Topic 6: Challenge Option 2: Building and Testing a Robot Prototype

FarmingBot

Learning Objectives

In this Topic you will work on the practical part of your challenge. It will implement a Prototype of a Robot to solve a concrete problem in Agriculture called **"FarmingBot"** This challenge fits perfect for you if you are interested in technology and would like to tinker with something.

Table of Content

Lesson 6.0 Prepare for challenge 2

You will be guided through your Challenge

To develop your robot solution, you can orient yourself by the standardized process model that was briefly introduced in chapter 4. As you will focus on Challengfe#2 **Building & Testing a robot** you will start in the **phase 3** (Implement)and go on to the Phase 4 (Testing). In each phase you will be asked some guiding questions to ensure that all aspects are considered. To build your **Robot** you have then to follow the Instructions and the recommendations described in this lessons. In this challenge You need not to work on the other phases (1,2,5).

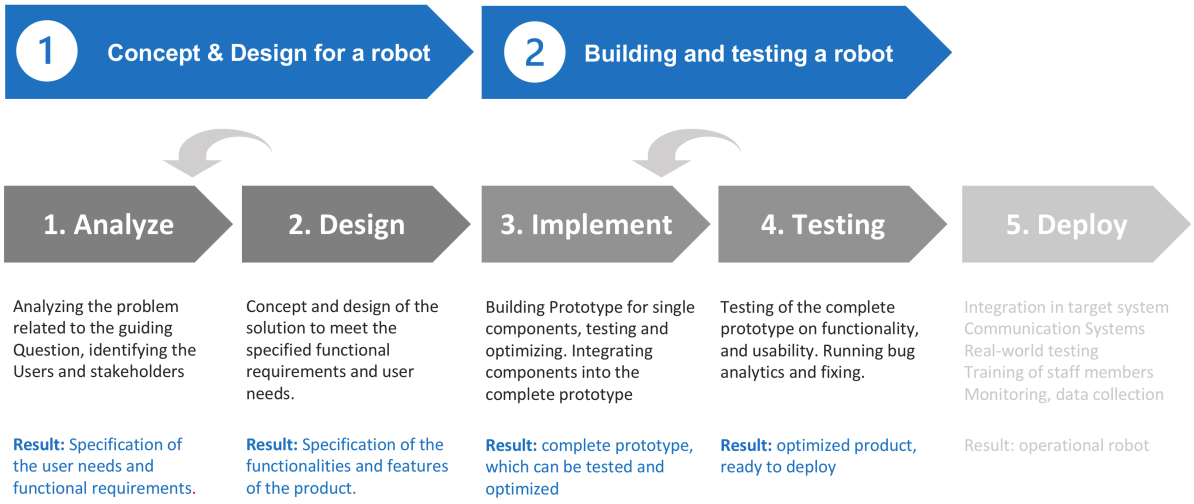


Image: Sequential development process for a robotic solution, based on a software development process, Dickel 2023 based on: Alavandhar, J. Nikiforova, O. (2017) https://www.researchgate.net/figure/Sequential-development-process_fig1_318181157

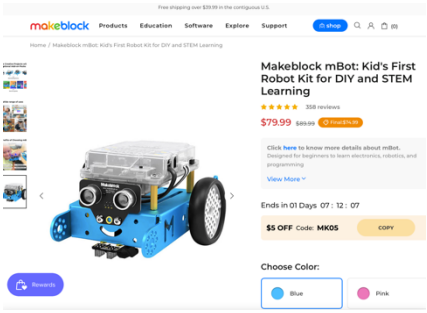
Lesson 6.1 Start your Challenge: set up your robot workbench

In this Engineering Challenge, you will build a small robot that performs some simple tasks that are key skills for autonomous robots in agriculture. You should start by purchasing the base robot. First, you will assemble this robot, explore it, program some basic functions with a simple non-coding application, and perform some tasks to complete the challenge. For all these tasks you will use the mBot from Makeblock and a web application from Makeblock.

The mBot is an easy-to-build and programmable robot kit for DIY and STEM projects for all ages, and an all-in-one educational robot kit for beginners. You already know the mBot and some of its basic functions from **Topic3**.

Figure: The mBot is available in two color themes, Source: Makeblock
https://www.makeblock.com/products/buy-mbot?_pos=1&_sid=5759897d5&_ss=r&variant=37655278420167 retrieved 12.12.2023

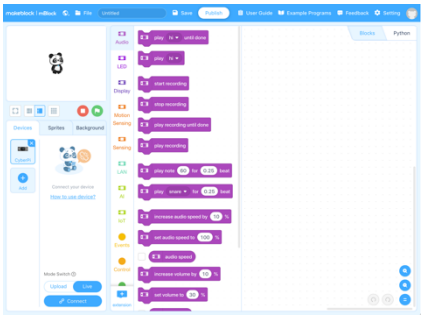
Purchase the mBot Building Kit from the Makeblock Store



So you should first purchase the **basic kit of the mBo**t form Makeblock. You can purchase the mBot online in stores with technical equipment. you can purchase an mBot directly from the Makeblock store. Here you find a link to the **mBot 1 in the Makeblock Store**, the first (and currently cheapest) bot building kit of Makeblock

Link to mBot 1 in the makeblock store: **https://www.makeblock.com/products/buy-mbot?_pos=1&_sid=5759897d5&_ss=r&variant=37655278420167**

Set up Makeblock development tool mBlock

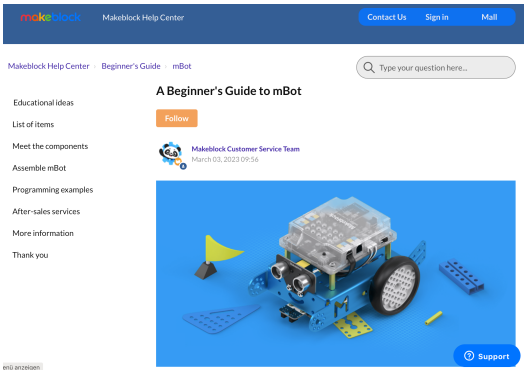


This is your robot programming workbench: Here you can program your mBot. Don't worry, you don't have to write a single line of code if you don't want to. With this tool, you can program your bot completely with blocks, which you simply assemble and then test on your mBot. Do you prefer real programming? That's possible too: switch to Python and get started. The best thing to do is to register and get familiar with the platform.

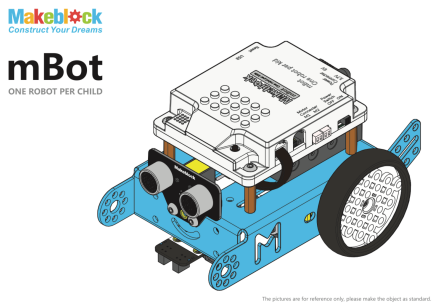
Link to mBlock developer web interface: **https://ide.mblock.cc/**

What you will get for Support:
To master your challenge, you need to know, how a robot works in terms of hardware and its software. You will get the first instructions directly in this Topic. Later in this course you may need some more support in different areas. In such cases, please access this additional supporting tutorials and technical Information as follows. It's recommended to download or bookmark these materials and keep them for later.

Getting started with the Makeblock Beginner's Guide to mBot
The fastest way to get started with your mBot is to access the **Makeblock Beginner's Guide**: Here you are guided step by step through the assembly process and the first operation with your mBot. Here is the downloadlink for the **Makeblock Beginner's Guide to mBot (**best choice for beginners)
https://support.makeblock.com/hc/en-us/articles/12822859943959-A-Beginner-s-Guide-to-mBot



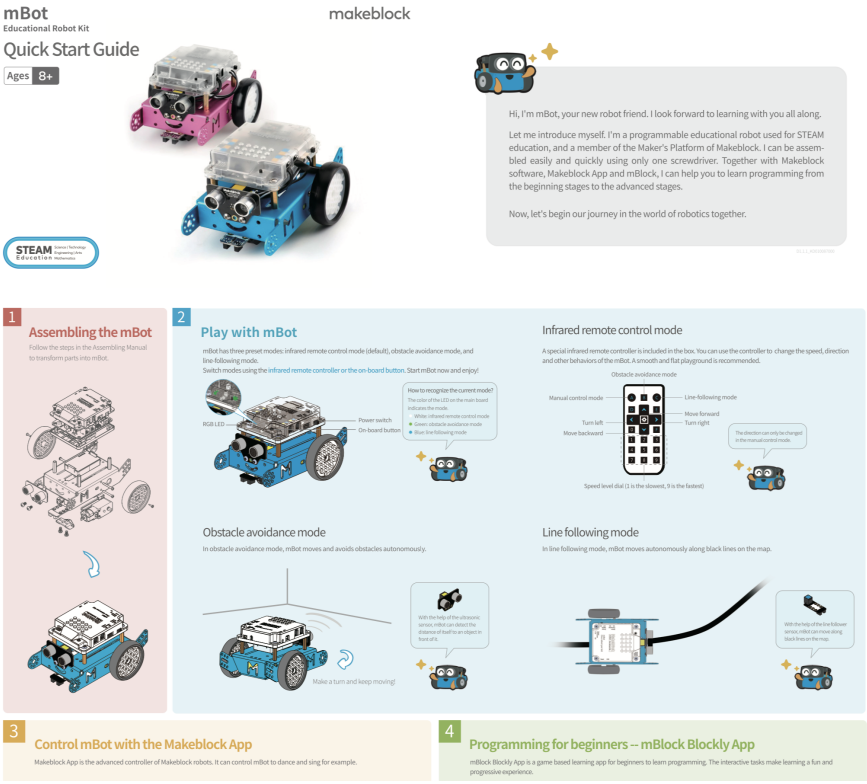
mBot Assembly guide download:
Has your new mBot arrived and need help assembling it? This is the basic document you will need to get started with the hardware. Here you find the **complete parts list, the assembly guide** and the **wiring instructions** for your mBot. It provides also **further information** about batteries, remote control and how to get started playing with your mBot:
Here is the downloadlink to the complete mBot assembly guide:
http://download.makeblock.com/mBot-V1.1_Blue_STD_Instruction_EN_D1.2.1_7.40.4100_Print.pdf



mBlock App on your tablet:
Here is the link if you want to use the mBlock App on your tablet:
https://support.makeblock.com/hc/en-us/articles/12993202671383-Program-mBot-with-the-mBlock-App

mBot Quickstart instructions:

The **mBot Quickstart instruction** tutorial will give you a quick overview of the main functions and the two modes you can set mBot in: avoiding obstacles and line-following.



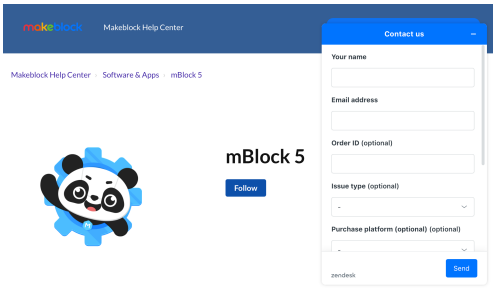
http://cdnlab.makeblock.com/mBot%20Quick%20Start%20Guide%20.pdf

Support from Makeblock:

If have any problems in hardware or software programming? You are not alone! The Make block support really works. You will get answers from friendly experts who wants to help you. This is the Link to the support page you should bookmark first.

Here is the link to the Support website from Makeblock, you should bookmark for any case:

https://support.makeblock.com/hc/en-us/sections/360001829013-mBlock-5



After you have finished this Lesson:

Now that you have set up your robot development environment, you are ready to start Lesson 6.2, which will provide you with the briefing for Challenge #2.

Lesson 6.2: Briefing for challenge # 2 – the FarmingBot

Now it´s time to learn more about the **Challenge**. Therefore, you need some Information about the purpose of the robot, you are working on. First: Why are Farming Robots important for our society? Robotic agriculture can significantly contribute to achieving the **UN-Sustainable Development Goal 2** (SDG) of **ending hunger** and promoting global food security.

Figure: Farmer with a remote control of a farmingbot (generated with Leonardo.ai)

Goal 2 is about creating a world free of hunger by 2030.The global issue of hunger and food insecurity has shown an alarming increase since 2015, a trend exacerbated by a combination of factors including the pandemic, conflict, climate change, and deepening inequalities.
https://www.un.org/sustainabledevelopment/hunger/#:~:text=Goal%202%20is%20about%20creating,climate%20change%2C%20and%20deepening%20inequalities

How can robotic agricultural production help achieve SDG2?
Achieving these goals will also require producers along the production chain to help grow, weed, harvest, transport, and process. There are three main ways that agricultural producers can use robots to help achieve SDG2:

1. **More efficient use of resources:** Precision farming techniques supported by robots enable more efficient use of resources such as water, fertilizers, and pesticides, leading to more sustainable production and reduced environmental impact.

2. **Compensate for labor shortages:** robots can compensate for labor shortages by automating tasks like sowing, harvesting, and weed control, which boosts productivity and increases the availability of food.

3. **Development of effective farming methods**: innovations in robotic agriculture contribute to the development of farming methods that are effective under various climatic conditions and on different soil types, thereby strengthening the resilience of food production to climate change.

Example

The image below shows an example: Twisted Fields, a research farm in California, recently unveiled its robot to the public, providing a simple but interesting look at how an agricultural robot is developed from the beginning.

https://hackaday.com/2021/03/05/a-new-open-source-farming-robot-takes-shape/

Now let's look again at your small FramingBot: As you already know, you robot should be able to run tasks, needed for agriculture robots. But what are these abilities? To answer this question, you must do some research about **robot supported farming.** To get a first impression, please watch the following two Videos:

Activity 1: watch the following video: Meet Robot One - Chemical Free weeding with a Smart Agricultural Robot. Pixelfarming Robotics, 2024, (3:40 to watch)

Link to the video: https://www.youtube.com/watch?v=D3fVdvA0cAs

Text of the video description:

"Robot One is an agricultural robot designed for smart farming. Equipped with 14 advanced depth sensing camera's and dual GPS antenna's it is ideal for large-scale and biodiverse environments. At Pixelfarming Robotics, we understand that every cultivation has its own set of challenges and

requirements. That's why we developed our Onboarding Program to help farmers and growers get started with Robot One and transforming to a sustainable business. For more information go to https://pixelfarmingrobotics.com/robo...

Manual weeding is a time-consuming, labor intensive and physically demanding task. A skilled workforce incurs substantial labor costs and is hard to find these days. To reduce manual labor you can use a Farming Robot (in this video called "Robot One"), to remove weeds with a automated process (called 'Scan and Act'). You start your season by sowing your crop as you normally would, and keep the first 100 meters, or about 500 individual plants, weed-free and in optimal condition. This is reference for the Farming Bot, as it learns from your example. When scanning the soil, it sees what plants you removed and what plants you keep as your crop. Just as the plants grow over the course of the season, the Farming Bot also understands your plant variations. The robot generates a GPS map of individual plant locations and updates this map every run. You can decide whether to use streamers for row spreading, weed control or lasers for pinpoint spreading.

The Farming Bot operates the following steps:

Step 1: You sow your crop as you normally would.
Step 2: You keep a small reference plot in optimal condition.
Step 3: Robot One, learns from your example.
Step 4: Robot One, Scans the soil.
Step 5: You decide what tool to use and what action to take.
Step 6: Let Robot One, do the rest.

Activity 2: watch this Video: Insiders Session, live from Pixelfarming Robotics HQ
Please watch the video from 08:00 to 10:00 min

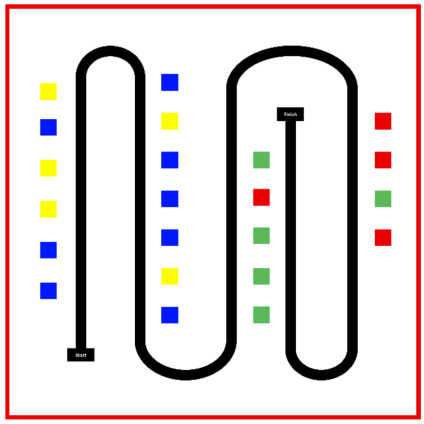
Link to the video: https://youtu.be/19DZhCk4pzc?t=487

Text of the video description:

In this Video sequence (minute 08:00 – 10:00) you can learn some of the basics of how to operate and training a farming robot to recognize crop and detect weed.

Concretizing Your challenge

As you have seen in the Video in this lesson, farming bots are used for planting and weeding. To do this, they must first be able to follow a set track. Your goal in this challenge is to build a simple miniature bot, your mBot, and program it yourself so that it follows a set track. In this challenge this track will be a line parkour (see illustration) Your challenge will be finished, when you mBot follows from the start to the finish on the black line.



Line parkour for tracking task to accomplish the challenge.

Your four sub-challenges:

To accomplish this task, you will have to overcome **four sub-challenges:**

1) First you will learn how to program a robot. Don't worry, you will use a simple tool to program your mBot without writing any code just by dragging and dropping. You will start with simple programming exercises before you start programming a robot.
2) Secondly you will assemble your mBot and test his basic functions.
3) You will than connect your mBot with you programming tool and program it.
4) At last you will test your programmed Farming Bot on the test parkour.

After you have finished this Lesson:

Now that you have set up your robot development environment, you are ready to begin Lesson 6.3, which will familiarize you with the non-code programming tool mBlock

Lesson 6.2 Introduction to Programming with a non-Code platform

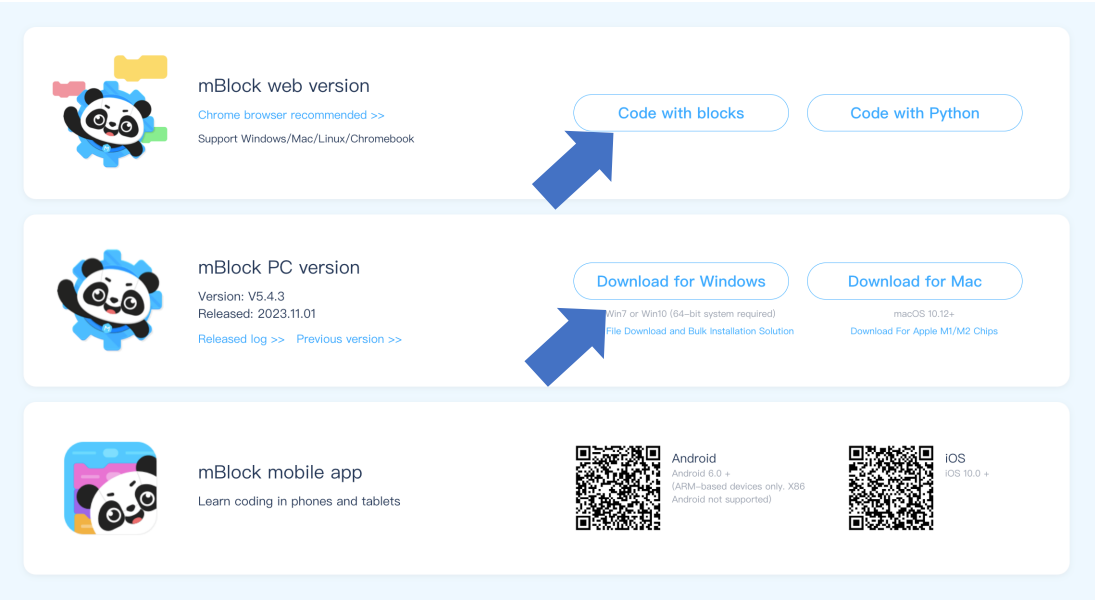
Learn how to program with non-code-platform mblock

You should take some time to learn programming. It is time well spent in several ways. First, you will need basic programming skills for your later task of teaching the mBot skills. But it is also time well spent for your personal digital literacy. Even if you don't become a software programmer, the basic programming skills you learn will help you better understand interdisciplinary projects and teams, and how to collaborate with different disciplines to find solutions.

In this lesson, you will download the MakeBlock programming tool called mBlock, set up your MakeBlock account, and run the application for the first time to familiarize yourself with mBlock non-code programming through seven exercises.

Download mBot from MakeBlock website

On the MakeBlock website you find in “Downloads” different options to access mBlock. There you can download the app for Windows or Mac PC or simply use the browser version: https://mblock.makeblock.com/en/download/

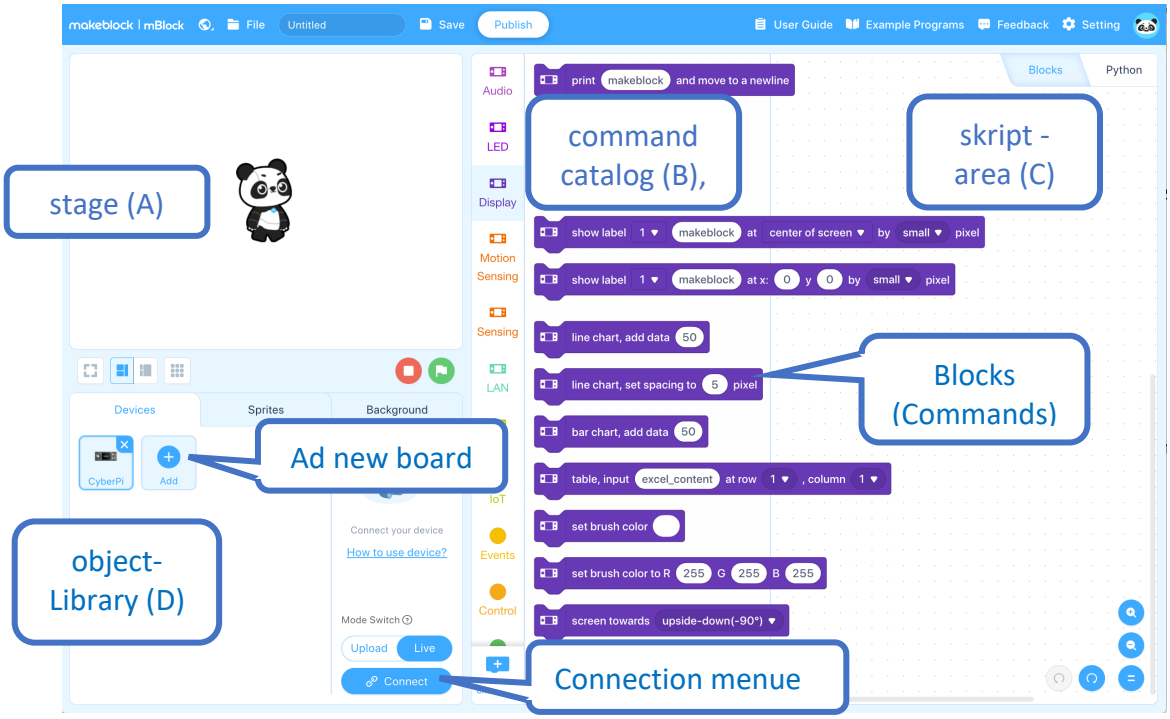


Download Area of mBlock (2024) https://mblock.makeblock.com/en/download/

Setup your MakeBlock Account

Let's have a closer look at the programming Interface of mBlock, which you will use in this project. If you have not yet set up your MakeBlock account, please register now so that you can follow the instructions below and complete the exercises.

For this course mBlock version 3.4 is used. Depending on the hardware, slightly older versions may be necessary. However, the differences are negligible for this workshop. mBlock version 3.4.12 is used. In some cases, a slightly older version is installed on the student laptops. However, the differences are negligible for this workshop.



familiarize yourself with the Programming Interface of mBlock

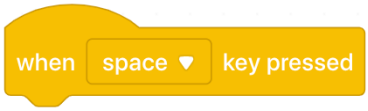
The software interface is divided into four main areas. The upper left area contains the **stage (A)** with the programmable objects. The result of the programming is visualized here. The program can be started with the green flag and ended with the red stop symbol. The middle column contains the **command catalog (B),** in which all available commands are listed categorically (in color). The self-explanatory "Costumes" and "Sounds" tabs are not required for this workshop. On the right-hand side is the **script area (C),** in which the program is formed by dragging and dropping the command blocks from the catalog. Additional programmable objects can be added and selected in the **object area (D)** at the bottom left. However, this is not required for the workshop, as the focus is on programming the mBot and only the required commands are demonstrated using the objects on the stage.

The menu bar options will be required at a later stage when programming the mBot and will be explained there.

Basics of commands

Commands are instructions to a computer (here a computer or data processing unit of the robot). In mblock all commands are represented as objects, **“Blocks”,** and sorted in a catalog according to areas of use. Their shape also provides an indication of their function and ability to be used with other commands. For example, under the category audio you find functions like play, stop, start recording, stop recording or say something and so on.

Individual commands are called "blocks" that can be combined into a program. They have small bulges or noses at the top or bottom. Here is an example of a typical command block used as a starting condition for a following command. Let's look at the most important types of command blocks.



Commands rounded at the top (such as the block “when button “A” pressed”) can therefore only appear at the beginning of a program and are therefore also called “start condition”; For example, no further command can follow the command “forever”.

Some commands have white fields for text (rectangular) or numbers (rounded), such as the command “say [Hello!] for (2) secs”. Please pay attention to the English typesetting here. No umlauts may be used; decimal numbers are written with a point instead of a comma.

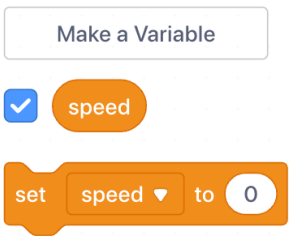
With bracketed commands (such as “if <> then” or “forever”), command blocks can be inserted into the brackets, which are then only executed under certain conditions.

Conditions have a tapered shape at the sides, such as the command “<> and <>” and can be “true” or “false”.

Commands whose output is a number are rounded on the sides. For example, the command “() + ()” fits into the round box or the in the command “say [] for () secs”.

Movements are changes in object such as their position, rotation, movement, and others. The example shows the movement of an object by 10 steps.

Sensing commands are used to detect or measure the state of an input device, sensor, or other object. This example shows a command that asks if the mouse pointer is touching an object



Variables are any kind of information types like time, speed, color or any other. A variable can be predefined, or user defined. The example shows a defined variable "speed". It can be used in an operator that sets the speed to a specific value.

Apply Commands

Next, you will perform a few simple exercises to familiarize yourself with the learning by doing programming environment. For these exercises you only need the **mblock app**. Work through the exercises one at a time. They build on each other and become more complex. First, try to complete the exercises on your own. If you get stuck, see the hints on the following pages.

Task 1 – Start condition & program structure
Let the character say "Hello!" for two seconds

Task 2 – Command chains
Have the figure walk 10 steps and then turn 10° to the right. (If you run the program several times, the figure should then walk in a circle.)

Task 3 – repetition loops
Modify the previous program in such a way that the figure runs in a circle all by itself.

Task 4 – Branches
Modify the previous program so that the figure interrupts its continuous run and says "Hello!" for two seconds if the figure is touched with the mouse pointer.

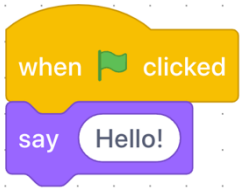
Task 5 – Variables (1)
Modify the previous program so that the speed at which the figure runs in a circle is expressed by a single variable. Try out different speeds.

Exercise 6 – Variables (2)
Modify the previous program so that the speed is increased by 5 when the up arrow key is pressed and decreased by 5 when the down arrow key is pressed.

Task 7 – Operators
Modify the previous program so that the speed can never go below 0 when the down arrow key is pressed.

Solution notes

Task 1 – Starting condition & program structure
Have the character say “Hello!” for two seconds.



Find a suitable one in the Events command category
Start condition, e.g. “when green flag clicked”, and drag it in the script area. (Of course, you can also choose a different start condition, such as having to first press a certain key for the program to start.) Then look for the command “say [...] for” in the “Looks” category (...) secs” and add it directly below the start condition. Make sure that the commands are actually connected to each other. By clicking on the green flag (in the corner of the stage) the character should now carry out the instruction. Try some other looks to explore their effects.

Task 2 – Chains of Command
First let the figure walk 10 steps and then turn it 10° to the right (clockwise).

This task shows (in a small form) that commands are strung together and accordingly processed one after the other.
The two required commands are in the “Motion” category “move (...) steps” and “turn right (...) degrees”. When executing the program (by clicking the green flag), the ten steps are barely visible. If the program is executed several times in a row, the figure should run in a small circle until it is back at its starting position. (Without the ten steps, the figure would rotate around its own center of gravity and not run in a circle.)

Task 3 – Repeat loops

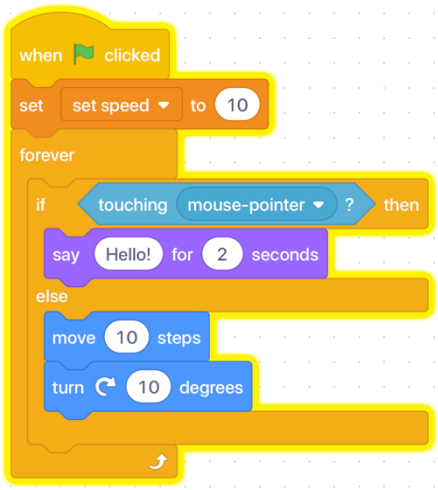
Change the previous program so that the figure runs continuously in circles on its own.

Recurring executions of certain program components are often desired and make operation easier. So that the green flag does not have to be pressed several times in the program, but the figure runs in a circle on its own, this program component must be executed repeatedly.

In the command category “Control” you can find several different commands (“repeat (until)”, “forever”) with which individual program components can be “bracketed” and then executed multiple or infinite times. These commands are absolutely necessary, especially with regard to the later use of the robots.

Task 4 - Branching/case distinctions

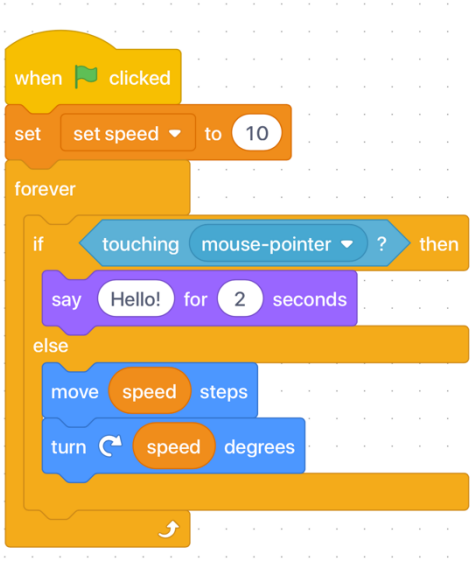
Modify the previous program so that the figure interrupts its continuous run and says "Hello!" for two seconds if the figure is touched with the mouse pointer.



If certain program components are only to be executed under executed under special conditions, Branches or case distinctions are required. The "Control" category contains commands with "if" for this purpose. What the commands have in common is that they require a condition (recognizable by the "pointed" placeholder field) under which the commands in brackets with these commands are executed. The condition "touching mouse-pointer?" from the "Sensing" category is required for this task. Later, when the mBots are used, the conditions are taken over by measurements from sensors.

Task 5 – Variables (1)

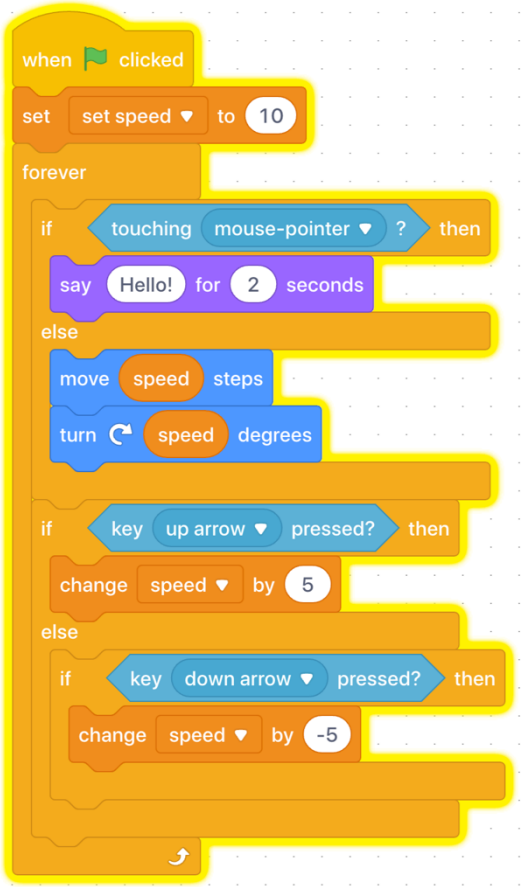
Modify the previous program so that the speed at which the figure moves in a circle is expressed by a single variable.



Try different speeds. The more complex a program becomes, the more frequently variable numerical values appear. If you want to change or adapt these, you would have to re-enter each individual field. With variables you can design a program so that you can do this only needs to be done in a few places. Especially when using the mBot, this can be helpful for the different speeds of the motors depending on whether the robot should go straight to the right or to the left. In the "Data&Blocks" category you can create and name variables. In the example, the speed variable was called "speed". Since a variable initially has no content, it must be assigned a value once at the start of the program ("set [...] to (...)"). Wherever a number is to be replaced by this value in the program, the variable can now be inserted instead of manually entering numbers.

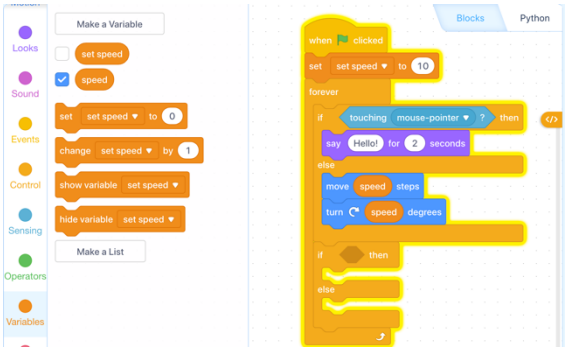
Task 6 - Variables (2)

Modify the previous program so that the speed is increased by 5 when the up arrow key is pressed and decreased by 5 when the down arrow key is pressed.



The program is supplemented by further components that should only be executed under special conditions. These are executed when the up and down arrow keys are pressed. Further case distinctions with corresponding conditions are therefore required. In this example, the queries as to whether the respective button was pressed were inserted below the program in a nest. They could just as well have been inserted above the actual program, or two individual, consecutive queries could have been made. The advantage of nesting is that it saves time, as the second query only takes place if the first was incorrect. Especially for more complex queries with several options, nesting can lead to a significantly faster program.

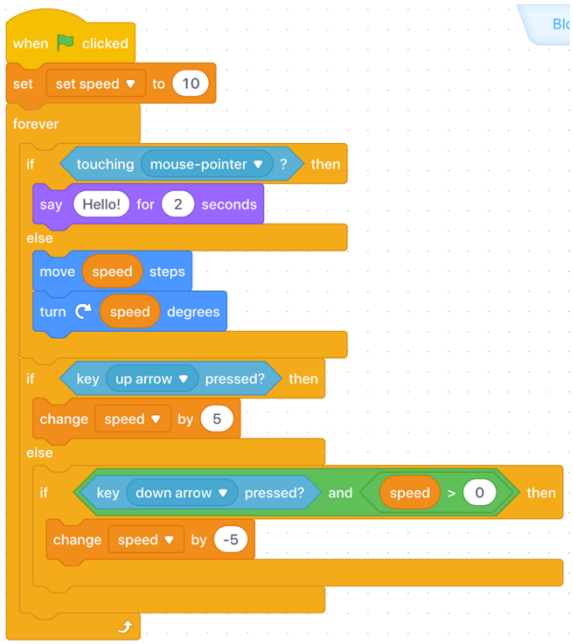
The screenshot at the right shows you how to generate a variable for “speed”: Go to the section “Variables”, klick on “Make a Variable”, give it a name and include it into your program.



Task 7 – Operators

Modify the previous program so that the speed can never go below 0 when the down arrow key is pressed.

In this task, the variable should only be decreased if two conditions are met at the same time. The down arrow key must still be pressed, but the variable must also be large enough so that the result of the change is not negative. In this case, the condition that the variable is positive, is sufficient (i.e. “greater than zero”). Since both conditions should apply at the same time, they can be linked using the “and” operator. Alternatively (not shown), two nested if case distinctions can also be used. The green operator commands can be found in the “Operators” category of the same name.



Congratulations!

By completing the seventh task, you have almost covered all the basic and recurring commands that are required for programming your mBot.

Commands that can only be carried out on the screen, e.g., from the categories “Motion” (blue), “Looks” (purple) or “Sensing” (light blue), cannot be used with the robots and are then used by special robot commands (petrol colored category “Robots”) replaced. These commands will be discussed in the next chapter when the hardware will be introduced at the same time.

After you have finished this Lesson:

Now that you have familiarized yourself with the mBlock non-code programming tool, it is time to introduce you to the hardware of your mBot robot in the next Lesson 6.4.

6.4. Introduction to the hardware of the Robot: the mBot

In the first chapter, you learned how to control a **virtual character** with commands. This character could also respond to additional inputs, like pressing arrow keys. Now, we want to apply what we did with the virtual character to a **real robot**, the mBot, which should eventually navigate a course on its own.

Programming basically means that you make an input, like pressing a button, and the robot reacts to it. For example, if you program the remote control, the mBot can move in different directions at the push of a button. But if we still have to control it manually, the mBot isn't a real robot. **A real robot does everything independently, without our intervention**. This is important for digitalization because machines can then work without our help, making everything more efficient.

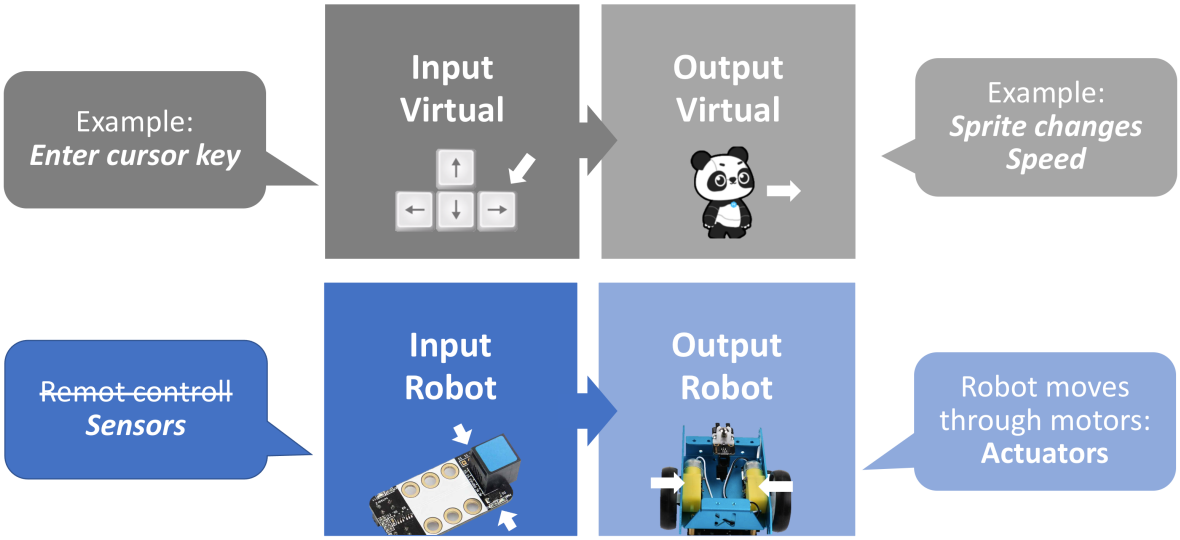


Figure: input and output of a virtual system (program) and a physical system (robot) in principle. Source: Dickel 2024 using images from Makeblock

For controlling the robot, we use **sensors**, specifically the **Line-Follower sensor**. This allows the robot to detect the line on the course and make decisions. Its movements are executed by **motors**, also known as **actuators**. This chapter focuses on sensors and actuators.

Compared to virtual control, where commands are clear (e.g., is the arrow key pressed: Yes or No?), the input and output data for the mBot are not always so direct. Sometimes we need to translate human language into machine language. A following diagram will help to understand this better.

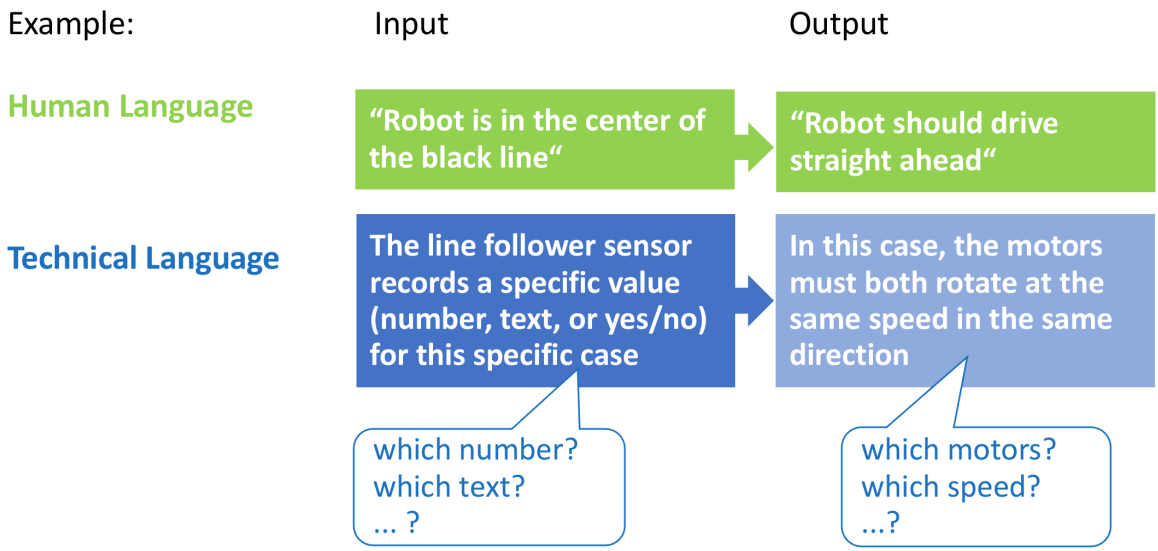


Figure: example for input output in human language and technical language. Source: Dickel 2024

In this example, lets discuss how to translate our instructions so that the robot understands them. The robot can't directly understand what we mean by "stay on the line" or "go straight." We need to translate our language into the robot's language, so the robot knows what to do. To do this, we first need to figure out what the Line-Follower sensor can detect and how the motors can respond to it. The line follower sensor is attached to a button on the front of the mBot.

(Task 8 Sensor data –>left out)

Figure: Line following sensor (left) and its location on the mBot (right)

Understanding Sensors

Before we start, we should think about what the sensor can detect. The Line-Follower Sensor has two light sensors that can distinguish between "light" (white) and "dark" (black). There are four possible situations: black-black, black-white, white-black, and white-white. For example, black-black indicates that the sensor is right on the black line. Then, the number "0" is returned. In the example above, the table can now be changed as follows.

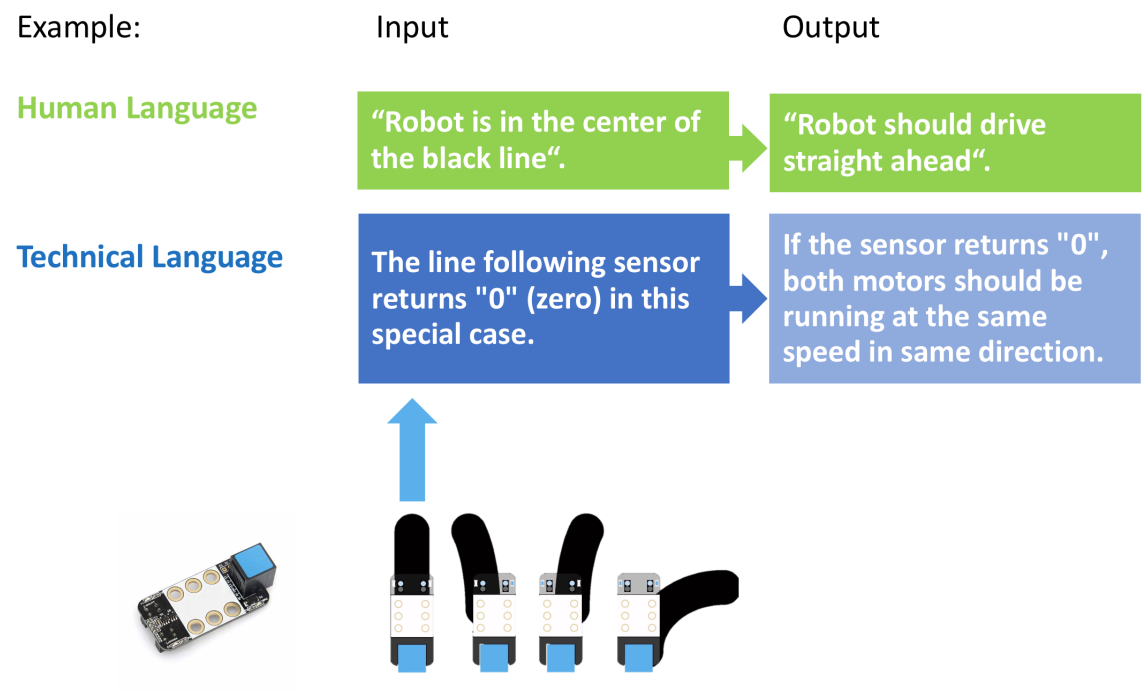


Figure: example for input/output in human language and technical language of **the line following Sensor**. Source: Dickel, using images from Makeblock 2024

Have a Look at the other three scenarios: They show that the robot should turn left when the sensor value is "1" and right when it's "2". The last case “3” means the robot has left the line, and it's up to the programmer to decide what to do next.

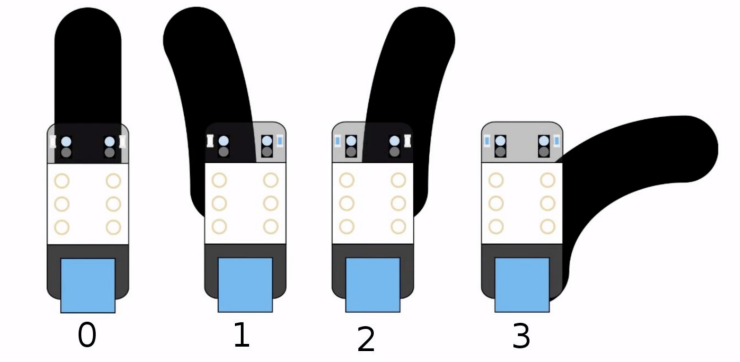


Figure: Four states of the line following the sensor, source Makeblock

Understanding the Motors

We still need to understand how the mBot's motors work. There are two types of commands: The first command has four pre-programmed movements (forward, backward, turn right, turn left) with various speeds (from "255" to "-255"). The second command controls the two motors (M1 and M2) individually. Which motor is on the left or right depends on the wiring. You can also program your own movements.

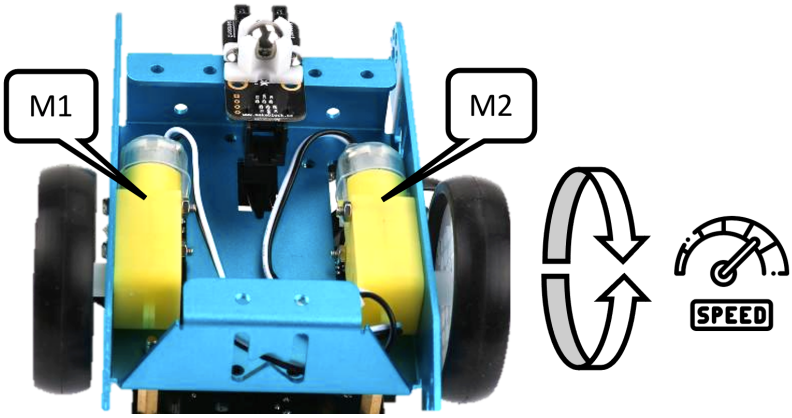


Figure: The motors M1 and M2 of the mBot can turn in two directions with different speed. Source: Dickel

The motors need a certain speed to move. Below ±70, they often don't have enough power. The robot won't move if the speed is too low. This also depends on the battery level, so the mBot should always be well-charged. It's important that commands are followed until a new one is given. This can be useful for efficient programming, which will be discussed further in chapter three
.

After you have finished this Lesson:

Now that you are familiar with the hardware of your mBot robot, you are ready to begin implementing your prototype by assembling the robot and connecting it to the programming tool in the next lesson 6.5.

6.5 Assembling and connecting the robot to the programming tool

Now it's time to start building the robot by assembling the mBot kit. You have already purchased your mBot and downloaded the assembly instructions as described at the beginning of this topic. If not, go back and follow these instructions step by step. If you have any questions, do not forget that you can always contact the makeblock support team directly via e-mail.

The following figure shows the main components you will need to assemble the robot. It is not very complex, as you can see. So be sure: you can do it.

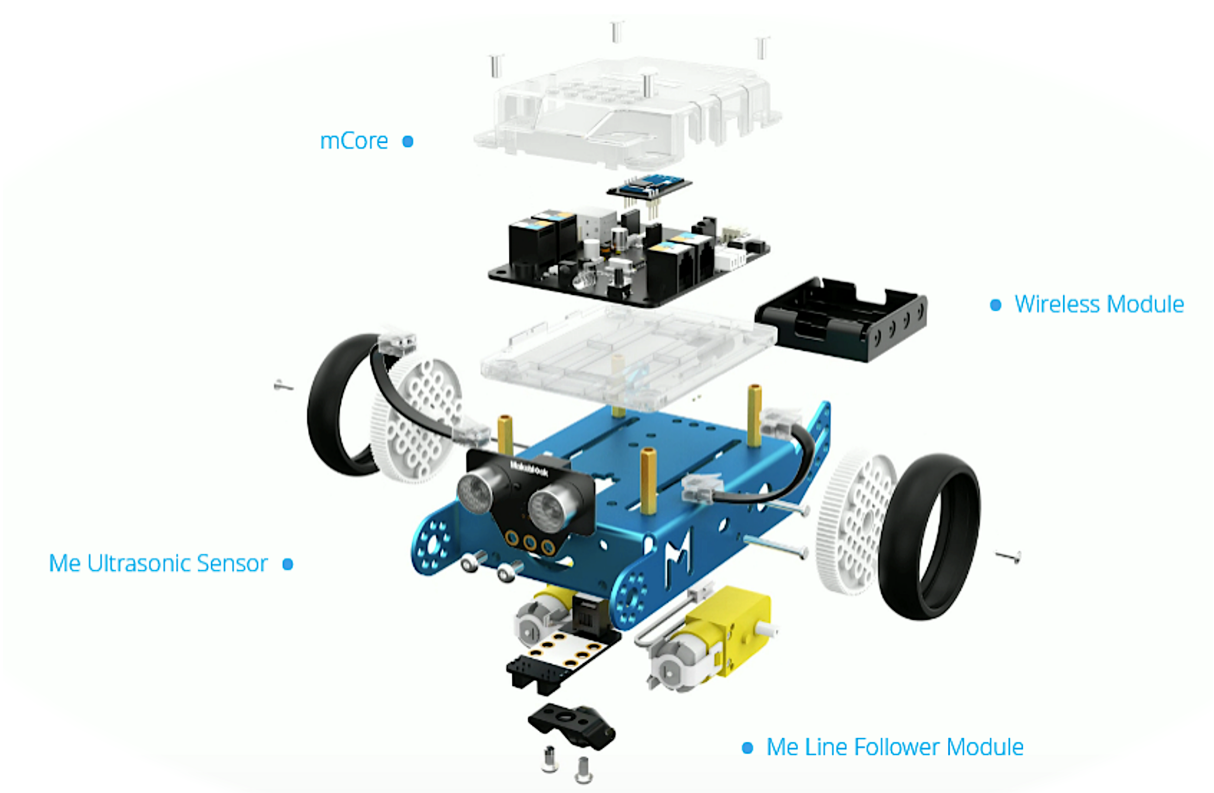


Figure: Components of the mBot 1. Source: Makeblock

The mCore – central processing unit

The central device of your mbot is called the mCore. It's the processing unit of your bot and will detect and process all sensor-inputs and send corresponding output signals to the actuators. This picture shows the mCore and its essential components and functions. You may later need some of them. But for now, you do not need to change or adjust anything on the mCore. Just build it into your mBot as described in the manuals.

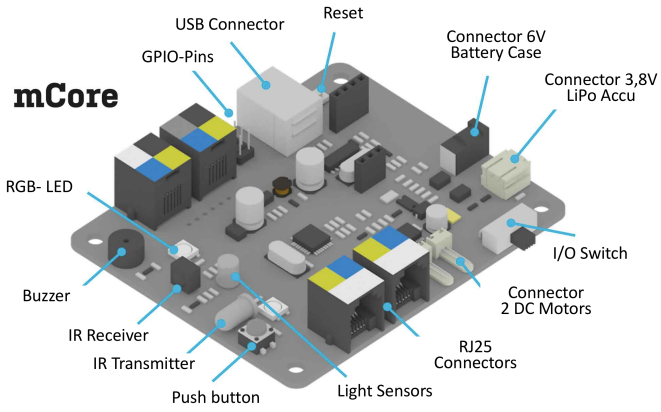


Figure: the mCore, the processing unit of the mBot, with its main components and connectors. Makeblock

Support to assemble your mBot.
Here are the complete resources you can use to assemble your mBot. You can find more comments on the links at the beginning of this topic.The package you received with your mBot also contains instructions.

mBot Quickstart instructions:
http://cdnlab.makeblock.com/mBot Quick Start Guide .pdf

Beginner's Guide to mBot:
https://support.makeblock.com/hc/en-us/articles/12822859943959-A-Beginner-s-Guide-to-mBot

Complete mBot assembly guide with part list:
http://download.makeblock.com/mBot-V1.1_Blue_STD_Instruction_EN_D1.2.1_7.40.4100_Print.pdf

Support website from Makeblock, you should bookmark for any case:
https://support.makeblock.com/hc/en-us/sections/360001829013-mBlock-5

Here is the link if you want to use the mBlock App on your tablet:
https://support.makeblock.com/hc/en-us/articles/12993202671383-Program-mBot-with-the-mBlock-App

Now we wish you every success in assembling your mBot!

Connecting your mBot with the programming tool mBlock

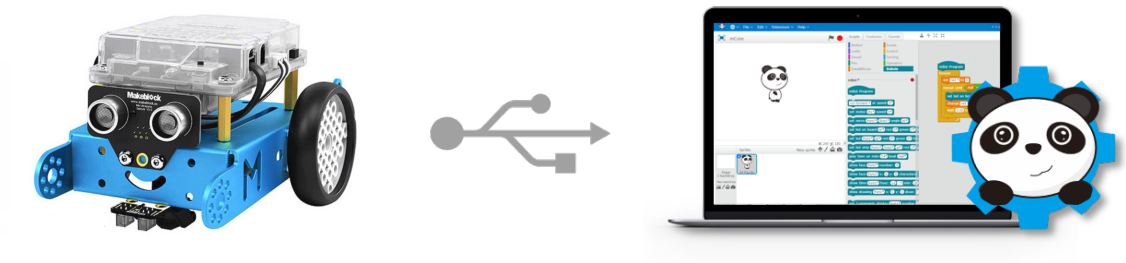
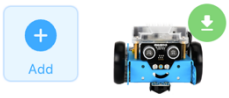


Figure: mBot and mBlock Programming tool, source makeblock

Is your mBot ready to go? Have you assembled your mBot according to the instructions? Did you try a first run as described in the manual? Maybe you had to fix some bugs? Did it finally work as expected? **Then congratulations!** You are now ready to connect your **mBot** to the **mBlock** programming tool. To connect your mBot to the mBlock programming tool, just follow these steps, which are also described in the manuals:

1) Connect laptop/PC and mBot with USB cable

2) Switch on the mBot on the mCore board



3) Select command set for mBot: Select the mBot (mCore) option in the Boards menu

4) Select the appropriate connection under Serial Port in the Connect menu

5) Clean up the memory: Select the Upgrade Firmware option in the Connect menu

After you have finished this Lesson:

Now that you have assembled the robot and connected it to the programming tool in the next lesson, Lesson 6.6, you will complete your prototype by programming two skills to meet the needs of your robot as a FarmingBot.

6.6 Implementing the Prototype for the FarmingBot

Under construction

6.7 Testing and optimization of the prototype, bug analytics and fixing

Under construction

After you have finished this Lesson:

Now you have completed your prototype by programming the two skills to meet the needs of your robot as a FarmingBot. And you finshed Testing and optimization of your prototype, did bug analytics and fixed the problems that accured.

Congratuations!

You did a great job!