Mobile Robotics with micro:bit

Welcome

This website is intended to provide the ressources for the students following the micro:bit mobile robotics summer school. But those ressources have been produced to be reusable, and teachers are of course welcomed to go through this site and take whatever they like for their own courses.

Overall, the program of the summer school will be as follows:

	Monday	Tuesday	Wednesday	Thursday
	Interacting with the world	**Control and navigation**	**Designing and building**	**Opening**
9:00 - 9:45		Building the Buggy	Building the cardbot	Project
10:00 - 10:45	Welcome, teams, program and material	Simple displacements	Designing a robot	
11:00 - 12:00	Introduction to micro:bit	Control systems	Building and programming an obstacle avoiding robot	Feedback on project
Lunch break				
13:30 - 14:15	Sensing - potentiometer	Mini-projects on radio communication and engineering **	Tutorial on CAD	EPFL presentation & questionnaires
14:30 - 15:15	Acting - Servo motor		Project	Ways to go further
15:30 - 16:30*	Sensing - Ultrasounds			Conclusion with discussion

* May vary: on Monday the program is quite packed and we might finish later, but not later than 17:30. The opposite applies on Thurday

**Sessions filled in green will not be guided on Zoom, however you'll have the necessary ressources and someone to contact in case of problems.

The structure of this site follows the chronology of the activities you will go through (before and) during the week of the summer school:

- Introduction to python
  As most of the activities will include some python programming, I'll need everyone to come with a minimum of experience of this language so that we can go through all the activities without leaving anyone behind. This is the purpose of this page which will make you practice the basics of python and discover the different tools you'll need during the week if you don't know them already.
- Micro:bit board
  On Monday you will discover the micro:bit controller which will be the brain of our mobile robots. You'll use your (freshly gained) skills in python to already start interacting with the board. In the

afternoon, you'll start to discover how to **connect a controller to sensors and actuators**, that is how to give him more the ability to sense and act on its environment.

- **Robotics and moving**
  While previously we were limited to a bunch of components lying around next to each other, on Tuesday you'll add a bit of structure to all this. You'll put together a simple mobile robot kit and learn to control it. In the afternoon, you'll work on the same theme, but learning to gain autonomy to **decompose and solve robotics problems** without guidance.
- **Designing and building**
  The robot we find in kits is rarelly able to satisfy all our desires. To free us from those limitations, we'll see how to design our own robot frame and build it using simple equipment first, in the morning, and then using professional equipment and **CAD** in the afternoon.
- Closing and opening
  On the last day we'll come back on what we learned and produced during the week and you will be presented different options to continue your **exploration of mobile robotics** smoothly in autonomy.

Have a good time and may you learn a lot !

Introduction to Python

During the upcoming robotics summer school, I'll need you to be confortable with the basics of python programming. I wrote this page using embedded codes to have everyone starting with the bare minimum going from a simple "Hello world" script to using variables, loops, conditions and functions. As the program of the summer school will be intense, I'll need you to be confortable with those.

For each section of this page I'll use embedded code that allow you to run python scripts without leaving this page, and interact with the code as you wish. So let's go: read the code, try to predict what's going to happen in the terminal (in black), and press the play button.

Again, don't hesitate to change the code and press play again to try things out, anything ! For that you can also directly type instructions in the terminal (the black window).

Variables and standard operations

Computers are so useful to us because they don't always do the same things over and over. It's the same with mobile robots, they can adapt to their environment. This would be impossible to do without taking into account some variations in our programs. To deal with those variations, we use what we call variables. A variable is like a box where we put some information that can vary during the execution of a program.

Same here, change the code, make it crash ! (Even more than it already does !)

Loops

Another way computers and robots change our lives is in their ability to cope with repetitive tasks without being bored. But the programmer who programmed them gets easily bored, so he needs to have a tool to take care of repetitions in his code. That's what loops are for.

Exercice 1

Add some lines in the script above to display the 20 first numbers of the Fibonacci sequence.

▼ Solution exercice 1

```
34  x_nm1 = 0
35  x_nm2 = 1
36  for i in range(20):
37    print(x_nm1)
38    x_n = x_nm1 + x_nm2
39    x_nm2 = x_nm1
40    x_nm1 = x_n
```

```
21
34
55
89
144
233
377
```

Functions

Do you know what assembly language is ? Take a few seconds to have a look at it on wikipedia and try to decipher what the program in the top right image does... That's not the language you want to program every day ! Take some time to contemplate the chance you have to not go through that and instead use python.

Well, you have this chance partly thanks to functions. As in mathematics, functions can take arguments and return a value. You should have all by now used the cosine function. Cosine takes onoe parameter, an angle, and returns a value in between -1 and 1. In computer science, functions come in many forms. They can take argument(s) or not. They can return values but they don't have to (in this case we could call it a procedure).

Such a tool allow to make our code readable, less repetitive, and combine and build functions with and on one another. That's how, thanks to programmers, your computer can understand some high level languages like python, while at its core it can only understand assembly. Let's see how to use this incredible tool in python.

Exercice 2

Define a function with one parameter *n* that computes and returns the

golden ratio using the fibonacci *n*th and *n+1*th numbers.

▼ Solution exercice 2

```
def approximateGoldenRatio(n):
  return fibonacci(n+1)/fibonacci(n)
```

See how simple the exercise is if we use the predefined fibonacci function. The exercise would have been much harder if we had to start from scratch.

Conditions

In our programs, depending on some variables, sometimes we want to do things, sometimes we don't. To perform such a feat the last ingredient we are missing in our arsenal of tools are conditions. Conditions are quite simple and we use them everyday: Does it rain ? Yes/No => Take an umbrella/Don't. Note that in computer science, conditions are made of: a test resulting in a boolean (True or False), what to do if the test is True, and sometimes what to do if it is not.

Exercice 3

Modify the code following the instructions inside it.

▼ Solution exercice 3

```
13  while True:
14    s = input('Which Fibonacci number do you want? ')
15    if s.isdigit():
16      # as input returns a string and we want a number
17      # we convert if to integer
18      n = int(s)
19      # print the result
20      print("Fibonacci(",n,") = ",fibonacci(n),"\n")
21    else:
22      print("I said which Fibonacci NUMBER...")
```

```
Which fibonacci number do you want? 10
Fibonacci( 10 ) =  55

Which fibonacci number do you want? 15
Fibonacci( 15 ) =  610

Which fibonacci number do you want? test
I said which fibonacci NUMBER...
Which fibonacci number do you want?
```

Conclusion

That's all for now. Thanks for going through this introduction and I see you soon to use those new skills to discover the world of mobile robotics.

Introduction to micro:bit

This morning, you will discover what will be the brain of the mobile robot you will build: the micro:bit. You'll see that this controller comes already out of the box with sensors and actuators, which are the building blocks of a robot, and you'll learn how to interact with them.

What you will need

Every pair of learners will need:

- A computer equipped with a connection to internet and a browser (ideally Chrome)
- One micro:bit
- A usb cable to connect the micro:bit to the computer.

Programming Micro:bit

Gather all the material listed above and connect the micro:bit to the computer. You should see an external drive appear on your computer called *MICROBIT*.

To program the micro:bit you will need an editor. For this open the Python online editor for micro:bit on your web browser. On this page you'll be facing with a big black area with some python code. This area is the editor where you'll type your programs. On top of that you should see different buttons/icons that will allow you to interact with your computer, and the micro:bit directly if you are using Chrome.

When you open the editor, it comes with a "Hello, World!" example. To start, you have to learn to transfer a program onto the micro:bit. For that:

- First option: you click the Download button. This will download the program as a *.hex* file named *microbit_program.hex* on your computer. To put it in the micro:bit, drag and drop the *.hex* to the MICROBIT external drive.

- Second and easiest option if you have Chrome: click *Connect*, select the micro:bit in the menu and click again on connect, and now the *Download* button should have transformed into a *Flash* button. This button allows you to directly put (flash) the program from the web-browser to the micro:bit.

At the end of both options, an orange LED should blink on the micro:bit, informing you about the transmission. When the transfer is over the program should start automatically and you should see the execution of the program on the red LEDs of the micro:bit. Congratulation, you gave your first program to the micro:bit !

Micro:bit specific functions

The **structure** of a program on a microcontroller is and will almost always be as follows: first, an **initialisation**, then a **loop**. Take the time to identify those two parts in the previous program.

Here is another program to show you some specific micro:bit functions that are available to use its buttons (*button_a.was_pressed()*) and its LED matrix (*display.set_pixel()* and *display.clear()*):

```
# Initialisation
# Get standard librairies
from microbit import *

# Loop
while True:
    if button_a.was_pressed():
        display.set_pixel(0,2,9)
        sleep(1000)
        display.clear()
    if button_b.was_pressed():
        display.set_pixel(4,2,9)
        sleep(1000)
        display.clear()
    sleep(100)
```

Exercise 1

- 1.1 Flash the micro:bit with this program. What do the the arguments of the functions *set_pixel()* and *sleep()* correspond to ? To answer either try to change the program or read the doc (sleep doc, display doc) or even better, do both.
- 1.2 Execute the instruction *display.set_pixel(-1,2,9)* in your program and see what happens.

▶ Solution exercise 1.2

Your first program

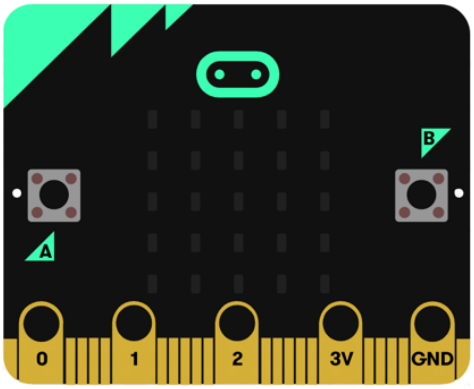
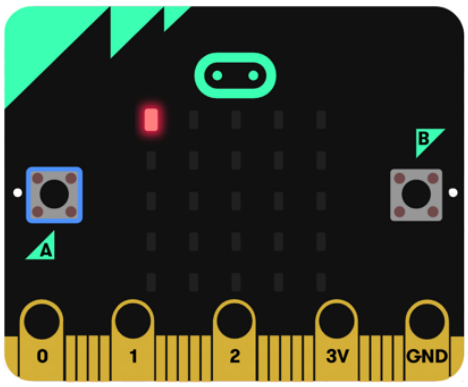
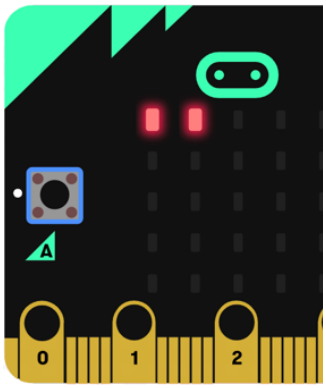
displayPositiveNumber(0)	*displayPositiveNumber(1)*	*displayPositiveNu*
		

Figure: A program to display rapidely the value of a variable from 0 to 5.

In the previous exercise you learned to interact with some basic functions of micro:bit. It is now time to bring your knowledge of python into play to interact with those. By reinvesting your knowledge of variables, conditions, loops and functions you will define a function that will help us work with micro:bit all through the week.

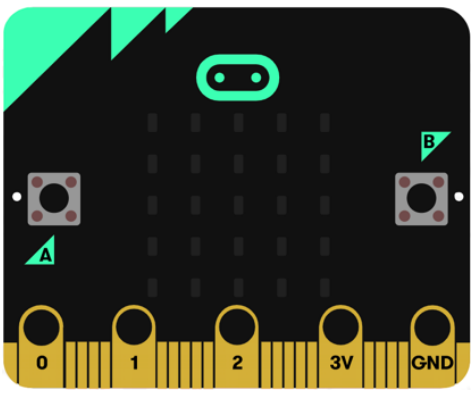
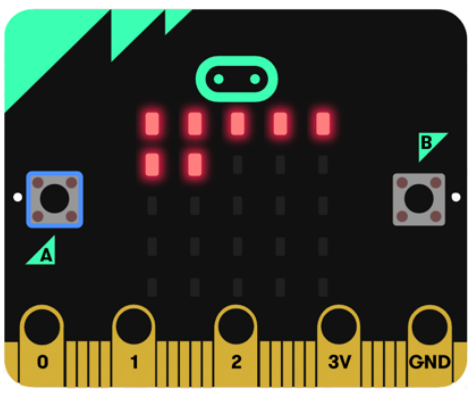
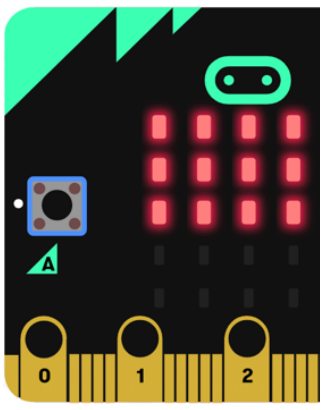
The functions *display.show()* and *display.scroll()* are powerful and can show many types of variables (integers, floats, strings) on the screen of the micro:bit. Such functions are useful to debug in showing us some valuable information. However, as we saw with the "Hello, World!" example, this function is also not very reactive, while we need to be reactive in robotics. We can not wait 2 seconds for each piece of

information. So you will define a function which will solve this problem.

Exercise 2

- 2.1. Define an integer variable *x*, initialise it to 0 and display its value every 100 ms using *display.scroll()*.
- 2.2. Program the micro:bit so that you can use the buttons A and B to increase and decrease the value of *x*. What happens if *x > 9*, can you show *x* value every 100 ms ?
- 2.3. Forget about the value of *x* for this sub-exercise. Replace the line that was displaying the value of *x* and instead, use a *for i in range(4):* ... loop and the function *display.set_pixel()* to light up the first line of LEDs.
- 2.4. Modify your program to display the value of *x* as in the pictures above. Hint: you can either change the for loop, or add a condition.
- 2.5. Define a function *displayPositiveNumber(n)*, that shows the value of an integer *n* for *n in [0-5]* on the LED matrix.

Going further

displayPositiveNumber(0)	***displayPositiveNumber(7)***	***displayPositiveNu***
		

If we plug some sensors to the micro:bit and we want to display some quantitative information coming from them, *displayPositiveNumber(n)* is exactly the sort of function we need. However, for now, our resolution is very little (only 6 possible values can be displayed). To get a richer feedback, let's use the whole LED matrix. For this you'll modify the previous function slightly:

```
def displayPositiveNumber(d):
    display.clear()
    for i in range(d):
        display.set_pixel(i%5,int(i/5),9)
```

Exercise 3

- 3.1. On a piece of paper, represent a 5x5 grid and set a number for each row and column, starting on the top left with the coordinate (0,0). For i = {4,5,6,24} compute *i%5* (meaning *i mod 5*... meaning the rest of the integer division of i by 5) and *int(i/5)* (meaning the integer part of i/5). Write down the value of i in the grid at the position (i%5,int(i/5)).
- 3.2. Now that you should have a better understanding of this function, test it by updating the code in the previous exercise. In the end you should have a program to set the value of *x* using the buttons and display its value for *x in [0-25]*.
- 3.3. (Optional) The function *displayPositiveNumber()* is reliable only for values in [0-25]. Make it more reliable by adding a condition so that if the provided argument is not in that range, you will display something special (for instance one LED on in the middle of the LED matrix).
- 3.4. (Optional) There are different ways to write a function such as *displayPositiveNumber(d)*. Find other ways to define the function. You could for instance use nested loops (a loop in a loop) or use two counters (integer variables) and a condition (that would allows to not use modulo or the integer division).

You are now equipped with a function which is going to be very useful this afternoon. To reuse it this afternoon, save it using the "Load/Save" button, followed by clicking the "Download Python Script" button. This will allow you to save it as a python file an reload it this afternoon. If you did not understand the functioning of this function, take some time to do so. Talk to the people around you if you need help.

Conclusion

This morning you saw one key component of what makes a robot: its **controller**. The controller is the brain of the robot which does the interface between its **sensors** and its **actuators**. But you saw more than that. Indeed by programming the micro:bit you already had a quick go at doing the interface between some integrated sensors (buttons) and actuators (LED matrix) of the board.

While this sensors/controller/actuators design is the standard design for mobile robots, it also applies to many other fields. Look around you and you'll see this arrangement is standard in technology as well as in life. A smartphone comes with sensors: a camera, touch(screen), microphones, radio receptor, accelerometer, compas; and with actuators: a screen, speakers and radio emitter. Human beings come with sensors too, although they are harder to isolate than on machine, but making it simple we could say that most of them come equipped with eyes, ears, nose, skin, tongue and proprioceptors (too often forgotten).

We also saw that for the controller to make the link between the sensors and the actuators, you need to give it a program. While you used the micro:bit this morning, what you learn applies to most other controllers:

- A programming language is most of the time composed of standard building blocks such as instructions, variables, conditions, loops and functions
- The structure of the program of a robot comes almost always with an initialisation and a loop (and sometimes with interrupts but I invite you to discover this when you are already familiar with the basics).

Input and outputs

This morning we saw how the micro:bit comes equipped with integrated sensors and actuators. For mobile robotics, those are not enough, so this afternoon you'll see how it is possible to add sensors and actuators to the micro:bit to go towards our goal of building an autonomous mobile robot.

What you will need

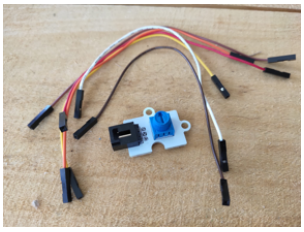
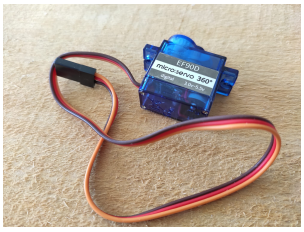
Buggy mainboard	Potentiometer and wires	Servo motor	Ultrasound sensor
			

Figure: Necessary equipment.

Every pair of learners will need:

- The equipment required in the previous session
- One bit:buggy mainboard, 5 screws and a screwdriver (all in the bit:buggy box)
- One potentiometer
- 6 Wires female-female
- One servo motor (in the bit:buggy box)
- One ultrasound sensor (the HC-SR04)
- 3 x AAA batteries

GPIO

Look at the micro:bit. On its bottom there is a strange set of golden lines (it's not gold, don't start to rip it appart and try selling it!). Those are conductive connectors that will allow you to connect more sensors and actuators to it and interact with them. They are what we call inputs and

outputs or GPIO (general purpose input/ouputs) which are standards in the world of microcontrollers. Each connector is called a *pin*, and most of them can be used as input or output. The micro:bit has about 20 of them in total (more information here).

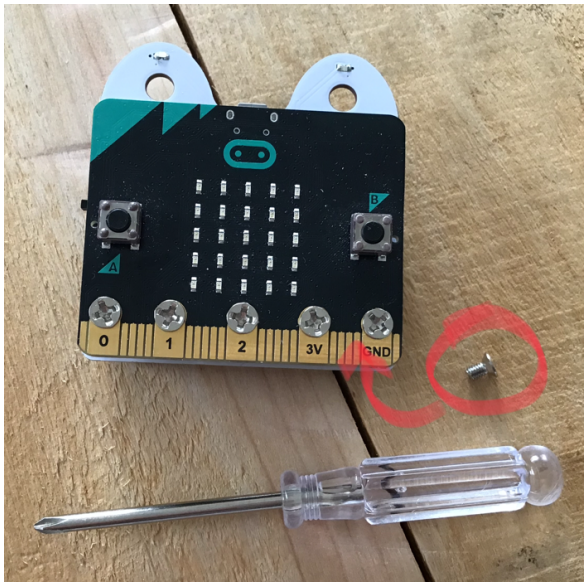
Adding a simple sensor

The first sensor you will add is called a potentiometer. For this you will use one pin as an input. You'll learn how to capture the position of the potentiometer in our program.

Before doing any programming, you will have to take care of the physical connection between the potentiometer and the micro:bit. **Begin by disconnecting the micro:bit from the computer. Before doing any changes in the wiring of a controller, always power it off !**

Then, screw the micro:bit on the bit:buggy mainboard using 5 screws (see figures below). This will give you access to pin0, pin1, pin2, 3.3V and Ground through femelle pin headers on the back of the buggy board. In this activity, you will use the pins pin0 (0), 3.3V (V) and ground (G). Connect the wires as shown in the image below (the colors are not important although we usually try to put a black cable for ground and a red one for power sources).

Screwing onto the buggy mainboard	Connecting the potentiometer



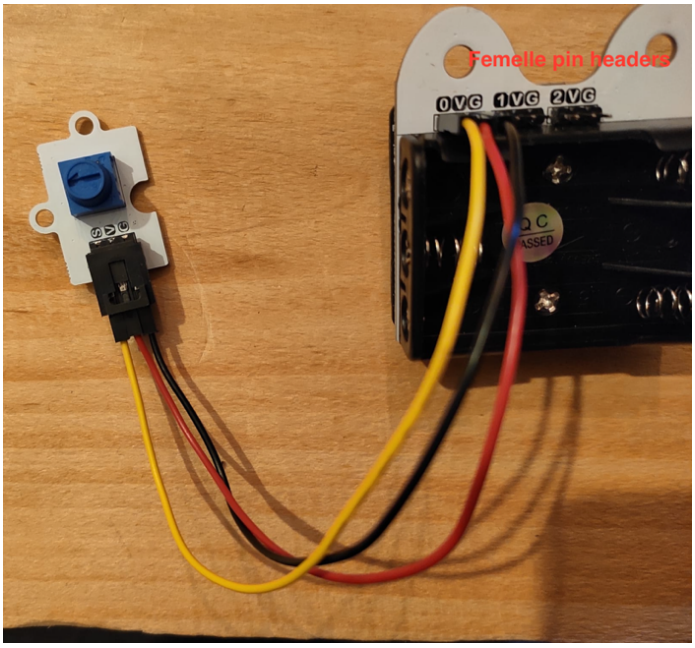


Figure: Connecting the potentiometer to the micro:bit. DO NOT PUT THE BATTERIES IN YET !! You might destroy the micro:bit if you do so...

In this montage, the potentiometer will ouput through S (for signal) a portion of the input voltage (3.3V) depending on the position of the knob.

To access the information of the pins in python, the microbit library defines for us the objects *pin0*, *pin1*, etc... Those objects come with two useful functions: *read_analog()* or *read_digital()* (see doc here). If we are interested in getting a binary information (is the voltage on the pin high or low?), we use *read_digital()*. That's what we'd use for a button (pressed or not). If we want more we have to use the function *read_analog()*. This is our case as we want to know the position of the knob, therefore, we want to know which portion of 3.3V is on pin0.

exercise 1

- 1.1. *pin0.read_analog()* returns an integer. Program micro:bit to display the value returned by this function using the function display.scroll() every 100 ms. Turn the knob to find what is the range of values that pin0.read_analog() can return. Can you display the value every 100 ms ?
- 1.2. Use the function *displayPositiveNumber(n)* defined this morning to display the position of the knob every 100 ms. Remember that this

function works well only for arguments in [0-25], so you'll need to do some scaling in between.
- 1.3. (Optional) If you are not comfortable with linear mapping (rescaling in our case), you could later take the time to define a function to do that for you so that *map(n, n_min,n_max,res_min,res_max) = res_min + n.(res_max-res_min)/(n_max-n_min)*.

▶ Solution exercise 1.1

While there exist many kinds of sensors, many of them work in a similar fashion as the potentiometer. A sensor will often require a power source (3.3Volts and ground), and it will output a signal.

Controlling a motor

Now that the micro:bit can sense, you will learn how to make it act on its environment with a motor. For this get one of the motors out of the bit:buggy kit. Notice that the motor comes with its cable. **Power off the micro:bit by disconnecting it from the computer**, then plug the motor as shown in the image below (connection on pin1, taking care of putting the brown/black wire facing the letter G). To make the rotation of the motor more noticeable, you can take one of the white plastic end that comes in the bag where you found the motor. This piece is called a servo horn and allow for easy mechanical constructions with the motor.

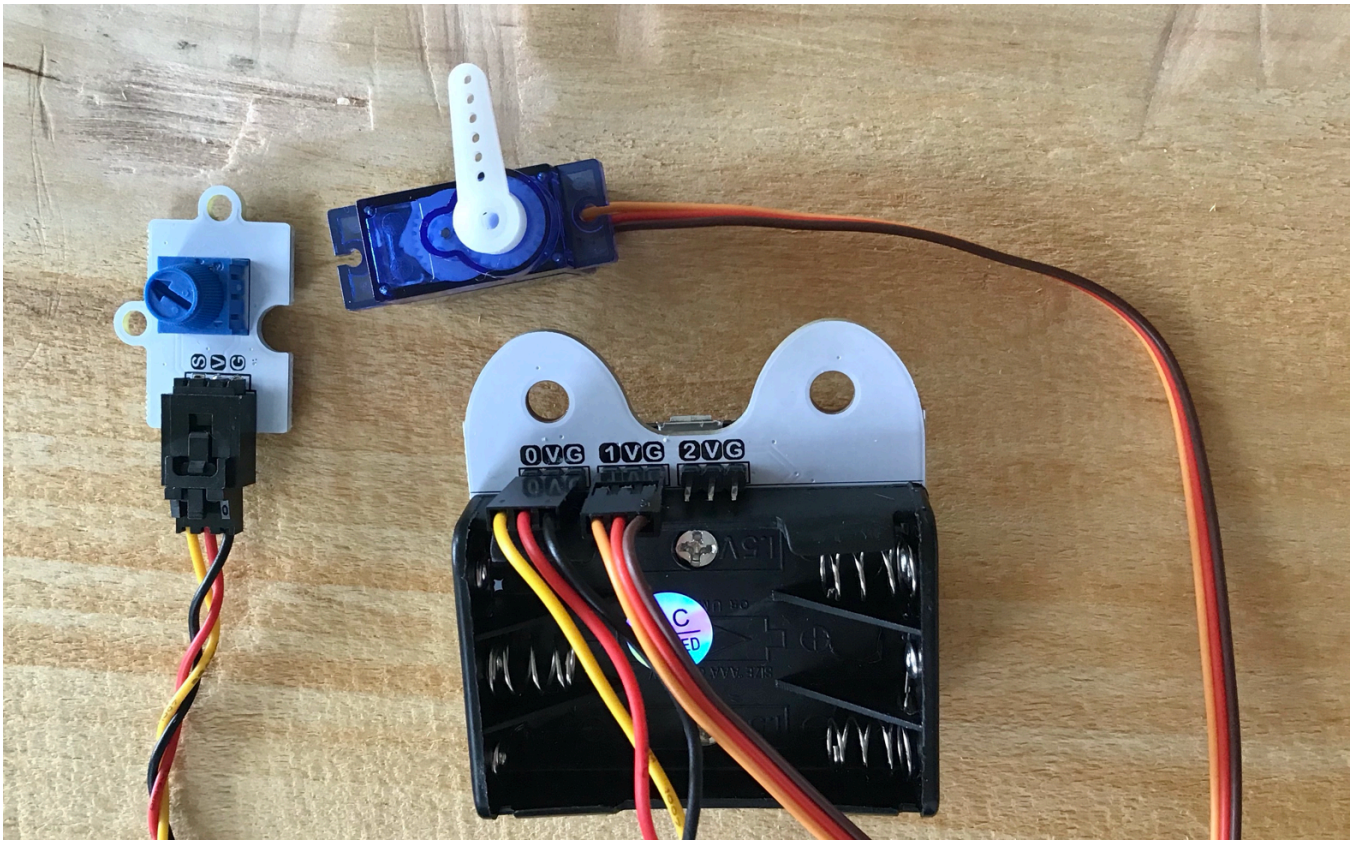


Figure: Connecting the servo motor to the micro:bit.

There are many kinds of motor. We will come back to this later. The one you'll use now is called a continuous servo motor. While the way it works will likely seem a bit obscure, it is a great choice to start with as it is very easy to control as it requires no extra components.

If you are just interested in making the motor turn, skip this paragraph. For the others, here is a little theory: Servo motors are controlled using pulse width modulation (PWM). PWMs are signals made out of regular high voltage pulses with varying width. The motor will measure the width of the pulses and will turn depending on it. In our case, to control the servo motor, we need to send a high pulse every 10 milliseconds. The width of the pulse needs to be 5% to 25% the period of the pulse (0.5ms to 2.5ms long) to be taken into account by the motor. More about PWMs here and more about continuous servo motors here.

Practically, to control the motor you will need to have pin1 output a high pulse with a 10 ms period. As the documentation explains, to do so we can use the instruction *pin1.set_analog_period(10)*. To change its speed you will have to use the function *pin1.write_analog(d)* with *d* in [50-250].

exercise 2

- 2.1. Program the micro:bit to make the motor turn. Try different values for *d* to make it turn at different speeds. What argument do you need to give to *the write_analog()* function to make the motor stop ? Does the function *pin1.set_analog_period(10)* need to be in the loop ?
- 2.2. Program the micro:bit so that you can control the speed of the motor by turning the potentiometer.
- 2.3. (Optional) Add a second motor and program the micro:bit so that you can set the speed of both motors independently using the buttons and potentiometer. Before programming, imagine how your idea would feel for the user to make it as user friendly as possible.

Notice that the motor is quite similar to the potentiometer in terms of wiring: it needs a power supply (3.3V and ground) and a signal, except that this time the signal is going from the micro:bit to the motor. This applies to many actuators.

Measuring a distance

The last component you will learn to interface today is the HC-SR04. The HC-SR04, while it is commonly called an "ultrasonic distance sensor" is both a sensor and an actuator. It sends an ultrasound signal when the controllers orders it to do so and it informs the controller when it receives an echo. Thanks to the time between emission and reception and a bit of physics (*distance = time * velocity*) we can estimate the distance of obstacles facing the sensor.

Do the montage shown in the pictures below. **Add the batteries only once you double checked the montage**. This montage is more complex than the previous ones and here is the reason why:

- The HC-SR04 needs about 5V of power and by adding the batteries the V pins of the buggy board go up from 3.3V to about 5V (more 3x1.5V in reality).

- The HC-SR04 sends an echo of 5V which if plugged directly to micro:bit could burn it. The montage **with the potentiometer set at its mid-position** allows to solve this problem[1].

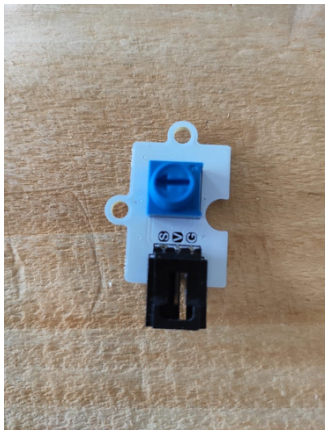
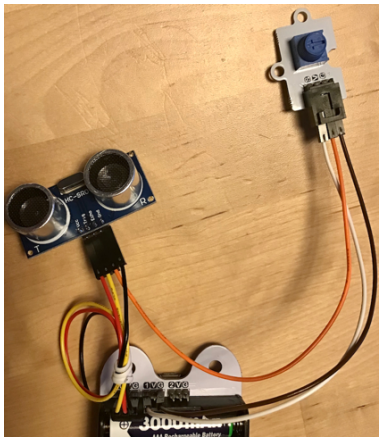
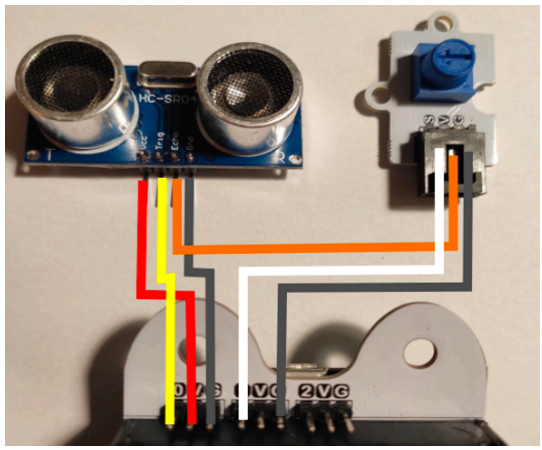
Potentiometer at its mid-position	Montage	Circuit diagram
		

Figure: Using the HC-SR04 with the micro:bit: set the potentiometer to its mid-position, do the wiring.

One way to use the HC-SR04 in our program is to use the following function:

```
from microbit import *
import machine
import utime

pinTrig = pin0
pinEcho = pin1
pinEcho.set_pull(pinEcho.NO_PULL) # some electronics that we can't explain

def get_dist_in_cm():
    # send a high pulse to the trigger of HC-SR04
    pinTrig.write_digital(0)
    utime.sleep_us(2)
    pinTrig.write_digital(1)
    utime.sleep_us(10)
    pinTrig.write_digital(0)
```

```
# measure the time in microseconds that it takes to receive a high puls
d = machine.time_pulse_us(pinEcho,1,10000)# timeout in 10000 us = 0.01
if d>0:# echo received
    return d/Constant
else:# timeout returns -1
    return d
```

exercise 3

- 3.1. Read the code to verify if it matches the sensor specifications that we defined previously. Will this work if I call the function *get_dist_in_cm()* ?
- 3.2. It won't work ! There is something missing: *Constant* is not defined and that's because I want you to find it now using the laws of physics.
- 3.3. Once you found the value of *Constant*, write a program to display the distance on the LED matrix using the function *display.scroll()*. What is the range of distance the sensor can perceive ? Don't forget to turn the switch of the buggy board on to test your program. When the switch is on you can actually unplug the usb cable and the circuit should continue working without the computer.
- 3.4. Display the distance on the LED matrix using the function *displayPositiveNumber(n)*.

Disco light (optional)

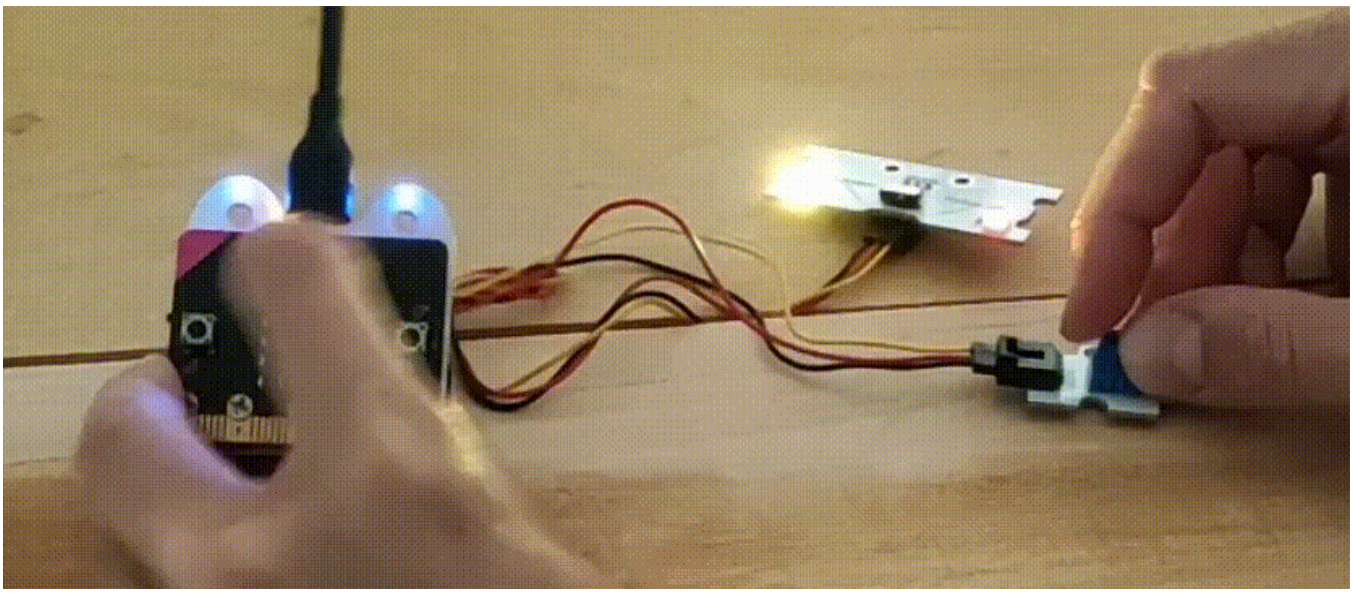


Figure: Controlling neopixel LEDs with the potentiometer (and showing the rgb values on the LED matrix)

If you finish the exercises in advance or if you are not tired enough and you want to learn more, one more thing you could do is to use the special extension board of the buggy, and learn to control it.

Warning: Before doing this activity **remove the batteries** from the buggy mainboard. Indeed, you will use the potentiometer as a sensor. If you kept the batteries it could provide, depending on the knob position, a voltage going from 0 to 5 Volts to the micro:bit. And as we learned previously 5V is likely to burn the micro:bit. By removing the batteries, the voltage will be in between 0 and 3.3V which is perfect.

The special extension board of the buggy comes with 2 neopixel LEDs which can be controlled thanks to the neopixel library. For instance, if the board is plugged on pin0, a simple program to light up one LED in green and one in blue would be the following:

```
from microbit import *
import neopixel

np = neopixel.NeoPixel(pin0, 2)
np[0] = (0, 255, 0)
np[1] = (0, 0, 255)
np.show()
```

exercise 4

Connect the neopixel and the potentiometer to the micro.bit. Program the micro:bit to change the colors of the LEDs with the sensors at your disposition, making it as user friendly as possible.

Conclusion

That's it for today. Turn the buggy mainboard off using it's switch to save the batteries for the coming days and unplug everything putting all the components back to their respective boxes.

With what you've learned today, you've got all the electronics tools you need to make an autonomous robot. While this morning you learned to interact with the controller of a robot, now you know how to give it eyes and muscles. Moreover while your familiarity with the use of functions is growing, reusing the work we did today will get easier and easier. The function *displayPositiveNumber(n)* made our work easier this afternoon to get a quick feedback from our sensors. All the work we did with the motor and the distance sensor this afternoon will make our work easier in the coming days.

Talking about ease: this afternoon we dabbled in the world of electronics. This can be intimidating at first. But like most other domains, by repetitive explorations out of your confort zone, you make your confort zone wider and soon this field can be yours to play with. For now, time to rest and see you tomorrow for some more challenges !

1. While we use the potentiometer as a sensor it is essentially a voltage divider. Therefore we can use it to lower the voltage of the echo signal down.

   The way we used it as a sensor was to connect it with 3.3V and the ground, and it outputed a portion of 3.3V depending on the position

of the knob, which is what we measured with the micro:bit. Instead, we can now feed it with the 0 to 5V echo signal coming from the HC-SR04, and make it output a portion of that, making sure it does not go over 3.3V. Also we need to take a sufficiently large portion of the signal, for instance imagine we outputed 0% of the signal with the potentiometer, then we would not be able to understand when the echo is received on the micro:bit. To get it to work we need the high value of the signal to be 2.3 Volt minimum (2.3 = 0.7 * 3.3... see here for more information). Therefore if we put the knob at its mid position, it should output a signal oscillating in between 0 and 2.5V and it should work. ↩

Mobile robotics - Control systems

Yesterday, you learned how to make a robot sense and how to make it act on its environment. But there was a big things missing to make the robot really move: a frame, a structure to hold all the parts together, we could say the bones of our mobile robot. Today we'll solve this problem: you'll build a frame, and thanks to this frame you will discover some standard mobile robotics problems and learn how to solve them.

What you will need

Every pair of learners will need:

- A computer equipped with a connection to internet and a browser (ideally Chrome),
- One micro:bit,
- A usb cable to connect the micro:bit to the computer,
- One bit:buggy kit,
- A flat piece of cardboard.

Building the Bit:Buggy

Read the manual	Collect the parts	Building	Done
			

Figure: Building the Bit:Buggy.

To build the robot, I invite you to follow the instructions provided in the Bit:Buggy isntruction manual. However this manual is missing some valuable information:

- Before starting identify the two bags that come along with the servo

motors (in them there should be screws and several servo horns, that are white plastic pieces to put on the servos). Those pieces won't be of any use and are not listed in the component list of the bit:buggy car, so put them aside.
- In step 1. pay attention to the wheels: their two sides are differents, only one side can plug onto the motor end. Also, when screwing the wheel onto the motor, hold the wheel (if you don't you could make the motor turn using the screws and that might damage the motor).
- In step 3. make the cables of the motors stick upward. Moreover you have to push through to really get the plastic parts completely in one another.
- In step 4. be delicate with the screw. You should not have to force. If it's hard to screw it in, it's probably that the alignment is not correct. If it is, unscrew first, correct the alignment and try screwing again.
- Before step 6 put the batteries in if it is not yet in, and turn off the board with the switch.
- After step 6, when you plug the cables, remember what you learned previously: the black cable plugs onto G ! The picture of the instructions might be a bit misleading. By the way the cables are long (not like the picture), simply tuck them inside the robot frame and you are ready to roll.

Moving basics

The robot has now some bones to hold together its muscles, eyes and brain, it's time for you to remember what you learned yesterday to do a simple program to make it move.

Exercise 1

- 1.1 Program micro:bit to move forward when the button A is pressed and stop when B is pressed.
- 1.2 Rewrite your program to make it clean and reusable: define the functions *setLeftSpeed(v)* and *setRightSpeed(v)* for *v* in [-100, 100] so that to go straight at full speed we just have to call

setLeftSpeed(100) and *setRightSpeed(100)*.

- 1.3 Change the functions *setLeftSpeed()* and *setRightSpeed()* so that *setLeftSpeed(v)* does the same thing as *setLeftSpeed(100)* for *v>100* and *setLeftSpeed(-100)* for *v<-100*.
- 1.4 Rewrite your program to make it even cleaner and reusable: define a function *goStraight(v)* that can be used to make the robot move forward (*v > 0*) or backward (*v < 0*) at the speed you want.

Control systems

Open-loop and closed-loop systems

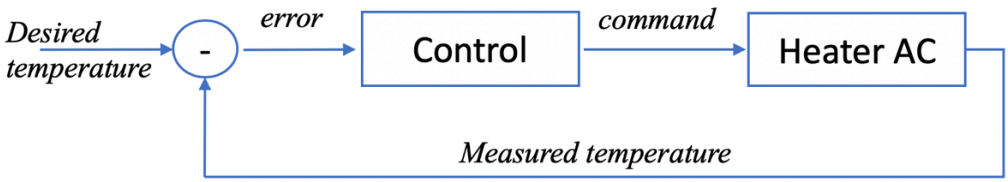


Figure: Closed-loop system of an heater AC.

In engineering there are two types of control systems. The first one is the open-loop system, also known as non-feedback system. It is the kind of systems that we used in the previous exercises: the outputs of those systems do not produce changes on their inputs. Here is an analogy with an air conditioner/heater: an open loop system in this case would be a system with three buttons, heat, cool down and stop, and the user would have to press the button whenever he wishes the machine to do something different.

The second type of control system is the closed-loop one. In this one, there is a feedback, from the outputs to the inputs. Coming back to the aircond/heater example, to use the previous machine in a closed-loop system, you'd have to equip it with a temperature sensor and a controller. One way to use such a system, could be to program the controller to activate the heater if the measured temperature is below a given value and activate the air conditioner if the temperature is above it. That's what

many of us have in our houses (without the air conditioner). In that case, the action of the heater would influence the temperature and therefore its measure. There is a feedback from the actuator of the machine to its input.

On/off clontrol law on the buggy

In this activity, you will learn how to program a closed-loop system using two different control laws. To do so you will use one more sensor which is integrated in the micro:bit that you did not use yet: its accelerometer. You will put the buggy on a slope and you will need to use a closed-control loop to avoid it flipping on the side. To know if the buggy is about to flip, you will need to look into the information that the accelerometer gives.

Question

Using the documentation of the accelerometer here find which of its axis informs us about the sideway (roll) inclination of the buggy.

Depending on this information, we can program the robot to turn left or right to face towards the top of the slope as presented in the figure below. Notice the feedback loop: turning has a direct effect on the roll and, as a result on the measure of the accelerometer.

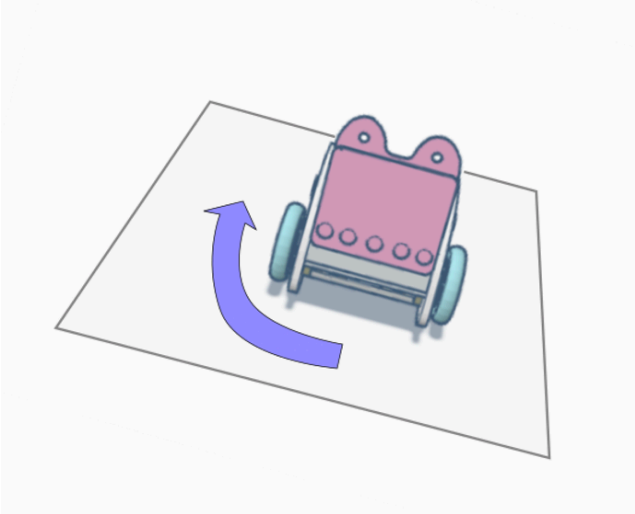
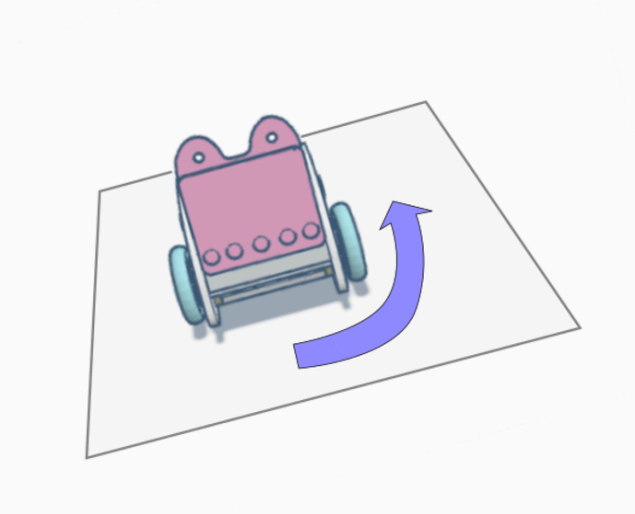
Roll	Roll
	

Figure: The buggy moves along the steepest ascent.

Exercise 2

- 2.1. Using *display.scroll()*, display the measure of the acceleration along the chosen axis. Tilt the buggy by holding it in your hands. What are the minimum values returned by the function *accelerometer.get_.()* ? What physical measure does that corresponds to ? What is the acceleration measure corresponding to no slope?
- 2.2. In the loop of the controller, define a variable *error* in which you will store the error between the desired measure along the chosen axis and its actual measure.
- 2.3. Program the buggy to move as described in the image above so that if the roll is over a certain threshold it turns towards the slope (which will reduce its roll) at 50% full speed. Test your program by putting the buggy on a piece of cardboard so that you can change the slope.

Propotional control law on the buggy

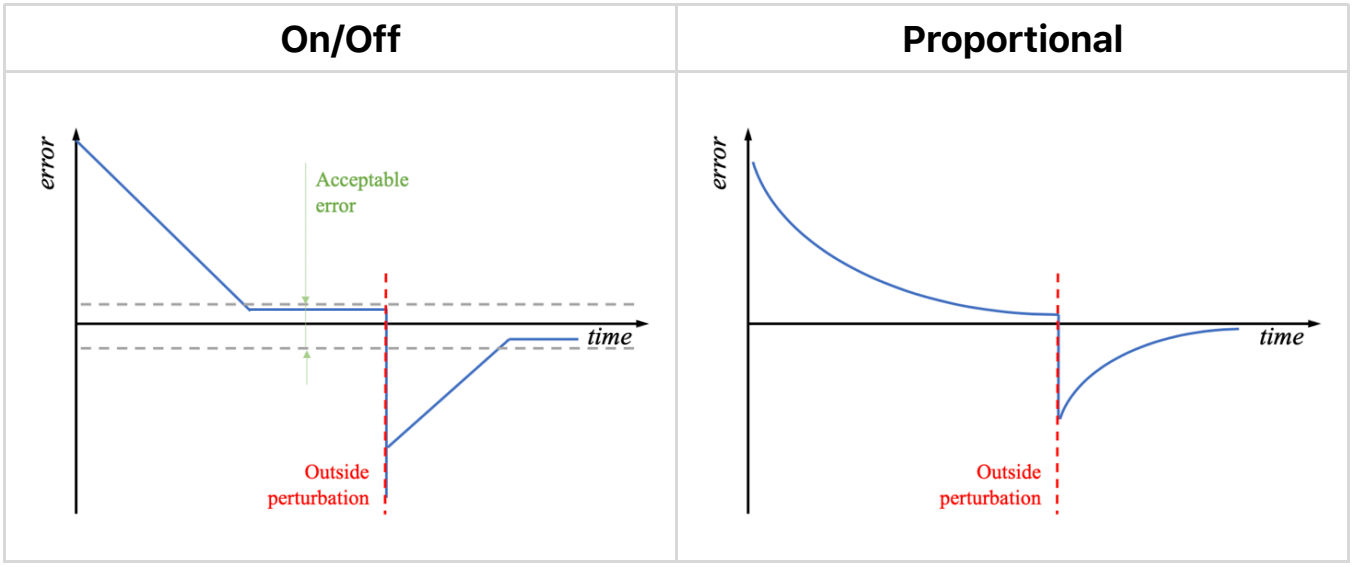


Figure: Evolution of the error with respect to time for both control laws.

Coming back to our heating analogy, the previous control law would correspond to this: the user sets a desired temperature or temperature range which is acceptable, if the measured temperature is lower than this the heating will be activated automatically, if the temperature is higher

the air conditioning will be activated. The problem with such a strategy (called an on/off control law) is that if the temperature is much lower than what is acceptable, if we could, we would prefer the heater to work much harder than if we were close to the desired temperature. This alternative is called a proportional control law.

As we are able to control how fast our robot turns, it is possible to set the turning speed so that the more the robot is tilted on the side, the faster it will turn.

Exercise 3

- 3.1. Use the error defined in the previous exercise to control the turning speed of the robot using a proportional control law. Warning, you will need to scale the error. Test your program.
- 3.2. (Optional) Change you program slighly so that when the buggy is flat it stops and when ts pitch increases it goes forward faster and faster.
- 3.3. (Optional) Combine the exercises 3.1 and 3.2 to make the buggy move toward the top of the slope it is on.

Question

Is the error ever converging to zero ? If not why could that be ?

Conclusion

Control systems are everywhere. We have more and more robots around us because they do what we want them to do but also because they can easily make things as we want them to be, and this is thanks to control systems and control laws. While the proportional control law that you programmed this morning is a must in mobile robotics, we often need to go beyond that. As you probably noticed, the error can not converge to zero with such a simple control law. To really make it converge to zero we need more complex laws, namely PID, standing for **Proportional-Integral-Derivative**. If you ever wondered why you are studying integrals

and derivatives, have a look at PIDs (here is a great ressource) and you'll find one reason to do so.

Engineering - Remote controlling

Before building a mobile robot such as the one we build this morning, an engineer is likely to first study what controller he can pick, choose one and test it. Once this is done, he might study the available motors, choose one type of motor, and make sure he's able to control them. Only once he mastered every individual elements will he put them together. Problem decomposition is a key skill for every engineer.

In the previous activities, the problems to solve have always been decomposed in simpler problems for you to advance safely in them in a similar fashion. Indeed this is standard in pedagogy too. Therefore, you have already been exposed to this process a lot. This afternoon's goal is to make you practice doing this problem decomposition yourself.

What you will need

The material is the same as this morning's material, except for the cardboard which is not needed. However , as we'll make two micro:bits interact, you will have to be at least two pairs (with one micro:bit each) and to organise yourselves among pairs.

Remote controlled buggy

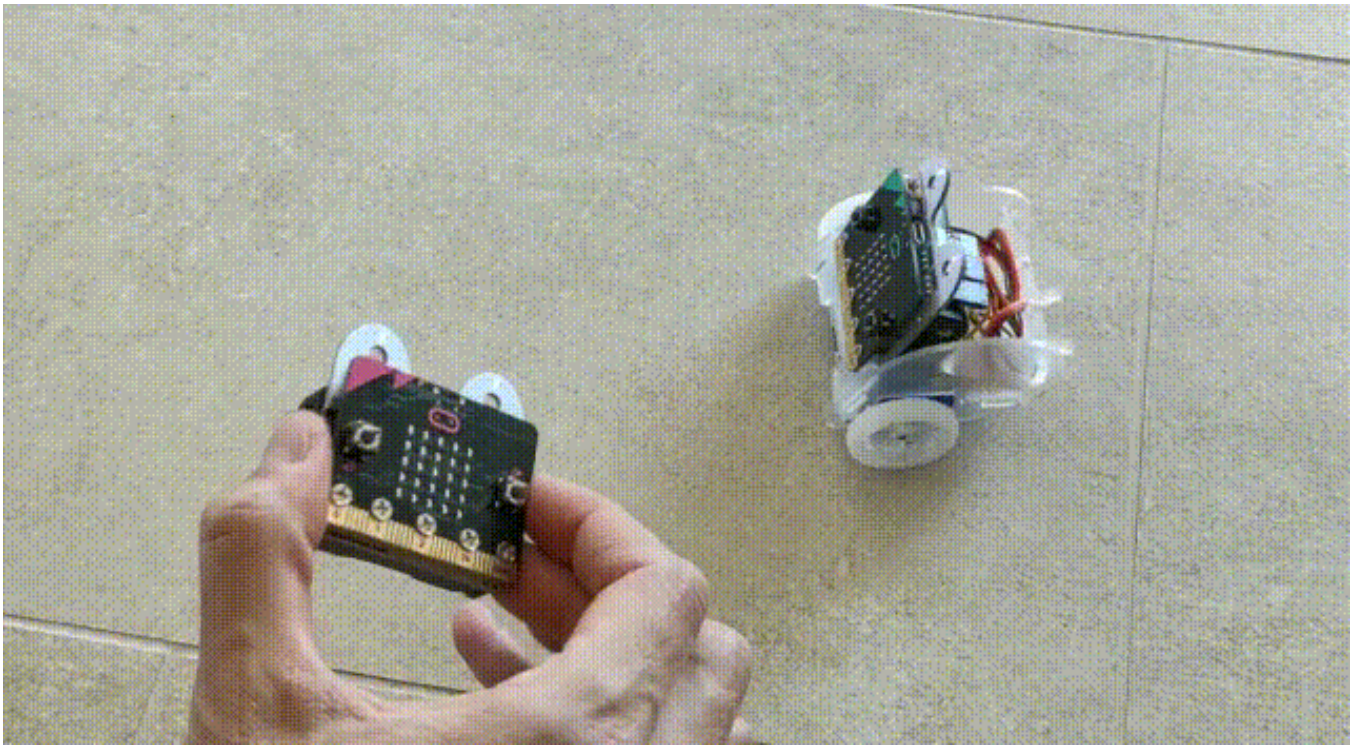


Figure: Radio controlled Buggy

The problem you will practice your problem decomposition skill on will consist in programming a remote controlled buggy: the remote control (or joystick) will be a micro:bit, which will send its commands to the buggy through radio waves: tilt the joystick forward and the buggy moves forward, tilt the joystick left and the buggy turns left...

Exercise

Define the milestones that you and your team will have to reach to eventually have the whole problem solved. In other words, find many smaller problems to solve that will help you solve the bigger problem at hand. As you will need to have two micro:bit to do your experiments, your team will have to be composed of at least two pairs of learners. The documentation of the radio emitter and receptor of the micro:bit can be found here.

▼ Solution

After some time, we will provide you with a proposition of milestones.

More projects

If you managed to program the micro:bit as asked in the previous project, here are some more ideas you can work on to develop your engineering skills:

- Project 1 (incremental): Control the buggy using another micro:bit equipped with two potentiometers (remove the batteries if you do so to make sure you don't give 5V to micro:bit and burn it). One potentiometer for the forward/backward speed, the other for turning (like on many RC vehicles)

- Project 2 (hard): Program the buggy to always come back to the orientation it started with. For this you'll need to use it's internal compas which is quite a special sensor.

Conclusion

This afternoon you focused on problem decomposition. This strategy is not only useful in engineering but in many, many other fields. As a **sportsman**, if you want to learn a new move, if you have the chance to have a coach, he will help you break down this new move in simpler moves. If you don't have a coach then you will increase you chances to do the move right in decomposing it in simpler movements and pratice them individually. As a **musician**, before learning to play a tune, you will benefit in learning to play scales beforehand, as boring as it might seem.

While it is easy to disregard such practices as boring, we should not forget their power: the smaller problems we solved or the simple skills we develop are often at the fundation of many other bigger problems or skills. Once we mastered one scale, it is not one tune that we can play with more ease but all the other tunes in this scale. And the same applies for problem solving in engineering.

Designing a robot

In the previous days we learned how to control a robot, to make it move, to add sensors to its controller, but we did not yet manage to have what we could call an autonomous robot. Today you will learn how to make this possible.

Question

With everything you learned so far and the material at hand, do you think that you could put together an autonomous mobile robot that will explore its surrounding while avoiding obstacles ?

▼ Answer

Yes you might be able to. Indeed, while you can't connect the HC-SR04 and 2 motors on the buggy board, you can connect the ultrasonic sensor on one board and two motors on another board and make the two board communicate. The problem is that the robot frame of the buggy does not allow that. Today you'll learn how to start prototyping andd building your own frames to solve such problems.

What you will need

Every pair of learners will need:

- A computer equipped with a connection to internet and a browser (ideally Chrome),
- One micro:bit,
- A usb cable to connect the micro:bit to the computer,
- One bit:buggy kit,
- Some cardboard,
- A ruler,
- A cutter.

Cardbot

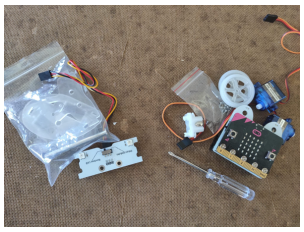
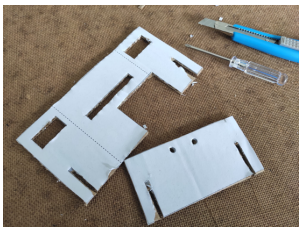
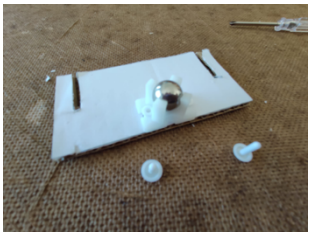
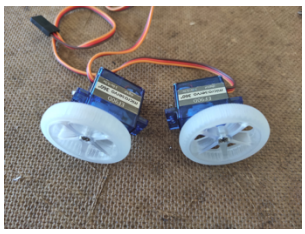
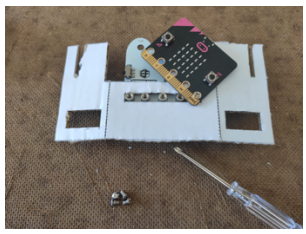
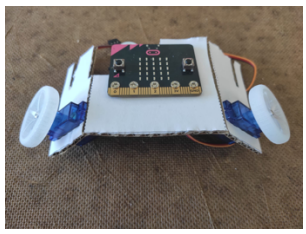
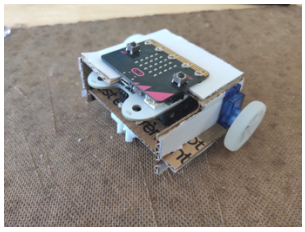
Disassemble the buggy	Glue the simple plan	Cut the parts (dashed lines = fold)	Mount the universal wheel
			
Assemble the wheels on the motors	**Mount the buggy board**	**Fit the motors and plug them**	**Assemble the top and bottom parts**
			

Figure: Mounting the cardbot. If you do not have the print of the cardbot plan, you can download it here.

You will start to learn how to design a cardboard robot. While cardboard is neither noble nor durable, it is a great tool to prototype and to learn the mechanical constraints in play in the designing process. By manipulating cardboard parts in your hands, you will also prepare yourself to manipulate them virtually while using computer aided designing softwares, which will be the topic of this afternoon.

Exercise 1

- 1.1. Build the *cardbot* by following the instructions given in the image above. Note that when disassembling the buggy, a good practice is to put the discarded elements back in the buggy box, taking care of putting the screws back in their plastic bags.
- 1.2. Once the cardbot is built, test it using the program to make the robot climb a slope.
- 1.3. As a designer, you'll have to pay attention to how user friendly your robot is. Is this design user friendly ? Can you for instance

easily switch it on and off ? Could you improve the design ?

Obstacle avoiding robot design

In the previous activity, you became more familiar with the use of cardboard to prototype and build. You will now build upon what you just learned and propose your own design to solve the original problem: build an obstacle avoiding robot.

In terms of electronics, you'll need to fit on the frame one micro:bit to look for obstacles using the HC-SR04 and the potentiometer (as in monday's afternoon activity) and one equipped with the motor to take care of the displacement. The "seeing micro:bit" will send it information to the "leg micro:bit". For this, there is a proposed design at your disposition, but try first not to use it and come up with your own design as proposed in the following exercise.

Exercise 2

- 2.1 **Design:** design (without building) a new cardboard frame for your robot to be able to move and sense obstacles. For this we recommend you to use a pen and paper, but you could also use an online tool such as Method of Action which allows you to design in vectorial format (and if you have a printer and you know it well you could print something similar to what we provided you with and make the building process easier).
- 2.3 **Program**: Program the mobile robot to move forward when there is no obstacle closer than 10 cm and stop otherwise.
- 2.4 Program (optional): Program the robot to explore the environment while avoiding obstacles.

Conclusion

This morning you learned how to design and build a robot frame with simple material. While there is a lot to learn in the field of mechanics, the theory is based on some strong foundations in physics and geometry.

However, there is no need to get involved in the theory too early. You can already learn a lot of practical knowledge in doing what you did this morning, and this will only makes you learn the theory more easily later if you are interested in this field.

Computer-aided design (CAD)

While this morning you learned a great way to get familiar with standard mechanical problems of robotics and an easy way to prototype a design, you will often want and need to build something clean. For this you could learn machining, but you could instead learn to do the same with the help of computers which is what you'll do this afternoon.

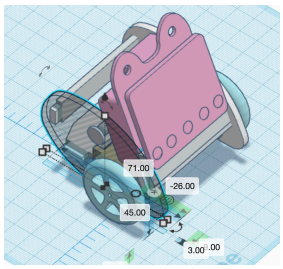
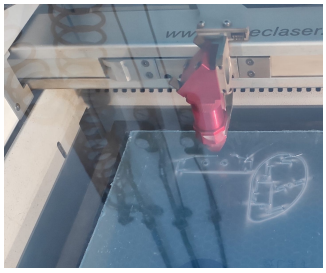
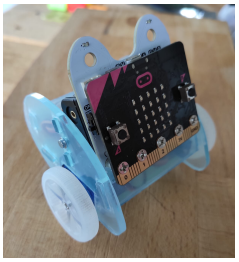
Drawing	Cutting	Cutted parts	Mounting
			

Figure: Producing a plexiglass buggy.

To build a frame using computers, the process consists of two phases: first, designing using a CAD software, secondly, building using a machine that accepts to take as input your CAD. Today you'll learn how to do the first phase using an online software called Tinkercad. You'll do this with the goal to eventually cut your model in 3mm thick sheets of plexiglass using a laser cutter as shown in the picture above (you can get access to those in fablabs which are becoming more and more common in Switzerland). Starting with laser cutting has the advantage of being very similar to the cardboard you used this morning in terms of mechanics.

What you will need

Every pair of learners will need:

- A computer equipped with a connection to internet and a browser.

Introduction to Tinkercad

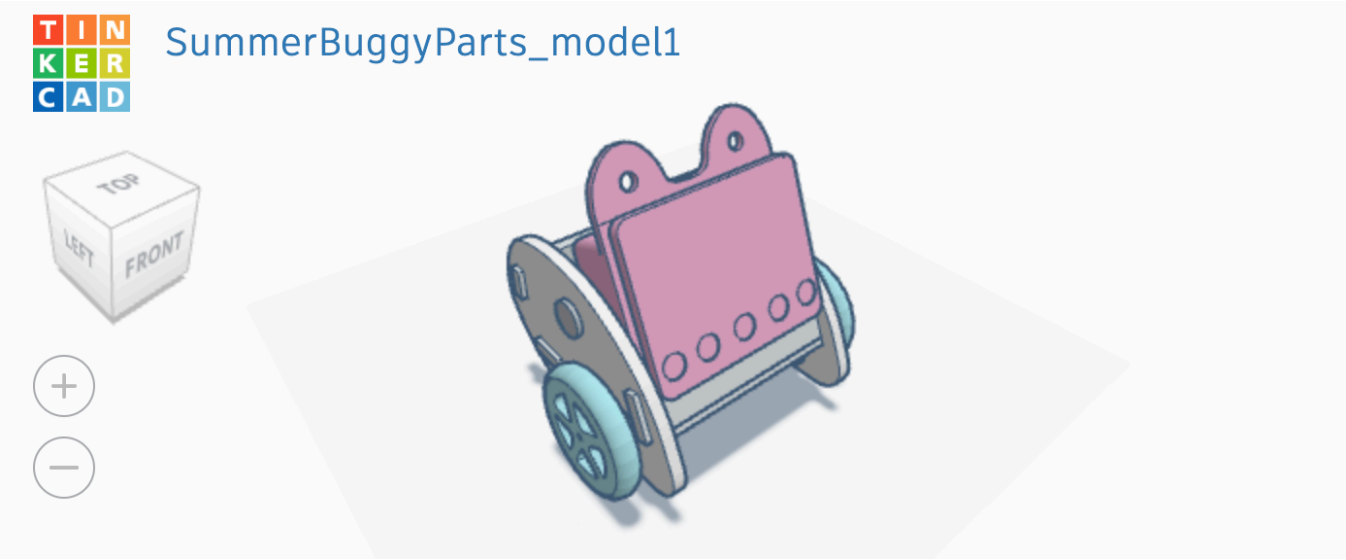
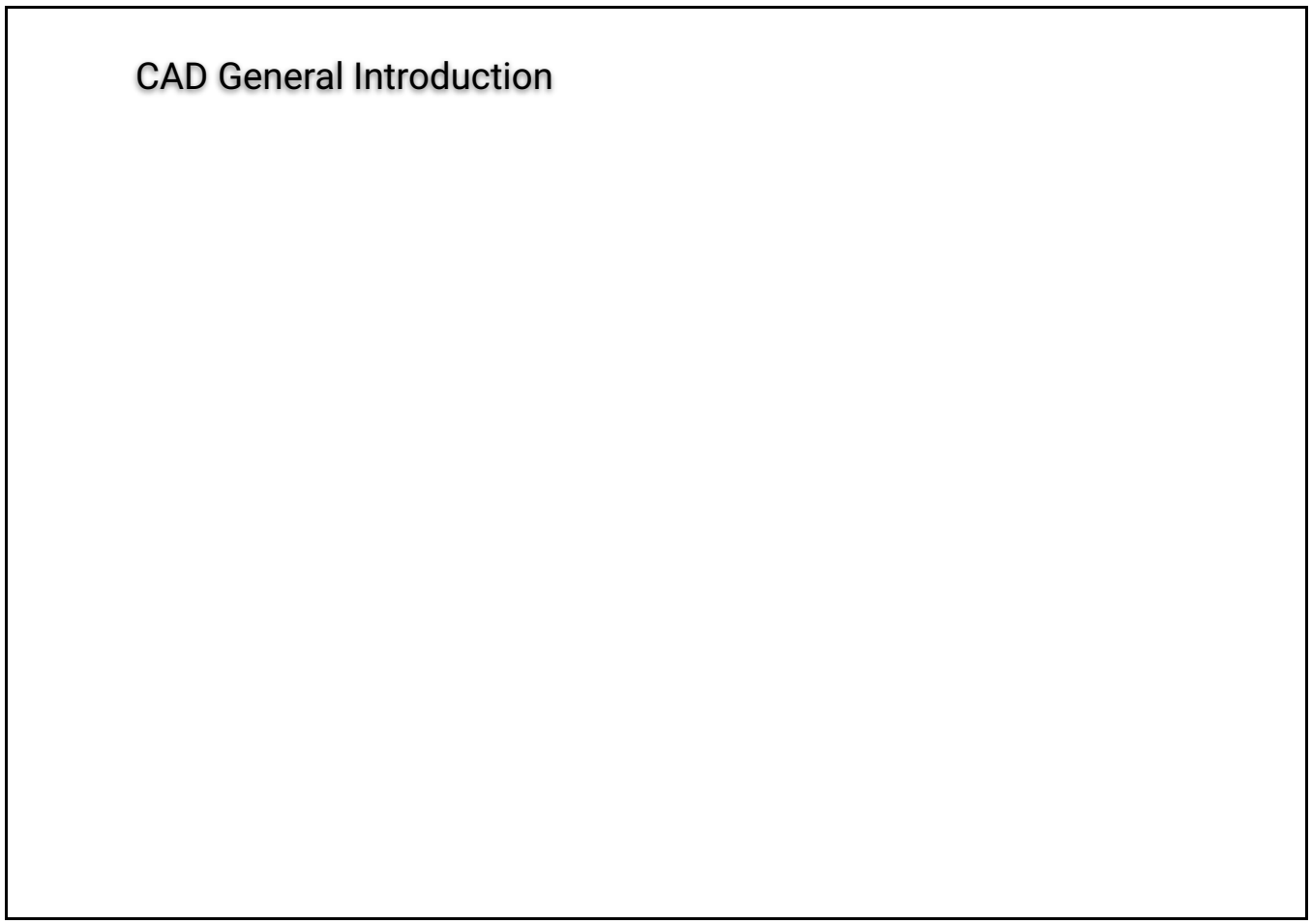


Figure: Objective: being able to design a buggy in 3D.

In this activity, you will learn the basics of CAD designing by reproducing the 3D model shown above of a robot very similar to the one we used this week. For this, go onto this virtual classroom. This will allow you to access Tinkercad using only a nickname that your teacher will give you. That will also allow us to follow your progress and help you if you need.

Then follow the videos in the following playlist, which consist of a quick general introduction to the activity, an introduction to Tinkercad and a tutorial to design a mobile robot. Please follow those videos on youtube directly to have access to the video descriptions. And excuse my lack of experience with such a teaching format. I had much time to develop my skills in robotics, but much less in youtubing !



Congratulations, if you followed the steps along, you should by now have designed your first mobile robot in plexiglass. With the skills you developped along the way you should be able to put together most of the sensors, controllers and actuators you'll ever want to use.

Project

The project will be given to you during the week.

Conclusion

Going further

Hopefully the previous activities raised your curiosity, and you'll be confident in your ability to learn more in autonomy. If that's the case, here are ways you could follow to deepen your knowledge and practice in mobile robotics while still taking incremental steps.

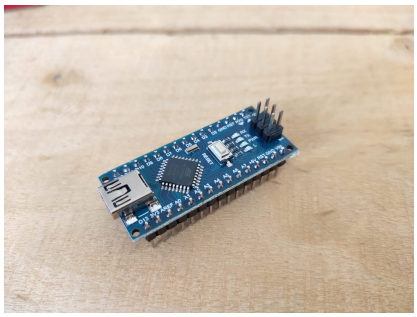
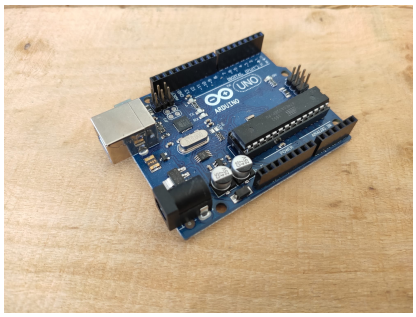
Programming

Combining knowledge about electronics and mechanics, we ended up building a mobile robot that can sense obstacles. Using this robot we could have solved many more complex problems than just moving forward when there is nothing in front and stop when there is something.

Challenge : Program a robot to explore its environment

Develop your algorithmic skills by solving more complex problem with the robot we built. For instance, you could program it to explore its environment randomly while avoiding obstacles.

Controllers

Arduino Nano	Arduino Uno	Raspberry
		

While we used the micro:bit this week, most of what you learned can be applied to other controllers. Among the popular options, you will often hear about the Arduino and the Raspberry Pi which are great options to continue your exploration of mobile robotics.

Using the Arduino you could for instance more easily use a very wide

variety of sensors thanks to all the libraries available for it. You will also have more easily access to the lower level architecture of the controller. One simple example is the use of interrupts which is very standard in mobile robotics but which is not available in python with the micro:bit.

Raspberry Pi is different. It has GPIO so that you can connect it to sensors and actuators but... it is a computer. So when you start it, it is not running a very basic program with a while true loop, it runs a whole operating system. It comes with great advantages like being able to use a camera, which is a really useful sensor for mobile robotics, but it comes at the cost of being much more complex to setup and harder to control its GPIO in real-time.

Challenge : Create an autonomous robot based on Arduino

Now that you know how to use servo motors, the HC-SR04 and how to design a robot frame, you can built a robot using Arduino. The only thing you'll have to learn is to program in C++ and find a way to power the arduino (I'd recommand using a power bank).

Sensors

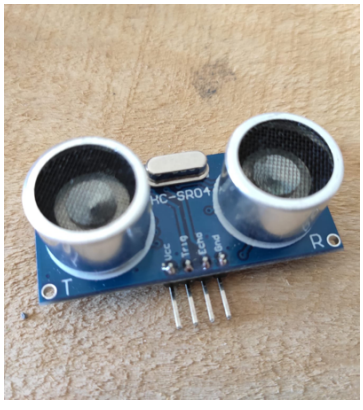
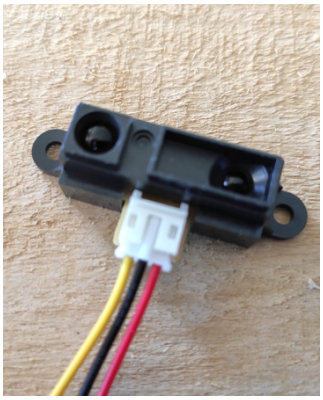
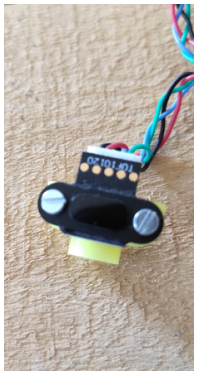
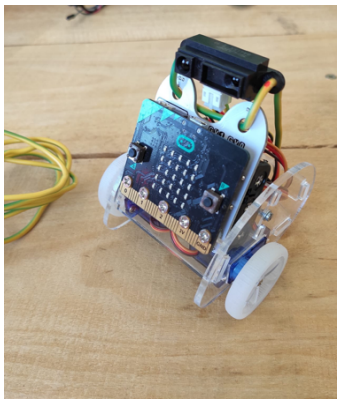
HC-SR04	Sharp sensor	TOF10120	Buggy with sharp sensor
			

Figure: Different distance sensors and an application

During this summer school you interacted with different sensors: buttons, a potentiometer, the HC-SR04, the accelerometer, the radio

receptor. This already covers quite a few types of sensors: active/passive, exteroceptive/proprioceptive (read more on this topic here). But those are just a few in so many different types of sensors. With micro:bit, as a distance sensor for instance, you could use a mechanical endstop (a bumper), or an infrared distance sensor, or a GP2Y0A60 or still a TOF10120! In doing so you'll learn not only different protocols to interface sensors but you can also learn the physics behind and link it to what you learn in school.

Challenge : Using new sensors

- Learn to use new distance sensors (sharp, TOF). Can you identify their advantages and disadvantages ? (At least a big advantage of the GP2Y0A60 is that it only requires one pin working with analog read so that you can mount it on the buggy).
- Make a robot that follows the light. For this keep the same controller, the micro:bit, and learn to use new type of sensors.

Actuators

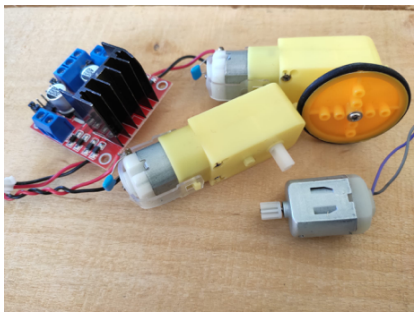
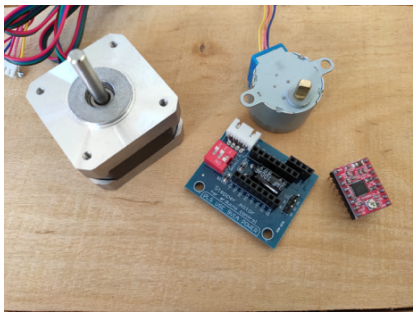
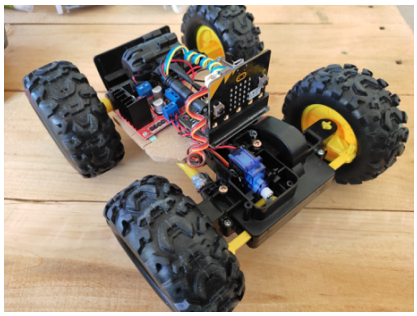
DC motors and L298N	Stepper motors and A4988	Upcycled RC car
		

Figure: Different distance sensors and an application

In terms of actuators, you already managed to control LEDs, continuous servo motors and a radio emitter. While LEDs are relatively standard to start with, it is less the case for continuous servo motors. Indeed those are servo motors that have been modified (and servo is actually a misnomer for them). So if you wanted to expand your possibilities with

robotics while staying in your comfort zone, standard servos could be a great options. As their angle of rotation can be controlled, with them you can for instance build robotic arms.

Some other options for motors could be to use DC motors or stepper motors. To use such motors you will usually need some extra electronics. For instance the L298N is a common in Arduino project involving DC motors, but works equally well with micro:bit. The advantage of DC motors is that they can go really fast. For stepper motors you will usually need a stepper driver like the A4988 (note that the minimum operating voltage of the A4988 driver is usually written to be 8V but it can work with small stepper motor on 5V). The advantage of stepper drivers is that you can control their rotation precisely, step by step, hence their name.

While the use of extra electronics can seem intimidating at first, don't get fooled, thanks to modular electronics the process involves mostly skills and knowledge similar to the ones in computer science. Each module can be seen as a function and your goal is to use those functions appropriately (one difference with computer science though is that if you make a mistake you might have more than a bug... like burn your pieces).

Challenge : Using new types of motors

- Mount the HC-SR04 on a standard servo motor, then design and program a movement alarm. The sensor is mounted on the motor so that you can detect any distance difference in time with a 180° field of view.
- Pump up your buggy and make it go much faster (for this you'll probably need more than 3 pins). For this you will probably need DC motors.
- Make a robot which is more precise in its displacement, so that for instance you can not only make it turn but also control at which angle it will turn (for this you'll probably need more than 3 pins too). For this use stepper motors.

Extension board, electronics and battery

management

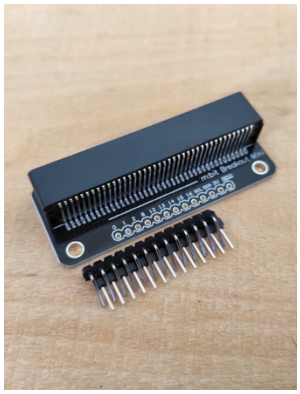
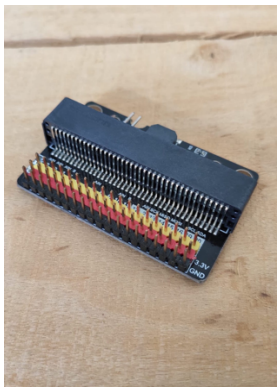
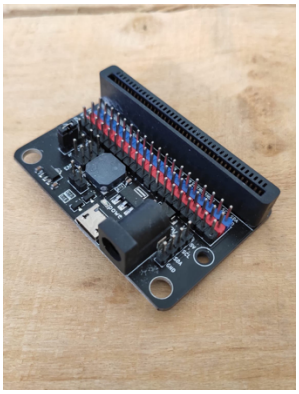
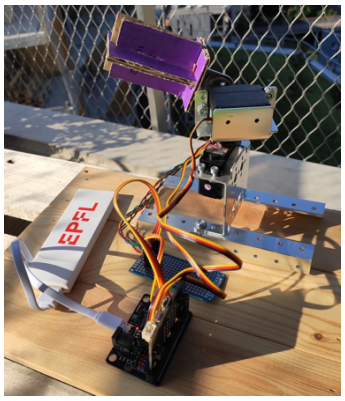
Extension with no power input, pins to solder	Extension with no power input	Extension with usb power input	Solar tracker from extension with power input
			

Figure: Different types of extension boards among many others and a project made possible with those.

To solve standard robotics problems (like building an avoiding obstacle robot), we saw that the buggy mainboard falls quickly short of pins. This limitation is not due to the micro:bit, but to the extension board. The micro:bit has about 20 pins which can be used, and to use them you only need another extension board. However the buggy mainboard offers more than an easy access to the pins, it also offers a power supply.

To tackle more complex robotics problems with micro:bit you will therefore need to find not only an extension board but also find a way to power it up if the battery does not come with the board which is often the case. If you don't want to get into soldering yet, then one option could be for you to use a board that gives you access to many pins with pin headers but also that comes with either a battery or with a usb connector which will allow you to power the board using any standard power bank. If you want to get in soldering, then the world is yours.

Challenge : Use more pins of the micro:bit

Find an extension board and adequate power supply to make an obstacle avoiding robot with the HC-SR04 and one micro:bit.

Making

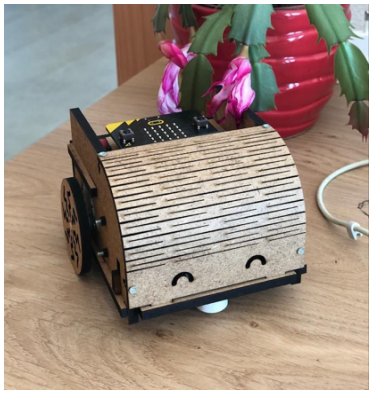
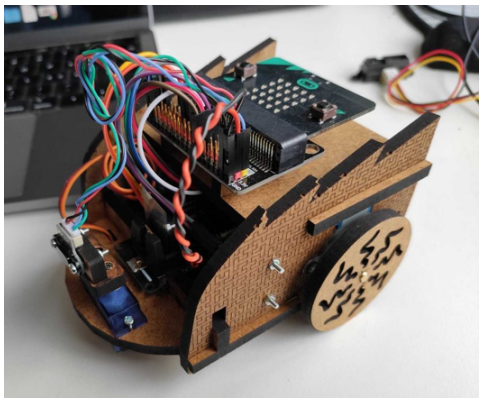
Laser cutted MDF frame v1	Laser cutted MDF framev2	3D printed frame and wheels
		

Figure: Playing with wood, 3D printing and electronics

While you have made your first steps into prototyping with cardboard and designing with CAD, the possibilities to go further are endless with both. Search for "Cardboard robot" on youtube and you'll probably be amased at the creativity and the patience of some authors. As having the tools to work with other materials is not a chance everybody has, reproducing similar works is a great way for you to get comfortable with the basics of mechanics and designing.

Finally, a step further into CAD could be for you to design anything you can imagine, go on a website like instructables if you temporarily lack inspiration, and of course go into the closest fablab to learn how to handle laser cutter or 3D printers, and hopefully meet a community of people thirsty for learning. Although if you enjoy some quietness, 3D printers are getting very affordable those days and are now cheaper than most smartphones so you don't even need to go to a fablab.

Challenge : Make

- Build a walking robot out of cardboard.
- Get in the closest fablab and laser cut or 3D print your own frame to hold all the electronics together.

About the author

After a few years in computer science research, working on very stimulating subjects such as autonomous robotics, augmented reality, 3D reconstruction and machine learning, my pleasure to share my enthusiasm for computer science has been growing. So, from computer science research I made a smooth transition to teaching it.

I enjoy computer science because it offers me an infinite source of puzzles, but it's not the only reason. I think that understanding computers and having skills in the field is paramount to understanding our world which is becoming more and more digital. So I wish for everyone to know that the computer science world is open to them. and I hope to give the keys to this world to as many people as possible in the future.

Luckily I joined a wonderfull team in EPFL which has a similar goal as mine (among others). That's how I ended up co animating the Thymio mobile robotics summer school with Professor Francesco Mondada to progressively animate the whole program. Finally COVID19 forced us to search for an online format, which led me to propose a new form of summer school based on micro:bit.

Collaborators

This summer school would not have been possible without the support of the Service de Promotion de l'Education of EPFL. It also wouldn't have been possible without the trust that Prof. F. Mondada gave me and the opportunities he gave me co-hosting, animating and tinkering in the previous editions of the summer school.

I'm also very thankful to Anthony Guinchard for his thorough proofreading of this whole website and the time he spent testing all the activities. I'm thankful to him as well as to Christian Giang and Aditya Mehrotra for the inspiration they gave me to integrate a makers touch

and for their help in the preparation of the animation.

Contact

Find the information to contact me on my professional page.