1 Robo-Camp 2019: Lesson Plan and Teaching Materials

In this chapter, the lesson plan used as well as the teaching material from the autumn camp "Robo-Camp" which was held in connection with the PANaMA project's teaching offer in the form of two science camps (Crypto-Camp 2019, Robo-Camp 2019) which were each held twice during of autumn 2019, October 7 - 11, and 14 - 18 will be reviewed (see Chapter IV. Autumn-Camps in research workshop Kiel, for a more detailed description of the autumn camps).

The lesson plan is built up in several steps and phases, of which the teacher is free to choose either to follow it all, select the parts that best fit the current level of the class, or to form their own lesson plans based on the material made available in connection with this. . The lesson plan and its review should therefore not be seen as a final decision on how such a course should be held, but as an inspiration for which topics, as well as information that may exist. may be relevant to involve, as well as convey. Please note that because the autumn camps were not divided into classic teaching modules, the times for the individual steps and phases in the lessons are therefore not sharply divided into modules of 45min.

The overall theme of the lesson plan is robots and automation, while the focus is on creating an understanding among students of what a robot is and is not, as well as giving them a basic knowledge of robot technology, including: Microcontrollers, sensors and actuators. The goal is to enable the students afterwards, to use the technology themselves for the development of their own robots, as well as for the solution of specific problems.

The developed material, parts of which will be presented in the present lesson plan, can be found in its entirety at the following link (note that the material is currently in English):
http://www.teknologiskolen.dk/robocamp2019/

Note: The lessons are based on the microcontroller technology Arduino Uno, as well as the sensors and actuators that were included in the course. This means that the specified units are also based on this, and a digital signal will therefore, for example, be described as being between 0V (LOW) and 5V (HIGH), despite the fact that this is platform-specific. Eg. operates both micro: bit and Calliope Mini, with a voltage between 0V (LOW) and 3.3V (HIGH) when connected via USB, whereas if they run on a 3V battery supply, the following applies, 0V (LOW) and 3V (HIGH).

1.1 Introduction to robotics – What is a robot? (70 min.)

Briefly about the content: The lesson is predominantly theoretical, but with great emphasis on dialogue with the students. The focal point is to establish a common understanding of what a robot is / is not, as well as what core components they consist of, their mutual function and relationship to each other.

Learning outcomes: Students become familiar with what a possible definition of a robot can be, they are enabled to identify and analyze which objects can be classified as being a robot, and why. In addition, they gain insight into what internal core components a robot contains, what role they play in the robot, as well as the interaction between them. In continuation of this, potential misunderstandings regarding. robots, due to modern media portrayal of these, will also be clarified, as well as possibly. demystified.

1.1.1 Phase 1 - The introductory interview (30 min):

Explain to the students that before you can work with and make our computer robots, you must first know what a robot is. Then start the conversation about what a robot is, by asking the class until what they understand by the word robot, as well as what constitutes one.

The class is now presented with a series of pictures (eg of humanoid robots, non-humanoid robots, electric machines, as well as non-electric tools such as a bicycle), to which the students for each picture are asked whether the object in the picture is a robot or not, as well as the justification on which the answer is based.

After this, the official definition or rather lack of the same can be brought into play, as there is no definitive definition of what a robot is, although there is a general consensus. a number of parameters. A possible definition of this may therefore be the following:

"*A programmable machine, which can function autonomously, as well as sense its surroundings and adapt its actions accordingly*".

Feel free to talk to the students about how to break down the above definition for use in an analysis of what a robot should be able to do to meet this definition, ie. a robot must be able to:

- Sense the world
- Trade in the world
- Make a decision (how to act, in relation to the sensed)

With the three requirements in place, the conversation can then naturally lead to a talk about how robots can meet the above requirements, where students are asked the following questions:

It can sense the world:

-How do we sense the world? (sight, hear, taste, smell, sense of touch).
-How can a robot sense the world? (it needs at least one sensor, eg a light sensor).

It can trade in the world:

-How do we trade in the world? (muscles).
-How can a robot act? (it needs at least one actuator, eg a DC motor)

It can make a decision:

-How do we make a decision? (the brain).
-How can a robot make a decision? (it needs at least one microcontroller - eg an Arduino).

Hereby, the three core components that in our definition, constitute a robot and as such must possess at least one of each, are identified: Sensors, actuators and microcontrollers.

Possible definition of a robot: "A programmable machine, which can function autonomously, as well as sense its surroundings and adapt its actions accordingly".

Core components: Microcontrollers, sensors and actuators.

1.1.2 Phase 2 - The core components, their function and interaction (30 min):

After identifying the three core components, these can then be reviewed one at a time to explain what they do and how they play together. As the microcontroller controls the sensors as well as the actuators, this is a good place to start.

1.1.2.1 What is a microcontroller?

A microcontroller is basically, a small computer. It has a processor (computing power), RAM (memory) and other necessary components, just like a computer has. Microcontrollers are found today, in almost all types of electrical equipment: in the soda machine, the television, the microwave, the alarm clock, etc.

The microcontroller has a number of inputs and outputs, its so-called I / O pins (Input / Output). The I / O pins of the microcontroller can be programmed to either measure the voltage on them, or to supply other components with it.

Often the microcontroller will be used to read what voltage a sensor supplies a given I / O pin with, after which it will make a selection based on this, and then supply an actuator with a voltage that triggers an intended action, in accordance with this choice. .

The signal the microcontroller receives from a sensor is called INPUT, while the signal the microcontroller sends to an actuator is called OUTPUT. INPUT signals can be read either as digital (LOW (0V) or HIGH (5V)), or analog (from 0 (0V) to 1023 (5V)), while OUTPUT signals can be written as digital (LOW (0V) or HIGH (5V))) or PWM (Pulse-width modulation) (from 0 (0V) to 255 (5V)) which is a simulation of an analog signal.

Microcontroller: A small computer that contains programmable I / O pins (Input / Output), eg an Arduino Uno. The role of the microcontroller is to execute the program that the programmer has transferred to it and it is therefore, so to speak, the "brain" of the robot.

The microcontroller will often be programmed to receive an analog / digital INPUT signal (0V - 5V), process the signal and make a decision on how to act on it, after which a digital / PWM OUTPUT signal (0V - 5V) supplies a connected actuators, performing this action.

Digital signals (input and output): Digital signals are either 0V or 5V.

Analog signals (input): Analog signals have a value that is somewhere between 0V and 5V and is read by the Arduino as a number between 0 (0V) and 1023 (5V).

PWM signals (output): PWM signals (Pulse-width Modulation), is a simulation of analog signals and is created by an I / O pin, in a custom range, oscillating between alternately being 0V and 5V. The oscillations take place so fast that the signal can in practice be used as an analog signal. In the Arduino, a PWM signal is written as a number between 0 (0V) and 255 (5V).

1.1.2.2 What is a sensor?

It is a good idea to start by asking the students what examples of sensors they already know, after which a sample list is presented, eg containing: Contacts, light sensors (LDR - Light Dependent Resistor), distance sensors (ultrasonic sensor), color sensors, motion sensors , microphones, temperature sensors, infrared sensors (IR-Sensor), etc.

The dialogue can now be turned over to whether one can find something that is common to them all? In particular that they register changes in the environment they are in.

In the same vein, it can be discussed whether robots can have sensors that can detect things humans cannot. Can people, for example, hear ultrasound, see infrared light, etc.?

This can be followed by a brief overview of how sensors are generally used: Some sensors constantly send a signal back to the microcontroller (eg LDR), while others must first be activated before they send a signal back (eg the distance sensor). Common, however, is that the microcontroller receives this signal as an INPUT, which can then be read and interpreted.

At the same time, the vast majority of sensors students will be working with will return an analog signal of 0V - 5V, which is translated by the microcontroller's 10-bit ADC (analog-to-digital converter) to a number of 0-1023 (LDR, Temperature, Potentiometer, IR sensor etc.), whereas eg the distance sensor measures the time that elapses between an INPUT signal going from HIGH (5V) to LOW (0V).

Finish by showing pictures of the different sensors the students will work with in the teaching, so that they already become visually familiar with what these look like.

Sensor: Sensors detect changes in the environment they are in, this can be, for example, the interaction with a switch, changes in the light level (LDR), the sound level (microphone), the sensor's distance to the nearest object (ultrasonic sensor), etc.

Signal: The microcontroller receives the value of the sensor as an INPUT signal, which is read either analogue or digitally.

1.1.2.3 What is an actuator?

As with sensors, students can be asked what examples they already know, after which a sample list can be displayed, eg containing: motors (DC, servo, stepper), LEDs (Light Emitting Diodes), speakers, heating elements, etc.

Then the common feature of these can be identified, in particular that they affect the environment they are in, while it can be concluded that eg infrared LEDs and ultrasonic speakers, can perform actions that humans are not capable of.

In contrast to sensors which send a signal to the microcontroller, which it receives as an INPUT, in this connection it is the microcontroller which here sends an OUTPUT, which the actuator receives. In connection with the actuators the students are expected to work with in the course, the signal used here is OUTPUT, either digital (LOW (0V) or HIGH (5V)), or PWM (Pulse-width Modulation) (from 0 (0V) to 255 (5V)).

Actuator: actuators affect the environment they are in, this can be through physically moving parts such as motors (DC, servo, stepper), but also through changes in the light level (LED), the sound level (buzzer, speaker), the temperature (heating elements) etc.

Signal: The microcontroller sends command to the actuator, through an OUTPUT signal, which is either digital or PWM.

1.1.3 Summary:

Summarize the following and then review the same series of pictures from earlier, in plenary with the students. Discuss once again whether the objects in the pictures are robots or not, do they meet our requirements for a robot? Can the three core components be identified (note that the microcontroller is often hidden, so you just have to assume that it is there)? The summary can advantageously be repeated several times throughout the course, in order to train the students in identifying and analyzing robots, as well as refreshing the interaction between microcontrollers, sensors and actuators. Therefore, feel free to use new photo series for each time, as well as objects from the students' everyday lives.

Definition:
"A programmable machine, which can operate autonomously, as well as adapt its behavior to the environment".

Ie. a robot can sense the outside world, make a decision, and act accordingly (possibly act accordingly):

Core components:
A microcontroller (making decisions)
A sensor (detects changes in the environment)
An actuator (performs changes to the environment)

Overview:

	Microcontrollers:	**Sensors:**	**Actuators:**
Purpose	Make decisions. Listens to sensors and commands actuators.	Detects changes in the environment they are in.	Affects the environment they are in.
Signal:	Receives an INPUT from sensors. Sends an OUTPUT to actuators.	Sends an INPUT to the microcontroller, which is read either analog or digital.	Receives an OUTPUT from the microcontroller, which is either digital or PWM.

1.1.4 Phase 3 - What is a robot not (10 min):

The current definition of what requirements an object must meet before we can call it a robot has now been defined, while there has also been a brief review of how the core components work and play together in relation to each other. But it is often at least as important to know what a robot cannot do as what it can do. This is eg very relevant when people work with robots, eg in industry, because the robot does not think of you and therefore does not take into account it is not programmed to take.

Ask the students in plenary if they have any ideas about what a robot is not / can not, even though they, like us humans, can sense the world, act in the world, and make a decision?

Some good topics to get into are, for example, that robots are not capable of having emotional emotions. This dialogue can advantageously be started by the students first being introduced to two robots, a "cute" robot a la. NAO, PARO etc., as well as an industrial robot, eg a welding machine. Then the students can be asked which of the two robots is able to have the greatest empathic feelings. None of the robots are of course capable of this, but the students will throughout their lives, be met with robots designed to indicate

that they possess empathic feelings, also they will through modern media, often be presented to robots as being capable of this.

At the same time, it is important to state to students that robots are not self-conscious and are not intelligent in the classical sense - again, no matter what movies and other media portray them as being. Explain to the students that they operate exclusively on the basis of a logical circuit, ie. all their decisions are based solely on mathematical calculations, which in turn they are lightning fast to perform. But for the same reason, should one therefore devote sufficient time to it, one will, by going through all the calculations with paper and pencil, always arrive at the same decision as the robot.

For the same reason, robots are neither reflective, intuitive nor creative. They do exactly what they are programmed for, and nothing else. Robots can therefore, for example, not have a stupid day, be slaves or not bother to do what we say to them. As an example, one can take a robot that is programmed to run straight out. If this robot is now set up on a table, what happens next? It runs directly over the edge without at any point having had an opinion on it, or in any way reflecting on the consequences of this, neither before nor after. So it does not stop intuitively, in the same way that we humans would do. This can sometimes be difficult for students to understand and it is therefore important to make it completely clear that robots only do what they are programmed to do, no more and no less.

Robots are not: Intelligent in the classical sense, intuitive, reflective and self-aware or able to have emotions.

Robots make decisions based on mathematics and logic: If you therefore set aside enough time, you will always come up with the same decision as the robot with paper and pencil.

1.2 Introduction to robots - Why is it so difficult to make robots? (30 min.)

Briefly about the content: The focal point of the lesson is a physical activity where students in groups of three must interact together and do it next to a robot, by simulating its three core components: the microcontroller, the sensors and the actuators.

Learning outcomes: Through physical activity, students become familiar with the basic internal communication channels that make up the interaction between the microcontroller, the sensors and the actuators (INPUT and OUTPUT). At the same time, they learn about the limitations of robots, which is related to our intuitive way of acting in unfamiliar environments.

Physical materials per. group (3 students per group): 2 pcs. fabric that can be used as a blindfold, and 3 pcs. similar colored plastic mugs or similar. objects.

1.2.1 Phase 1 - Introduction, Rules, Setup, Challenges and Objectives (12 min):

Introduce students to why it is so difficult to make robots, even though so far they can be compared to humans (brain: microcontroller, senses: sensors, muscles: actuators)? One of the basic explanations is, of course, related to the concluding talk in the last lesson about what robots are not. Just think of how often we do, for example, an intuitive action, just by walking around the forest floor.

Ask if necessary. students which robot they think is the easiest to make: A robot that can perform 1000s of things in a controlled environment, or a robot that can perform 1 thing in an uncontrolled environment? Then explain that the easiest thing is the controlled environment, because all variables involved are known

in advance, which is why the robot's lack of intelligence and ability to reflect and think intuitively is no longer problematic to the same degree.

At the same time, our communication between the brain, the senses and the muscles, refined through millions of years of natural development, which is why it is now so embedded in our system that we no longer have to actively think about how the input from our senses should be interpreted or how the output to our muscles need to be formulated.

After the introduction to the problem, the students are introduced to the activity's rules, setup as well as challenges and goals, see below

Problem: As living beings, our brains, senses and muscles have, through millions of years of natural evolution, been fine-tuned to work together, this does not apply to robots. At the same time, robots are not intelligent in the classical sense or able to reflect and perform intuitive actions. Therefore, it is also incredibly difficult to make robots because their shortcomings mean that they cannot automatically adapt to unknown variables.

Rules:

- Three students together make up one robot (microcontroller, sensor and actuator):
- Microcontroller (blindfolded):
  Can make decisions.
  Can ask the sensor for information and listen to the answer.
  Can give the actuator commands.
- The sensors:
  Can sense the world and answer the microcontroller when asked.
- The actuator (blindfolded):
  Can listen to commands from the microcontroller, as well as execute them.

Setup:

- Designate a group table for each group and place them in front of it.
- After the microcontroller and the actuator have been blindfolded, the three plastic mugs are distributed in the immediate vicinity of the "robot", preferably at different height levels (under a chair, on the next table and the like).

Challenges and goals:

- Locate the three plastic mugs.
- Collect the three plastic mugs.
- Stack the three plastic mugs on the designated table.

1.2.2 Phase 2 - The activity (18 min):

Divide the students into groups of three students per. group, after which the activity can begin. After this, the students must now, working with the robot they make up, try to work together to solve the challenge and reach the goal of stacking the plastic mugs on their respective tables. When the goal is reached, it is rotated and the students in the group are assigned a new role in the robot, this is repeated three times until all three students have tried all three roles (microcontroller, sensor, actuator).

1.3 Introduction to Arduino - Get to know the Arduino system (6 hours and 30 minutes):

Briefly about the content: The lesson mixes theory with practical and concrete hands-on exercises, re. the understanding and application of a number of selected sensors and actuators, as well as basic programming principles. The programming principles, sensors and actuators reviewed herein are all part of the final theme project of the lesson plan.

Learning outcomes: Students gain a basic understanding of electricity, microcontroller technology (Arduino Uno), programming principles, as well as sensors and actuators. In addition, they are enabled to work with and construct simple input / output systems themselves.

Physical materials per. group (2 students per group): Lesson includes a larger selection of different materials and components, the full list can be found on the website linked to at the top of the section. The same can the developed cheat sheets, parts of which are included in this section, as well as examples of programs for the various sensors and actuators, as well as solution proposals for the various tasks. The developed cheat sheets can advantageously be printed out and distributed already in the first phase of this lesson.

Note: After each phase, a few examples of assignments are given that allow students to experiment with the setups reviewed. It is therefore recommended that each phase be started by handing out the materials used in it to the students, so that they can copy these on an ongoing basis.

1.3.1 0. Phase - Introduction to the further course (10 min):

Introduce the students to the final theme project, in order to put the upcoming lessons in a concrete context, with the aim of removing part of the abstract and non-concrete element thereof. The following phases can also be presented with advantage - where possible - in relation to the theme project. This is so that the students, for example, become acquainted with the transport robot using IR sensors, as well as why and where they are located, before they start learning about their use in relation to the microcontroller.

1.3.2 Phase 1 - Introduction to electricity and actuators (LED) (20 min):

Introduce or refresh basic knowledge of electricity and how current flows in electrical circuits, briefly explain what current, voltage and resistance are. The detailed calculations can be omitted.

Draw a diagram of a power source (5V) that supplies an LED (Red - 5mm), incl. presupposition 150Ω and review this for students.

Then introduce the breadboard and how its holes are internally connected, please show illustrations of this. Construct the circuit on the breadboard, while the Arduino can act as a power supply for this, using its 5V and GND pins.

When using an actuator (LED), one of the three requirements for a robot is hereby met.

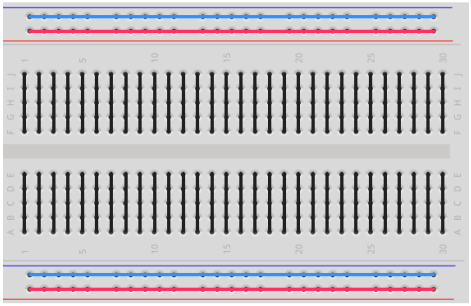


Fig. 1: Overview of the breadboard internal connections.

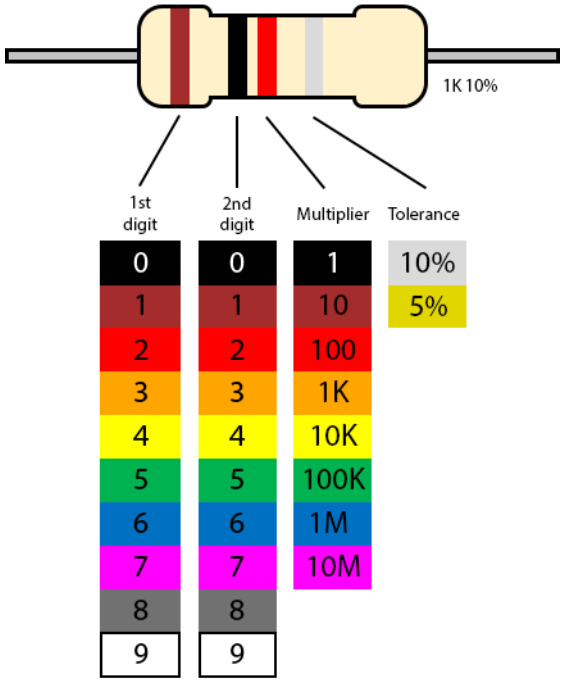


Fig. 2: Overview of how to interpret the color-codes found on a resistor.

1.3.3 Phase 2 - Introduction to the Arduino (45 min):

Introduce the Arduino and review its functionality in general, what are pins, what is their function etc. as well as introducing them to the Arduino programming environment.

Now use the Arduino to control the LED from the past, rather than simply acting as a power supply. This can be done with advantage on a project, so that the students can follow on their own computers and program the program at the same time as the teacher. Start by introducing what variables are and then explain how the program structure works, then set the connected pin to OUTPUT in setup, while in loop it is set to HIGH. Review again the difference between INPUT and OUTPUT, why the LED is set to OUTPUT, as well as the difference between LOW (0V) and HIGH (5V). It can also be a good idea, in the same vein, to briefly get into the most used data types (int, String, boolean), in connection with the connecting pin being saved as an int.

Possible analogy on variables: A good analogy can be to compare them with the moving boxes many have standing outside in the garage, on which a label has been written and something has come in it. Now the desired box can always be found again and it can be accessed, just as it can be replaced with something else. Explain that variables are therefore a method of storing things in the microcontroller's memory so that they can be found again later.

When all students have had the LED light up, it can be set to flash by letting the connected pin switch between HIGH and LOW, but without inserting a delay. Students will quickly discover that they can see no difference between then and now. This is a great approach to let them experience how fast a processor can perform calculations and actions based on it. Introduce the delay function, but first ask the students where it should be inserted. In many cases the answer will be that it is between the pin set to HIGH and

LOW. If this is the case, it leads on to another good exercise for the students, because as before, they will again not see any noticeable difference as an effect of this. A student can advantageously be asked to flash the light in the classroom at intervals of approx. 2 seconds, and explain what it does, before each action: On, Wait, Off, Wait, On, etc. Then ask students to review the code in the program: On, Wait, Off, On, etc. Here they will most likely quickly detect that when using a loop, there is no waiting time between the transition from the last line to the first line, so a delay must also be inserted after the pin is set to LOW.

Assignments for the students:

- Experiment with flashing speed. Where is the transition in milliseconds, from the eye being able to detect that it is flashing and to the fact that it looks like it is switched on constantly.
- Experiment with two LEDs flashing either simultaneously or alternately.

Show how by adjusting the size ratios of the inserted delays, it is possible to simulate how the LED, for example, looks as if it is switched on at half power. This leads to a conversation with the students about the difference between digital, analog and PWM (Pulse-width Modulation) signals.

Review the differences between digital and analog signals and at the same time explain to students that the Arduino can only read analog signals, but not generate them. Instead, the Arduino uses a simulated analog signal, called PWM. Explain to them that a PWM signal is basically the same as the one they constructed just before, through the manipulation of the aspect ratio of the delays used (see Figure 3).

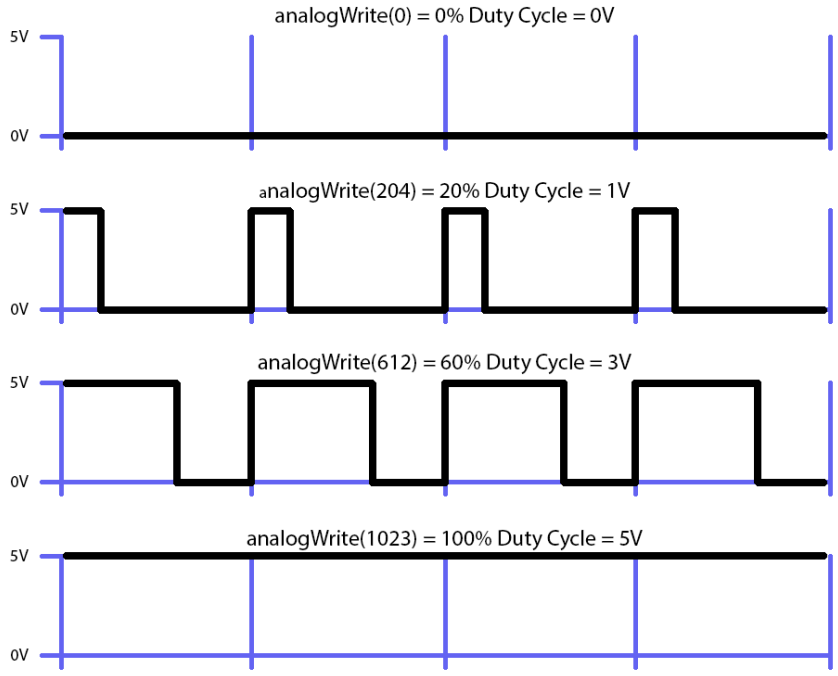


Fig. 3: Illustration of how a PWM-signal, simulates an analog-signal.

Arduino Pins:

```
analogRead:    A0-A5
digitalRead:   A0-A5, 2-13

analogWrite:   3, 5-6, 9-11
```

Datatypes:

```
int:     -32768 to 32767
         LOW or HIGH
         A0-A5 (analog pins)
         2-13 (digital pins)
```

```
digitalWrite: A0-A5, 2-13

Used by the R3 Motorshied:
A0-A1, 3, 8-9, 11-13
```

```
bool:    true or false
         0 or 1
         LOW or HIGH

String: "text text"

void:    no data
```

Program Structure:

```
//variables can be initialized
//here

//runs one time only
void setup(){
  //setup pins and enable serial
  //here
}

//runs repeatedly
void loop() {
  //write code for the program
  //here
}
```

Fig. 4: Overview of the Arduinos pins, and the most common datatypes and the Arduino programstructure.

Variable (with a value):

```
//datatype name = value
int varName = 13;

//datatype name = value
bool varName = true;

//datatype name = value
String varName = "Hello!";
```

Delay:

```
void loop() {
  //pause for 1000 milliseconds
  delay(1000);
}
```

Fig. 5: Overview of how variables and delays, can be used.

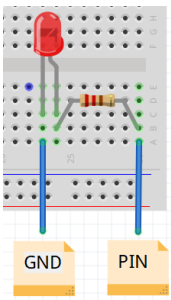
LED: 5mm Red LED (resistor size: 150Ω)

```
int LEDPin = 2;                 //the used pin

void setup() {
  pinMode(LEDPin, OUTPUT);      //set pin as OUTPUT
}

void loop() {
  //turn the LED off (LOW) or on (HIGH) *
  digitalWrite(LEDPin, HIGH);

  //turn it gradually from off (0) to on (255) **
```



GND: Short leg
PIN: Long leg

```
  analogWrite(LEDPin, 180);
}
```

*PIN: 2, 4-7, 10, A2-A5
**PIN: 5, 6, 10

Fig. 6: An examples of how and LED can be used.

1.3.4 Phase 3 - Introduction to sensors (contact) (25 min):

Introduce the contact and review for the students how it works and can be seen as a broken wire that, when pressed together, creates a connection through it.

Make a circuit with it on the breadboard and connect it to the Arduino, at the same time let the students copy this. Briefly review why it is necessary to use a resistor in connection with this, so as not to burn off the circuit. Explain that because contacts are sensors, they must be specified as INPUT in the setup part of the Arduino code. The use of INPUT, as well as a resistor in connection with. the switch, provides a natural introduction to voltage dividers and the construction of own sensors (more on that in the following sections). For later courses, it is worth noting that the resistor used is not strictly necessary, because instead the Arduino's built-in pull-up resistors can be used, which are activated by replacing "INPUT" with "INPUT_PULLUP".

Create a new program in which the value of the switch is read. Initially omit all serial communication (Serial.begin (9600); and Serial.println (switchValue)) so that there is only analogRead (switchPin). Then talk to the students about how to gain insight into what the microcontroller "thinks", because even though an analogRead command has been inserted, they can currently. do not see the value anywhere. It is not inconceivable that one or more of them, possibly. from previous teaching or independent projects from home, are familiar with print statements.

Then introduce the serial monitor, as well as how serial communication is activated and print statements are used, to get the values of variables printed, as well as own text messages for the monitor. Explain how print statements are an important tool in connection with. debugging, because this is the most immediate method of gaining insight into which parts of the program are being executed, as well as what values are being operated with.

When using a sensor (switch), two of the requirements for a robot are hereby met. Therefore, introduce and review how conditions can be used to let the microcontroller turn the LED on / off, depending on whether the switch is depressed or not, whereby the third and final requirement has been met.

Depending on the type of switch, but it well stand and swing a few times between being up and down when this stage changes. It can therefore often be a good idea to insert a very short delay, of a few milliseconds.

Assignments for the students:

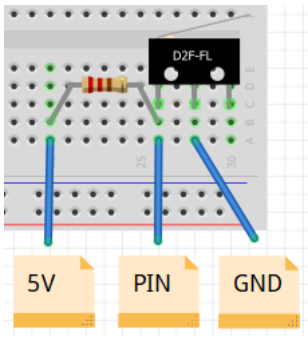
- Experiment with what size a delay should potentially be before the oscillations are filtered off, so that the switch is not registered as being pressed down more times than it actually is.

Switch: Resistor size: 1KΩ

```
int switchPin = A5;          //the used pin
int switchValue;             //to store the value
```

```ino
void setup() {
  pinMode(switchPin, INPUT);   //set pin as INPUT
  Serial.begin(9600);          //enable serial
}

void loop() {
  //read and store the value, then prints it
  switchValue = digitalRead(switchPin);
  Serial.println(switchValue);
}
```



*PIN: 2, 4-7, 10, A2-A5

Fig. 7: An examples of how a switch can be used.

Conditional (Structure):

```ino
void loop() {
  if (x == y) {
    //if yes, run code
  }
  else if (x < 4 && y < 4) {
    //if yes, run this instead
  }
  else {
    //else, run this instead
  }
}
```

*there can only be one “if”
*there can be multiple “else if”
*there can only be one “else”

Conditional (Operators):

```
==   equal
!=   not equal
>    larger
<    smaller
>=   larger or equal
<=   smaller or equal

&&   and
||   or
```

Fig. 8: Overview of how conditionals can be used.

1.3.5 Phase 4 - Explanation of voltage dividers and sensors (LDR) (65 min):

Voltage dividers form the basis of many sensors and it is also an easy way to construct our own. Therefore, introduce voltage dividers and how the relationships between the resistors play together.

Possible analogy on variables: A good analogy for the purpose is that the voltage can be seen as a dish with five cakes (one cake for each of the 5 volts). The two resistors can therefore be compared to two people. Add the requirement that all the cakes must be eaten after the dish has passed the last person. It is important to emphasize that it does not matter how hungry the people are, because all five cakes must be eaten no matter what. Explain that what matters, on the other hand, is the relationship between how hungry they are. If they are equally hungry in relation to each other, the cakes are divided equally between them. While if the first person is very hungry compared to the second, he will therefore eat most cakes while the second does not eat quite many and vice versa.

After the dish of cakes has passed the first person, you can now see how many cakes are left on the dish, this is our indicator of the relationship between the two people's hunger, because you know that the second person no matter what, must eat the remaining .

After introducing and explaining how voltage dividers work, the following exercises can then be reviewed and made (the contact is simply taken out of the circuit and then replaced by a resistor - then the setup is the same as from the previous one). Repeat that analog values of the Arduino are read as a number between 0 (0V) and 1023 (5V).

Reistor 1:	Resistor 2:	Read value (Arduino):	Value in volt:
1KΩ	1KΩ		
560Ω	560Ω		
820Ω	180Ω		
180Ω	820Ω		
1KΩ	Unlimited (switch is up)		
1KΩ	None (switch is down)		

With the exercises over, the formula for voltage dividers can be reviewed, after which further setups can be calculated first and then tested in practice..

$$U1 = U * \frac{R1}{R1 + R2}$$

Explain to students that although many sensors are based on voltage dividers, the above that have just been constructed are not sensors. This is because they are static in nature and thus do not register changes in the environment they are in. If instead of static resistors, dynamics had been used that change their internal resistance in step with changes in the environment they are in, this would have met the requirements for a sensor. Therefore, introduce the LDR (Light Dependent Resistor) and explain that its internal resistance is dynamically variable depending on how much light falls on it.

Insert the LDR into the voltage divider and then read and print the value of the voltage divider to the monitor. Because the voltage drop across the individual resistor in the voltage divider is no longer constant, the value read now represents the changes in the environment where the sensor is located (light level).

Assignments for the students:

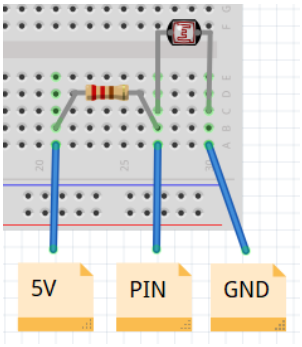
- Design a system similar to a street lamp to turn on an LED when it is dark and turn it off again when it is light.
- Design a system inspired by smarthomes, where through the use of conditions, the brightness of the LED is gradually turned up / down, as the light changes, so that a constant light level is achieved inside the living room.

LDR (Light Dependent Resistor / Photoresistor): Resistor size: 1KΩ

```
int LDRPin = A5;                //the used pin
int LDRValue;                   //to store the
value

void setup() {
  pinMode(LDRPin, INPUT);       //set pin as INPUT
  Serial.begin(9600);           //enable serial
}
```

```
void loop() {
  //read and store the value, then prints it
  LDRValue = analogRead(LDRPin);
  Serial.println(LDRValue);
}
```



*PIN: A2-A5

Fig 9: An example of how an LDR can be used.

1.3.6 1.3.6 Phase 5 - Expansion with additional sensors (Potentiometer, IR-Sensor) (60 min):

Introduce the students to the potentiometer and explain that, like the light sensor (LDR), it is also based on a voltage divider, so the components can be freely switched around without the need to make changes to the code. Explain that the potentiometer has a sliding transition in the ratio between the values of the two resistors, based on where the slider interacts with the circular resistance.

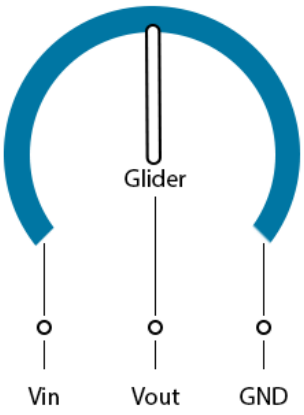


Fig. 10: Illustration of the innerworkings of a potentiometer.

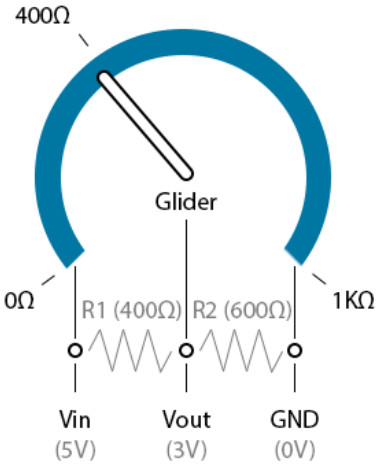


Fig. 11: Illustration of a 1K potentiometer in use (5V).

Introduce the IR sensor and review how it is constructed of an IR LED, as well as an IR Photodiode. The IR sensor works by the IR LED emitting an infrared light (this is invisible to the human eye, but can often be seen with a camera in a mobile phone) which is then reflected back to the IR photodiode to varying degrees depending on the surface. It is the amount of reflected light that is read by the sensor. Students will therefore also find that if they keep it away from the assault, then it reads "black" (or rather, it detects a lack of reflection).

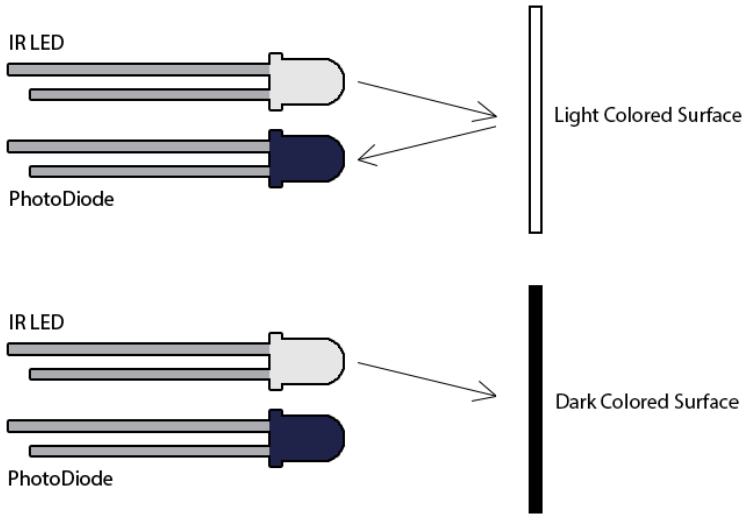


Fig. 12: Illustration of the principle behind how an IR-sensor works.

Assignments for students (potentiometer):

- Design a dimming system that gradually turns on the LED with a five-step transition when the potentiometer is turned from left to right.
- Design a dimming system that gradually turns on the LED with a smooth transition when the potentiometer is rotated from left to right.

Assignments for students (IR sensor):

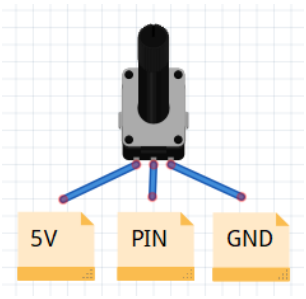
- Experiment with the distance at which the IR sensor is best for capturing the difference between white and black paper.
- Reuse the code from the potentiometer and construct a system where an LED turns on if the IR sensor reads a low value (solid glare from white paper), while it goes out if a high value is read (minimal glare from black paper).
- Reuse the code from the potentiometer and construct a system where an LED turns on more and more, in five intervals, based on the amount of glare on the IR sensor.

Potentiometer:

```
int PotPin = A5;                    //the used pin
int PotValue;                       //to store the
value

void setup() {
  pinMode(PotPin, INPUT);           //set pin as INPUT
  Serial.begin(9600);               //enable serial
}

void loop() {
  //read and store the value, then prints it
  PotValue = analogRead(PotPin);
  Serial.println(PotValue);
}
```



*PIN: A2-A5

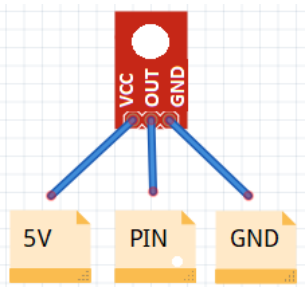
Fig 13: An example of how a potentiometer can be used.

IR-Sensor: QRE1113 (Analog)

```
int IRPin = A5;                 //the used pin
int IRValue;                    //to store the
value

void setup() {
  pinMode(IRPin, INPUT);        //set pin as INPUT
  Serial.begin(9600);           //enable serial
}

void loop() {
  //read and store the value, then prints it
  IRValue = analogRead(IRPin);
  Serial.println(IRValue);
}
```



*PIN: A2-A5

Fig 14: An example of how an IR-sensor can be used.

1.3.7 Phase 6 - Actuators (Servo and DC Motor) (60 min):

Introduce the students to the servomotor and thereby also to libraries and explain to them that this is a collection of pre-programmed code, which in this case should be used to check the servo-motor. Have students set the servomotor to a given position and then have them try setting it to move from side to side. Here it is important to get into the application of delays, in addition to which the servo motor does not have time to go to the specified position until it receives the signal of the next position and vice versa.

Assignments for the students:

- Design a system that can be used in an exoskeleton, with the aim of strengthening the human elbow joint, so that storage workers do not wear the body through heavy lifting, by letting the position of the servomotor follow the potentiometer's.
- Design a system for automatic opening of windows, by making the servo motor switch between standing in a position of 0 and 90o when a switch is pressed.

Introduce students to the motor shield, as well as the DC motor. In the same embrace, the engine shield's built-in H-bridges (it has two, one for each engine) can be explained quite briefly. Figure 15 shows a simplified version of the circuit for an H-bridge, the important thing to notice is the four transistors P1, P2 and N1, N2, as well as the two connections to it F1 and F2. They are set up so that when voltage is set to F1, P1 is opened while N1 is closed and vice versa when there is no voltage at F1. Similarly, P2 and N2 depend on the voltage at F2. Therefore, by setting 5V to F1 and 0V to F2, the current will flow one way through the motor, while it will run the other way if 0V was set to F1 and 5V to F2. If, on the other hand, you set 5V or 0V to both F1 and F2, the current will not be able to run on either of the roads. Explain that the reason the motor shield and its H-bridges are used to control the DC motors is that they can conduct a stronger current through the motors than the Arduino itself can supply them with. The current the motor shield conducts through the DC motors comes directly from "Vin" (voltage in - ie the connected battery pack (9V)).

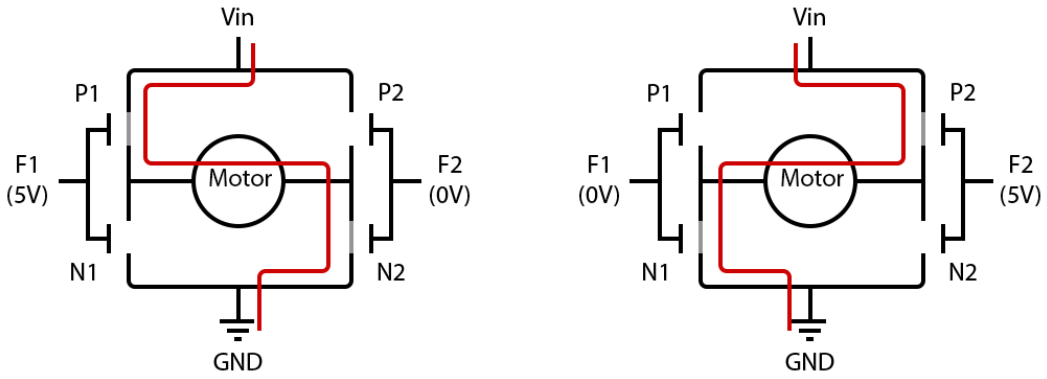


Fig. 15: Simplified illusttation of the circuit forming an H-Bridge.

Assignments for the students:

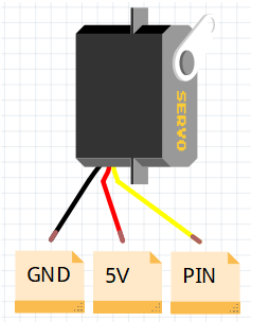
- Experiment with getting the DC motor to run alternately clockwise and counterclockwise at intervals of 5 seconds, as well as different speeds.
- Design a system designed for ventilation, in which the DC motor gradually changes its speed as a potentiometer is rotated from left to right and vice versa.

Servo-Motor:

```
#include <Servo.h>          //include library
Servo servoMotor;           //create new servo
object
int servoPin = 2;           //the used pin

void setup() {
  //attach the used pin to the servo object
  servoMotor.attach(servoPin);
}

void loop() {
  //turn the servo to a position (value: 0-179)
  servoMotor.write(90);
}
```



*PIN: 2, 4-7, 10, A2-A5

Fig. 16: An example of how a servomotor can be set to a specific position.

DC-Motor:

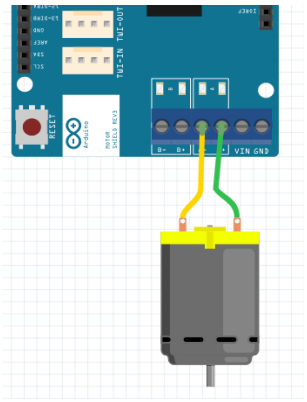
```
int motorASpeed = 3;                    //PWM A
int motorADirection = 12;               //DIR A

void setup() {
  //set all motor pins to PUTPUT
  pinMode(motorASpeed, OUTPUT);
  pinMode(motorADirection, OUTPUT);
}

void loop() {
```

```
  //DIRECTION: Turn clockwise = HIGH,
  //turn counter-clockwise = LOW
  digitalWrite(motorADirection, HIGH);

  //SPEED: no speed = 0, full speed = 255
  analogWrite(motorASpeed, 150);
}
```



*for the pins used for motor B, see the motor-shield

Fig. 17: An example of how a DC-motor can be controlled.

1.3.8 Phase 7 - Other types of sensors (Distance sensor, Color sensor) (60 min):

Introduce the distance sensor to the students and review how it can be used to measure the distance from the sensor to the nearest object. Review how it is constructed over a speaker and a microphone, where the speaker emits a ping (the ping cannot be heard by humans because the frequency is too high), which when it hits a surface, is reflected back to the microphone. The time it takes between the ping being sent out and returning can then be used to calculate the distance between the sensor and the affected object. For this purpose, the method pulseIn () is used, which registers the time it takes from a pin either going from HIGH to LOW, or LOW to HIGH, the time is measured in microseconds (0.000001s). The following formula can therefore be used to calculate the distance in centimeters.

$$Pulse\ In\ (measured\ in\ microseconds, 0.000001s)$$

$$The\ speed\ of\ sound = \frac{343m}{s} = \frac{34300cm}{s} = \frac{0.034cm}{ms}$$

$$Distance\ to\ the\ object\ in\ cm: Pulse\ In\ (ms) * \frac{0.034cm}{1ms} / 2\ (forth\ and\ back)$$

Assignments for the students:

- Experiment with measuring different distances inside the classroom and printing these to the screen. Find the smallest and longest distance at which the sensor works.
- Design an alarm system for an industrial robot that warns employees that they are too close to the machines, by letting an LED light up when the distance sensor detects that there is a person within 2 meters.

Introduce the color sensor and briefly explain how it works. It is designed so that it, through a filter, measures the reflected value of Red, Green and Blue (RGB) light, one color at a time. Based on the RGB values, one can create all colors, but it is important to know that all three together give white, while the absence of all three results in black. Ideally, the values will behave in Figure 18, in reality, however, this is not the case. When the sensor is used, it first detects a color and writes down which RGB values the sensor

returns, after which these values can be used to define a given color (in the developed library, only the colors red, green, blue and yellow can be defined - see figure 20). In addition, due to the fact that the RGB values represent the amount of reflected light, one must be aware of changing light conditions. If, for example, the color yellow has been defined in high sunshine, based on a set of read RGB values, these will not necessarily be the same when it has become evening. To get around this, it is therefore often a good idea to shield what you read, in order to keep the lighting conditions static to a greater extent..

Assignments for the students:

- Experiment with measuring and defining the colors Red, Green, Blue, Yellow, White and Black. Which colors' RGB values differ most from each other and are there some that are closer than others?
- Experiment with the difference in value of the colors, depending on the lighting conditions.
- Design a system that uses conditions to print the detected colors to the screen.

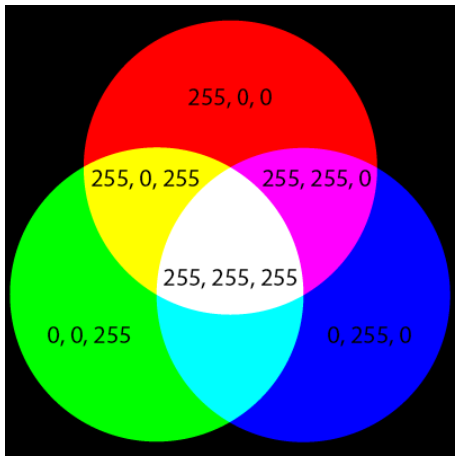


Fig. 18: Overview of the ideal RGB-values.

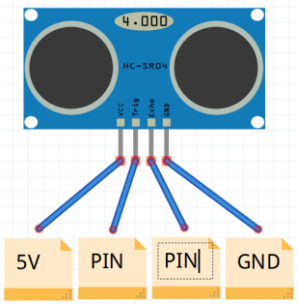
Ultra-Sound Sensor: HC-RS04

```
int trigPin = 2;              //the used trig pin
int echoPin = 4;              //the used echo pin
int distance;                 //to store the value

void setup() {
  pinMode(trigPin, OUTPUT); //sets pin as OUTPUT
  pinMode(echoPin, INPUT);  //sets pin as INPUT
  Serial.begin(9600);       //enables serial
}

void loop() {
  //store the returned value from the function
  distance = getDistance();

  //prints the stored value
  Serial.println(distance);
}
```



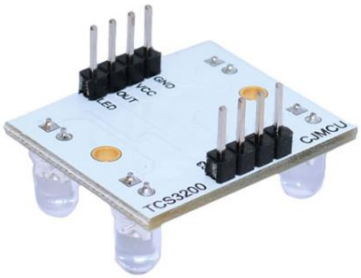
*PIN: 2, 4-7, 10, A2-A5

```
//function - returns the distance
int getDistance() {
  //sends out a trigger sound
  digitalWrite(trigPin, LOW);
  delayMicroseconds(10);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  //returns the received echo in centimeter
  return pulseIn(echoPin, HIGH)* 0.034/2;
}
```

Fig. 19: An example of how the ultrasound distance-sensor can be used.

Color-Sensor: TCS3200




```
//includes the library
#include <Color.h>

//the used pins
int S0 = 2;
int S1 = 4;
int S2 = 5;
int S3 = 6;
int OUT = 7;

//creates a new color-sensor object
Color color(S0, S1, S2, S3, OUT);

//used to store the color-value
String colorValue = "";

void setup() {
  setColors();
  Serial.begin(9600);  //enables serial
}

void loop() {
  //prints the R(ed), G(reen), B(lue) and W(hite) values from the
  //sensor.
  //write down these values for each color you want to identify
  //(red, green, blue, yellow) and insert them into the
  //setColors function.
  Serial.println(color.getRGBValues());

  //gets the color and prints it
  colorValue = color.getColor();
  Serial.println(colorValue);
}
```

```
//Defines the Colors
void setColors() {
  //color.setRed(R, G, B, W);
  color.setRed(40, 104, 22, 81);
  color.setBlue(46, 36, 14, 56);
  color.setGreen(89, 56, 16, 33);
  color.setYellow(29, 36, 12, 55);
}
```

READ ME NOTE: In order to enable the inclusion of the Color library, copy paste the entire "Color Library" folder, into the "Documents -> Arduino -> Libraries" folder.

Fig. 20: An example of how the color-sensor can be used.

1.3.9 Phase 8 - Introduction to additional programming principles (Functions and While-Loops) (45 min):

In connection with the use of the distance sensor and the color sensor, a function was used for both. Therefore, introduce students to features and explain how they work by encapsulating a piece of code that can then be called over and over again. This makes it both easier to use (changes to the code only need to be made in one place, rather than many places), as well as easier to read (the function can encapsulate many lines of code, under a simple and meaningful name).

Assignments for the students:

- Create a function called "printHello" that when called printer "Hello" for the screen.
- Create a function called "sayHelloTo" which takes a String as a parameter, which when called printer "Hello", as well as the parameter.
- Create a function called doubleThisNumber that takes an int as a parameter, which when called multiplies the value from the parameter by two, after which the new value is returned. The returned value must be stored in a variable and printed.

Introduce while loops and explain how they repeat a piece of code over and over as long as the associated condition is true.

Assignments for the students:

- Create a function containing a while loop that runs 10 times in which the counter is printed.
- Create a function with a while loop that prints "Happy Birthday to You" at intervals of 1 second, if a switch is pressed, the loop must be interrupted.

Function (return no value):

```
void loop() {
  //runs the code inside-
  //myFunction
  myFunction();
}

//datatype name() {}
void myFunction() {
```

Function (return a value):

```
void loop() {
  //stores the returned value-
  //from myFunction
  int returnedValue;
  returnedValue = myFunction();
}

//datatype name() {}
```

```
  //some code to run
}
```

```
int myFunction() {
  //some code to run

  //return a value
  int value = 100;
  return value;
}
```

Fig 21: Oversigt hvordan funktioner kan anvendes.

While loop:

```
void loop() {
  int x = 0;

  //while (argument is correct)
  while(x < 10) {
    x = x + 1;
  }
}
```

Break:

```
void loop() {
  //while (argument - is correct)
  while(true) {

    if(something) {
      //break out of the loop
      break;
    }

  }
}
```

Fig. 22: Overview of how while-loops can be used.

1.4 Theme project – Automation (12 x 45 min.)

Briefly about content: The project deals with the automation of a coffee bean center, for which two types of robots are to be developed. A robot that sorts the beans according to quality (color sorting), as well as a robot that transports them around the warehouse (line tracking). The students choose which of the two robots they want to work with, after which it is intended that they work relatively independently with the project in small groups (it is recommended that the students are two students per group).

Learning outcomes: Students learn to apply what they have learned from previous lessons in a practical context, to solve concrete problems inspired by the industrial workplaces of reality. At the same time, this gives them a deeper insight into and understanding of the development of complex automated systems, as well as their limitations.

Materials: In the developed material, LEGO has been used as a platform for the developed robots. In order to facilitate the integration of electronics in the LEGO platform, a number of 3D models that are compatible with it have therefore been developed at the same time. Manuals for this, as well as 3D models are, as previously described material, made available on the website. The same applies to a complete overview of the components used in the project, solution proposals, as well as proposals for further development of the robots etc. It is important to note that the robots can be constructed in many different ways and from many different materials, the developed LEGO manuals are therefore only an offer for a solution to this.

1.4.1 Introduction to the Projekt:

Introduce the students to the following project, which is based on a coffee bean center, where the beans are first sorted by quality, after which they are distributed around the associated warehouse. At the center, the beans have previously been hand-sorted, after which they have been distributed by hand in the warehouse. This involves hard physical work, while at the same time taking a long time. The center's desire is therefore for these two processes to be automated, which is why two types of robots must be built and programmed, each of which can handle one of these functions.

Coffee bean sorting (color sorting): The beans are sorted according to quality, which is determined by their color, in connection with the project, the beans consist of 3D printed bricks (red, green, blue and yellow). The robot must therefore be able to receive a bean, register which color category it belongs to, rotate four associated boxes around so that the corresponding box faces the conveyor belt, and then deliver the bean therein and rotate the boxes back to the starting point again.

Transport (line tracking): The sorted beans are packed in boxes, which in connection with the project consist of 3D printed boxes (red, green, blue and yellow), the color indicates which of four corresponding areas in the warehouse they are to be delivered to. The warehouse has been equipped with black lines in the floor, which the robots must navigate by. In addition, to avoid accidents, the robots must stop if there is an object in front of them blocking the road, this could be human employees or other robots.

The control panel is already equipped with an area where the sorting takes place and in addition, the employee has just completed the work of drawing the lines the robots must navigate according to, see Figure 23.

Once the students have chosen which of the two robots they want to work with, they can follow the corresponding procedure from the following two sections.

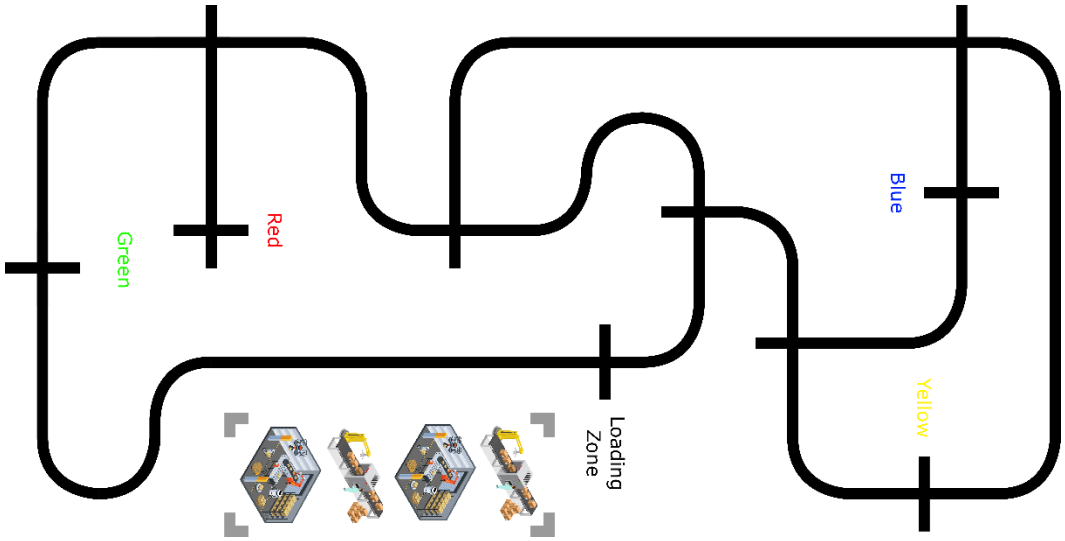


Fig. 23: Overview of the warehouse sorting/loading zone and the transportation routes.

In the videos below, you can see examples from the real world, of robots designed for automatic sorting of coffee beans, as well as transportation.

Coffebean sorting: https://www.youtube.com/watch?v=i0nmIl-bNLI

Transportation: https://www.youtube.com/watch?v=WIlS3vNSuQ4

1.4.2 Coffee bean sorting:

The robot consists of two main parts, which are combined into one. The first part is the conveyor belt, on which the coffee beans are first placed, then they are transported under a color sensor after which their color is registered (red, green, blue or yellow), then the task of the conveyor belt is to transport them the last piece to and down in a corresponding box. The second part is the mechanism that rotates the corresponding boxes, so that when the beans reach the end of the conveyor belt, they fall into the correct box. See Figure 24 below.

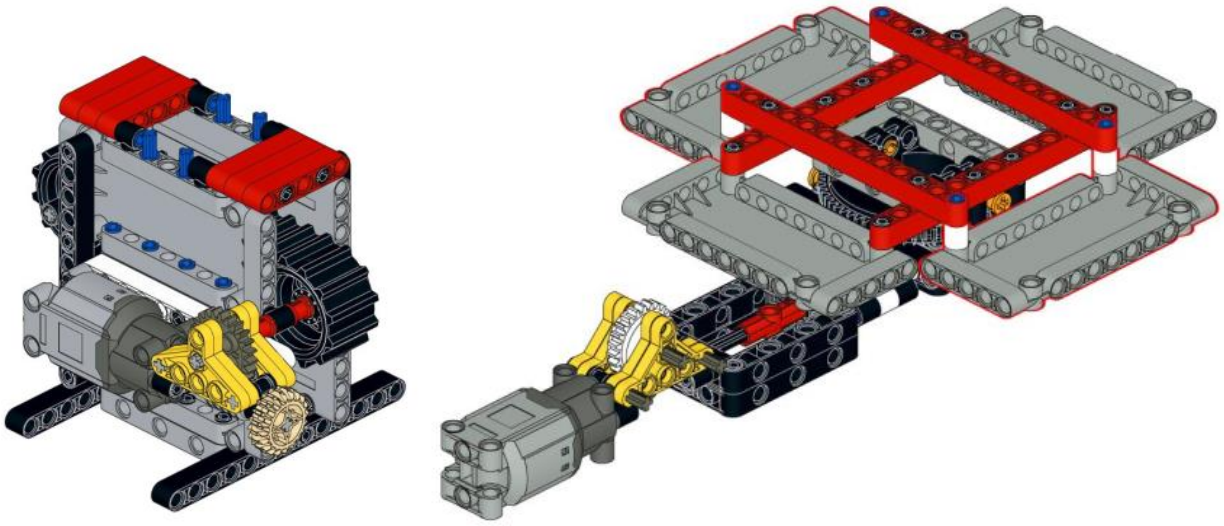


Fig. 24: The LEGO-models for the belt and the box rotator.

1.4.2.1 The initial steps:

1. Construct the robot:

The first thing to do is to design the robot itself so that there is something to work from.

Task:

- Follow the LEGO manual "Bean Sorter - Belt" and build the conveyor belt.
- Follow the LEGO manual "Bean Sorter - Rotator" and build the rotation mechanism for the boxes.
- Follow the LEGO manual "Bean Sorter - Combine" and build the two parts together.
- Now assign the boxes each their individual color, possibly. with a felt-tip pen and choose which, by default, should face the conveyor belt.

Materials:

- 2 pcs. LEGO Power Functions Motor (Large).

2. Integrate the Arduino platforms:
Now the Arduino must be added, which is the microcontroller that will be responsible for controlling our robot, as well as the associated motorshield, which is an extension to the Arduino that is used to control the motors, as well as a breadboard.

Task:

- Follow the manual "Bean Sorter - Integrate the Arduino platform" and add the Arduino, motor shield, breadboard and battery pack.

Materials:

- Arduino Uno.
- Motor shield. R3
- Breadboard (half size).
- Battery pack + Cord.
- Arduino module (3D printed).
- Breadboard module (3D printed).
- Battery pack module (3D printed).
- 2 pcs. he / he wires.

3. Test the engines:
It is now time to connect the motors and at the same time carry out a number of smaller tests to make sure that they work as intended.

Task:

- Connect the motors to the motor shield.
- Make the conveyor belt travel forward for 2 seconds, at full speed.
- Make the conveyor belt run backwards for 2 seconds, at half speed.
- Make the boxes rotate clockwise for 2 seconds, at full speed.
- Make the boxes rotate counterclockwise for 2 seconds, at half speed.

Cheat Sheets:

- Effectors - DC Motor.
- Code - pinMode.
- Code - digitalWrite.
- Code - analogWrite.
- Code - delay.

1.4.2.2 Transportation belt:

It is recommended to create a new program file for this part, but please save it earlier from the Initial Steps.

1. Use functions to control the conveyor belt:
Features are a great way to encapsulate a piece of code so that it can be used many times without having to type the code every single time. At the same time, it makes the code easier for people to read.

Task:

- Create a "startBelt" function that starts the conveyor belt.
- Create a “stopBelt” function that stops the conveyor belt.

- Create a "deliverBean" function that delivers the prayer (starts the conveyor belt and lets it run for 5 seconds, after which it is stopped again).
- Test that all three functions work.

Cheat Sheets:

- Code - Functions.

2. Integrate the color sensor:

Before the conveyor belt can be used, the color sensor must first be integrated in it, and it must be tested whether it works as intended.

Task:

- Follow the manual "Bean Sorter - Integrate Color-Sensor" and integrate the color sensor.
- Place a “bean” on the conveyor belt under the color sensor and print to the screen what color it is. Do this for all the colors (red, green, blue and yellow).

Materials:

- Color sensor (TCS3200).
- Color sensor module (3D printed).
- 7 pcs. he / she wires.

Cheat Sheets:

- Sensors - Color-Sensor (TCS3200).
- Code - Variables.
- Code - Serial Print.

3. Reacts when reading colors:

The conveyor belt can now be checked, as well as use the color sensor to detect colors, but so far it does the same all the time. It is therefore time to create a structure for the program that uses conditions to allow the program to perform different actions, depending on what color the color sensor detects (red, green, blue or yellow).

Task:

- Start the conveyor belt.
- If the color sensor reads "red":
- "red" to the screen.
- Stop the conveyor belt and deliver the "prayer".
- Test that this works when it does, then make an “if else” for each of the other colors (green, blue and yellow) that contains the same functionality as that for the red color.

Cheat Sheets:

- Code - Conditionals.

1.4.2.3 The boxes:

It is recommended to create a new program file for this part, but save it earlier from the Conveyor Belt.

1. Use functions to rotate the boxes:

Use, as with the conveyor belt, once again functions to encapsulate the engine control code so that it can be reused again and again.

Task:

- Create a “startRotation” function that starts the boxes rotating counterclockwise.
- Create a "stopRotation" function that stops the rotation of the boxes.
- Create a function "rotatePosition" that takes an int as a parameter (call the parameter "position"), except to print the parameter to the screen, this function is left blank.
- Test that the functions work.

Cheat Sheets:

- Code - Functions (with parameters).

2. Integrate the switch:

Until now, the boxes could only rotate back and forth, based on time, but because the time it takes to rotate a lap varies depending on how much power is left on the batteries, how much dirt has entered the gearing etc. is this not a good solution. Therefore, integrate the switch in the construction so that it can be used to register each time the boxes have rotated one space.

Task:

- Follow the manual "Bean Sorter - Integrate Switch" and integrate the switch.
- Test that the switch works by printing its mode to the screen.

Materials:

- Switch.
- Switch module (3D printed).
- 1KΩ resistor.
- 3 pieces. he / he wires.

Cheat Sheets:

- Sensors - Switch.

3. Control the number of rotations:

Because the switch is automatically pressed each time a box passes it, it can use this to control how many positions the boxes have moved, by counting the number of times the switch is pressed. This way you can always make sure that the boxes move exactly the number of positions desired. For this purpose, the function "rotatePosition" must be completed so that it can be called with the number of rotations desired.

Task:

- Inside the function "rotatePosition":
- Start the rotation of the boxes.
- A new counter variable called "i" and give it the value 0.
- Do a while loop with the condition (in <position).
- If the button is pressed:
- Increment "in" by 1.
- Wait approx. 200ms (depending on the rotation speed).
- Stop rotationen af kasserne.

- Test at funktionen virker, ved at kalde den med forskellige parametre (1, 2, 3 osv).
- Tæl nu hvor mange gange hver af kasserne skal rotere en position, førend de står foran transportbåndet, samt hvor mange gange de skal rotere førend de er tilbage ved deres udgangspunkt igen. Skriv disse tal ned på et stykke papir.

Combine the transportation belt and the box rotator:

It is recommended to create a new program file for this part, but please save it earlier from the Initial Steps.

1. Combine it all:

It is now time to combine the conveyor belt with the boxes so that the robot is fully automated. Fortunately, almost all the work required for this has already been done, through our use of features.

Assignment:

- Start the transportation belt.
- If the color sensor reads "red":
  - Print "red" to the screen.
  - Stop the conveyor belt.Roter kasserne sådan at den røde kasse står ud for transportbåndet.
  - Deliver the bean.
  - Rotate the boxes so that the red box is next to the conveyor belt.
  - Rotate the boxes so that the red box is at its starting point again.
  - • Test that this works and then expand the program to include the other colors (green, blue and yellow).

1.4.2.4 Ideas to expand upon the sorter:

Feel free to come up with some ideas for potential expansions, for example, to make different LEDs light up to indicate what actions the robot is currently doing. is about to undertake. It can also be to build a holder that, with the help of a servo motor, can even place new beans on the conveyor belt, one at a time.

1.4.3 Transportation:

The robot rests on two front wheels, which act as its driving force, as well as a ball wheel. See Figure 25 below.

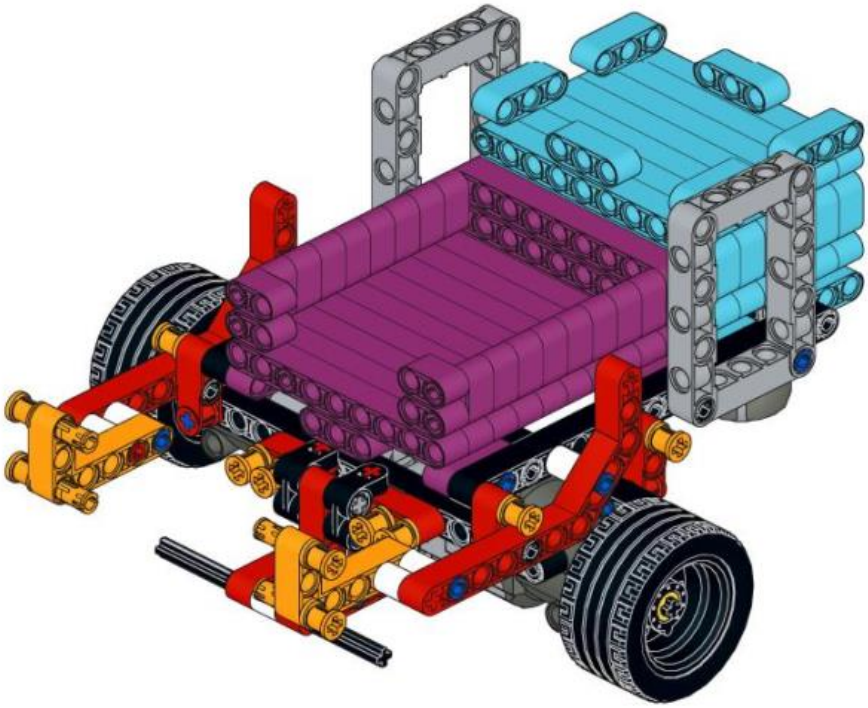


Fig. 25: The LEGO-model for the transporter.

1.4.3.1 The initial steps:

1. Design the robot:

The first thing to do is to construct the robot itself so that there is something to work from.

Task:

- Follow the LEGO manual "Transporter - Transporter" and build the transporter.

Materials:

- LEGO + 2pcs. LEGO Power Functions Motor (L).

2. Integrate the Arduino platforms:

Now the Arduino must be added, which is the microcontroller that will be responsible for controlling our robot, as well as the associated motorshield, which is an extension to the Arduino that is used to control the motors, as well as a breadboard.

Task:

- Follow the manual "Transporter - Integrate the Arduino platform" and add the Arduino, motor shield, breadboard and battery pack.

Materials:

- Arduino Uno.
- Motor shield. R3
- Breadboard (half size).

- Battery pack + Cord.
- Arduino module (3D printed).
- Breadboard module (3D printed).
- Battery pack module (3D printed).
- 2 pcs. he / he wires.

3. Test the engines:
It is now time to connect the motors and at the same time carry out a series of smaller tests to make sure that they work as intended.

Task:
- Connect the motors to the motor shield.
- Make the conveyor move forward for 2 seconds, at full speed.
- Have the conveyor reverse for 2 seconds, at half speed.
- Turn the conveyor to the left for 2 seconds, at full speed.
- Turn the conveyor to the right for 2 seconds, at half speed.

Cheat Sheets:
- Effectors – DC Motor.
- Code – pinMode.
- Code – digitalWrite.
- Code – analogWrite.
- Code – delay.

1.4.3.2 Navigate the lines:
It is recommended to create a new program file for this part, but please save it earlier from the Initial Steps.

Use functions to control the conveyor:
Features are a great way to encapsulate a piece of code so that it can be used many times without having to type the code every single time. At the same time, it makes the code easier for people to read.

Task:
- Create a “driveForward” function that drives the robot forward.
- Create a function "crossLine", which runs the robot approx. 5cm forward and stop it again.
- Create a "turnLeft" function that turns the robot a quarter turn to the left.
- Create a "turnRight" function that turns the robot a quarter turn to the right.
- Create a "uTurn" function, which rotates the robot half a turn, the direction does not matter.
- Create a “stopRobot” function that stops the robot for 1 minute.
- Test that all six functions work.

Cheat Sheets:
- Code – Functions.

2. Integrate the IR-sensors:

Before the carrier can follow the lines in the floor, its two IR sensors must first be integrated and it must be tested whether they work as intended.

Task:

- Follow the manual "Transporter - Integrate IR-Sensors" and integrate the IR sensors.
- First read the value of the left IR sensor and print it to the screen, then write down the values for white and black. Then repeat this for the right IR sensor.

Materials:

- IR sensor (QRE1113, Analog).
- IR sensor module (3D printed).
- 6 pieces. he / he wires.

Cheat Sheets:

- Sensors – IR-sensor (QRE1113, Analog).
- Code – Variables.
- Code – analogRead.
- Code – Serial Print.

3. Follow the line:

It is now time to use the IR sensors to control the robot so that it can follow a line. For this, for each IR sensor, the value that lies directly between the value for white and the value for black must be calculated. This value is called the threshold. To follow a line, the motors must run if their respective IR sensor looks white, otherwise they must stop. That way, the robot will automatically correct if it is about to skew and thereby cross in over the line, in the same way it will also automatically stop when it comes to an intersection because both sensors now look black. Feel free to start by simulating this, by controlling the robot by hand.

Task:

- Make the left motor run if the left IR sensor reads less than the threshold, otherwise it must stop.
- Make the right motor run if the right IR sensor reads less than the threshold, otherwise it must stop.
- Test that the robot can now follow the line, and that it stops when it reaches an intersection.

Cheat Sheets:

- Code – Conditionals.

4. Follow the line as a function:

So far, the robot stops automatically when it reaches an intersection, but it also starts again by itself if you manually push it past the intersection, or put it on a new line. Therefore, if you want to use the rotate commands stored in the functions "turnLeft" and "turnRight", to navigate in a cross, this creates a problem. Feel free to call these from the code - now the robot just stands and turns around and does not follow the line at all. This is because the lines of code that cause it to follow the line are executed in a split second, after which the turn command is called again. Therefore, an independent function must be created that causes the robot to follow the line until it reaches an intersection, after which the function ends.

Task:

- **Create a function "followLine".**
- **Use a while loop with the condition (true).**
- **Make the left motor run if the left IR sensor reads less than the threshold, otherwise it must stop.**
- **Make the right motor run if the right IR sensor reads less than the threshold, otherwise it must stop.**
- **If both IR sensors read less than the threshold, stop both motors and disconnect while looping.**
- **Test if the function works.**
- **Now test whether it works together with the turning functions, as well as the function for driving straight out at an intersection, for example by calling the functions below and then observing whether the robot is running the expected route. Remember that when a minute has passed, the function "stopRobot" is complete, the sequence also starts again.**
- **followLine.**
- **turnLeft.**
- **followLine.**
- **crossLine.**
- **followLine.**
- **turnRight.**
- **followLine.**
- **uTurn.**
- **stopRobot**.

Cheat Sheets:

- Code – While loop (with break).

1.4.3.3 Navigate the surroundings:

It is recommended to create save a copy of the program, then continue working on the copy.

1. Integrate the distance sensor:

The robot is now able to navigate along the lines on the warehouse floor, but what if there is another robot in front of it, or that an employee has overlooked that it is coming right towards them. To avoid the problems and damage that can occur, a distance sensor can be integrated, which allows the robot to measure the distance to the nearest object. This allows it to be programmed to stop afterwards, should an object be too close to it.

Task:

- Follow the manual "Transporter - Ultrasound Sensor" and integrate the distance sensor.
- Print the distance to the nearest object on the screen.

Materials:

- Distance sensor (HC-RS04).
- Distance sensor module (3D printed).
- 4 pieces. he / he wires.

Cheat Sheets:

- Sensors – Ultrasonic-Sensor (HC-RS04).

2. Stop for objects blocking the robot:
Upgrade function "followLine", so that the robot stops if the distance to an object in front is less than 20cm.

Task:
- **Upgrade the "followLine" function so that:**
- **The first thing that happens in the while loop is to read the distance to the nearest object.**
- **Upgrade the conditions for the engines to run so that the condition now reads:**
- **If the left IR sensor is below the threshold and the distance is more than 20cm, run the left motor, otherwise stop it.**
- **If the right IR sensor is below the threshold and the distance is more than 20cm, run the right motor, otherwise stop it**.

1.4.3.4 Deliver the box with the beans:
It is recommended to create save a copy of the program, then continue working on the copy.

1. Integrate the servo motor:
Employees can now put a box on the robot and program it to follow a route, but it cannot yet hand in the box itself when it has arrived. Therefore integrate a servo motor that can be used to tip the box off the barn, after which the robot can drive back for a new box.

Task:
- Follow the manual "Transporter - Servo-Motor" and integrate the servo-motor.
- Test the servo motor as intended by setting it in different positions.

Materials:
- Servo motor.
- 6 pieces. he / he wires.

Cheat Sheets:
- Effectors – Servo-Motor.

2. Deliver the box:
Used a function to let the robot deliver a box when it reaches the correct area.

Task:
- Create a "deliverBox" function that, with the help of the servo motor, tips the load so that the box falls off, after which the servo motor runs back to its starting point.
- Test that the function works, and that the robot can now run a route where it at the correct place, can now deliver the box it is transporting.

1.4.3.5 Ideas for expanding upon the transporter:
Feel free to come up with some ideas for potential expansions, for example, to make different LEDs light up to indicate what actions the robot is currently doing. is about to undertake. It may also be to use an LDR (Light Dependent Resistor) to check if the robot has a box on the bed.