Lesson 10:

Parking aid

Subject: *Computer Science, STEAM* **Grade(s):** *7th and up*

Duration: *45 minutes* **Difficulty:** *Medium*

Lesson Objectives

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

- *Program complex driving movements in a sensor-controlled manner*
- *Acquire, process and analyze measured values*
- *Modify program sequences with the help of sensor data*
- *Solve technical problems and present different solution strategies*
- *Recognize & formulate algorithms*

Overview

Modern assistance systems make people's everyday lives easier and help them avoid having to worry about routine tasks. In the area of traffic, such systems help to avoid accidents or to find parking spaces. This exercise involves using the mBot2 as a vehicle that searches for parking spaces in "parking" mode. In the next step, sensors are used to check whether the parking space is large enough. If everything is correct, automatic parking starts.

The exercise deals with data acquisition and evaluation, uses several sensors to coordinate the different program sequences and offers several possible solutions.

It does not include complete solutions, but explains key concepts and code examples so that students can develop the entire algorithm and program their code rather than just replicating a given solution.

Focus

By the end of the lesson, students will know:

- *Acquisition and evaluation of measured values*
- *Control structures*
- *Variables (status variables, storage of measured values, calculations)*
- *Development of algorithms*

Pre-lesson Checklist

What do you need?

- *PC or laptop (with USB output) with the mBlock software installed, the web version (also for Chromebook), or a tablet with the mBlock app installed*
- *The mBot2 with a CyberPi*
- *A USB-C cable or Makeblock Bluetooth dongle*
- *Several boxes or cartons to simulate parked vehicles and parking spaces*

Lesson plan

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

Duration	***Contents***
5 minutes	***1. Warming up*** • *Connection with everyday life* • *Steps during the search of a parking space* • *Implementation possibilities with the mBot2 robotic system*
10 minutes	***2. Hands-on*** • *Required coding blocks* • *Derivation of individual command sequences* • *Test of the individual steps*
25 minutes	***3. Trying out*** • *Independent extension of the tested program steps* • *Test and further development of the parking assistant*

5 minutes	***4. Wrap-up*** • *Showtime: show what you did with your robot in a fun, short movie for later discussion* • *In consultation with your teacher: post on social media channels with the hashtag #mBot2inclass* • *Reflection: Which development steps were particularly successful, which can be improved?*

Activities

1. Warming up (5 min)

Step 1: Warming up

This step consists of two parts:

1. *Assistance systems and modern traffic systems*
2. *Steps in the search for a parking space - implementation possibilities with the mBot2*

1. Assistance systems and modern traffic systems

Assistance systems relieve the driver by taking over routine tasks or reacting faster in emergency situations than humans can. Such systems can, for example, maintain a constant speed and still prevent a rear-end collision, monitor the sides of the vehicle when overtaking or turning ("avoiding blind spots"), or even help with parking. The last case will be simulated with the mBot2 robot vehicle.

2. Steps in the search for a parking space - implementation possibilities with the mBot2

The students should first describe the process of searching for a parking space in bullet points - from these notes a structural representation of the program (UML or flowchart) can be made in the course of the exercise, in which a check of the width of the gap and a renewed search are integrated.

The first steps are about the rough concepts:

- *Locating a gap*
- *Measuring its length*
- *Starting the parking process*

For these steps, the students make suggestions for implementation (which sensors, how to evaluate measured values). The situation of finding a parking space can also be done "by hand" with the mBot2, so that the individual steps can be traced.

2. Hands-on
(10 min)

Step 2: Hands-on

In the second step of the lesson, the different code blocks used for a parking assistance system are presented.

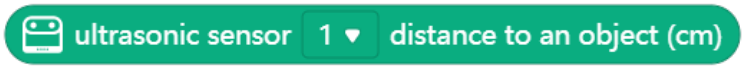
The step is divided into three parts:

1. *Required code blocks*
2. *Derivation of individual command sequences*
3. *Test of the individual steps*

1. Required code blocks

The following code blocks are needed to find a parking space and measure it.

Code block:



This block measures the distance to an object by ultrasound: for this purpose a fast sequence of inaudible sounds is emitted and the time until an echo arrives is measured. Using the speed of sound (in air), the distance to the object can be calculated. The sensor already provides this result.

In this use case, two ultrasonic sensors are installed on the mBot2: the first one is directed to the front, the second one to the right side. With the selection field one specifies which sensor should measure....

Code block:

The motors on the mBot2 count the wheel revolutions. The measurement is done in degrees, so one full rotation of the wheel corresponds to a full circle with 360°. If the motor rotates further, the value increases, if the mBot moves back, it decreases again.

At the start of the program the rotation is zero - but the mBot does not start at a parking space, but must first

find it. With this code block you set the counted revolutions back to zero.
Code block:    *This report block indicates how far the left (right also selectable) motor has rotated since switching on or reset-ting. From this angle in degrees the distance can be calculated.*

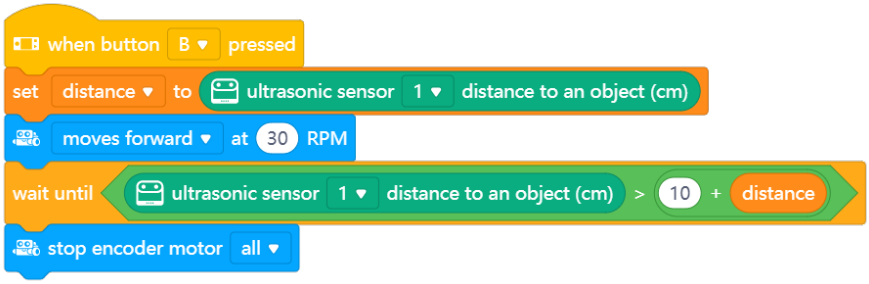
Code blocks that control the driving behavior of the mBot2 have already been introduced in lesson 1 of the Getting Started Activities.

2. Derivation of individual command sequences

Finding a parking space requires that a sensor on the side (right) can measure the distance of its own vehicle (mBot2) to stationary vehicles or objects.

Since sound spreads out in front of the sensor similar to a cone, objects that are slightly offset to the side are also detected. This still plays a role in measuring the parking space and the position of the mBot2... Furthermore, it is important to know that the sensor cannot properly detect distances smaller than about 5 cm. So the robot should not drive too close to the standing "vehicles" (boxes, cartons) when searching for a parking space. It is best to position the mBot2 with a distance of 7-8 cm next to these "obstacles" in such a way that a drive straight ahead is parallel to the simulated parked vehicles. The starting position (distance) should remain the same so that the driving movements can be adjusted for parking.

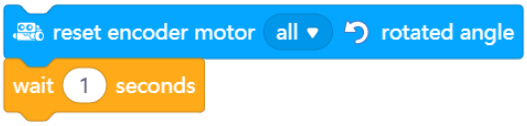
A parking space can be detected when a sudden and very significant increase in the lateral distance is detected by the sensor. Since the actual distance to the vehicles is not known in advance when the program is started, a variable must be used:



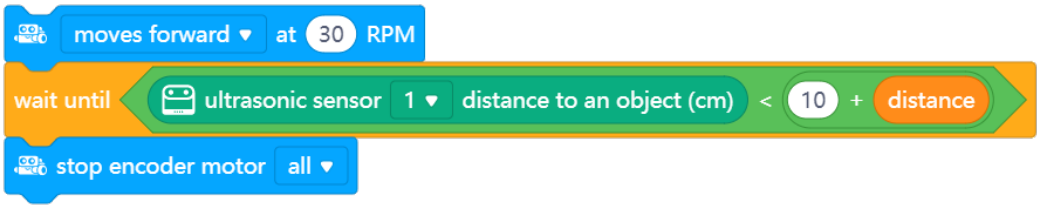
In the "distance" variable, the current distance to vehicles or objects on the right side is stored, then the mBot2 moves slowly forward. The following program sequence is stopped until the distance increases abruptly; in the example this is 15cm more than the original value (15+distance). Then the movement stops.

The stop is helpful to mark the position of the mBot in relation to the detected gap, later a stop at this position (and further pauses) can be omitted.

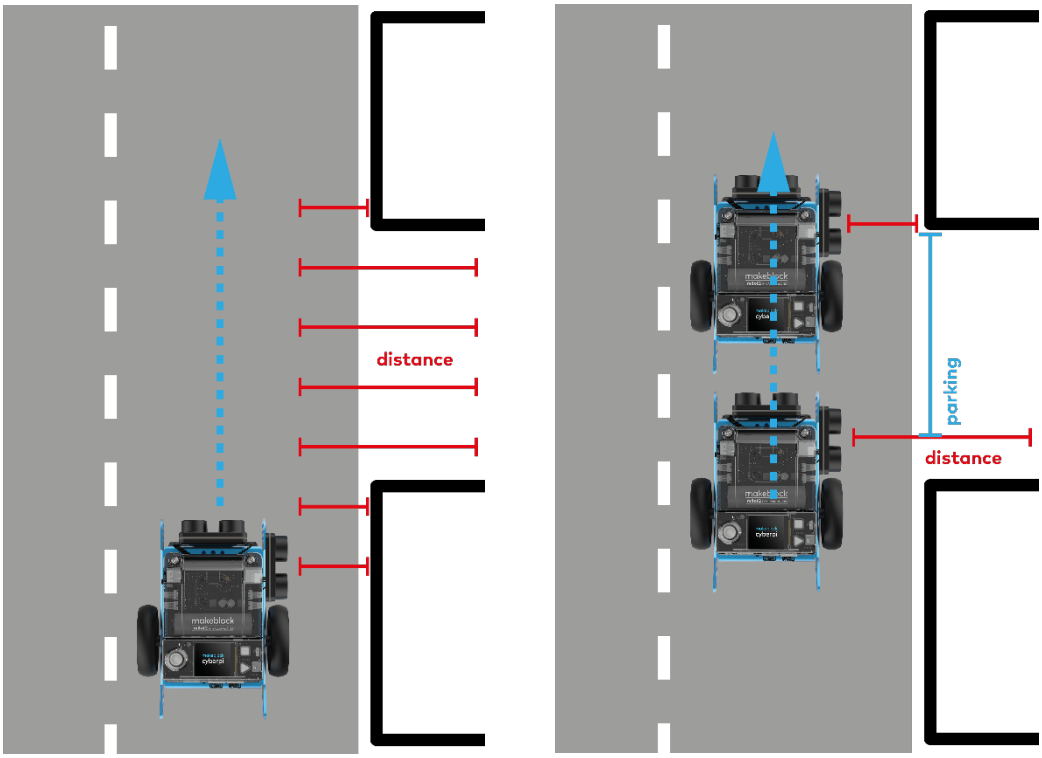
Now that the beginning of a gap has been detected, it is necessary to clarify how to measure its length. Students should come up with their own ideas for this; there are several possible solutions. One possibility is to measure the distance to the end of the gap using the wheel revolutions. To do this, the counter must be reset:



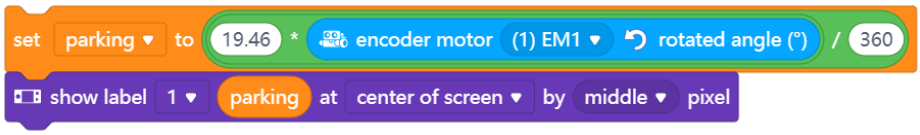
In the program flow, you can insert a short pause at this point so that it is visually clear from which position the gap is measured. This facilitates the planning of the actual parking maneuver...
Then drive to the end of the gap - this time waiting for a rapid decrease in the lateral distance:



As soon as the end of the parking space is detected, the mBot2 stops. This position is also important to plan the parking maneuver - because the sensor at the right has detected the end, the mBot is still in front of the gap.
Accordingly, the robot drives (straight) along the objects and continuously measures the distance. The position of the mBot2 at the beginning and end of the parking space must also be taken into account:



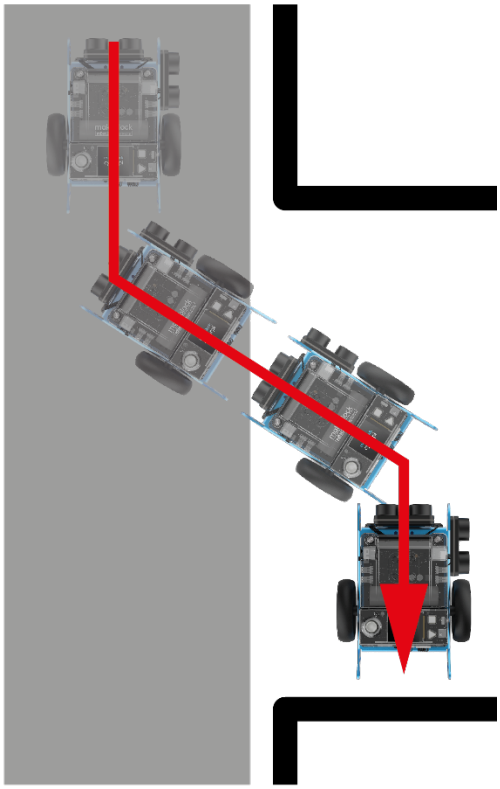
The distance between the detected beginning and end of the gap can be calculated using the circumference of the wheels. Students can also find different ways to determine the circumference here (e.g., measuring with a string, using circular formula from diameter, angle of rotation with known distance traveled, etc.). Here we work with the value 19.46cm:

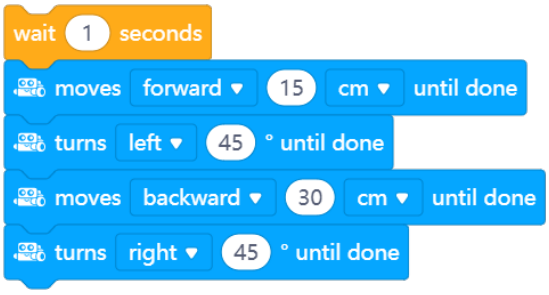


This value corresponds to a full revolution; however, the revolution sensor is specified in degrees. Therefore its value must be divided by 360. The determined distance is stored and displayed in the variable "parking" for possible later use.

At this point, for a fully automatic parking process, the adjustment is made whether the found gap is sufficiently large at all. This addition is to be done by the students independently as soon as the simple version works, which assumes that the gap is sufficiently large.

For the parking process itself, how the mBot2 has to drive to get into the parking space still needs to be clarified. This depends on the distance to the parked vehicles and the length of the gap. For the first steps, this gap should be sufficiently large and always the same. A parking procedure could look as follows (see below).





The exact distances and angles have to be adapted to the situation on site. Differentiation question: can this also be solved algorithmically, since the original distance to the parked vehicles is known?

3. Testing the individual steps

Students test out the examples presented and their own improvements and adapt them to the parking situation they are recreating. There are several ways to implement the parking sequence requirements: code fragments of a sample solution were shown here.

Once the code changes to it, and the effects are understood, the next step should be to test and improve the overall sequence.

3. Trying out (25 min)

Step 3: Trying out

In this step, the tested code fragments are integrated into an overall sequence. The mBot2 starts next to an obstacle ("parked vehicles") and searches for a parking space in a straight line, measures it out and starts the parking process.

If these subtasks work well, the requirements can be changed and increased accordingly:

How small should the gap be so that the robot can still park?

But what if... the gap is too small? So far, it is assumed that the gap is sufficiently large. How would the program

structure have to be changed so that the search for a suitable gap starts again?

It is important for the subsequent reflection phase that the interaction of the individual program steps is documented: students should take notes, discuss and debate their changes, and film the test runs if possible.

4. Wrap-up (5 min)

Step 4: Wrap-up

Was the parking lot search successful? How did the student groups' self-developed assistance systems work? Since the mBot2 steers via the two wheels, turns on the spot are possible and thus, of course, parking in narrower gaps - which team mastered this challenge best? What solutions did the students develop and what were their considerations in doing so?
Do the solutions work even if the gaps are too small? What about parking out?

In this exercise, a second ultrasonic sensor was used to control the robot's travel. The "parking assistance" system started a measurement at a sensor-controlled time and stopped at a sensor-controlled time. Students collected measurements using multiple sensors, analyzed them, as well as used them for subsequent control. The exercise involved developing their own algorithms, use of variables, and use of control structures. The code examples are merely an incentive to develop further and better solutions.
For a brief reflection, the following points should be discussed:

- *What worked particularly well? What could have been implemented better?*
- *Which parts of this exercise were easy to understand and which were difficult?*
- *Are there topics for which further explanations are desired?*