehicles be useful?	Parking: Pexels, Tuur Tisseghem
SU What questions does an autonomous vehicle need to answer?	Geni vehicle: Oxbotica
SU When can a car be considered autonomous?	Hands-free driving: Oxbotica
SU How to set up mBlock	mBot cartoon: Makeblock
SU Coding tips: writing elegant code	mBot cartoon: Makeblock
SU Coding tips: failing and debugging	Robot fail: DARPA
SU Coding tips: commenting	mBot cartoon: Makeblock
SU Troubleshooting guide	mBot components: Makeblock
SU mBlock glossary	mBot cartoon modified with bug from mBot cartoon by Makeblock
SU About Oxbotica	Geni vehicle: Oxbotica
All other photos	Encounter Edu



Code Smart is a new approach to teaching computing to prepare students for the future. Computer scientists and engineers are designing the next generation of driverless cars and the careers and opportunities in this space are expanding rapidly. Set against this real-world context of autonomous mobility, Code Smart brings the excitement of robotics, artificial intelligence and coding into the classroom.

In nine lessons, students will learn about aspects of making, programming and design thinking as they build and program their own robot cars. Working in teams, they will tackle coding challenges based on the problems that engineers are faced with in designing autonomous vehicles. In a final design challenge, students will work on not only the technical aspects of driverless cars, but also on how they can improve lives and create smarter and safer communities.

**Where
Learning
Meets
The World**

www.encounteredu.com

Encounter Edu designs and runs STEM and Global Citizenship education programmes, which make use of virtual exchange, live broadcast and virtual reality. These technologies create classroom encounters that widen young people's world view. Learning is further underpinned by an online library of teacher resources and training. Combined, these provide children with the experience and knowledge to develop as engaged citizens and critical thinkers for the 21st Century.