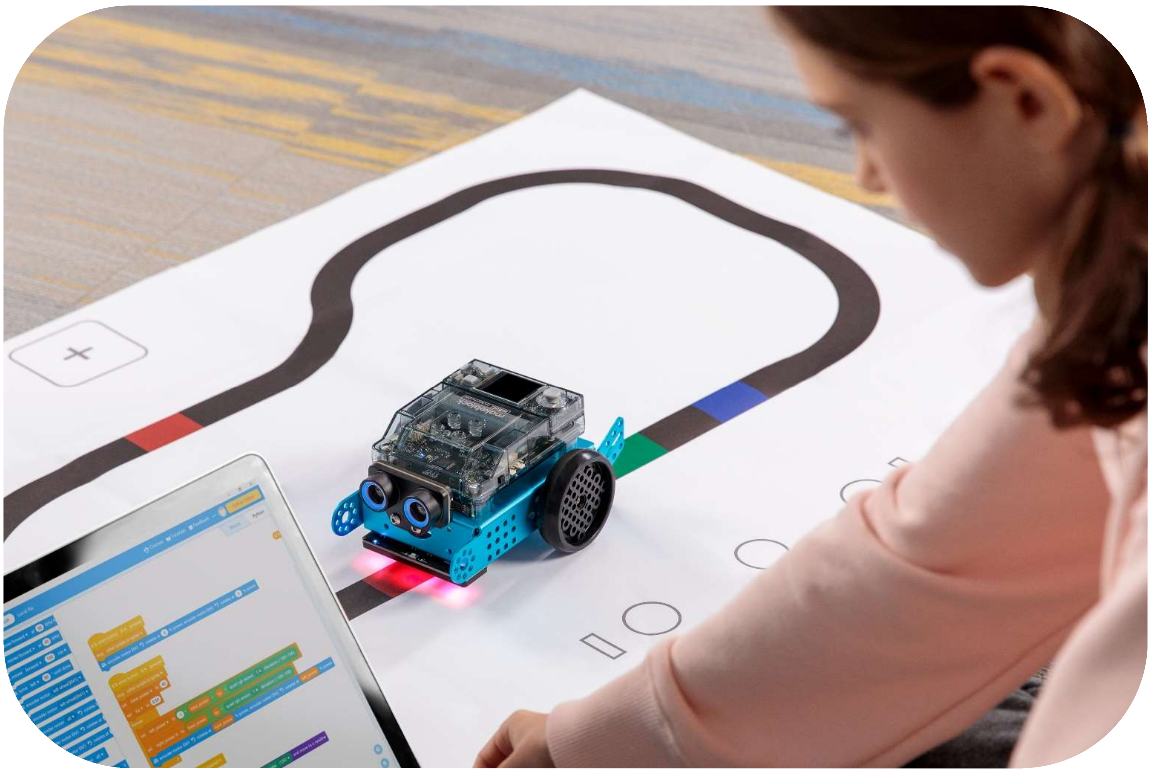
mBot2

Getting Started Activities



Getting Started Activities

Contents

Introduction

Lesson 1 – Let's move

Lesson 2 – Sensing = data

Lesson 3 – Listen to mBot2

Lesson 4 – Seeing with sound

Lesson 5 – Sightseeing

Lesson 6 – Careful drive

Lesson 7 – A network game

Lesson 8 – mBot2 at your service

Lesson 9 – mBot2 in the wild

Introduction

1. Pedagogical approach

With Makeblock Education, the development of knowledge and skills is the combination of digital and physical tools that work together to provide a hands-on learning experience. Students actively engage with the lesson's topic in order to solve a problem or create something new. It allows them to develop theoretical concepts from practical experience as well as transfer and apply these concepts back to practical tasks, deepening their understanding. This includes learning from mistakes through detailed and systematic analysis of the processes involved.

With hands-on learning, students manage and foster their education process by playing a more active role during the class, rather than just listening to a lecture from the teacher.

This introduction will give an overview of the new features of mBot2 and the benefits for its use in education. It will briefly show the steps to start programming the robot with mBlock5, Makeblock Education's block-based editor, and then portrait the individual Getting Started Activities. These activities are a staged learning experience and cover the new features step by step. Starting with real-world references in the tasks given, the background of sensors, actuators and computational thinking are discussed along with sample codes that are easy to understand and expand for each task. The activities always suggest to involve further digital media and think ahead to apply the learning to new challenges and end with a reflection of the activity.

2. An introduction to mBot2

mBot2 is a next-generation educational robot designed for Computer Science and STEAM learning. Having extended capabilities makes it an ideal entry-level solution for lower secondary education, but it can also go all the way to upper secondary and beyond. mBot2 is designed for students to carry out interactive and smart lessons that are engaging, fun and reflect real-world applications of cutting-edge technologies.

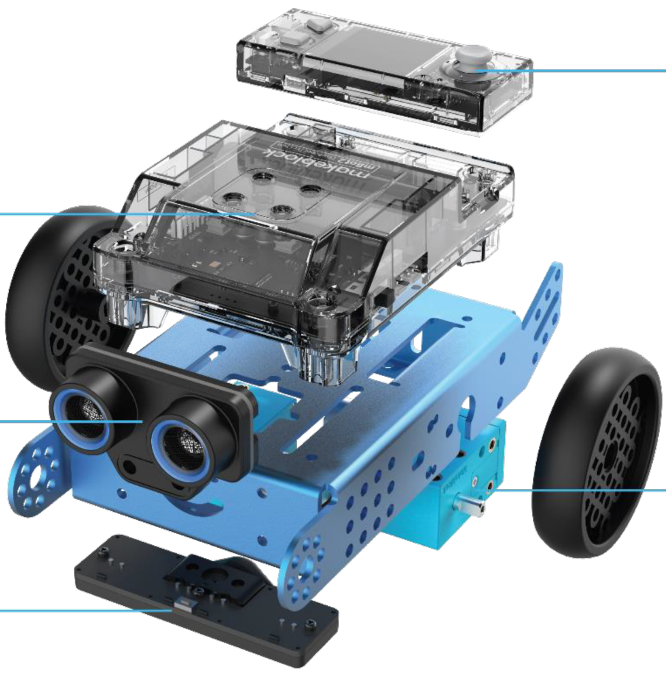
mBot2 is powered by CyberPi, a powerful and versatile microcontroller. Its integrated sensors and actuators - like microphone and speaker, inertial measurement unit with gyroscope ad acceleration sensor, a light sensor, buttons for menu operations and a color display – are complemented by an ultrasonic range sensor and a line-following sensor with four RGB elements.

Next to this range of sensors and actuators, the mBot2 is capable of Wi-Fi communication, allow for a wide range of applications on curriculum topics for Coding, Robotics, Data Science and Artificial Intelligence, in connection with other subjects such as Maths, Physics, etc.

Teachers can, for instance, wirelessly connect multiple mBots in a classroom, to create a local network of robots that communicate between themselves, share information, and perform tasks. They can also use a standalone CyberPi as a smart device to communicate with the mBot2, creating a smart ecosystem or a fun remote controller. When connected to the Internet, mBot2 can perform advanced features such as speech recognition or communicate with the cloud to obtain information.

The sensors and actuators specific to the mBot2 will be introduced here in the following sub-sections and the full range will be part of the getting started activities. You will find an overview of them at the end of this document as well.

Here is an overview of the different components that the mBot2 consists of:



CyberPi

ESP-32 microprocessor for wireless communication. Compatibility with block-based and Python coding

mBot2 shield

Compatible with a wide variety of external components, includes a built-in rechargeable battery

Ultrasonic sensor 2

Object detection is accompanied with 8 programmable LEDs for an enhanced interaction

Encoder motors

Measure rotation to 1-degree accuracy, distance travelled and precisely control speed up to 200RPM

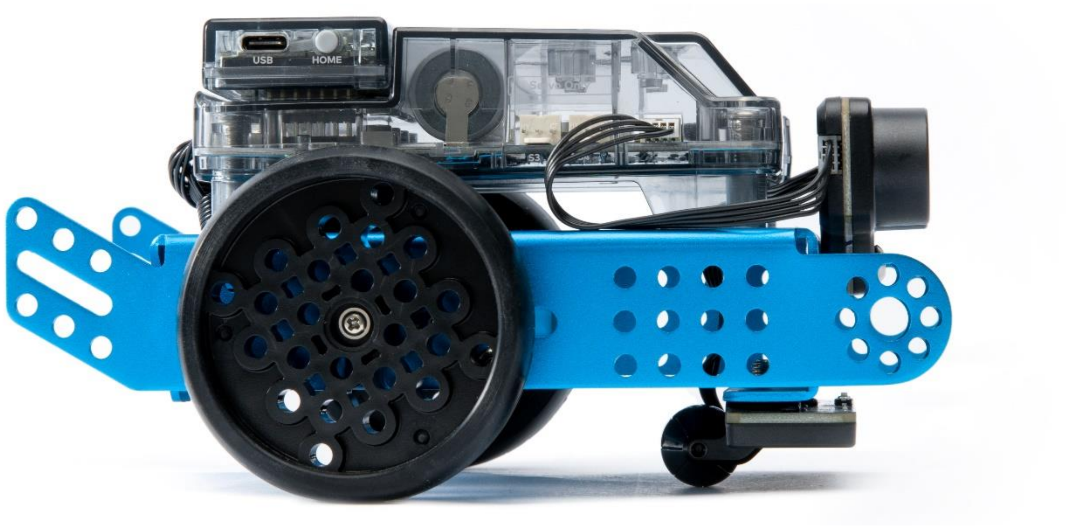
Quad RGB sensor

4 sensor probes support color recognition, as well as basic and advanced line detection programs

2.1 Encoder Motors

mBot2 motors are equipped with optical encoders that enable high accuracy control. With this, students can precisely control the rotation, speed and position of the wheels and the robot. Additionally, the motors can also be used as servos, and even as knobs, to feedback data to the system, just like a sensor.

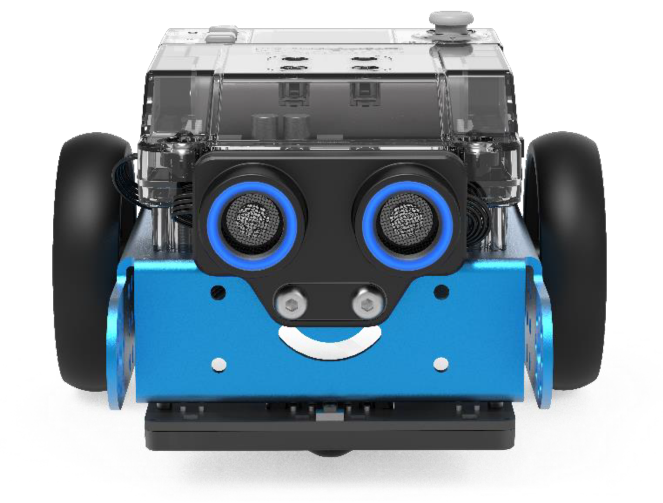
Thanks to these features, lessons can be more realistic and educational in comparison with robots that have less control of their motors. Activities can do a proper integration of Maths concepts, for instance, by driving precise distances, calculating exact turns and even mapping a route through a maze and transferring the results back to the computer.



Lesson 1 from the Getting Started Activities offers an introduction to the different coding blocks for the Encoder Motors in the programming environment mBlock, as well as an easy assignment for students to explore the possibilities by themselves.

2.2 Ultrasonic sensor and Quad RGB sensor

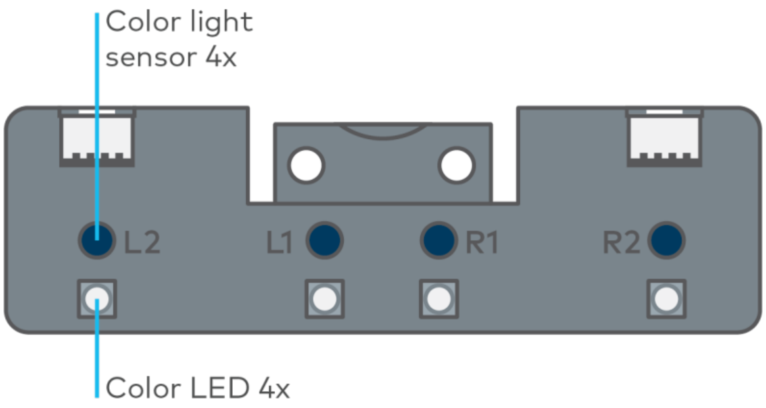
The ultrasonic sensor that comes with the mBot2 can perform more accurate and consistent readings than its predecessors. Additionally, it incorporates blue LEDs that support new ways of interaction with the robot. Students can use these additional lights to display responses.



mBot2 integrates another advanced component: the Quad RGB sensor. This sensor has the capacity to not only identify colors, but also to track lines in order to help the mBot2 follow paths, or detect junctions or 90° turns - all at the same time.

The basic functions can be carried out with little prior knowledge, but due to its advanced features, it's important to know what are the best practices in order to make the most out of the sensor.

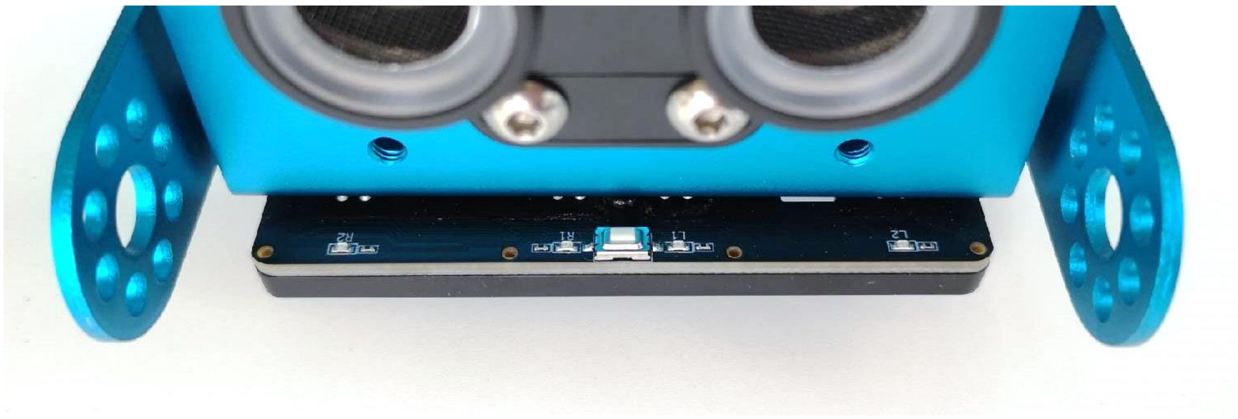
Instead of using 2 sensors to track a single line in front of the robot, the Quad RGB sensor has 4 sensors that allow it to identify a broader range of conditions: while following a track, the sensor can identify junctions on the left, right or even both sides, which means that the mBot2 can navigate through more complex maps than other robots. And since each of the four sensors is a color sensor of its own, colored marks can indicate positions on the map – the robot "knows" its location.



This wider detection range can be used for maps with a network of lines with multiple mBot2s moving between different stations as in a "Smart Warehouse", or simulate the roads in a city, with its different traffic signals and traffic rules.

Previous line following sensors used infrared light that is not visible to the human eye. These sensors instead operate on a single wavelength, while the new one looks at three different wavelengths simultaneously, just like the human eye: red, green and blue. A mixture of the intensity of theses wavelengths is interpreted by our brain as color perception. The sensor works similar and next to perceiving colors, it tries to differentiate a track from the background for line-following tasks.

With the provided map, we have extra color markers inside the track to trigger additional actions, if programmed accordingly. In order to make the sensor interpret these colors as part of the line, it must be calibrated to the brightest color on the track, which is yellow. Place the sensor over the yellow color code while the mBot2 is switched on and double-click the small button on the top side of the sensor (see the image below). As the LEDs will start flashing, swipe the sensor across the color and white background until the flashing stops (2-3 sec.). After that, the color will be regarded as part of the track (for an in-depth discussion see the additional info on the Quad-RGB-sensor in different environments and lesson 5).

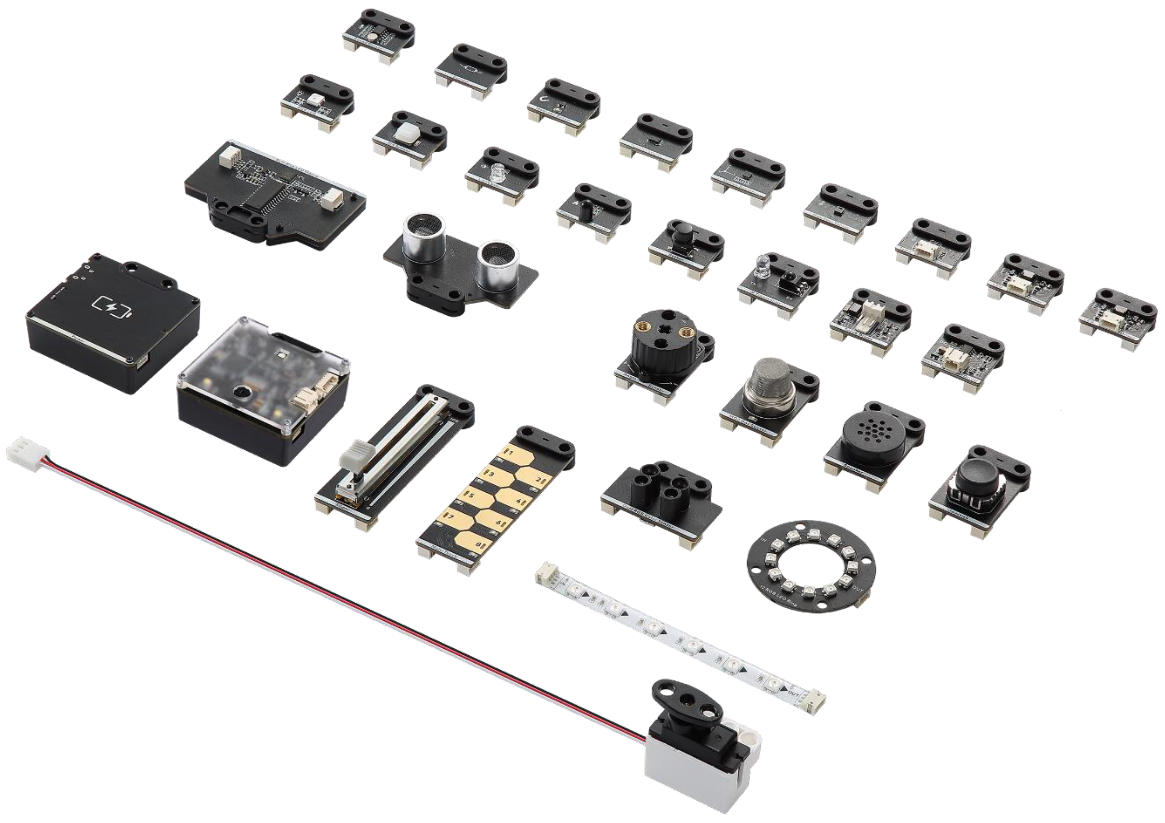


Lesson 5 from the Getting Started Activities provides an explanation of the different blocks that are available in mBlock for the Quad RGB sensor, and students can also practice their learnings with a simple assignment.

2.3 More possibilities with mBot2

The scope of activities can be further expanded by combining the mBot2 with the mBuild modules developed by Makeblock Education. Thanks to their internal Micro-Controller Unit (MCU), these smart sensors and actuators can be directly connected without the need of complex wiring or setups, allowing students to spend more time ideating and creating. These components use a single type of connector that can't be reversed, so students can make no mistakes in connecting them. This gives them more confidence and therefore a better learning experience.

Some examples of the sensors and actuators are: smart camera, multi touch, slider, temperature sensor, etc. With all these sensors, teachers have a variety to choose from for creating real-world scenarios and adjust to teaching and curriculum needs – and students' interests.



But with the mBot2 alone you already have plenty of opportunities (see the activities), plus the mBot2 Shields offers further connections: the integrated 2-pin and 3-pin interfaces can be used to directly connect DC motors, servos, LED strips, and even third-party components that are commonly available, including a wide variety of Arduino-compatible sensors, or even make custom ones.

3. Programming mBot2

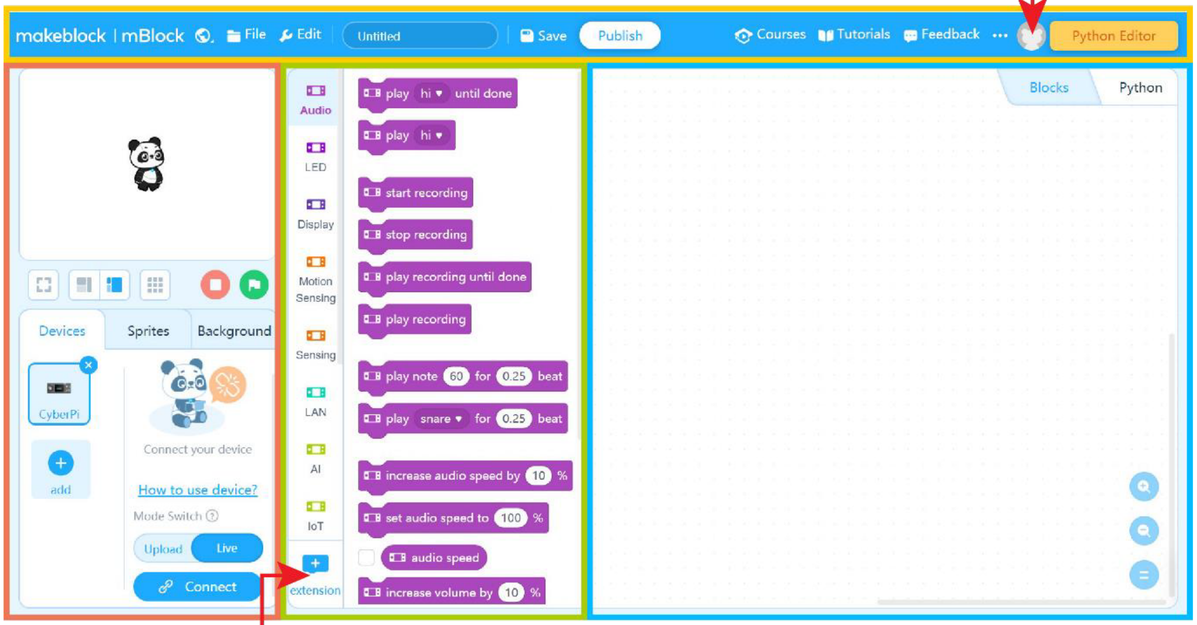
mBlock is the coding platform for mBot2, designed to deliver an enhanced educational experience, and a continuous path of growth for the student. Thanks to the extensions in mBlock5, educators can easily include some of the latest and most influential technologies in their lessons, such as the Internet of Things or Artificial Intelligence. Also, by integrating both block-based coding and Python, mBlock5 offers a learning path for students to develop from basic to professional computational skills.

mBlock5 can be used as installed software on computers, laptops, and mobile devices, or on a web browser. It's compatible with different operating systems such as Windows, Mac, Linux, Chromebook, iOS, Android. Being open source, mBlock5 offers the opportunity to create new extensions for software and hardware, which means that educators can customize the coding tools according to their needs. Users can also search and share projects in the the Makeblock Community.

To start programming with mBlock is as easy as dragging and dropping blocks. The mBlock interface has the following elements:

Set your account to access the cloud services

Menu Bar
Select language, open or save a file, find example programs, etc.



Stage area
View project stage, customize sprites and backgrounds, connect devices.

Block area
Library of code blocks sorted by color-coded categories.

Script area
Drag the coding blocks to this area and arrange them to control the stage or connected devices.

Extension center

Detailed information about the characteristics and use of mBlock can be found in https://education.makeblock.com/help/category/mblock-block-based/

4. Outline of activities

The mBot2 Getting Started Activities are developed for students between 11 to 14 years old and their educators. Overall, the lessons are designed to be accessible, despite having different and increasing complexity levels.

Each lesson introduces a different and exciting feature from the mBot2, along with some basic concepts of programming. The knowledge gained from each lesson helps to build towards more advance activities, while the lessons increase in complexity as the learner moves on. For example, the movements of the robot are introduced in Lesson 1 and this is used in many of the subsequent lessons. The possibilities of the Quad RGB sensor are presented in lesson 5, to be later used in lesson 8.

The lessons are briefly described below:

Activity Name	Description	Key concepts
1. Let's move	Students discover the mBot2 and the mBlock software and learn how to drive the robot with precision. This knowledge will be used in most of the following lessons. Students will also design a simple maze and program the mBot2 to (manually) navigate through it.	Precise movements and corresponding coding blocks.
2. Sensing = data	Students will work with the different sensors integrated in the mBot2; they will learn how to use them with their corresponding code blocks, and to visualize data from the sensors on the integrated full color display.	Mode of operation of the sensors. Different approaches to display and visualize data on the display. Differences between Live and Upload mode in mBlock 5.
3. Listen to mBot2	Students will learn how to control the speaker and microphone with the code blocks in mBlock 5. They will also create a program where the mBot2 plays a recorded sound if it meets a certain condition while driving around.	Text to speech (TTS) and Voice recognition (Speech to Text, STT) using the build-in speaker and microphone. Running multiple tasks side by side.
4. Seeing with sound	Students will learn what ultrasound is, how it is used in a sensor, and they will also create a program to make the mBot2 drive in a loop by turning after detecting obstacles on the road.	Detecting an obstacle or a range by using the ultrasonic sensor. Using loops and conditional statements for making the mBot2 drive while avoiding obstacles.
5. Sightseeing	Students will learn how a color sensor works, how are they used in real life, and they will program the mBot2 to become a tour bus that visits different landmarks in a city. This knowledge will be also applied in lessons 7 and 8.	Mode of operation (physics of light) of the color sensor/line follower. Color and line identification- Making the mBot2 follow a line and having it perform actions based on color detection.

6. Careful drive	Students will learn how to use the gyroscope accelerometer of the mBot2 and its code blocks, and they will program the mBot2 to adjust its driving behavior if it detects inclinations on the road.	Mode of operation of gyroscopes and accelerometers (as Inertial Measurement Units, IMU). Coding the mBot2 to adapt to road conditions based on IMU data.
7. A network game	Students will learn to have multiple mBot2s communicating with each other wirelessly without the need of a WIFI access point. They will program a simple game where multiple mBot2s search for a color and the first one to find it wins. This knowledge will also be used in lesson 8.	Wireless data transfer in ad-hoc networks. Data exchange in loops and events.
8. mBot2 at your service	By learning how to set up a WIFI connection with the mBot2, students will also learn to use onboard speech recognition, and they will apply this knowledge in an activity where the mBot2 becomes a robot waiter who talks to its customers.	Using WIFI infrastructure mode with the mBot2 for speech recognition and speech synthesis. Offloading heavy computing like speech recognition to cloud services. Structuring code by applying "own blocks" (functions).
9. mBot2 in the wild	In this special lesson, students will learn some principles of Artificial Intelligence by using the Teachable Machine extension in mBlock5. They will apply their knowledge to recreate a natural ecosystem where the mBot2 behaves like an animal.	Learning about Machine Learning and applying it with local processing only on block-based programming. Establishing a new communication protocol between the mBot2 and the computer.

Throughout the lessons students are encouraged, with the consent of educators, to document their learning outcomes by video and publish them, as a way to gain pride of their work and lead their conversation about Computer Science and STEAM in the classroom.

makeblock education

ROBOTIX Hands-on Learning

Lesson 1: Let's move!

Subject: STEAM

Grade(s): 5th and up

Duration: 45 minutes

Difficulty: Beginner

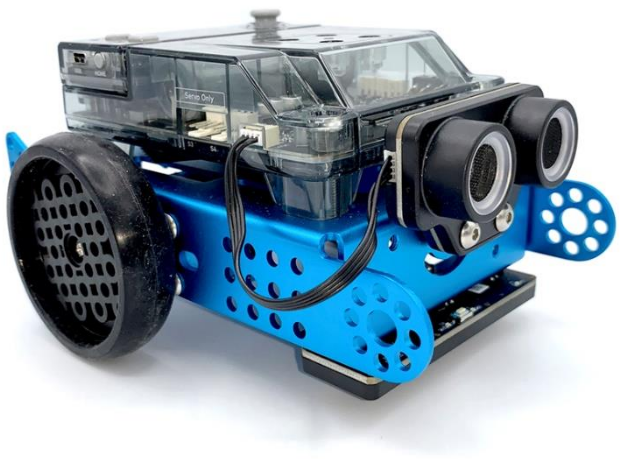
Lesson Objectives

By the end of this lesson, students will be able to:

- Drive the mBot2 with precision.

Overview

Robots are autonomous machines that replace human effort; they can perceive their environment and perform computer programs to make decisions and take actions. A computer program is a set of instructions and conditions run by a computer to perform a task. You may think you have never seen or used a robot, but that chance is slim. Robots can take many different shapes and have different capabilities depending on their context. For example, robots are present in households, such as the vacuum cleaner robot or a lawn mowing robot; you can also find them in factories assembling products. We have even sent robots to other planets.



Focus

By the end of the lesson, students will know:

- What movements can mBot2 do
- What different programming blocks you can use to make the mBot2 move

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for ChromeBook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle
- A3 paper
- Pens

Lesson plan

Duration	Content
5 minutes	1. Warming up • Sensors and data in everyday life. • What is the CyberPi?
10 minutes	2. Hands-on • Getting acquainted with the different programming blocks of the mBot2.
25 minutes	3. Trying out • Programming your own robot through a self-made maze.
5 minutes	4. Wrap-up • Showtime: show what you did with your robot in a fun, short movie for later discussion. • If your teacher allows, share the end result on social media with the hashtag #mBot2moves • Reflection: What are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up
(5 min)

Step 1: Warming up

This step consists of two parts:

1. Robots in everyday life
2. Getting to know the mBot2

1. Robots in everyday life

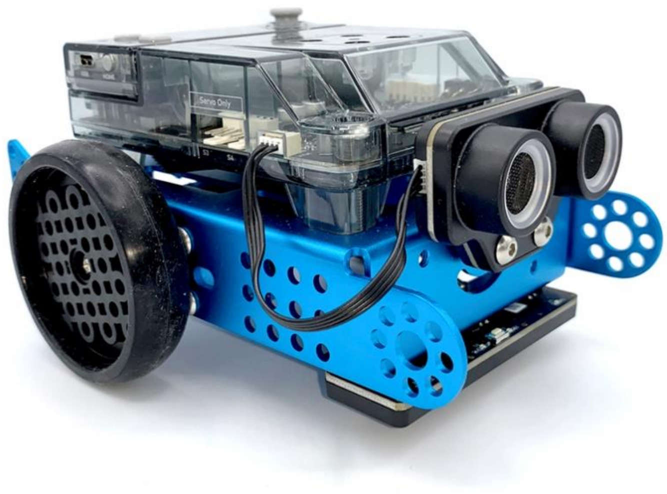
We have robots in many different places. Some people have them in their homes and others work with them every day. Robots can take many different shapes and have different capabilities depending on their context. Can you think of three robots that you have to deal with (regularly) in everyday life?

There are probably many more robots that have been developed that you have never seen. Search the Internet for three more robots that you can use in everyday life.

There is a good chance that you will have to deal with robots later on you professional or personal life. There are robots working in many different companies. Usually, these robots perform tasks that are repetitive, that need to be done with great accuracy, or that are dangerous. In this lesson we will be working mainly on programming a robot to move accurately. Can you think of three professions or businesses where it is very important for a robot to work very accurately?

2. Getting to know the mBot2

The mBot2 is a programmable robot equipped with different components that allow it to sense, act and communicate with its environment. The mBot2 is intended for students to learn about Computer Science and Technology, including ways that you can use a robot in real life. The mBot2 can be programmed with the help of mBlock. In mBlock, programming can be as easy as dragging and assembling blocks. As you will see in this lesson, you can, for instance, program the mBot2 to drive through a maze - with your help (not autonomously). mBot2 comes with a pair of special motors, which can record the rotation of the axle and therefore the speed and distance the robot has travelled. They are called encoder motors due to the sensor that is built-in (encoder). This type of motors allows to control specific parameters such as the rotation angle and speed of the motors. You will learn how to make use of these motors in the following steps.



2. Hands-on (10 min)

Step 2: Hands-on

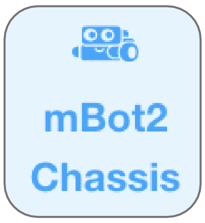
This step consists of two parts:

1. Getting acquainted with the different programming blocks of the mBot2.
2. Recreating and testing some programming examples to control the mBot2.

1. Getting acquainted with the different programming blocks of the mBot2

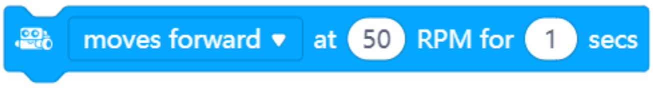
As you observed previously, robots can be used in many different ways. If you have thought of a purpose for a robot, most likely it means that you know what the robot needs to do to help you, and some of these tasks might need to be executed with certain precision.

When you start programming the mBot2, you will notice that there are many different coding blocks you can use to make the robot move. You will find these blocks in mBlock 5 in the category 'Chassis'. These coding blocks are blue.

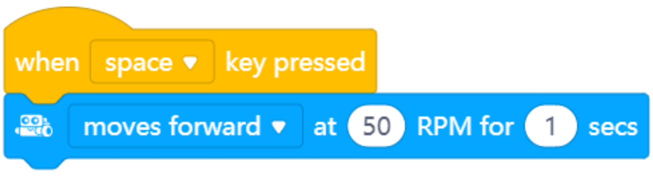

mBot2 Chassis

Below you can see some examples of the coding blocks needed to make the robot move. Some of them look similar, yet they all do something different. For this lesson, we will work in Live mode. Make sure the correct mode is selected in the software. The differences between Live mode and Upload mode will be explained in lesson 2.

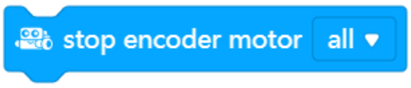
Code block:


moves forward ▾ at 50 RPM for 1 secs

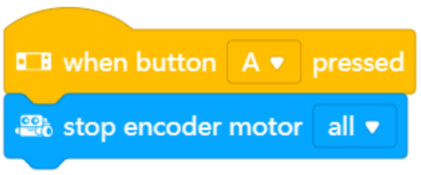
This code block allows you to move the mBot2 forward, backward, left and right at a specific rotational speed of the wheels and for a number of seconds.
The example below shows how you can make the mBot2 move for two seconds at a speed of 50 rotations per minute. This is useful, for example, when the mBot2 needs to push a load forward.


when space ▾ key pressed
moves forward ▾ at 50 RPM for 1 secs

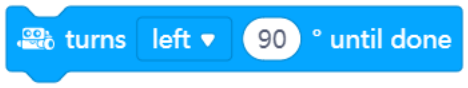
Code block:


stop encoder motor all ▾

This code block allows you to make your robot stop moving. This can be a useful block during a testing phase. If a program does not work as expected you can use this block to make the robot stop immediately.
For example, you can make the robot stop when the A button of the CyberPi is pressed. To do this, use the programming example below.

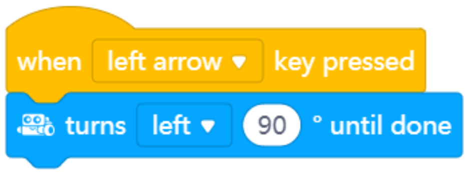

when button A ▾ pressed
stop encoder motor all ▾

Code block:

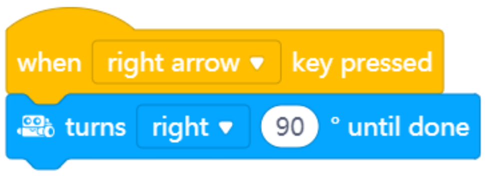


```
turns left ▾ (90) ° until done
```

With this code block you can make the mBot2 rotate by a number of degrees, and you can choose whether the rotation should be to the left or to the right.
When the programming example below is set, the mBot2 can be controlled by using the arrow keys. When pressing the right arrow, the mBot2 will turn 90 degrees to the right, when pressing the left arrow, it will turn 90 degrees to the left.

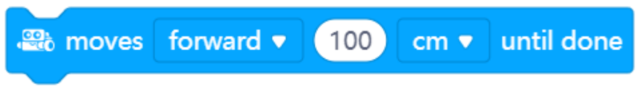


```
when left arrow ▾ key pressed
turns left ▾ (90) ° until done
```



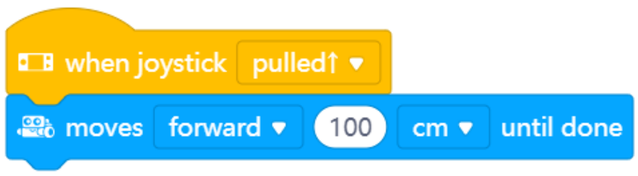
```
when right arrow ▾ key pressed
turns right ▾ (90) ° until done
```

Code block:



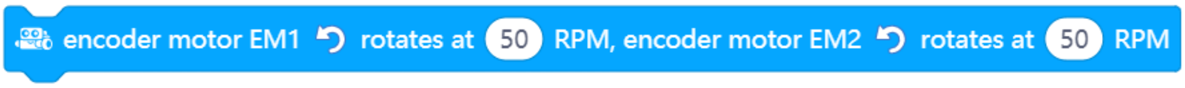
```
moves forward ▾ (100) cm ▾ until done
```

With this code block you can make the mBot2 move forward or backward a certain distance.
When you use the programming example below, the mBot2 will move 100 cm forward. To start this, move the joystick of the CyberPi upwards.



```
when joystick pulled↑ ▾
moves forward ▾ (100) cm ▾ until done
```

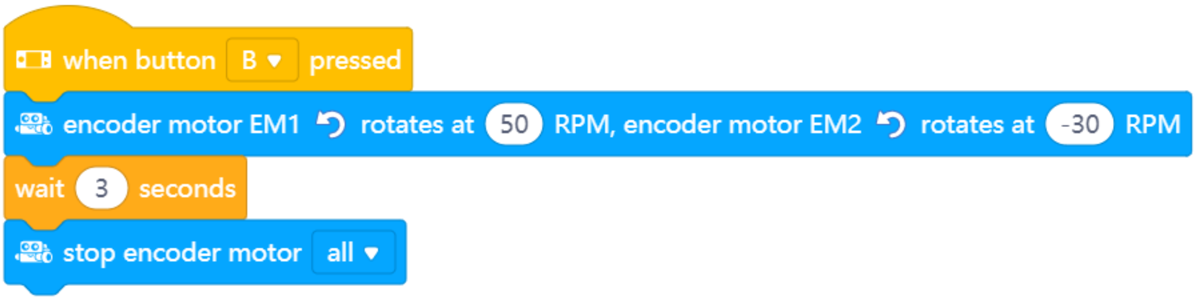
Code block:



```
encoder motor EM1 ⟲ rotates at (50) RPM, encoder motor EM2 ⟲ rotates at (50) RPM
```

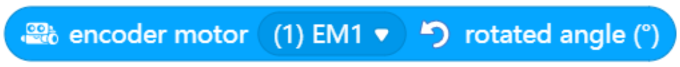
You can also control the motors of the mBot2 independently, and this is one of the blocks which you can use for that.

In the example below, the mBot2 will move in a curve for 3 seconds before it stops. Notice that the value for one of the motors is negative; this is because the motors are mounted in opposite directions, so in order to move the robot in one direction, you need to make one of the motors rotate the opposite way. What would happen if both wheels rotate at 40 RPM?



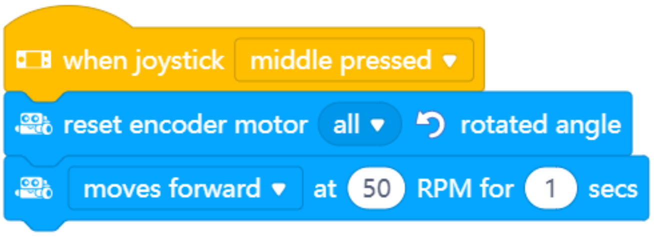
```
when button (B ▾) pressed
encoder motor EM1 ⟲ rotates at (50) RPM, encoder motor EM2 ⟲ rotates at (-30) RPM
wait (3) seconds
stop encoder motor (all ▾)
```

Code block:



```
encoder motor ((1) EM1 ▾) ⟲ rotated angle (°)
```

Because the motors of mBot2 can measure their speed and rotation, you can also read these values. The speed measured is the rotational speed in Rounds per Minute (360° = 1 round; in science, it is more common to use "degree per second" (1 RPM = 6° per second)). In mBlock 5, if you tick the box next to this block, you can read the values on the stage above the panda. The programming example below changes the values of the rotation back to zero, and then orders the mBot2 to move forward for one second. What value do you read on the stage after the robot moved?



```
when joystick (middle pressed ▾)
reset encoder motor (all ▾) ⟲ rotated angle
(moves forward ▾) at (50) RPM for (1) secs
```

2. Recreating and testing some programming examples to control the mBot2.

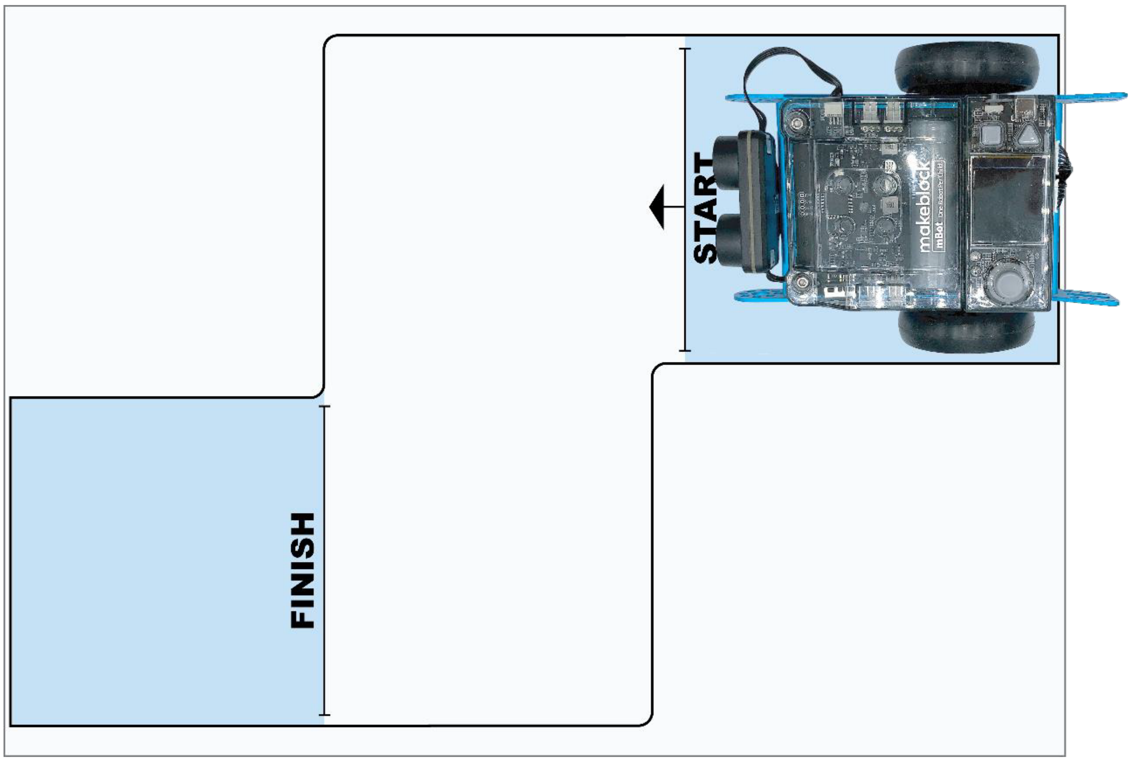
Below you can see some pairs of blocks. Try out each pair side by side. What happens? What are the differences between the code blocks?

	Option 1	Option 2
1	moves forward ▾ at 50 RPM	moves forward ▾ at 50 RPM for 1 secs
2	turns left ▾ 90 ° until done	turns left ▾ at 50 RPM for 1 secs
3	moves forward ▾ 100 cm ▾ until done	encoder motor EM1 ↺ rotates at 50 % power, encoder motor EM2 ↺ rotates at 50 % power

3. Trying out (25 min)

Step 3: Trying it out

It is now time to put your mBot2 to work. Take an A3 paper and draw a maze that your mBot2 has to drive through. Don't make it too complicated and take into account the width of the robot. The mBot2 doesn't have to find the right route by itself. The mBot2 just needs to drive the route that you program. Need inspiration? Below you can see an example.



Use the knowledge you gained in 'Step 2' of this lesson. Of course, you can do plenty of experimenting yourself with the different programming examples in mBlock5.

When doing this assignment, it is helpful to use the following step-by-step plan.

	Explanation
Step 1: What do you want to do?	• What route do you want the mBot2 to drive? • What parts does the maze consist of? • How long are the distances the robot needs to travel? • Does the robot also need to make turns? Which way and by how many degrees?
Step 2: What do you need?	• What do you need in addition to the mBot2?
Step 3: What code blocks do you need to make the mBot2 drive?	• How are you going to make the mBot2 drive? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every code block in mBlock as well.
Step 4: Testing and implementation	• Is the first version ready? Test it! During the testing round, write down areas of improvement. • Work on the improvement points until your mBot2 drives through your maze without errors.

Is it possible to get the mBot2 to drive through the maze without a hitch? Then challenge yourself with a more difficult maze. Or have one of your classmates work out a route for you.

4. Wrap-up (5 min)

Step 4: Wrap-up

Did you succeed in driving the robot through the maze completely without error?

In this lesson you have been introduced to robots in everyday life and to the mBot2. You know how to control the movements of mBot2 and which programming blocks you can use for this.

It is now time for a short reflection. Think on your own and discuss with the group:

- What do you think turned out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 2:
Sensing = data

Subject: STEAM **Grade(s):** 5th and up

Duration: 45 Minutes **Difficulty:** Beginner

Lesson Objectives

At the end of this lesson, students will be able to:

- Recognize and use the code blocks for operating the sensors
- Show the data from the sensors on the display of the mBot2
- Build your own computer program in mBlock for the mBot2 to control the sensors and show the data on the display

Overview

A robot can be a fun toy, but did you know that you can also use a robot in research? A robot has several sensors (=senses) with which it can perceive its surroundings and also collect data. For example, you can have the mBot2 measure sound and temperature. You can read the data on the screen of the mBot2 and use it for research into sound density or the temperature of the classroom, for example.

Focus

At the end of this lesson, students will know:

- What sensors the mBot2 has
- How the sensors of the mBot2 work
- What the difference is between live mode and upload mode
- How to show the data from the sensors on the display of the CyberPi

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for ChromeBook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle

Lesson plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	1. Warming up • Sensors and data in everyday life. • What is the CyberPi?
10 minutes	2. Hands-on • Getting acquainted with the sensors of the mBot2. • Extend and test some programming examples of the sensors. • Showing the data from the sensors on the display. • Difference between live mode and upload mode.
25 minutes	3. Trying out • Writing your own program for the robot.
5 minutes	4. Wrap-up • Showtime: show what you did with your robot in a fun, short movie for later discussion. • If your teacher allows, share the end result on social media with the hashtag #mBot2data • Reflection: What are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up (5 min)

Step 1: Warming up

This step consists of two parts:

1. Sensors and data in everyday life
2. What is the CyberPi?

1. Sensors and data in everyday life

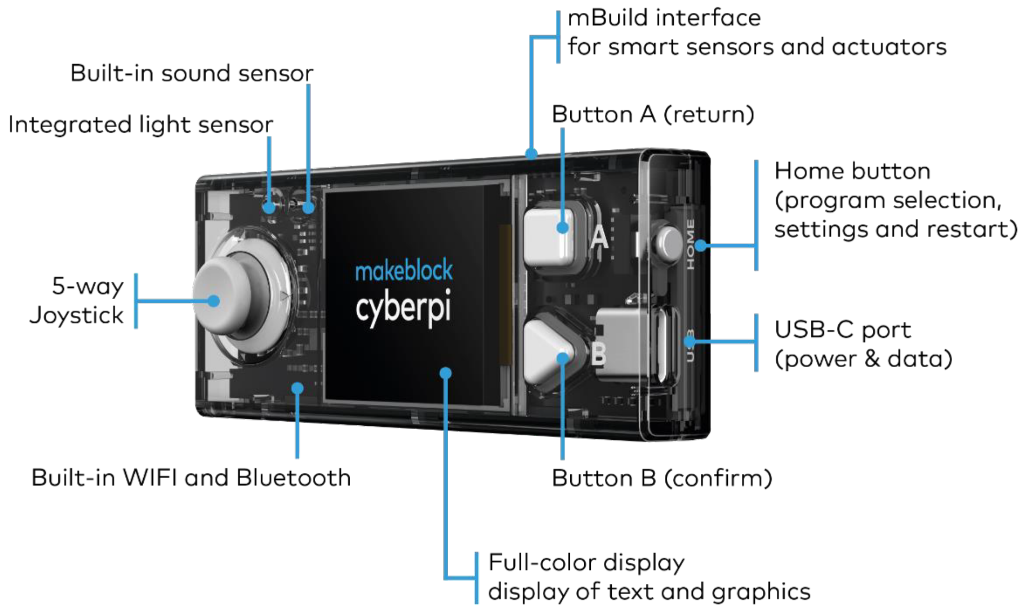
Sensors that collect data are found in many different places in everyday life. Much more than you probably think at first. For example, with the help of sensors the exterior lighting of your house is turned on automatically when it gets dark outside. Or the heating is automatically turned off when it gets too hot in the classroom. The amount of outdoor light and the heat in the classroom is all recorded using sensors. Can you and your classmates think of any other examples?

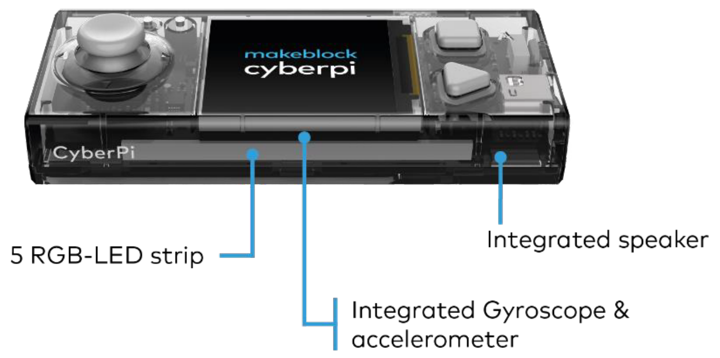
2. What is the CyberPi?

To get the mBot2 to work, you need to write a computer program. You do that in the code editor mBlock. The computer program you write consists of a series of commands that the mBot2 has to execute.

Mounted on the mBot2 there is a CyberPi. A CyberPi is a small, programmable microcomputer. You give the CyberPi commands using the code blocks in mBlock. The CyberPi passes these commands back to the mBot2. You can remove the CyberPi from the mBot2.

The CyberPi has many different functions, such as a microphone, a speaker and a joystick. There are also many sensors on the CyberPi. Take a look at the image below.





2. Hands-on (10 min)

Step 2: Hands-on

This step consists of four parts:

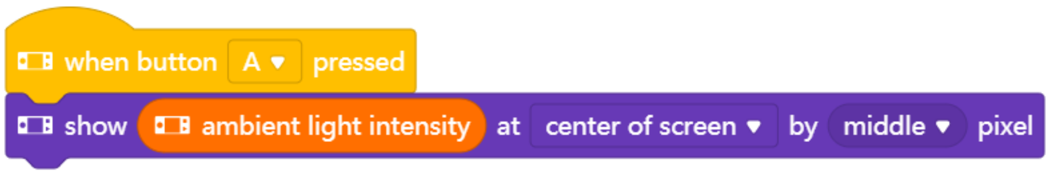
1. Getting acquainted with the sensors of the mBot2.
2. Extend and test some programming examples of the sensors.
3. Display the data from the sensors.
4. Difference between live mode and download mode.

1. Getting to know the mBot2 sensors

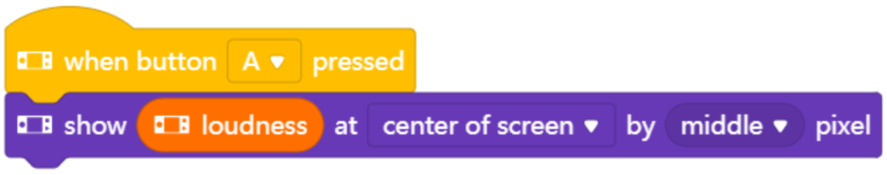
Every robot works with sensors. Sensors can be compared with your senses (=taste, touch, smell, hearing, sight). Through these sensors the mBot2 'sees' its environment. There are different types of sensors that can make the mBot2 'see', such as:

- Light sensor
- Sound sensor
- Gyroscope accelerometer
- Quad RGB sensor
- Timer

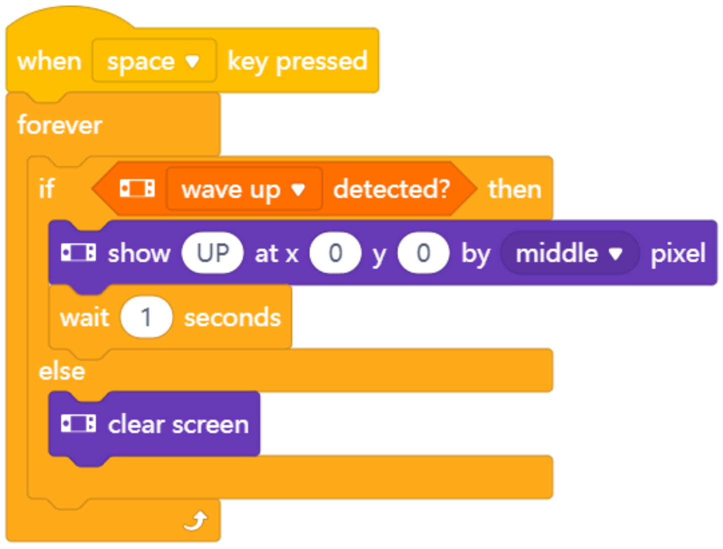
The table below lists the sensors of the mBot2. Each sensor is accompanied by a brief explanation and programming example.

Sensor type	What does this sensor do?
Light sensor	Light has a certain strength (=light energy). The light sensor is a device that converts light energy into electrical energy. A light sensor is often used, for example, to: • Automatically adjust the brightness of the display of smartphones to the environment; • To control the lighting in homes or to automatically switch on headlights of vehicles In the programming example below, the light energy of the environment is shown on the display of the CyberPi. You can extend the programming example so that the mBot2 stops driving, for example, when the light energy of the environment is below a certain value, as when it gets dark.
	Programming example
	

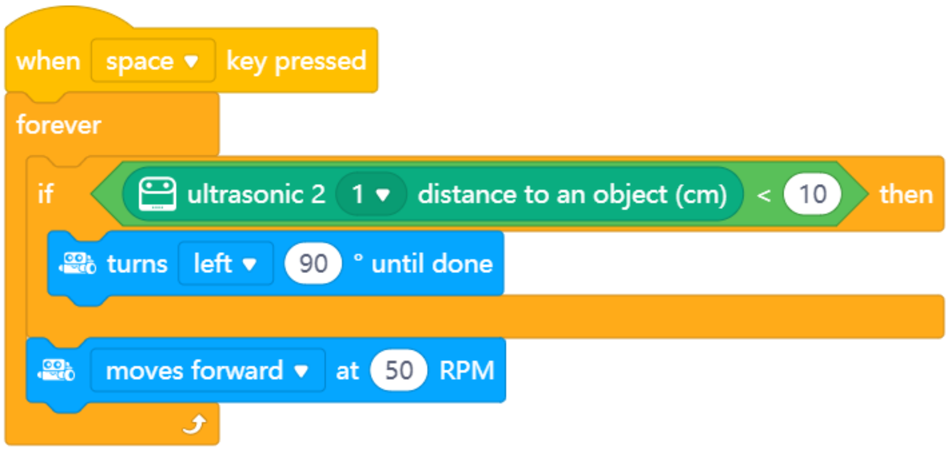
```
when button A ▾ pressed
show ambient light intensity at center of screen ▾ by middle ▾ pixel
```

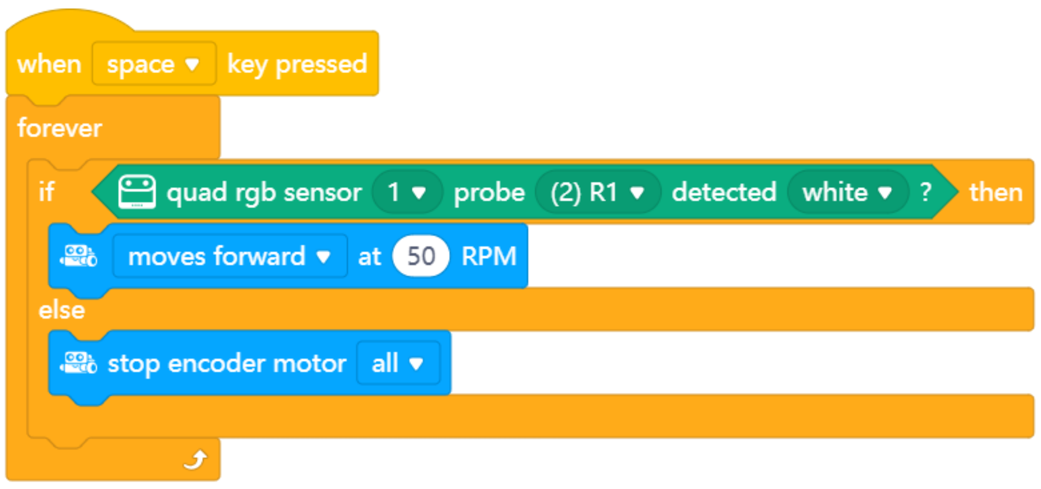
Sensor type	What does this sensor do?
Sound sensor	Sound is a mechanical vibration that propagates in waves. In the air these are pressure and density fluctuations. If the vibration is in the audible range (between 16 and 20000 vibrations per second) and sufficiently intense, we can hear it as tones or sounds. The power of the sound is called the sound intensity. The sensor in the microphone converts the sound into an electrical signal that can be evaluated in terms of pitch and power. A sound sensor is often used for the following things: • Telephone calls, sound recording • Voice-controlled assistance systems, such as on a cell phone or in a Smart Home In the programming example below, the sound intensity of the environment is shown on the display of the CyberPi. You can extend the programming example so that the mBot2 constantly measures the sound intensity during a tour of the classroom and shows it on the display.
	Programming example
	 when button A ▾ pressed show loudness at center of screen ▾ by middle ▾ pixel

Sensor type	What does this sensor do?
Gyroscope & accelerometer	A gyroscope measures tipping movements, more precisely the speed of turning/tipping movements. An accelerometer measures the change in velocity. These sensors can be implemented as micromechanical components on an electronic component. Both sensors provide different information about the position in space. A gyroscope is often used to, for example: • Keep ships stable on the high seas or • Balance a Segway or a "hoverboard" so that you don't fall over too quickly with it Acceleration sensors, in turn, indicate whether a smartphone is facing up or down, or whether a vehicle is having an accident (violent, very rapid change in speed) - in order to then trigger the airbags. In the programming example below, the tilt movement of the mBot2 is shown on the display. You can extend the programming example so that the mBot2 constantly measures and records the tipping motion on the display during a lap around the classroom. You will learn more about this sensor and its capabilities in Lesson 6.
	Programming example

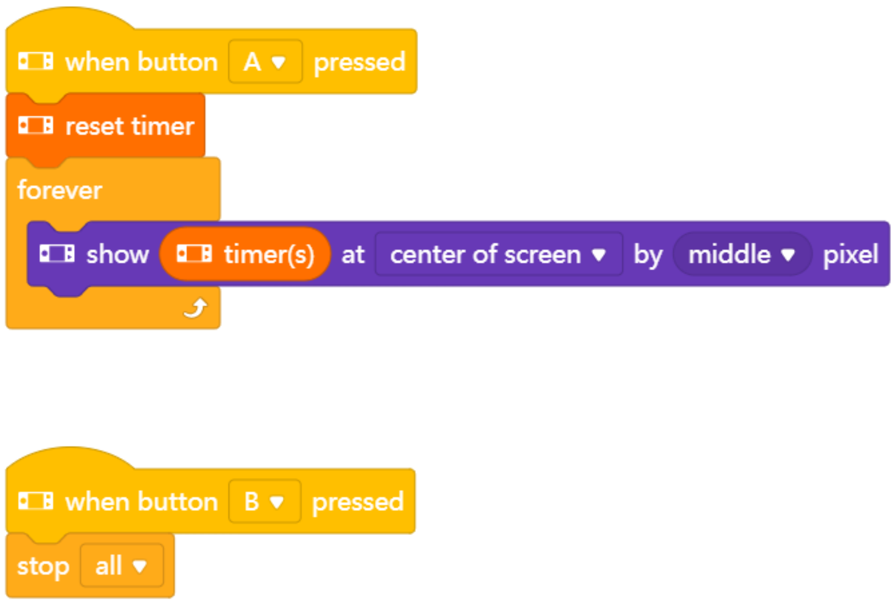
	
	when space ▾ key pressed
	forever
	  if wave up ▾ detected? then
	    show UP at x 0 y 0 by middle ▾ pixel
	    wait 1 seconds
	  else
	    clear screen

Sensor type	What does this sensor do?
Ultrasonic sensor	Sound, which is vibration in the form of variations in density and pressure, propagates in air at a constant speed in the form of waves (speed of sound in air approx. 334 m/s). The higher the vibration, the higher the sound appears - up to a limit above which people can no longer perceive this sound. These very high vibrations are called ultrasound. Since the speed of sound in the air is known, sound can be used to determine distances to objects; to do this, one emits a sound and measures the time until this sound is reflected back from the object. This reflected sound wave is also called an echo. Ultrasounds usually are used for this purpose. For example, an ultrasonic sensor is often used: • for an imaging procedure, e.g. in pregnancy (each pixel is a distance measurement) or • in the control of robots to prevent collisions In the programming example below, the ultrasonic sensor is used to prevent the mBot2 from driving into an obstacle. When the mBot2 is less than 10 cm from an obstacle, the robot makes a 90° turn to the left and then simply continues driving. You can extend this programming example so that the mBot2 moves randomly through a classroom without bumping into tables and chairs. You'll learn more about this sensor and its capabilities in Lesson 4.
	Programming example

	
	`when space ▾ key pressed`
	`forever`
	`  if ultrasonic 2 1 ▾ distance to an object (cm) < 10 then`
	`    turns left ▾ 90 ° until done`
	`  moves forward ▾ at 50 RPM`

Sensor type	**What does this sensor do?**
Quad RGB Sensor	The Quad RGB sensor consists of four individual light and color sensors. They measure the intensity of light entering the sensor from red, green and blue areas of the light spectrum. This allows the sensor to detect colors of objects directly in front of it, like markings on the ground, and also allows the robot to follow a black line for orientation. A Quad RGB sensor is widely used in applications such as: • show a warehouse robot a way through the warehouse (also different colors for different paths), or • ensure that exactly the right color is used during painting In the programming example below, the mBot2 drives forward when the robot sees the color white. You can extend the programming example so that the mBot2 drives a route through the classroom, stops, or turns based on different colors. You will learn more about this sensor and its capabilities in Lesson 5.
	Programming example
	

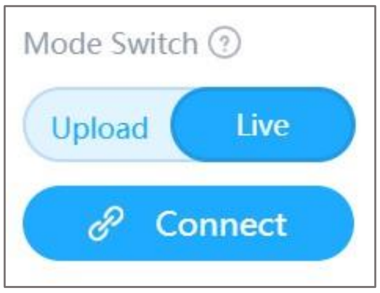
```
when [space ▾] key pressed
forever
    if <quad rgb sensor [1 ▾] probe [(2) R1 ▾] detected [white ▾] ?> then
        [moves forward ▾] at (50) RPM
    else
        stop encoder motor [all ▾]
```

Sensor type	What does this sensor do?
Timer	The timer is a kind of stopwatch that tells the time in seconds since the CyberPi was turned on or reset. This counter can also be set to zero with a command to make time measurements easier. For example, you can use the timer to have a race of several mBots against each other and measure exactly how fast they are at the finish line. The following programming example shows you how to set and use the timer. Pressing the 'A' key resets the timer to zero and then shows it permanently in the display. You can end the display and the program as a whole by pressing 'B'. Try to extend the programming example so that the timer starts counting when the mBot2 starts moving.
	Programming example
	 when button A ▾ pressed reset timer forever show timer(s) at center of screen ▾ by middle ▾ pixel when button B ▾ pressed stop all ▾

2. Testing and extending programming examples for the sensors

In the table above, there is a programming example for each sensor. You are going to recreate these programming examples in mBlock and test them. Think of an extension for two of the programming examples. The table above already mentions an extension suggestion for every programming example. Maybe you know a much nicer extension!

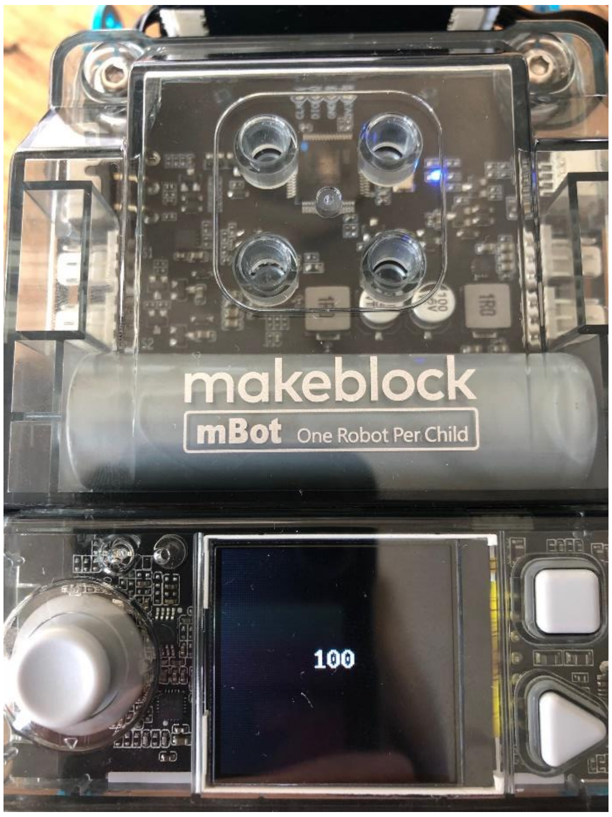
While writing a computer program, you can immediately test what you are creating. You do this by using live mode. You select live mode by moving the Mode switch to the right. Take a look at the image below.

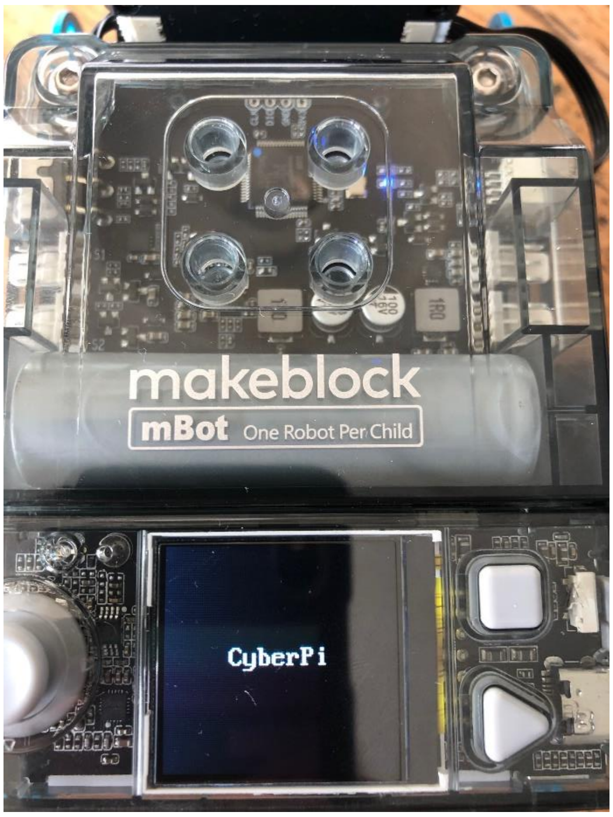


3. Displaying the data from the sensors

In testing the programming examples, you have seen that each sensor shows something on the CyberPi's display. For example, the light sensor records the light intensity of the environment. The timer registers the time, and the gyro sensor keeps track of which way the mBot2 tilts. Everything that a sensor registers is called data.

You can show the data from a sensor on the display of the mBot2. In the programming examples you have seen that you can display the data with, for instance, a number or a text.





You can display the data from the sensors in different ways on the display of the mBot2. You can use different code blocks for this. These code blocks can be found in mBlock under the category 'Display'.
In the diagram below you see three examples of how to show data on the CyberPi's display. Reproduce the programming examples and test in live mode. See what happens on the display of the mBot2.

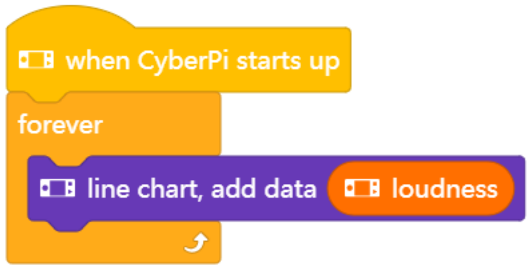
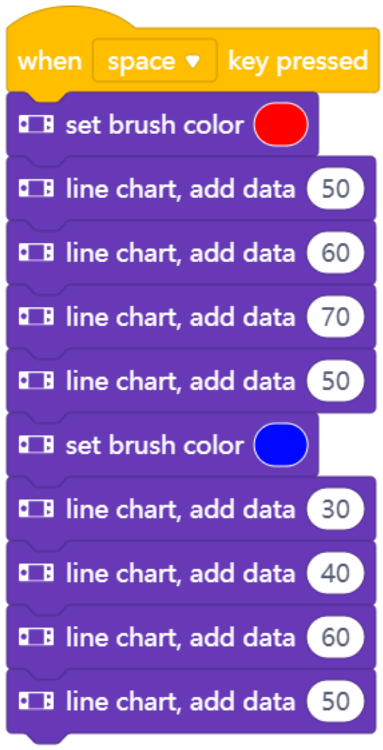
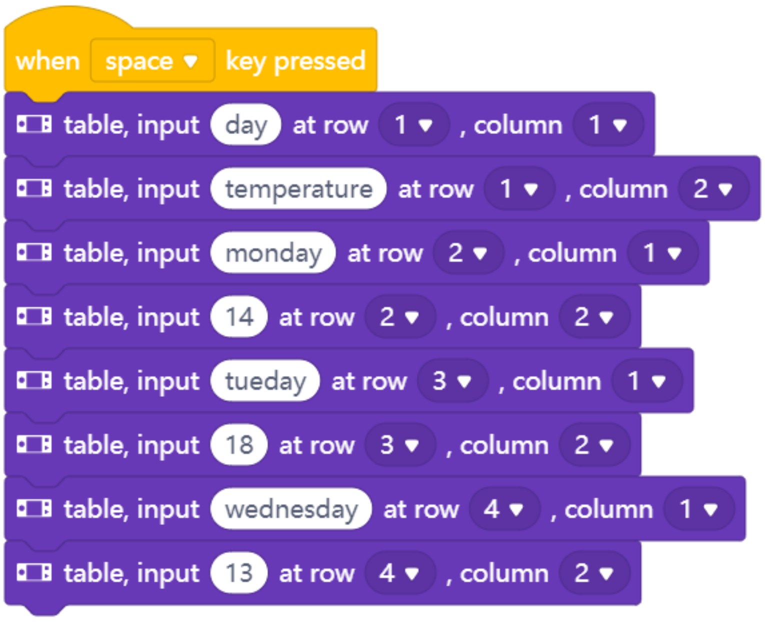
	Programming example
Line diagram	 when CyberPi starts up forever line chart, add data (loudness)
Line diagram (color)	 when (space ▾) key pressed set brush color 🟥 line chart, add data (50) line chart, add data (60) line chart, add data (70) line chart, add data (50) set brush color 🟦 line chart, add data (30) line chart, add data (40) line chart, add data (60) line chart, add data (50)

Table	 when space key pressed table, input day at row 1, column 1 table, input temperature at row 1, column 2 table, input monday at row 2, column 1 table, input 14 at row 2, column 2 table, input tueday at row 3, column 1 table, input 18 at row 3, column 2 table, input wednesday at row 4, column 1 table, input 13 at row 4, column 2

4. Live mode and upload mode

You need to transfer the computer program you build in mBlock to the CyberPi.

The CyberPi puts the mBot2 to work according to the commands you wrote in mBlock. While writing a computer program, you can directly test what you are making. You do that by using live mode. You select live mode by moving the Mode switch to the right. You must not disconnect the CyberPi from the computer while testing in live mode!

The CyberPi also has an upload mode. In upload mode, the computer program is transferred to the CyberPi. The program is stored on the CyberPi until you replace it with another program. You can simply unplug the CyberPi.

In the diagram below you can see the differences between live mode and upload mode.

Mode	Differences explained
Live mode	• Programs run on your computer (=your PC or laptop). They will not be stored on the CyberPi. • The CyberPi must remain connected to your computer. • The mBlock code editor must always be open on your computer. • This mode is used for programming in the stage.
Upload mode	• Programs run on the CyberPi instead of on your computer and are also stored on the CyberPi. • The CyberPi does not need to be connected to your computer. • The mBlock code editor does not need to be open on your computer. • A program that you upload to the CyberPi is saved on the CyberPi until you replace it with another program.

3. Trying out (25 min)

Step 3: Trying it out

You have already learned a lot about the sensors of the mBot2 and how to show the data from the sensors on the display of the CyberPi. You are now going to collect data yourself using the sensors of the mBot2 and show it on the CyberPi's display. You will do this while driving the mBot2 around the classroom or school. What data you collect and how you show it on the CyberPi's display is up to you.

Use the knowledge you gained in 'Step 2' of this lesson. Of course, you can do plenty of experimenting yourself with the different programming possibilities in mBlock.

When thinking about this assignment, it is helpful to use the following step-by-step plan. Do you have an idea of what you want to make? If so, first discuss with your teacher whether this is feasible.

	Explanation
Step 1: What do you want to do?	• What route do you want the mBot2 to drive? • What data do you want the mBot2 to collect?
Step 2: What do you need?	• What do you need in addition to the mBot2?
Step 3: What code blocks do you need to make the mBot2 drive?	• How will you make the mBot2 drive? • What data will you have the mBot2 collect? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well.
Step 4: In what way do you want to show the data from the sensors on the display?	• How do you want to show the data on the display? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well.
Step 5: Testing and implementation	• Is the first version ready? Test it! During the testing round, write down areas of improvement. • Work on the improvement points until the mBot2 does exactly what you had in mind. • Successful? Film the end result and ask your teacher if you can post it on social media with the hashtag #mbot2.

4. Wrap-up (5 min)

Step 4: Wrap-up

What data did your mBot2 collect? Did it go well? Did you get to see all the data on the display of the CyberPi?

In this lesson, you learned about the different sensors on the mBot2 and where you might encounter them in everyday life. You know how to program these sensors and how to show the data on the display. You also learned about the difference between the upload and live modes of mBlock.

It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think worked out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 3:
Listen to mBot2

Subject: STEAM **Grade(s):** 5th and up

Duration: 45 Minutes **Difficulty:** Beginner

Lesson Objectives

By the end of the lesson, students will be able to:

- Recognize and use the code blocks for controlling the speaker and microphone
- Build your own computer program in mBlock to make the mBot2 record sound and play it back
- Run multiple programming tasks side by side in a computer program

Overview

Robots can often talk. Take, for example, the vacuum cleaner robot that uses a voice message to let you know its battery is low. Or a toy robot that sings a song. The mBot2 can also produce and record sound. It has a microphone and speaker for that. With the microphone, the mBot2 records sound and the speaker plays back the recorded sound.

Focus

By the end of this lesson, students will know:

- How to make use of the speaker in coding and robotics?
- How to record and play back sound.
- How to run multiple programming tasks side by side.

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for ChromeBook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle

Lesson plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	1. Warming up • Speakers in everyday life • How does the speaker work?
10 minutes	2. Hands-on • Getting acquainted with the different code blocks of the speaker and microphone. • Reproduce, test and extend some programming examples of the speaker and microphone.
25 minutes	3. Trying out • Writing your own program for the robot.
5 minutes	4. Wrap-up • Showtime: show what you did with your robot in a fun, short movie for later discussion. • If your teacher allows, share the end result on social media with the hashtag #mBot2speaker • Reflection: What are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up
(5 min)

Step 1: Warming up

This step consists of two parts:

1. Speakers in everyday life
2. How does the speaker work?

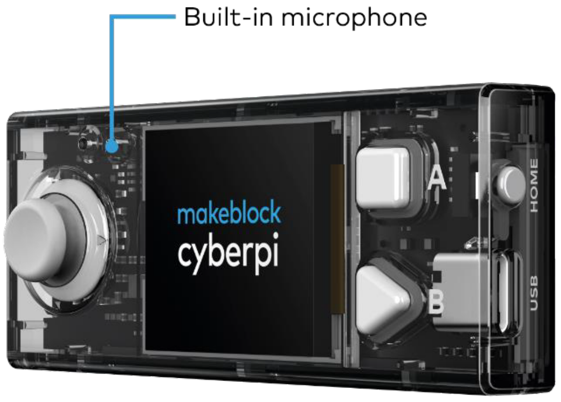
1. Speakers in everyday life

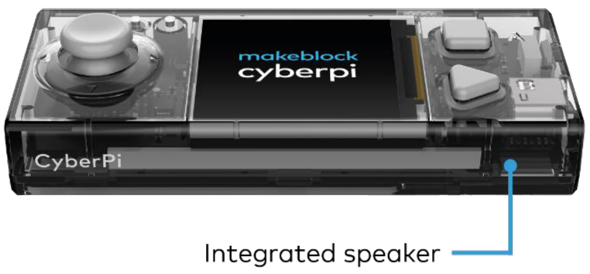
Speakers are used all around you in everyday life. You will find large speaker systems in theaters, concert halls and event halls. These speakers are used to amplify the sound for a large group of people. Smaller speaker systems are found in appliances, such as televisions, musical instruments, and smartphones. Can you and the classmates in your group think of more applications?

2. How does the speaker work?

The purpose of a speaker is to produce sound. Speakers convert electromagnetic waves into sound waves for this purpose. Next to its main purpose, a speaker can actually convert sound waves back to electromagnetic waves, too. Microphones are devices that use the same principle, but are optimized for the purpose of converting sound waves to electrical signals.

On the mBot2 there is a speaker and a microphone. The microphone is on the top left of the CyberPi. The speaker is on the front of the CyberPi. With the microphone you can record sounds. These sounds can be played back later with the speaker.





By using the microphone and the speaker in combination with the other sensors of the mBot2 you can, for example, have your own recorded text played back at a specific time. For example, think of a public bus which tells you which stop you are almost at. There are countless other possibilities for using the microphone and speaker. Together with the classmates from your group, think of more examples where the mBot2 can make good use of the speaker and microphone.

2. Hands-on
(10 min)

Step 2: Hands-on

This step consists of two parts:

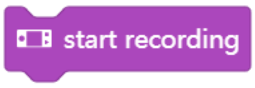
1. Getting acquainted with the different code blocks for the speaker and microphone.
2. Reproduce, test and extend some programming examples of the speaker.

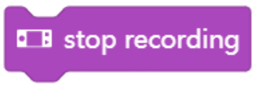
1. Getting to know the different code blocks for the speaker and microphone

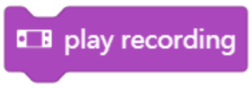
In mBlock 5, there are several code blocks that you can use to program the speaker and the microphone. You will find these code blocks in the 'Audio' category of the block area in mBlock. These code blocks are purple.

In the table below you will see some of these code blocks to program the speaker and microphone.

Code block:

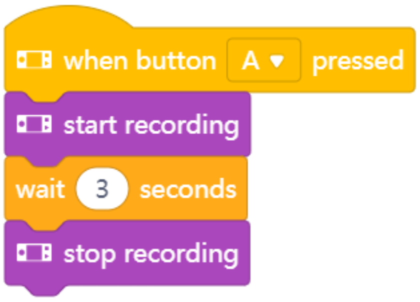

start recording


stop recording

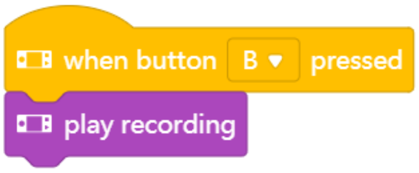

play recording

When recording and playing back a sound, you actually always use three code blocks. With the code block 'Start recording' you let CyberPi start recording a sound. The electrical signal from the microphone is stored in the memory of the CyberPi, and the recording automatically stops after 10 seconds. You can also indicate when the recording should stop. This is done with the code block 'Stop recording'. The recorded sound is temporarily stored in the memory of the CyberPi. Do you switch off the mBot2? Then the recorded sound is lost. Playback of the recorded sound is controlled by the 'Play recording' code block.

In the programming example below, when you press button A, CyberPi starts recording. After 3 seconds the recording stops. When you press button B, you hear what was recorded. By using two event blocks, two programming tasks can run independently from each other without the need for a main loop.

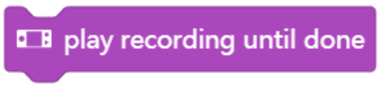


```
when button A ▾ pressed
start recording
wait 3 seconds
stop recording
```



```
when button B ▾ pressed
play recording
```

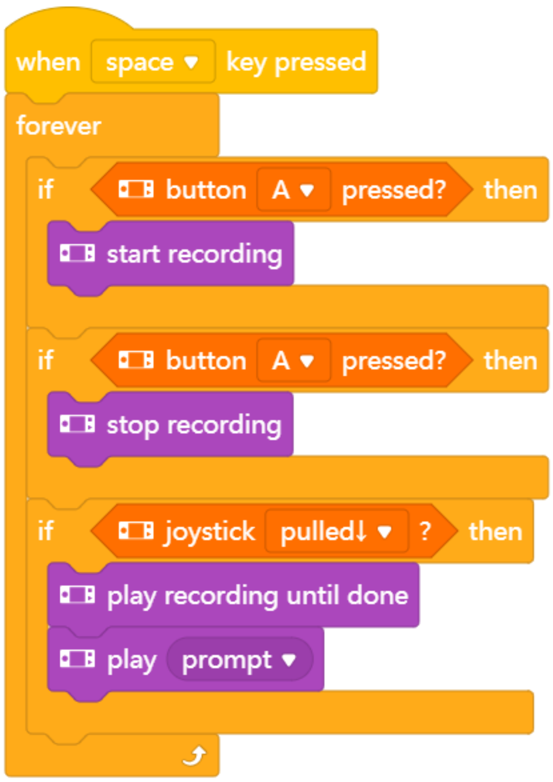
Code block:



```
play recording until done
```

You just learned how to playback a recorded sound. With this code block you make sure that the sound recorded by the microphone is played all the way through. This means that the execution of further blocks is blocked until this one is finished. After the sound is finished you can make the mBot2 do something else. For example, make a certain sound so you know the recording is finished.

In the programming example to the right, the CyberPi starts recording when you press button A. If you press button B, the recording stops. If you move the joystick down, you will hear what the mBot2 recorded. After the entire recording ends, you will hear a "prompt" sound. The code blocks in this example are placed in a so-called loop structure. This ensures that the input from the buttons and joystick are continuously monitored.



```
when space ▾ key pressed
forever
    if button A ▾ pressed? then
        start recording
    if button A ▾ pressed? then
        stop recording
    if joystick pulled↓ ▾ ? then
        play recording until done
        play prompt ▾
```

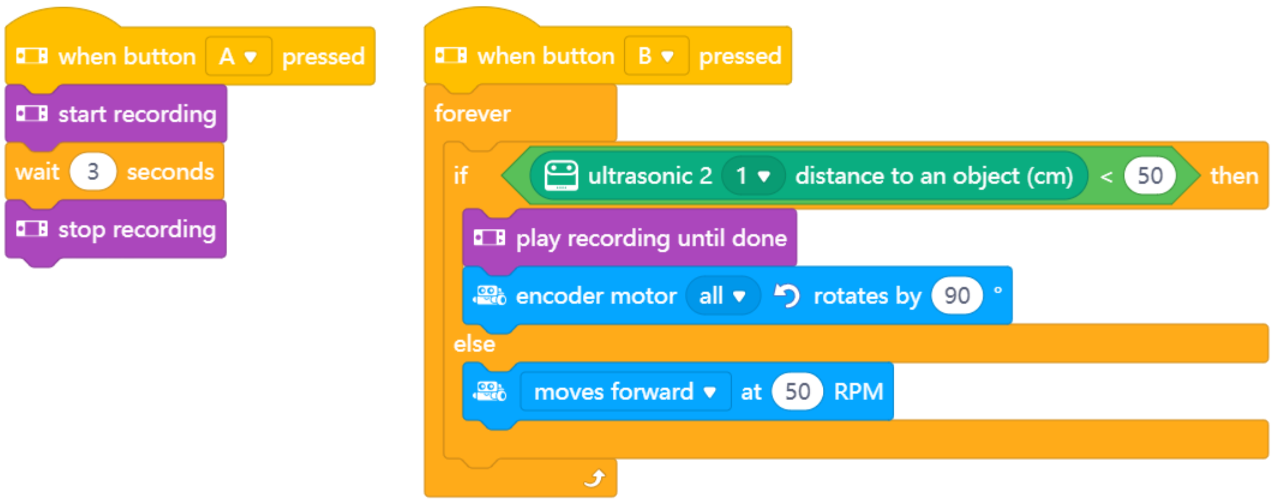
2. Reproducing, testing and extending some programming examples from the speaker
In the table above, each code block of the speaker is accompanied by a programming example. You are going to recreate these programming examples in mBlock and test them. Think of one extension to a programming example. Test the programming examples and your own extension in live mode.

3. Trying out (25 min)

Step 3: Trying it out
You've already learned a lot about the speaker and microphone on the mBot2. You are now going to work on them by yourself. You are also going to use the ultrasonic sensor. This sensor can sense an object from a distance. You will learn more about the ultrasonic sensor in lesson 4.
In this assignment you are going to make the mBot2 drive a random lap around the classroom. Random means that you don't predetermine a route. If the mBot2 comes near an obstacle, such as a chair, table or backpack, you will hear a sound and the mBot2 will turn around. You will record the sound yourself with the microphone. Such a sound could be, for example, a loud beeping sound or the sound of a car slowing down. You can also record a text or have the mBot2 play a song.
It's quite difficult to come up with a computer program for this all by yourself! Fortunately, you can get help. On the next page you see two examples of a computer program. You can expand and adapt these computer programs by yourself. Be creative!

Example 1

The mBot2 makes a sound when it approaches an obstacle and turns around. You recorded the sound yourself earlier.

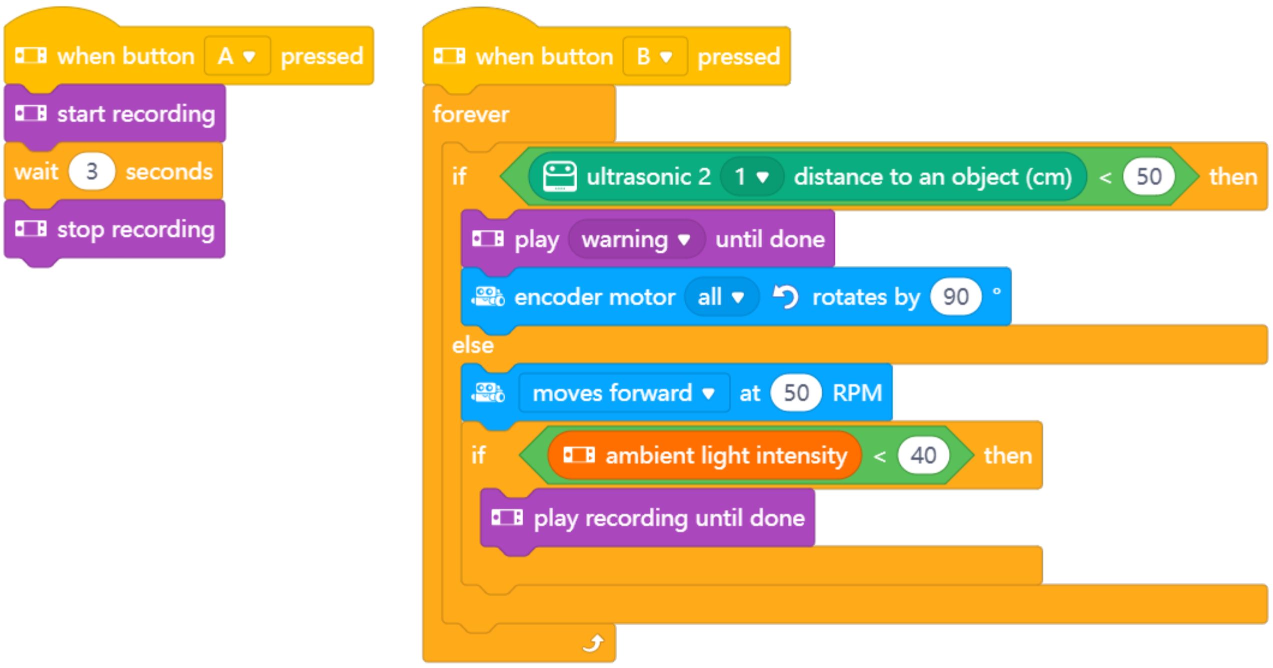


```
when button A pressed
start recording
wait 3 seconds
stop recording
```

```
when button B pressed
forever
    if ultrasonic 2 1 distance to an object (cm) < 50 then
        play recording until done
        encoder motor all rotates by 90 °
    else
        moves forward at 50 RPM
```

Example 2

The mBot2 drives around and plays a sound you recorded when the ambient light is low. When the mBot2 comes near an obstacle, it lets out a warning sound and turns around.



```
when button A pressed
start recording
wait 3 seconds
stop recording
```

```
when button B pressed
forever
    if ultrasonic 2 1 distance to an object (cm) < 50 then
        play warning until done
        encoder motor all rotates by 90 °
    else
        moves forward at 50 RPM
        if ambient light intensity < 40 then
            play recording until done
```

Use the knowledge you gained in 'Step 2' of this lesson. Of course, you can do plenty of experimenting yourself with the different programming possibilities in mBlock.
When thinking about this assignment, it is helpful to use the following step-by-step plan. Do you have an idea of what you want to make? If so, first discuss with your teacher whether this is feasible.

	Explanation
Step 1: What do you want to do?	• When should the mBot2 turn around? • What sound should the mBot2 make when it gets near an obstacle?
Step 2: What do you need?	• What do you need in addition to the mBot2?
Step 3: What code blocks do you need to make the mBot2 drive?	• How are you going to make the mBot2 drive? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well.
Step 4: In what way do you want to show the data from the sensors on the display?	• What sound should the mBot2 make? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well.

Step 5: Testing and implementation	• Is the first version ready? Test it! During the testing round, write down areas of improvement. • Work on the improvement points until your mBot2 does exactly what you had in mind. • Successful? Film the end result and ask your teacher if you can post it on social media with the hashtag #mBot2

4. Wrap up (5 min)

Step 4: Wrap up

How did this assignment go? Did the mBot2 immediately do what you had in mind?
In this lesson, you learned how speakers work and where you encounter them in everyday life. You know how to program the CyberPi and the mBot to record via the built-in microphone and play back through the speakers. You also know how to run multiple programs independently from each other.
It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think turned out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 4:
Seeing with sound

Subject: STEAM
Grade(s): 5th and up
Duration: 45 Minutes
Difficulty: Beginner

Lesson Objectives

By the end of this lesson, students will know:

- What ultrasound is and how a sensor based on ultrasound works
- Applications of these sensors in everyday live and in robotics

Overview

Autonomous robots need to be aware of their surroundings when driving unattended to avoid any collisions with obstacles or persons. This also applies to the mBot2. The mBot2 has a specific sensor for this at the front. It is called the ultrasonic sensor and it enables the robot to detect objects in its path using sounds that humans can't hear – giving it the name (ultrasound).

Focus

By the end of this lesson, students will be able to:

- Use an ultrasonic sensor for range and obstacle detection
- Make the mBot2 react to obstacles and avoid them in driving

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for ChromeBook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle
- Small obstacles (minimum size is 10x10x10 cm)
- A workspace of at least 1m^2 where the mBot2 can drive

Lesson plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	1. Warming up • Ultrasonic sensors in everyday life • How does an ultrasonic sensor work?
10 minutes	2. Hands-on • Getting to know the different code blocks of the ultrasonic sensor. • Recreating and testing some programming examples of the ultrasonic sensor.
25 minutes	3. Trying out • Writing your own program for the robot.
5 minutes	4. Wrap-up • Showtime: show what you did with your robot in a fun, short movie for later discussion. • If your teacher allows, share the end result on social media with the hashtag #mBot2 • Reflection: What are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up (5 min)

Step 1: Warming up

This step consists of two parts:

1. Ultrasonic sensors in everyday life
2. How does the ultrasonic sensor work?

1. Ultrasonic sensors in everyday life

Ultrasound is sound with a very high frequency. So high, that people cannot hear it. With special microphones this sound can be recorded and displayed on a computer: it seems like vibrations in the form of a wave. Since ultrasound has a very high frequency, the waveform shows valleys and peaks very close together. Ultrasound can be a useful thing. For example, you can use the vibrations of the ultrasound to clean objects, or you can also use it to detect objects.

Dolphins and bats use ultrasound this way. They emit an ultrasonic sound, like a short scream. When this sound bounces off something they catch with their ears the vibrations (echo) that come back. This is how they know where other animals are, for example, or if an obstacle is in their way. The radar we have today uses the same principle, but with radio waves instead of sound.

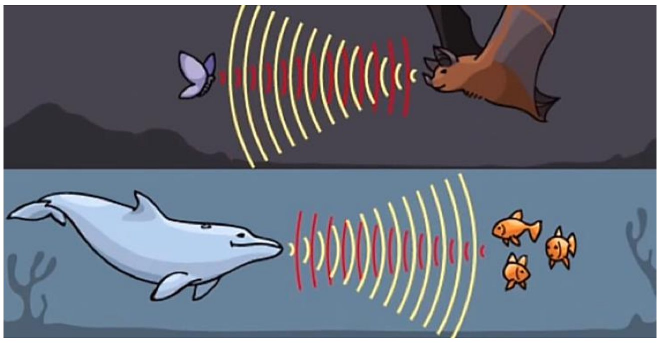
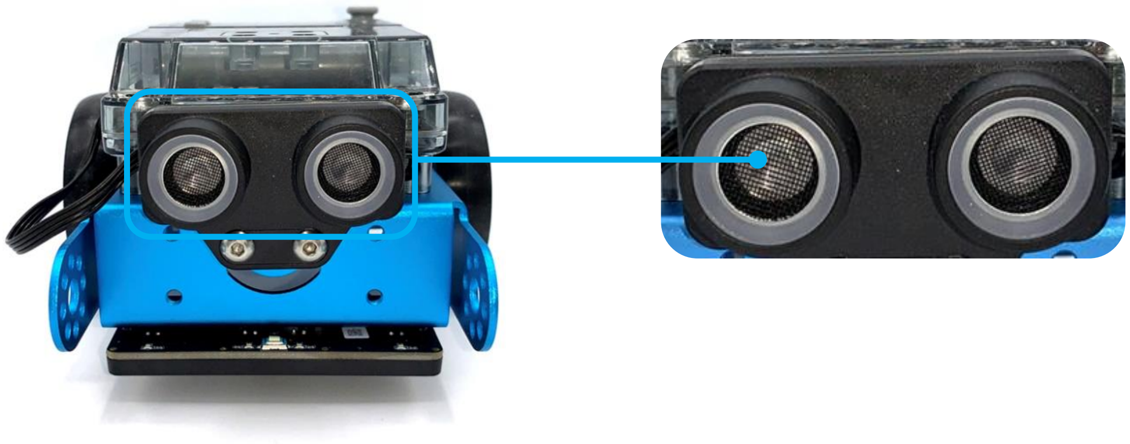


Figure 2. Example of bat and dolphin

2. How does the ultrasonic sensor work?

The mBot2 has an ultrasonic transmitter and a receiver. These are located on the front of the mBot2 in the little cylinders that can be considered as "eyes". This sensor sends out short bursts of ultrasonic sounds and listens for any echo. Does the mBot2 come close to an obstacle? Then the ultrasonic sound is reflected back to the mBot2. Based on the time it takes for the sound to return to the sensor, the mBot2 internally calculates the distance to it.

This data can be used to decide how the robot should react: you tell it to the robot by programming it. Such an action could be, for example, that the mBot2 should stop driving or make a turn.



2. Hands-on (10 min)

Step 2: Hands-on

This step consists of two parts:

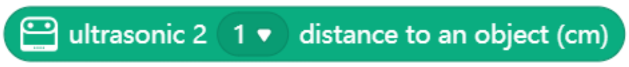
1. Getting acquainted with the different code blocks of the ultrasonic sensor
2. Recreating and testing some programming examples of the ultrasonic sensor.

1. Getting acquainted with the different code blocks of the ultrasonic sensor

In mBlock, there are several code blocks for the ultrasonic sensor you can use in your programs. For this, you need to add the extension 'Ultrasonic Sensor 2' from the extension library. Then you will find the code blocks in the 'Ultrasonic Sensor' category of the blocks field in mBlock. These code blocks are green.

In the table below, you can see some examples of the code blocks to program the ultrasonic sensor.

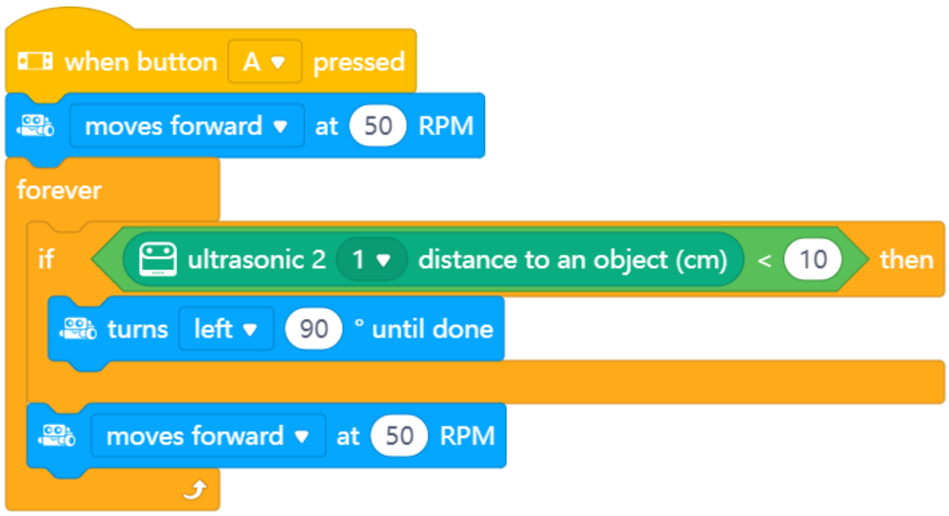
Code block:



```
ultrasonic 2 [1 ▾] distance to an object (cm)
```

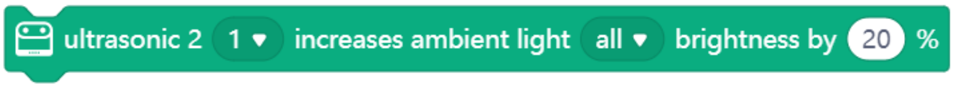
With this code block you measure the distance between the sensor and an obstacle. The sensing range is between 3 and 300 cm. You can use the value of the distance to make the robot perform a certain action. For example, you can prevent the mBot2 from colliding with obstacles.

In the programming example below, the mBot2 drives forward. When the mBot2 is less than 10 cm from an obstacle, the robot makes a 90 degree turn to the left. Then the mBot2 drives forward again.



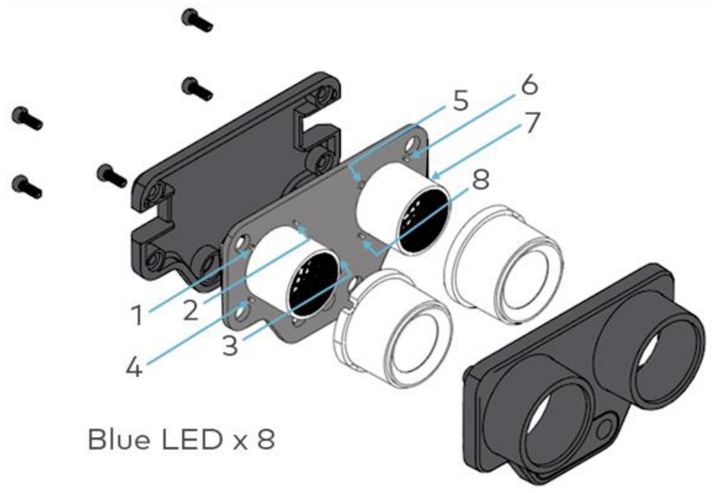
```
when button [A ▾] pressed
moves forward ▾ at (50) RPM
forever
    if <(ultrasonic 2 [1 ▾] distance to an object (cm)) < (10)> then
        turns [left ▾] (90) ° until done
    moves forward ▾ at (50) RPM
```

Code block:

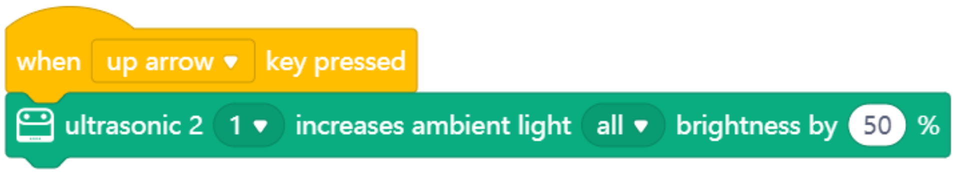


```
ultrasonic 2 [1 ▾] increases ambient light [all ▾] brightness by (20) %
```

The Ultrasonic sensor on mBot2 has eight blue LED lights. You can use these lights to have the mBot2 show a certain "emotion", for example, or as a general visual non-verbal communication. Make the LED lights shine very brightly when the mBot2 is happy and less brightly when it is gloomy. There is a specific code block to show pre-defined "emotions", too. In the image below you can see where exactly the LED lights are located.



In mBlock there are several code blocks that allow you to program the brightness of the LED lights: you can either set the brightness of the LEDs by decreasing or increasing the brightness by a certain percentage. Or you can set the brightness directly to a specific percentage. The range is 0% to 100%. Both can be done either for a single LED or for all at the same time. In the programming example below, the brightness of the LED lamp 1 is increased by 50%.



```
when [up arrow ▾] key pressed
ultrasonic 2 [1 ▾] increases ambient light [all ▾] brightness by (50) %
```

2. Recreating and testing some programming examples of the ultrasonic sensor

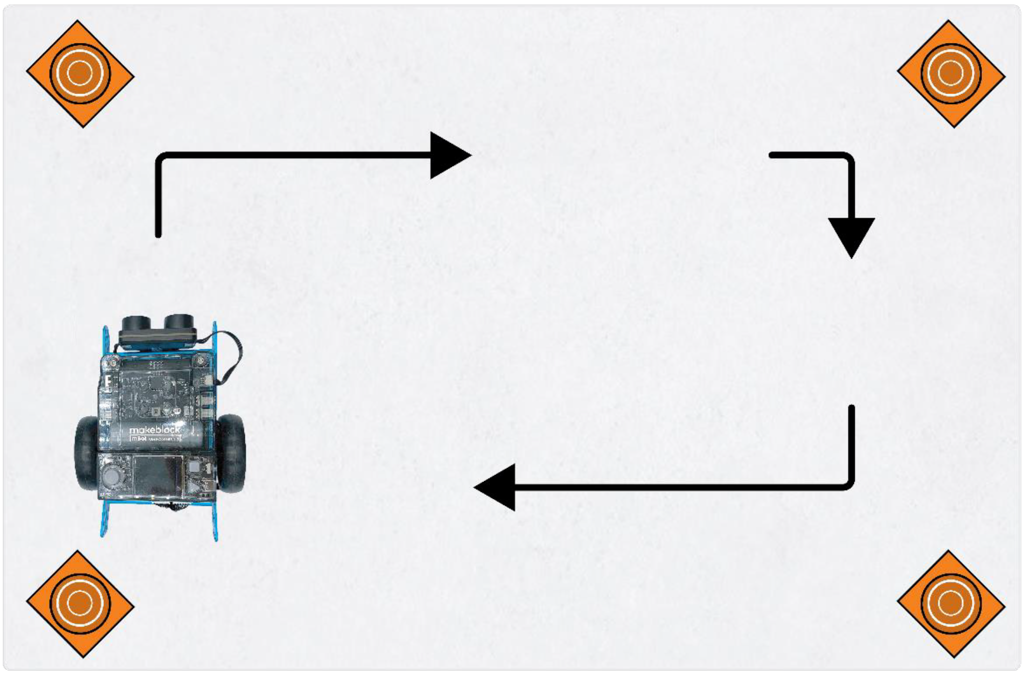
In the table above, each code block of the ultrasonic sensor has a programming example. You are going to recreate these programming examples in mBlock and test them. For one programming example, come up with your own ideas on how to extend the application.

3. Trying out
(25 min)

Step 3: Trying it out

You have already learned a lot about the ultrasonic sensor on the mBot2. You are now going to create your own computer program in mBlock using the ultrasonic sensor.

In this assignment you are going to make the mBot2 drive in infinite 'rounds'. You do this by placing an object at each vertex. You program the mBot2 in such a way that it makes a turn at each object and eventually returns to the starting point. Take a look at the image below.



Use the knowledge you gained in 'Step 2' of this lesson. Of course, you can also experiment with the different programming possibilities in mBlock.

In working out this assignment, it is helpful to use the following step-by-step plan. Do you have an idea of what you want to make? If so, first discuss with your teacher whether this is feasible.

	Explanation
Step 1: What do you want to do?	• In what way should the mBot2 move?
Step 2: What do you need?	• What do you need in addition to the mBot2?

Step 3: What code blocks do you need to make the mBot2 move?	• How can you make the mBot2 change direction at each corner? • What code blocks do you need to do this? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well
Step 4: Testing and implementation	• Is the first version ready? Test it! During the testing round, write down points of improvement • Work on the improvement points until your mBot2 runs error-free over your course • Successful? Film the end result and ask your teacher if you can post it on social media with the hashtag #mBot2

4. Wrap-up (5 min)

Step 4: Wrap-up

Did you manage to get the mBot2 to drive one round?

In this lesson, you learned about ultrasonic sensors and where you might encounter them in everyday life. You know how to program the ultrasonic sensor of the mBot2, for example, to prevent the mBot2 from driving into obstacles. You also know how to use the LED lights in the Ultrasonic sensor to make the mBot2 show a certain emotion.

It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think turned out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

makeblock education

ROBOTIX Hands-on Learning

Lesson 5: Sightseeing

Subject: STEAM

Grade(s): 5th and up

Duration: 45 minutes

Difficulty: Beginner

Lesson objectives

By the end of this lesson, students will know:

- How color sensors work
- Their applications in everyday life and in robotics

Overview

Imagine being in a foreign city, taking a tour in a coach along the most beautiful sight. Perhaps you've made such a trip yourself, and if not, you probably know the pictures from TV and the Internet. In this lesson the mBot2 is transformed into such a touring car.

Because it would be very boring for a driver to drive the same route every day (and explain every sight over and over again), we are going to program this coach. This way people can enjoy a nice ride without the driver getting bored on repetitive tasks.

Focus

At the end of this lesson, students will be able to:

- Make the mBot2 follow a line
- Have the mBot2 perform actions based on colors

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for Chromebook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle

Lesson plan

This lesson consists of four steps and takes a total of 45 minutes to complete.

Duration	Contents
5 minutes	1. Warming up • Color sensors in everyday life • How do color sensors work? What is specific about the mBot2's color sensor?
10 minutes	2. Hands-on • Becoming familiar with the different code blocks of the Quad RGB sensor • Recreating and testing programming examples of the Quad RGB sensor • How can the mBot2 follow a line?
25 minutes	3. Trying out • Writing your own program for the robot
5 minutes	4. Wrap-up • Showtime: show what you did with your robot in a fun, short movie for later discussion • If your teacher allows, share the end result on social media with the hashtag #mBot2 • Reflection: What are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up
(5 min)

Step 1: Warming up

This step consists of two parts:

1. Color sensors in everyday life
2. How do color sensors work? What is specific about the mBot2's color sensor?

1. Color sensors in everyday life

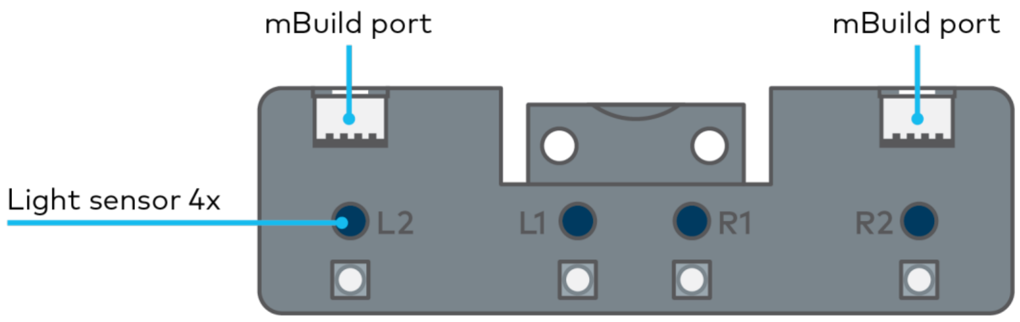
Color sensors can detect light intensity and colors. You will encounter this type of sensor in many different places in everyday life.

For example, a warehouse robot can find the correct route through a warehouse based on different colors on the floor. This kind of sensor is also used if precise color-matching is required, e.g., in vehicle coloring. A measuring device with a color sensor, the color meter, can check if the right color is used in spray-painting a vehicle. Can you and the classmates from your group think of any other examples?

2. How do color sensors work? What is specific about the mBot2's color sensor?

A color sensor consists of three different (internal) sensors that measure the light reflected by an object based on the intensity of the Red, Green and Blue color values. Color sensors are therefore called "RGB-sensors" as well. Any color is decoded into three values by the sensor – so a specific "mix" of these values represents a color.

The mBot2 has four of these RGB-sensors integrated into a single sensor. Take a look at the image below.



This is the Quad RGB sensor that is on the mBot2. The RGB-sensors are named L1, L2, R1, and R2 (L for left side, R for right side). They automatically detect the RGB values of the reflected color and compare the mix of values internally with different pre-sets for colors. This will make coding much easier, because the sensor can then report the color and you don't have to check the RGB-codes yourself. The sensor can detect 6 different colors plus black and white. Instead of color it can report on shades of grey or lines and junctions.
The Quad-RGB-sensors allows the mBot2 to follow a line on the floor and react to different color markers. The Quad RGB sensor is placed on the front of the mBot2. At the bottom face you will see the four sensors. At the topyou can find the names of each one, so you can identify and program the sensors in a specific way. Take a look at the images below.

Quad RGB sensor, front view

Top view

Bottom view

The little button on the top side of the sensor is used for calibration. If you double-click it, the LEDs will start flashing and you can "swipe" the mBot2 with the sensor across the line to be followed. This will calibrate the sensor to differentiate the line from the background. If you are using a black line on a white surface, this is normally not needed.
The map provided with the mBot2 has colors marked inside the track itself to react to the color codes. This might cause the sensor to misunderstand a bright color (like yellow) for the background instead of the track. So as a first exercise, please calibrate the sensor to the yellow color code, it will then automatically recognize reliably this and all "darker" colors as the line.
When you program the mBot2, you will use one or more of these four sensors to check for certain colors on the floor along the way. You can have the mBot2 perform a command when a color is spotted. For example, 'stop' when red and "go left" when green.

2. Hands-on
(10 min)

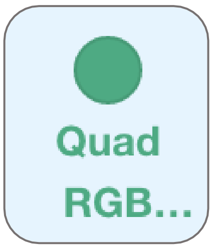
Step 2: Hands-on

This step consists of three parts:

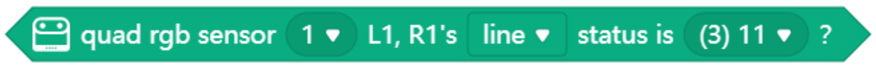
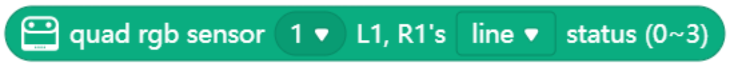
1. Getting acquainted with the different code blocks of the Quad RGB sensor
2. Reproducing and testing some programming examples of the Quad RGB sensor
3. How can the mBot2 follow a line?

1. Getting acquainted with the different code blocks of the Quad RGB sensor

In mBlock, there are several code blocks that you can use to program the Quad RGB sensor. You can find these code blocks in the 'Quad RGB' category of the block field in mBlock. These code blocks are green.



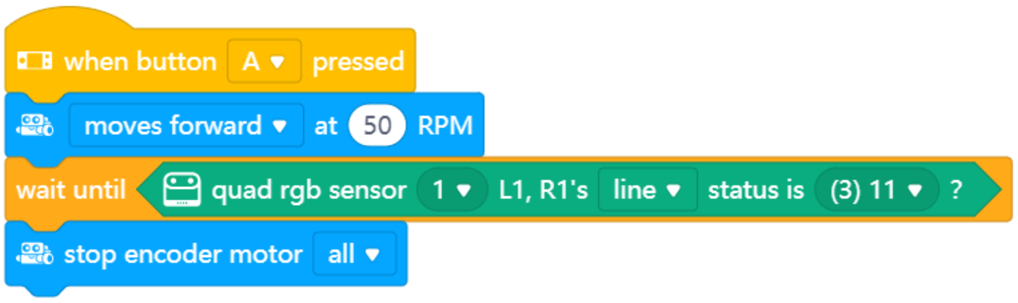
In the table below, you can see some of the code blocks you can use in your program with the Quad RGB sensor. The first four code blocks are for (simple) line-identification. They can be used for simple and advanced line following, but also for detecting crossings and sharp (90°) turns. The calibration of the sensor to different color tracks is briefly mentioned in the section above. The blue LEDs on the top side of the sensor indicate the line/background detection: if they are off, they sense dark color (line), if they are shining blue, they indicate bright (background) color. The status of the LED therefore indicates the reflectivity of the floor or map.

Code blocks:
 quad rgb sensor 1 ▾ L1, R1's line ▾ status is (3) 11 ▾ ?    quad rgb sensor 1 ▾ L1, R1's line ▾ status (0~3)   These code blocks take only the inner two sensors L1 and R1 into consideration for line detection. These blocks are intended for simple line-following.   Use the first block in conditional statements. Use the second block if you want to report the value to the screen or save it in a variable.

If the sensor is above a black line, both inner LEDs should be off. Since each sensor (L1, R1) can either detect the line or the background with this block, there are 4 possible combinations: they will be represented by numbers from 0-3 (for reference, the block shows the binary pattern as well):

L1	R1	Meaning	Status (Decimal)
🟦	🟦	Sensor on white background, both LEDs are on	0
🟦	⬛	R1 detects a black line	1
⬛	🟦	L1 detects a black line	2
⬛	⬛	Both inner sensors detect the black line	3

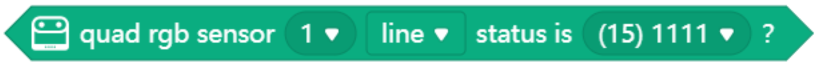
Remember, the LEDs show the background state (on or logical "true" = background detected), while this code block detects the line color (logical "true" = line detected).
If you wanted to use more than one Quad-RGB-sensor, you can tell the program which sensor to use with the first number in all the following code blocks. Take a look at the image below. Number 1 indicates the first connected sensor, number 2 the second connected sensor and so on.
The following example will use the line detection function of this block and stop the robot as soon as it arrives at a black line. This example will be changed later to line-following.



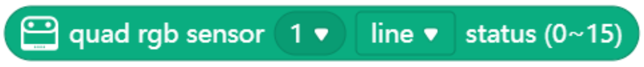
```
when button A ▾ pressed
moves forward ▾ at 50 RPM
wait until quad rgb sensor 1 ▾ L1, R1's line ▾ status is (3) 11 ▾ ?
stop encoder motor all ▾
```

Since reacting while driving can be time-critical, it is important to use the Upload mode. You can try by yourself and see the difference if you switch modes.

Code blocks:



```
quad rgb sensor 1 ▾ line ▾ status is (15) 1111 ▾ ?
```



```
quad rgb sensor 1 ▾ line ▾ status (0~15)
```

Use the first block in conditional statements. Use the second block if you want to report the value to the screen or save it in a variable.

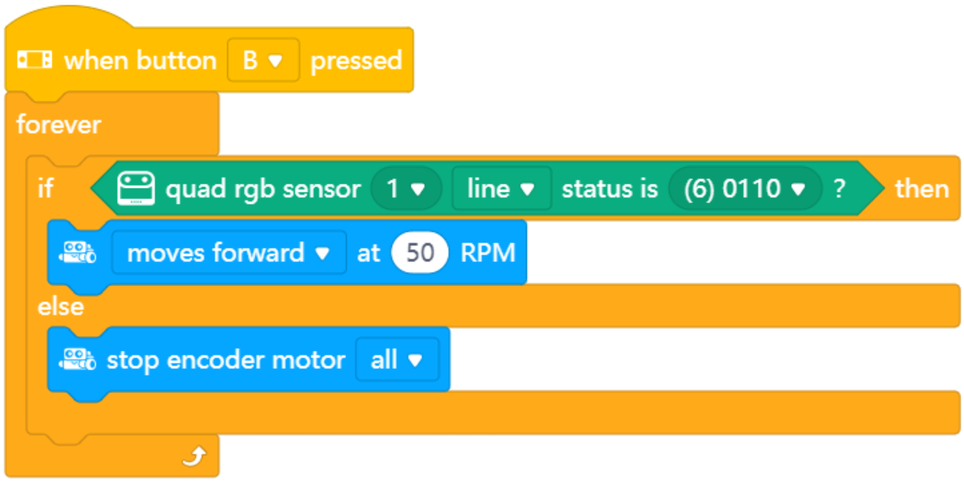
These blocks follow the exact same logic as the previous ones, except they have a larger detection are and can identify crossings while still doing line-following. Remember, the blocks check for the line status, so if one of the four sensors identifies a (dark) line, the corresponding binary digit will be set to "1" (logic true), but the LEDs will go off (no reflection on this sensor from background).

With now 4 individual sensors, the range increases to 16 different statuses. The following table presents the most relevant statuses:

Binary pattern	Meaning	Status (Decimal)
0000	all 4 sensors on white background (no track)	0
0010	only R1 on black line	2
0100	only L1 on black line	4
0110	inner sensors R1 and L1 on black line	6
0111	sharp turn right (L1 and R1 on black line, junction to the right; L-shape junction to the right)	7
1110	sharp turn left (L1 and R1 on black line, junction to the left; L-shape junction to the left)	14
1111	T-shape junction (L1 and R1 on black line, junction to both right and left)	15

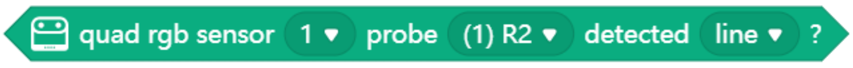
An X-shape junction can only be detected by driving over the T-Junction and checking if the middle line continues.

The following example program will make the robot drive as long as it is on a black line with the inner two sensors. It will stop if it goes off track, a junction is detected, or the line suddenly ends:



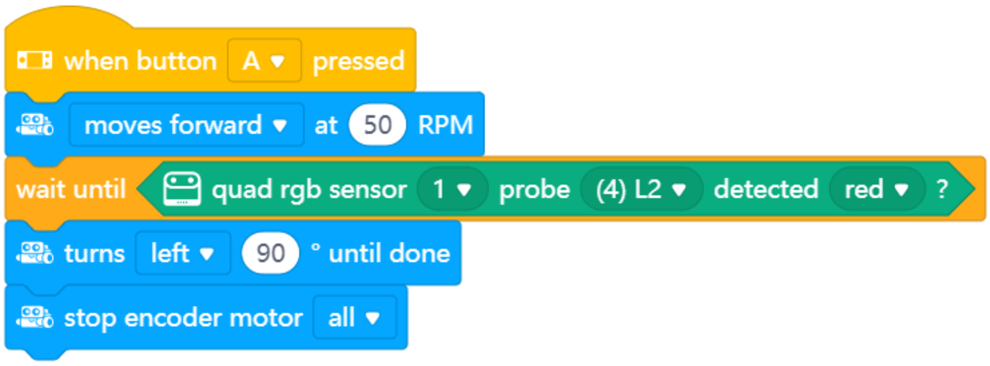
```
when button B ▾ pressed
forever
    if quad rgb sensor 1 ▾ line ▾ status is (6) 0110 ▾ ? then
        moves forward ▾ at 50 RPM
    else
        stop encoder motor all ▾
```

Code block:



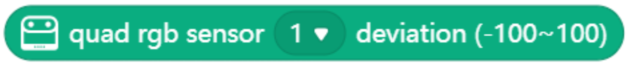
```
quad rgb sensor 1 ▾ probe (1) R2 ▾ detected line ▾ ?
```

With this code block you select one of the Quad RGB sensors (L2, L1, R1, R2) and determine whether it should recognize a line, background, white, black or any other of the 5 colors (red, green, blue, yellow, cyan, purple). For line-detection and following, the previous blocks are recommended, because they check more than one sensor at the time (faster, more accurate since the robot does not move between readings). But you can use this block to detect for example a red color marker on the left side of the robot – while the robot is following a line. In the following programming example, if you press the B-button, the mBot2 drives straight (assuming it follows a line) and does a 90° turn when a red marker is detected on the left side of the (imaginary) track.



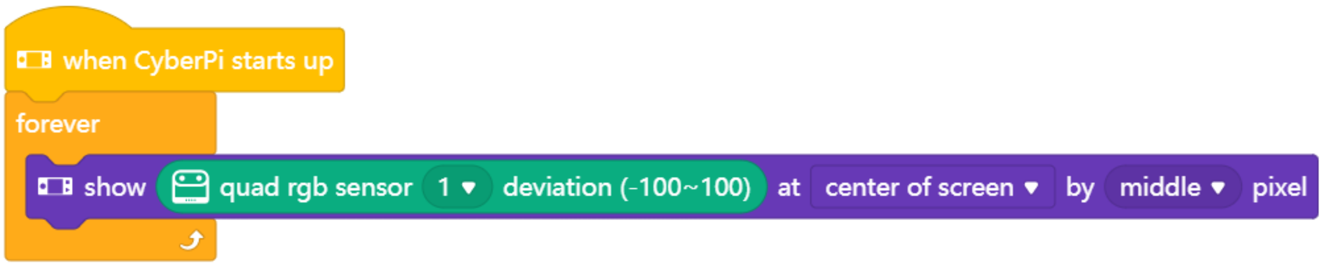
```
when button A ▾ pressed
moves forward ▾ at 50 RPM
wait until quad rgb sensor 1 ▾ probe (4) L2 ▾ detected red ▾ ?
turns left ▾ 90 ° until done
stop encoder motor all ▾
```

Code block:


quad rgb sensor 1 ▾ deviation (-100~100)

This code block measures how far the robot is off from a black track. You can use it for advanced and smooth line following. The previous line following code block uses a triple distinction: on track, left off or right off (plus added junction detection in some cases). But in everyday life riding a bicycle, you would adjust the steering to the curve more precisely. This block allows a proportional line following: the more the robot is off the track, the more it will steer into the opposite direction. If the offset from the track is 0, the steering is 0 as well (straight).

In step3, we will discuss how we can use this block to follow a line "smoothly" by programming a proportional line follower. The example program below will display the deviation on the screen, so you can try moving the robot on the track by hand first:


```
when CyberPi starts up
forever
    show quad rgb sensor 1 ▾ deviation (-100~100) at center of screen ▾ by middle ▾ pixel
```

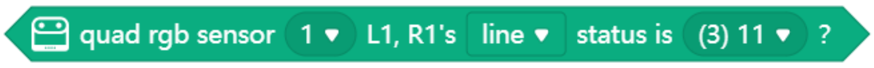
2. Reproducing and testing some programming examples of the Quad RGB sensor

In the diagram above, each code block of the Quad RGB sensor has a programming example. You are going to recreate these programming examples in mBlock and test them. For one programming example, come up with your own modification to the code.

3. How can the mBot2 follow a line?

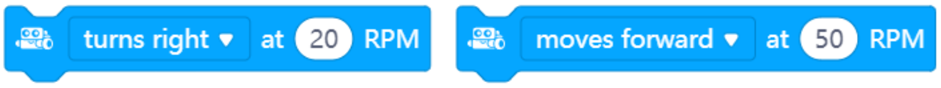
The basis of the assignment in 'Step 3' of this lesson is that the mBot2 is going to follow the track on a map, being a touring coach. For this purpose, you will need to use the code examples explained and tested so far and modify them accordingly.

First use this coding block:


quad rgb sensor 1 ▾ L1, R1's line ▾ status is (3) 11 ▾ ?

If the robot is on the track (both L1 and R1 detect a line, code value3), the robot should drive straight. If only one of the two inner sensors identifies a line, then it should turn left or right accordingly.

Use these driving blocks (only turning right is shown):



```
turns right ▾ at 20 RPM
moves forward ▾ at 50 RPM
```

They just start the motors as needed, and the next code block will be executed immediately. Since you don't know how far the robot is off or when a next curve will occur, the driving commands cannot contain any driving time or distance (same for turning). The next iteration of a program loop will check the new sensor data and select the driving accordingly.

If you are using the provided map, make sure to calibrate the sensor as described to the brightest color (yellow section on the track).

3. Trying out (25 min)

Step 3: Trying it out

You have already learned a lot about the Quad RGB sensor on the mBot2. You are now going to create your own computer program in mBlock using the Quad RGB sensor.

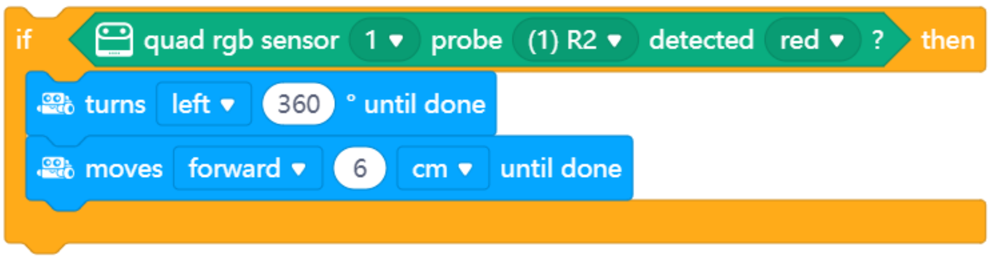
In this assignment you will turn the mBot2 into a touring car. The tour coach takes a tour of landmarks in a city. In the box of the mBot2 there is a map with a track. The coach follows the black line on the map. In the previous part you should have already programmed a simple line following robot, now it is time to add more enhancements.

To make the program easier to understand, you can "build" your own coding blocks. The entire line-following part of the program can be added up to one simple command, e.g., "simple_line". The red/pink "My Blocks" in the code menu allows to define a custom block and structure your code in a nice way. The example program below makes use of custom blocks.

In addition to following the track, the coach performs an action at some of the sights. The sights are indicated by a colored area (red, yellow, green, and blue).

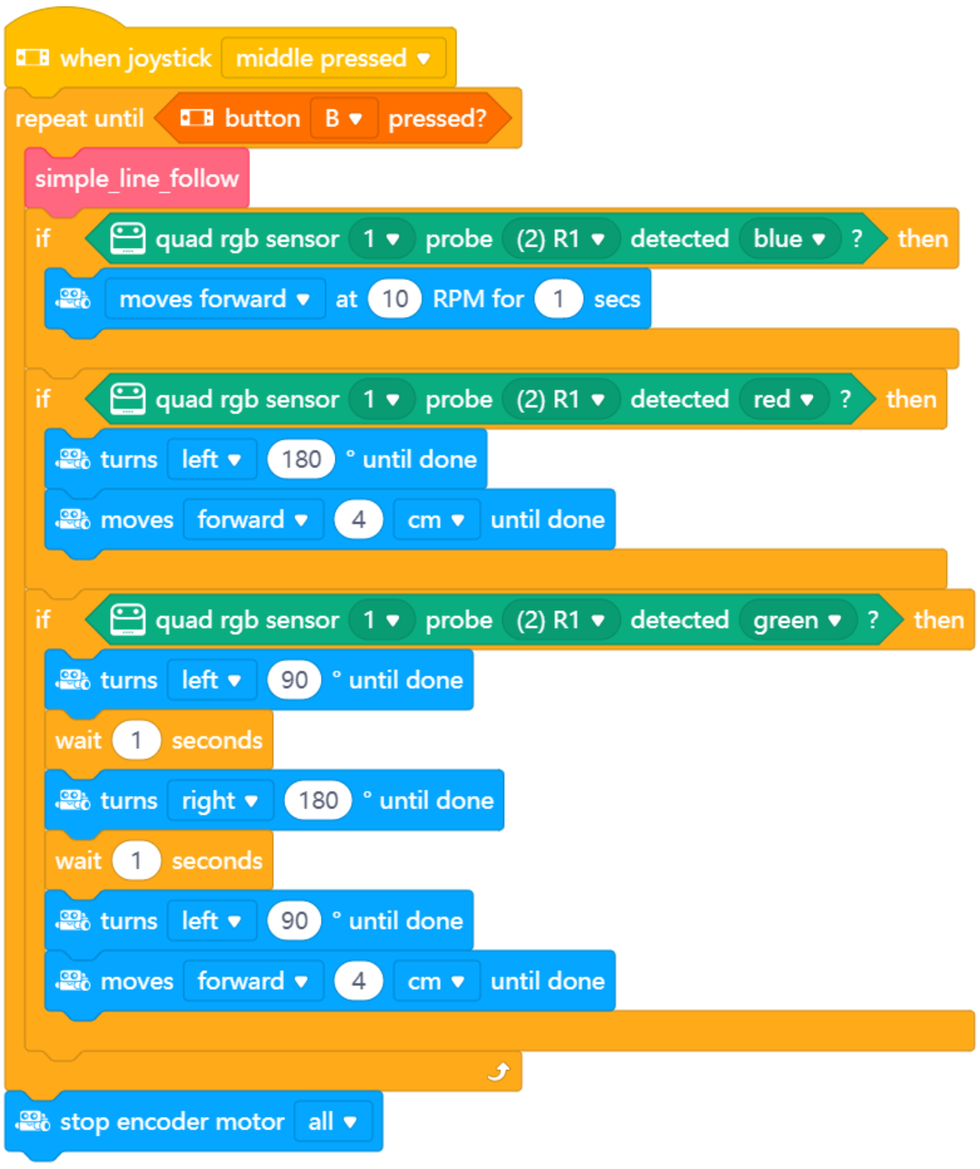
You can think up the actions that the coach should perform. Are you going to have the coach drive around Paris? Then it might be fun to play the French national anthem. Or the bells of Big Ben if the coach is driving through London.

Of course, you can also keep it a little simpler. In the example below, you have the coach turn around at the red marker and do different activities at other colored markers.



```
if quad rgb sensor 1 ▾ probe (1) R2 ▾ detected red ▾ ? then
    turns left ▾ 360 ° until done
    moves forward ▾ 6 cm ▾ until done
```

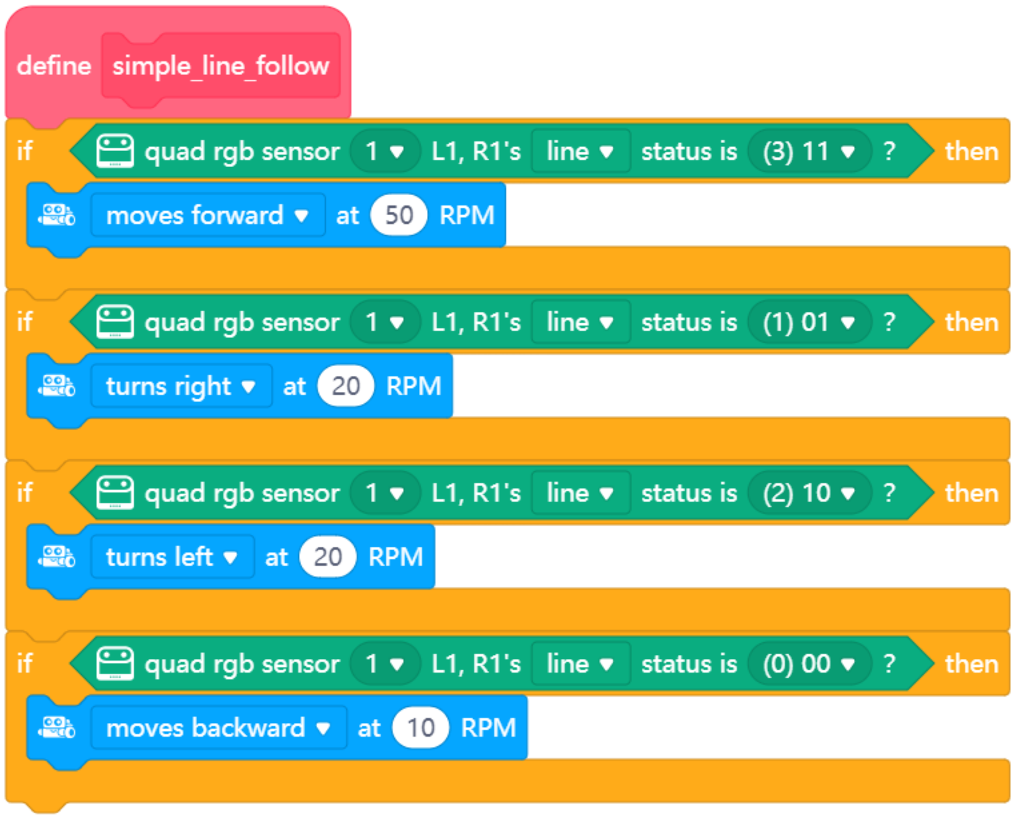
Use the knowledge you gained in 'Step 2' of this lesson. Of course, you can do plenty of experimenting yourself with the different programming possibilities in mBlock. Do you get stuck? Then you can use the programming examples below. You can copy it and adapt it to your own wishes and preferences.



```
when joystick middle pressed ▾
repeat until button B ▾ pressed?
    simple_line_follow
    if quad rgb sensor 1 ▾ probe (2) R1 ▾ detected blue ▾ ? then
        moves forward ▾ at 10 RPM for 1 secs
    if quad rgb sensor 1 ▾ probe (2) R1 ▾ detected red ▾ ? then
        turns left ▾ 180 ° until done
        moves forward ▾ 4 cm ▾ until done
    if quad rgb sensor 1 ▾ probe (2) R1 ▾ detected green ▾ ? then
        turns left ▾ 90 ° until done
        wait 1 seconds
        turns right ▾ 180 ° until done
        wait 1 seconds
        turns left ▾ 90 ° until done
        moves forward ▾ 4 cm ▾ until done
stop encoder motor all ▾
```

This program starts if you press B and it stops with pressing A. It performs a simple line following using custom "My Blocks"-code and then checks if the R1 sensor detects either blue (drive very slowly) red (turn around), or green (look to each side). To make sure the color does not get recognized again after the activities were done, the robot drives 4cm forward each time.

The "simple_line" block is defined as follows:



```
define simple_line_follow
if quad rgb sensor 1 L1, R1's line status is (3) 11 ? then
    moves forward at 50 RPM
if quad rgb sensor 1 L1, R1's line status is (1) 01 ? then
    turns right at 20 RPM
if quad rgb sensor 1 L1, R1's line status is (2) 10 ? then
    turns left at 20 RPM
if quad rgb sensor 1 L1, R1's line status is (0) 00 ? then
    moves backward at 10 RPM
```

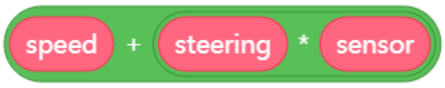
Advanced version:

In the previous step, the deviation from the line was introduced. This block reports how far the robot is off track (-100 to +100; 0 for perfect in center) and it can be used to calculate the steering of the robot. This is a proportional line following – the more the robot is off track, the more it needs to steer against this. If the robot is right off track, the deviation is positive (robots needs to drive to the left); for negative values, the robot is left to the track (needs driving right).

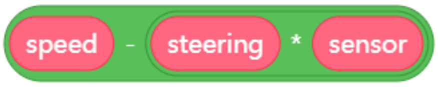
For implementing this, you need a basic speed – that is the speed for moving forward on a straight track. For steering left or right, one of the wheels needs to turn faster than the other – the greater the difference, the greater the steering.

To calculate the steering, we need to add the deviation to one motor's basic speed and subtract it from the other. To allow for a finder adjustment, a factor will be used to adjust how strong the deviation affects the steering, since the range for deviation itself is too big to be used directly for controlling the motors.

But to which motor must the calculated value be added or subtracted? A positive deviation means the robot is off track on the right side and must turn left. This means that the right wheel must run faster – we add the scaled deviation to the right wheel and subtract it from the left therefore.

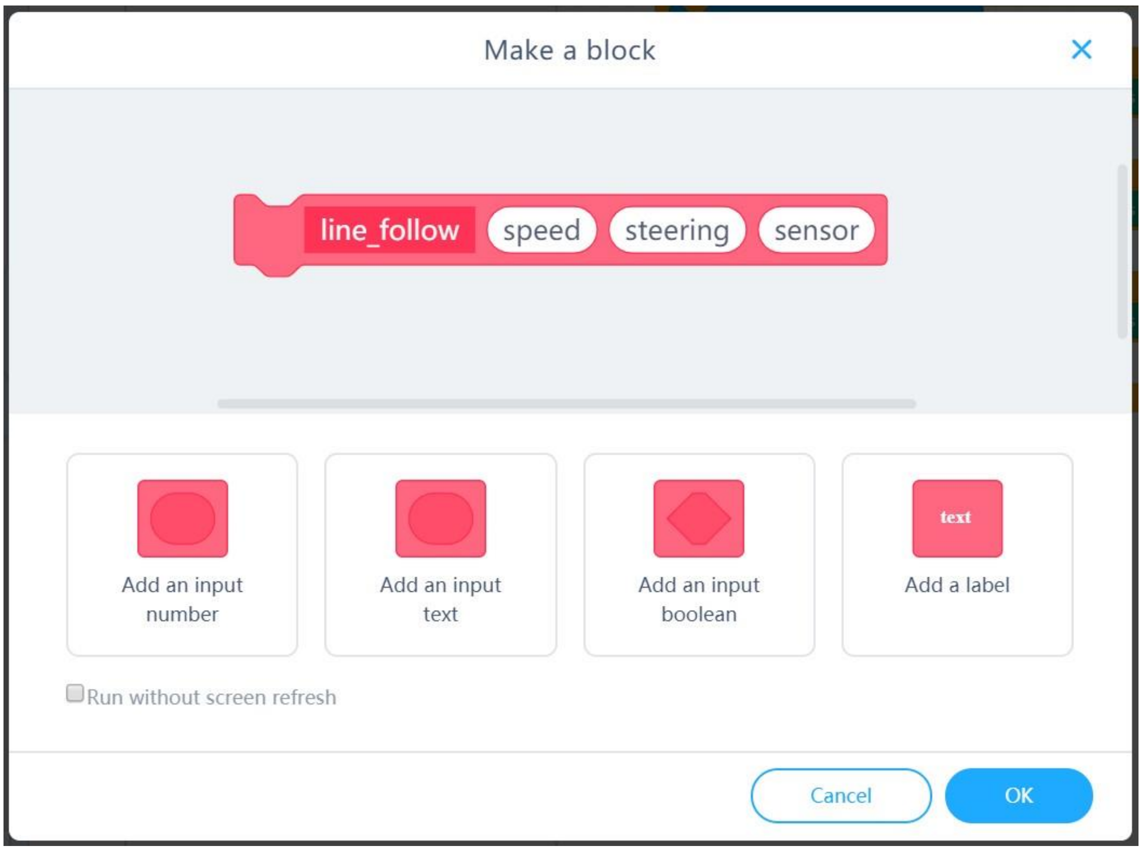


```
speed + steering * sensor
```

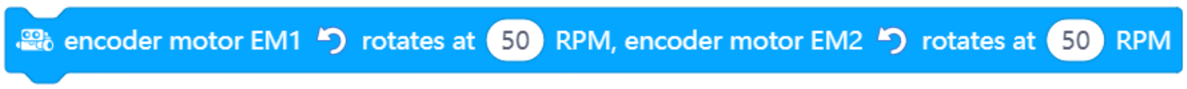


```
speed - steering * sensor
```

When setting up the "My Blocks", you can add inputs and rename them to a meaningful name:

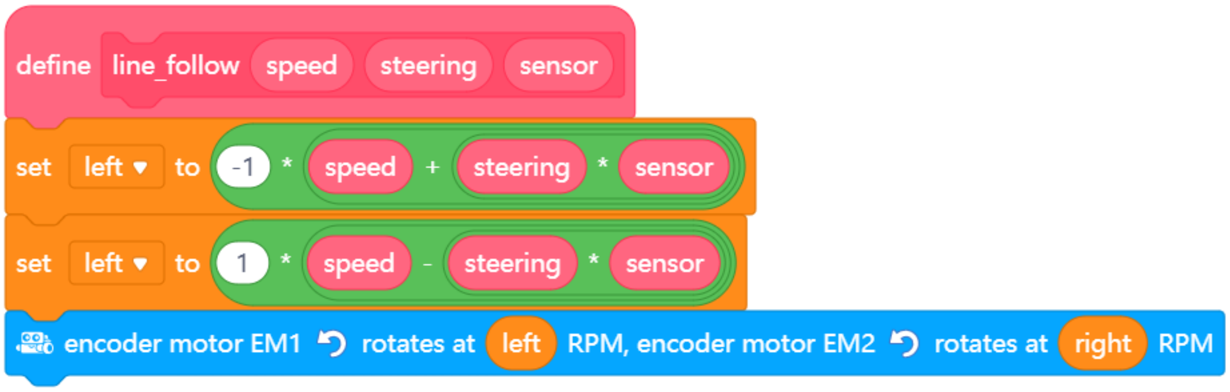


The only thing left to take care of, is that the movement block (shown below) requires negative values for the right motor and positive for the left one to drive forward (this is because they are mounted facing each other).



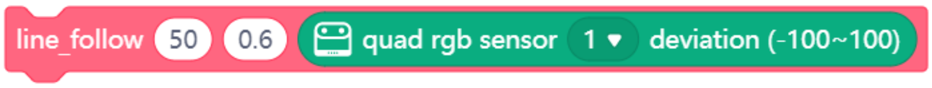
```
encoder motor EM1 ↺ rotates at 50 RPM, encoder motor EM2 ↺ rotates at 50 RPM
```

To finalize the calculation, the values for the right motor need to be multiplied by -1. This is the definition of the proportional line following block:



```
define line_follow speed steering sensor
set left ▾ to -1 * speed + steering * sensor
set left ▾ to 1 * speed - steering * sensor
encoder motor EM1 ↺ rotates at left RPM, encoder motor EM2 ↺ rotates at right RPM
```

In the main loop of a program, you simply use this block with one command:



```
line_follow 50 0.6 quad rgb sensor 1 ▾ deviation (-100~100)
```

This calls the sub-program with a basic speed of 50 and a steering factor of 0.6 and sends the current deviation from the sensor reading as well. You can experiment with the values a bit – if the robot "oscillates" (rapidly switching from left to right turning etc.) the steering factor is too high; if it goes off track even on weak curves, it is too low.

When working on this assignment, it is helpful to use the following step-by-step plan. Do you have an idea of what you want to make? If so, first discuss with your teacher whether this is feasible.

	Explanation
Step 1: What do you want to do?	• In what way should the mBot2 move?
Step 2: What do you need?	• What do you need in addition to the mBot2?
Step 3: What code blocks do you need to make the mBot2 move?	• How can you make the mBot2 change direction at each corner? • What code blocks do you need to do this? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML)

	• If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well
Step 4: Testing and implementation	• Is the first version ready? Test it! During the testing round, write down points of improvement • Work on the improvement points until your mBot2 runs error-free over your track • Successful? Film the end result and ask your teacher if you can post it on social media with the hashtag #mBot2

4. Wrap-up (5 min)

Step 4: Wrap-up

Did you manage to program the mBot2 as a touring car around the track?

In this lesson, you learned about the Quad RGB sensor and where you might encounter it in everyday life. You know how to program the mBot2's Quad RGB sensor to make the robot follow a line. You also learned how to program the mBot2 to drive a route based on different colors.

It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think turned out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 6: Careful drive

Subject: STEAM **Grade(s):** 5th and up

Duration: 45 minutes **Difficulty:** Beginner

Lesson objectives

By the end of this lesson, students will be able to:

- Recognize and use the code blocks for the gyroscope accelerometer in CyberPi
- Build their own computer program in mBlock to measure the changes on the inclination of mBot2 and adjust its movements accordingly

Overview

A gyroscope measures tipping movements, more precisely the speed of turning/tipping movements. An accelerometer measures the change in velocity. Both sensors provide different information about the position in space.

Combining both types of data, the position and the movement of vehicles, robots or even space probes can be calculated. On bumpy roads, for example, robots might drive slower to prevent tipping over.

Focus

By the end of this lesson, students will know:

- How gyroscopes and accelerometers work and how their measurements can be combined
- How to make a robot react to sudden changes and detect crashed or bumpy roads

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for Chromebook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle

Lesson plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	1. Warming up • Gyroscopes and accelerometers in everyday life • Which tilt movements can the mBot2 detect?
10 minutes	2. Hands-on • Getting acquainted with the different programming blocks of the gyroscope and accelerometer • Recreating and testing some programming examples of these sensors.
25 minutes	3. Trying out • Writing your own program for the robot
5 minutes	4. Wrap-up • What did you learn in this lesson? • Showtime: show what you did with your robot in a fun, short movie for later discussion • If your teacher allows, share the end result on social media with the hashtag #mBot2 • Reflection: What are you most proud of? What would you like to improve about your robot?

1. Warming up
(5 min)

Step 1: Warming up

This step consists of two parts:

1. Gyroscopes and accelerometers in everyday life
2. Which tilt movements can the mBot2 detect?

1. Gyroscopes and accelerometers in everyday life

How do devices like smartphones identify their position (hold upright, display facing up/down)? How to determine if a crash is severe enough to ignite airbags? Estimating the current position without an external precise reference has been a challenge at sea with cloudy skies for centuries – and it is still a very important task for household items, robots and even space probes. They all use at least two different sensors, gyroscopes, and accelerometers. A gyroscope works with a crystal that is under electric tension. This converts the direction of the force of the tipping motion into an electrical signal. Gyroscopes measure the rotational speed in degrees per second and with the help of math, they can estimate the rotated angle as well – but the latter is not that accurate: gyroscopes "drift" over time. Accelerometers report the change of position (like in- or decrease of speed) – and regarding the earth's gravity, they report the gravitational acceleration (as compared to free fall). Both types of sensors can measure on each spatial axis, reporting either rotational speed or acceleration on X-, Y- and Z-axis.

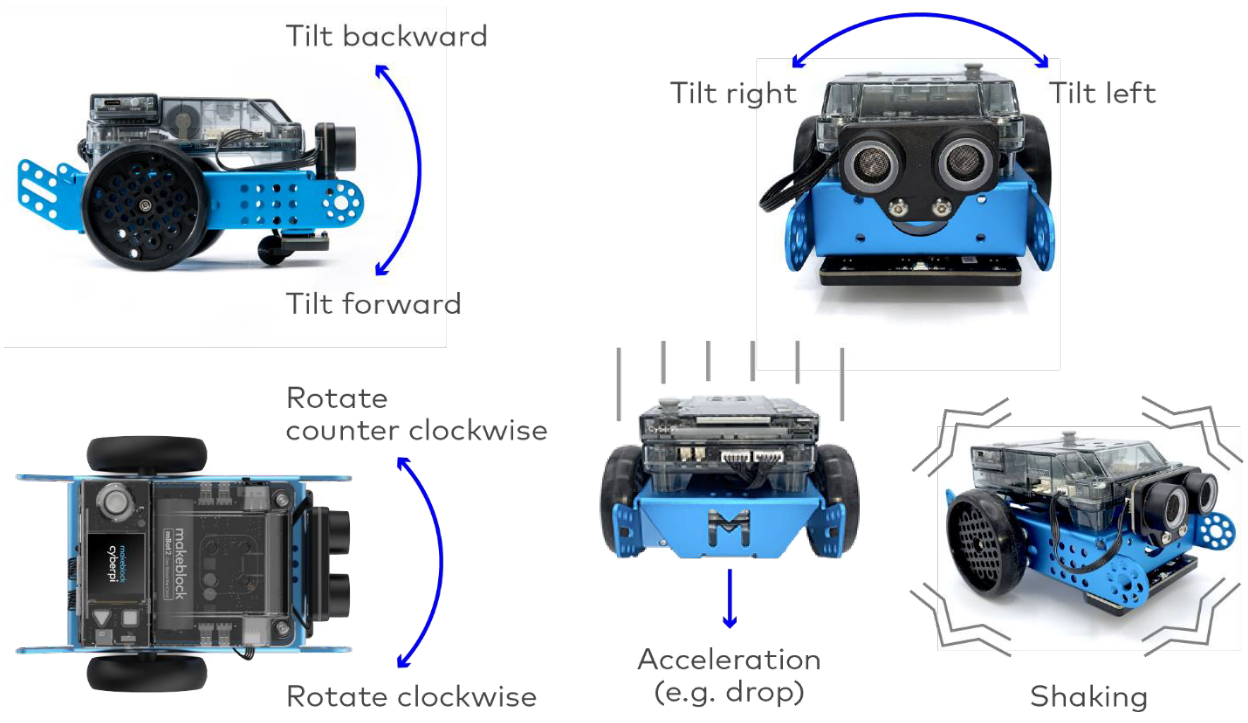
Combining data from both gyroscope and accelerometer, the direction and strength of movements can be calculated (in reality this is not super precise, so it is more a very good estimation). In Robotics these sensors are used to stabilize vehicles in uneven ground (or in flight) and help them maintain their programmed path. For the mBot2, these sensors are integrated into a small chip on the CyberPi.

Therefore, we will address the combined sensor as "motion sensor".

2. Which tilt movements can the mBot2 detect?

Before you learn to program in mBlock5 with the motion sensor, it is useful to know which tilt motions the mBot2 can measure. You can show the measurements of the tilting movements on the CyberPi display. In lesson 2 you have already learned about how to do this.

The mBot2 can detect the following tilt movements:



2. Hands-on (10 min)

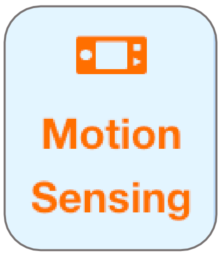
Step 2: Hands-on

This step consists of two parts:

1. Getting acquainted with the different programming blocks of the motion sensor
2. Recreating and testing some programming examples of the motion sensor

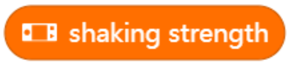
1. Getting acquainted with the different programming blocks of the motion sensor

In mBlock there are several code blocks for the motion sensor you can use in your own programs. These code blocks can be found in the 'Motion Sensing' category of the block field in mBlock. These code blocks are orange.



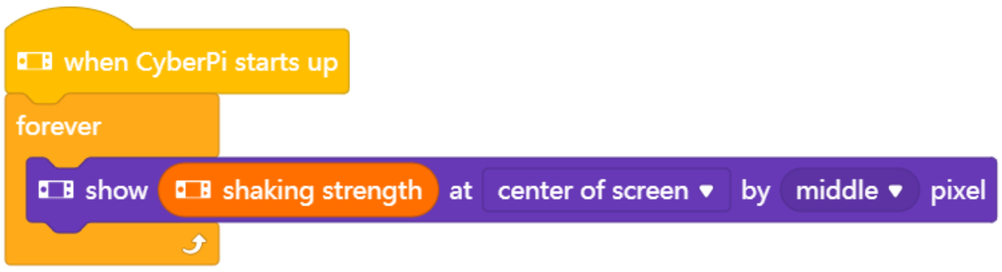
The table below gives a detailed explanation on the code blocks together with some sample program code to test it.

Code block:



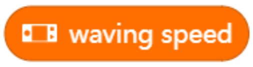
```
shaking strength
```

This code block reports how hard the mBot2 is being shaken. At a value of '0' the mBot2 is not shaken at all. With a value of '100' a very hard shaking movement is detected. In the programming example below, the strength of the mBot2's shaking is shown on the display. If the mBot2 stops shaking, the display is blank.



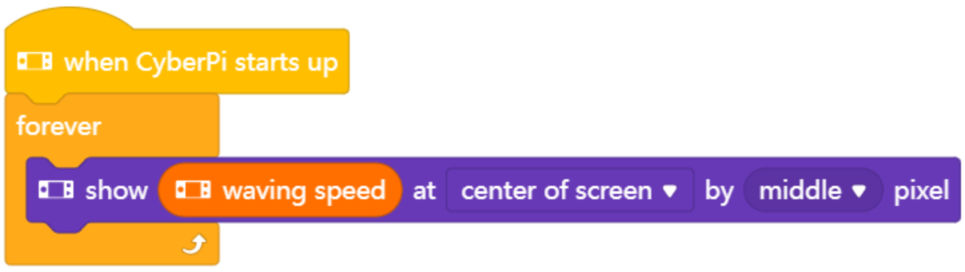
```
when CyberPi starts up
forever
    show (shaking strength) at [center of screen ▾] by [middle ▾] pixel
```

Code block:



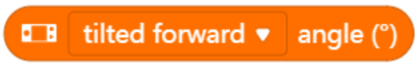
```
waving speed
```

This code block is used to record the waving speed. The waving speed ranges from 0 to 100. These units do not represent speed in m/s or in degrees/s, they are just a simple reference. In the programming example below, the waving speed of the mBot2 is shown in the display. If the mBot2 stops waving, the display is blank.



```
when CyberPi starts up
forever
    show (waving speed) at [center of screen ▾] by [middle ▾] pixel
```

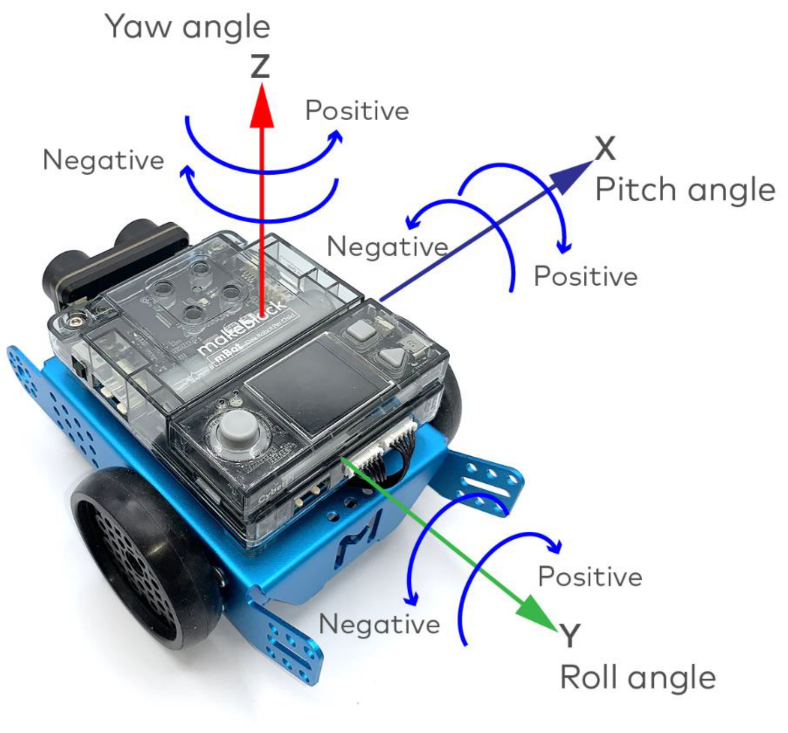
Code block:



tilted forward ▾ angle (°)

Each tilt makes a movement at a certain angle in a certain direction. When specifying an angle, the X, Y and Z axes are used to refer to the direction. Instead of these mathematical terms, they are addressed as roll, pitch and yaw as well. These terms originate from navigation of planes. Roll is the axis front-back, pitch refers to up/down turns (axis from left to right) and yaw is the top-down axis responsible for tuning left/right.

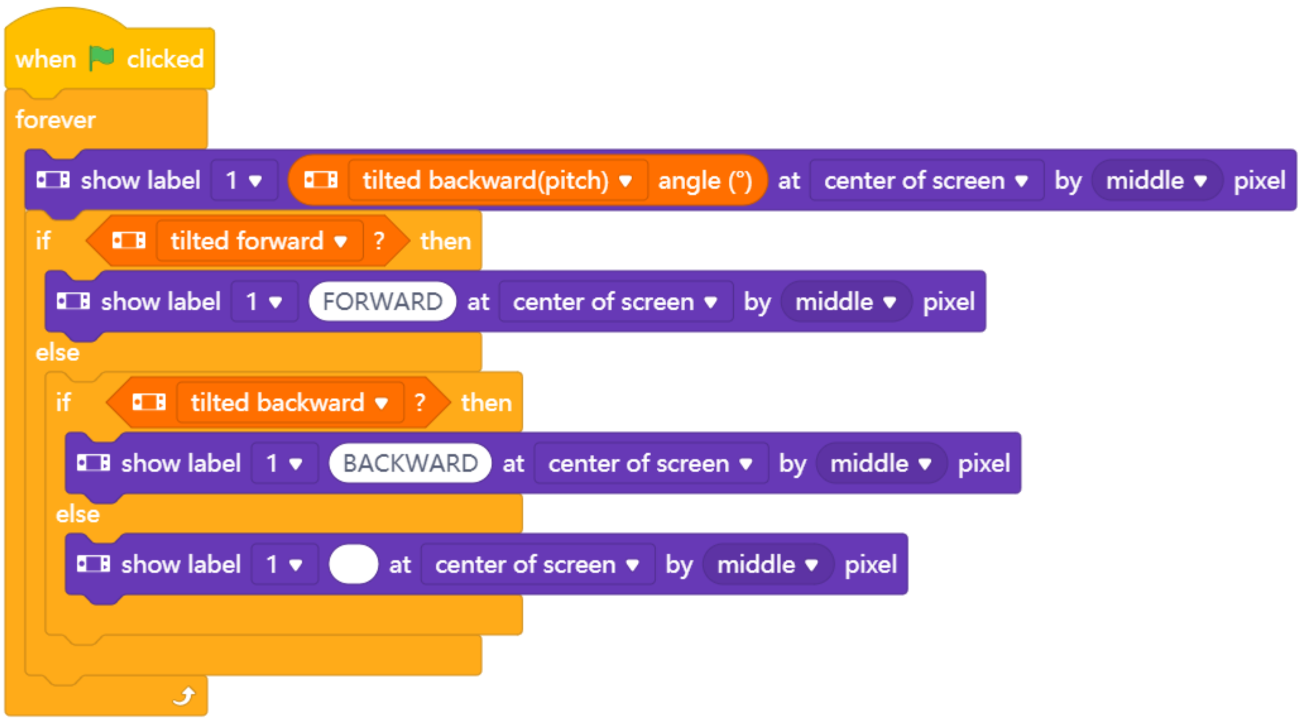
Take a look at the image below.



Angles are given in degrees:

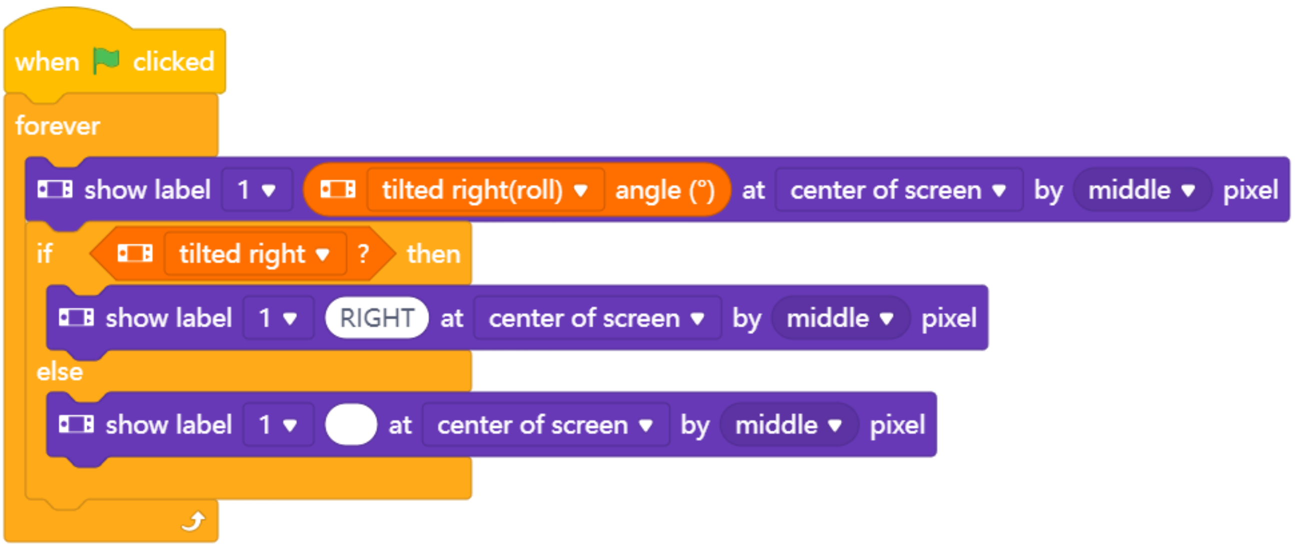
- Roll angle: -180° to +180°
- Yaw angle: -180° to +180°
- Pitch angle: -90° to +90°

In the program below, the pitch angle will be displayed in the top row and if a tilting forward or backward movement is detected, the words “forward” or “backward” appear on the last line. Can you identify the threshold for detection?



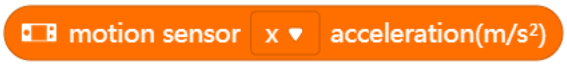
```
when 🏴 clicked
forever
    show label 1 ▾ (tilted backward(pitch) ▾ angle (°)) at center of screen ▾ by middle ▾ pixel
    if <tilted forward ▾ ?> then
        show label 1 ▾ (FORWARD) at center of screen ▾ by middle ▾ pixel
    else
        if <tilted backward ▾ ?> then
            show label 1 ▾ (BACKWARD) at center of screen ▾ by middle ▾ pixel
        else
            show label 1 ▾ ( ) at center of screen ▾ by middle ▾ pixel
```

Try to experiment with the different settings from the dropdown-menu. Now you have analyzed the tilting movement on the pitch/x-Axis. If you switch to tilt_left/right, remember to switch the Boolean reported blocks as well:



```
when 🏴 clicked
forever
    show label 1 ▾ (tilted right(roll) ▾ angle (°)) at center of screen ▾ by middle ▾ pixel
    if <tilted right ▾ ?> then
        show label 1 ▾ (RIGHT) at center of screen ▾ by middle ▾ pixel
    else
        show label 1 ▾ ( ) at center of screen ▾ by middle ▾ pixel
```

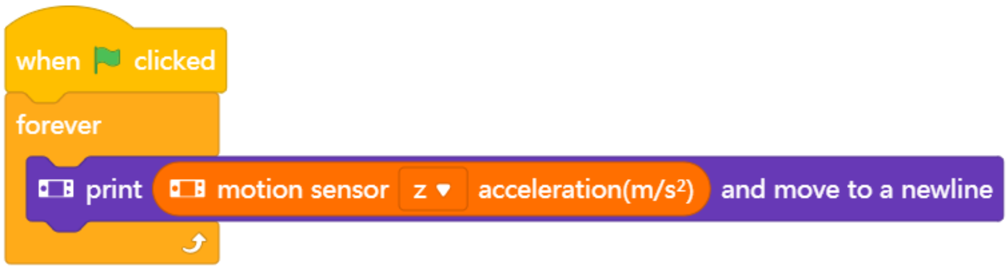
Code block:



```
motion sensor [x ▾] acceleration(m/s²)
```

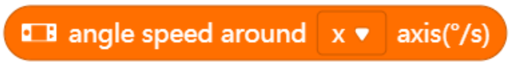
So far, we have looked at tilting movements, but the motion sensors incorporate another sensor, the accelerometer. It measures changes in the velocity of the mBot2 over time. This can be slow increases in driving speed as well as a sudden stop or even a side-impact from another vehicle. Since these measures are based on the x, y and z axes of the robot, you can detect a crash if you look for a rapid change in acceleration. The accelerometer indicates the acceleration in m/s2.

In the programming example below, the acceleration of the mBot2 on the z-axis is shown in the display. Move the mBot2 by hand and check the display. What readings does it show if the robot "crashes" into your hand or a book? Can you change the program and simulate a side-impact?



```
when 🏁 clicked
forever
    print (motion sensor [z ▾] acceleration(m/s²)) and move to a newline
```

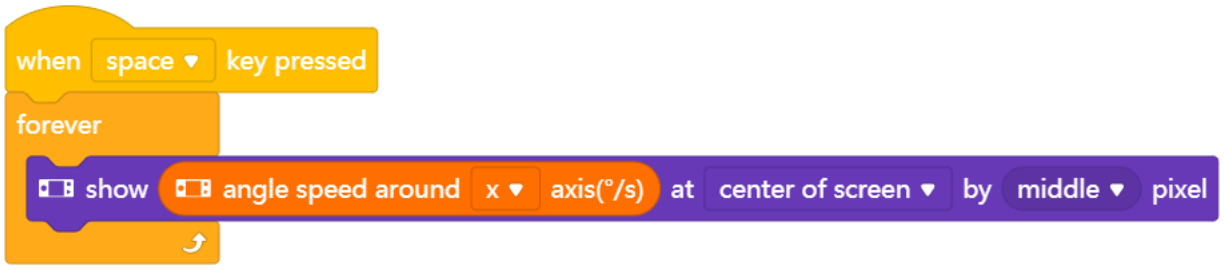
Code block:



```
angle speed around [x ▾] axis(°/s)
```

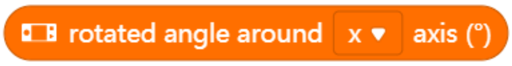
This code block shows the speed at which the mBot2 rotates around a certain axis. This is also called the angular velocity. The angular velocity is shown in degrees per second (°/s).

In the programming example below, the angular velocity of the mBot2 around the x-axis is shown on the display.



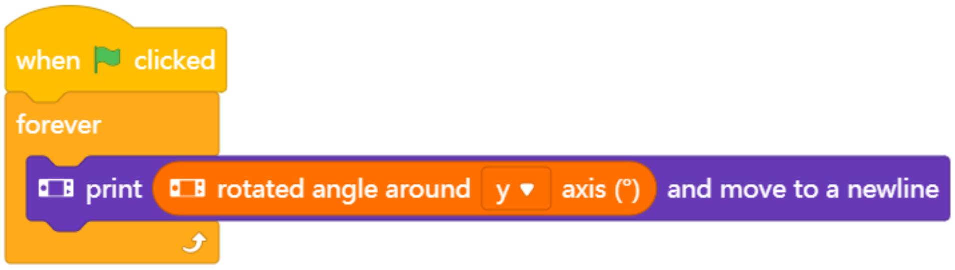
```
when [space ▾] key pressed
forever
    show (angle speed around [x ▾] axis(°/s)) at [center of screen ▾] by [middle ▾] pixel
```

Code block:



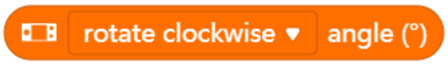
```
rotated angle around x ▾ axis (°)
```

This code block indicates the angle at which the mBot2 rotates around a given axis. The angle is displayed in degrees (°) and is measured relatively. The initial position is set when the mBot2 is turned on. This data is calculated from the rotational speed the gyroscope reports. It can drift over time and it is not that stable – try tilting the mBot2 sideways with the program below and check the values. Do they go back to their initial state?
In the programming example below, the angle at which the mBot2 rotates around the y-axis is shown on the display.



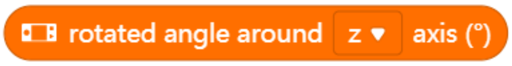
```
when 🏁 clicked
forever
    print (rotated angle around y ▾ axis (°)) and move to a newline
```

There are two important aspects: this block will keep track of rotational movements that are greater than one revolution. Use this block to count how many times the robot has turned. Looking at the z-Axis, a clockwise rotation will increase the values reported by this block:



```
rotate clockwise ▾ angle (°)
```

While the rotated angle around the z-axis decreases:



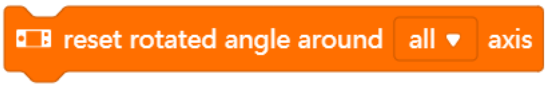
```
rotated angle around z ▾ axis (°)
```

After 3 Rotations of 360° the last block will report "-1080°", but the previous block will just show "0" (back to initial position, ignoring multiple rotations). If then rotated counterclockwise 4 times (one more time previously), the "rotate_clockwise" angle is at 0° again, while "rotated_angle_around_z axis" now shows 360° (all figures without gyroscope drift).

If you just need to take small rotational movements into consideration, for the x- and y-axis, the "tilted (___) angle" blocks discussed beforehand provide a more robust measurement.

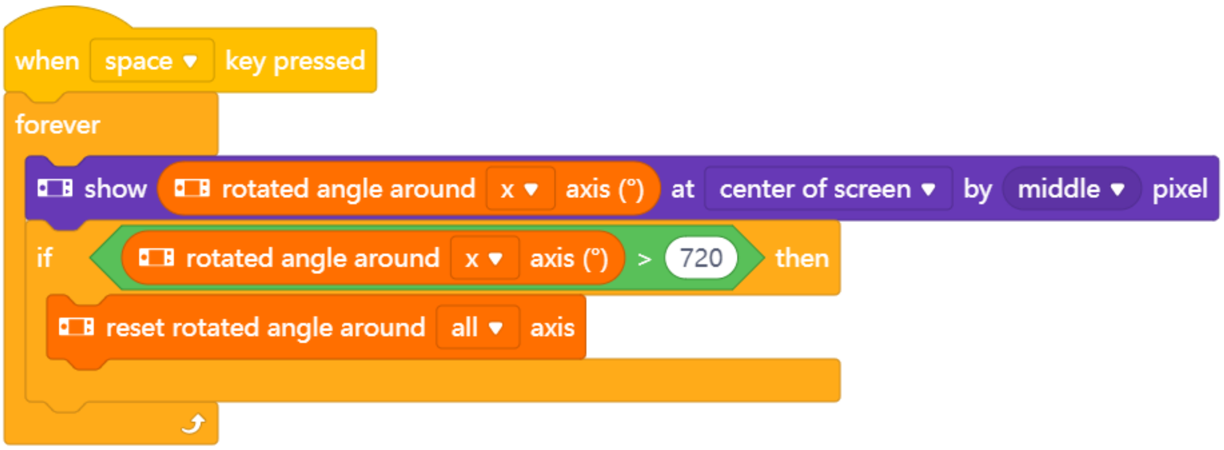
There is one exception: rotations around the yaw/Z-Axis can only be measured with the help of the gyroscope. If you want to measure the degrees, the mBot2 is turning left/right, you need to work with this data.

Code block:



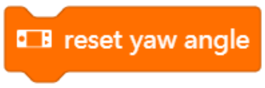
```
reset rotated angle around [all ▾] axis
```

If you want to determine the angle that the mBot2 is making, it is easier to measure the change from the value 0. This code block resets the measured value of an angle to the starting value 0, defining a new starting position for future turns.
In the programming example below, the angle at which the mBot2 turns around the z-axis is shown on the display. If the angle is greater than 720°, then the measured value of the angle is set to 0 and it is measured again.



```
when [space ▾] key pressed
forever
    show (rotated angle around [x ▾] axis (°)) at [center of screen ▾] by [middle ▾] pixel
    if <(rotated angle around [x ▾] axis (°)) > (720)> then
        reset rotated angle around [all ▾] axis
```

Remember, there are two blocks to measure the rotation: one only cares for full rotations, the other keeps counting. This reset-block sets all (or selected) "rotated_around_axis" blocks to zero. It does not have an effect on the "rotate clockwise"/ "rotate counterclockwise" block. This one has its own reset block:



```
reset yaw angle
```

Any setting-to-zero only applies to rotations reported by the gyro sensor alone. The tilted ___ angle reports always from a perfectly leveled plane perpendicular to the earth center and cannot be set to zero at will.

2. Recreating and testing some programming examples of the motion sensor

In the table above each code block of the motion sensor has a programming example. You are going to recreate these programming examples in mBlock and test them.

The motion sensor on the CyberPi registers the tilting movements of the mBot2. When testing the above programming examples; therefore, you actually only need the CyberPi. If it is more convenient, remove the CyberPi from the mBot2 and make the tilting movements with the CyberPi yourself during testing. You can test the programming examples in live mode. Since the battery is in the mBot2 shield, you will need to connect the CyberPi to the computer via the USB-C cable to run the programs.

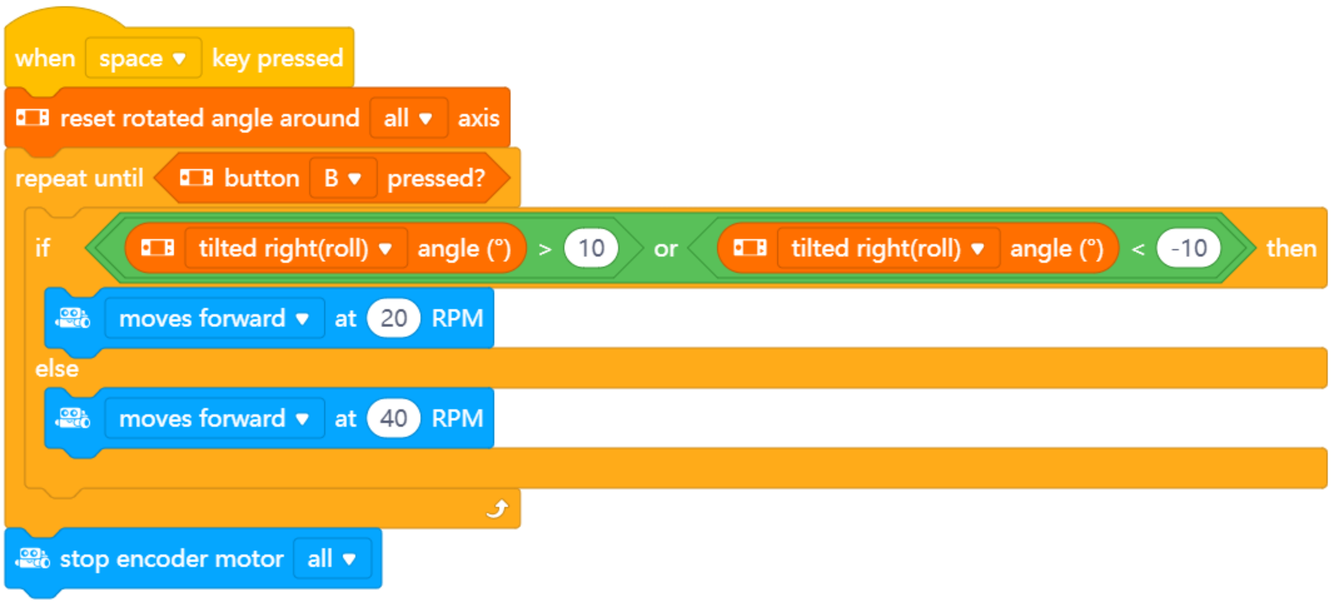
3. Trying out (25 min)

Step 3: Trying out

You have already learned a lot about the motion sensor on the mBot2. You are now going to create your own computer program in mBlock using this sensor.

In this assignment you are going to create a bumpy track for the mBot2. You can make this track from, for example, (small) stones or plasticine on cardboard. The idea is for the mBot2 to drive over the course. If the mBot2 is shaken (hard), it must drive slower. You can also put an object on the mBot2 that must not fall while driving.

Tricky task, isn't it? Fortunately, you don't have to invent everything yourself! Below you can see a programming example that you can extend. In this programming example, the mBot2 will slow down if it makes too large tipping movements.



```
when space key pressed
reset rotated angle around all axis
repeat until button B pressed?
    if tilted right(roll) angle (°) > 10 or tilted right(roll) angle (°) < -10 then
        moves forward at 20 RPM
    else
        moves forward at 40 RPM
stop encoder motor all
```

Use the knowledge you gained in 'Step 2' of this lesson. Of course, you can do plenty of experimenting yourself with the different programming possibilities in mBlock.
When thinking about this assignment, it is helpful to use the following step-by-step plan.

	Explanation
Step 1: What do you want to do?	• What route do you want the mBot2 to drive? • When should the mBot2 slow down? • Do you want the mBot2's sensors to also show data on the display?
Step 2: What do you need?	• What do you need in addition to the mBot2?
Step 3: What code blocks do you need to make the mBot2 drive?	• How are you going to make the mBot2 drive? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every code block in mBlock as well
Step 4: How do you want to show the data from the sensors on the display?	• How do you want to show the data on the display? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well
Step 5: Testing and implementation	• Is the first version ready? Test it! During the testing round, write down areas of improvement • Work on the improvement points until your mBot2 does exactly what you had in mind • Successful? Film the end result and consult with your teacher if you can post it on social media with the hashtag #mBot2

4. Wrap up (5 min)

Step 4: Wrap up

How did this task go? Did the object stay neatly on the mBot2 while driving?

In this lesson, you learned about sensors for measuring movement around axes and acceleration, the gyroscope and the accelerometer. They are integrated into a single little component placed on the CyberPi. These combined sensors are used to determine the position of a robot in 3D-space. They are called inertial measurement units, IMU. You have learned where to encounter them in everyday life. You know what tilt-movements and movement-changes the mBot2 can measure and how you can use different code blocks to take advantage of these measurements in your programs with mBot2.

It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think turned out well?
- What could be even better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 7:

A network game

Subject: STEAM **Grade(s):** 5th and up

Duration: 45 minutes **Difficulty:** Beginner

Lesson objectives

By the end of this lesson, students will be able to:

- Have multiple mBot2 communicate with each other wirelessly without the need for a WIFI access point
- Write their own computer program in mBlock to control multiple mBot2
- Apply a random selection in a computer program

Overview

LAN is the abbreviation for Local Area Network. The mBot2, with its built-in WIFI module, automatically establishes a network connection as soon as there are multiple mBot2 or CyberPi in the same room within a 30-meter radius. Over this network connection, the mBot2 and CyberPi can communicate with each other and execute commands.

Focus

By the end of this lesson, students will know:

- What a LAN connection is
- How to set up this connection and where to apply it
- How random selection works when programming

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for Chromebook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi (multiple mBot2 and/or CyberPi if possible, for the Trying Out step)
- A USB-C cable or Makeblock Bluetooth dongle
- Color cards (blue is suggested)

Lesson plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	1. Warming up • A LAN connection in everyday life • How to set up a LAN connection with mBot2 and CyberPi
10 minutes	2. Hands-on • Getting acquainted with the different code blocks to set up a LAN connection • Recreating and testing programming examples of the LAN connection
25 minutes	Section 3 – Trying out • Making your own game
5 minutes	4. Wrap-up • Showtime: show what you did with your robot in a fun, short movie for later discussion • If your teacher allows, share the end result on social media with the hashtag #mBot2LAN • Reflection: What are you most proud of? What would you like to improve about your robot?

1. Warming up (5 min)

Step 1: Warming up

This step consists of two parts:

1. A LAN connection in everyday life
2. How to set up a LAN connection with mBot2 and CyberPi

1. A LAN connection in everyday life

LAN stands for Local Area Network, and it describes a network of computers and peripherals with a certain spatial extent, usually within a building. The network enables the computers to exchange data with each other inside this network, and (not mandatory) with other networks like the internet. Therefore, they need some kind of connection (wired by cable or by wirelessly/WIFI by radio waves) and a common “language” (called protocol). Normally, the individual computers connect to a managing device that switches and routes the traffic of data (“router”). On a WIFI network this is a dedicated access point or the Internet router offers a built-in n access point.

Can you think of some examples where a LAN network is used? In this lesson, the mBot2 and the CyberPi are the different devices that communicate with each other over a LAN connection.

2. How do you set up a LAN connection with mBot2 and CyberPi

To set up a LAN connection between mBot2 and CyberPi you don't need much. The mBot2 and the CyberPi (or mBot2 among themselves) will automatically be able to communicate with each other wirelessly, there is no need to connect to a Wi-Fi Access point or router for this.

Is a router used? Then all the mBot2 and CyberPi must be put on the same channel of the router. This needs to be set in the router’s settings.

2. Hands-on (10 min)

Step 2: Hands-on

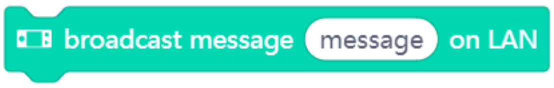
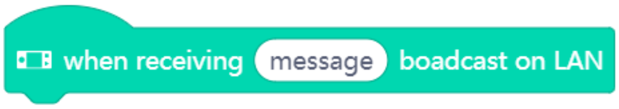
This step consists of two parts:

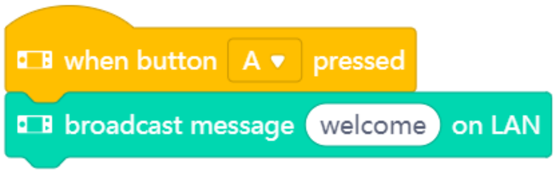
1. Getting acquainted with the different code blocks for the LAN communication
2. Reproduce and test programming some examples of the LAN communication

1. Getting acquainted with the different code blocks for the LAN communication

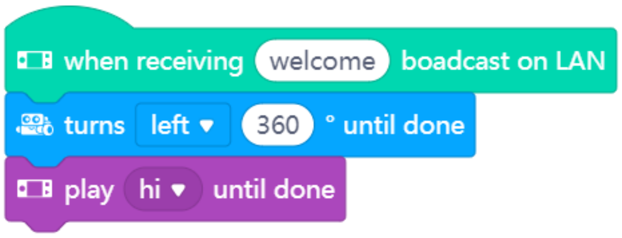
In mBlock, there are several code blocks that you can use to set up a LAN connection. You will find these code blocks in the 'LAN' category of the block field in mBlock. These code blocks are green.

The table below shows an example of these code blocks to set up a LAN connection.

Code block:  broadcast message (message) on LAN This code block allows you to send a message to all other mBot2 and CyberPi on the same network. You can name the message (like a topic).
Code block:  when receiving (message) boadcast on LAN Instead of constantly checking for a received broadcast, there is an event-block. The code attached to this block is only executed once, if a message with the specified topic is received. In the programming example below, there are two mBot2 and/or CyberPi communicating with each other over a LAN connection using these two code blocks. The first program is the "broadcasting unit", intended for the first mBot2 or CyberPi. The second program is the receiver-code for the second mBot2 or CyberPi. You can upload both code parts to both CyberPi/mBot2, since the underlying code will make sure that the sending unit does not receive its own message.



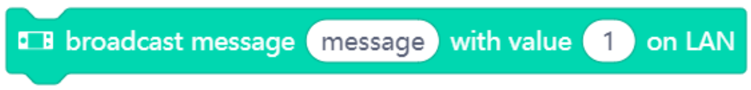
```
when button A ▾ pressed
broadcast message (welcome) on LAN
```



```
when receiving (welcome) boadcast on LAN
turns left ▾ (360) ° until done
play hi ▾ until done
```

After uploading the programming codes, you must restart the mBot2 and/or CyberPi. Now when you press A button of the first mBot2 or CyberPi, the second mBot2 or CyberPi will receive the sent message (e.g. welcome). What happens if you press B on the second CyberPi or mBot2? What would you need to change to send different messages back?

Code block:

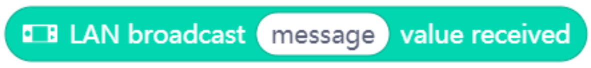


```
broadcast message (message) with value (1) on LAN
```

This code block allows you to send a message (with a given topic like before) and a specific value (number or text) to all other mBot2 and CyberPi on the network.

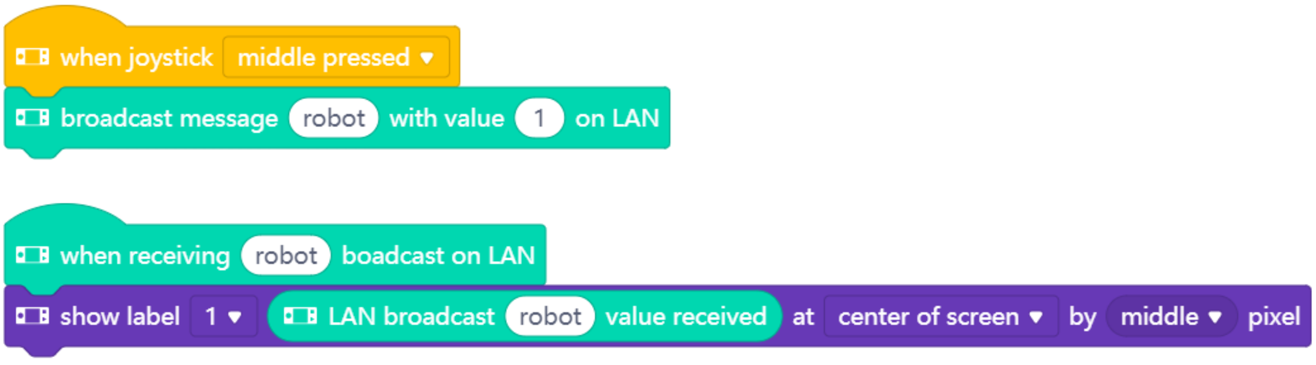
Code block:

For the receiving code, you can use the same event block and specify the topic it should react to exactly like the previous example. To make use of the value (text or number), there is a dedicated reporter block:



```
LAN broadcast (message) value received
```

In the programming example below, there are two mBot2 and/or CyberPi communicating with each other over a LAN connection using these two code blocks. The first program is the "broadcasting unit", intended for the first mBot2 or CyberPi. The second program is the receiver-code for the second mBot2 or CyberPi. You can upload both code parts to both CyberPi/mBot2, since the underlying code will make sure that the sending unit does not receive its own message.



```
when joystick [middle pressed ▼]
broadcast message (robot) with value (1) on LAN

when receiving (robot) boadcast on LAN
show label [1 ▼] (LAN broadcast (robot) value received) at [center of screen ▼] by [middle ▼] pixel
```

After uploading the programming codes, you must restart the mBot2 and/or CyberPi. If you now press the joystick on the first mBot2 or CyberPi, the second mBot2 or CyberPi will receive the sent message (value 1).

2. Reproducing and testing programming some examples of the LAN communication

In the table above, each code block of the LAN connection is accompanied by a programming example. You are going to recreate these programming examples in mBlock and test them. For one programming example, come up with your own extension.

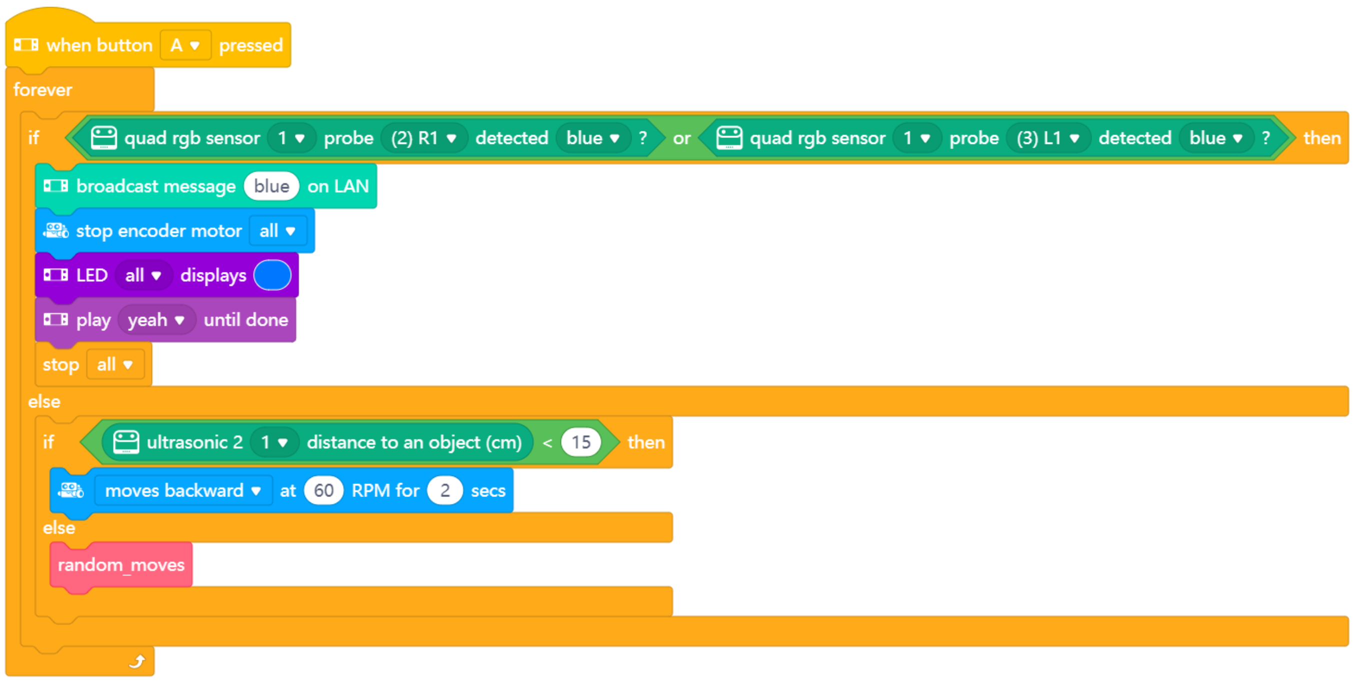
3. Trying out (25 min)

Step 3: Trying out

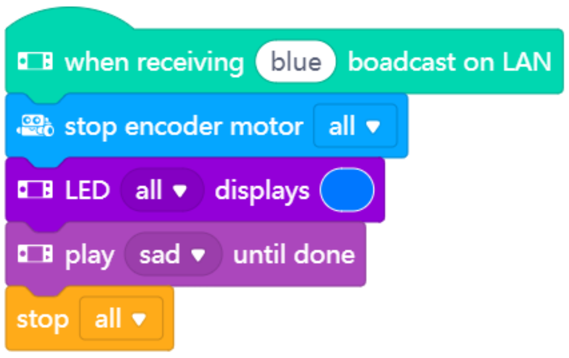
You have already learned a lot about the LAN connection of the mBot2. You are now going to create your own computer program in mBlock using the LAN communication.

In this assignment you are going to program a game by yourself. It is called 'Looking for ... the color blue'. In this game, a group of mBot2 searches for the color blue on the floor. If one of the mBot2 finds the color, it will inform the others via a LAN message and win the game.

Sounds quite complicated! Fortunately, you don't have to program everything yourself. Below you can see a programming example. The same program should be uploaded to all mBot2 so they all follow the same instructions.

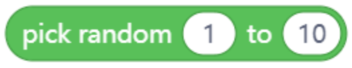


```
when button A pressed
forever
    if quad rgb sensor 1 probe (2) R1 detected blue ? or quad rgb sensor 1 probe (3) L1 detected blue ? then
        broadcast message blue on LAN
        stop encoder motor all
        LED all displays
        play yeah until done
        stop all
    else
        if ultrasonic 2 1 distance to an object (cm) < 15 then
            moves backward at 60 RPM for 2 secs
        else
            random_moves
```



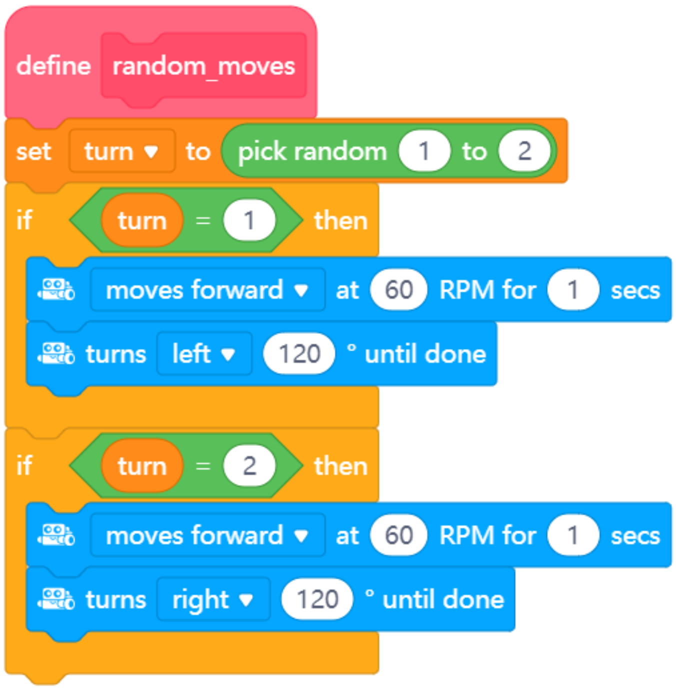
```
when receiving blue boadcast on LAN
stop encoder motor all
LED all displays
play sad until done
stop all
```

To make the mBot2 drive randomly, you can use the following code block:



```
pick random 1 to 10
```

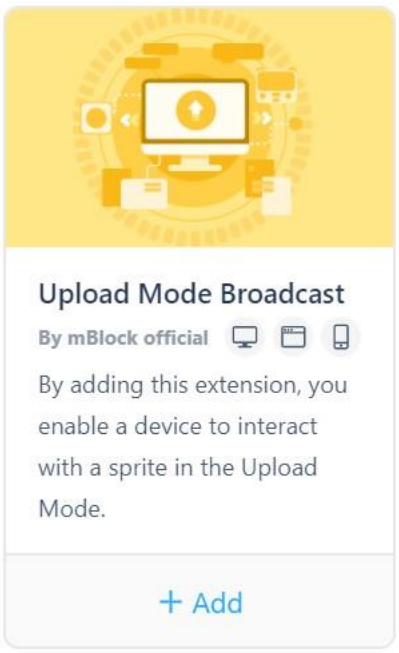
This code block is used to randomly select a value within a certain range. A particular instruction can be assigned to each value of the range. In this way, the mBot2 can choose in a non-linear way from a number of instructions. To make your program easier to understand, you can separate the blocks for this instruction and place them under a custom block, which you can then use in your main program:



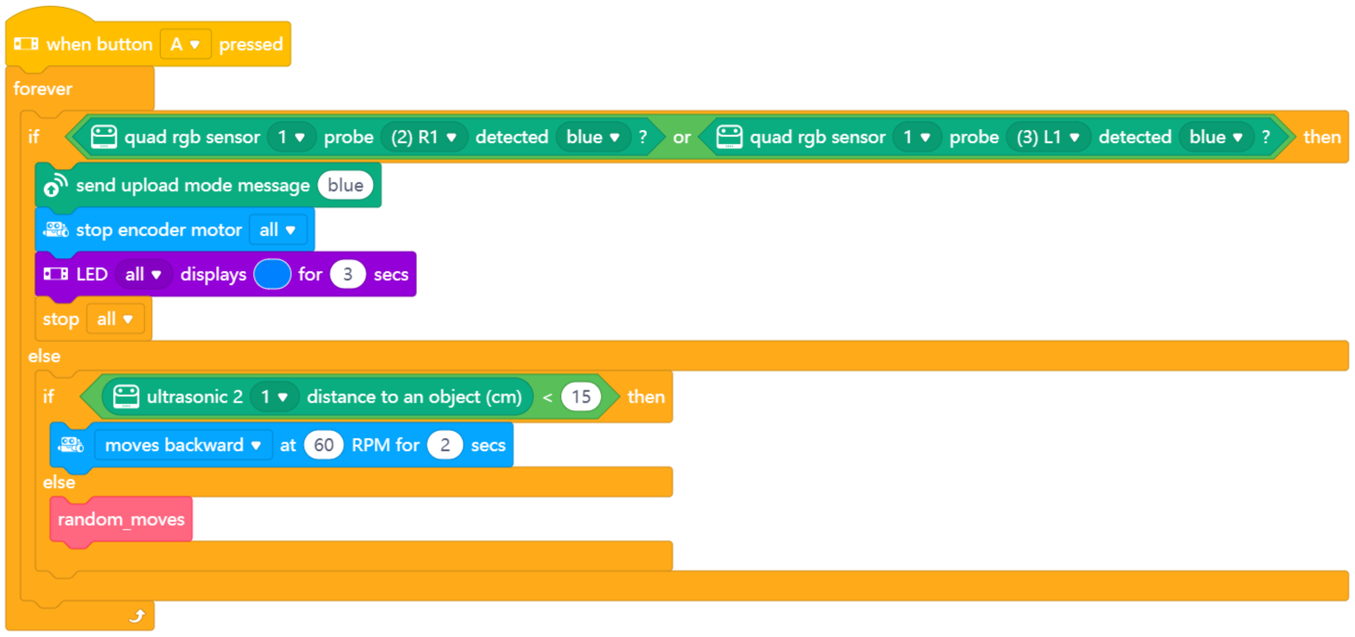
```
define random_moves
set turn to pick random 1 to 2
if turn = 1 then
    moves forward at 60 RPM for 1 secs
    turns left 120 ° until done
if turn = 2 then
    moves forward at 60 RPM for 1 secs
    turns right 120 ° until done
```

You only have one mBot2? Then you can simulate the communication between two devices by including a sprite in the mBlock stage. The stage is the area on the upper left corner of the mBlock's screen.

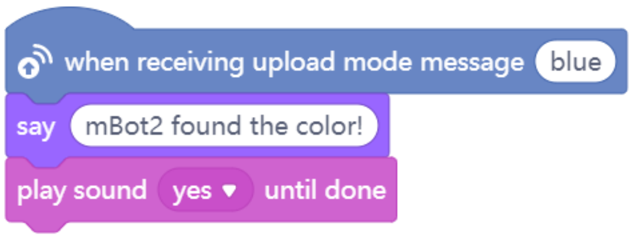
You need to add a new extension to the code blocks for the sprite and the mBot2, called "Upload Mode Broadcast". It allows you to send and receive messages between the sprite and the robot while the robot is in Upload mode, similar to the messages sent via LAN communication.

	Extension	Add
For the sprite	 **Upload Mode Broadcast** By mBlock official By adding this extension, you enable a device to interact with a sprite in the Upload Mode. + Add	• Click on 'Sprites' • Click on 'Extensions' • Click on 'Sprite extensions' • Find the extension labelled "Upload Mode Broadcast" and add it
For mBot2		• Click on 'Devices' • Click on 'Extensions' • Click on 'Device extensions' • Find the extension labelled "Upload Mode Broadcast" and add it (might not be on the first page of the Extensions library)

In this programming example, when mBot2 has found the color, it will send a message to the panda sprite in the stage of mBlock, and the panda will say that the mBot2 has found the color.



```
when button A pressed
forever
    if quad rgb sensor 1 probe (2) R1 detected blue ? or quad rgb sensor 1 probe (3) L1 detected blue ? then
        send upload mode message blue
        stop encoder motor all
        LED all displays ● for 3 secs
        stop all
    else
        if ultrasonic 2 1 distance to an object (cm) < 15 then
            moves backward at 60 RPM for 2 secs
        else
            random_moves
```



```
when receiving upload mode message blue
say mBot2 found the color!
play sound yes until done
```

You can copy the above programming examples, but you can also come up with your own extension. It is then useful to use the following step-by-step plan. Do you have any idea of what you want to make? If so, first discuss with your teacher whether this is feasible.

	Explanation
Step 1: What do you want to do?	• What do you want to program?
Step 2: What do you need?	• What materials do you need to do this?
Step 3: What code blocks do you need?	• How are you going to make the mBot2 drive and communicate? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML)

	• If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well.
Step 4: Testing and implementation	• First version ready? Test it! During the testing round, write down points of improvement • Work on the improvement points until the mBot2 does exactly what you had in mind • Successful? Film the end result and ask your teacher if you can post it on social media with the hashtag #mbot2LAN

4. Wrap up (5 min)

Step 4: Wrap up

Did you succeed in programming the game?

In this lesson, you learned what a LAN connection is and where you might encounter it in everyday life. You know how to set up a LAN connection between different devices. You also learned how to create your own LAN network using the mBot2 and the CyberPi. In addition, you know how to add a random selection to a computer program.

It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think worked out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 8:

mBot2 at your service

Subject: STEAM **Grade(s):** 5th and up

Duration: 45 minutes **Difficulty:** Intermediate

Lesson objectives

By the end of this lesson, students will be able to:

- Make the mBot2 carry out tasks using speech recognition and WIFI connection
- Combine speech recognition with variables for dialogs
- Send messages via cloud messaging
- use Boolean statements with sensor data

Overview

Speech recognition is the technology that converts the spoken word (but also entire sentences) back into text. It is a quite computational task to recognize even a single word just by the variation in frequency. And to be of practical use, speech recognition should work independently from an individual voice: having a high or low voice, slow or fast speaking should not make any difference. Speech recognition is a part of user interaction in consumer electronics (like smartphones, TVs, smart home), but also for assistive systems.
In this activity, you will program a short dialog with the mBot2 acting as a service robot. You can extend this task later to simulate a user dialog for other tasks as well.

Focus

By the end of this lesson, students will know:

- What speech recognition is and how to apply it
- How to make an internet connection using WiFi
- How to use speech recognition and develop a dialog with the user

- Send messages to remote computers via cloud messaging
- Use variables to describe a stage in a process (like ordering, user interaction)

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for Chromebook), or a tablet with the mBlock app installed
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle

Lesson Plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	**1. Warming up** • Speech recognition in everyday life • Getting to know speech recognition with mBot2
10 minutes	**2. Hands-on** • Setting up a WiFi connection • Programming and applying speech recognition • Cloud messaging
25 minutes	**3. Trying out** • Programming mBot2 to become a robot waiter
5 minutes	**4. Wrap-up** • Showtime: show what you did with your robot in a fun, short movie for later discussion • If your teacher allows, share the end result on social media with the hashtag #mBot2WIFI • Reflection: what are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up
(5 min)

Step 1: Warming up

This step consists of two parts:

1. Speech recognition in everyday life
2. Getting to know speech recognition with the mBot2

1. Speech recognition in everyday life

Speech recognition is the technology that can identify words and sentences in the sound of the human voice and translates this back to text. This is a very computational task, since what all the algorithms can look at is a change of frequency over time. But even this differs from person to person – just think of low or high voice, speaking fast or in a dialect. Even the mood can change the "tone" which the same sentence sounds like. The research of decades in computer science and linguistics allows these complex tasks to be incorporated more and more in everyday life.

Some recognition systems require a form of "training", where the user reads specific texts known to the detecting algorithm – adapting a computer model to the individual tone of voice. Other only identify very specific words – they are used in user dialog systems mostly. Some systems can recognize and translate texts regardless of the context.

The CyberPi and the mBot2 can use voice recognition without the need for complex coding. However, since the output is "just" text, the program must analyze it and process further.

Speech recognition is used in consumer electronics like TVs and smartphones or smart homes to give extra comfort for the user – asking for a connection, putting an appointment into the calendar, or having the room temperature and light changed just by telling.

Next to comfort only, speech recognition can be a great help for disabled people or provide extra safety, e.g., a driver in a car can approve a navigation change due to upcoming traffic jams just by voice, instead of interacting with knobs or a screen-based interface.

In this activity, you will make use of some of the features of voice recognition.

2. Get to know speech recognition with the mBot2

The mBot2 offers a speech recognition based on block coding – it is as simple as starting the recognition for s certain time (like 2,3 or 5 seconds) using a single block and retrieving the recognized text as a string by another block. The coding blocks will be explained below.
This voice recognition can understand many different languages and is not restricted to a small set of predefined words. Therefore, it is a very computational task - it is so demanding, that the mBot2 or the CyberPi can't handle it themselves… they need to call for support using the internet. That's why first of all, an Internet connection must be established, and a user account is required as well. Setting up a user account generates a "key" to sign in into the services used (for data usage and privacy see below)
The speech recognition records the audio like you did this in the 3rd activity, and then sends this recording to a very powerful network of computers for processing. They take care of the translation back to text and send the text back to the CyberPi/mBot2. The network of computers is usually referred to as "the cloud". It is not a specific computer anymore but rather a huge cluster of them shifting the computational tasks between them.
You can use the Internet for transmitting data between multiple CyberPis in different networks and locations as well (like in school and at home). This digital service is processed in the cloud as well.

Data usage and privacy note:
Speech recognition requires an internet connection for transferring the recorded audio and sending back the transcribed text. The recorded audio is processed on servers in Europe, the Microsoft Azure cloud, but it will not be stored permanently. For using the services, a "key" is required, that will automatically be generated by creating a user account on Makeblock.com and used internally by the coding blocks. There is no need to manually enter this key anywhere. Apart from an Email and a self-chosen password, no further personal information (like names, pictures, gender…) is mandatory.
The internet connection requires a WiFi access via SSID (that is the name of the wireless network) and password – certificate-based logins are not supported. The SSID and password are stored with the program in plain text, so using a temporary hotspot might be a good idea. Ideally, you can use a smartphone with a flat-rate data plan.

2. Hands-on
(10 min)

Step 2: Hands-on:

This step consists of three parts:

1. Getting acquainted with the different code blocks of the WiFi connection
2. Programming and applying speech recognition
3. Messaging and Communication in LAN and Cloud

1. Setting up a WiFi connection

Open mBlock and connect the CyberPi in the 'Devices' tab. Switch to upload mode. Once connected look for "AI" (this means Artificial Intelligence) in the code blocks and click on it. You will now see the available functions including speech recognition (if they are greyed out, you forgot to switch to upload mode...).
To use speech recognition, you need to connect the CyberPi to the Internet. Below you will see the programming code that you will use in order to have the CyberPi connect to the Internet. As soon as the CyberPi is connected to the Internet you will see this on the display and the LEDs will turn green.
These examples use an event-based approach, so one thread in the program tries to enable the internet connection on start-up. The other threads start the recognition. But for now, it does not check if the Internet connection ban be established before continuing – nor does it give any indication to the user about what is happening except if it was successful. You might want to change this at a later stage when trying out own ideas...

Make sure that you put in the name (SSID) and password for the WiFi network correctly (the picture shows an example!). These are both case sensitive. Connect the mBot2 to your computer with a USB cable and switch on the upload mode. Recreate the programming code from above and try if the code works. A tip is to do this with a hotspot from your cell phone, as a school network often only allows known devices to connect (white-listed MAC addresses).

Make sure that the data plan covers the internet usage! Now connect the CyberPi to your computer.

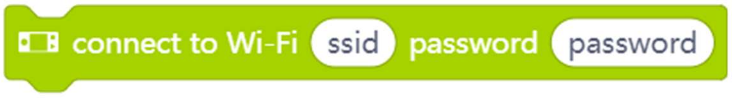
You have now connected the mBot2 to the Internet. You can now communicate with the mBot2 via the cloud. This is what you call Internet of Things (IoT).

If you are using a separate CyberPi to show the status (like the order sent to the kitchen or the delivery process), it must connect to the same WiFi network. This is to make sure that both devices (mBot2-Cyberpi and stand-alone CyberPi) are using the same channel in the WiFi network. The router/access point decides on the channels. This ensures that all devices are on the same channel for data exchange. You can also refer to LAN communication in lesson7.

2. Programming and applying speech recognition

You are now going to program and test the speech recognition. You are going to supplement the programming code in which you let the mBot2 connect to the WiFi with a code for speech recognition.

Code block:



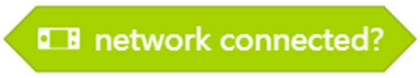
```
connect to Wi-Fi (ssid) password (password)
```

This block starts connecting attempts to the wireless network provided. You need a router or an access point that allows devices to connect with the network and access the Internet. Some schools might only allow known devices – each network adaptor (wired or wireless) has a unique number, the MAC address. If this number is unknown to these secured networks, they refuse the connection even if the password is correct.

The easiest way around this then is a temporary hotspot, e.g. by a smartphone with a flat-rate data plan that is used only for the CyberPi/mBot2 in a lesson (check your school's legal regulations).

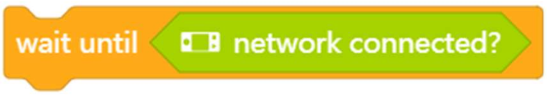
Any blocks following this code block will be executed immediately, so it does not wait for a connection to be made ("non-blocking"). You need to code a waiting routine yourself with the next block:

Code block:



```
<network connected?>
```

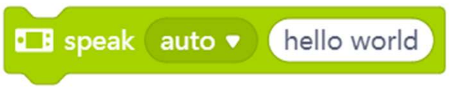
This Boolean block checks if the connection to the network could have been made – if so, it reports a True. You can use this together with a wait block:



```
wait until <network connected?>
```

This combination stops all further code executing in the same thread until a connection is successfully made. You might want to make this more user-friendly at a later stage, informing about connection attempts, indicating a successful connection, or even adding a time-out (like a reboot if no connection can be made for 30 sec.).

Code block:

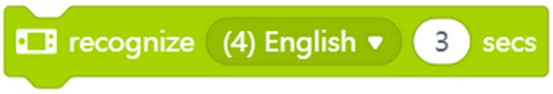


```
speak [auto ▾] (hello world)
```

This code block will be used in an example below – it does the opposite of speech recognition: with this block you can make the CyberPi/mBot2 read out loud a text provided. Here the text is transferred to the cloud service, and the audio file synthesized is sent back. It has an auto-detection on the language used, so the spoken language should be the same as the text uses.

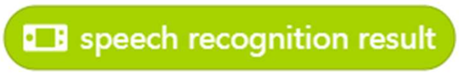
Code block:

Speech recognition is available in 12 different languages. If your native language is not supported, maybe you can use English. This code block starts an audio recording for the specific time and sends it to the cloud service for detection. For the results, you need the following code block:



```
recognize (4) English ▾ 3 secs
```

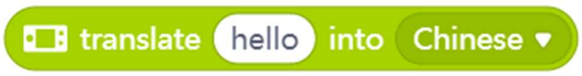
Code block:



```
speech recognition result
```

The results from the speech recognition can be accessed through this reporter block. You can save the result to a variable, in case you need to re-use it, or display it on the screen or process further with some analysis (like "does the text contain the word 'yes'?").

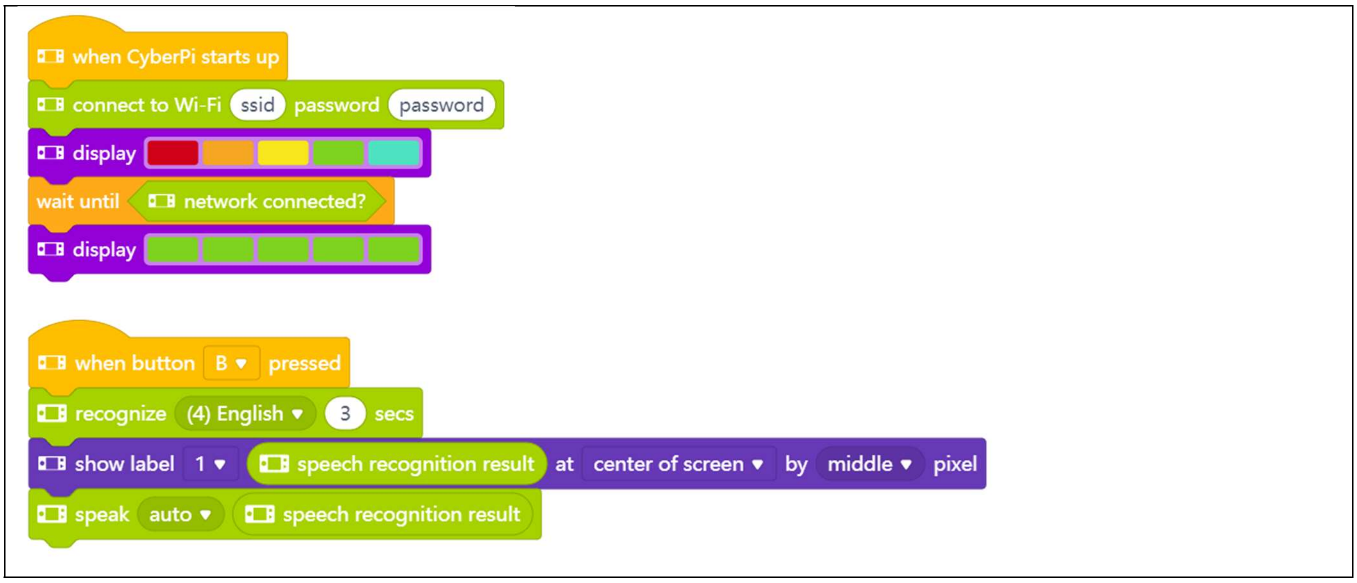
Code block:



```
translate hello into Chinese ▾
```

The AI features even allow for language translation of texts. This code block is a reporter block that allows you to translate the text provided into a (supported) language of your choice. You could use this block together with the previous ones and the knowledge you gained from previous activities and build a voice-operated translator: press a button and translate 5 seconds of speech to a foreign language...you could even try different ones and test your foreign language skills 😉.

In the programming example shown here, the mBot2 is programmed so that when you press button B, the mBot2 recognizes speech for 3 seconds. The result is shown on the CyberPi display and the recognition is spoken. Recreate the programming code below and test it out. Check what you said and what was "understood" and translated back.
Does it work? Then see what else you can do with speech recognition!

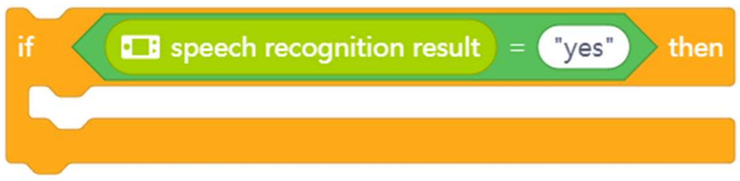

```
when CyberPi starts up
connect to Wi-Fi (ssid) password (password)
display 🟥🟧🟨🟩🟦
wait until <network connected?>
display 🟩🟩🟩🟩🟩

when button B ▾ pressed
recognize (4) English ▾ (3) secs
show label 1 ▾ (speech recognition result) at center of screen ▾ by middle ▾ pixel
speak auto ▾ (speech recognition result)
```

3 Messaging and Communication in LAN and Cloud

You can also use speech recognition to voice-control even multiple mBot2s. This can be done via a LAN connection. In lesson 7 you have already learned about the LAN connection and how to set it up. You use the result of the speech recognition to interact with another mBot2 or a CyberPi. On the "voice-commander" CyberPi, you could display a message like "Shall the robot drive?" ... By pressing a key, you start recognizing text and checking for a yes-or-no answer... If the answer is "yes", then the robot drives 10 cm. The "voice-commander"-CyberPi will send a broadcast message (LAN, like in lesson 7) to the robot.

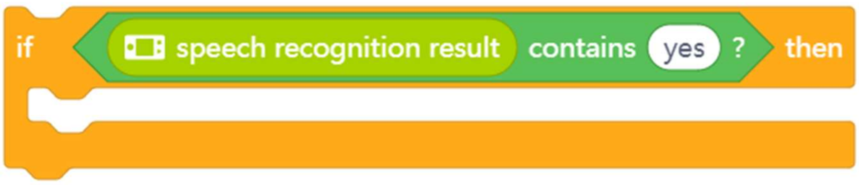
But what happens if you answer "definitely yes" or "yes of course"? Then a direct matching with...


```
if <(speech recognition result) = ("yes")> then

```

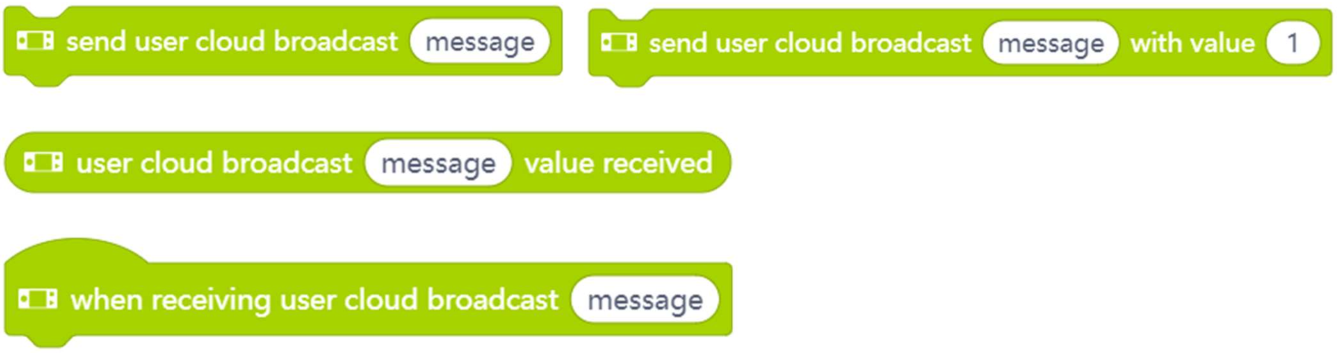
... would not work... the recognition result would be different (string "yes" is not equal to string "definitely yes").

In order to allow a broader set of answers all meaning "yes", you can simply check if the answer-string contains the word "yes":



```
if speech recognition result contains yes ? then
```

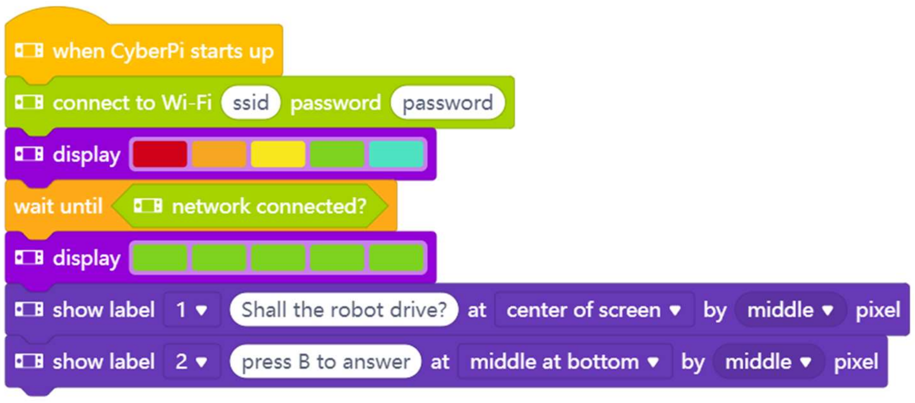
Next to sending a message on the LAN, which requires the devices to be "local" to each other, you can use another cloud service to send messages to devices on remote locations – as long as they have an Internet connection.
These blocks work identical to the ones handling the LAN communication:



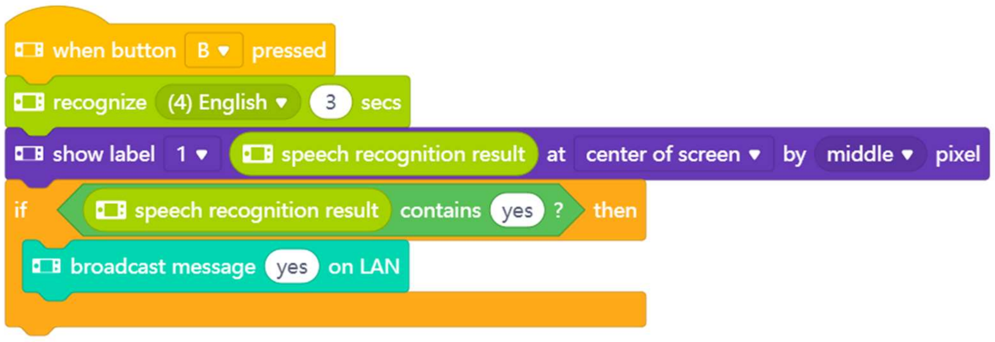
```
send user cloud broadcast message
send user cloud broadcast message with value 1
user cloud broadcast message value received
when receiving user cloud broadcast message
```

To make sure the remote side can communicate only with the devices you intend to, you need to have a Makeblock user account. This will create a "key" for the cloud communication – this key is hidden within the coding blocks and unique to a user. To make 2 remote devices communicate over the Internet, both need to be programmed from the same user account.
The cloud broadcast blocks have the advantage of being able to communicate worldwide, but they are not useful for speedy communication – it might take a few seconds for the message to arrive. The LAN-broadcasts on the other hand are fast but work only locally.

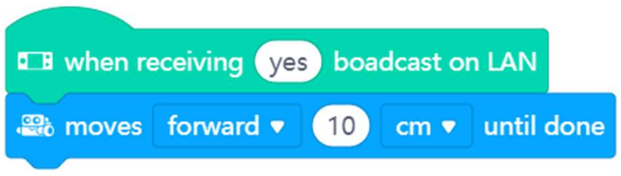
Below you see an example programming code that summarizes the above. Recreate the programming first and try if everything is working with the recognition and communicating between the CyberPi and another mBot2. For this exercise, you need to upload the same program code in upload-mode to both devices.
After you made sure everything works fine, you can modify and extend the example – maybe directional commands? Think of lesson1 and the exercise to navigate the mBot2 manually through a maze… can you do the same with voice-commands?



```
when CyberPi starts up
connect to Wi-Fi (ssid) password (password)
display 🟥🟧🟨🟩🟦
wait until <network connected?>
display 🟩🟩🟩🟩🟩
show label 1 ▾ (Shall the robot drive?) at center of screen ▾ by middle ▾ pixel
show label 2 ▾ (press B to answer) at middle at bottom ▾ by middle ▾ pixel
```



```
when button B ▾ pressed
recognize (4) English ▾ (3) secs
show label 1 ▾ (speech recognition result) at center of screen ▾ by middle ▾ pixel
if <(speech recognition result) contains (yes) ?> then
    broadcast message (yes) on LAN
```



```
when receiving (yes) boadcast on LAN
moves forward ▾ (10) cm ▾ until done
```

3. Trying out
[25 minutes]

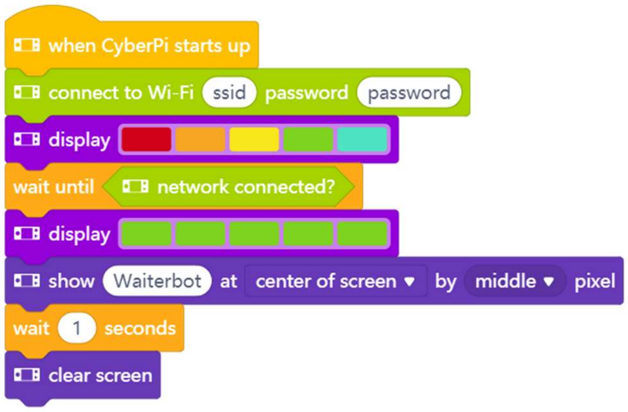
Step 3: Trying out

You now know how to make a WiFi connection, program speech recognition and deploy Cloud or LAN messaging. It is now time to get started yourself. You are going to program a polite service robot – a robot waiter. There are already restaurants in some countries where a robot comes to take the order and brings the food from the kitchen. Take a look at the following video: https://www.youtube.com/watch?v=FFCPKmLAZb4

The robot waiter you are going to make must be able to do a number of things:

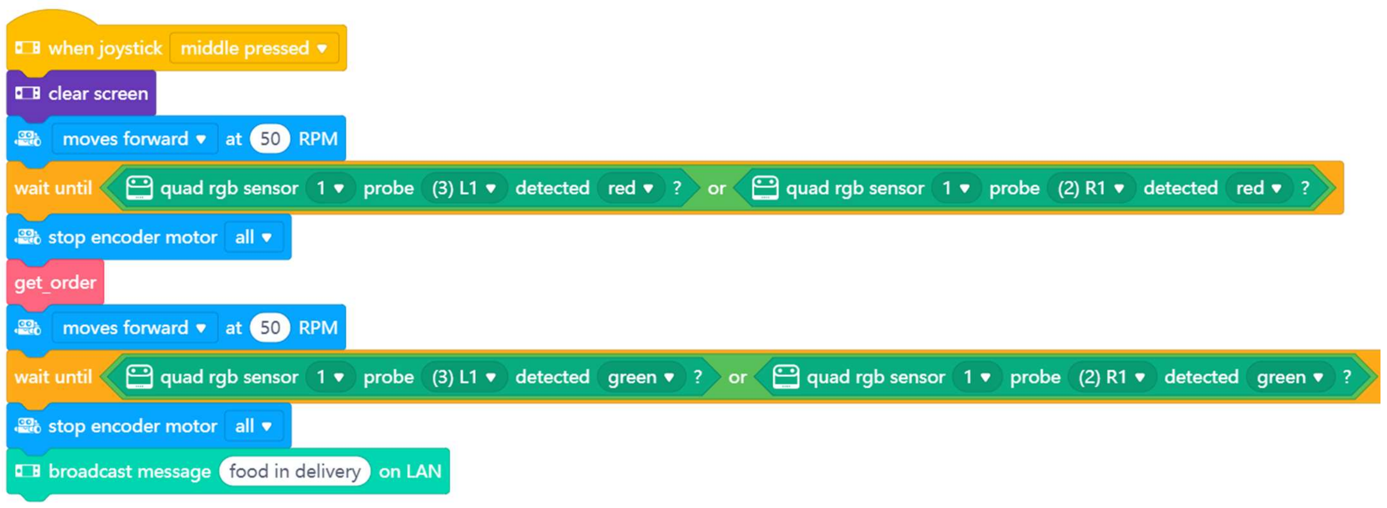
- Drive along a straight line
- Stop at a red area (your table), take an order and send the order to the kitchen
- Additional option: if a second mBot2 or CyberPi is available, then display the orders to the kitchen (so they can start preparing the food already)
- Continue driving until the mBot2 encounters a green surface (the kitchen) and stops to pick up the food from the kitchen
- Optional: if a second mBot2 or CyberPi is available, then give a notification that the food is being delivered

The start-up sequence is nearly identical to the previous examples – remember to put in the correct SSID and password:



```
when CyberPi starts up
connect to Wi-Fi (ssid) password (password)
display 🟥🟧🟨🟩🟦
wait until <network connected?>
display 🟩🟩🟩🟩🟩
show (Waiterbot) at (center of screen ▾) by (middle ▾) pixel
wait (1) seconds
clear screen
```

The next code block is programmed in the same script field under the previous example. In this code block, you instruct the mBot2 to start driving and then stop at a red marker to take the order. After that, the mBot2 drives on to a green mark. The colour markers indicate the table and the kitchen. To start easier, we assume they are in a straight line. If you have everything working, you can extend this further and include the knowledge you gained from previous lessons and make the robot follow a line to the table (the provided map has red and green markers on the track that can be used for this purpose). Lesson 5 explains all about line following…
As an extra addition to voice recognition, staying with the straight line between table and kitchen, you can use cloud or LAN messaging to send the order to another mBot2 or CyberPi to display.

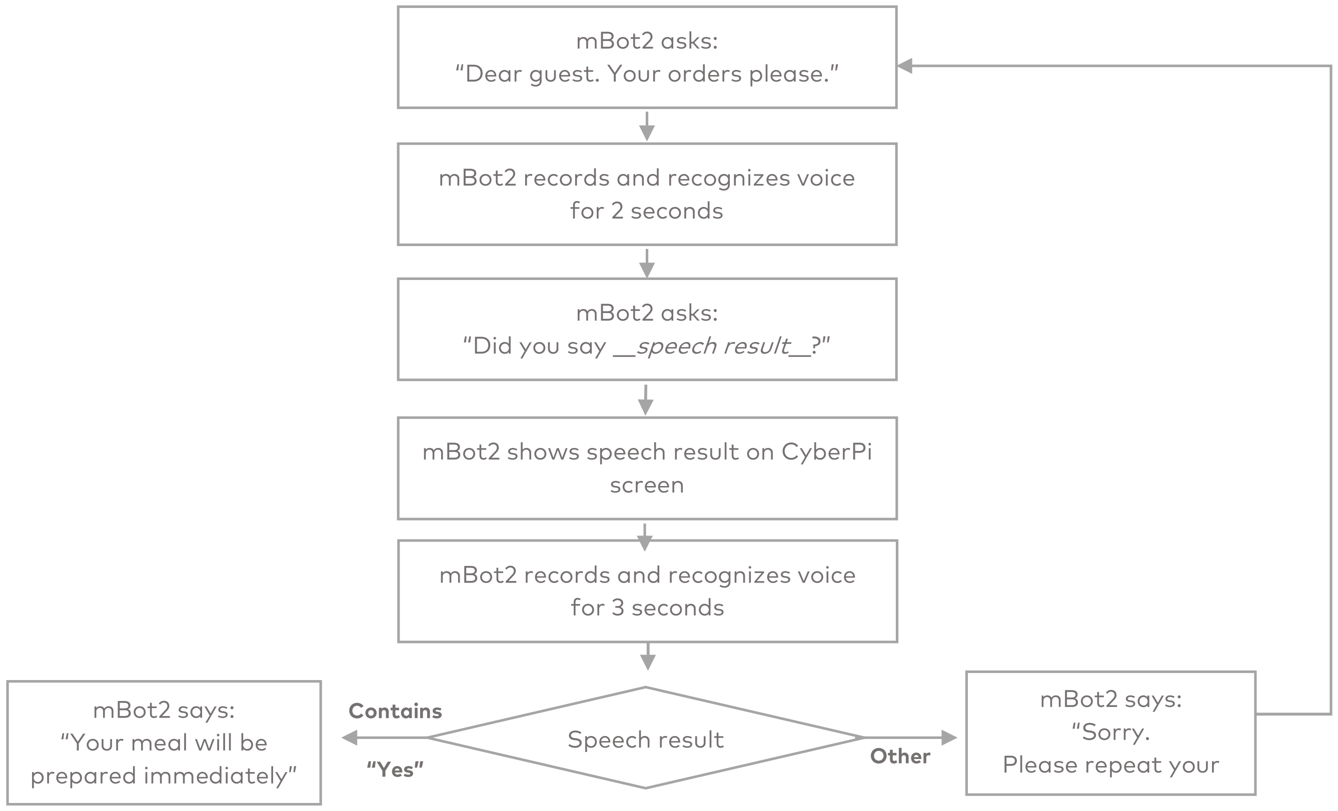


```
when joystick [middle pressed ▾]
clear screen
moves forward ▾ at (50) RPM
wait until < quad rgb sensor [1 ▾] probe [(3) L1 ▾] detected [red ▾] ? > or < quad rgb sensor [1 ▾] probe [(2) R1 ▾] detected [red ▾] ? >
stop encoder motor [all ▾]
get_order
moves forward ▾ at (50) RPM
wait until < quad rgb sensor [1 ▾] probe [(3) L1 ▾] detected [green ▾] ? > or < quad rgb sensor [1 ▾] probe [(2) R1 ▾] detected [green ▾] ? >
stop encoder motor [all ▾]
broadcast message (food in delivery) on LAN
```

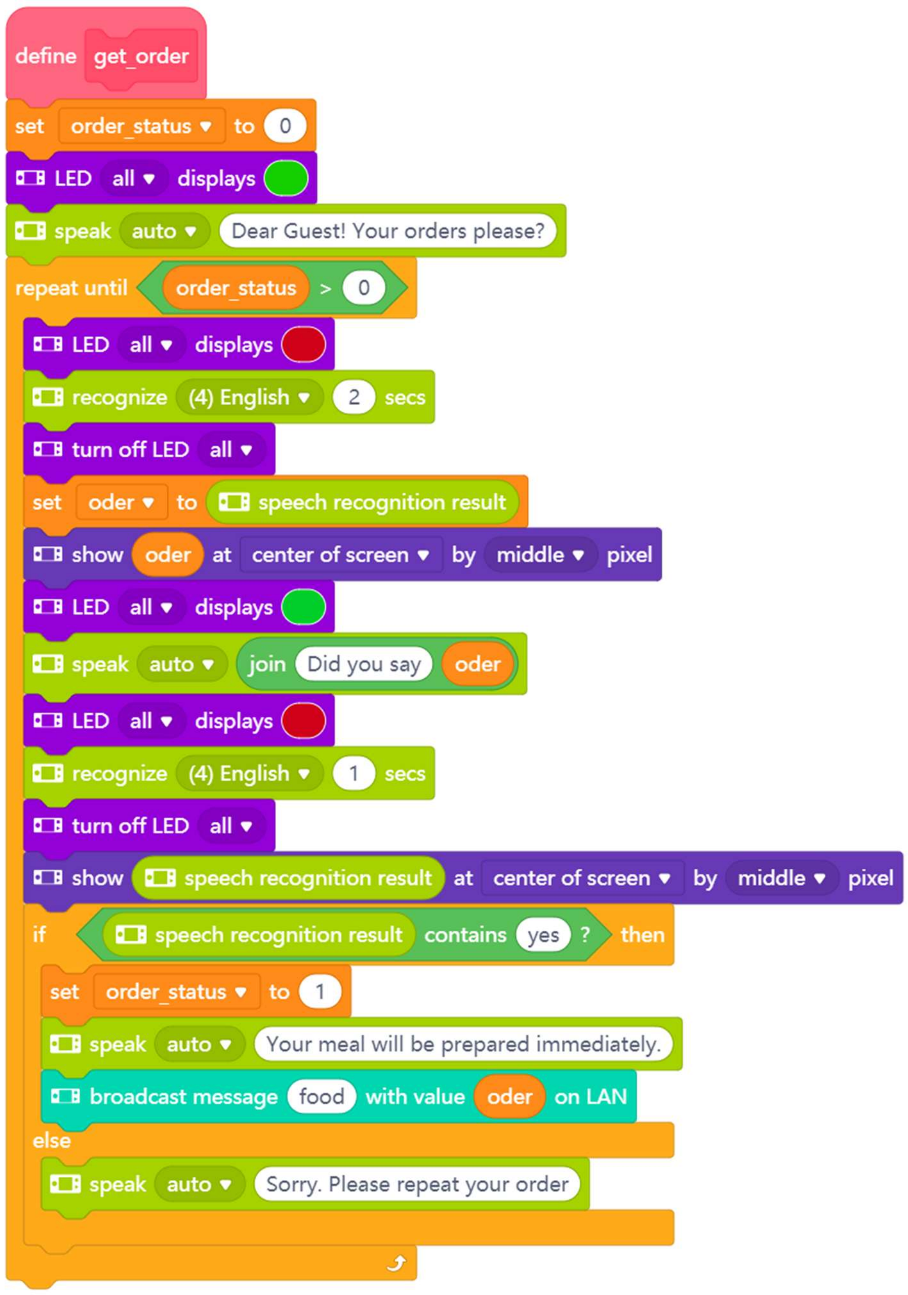
You have to create yourself the block 'get_order' in 'My blocks'. This helps keeping an overview of the main program – so it's just driving to coloured markers with getting the order in between. To actually define what "get_order" means, let's first think about an order process:

1. Ask for the order
2. Recognize the answer
3. Check back that the understood order is correct
4. If yes, proceed (send order to kitchen and end the order sequence)
   If no, ask again.

Checking is a sensible thing to prevent wrong orders. To translate these steps in a structured way, you can use a flow diagram like the following one:


mBot2 asks: "Dear guest. Your orders please."
mBot2 records and recognizes voice for 2 seconds
mBot2 asks: "Did you say __*speech result*__?"
mBot2 shows speech result on CyberPi screen
mBot2 records and recognizes voice for 3 seconds
Speech result
Contains "Yes"
mBot2 says: "Your meal will be prepared immediately"
Other
mBot2 says: "Sorry. Please repeat your

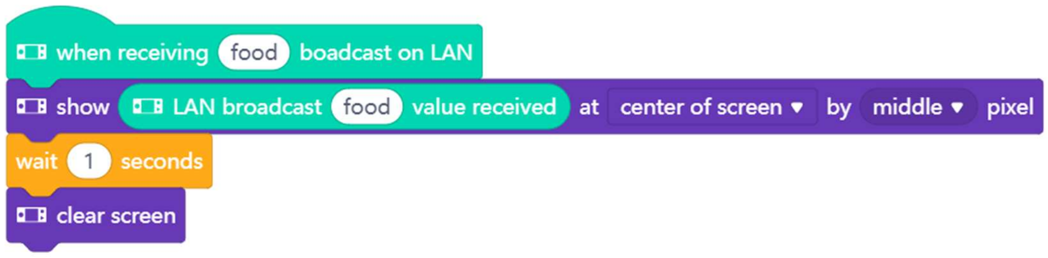
From here, it is much easier to create the necessary coding blocks: you start with the code block 'define order'. You create this code block again in 'My blocks'.
In addition, you need to create two variables. In one variable we will store the recognition result from the order, so if everything was understood correctly, this can be sent to the kitchen (attention: checking back with yes/no – the answer to this would override the previous recognition result in the report block; that's why we need to store the recognized order in a variable).
The second variable keeps track of the entire process: is the order complete (meaning, understood correctly)? If not, the ordering process will start all over again. This second use of the variable reports on process data (stage).



```
define get_order
set order_status ▾ to 0
LED all ▾ displays 🟩
speak auto ▾ Dear Guest! Your orders please?
repeat until order_status > 0
    LED all ▾ displays 🟥
    recognize (4) English ▾ 2 secs
    turn off LED all ▾
    set oder ▾ to speech recognition result
    show oder at center of screen ▾ by middle ▾ pixel
    LED all ▾ displays 🟩
    speak auto ▾ join Did you say oder
    LED all ▾ displays 🟥
    recognize (4) English ▾ 1 secs
    turn off LED all ▾
    show speech recognition result at center of screen ▾ by middle ▾ pixel
    if speech recognition result contains yes ? then
        set order_status ▾ to 1
        speak auto ▾ Your meal will be prepared immediately.
        broadcast message food with value oder on LAN
    else
        speak auto ▾ Sorry. Please repeat your order
```

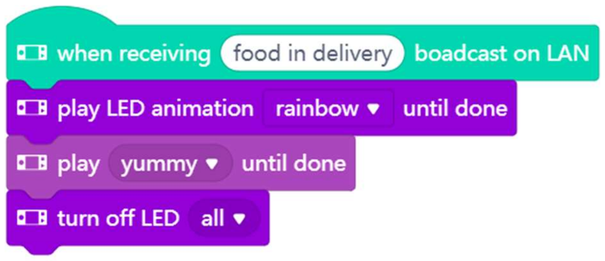
You can use the knowledge from previous lessons to help with the program, e.g. light up LEDs to make the user understand that the robot is about to speak (green) or listen (red), or you can use the screen on the waiter-mBot2 to display data. Use this at checkpoints in your program, so evaluate if your code works and give feedback to the users.
As an extra addition, you can use cloud or LAN messaging to send data to another mBot2. For a better learning experience, there are examples on how to solve specific tasks in this lesson. You can refer to them in case you have problems with your own code or get ideas on how to deal with a specific situation. In general, it is better to get a draft working and refine the code in iterations rather than trying to have everything done and programmed perfectly at once. Start with the driving sequence and instead of having the get_order-Block ready, just make a beep there to show that everything is working so far. Then work on the order process but keep the driving out. This way you can iterate faster without the robot having to drive between the markers every time you want to try your code. as final stages, combine them to solve the given task.

In the programming examples above, you also sent a message to a second mBot2 as an additional option. For example, in the 'order process' you have said that once the customer confirms the order the mBot2 sends a message via cloud or LAN messaging to a second mBot2 or CyberPi (for displaying the incoming orders in the kitchen). Now you have to program the second mBot2 to receive the message.



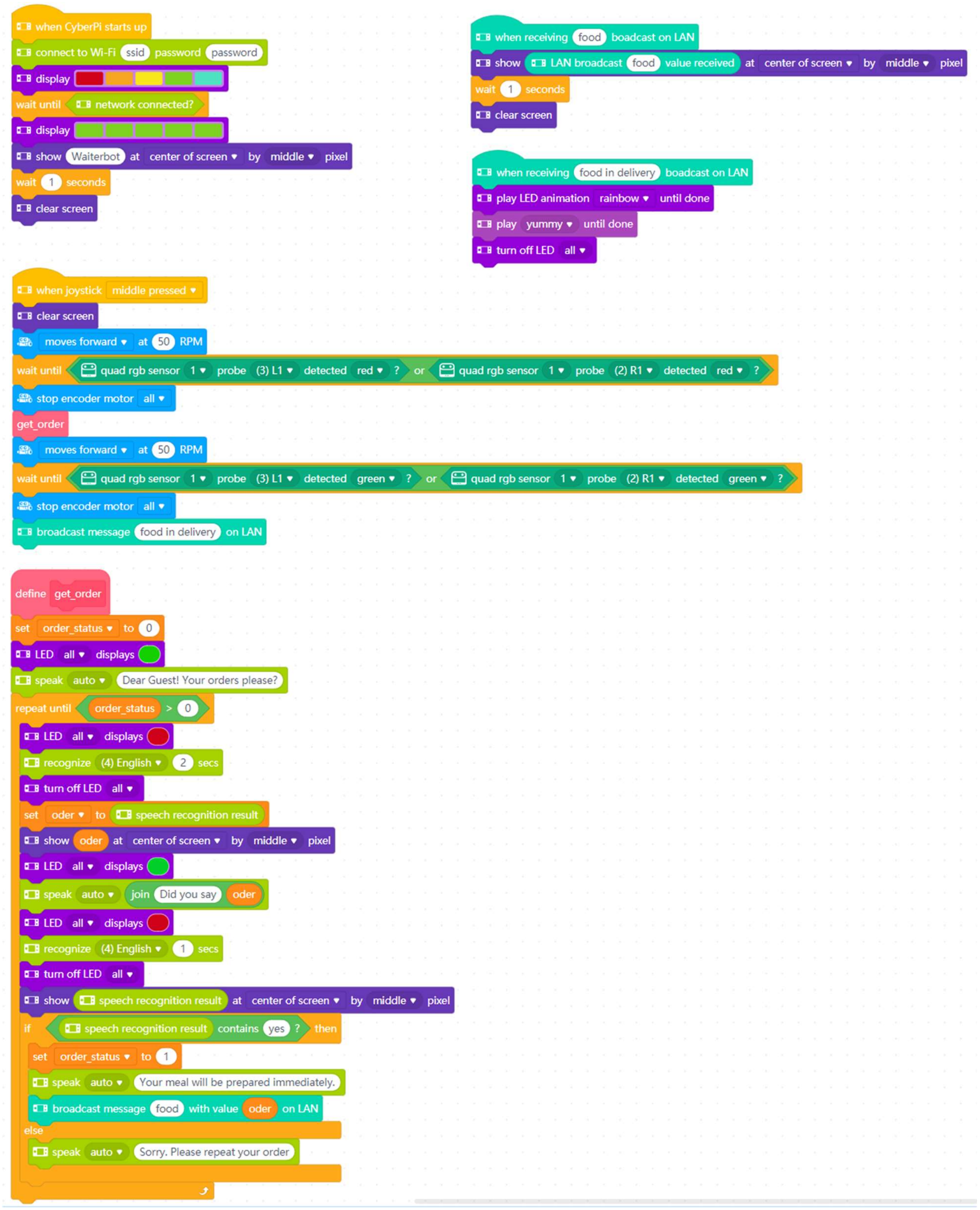
```
when receiving (food) boadcast on LAN
show (LAN broadcast (food) value received) at [center of screen ▾] by [middle ▾] pixel
wait (1) seconds
clear screen
```

In the programming example below, the kitchen receives the order and displays it. Create the programming example below in the same script. Place the example next to the first programming example. This way you keep the script clear.
In the programming example 'controlling a robot' you have programmed that after detecting the green area a cloud message is sent to the customer to inform him that their order is coming. This is also programmed in the same script field next to the other blocks – you could use another CyberPi (as a smart device on the customer's table) to inform about the order status:



```
when receiving (food in delivery) boadcast on LAN
play LED animation [rainbow ▾] until done
play [yummy ▾] until done
turn off LED [all ▾]
```

When you have programmed everything, your script area will look like this:



```
when CyberPi starts up
connect to Wi-Fi (ssid) password (password)
display [🟥🟧🟨🟩🟦]
wait until <network connected?>
display [🟩🟩🟩🟩🟩]
show (Waiterbot) at [center of screen ▾] by [middle ▾] pixel
wait (1) seconds
clear screen
```

```
when receiving (food) boadcast on LAN
show (LAN broadcast (food) value received) at [center of screen ▾] by [middle ▾] pixel
wait (1) seconds
clear screen
```

```
when receiving (food in delivery) boadcast on LAN
play LED animation [rainbow ▾] until done
play [yummy ▾] until done
turn off LED [all ▾]
```

```
when joystick [middle pressed ▾]
clear screen
moves forward ▾ at (50) RPM
wait until <<quad rgb sensor [1 ▾] probe [(3) L1 ▾] detected [red ▾] ?> or <quad rgb sensor [1 ▾] probe [(2) R1 ▾] detected [red ▾] ?>>
stop encoder motor [all ▾]
get_order
moves forward ▾ at (50) RPM
wait until <<quad rgb sensor [1 ▾] probe [(3) L1 ▾] detected [green ▾] ?> or <quad rgb sensor [1 ▾] probe [(2) R1 ▾] detected [green ▾] ?>>
stop encoder motor [all ▾]
broadcast message (food in delivery) on LAN
```

```
define get_order
set [order_status ▾] to (0)
LED [all ▾] displays (🟢)
speak [auto ▾] (Dear Guest! Your orders please?)
repeat until <(order_status) > (0)>
    LED [all ▾] displays (🔴)
    recognize [(4) English ▾] (2) secs
    turn off LED [all ▾]
    set [oder ▾] to (speech recognition result)
    show (oder) at [center of screen ▾] by [middle ▾] pixel
    LED [all ▾] displays (🟢)
    speak [auto ▾] (join (Did you say) (oder))
    LED [all ▾] displays (🔴)
    recognize [(4) English ▾] (1) secs
    turn off LED [all ▾]
    show (speech recognition result) at [center of screen ▾] by [middle ▾] pixel
    if <(speech recognition result) contains (yes) ?> then
        set [order_status ▾] to (1)
        speak [auto ▾] (Your meal will be prepared immediately.)
        broadcast message (food) with value (oder) on LAN
    else
        speak [auto ▾] (Sorry. Please repeat your order)
```

You need to upload this code to all the mBot2s and/or CyberPis you want to use. It can be the same code because they are automatically connected via the LAN or Cloud connection – and the one sending the message won't react upon it.
Create a straight line for the mBot2 to drive along and make sure there is a wide red and green marker. Upload the code and try whether the robot waiter is processing the customers' orders. Of course, there is always room for improvement. Adjust the code where you think it could be better or prettier. Maybe you start with a line-follower that can take the orders? Or you extend this exercise further and have a dedicated waiter mBot2 taking the orders and a delivery mBot2 that will bring the food? You can cooperate with other teams in your class and integrate their robots…

Discuss possible extensions like these in the class when reviewing the progress so far and elaborate a use case with multiple robots – it doesn't need to be a restaurant…

4. Wrap up (5 min)

Step 4: Wrap up
Did you succeed in programming the waiter-bot?
In this lesson, you learned about speech recognition and speech synthesis and how to apply it via a cloud service. You managed to connect the mBot2 to the Internet and so send data locally or even remotely.
Creating a dialog (like for ordering a meal) was part of the exercise, and you learned to filter the text answer for specific trigger words to have a more 'natural' conversation (not expecting the user to answer with specific responses only).
It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think worked out well?
- What could be better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?

Lesson 9: mBot2 in the wild

Subject: STEAM **Grade(s):** 5th and up

Duration: 45 minutes **Difficulty:** beginner

Lesson objectives

By the end of the lesson, students will be able to:

- Integrate Machine Learning into programming the mBot2
- Make multiple technical features of the mBot2 work together
- Establish a communication between the robot and the computer, even though the robot is running independently in upload mode for interaction between physical hardware (robot) and sprite stage of mBlock software

Overview

Machine Learning is a weak form of Artificial Intelligence (AI). Artificial Intelligence refers to devices or machines that mimic human intelligence. Based on collected data, Machine Learning enables computer or robot systems to choose actions based on history of data and probability without the need to explicitly formulate an algorithm for it.

Focus

By the end of the lesson, students will know:

- What Machine Learning is and how to apply it,
- How to make mBot2 work based on conditions,
- How the different technical features of mBot2 work together in an ecosystem,
- Establish a communication protocol between the robot running programs independently and the computer

Pre-lesson Checklist

What do you need?

- PC or laptop (with USB output) with the mBlock software installed, the web version (also for ChromeBook)
- The mBot2 with a CyberPi
- A USB-C cable or Makeblock Bluetooth dongle
- Webcam, on your laptop or external
- Printer (optional) or paper, pencils and markers for drawing or smartphone to display pictures

Lesson Plan

This lesson consists of four steps and takes a total of 45 minutes.

Duration	Contents
5 minutes	1. Warming up • Machine Learning in everyday life • Getting to know the Machine Learning
25 minutes	2. Hands-on • Applying Machine Learning • Machine Learning in combination with the mBot2 in mBlock • Establishing a communication protocol from inside running programs
10 minutes	3. Trying out • Final task: ecosystem of a mouse
5 minutes	4. Wrap-up • Showtime: show your ecosystem in a fun, short movie for later discussion. • If your teacher allows, share the end result on social media with the hashtag #mBot2 • Reflection: What are you most proud of? What would you like to improve about your robot?

Activities

1. Warming up
(5 min)

Step 1: Warming up

This step consists of three parts:

1. Machine Learning in everyday life
2. Getting to know the Machine Learning extension in mBlock
3. Applying Machine Learning

1. Machine Learning in everyday life

Machine Learning works on a statistical basis: training data is provided to the algorithm, like pictures of different objects plus their descriptions. The learning phase then tries to distinguish best between these pictures and match each picture to the correct category. After this phase, the learned patterns are tested against new sets, and the learning can be reinforced. The difference to "classical" sorting programs is that there is no clear indication beforehand on how to decide between the different categories, like you can differentiate between colors.

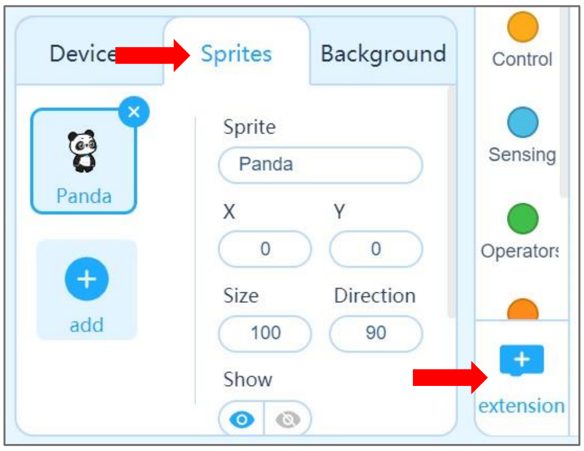
By "learning from past data" the computer program can better adjust to changing situations. Think about facial recognition on your phone: the first times after setting this up, it is harder for the phone to correctly identify you, but over time, the algorithm "learns" to identify your face in different environments as well. Machine Learning is broadly used in image classification (e.g., on social media) to automatically analyze what pictures uploaded show: a cat, a dog, furniture, people etc.

Machine Learning is considered as "weak Artificial Intelligence", since the computer program is not aware of itself nor can "think" like we do – but its current behavior is influenced by data collected and analyzed.

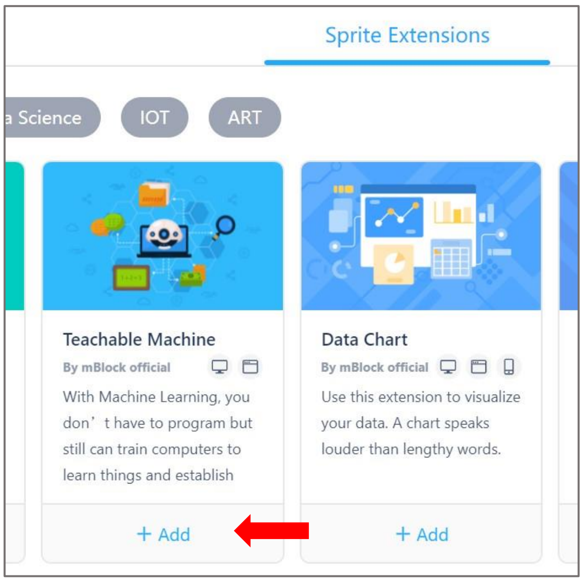
2. Getting to know the Machine Learning extension in mBlock

In mBlock there is a Machine Learning tool. You need to add this tool from the extension library. Proceed as follows:

- Click on 'Sprites'.
- Click on 'extensions' (at the bottom in the middle of the screen).



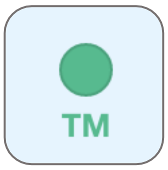
- You will now enter the extension library. At the top of the screen, click 'Sprite Extensions'.
- Find the extension 'Teachable Machine' and add it.

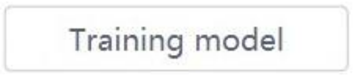


With Machine Learning, you don' t have to program but still can train computers to learn things and establish

Use this extension to visualize your data. A chart speaks louder than lengthy words.

- The extension will now be added to the code blocks. You can recognize the extension by the abbreviation TM in green.
- Make sure you have a working webcam.
- Click on the 'Training model' button.





The extension consists of a training model in which you can 'teach' to differentiate between sets of pictures taken with a webcam. The basic model you see consists of 3 categories. If you want more, click on "build a new model" and choose the number of categories.

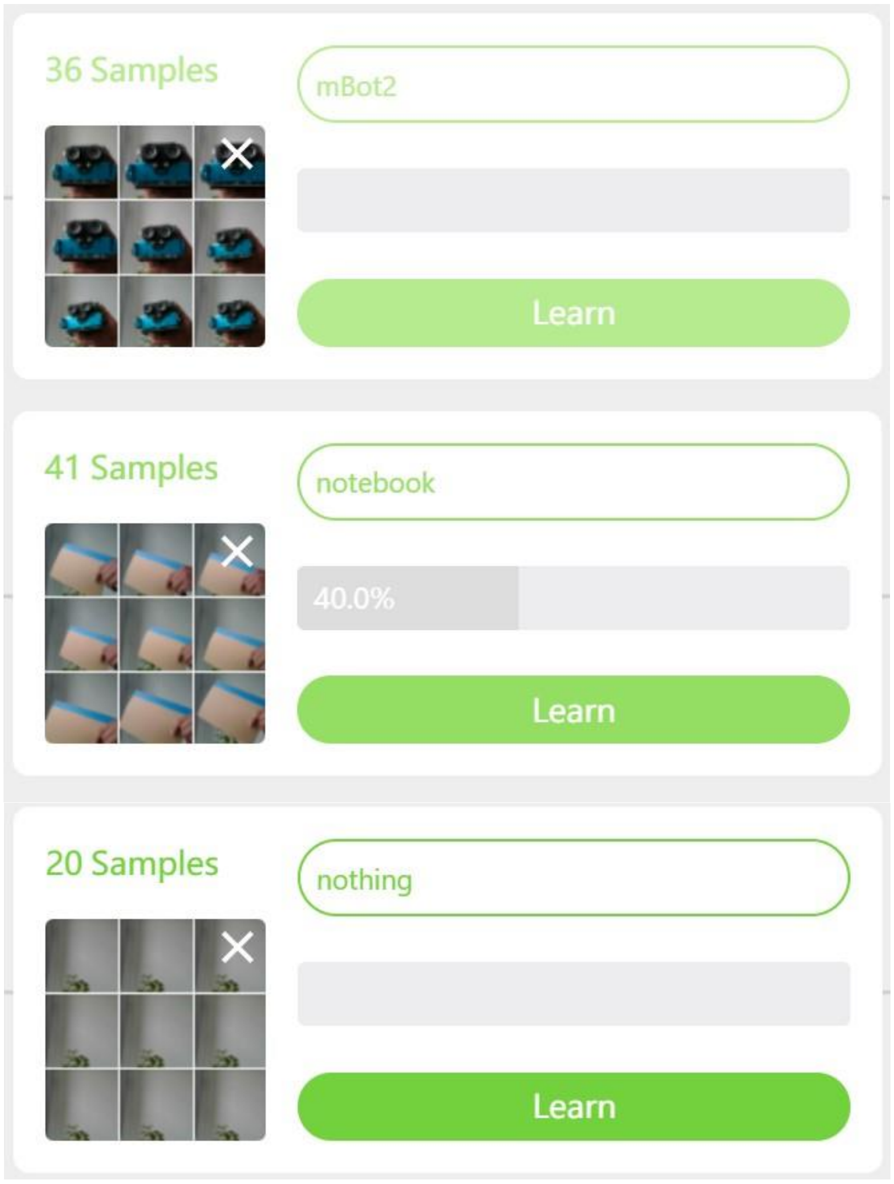
Some advice:

- Think about the number of categories and the recognition beforehand. You cannot add a category later, you would need to start all over again.
- Choose 1 extra category to be used as "background". So, if you want to recognize 2 different animals by pictures you hold in front of the webcam, choose 3 categories: The third one will be the one where there is no picture of an animal shown. Otherwise, you "force" the model to decide e.g., between cat and dog, even if the webcam shows the empty background.
- Background should generally be simple and static (not: classroom with pupils, webcam looking at the pupils etc.) with a clear contrast to the objects to be recognized.

The images will be analyzed just on the computer and the result can be stored. The pictures themselves will not be kept! So, if you save the program file in mBlock and reopen it later, you won't see the original pictures the model was trained on. You can always add new pictures to existing categories, but not delete single ones (only all pictures and learnings from a category).

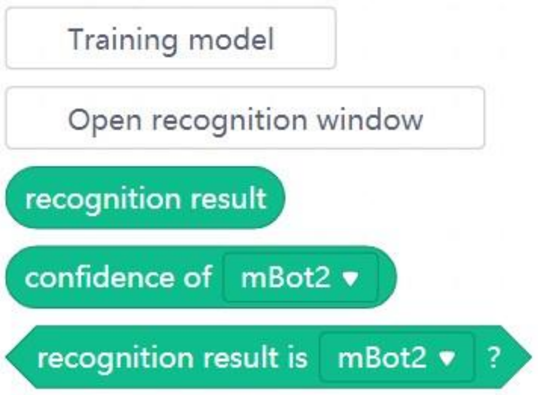
You can directly 'teach' the categories using images taken with the camera. You will use the standard 3 categories for this exercise:

1. You can start with the mBot2. Place it on the desk and align the camera to get a good view – or hold it steady in your hand if you use a laptop with a built-in camera. Then click the 'Learn' button. Hold the button until the camera has recorded at least 20 samples. Above the mosaic of samples, you can see the number of samples stored for this category. Give this category a name, for example 'mBot2'.
2. Now choose something else, e.g., a notebook, hold it in front of the camera and click on the 'Learn' button. Hold the button until the camera has recorded at least 20 samples. Give this category a different name, for example "notebook".
3. Now record the "background". This should be used whenever none of the other objects are presented to the camera. Click and hold the 'Learn' button until the camera has recorded at least 20 samples. Give this category a different name, for example "background".



You have now 'trained' the Machine Learning different objects (mBot2, notebook and background) that the algorithm will now recognize in the camera picture. For every category you can see a bar measuring a percentage. This represents the confidence that the algorithm has in the recognition result. You can increase the percentage of confidence by increasing the number of samples and doing some slight variations like changing the angle of the object. This will help the camera recognize the objects even better and faster.
You can test the recognition live without the need for any coding at this stage. Add pictures until the recognition is fast and accurate (not switching between results and showing a high percentage >90%). Are you satisfied? Then click on 'Use the model'. You will now return to the screen for coding on sprite level.

There are three new code blocks now. One block will report the recognition result (as you typed it), the second one is the confidence for a given category and the third one is the Boolean block for conditional statements. It lets you choose the category by a drop-down menu. If you click on one of the code blocks, a new window will appear. This is the recognition window, which displays the camera image and the recognition result. Now your webcam functions as a smart sensor, identifying different objects or pictures.



2. Hands-on (25 min)

Step 2: Hands-on

This step consists of three parts:

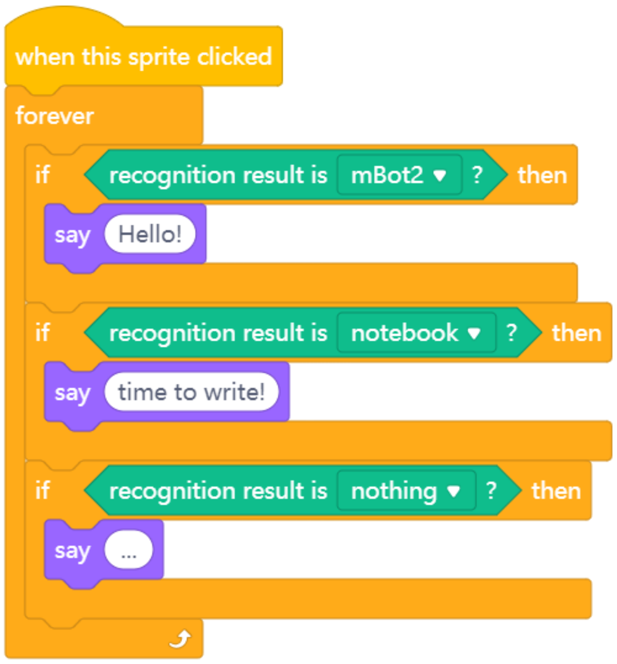
1. Applying Machine Learning
2. Machine Learning in combination with the mBot2
3. Establishing a LAN connection

1. Applying Machine Learning

You have 'taught' two different objects, the mBot2 and a notebook, plus the background to the Machine Learning model. Now you can use the output of the recognition like any other sensor blocks in your program code. You can use conditional statements to differentiate between the objects and assign a certain action separately to each result.

You are now going to try to make the sprite (in this case the panda) talk when the webcam recognizes a learned object.

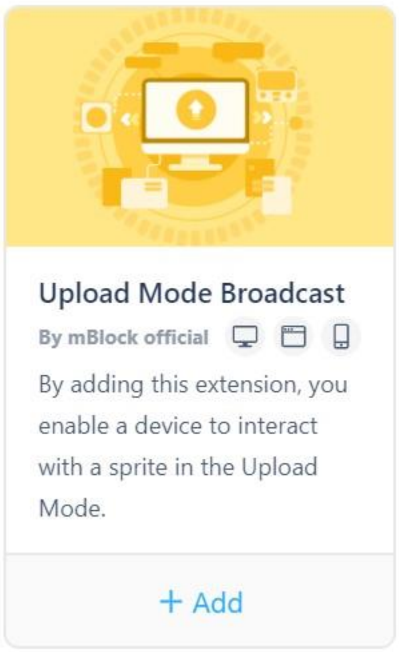
On the right you see a programming example. Reproduce the programming example and see what the panda does when the webcam recognizes your learned objects.
Was it successful? Then investigate what other possibilities there are. Can you make the panda jump or walk?



```
when this sprite clicked
forever
    if recognition result is mBot2 ? then
        say Hello!
    if recognition result is notebook ? then
        say time to write!
    if recognition result is nothing ? then
        say ...
```

2. Machine Learning in combination with the mBot2

Now you know how to make use of the Machine Learning extension for image recognition and control a sprite with it, for example the panda. You are now going to learn how to communicate from the computer to the mBot2 running its own program. For this the mBot2 needs to stay "connected" to the computer – this is done via USB cable or the Makeblock Bluetooth Dongle. The dongle allows the robot to move independently from the computer; if you use a USB-cable, make sure you only perform slight movements of the mBot2, like small turns and distances.
With this connection in place, the sprite (in this case the panda) can send data to the mBot2. The mBot2 receives this data and the program running on the robot can react to it.
You need to add a new extension to the code blocks for the sprite and the mBot2, called "Upload Mode Broadcast".

	Extension	Add
For the sprite	 Upload Mode Broadcast By mBlock official By adding this extension, you enable a device to interact with a sprite in the Upload Mode. + Add	• Click on 'Sprites' • Click on 'Extensions' • Click on 'Sprite extensions' • Find the extension labelled "Upload Mode Broadcast" and add it
For mBot2		• Click on 'Devices' • Click on 'Extensions' • Click on 'Device extensions' • Find the extension labelled "Upload Mode Broadcast" and add it (might not be on the first page of the Extensions library)

In this way, the extension will be added both for the mBot2 and the sprite. In mBlock it appears in dark blue. Click on the new extension and see the different code blocks associated with it. Basically, the extension adds a communication protocol. It allows you to send and receive messages between the sprite and the robot while the robot is in Upload mode.

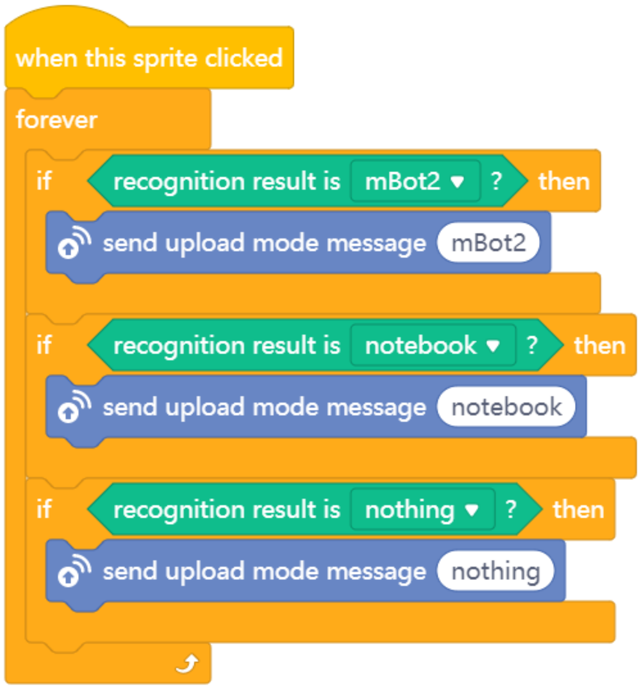
3. Establishing the connection with the mBot2

You have now installed all the necessary extensions for the hardware and software to be able to communicate with each other. Now you will program the mBot2 to be controlled by Machine Learning. The idea is that the mBot2 will perform actions based on recognition results. You programmed a similar task earlier just for the sprite (mBot2, notebook, and background).

Now, the recognition results must be sent and received over the connection between the Bot2 and the computer (either cable or wirelessly via Bluetooth Dongle). The receiving code uses events, so the corresponding code is executed, once the message is received. With events there is no need to check for a sensor or message in a loop, making the code more efficient and easier to understand.

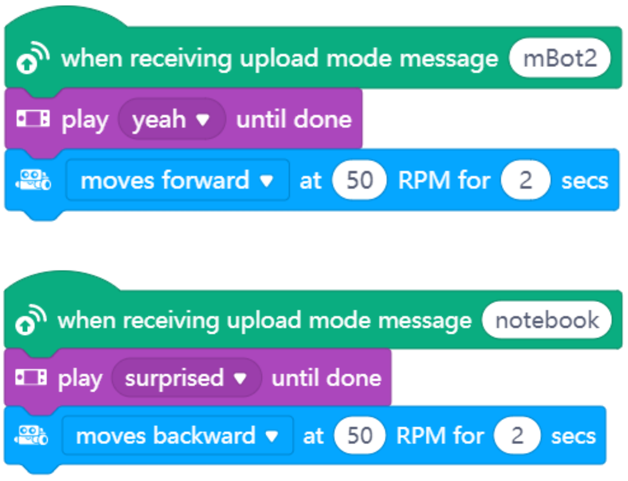
You need to create two computer programs – one for the device (mBot2) and one for the Sprite that uses the Machine Learning add-on:

Sprite program



```
when this sprite clicked
forever
    if recognition result is mBot2 ? then
        send upload mode message mBot2
    if recognition result is notebook ? then
        send upload mode message notebook
    if recognition result is nothing ? then
        send upload mode message nothing
```

mBot2 program



```
when receiving upload mode message mBot2
play yeah until done
moves forward at 50 RPM for 2 secs

when receiving upload mode message notebook
play surprised until done
moves backward at 50 RPM for 2 secs
```

After reprogramming the above code blocks, make sure your computer stays connected to the mBot2 if you use a cable (don't disconnect the cable). Ideally, they would be connected via Bluetooth using the Makeblock Bluetooth Dongle, because this gives the mBot2 more flexibility in driving around.

If you use the dongle:	Install the dongle on the computer and turn on the mBot2. You need to pair the mBot2 with the Bluetooth Dongle. You do this by clicking on the button on the dongle. The LED will start flashing and the dongle tries to connect to the closest Makeblock controller (like mBot2 and others). In a classroom situation, make sure this is done team by team, to ensure the corresponding match between Dongle and mBot2. Once the pairing is done, you will hear a beeping sound from the mBot2. This means that the computer and the mBot2 have made a wireless connection.

The Makeblock Bluetooth Dongle will remember the last paired device and instantly reconnects.

In case you don't have the Bluetooth dongle from Makeblock, you can verify the compatibility of the Bluetooth in your computer via the following link: https://www.yuque.com/makeblock-help-center-en/cyberpi/bluetooth-compatibility

Now switch mBlock to Upload Mode and connect the mBot2 to mBlock. Click the 'Connect' button and upload the computer program. You can now start testing!

3. Trying out (25 min)

Step 3: Trying out

You have all the knowledge and skills to use the mBot2 in any situation. Now you are going to work with the ecosystem of a little mouse. A mouse lives in nature where it must look for food and water. There are also many dangers, such as predators who feed on mice.

In this assignment, you will program the mBot2 to act as a little mouse that searches for food and flees from predators.

Find a picture of a flying eagle and some corn. The corn is the food of the little mouse. The eagle is the hunter. You can print these images or use your smartphone to display them to the webcam. You can also hand-draw pictures for food and predator if you like. Just use the same method you decided for 'teaching' them to your algorithm using the Machine Learning extension and for testing the program.

Program the mBot2 so that if you show the corn to the camera, the mBot2 will move forward (if you use the cable, make a small movement), but if you show the image of the eagle, it moves backwards.

You can also add more code: if the little mouse sees the food, it says "yummy" and when it sees the eagle, we can hear its heartbeat.

When creating the computer program for this assignment, it is helpful to use the following step-by-step plan.

	Explanation
Step 1: What do you want to do?	• What do you want to program?
Step 2: What do you need?	• What do you need in addition to the mBot2?
Step 3: What code blocks do you need?	• How are you going to make the mBot2 move? • What code blocks will you use? • Make a brief description on how your program works (pseudocode/natural language, flowchart or UML) • If you need further explanation, you can discuss with your fellow students, the teacher, or do a research on the topic. There is help available for every coding block in mBlock as well.

Step 4: Testing and implementation	• First version ready? Test it! During the testing round, write down points of improvement • Work on the improvement points until the mBot2 does exactly what you had in mind • Successful? Film the end result and ask your teacher if you can post it on social media with the hashtag #mbot2

4. Wrap-up (5 min)

Step 4: Wrap-up

Were you able to recreate the mouse ecosystem?

In this lesson, you learned about Machine Learning and how to apply it. You also know how to use a communication (and why preferably wireless) between a sprite and the mBot2, and to make use of it to exchange data between the two. In addition, you have made various technical features of the mBot2 work together.

It is now time for a brief reflection. Think on your own and discuss with the group:

- What do you think turned out well?
- What could be even better?
- Which parts of the lesson did you find easy and which did you find more difficult?
- What would you like more explanation about?
- Who could help you with that?