# Experiential Learning of Robotics

This book is designed for beginners to introduce them to the field of robotics. The contents are based on the integration of electronics, computing, and mechanics.

dr. David Rihtaršič

# Kazalo

# 1 INTRODUCTION AND PREPARATION

> TO-DO:
>
> - nakaj o namenu te knjige.
> - kako je napisano in
> - kako ga uporabljati

Welcome to the educational robotics lecture using Arduino, Robduino module, and Fischertechnik parts! In this lectures, we will learn how to use these tools and materials to build and program simple robots for educational and recreational purposes.

First, we will introduce the Arduino controller and the Robduino module, and discuss their capabilities and limitations. We will also cover the basics of the Arduino programming language, including variables, functions, and control structures.

Next, we will discuss the Fischertechnik parts and how they can be used to construct robots with various shapes, sizes, and capabilities. We will cover the different types of parts that are available, such as beams, gears, motors, and sensors, and how they can be combined to create a wide range of structures and mechanisms.

We will then demonstrate how to use the Arduino controller and Robduino shield to program and control Fischertechnik robots. We will cover topics such as sensor input, actuator output, and feedback control.

Throughout this lecture, we will use hands-on activities and examples to illustrate the concepts and techniques that are covered. We will also discuss some of the challenges and considerations that are involved in building and programming robots with these tools and materials.

## 1.1 Introduction to embedded systems

## 1.2 Overview of robotics and its applications

## 1.3 Basic concepts and terminology

## 1.4 History of robotics

## 1.5 Teaching robotics

### 1.5.1 What is robotics

- Science of robots. :)
- What is a robot?
- How does the robot works?
- How are robots constructed?
- What is intended task of the robot?
- How do we control a robot?

### 1.5.2 What is a robot?

- automated (coffee) machine
- …
- Printer
- 3D printer
- CNC machine
- …
- “Robot” Vacuum cleaner (a.k.a. Roomba)
- Industrial robot arm (YASKAWA)
- Humanoid robot

It is not defined by the definition… but we have to describe it.

### 1.5.3 International Organization for Standardization - ISO

- Standards are not excluding each other…
- `ISO 2806` - defining the CNC machines
  - describing the processing technology
- `ISO 8373` - defining the robots
  - describing machine autonomy

#### 1.5.3.1 ISO 8373 - General Terms in Robotics

**1.5.3.1.1 ROBOTICS** science and practice of designing, manufacturing, and applying robots (2.6)

**1.5.3.1.2 ROBOT** actuated mechanism programmable in two or more axes (4.3) with a degree of autonomy (2.2), moving within its environment, to perform intended tasks

- Note 1 to entry: A robot includes the control system (2.7) and interface of the control system.
- Note 2 to entry: The classification of robot into industrial robot (2.9) or service robot (2.10) is done according to its intended application.

**1.5.3.1.3 REPROGRAMMABLE** designed so that the programmed motions or auxiliary functions can be changed without physical alteration (2.3)

**1.5.3.1.4 AUTONOMY** ability to perform intended tasks based on current state and sensing, without human intervention

**1.5.3.1.5 MANIPULATOR** machine in which the mechanism usually consists of a series of segments, jointed or sliding relative to one another, for the purpose of grasping and/or moving objects (pieces or tools) usually in several degrees of freedom (4.4)

- Note 1 to entry: A manipulator can be controlled by an operator (2.17), a programmable electronic controller, or any logic system (for example cam device, wired).
- Note 2 to entry: A manipulator does not include an end effector (3.11).

**1.5.3.1.6 CONTROL SYSTEM** set of logic control and power functions which allows monitoring and control of the mechanical structure of the robot (2.6) and communication with the environment (equipment and users)

**1.5.3.1.7 ROBOTIC DEVICE** actuated mechanism fulfilling the characteristics of an industrial robot (2.9) or a service robot (2.10), but lacking either the number of programmable axes (4.3) or the degree of autonomy (2.2) EXAMPLE:Power assist device; teleoperated device; two-axis industrial manipulator (2.1)

**1.5.3.1.8 INDUSTRIAL ROBOT** automatically controlled, reprogrammable (2.4), multipurpose (2.5)manipulator (2.1), programmable in three or more axes (4.3), which can be either fixed in place or mobile for use in industrial automation applications Note 1 to entry: The industrial robot includes: — the manipulator, including actuators (3.1); — the controller, including teach pendant (5.8) and any communication interface (hardware and software). Note 2 to entry: This includes any integrated additional axes.

**1.5.3.1.9 SERVICE ROBOT** robot (2.6) that performs useful tasks for humans or equipment excluding industrial automation applications Note 1 to entry: Industrial automation applications include, but are not limited to, manufacturing, inspection, packaging, and assembly. Note 2 to entry: While articulated robots (3.15.5) used in production lines are industrial robots (2.9), similar articulated robots used for serving food are service robots (2.10).

**1.5.3.1.10 MOBILE ROBOT** robot (2.6) able to travel under its own control Note 1 to entry: A mobile robot can be a mobile platform (3.18) with or without manipulators (2.1).

**1.5.3.1.11 ROBOT COOPERATION** information and action exchanges between multiple robots (2.6) to ensure that their motions work effectively together to accomplish the task

**1.5.3.1.12 INTELLIGENT ROBOT** robot (2.6) capable of performing tasks by sensing its environment and/or interacting with external sources and adapting its behaviour EXAMPLE:Industrial robot (2.9) with vision sensor to pick and place an object; mobile robot (2.13) with collision avoidance; legged robot (3.16.2) walking over uneven terrain.

## 1.5.4 Robotics and Education

### 1.5.4.1 Definition of the robots in education

Slangen:
Definition of the robot must be based on the main operation that robot performs:

- zaznavanje (angl. Sensing),
- sklepanje (angl. Reasoning) &
- delovanje (angl. Acting).

This operation is constantly executing in a.k.a. S-R-A loop.

Slo. nat. curriculum:`Robotics in Engineering`
- almost exact interpretation of S-R-A loop Krmiljenje s povratnim delovanje (angl. feedback control regulation)

- including learning objective: …kjer učenci ugotovijo potrebe po **krmiljenju s povratnim delovanjem** in izpostavijo pomanjkljivosti, če takega krmiljenja ni.

(angl. where students identify the need for **feedback control** and point out shortcomings in the absence of such control)

- misconception: Playing with robots or using a robot is robotics.
- Robots are meant to be user friendly.

#### 1.5.4.2 Robotics in Schools

- very popular in last decade

We can find robots in learning process as:

1. Robotics curses:

- Electronics
- Computer Science
- Engineering

2. motivation for learning other disciplines:

- Science
- Technology
- Engineering
- Math

#### 1.5.4.3 Important educational impacts

**1.5.4.3.1 LEARNING by DOING** … learning as “BUILDING KNOWLEDGE STRUCTURES” through progressive internalization of actions… this HAPPENS especially felicitously in a context where the

LEARNER is consciously engaged in CONSTRUCTING A PUBLIC ENTITY, whether it's a sand castle on the beach or a theory of the universe. (Papert, S. (1980). Mindstorms. Children, Computers and Powerful Ideas. New York: Basic books.)

**1.5.4.3.2 PRACTICAL APPLICATIONS** Applying knowledge and skills learned into a `public entity` make us proud of ourself. We have something to show to people that matters to us (friends, parents, classmates).

**1.5.4.3.3 CREATIVITY** There is not an only one solution to the problem. Kids can explore their ideas and put it to the test.

**1.5.4.3.4 LEARNING from MISTAKES** Kids are ALLOWED to LEARN from MISTAKES!?! In general, MISTAKES has very bad reputation in school sistem. To degree, that kids are often afraid to give an answer so as not to make a mistake (-> they stop trying). However, Robotics is so complicated field that mistakes can not be avoided. Thus, MISTAKES are very common thing in this learning proces of robotics.

**1.5.4.3.5 CRITICAL THINKING** Critical thinking is ability to do analysis of facts and form objective judgments based on reasonable arguments.

**1.5.4.3.6 SELF-ASSESSMENT** Kids are able to see if they fulfill the intended task or not. They can asses their own performance based on results of intended tasks.

## 1.6 Current state of the field

## 1.7 Robotics Equipment

Fischertechnik and LEGO are both brands of construction toy systems that allow users to build and create a wide range of structures and mechanisms. Both systems use a modular approach, with a variety of interlocking parts that can be easily snapped together.

However, there are some key differences between Fischertechnik and LEGO parts:

Material: Fischertechnik parts are made of a durable, high-quality plastic called polycarbonate, which is known for its strength and resistance to wear and tear. LEGO parts are made of a softer plastic called acrylonitrile butadiene styrene (ABS), which is more flexible and less durable.

Precision: Fischertechnik parts are designed with high precision and tolerances, which allows for more accurate and stable constructions. LEGO parts have slightly looser tolerances, which can make them more prone to wobbling or sagging.

Size and shape: Fischertechnik parts are generally smaller and more compact than LEGO parts, which allows for more detailed and precise constructions. LEGO parts are larger and more blocky, which makes them more suitable for building larger structures.

Functionality: Fischertechnik parts are designed with a focus on mechanical and electrical functionality, and include a wide range of components such as gears, motors, and sensors. LEGO parts are more geared towards aesthetics and playability, and include elements such as minifigures and decorative elements.

Price: Fischertechnik parts tend to be more expensive than LEGO parts, due to their higher quality and greater functionality.

Overall, Fischertechnik and LEGO are both excellent construction toy systems, and the choice between them will depend on the specific needs and preferences of the user.

We can divide the equipment for robotics into three different groups: 1. Electronics, 2. Computer science, 3. Engineering.

### 1.7.1 ELECTRONICS

- WIRES
  - 4x 15cm
  - 4x 10cm
- CONNECTORS
  - 8x 2.5mm FT
  - screw driver
- RESISTORS
  - 2x 330
  - 2x 3.3k
  - 2x 33k
  - 2x 330k
  - 10k potenciometer (with wires)
- NON-LINEAR RESISTORS AND SENSORS
  - 1x foto-tranzistor FT & aperature

  - 1x reed switch
  - 1x key FT
  - IR distance sensor

- ACTUATORS
  - light bulb
  - 2x DC motor FT
  - 1x servo-motor
  - 1x servo attach
  - LCD (i2c)

### 1.7.2 COMPUTER SCIENCE

- Arduino UNO controller
- modul RobDuino-v2 (shield)
- Arduino UNO adapter -> FisherTechnik (3D print)
- USB kabel
- battery charger for 2x18650 LiIon battery
- 2x 18650 LiIon battery's
- 9V Power Supply

### 1.7.3 MECHANICAL ENGINEERING

#### 1.7.3.1 CONSTRUCTION ELEMENTS

- 12x square block 15x15x30mm
- 6x square block 15x15x15mm
- 2x square block 7.5x15x30mm
- 5x square block 7.5x15x15mm
- 3x "L" profile 15x15x45mm
- 2x "L" profile 15x15x30mm
- 4x rim R1" fiksno
- 2x tire 11/90R1
- 4x square holder 15x15x15mm
- 2x angled block 60° 15x15mm
- 2x angled block 30° 15x15mm
- 1x pin rail 15mm
- 2x M4 nuts and bolts L=25mm

#### 1.7.3.2 GEARING (GEARS and GEARBOX)

- 2x gearboxes with shafts
- 2x sliding bearing
- 1x axle/shaft 45mm
- 1x axle/shaft 90mm
- 2x mechanical pivot joint
- 2x sliding bearing
- 2x spojka osi 15mm (BCA)
- 1x objemka 5mm (RD)
- 1x worm gear with attachment nut
- 1x gear fi48mm Z30
- 1x os elise 30mm

### 1.7.4 OPTIONAL

- rubber bands
- black isolating tape

# 2 ARDUINO

## 2.1 Overview of the Arduino platform and its capabilities

Arduino originated from the Wiring project, which was developed at the Interaction Design Institute Ivrea in Italy. The Wiring project was an open-source electronics prototyping platform that was designed to provide a low-cost and easy-to-use environment for creating interactive physical computing applications. The project was led by Hernando Barragán, a professor at the Institute, and the platform was based on the open-source, programmable Atmel microcontroller. Arduino was derived from the Wiring project and was released in 2005.

Arduino is an open source hardware and software platform used for building interactive electronics projects. The Arduino platform was designed to facilitate creating digital projects for the physical world. It consists of a physical programmable circuit board (often called a microcontroller) as well as a set of software tools for writing code for the board.

The Arduino platform is based on the Atmel AVR microcontroller, so it is capable of running programs written in C or C++. The board itself is made up of a number of components, including a voltage regulator, a USB connection, an LED, and a set of analog and digital pins that allow you to connect external components to the board. The board also includes a reset button and a power switch, allowing you to reset and power the board on and off.

The Arduino platform has a huge amount of flexibility and can be used to create a range of projects from simple to complex. For example, you can use the Arduino platform to create a basic home automation system that turns lights on and off, or you can use it to create a complex interactive art installation. You can also use the Arduino platform to create robots and other self-controlled devices.

The Arduino platform has grown to become an incredibly popular choice for makers, hobbyists, and professionals alike. It is incredibly easy to use, and the large community of users provides a wealth of tutorials and information. Additionally, the open-source nature of the platform makes it easy to customize and expand upon existing projects. It is a great platform for anyone looking to get started with physical computing projects.

## 2.2 Arduinouno and electronics simulation

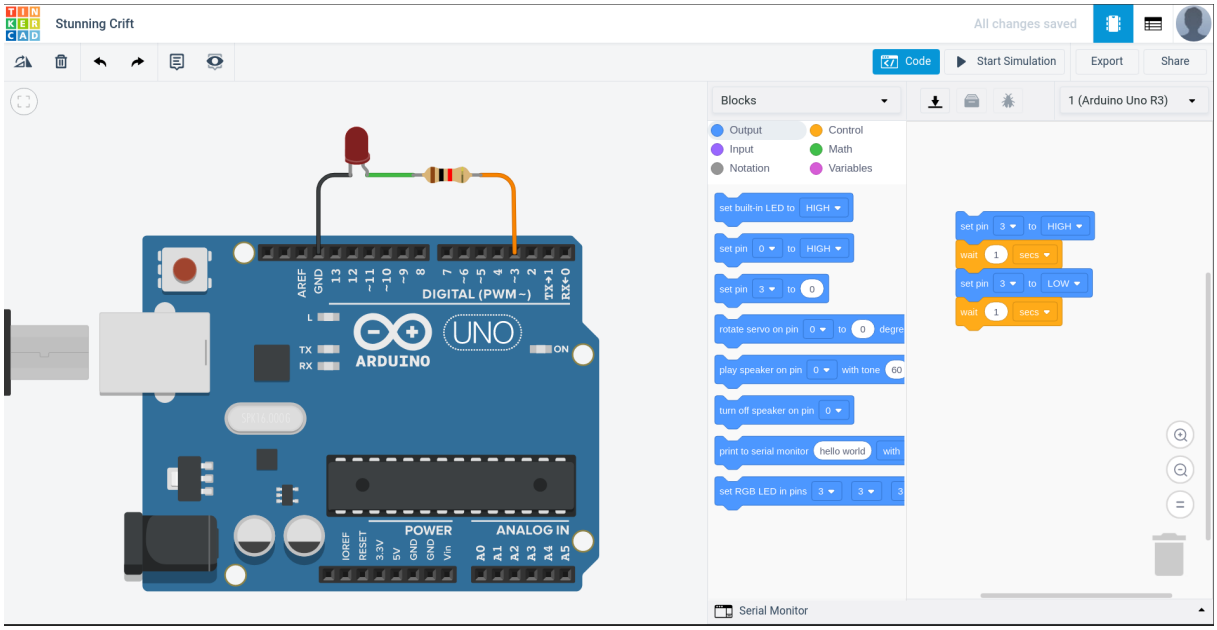
We can use several simulating programs to simulate robots. There are awesome platforms that allow simulations like: 3Dvisualizer or Webots … But since our robot will be based on the Arduino Uno controller probably the best option is:

- Thinkercad

You can sign in with your google account.

### 2.2.1 Task:

1. Try to do some basic project (e.g. Blink) to turn on and off an LED.
2. Try to add your own LED on the different output pin and change the program like is shown on the sl. 2.1 to make it work (LED must blinking).




**Slika 2.1:** Blink example in Thinkercad.

### 2.2.2 Questions:

1. How can you control the output voltage potential?
2. In which direction will electric current flow?

3. What voltage is present on the resistor?

- How can we determine the voltage on the resistor (describe 2 possibilities)?

4. What voltage is present on the LED?

## 2.3 Arduino IDE

The Arduino platform is based on the Atmel AVR microcontroller family, and the Arduino Uno is based on the ATmega328 microcontroller. The Arduino Integrated Development Environment (IDE) is a software application that provides a way to write and upload code to the microcontroller. The Arduino IDE is available for Windows, macOS, and Linux, and it is open source.

To program the Arduino Uno in C++, you will need to use the Arduino IDE. First, make sure that you have the Arduino IDE installed on your computer. Then, follow these steps:

1. Connect your Arduino Uno to your computer using a USB cable.
2. Open the Arduino IDE.
3. Select the correct board and serial port in the Arduino IDE.
4. Write your C++ code in the Arduino IDE editor.
5. Click the “Verify” button to compile your code.
6. Click the “Upload” button to upload your code to the Arduino Uno.

Happy programming!

## 2.4 Software installation

We will need softwate listed bellow:

1. Arduino IDE is basics “development environment”
2. RobDuino library for easier programming
3. Ardublockly is needed for introduction to programming

- Python is needed for running Ardublockly

4. VSC in PlatformIO proper IDE include:

- auto-completion,
- error marking (e.g. forgotten ";"),
- auto-detect USB port,
- function information

### 2.4.1 Arduino IDE

1. Go to Arduino web page Arduino->Software->Download.
2. Download Arduino IDE 1.8.9 choose Windows Install…
3. … click JUST DOWNLOAD.
4. run arduino-1.8.9.exe and follow the instructions.
5. … don't forget to install also 3rd party drivers (for Chinese version of Arduino UNO controller)…
6. if you do forget… Try this Russian drivers from page.

#### 2.4.1.1 Getting started

1. Run Arduino IDE
2. Connect Arduino Uno controller to USB port.
   Arduino Uno
3. Open simple basic program:
   files -> examples -> 01.basics -> blink

```
1  void setup() {
2    pinMode(LED_BUILTIN, OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the
       voltage level)
7    delay(1000);                       // wait for a second
8    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the
       voltage LOW
9    delay(1000);                       // wait for a second
10 }
```

4. Make this settings in Tools menu ->
   1. Board: Arduino/Genuino Uno
   2. Port: COM3 *or similar*
5. Run :
   Upload to transfere the program to Arduino UNO controller.
6. If everything is OK you will get this message:

```
1  Done uploading.
2  Sketch uses 970 bytes (3%) of program storage space. Maximum is 32256
     bytes.
3  Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes
     for local variables. Maximum is 2048 bytes.
```

9. Optional this preferences are suggested:
   `File` -> `Preferences`:
   1. `Editor Language`: English
   2. `Editor font size`: 20
   3. `Show verbose output during`: [ ]compiling [x] upload
   4. `[x] Display linenumbers`
   5. `[x] Enable code folding`

### 2.4.2 RobDuino

RobDuino is Arduino library which include some usefull functions for driving motors and on-board key usage…

#### 2.4.2.1 RobDuino Library Installation

1. Download zip file:
   - RobDuino-master.zip
2. rename RobDuino-master.zip in:
   - **RobDuino.zip**
3. run Arduino IDE
4. choose:
   - `Sketch` –> `Include Library` –> `Add .ZIP Library...`
5. find
   - `.../Download/RobDuino.zip`
   - `[OK]`

### 2.4.3 Ardublockly

Ardublockly is graphical programming environment for programming Arduino controllers. A demo version of the program is also available on-line.

Note: For actual programming you will need Arduino IDE installed.

Note: For running Ardublockly you will need to install Python program.

#### 2.4.3.1 Python Installation

1. You will have to install Python 3.7 or grater. First Download the newest version of Python.
2. Run installation file and set this settings:
    1. `[x] Add Python to PATH` in
    2. choose `Clasic Instalation`

#### 2.4.3.2 Ardublockly Installation

3. From github.com/.../ardublockly download **zip** file by clicking **Clone or download** and choosen Download ZIP file.
4. Extract `ardublockly-master.zip` to dirrectory of your choice e.g. `C:\\Program Files(x86)`
5. That is it! Installation is complete.

##### 2.4.3.2.1 Running Ardublockly

6. Find this file `C:\\Program Files(x86)\\ardublockly-master` and double-click on `start.py`. Python program should run and you should see:
    1. terminal window with some code running...
    2. and a new window should apear in your Internet Browser. If this is will not happend try to run `start.py` with right mouse button and `Start program with` then choose `Python 3.7`.

#### 2.4.3.3 Settings

7. Click `menu` and choose `Settings`:
    1. `Compiler Location`: C:\Program Files (x86)\Arduino\arduino_debug.exe
    2. `Arduino Board`: Uno
    3. `Com port`: COM3 *or appropriate one*
    4. Click `[RETURN]`.

### 2.4.4 VSC in PlatformIO

Note: For programming Arduino controllers you will need Arduino IDE installed.

Download installation file:

1. run `VSCodeUserSetup-ia32-1.49.3.exe` installation file.
2. run VSC program and click `Extensions`
3. search for `PlatformIO IDE` and
4. run `Install`.
5. restart VSC or click `Reload now`.

#### 2.4.4.1 Getting Started

Write basic program `Blink`:

1. plug in Arduino Uno.
2. open `PlatformIO - Home Page`:
   - in left icon bar find `PlatformIO`
     - `QUICK ACCESS` -> `PIO Home` -> `Open`
3. choose + `New Project`
4. Setup:
   - `Name`: ime_projekta
   - `Board`: Arduino UNO
   - `Framework`: Arduino Framework
5. click `Finish`
6. Find directory `src` (e.g. `source code`), where you can find main program code in file `main.cpp`
7. Copy-Paste this example:

```
#include <Arduino.h>
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13,HIGH);
  delay(500);
  digitalWrite(13,LOW);
  delay(500);
}
```

8. Run `Build` and `Upload`.

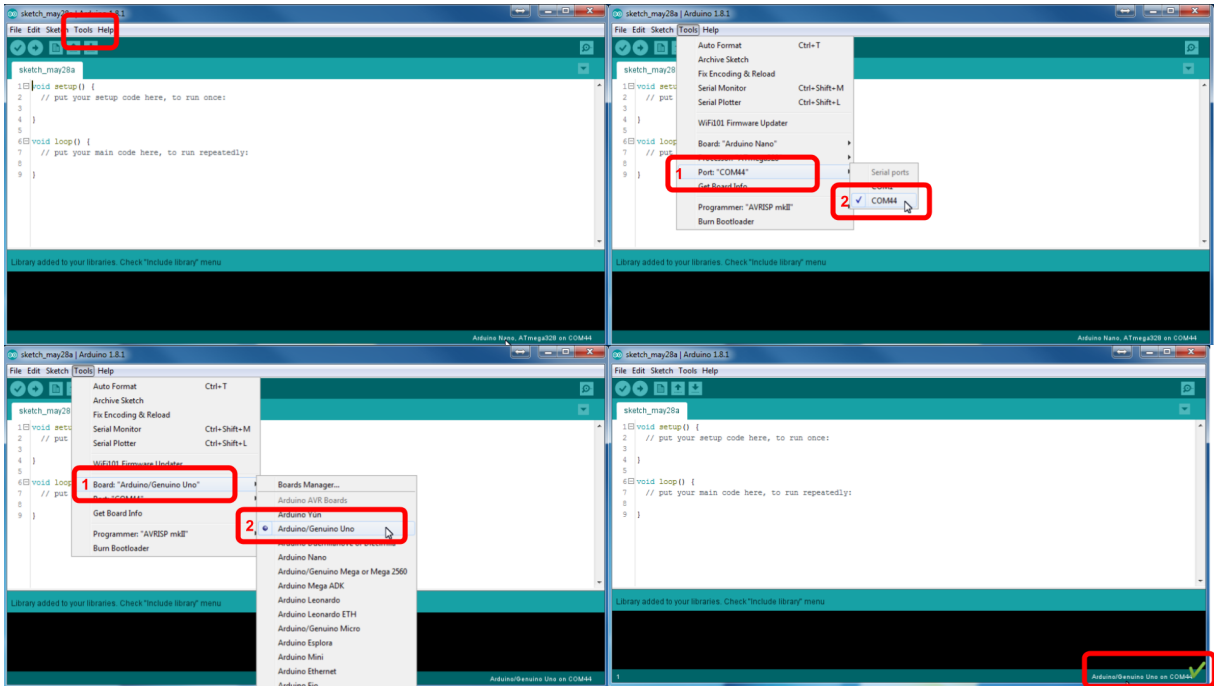
## 2.5 Setting up an Arduino development environment

## 2.6 Equipment testing

### 2.6.1 Basic testing in Arduino IDE

1. Connect the Arduino Uno to PC with proper USB cable.
   `[Arduino Uno]`
2. Open Arduino IDE program and open program with:
   `Files - Examples - 01. Basics - Blink.ino`
3. Make shure that you will set the proper settings (see sl. 2.2). From the menu choose:
   `Tools`-

   1. `Board`: Arduino/Genuino Uno
   2. `Port`: COM3



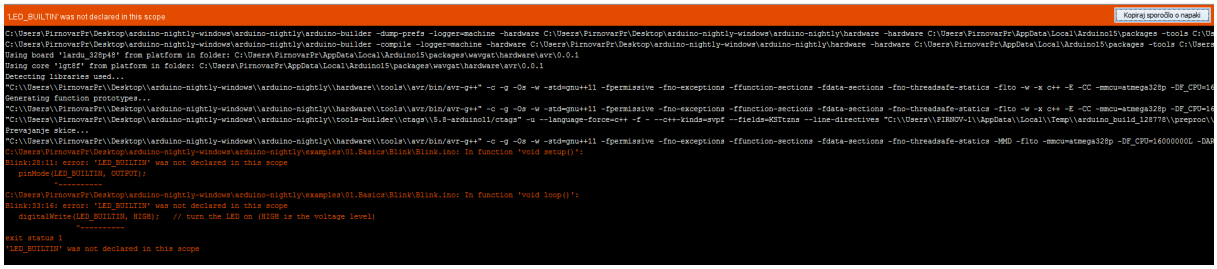

**Slika 2.2:** Arduino basic setup.

- To upload the code you can click the icon `Upload`.
  If the uploading was successful you will be prompted with the text like:

```
    Done uploading.
    Sketch uses 970 bytes (3%) of program storage space. Maximum
    is 32256 bytes. Global variables use 9 bytes (0%) of dynamic
    memory, leaving 2039 bytes for local variables. Maximum is
    2048 bytes.
```

### 2.6.2 Issues

#### 2.6.2.1 LED_BUILTIN was not declared in this scope




**Slika 2.3:** Error image.

Compiler ne ve kaj naj bi bilo “LED_BUILTIN” … na tem mesu naj bi bila številka priključka, ki ga želimo krmiliti. V tem primeru je to številka 13. Rešitvi sta lahko 2:
1. vse LED_BUILTIN zamenjaš s 13 ali
2. v vrstico pred “void setup()” dodaj `const int LED_BUILTIN = 13;`

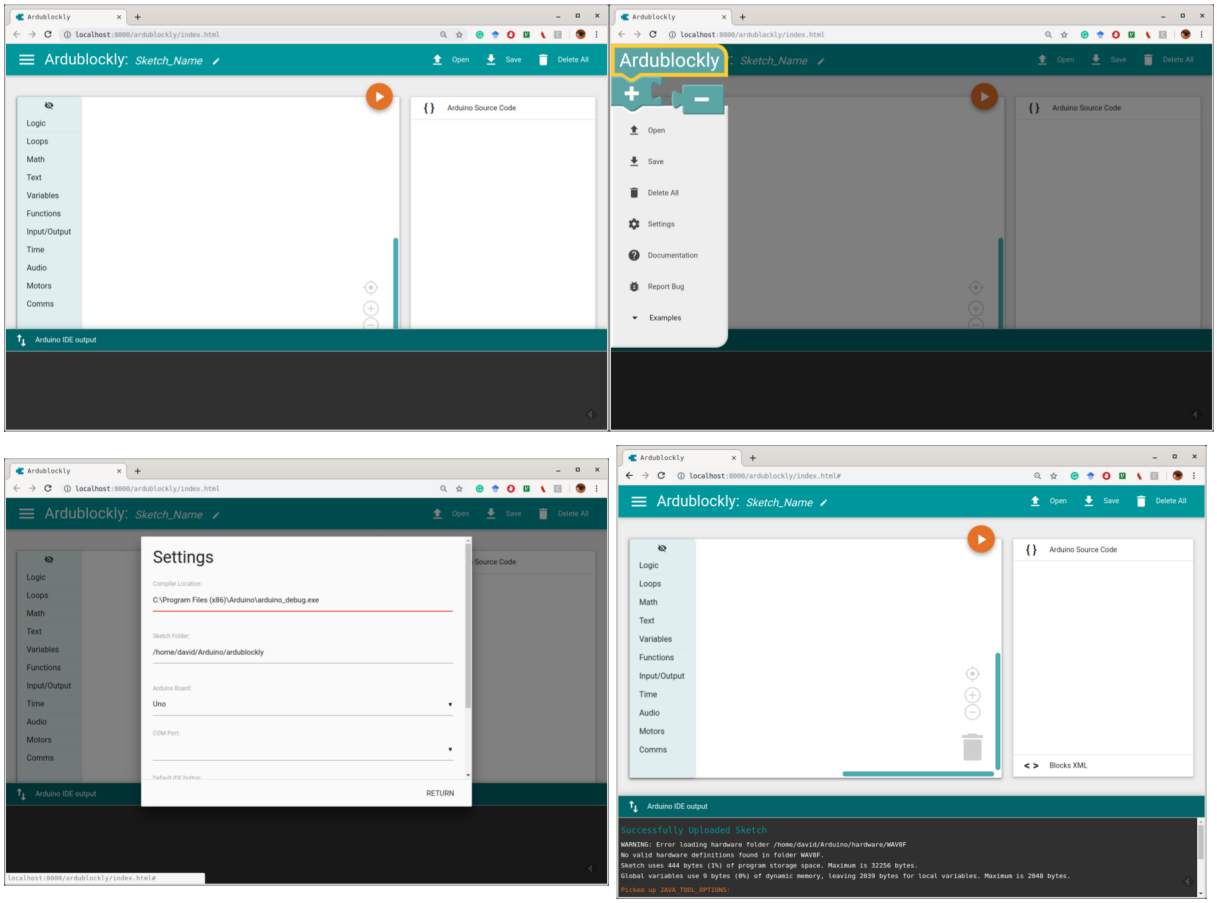
Zadnja (druga) rešitev je boljša, ker pripomore k berljivosti programa… Spremenljivka LED_BUILTIN se imenuje “razlagalna spremenljivka” ker pomaga razlagati program. Tako postane tisti komentar “// turn the LED on (HIGH is the voltage level)” nepotreben, saj sama koda pove točno enako.

### 2.6.3 Basic testing in Ardublockly

1. Connect the Arduino Uno to PC with proper USB cable.
   [Arduino Uno]
2. Run Ardublockly program. Which will be running as localhost and you will be using internet browser as IDE. The addres will be:
   http://localhost:8000/ardublockly/index.html
3. In the left corner of the program you can find [=] menu icon. From where you can choose (Slide 2 and 3)
   [] Settings:

1. `Compiler Location`: C:\Program Files (x86)\Arduino\arduino_debug.exe
2. `Arduino Board`: Uno
3. `Com port`: COM3
4. And press: `[ RETURN ]`

4. Finaly you can press button `PLAY` And if uploading was successful you will be prompted with the text (Slide 4):




**Slika 2.4:** Ardublockly basic setup.

```
    Successfully Uploaded Sketch
    WARNING: Error loading hardware folder /home/david/Arduino/hardware/
        WAV8F.
    No valid hardware definitions found in folder WAV8F.
    Sketch uses 444 bytes (1%) of program storage space. Maximum is
    32256 bytes. Global variables use 9 bytes (0%) of dynamic memory,
    leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

### 2.6.4 Summary

Before uploading the programming code always check that the right board and serial port are set.

### 2.6.5 Issues

**Ardublockly returns the Error id 55: Serial port Serial Port unavailable.**
Try to re-connect the Arduino board. Wait a moment, check the settings and choose the COM port again then try again.

### 2.6.6 RobDuino module

1. Na krmilnik Arduino Uno priključite modul `RobDuino` in naložite naslednji program:

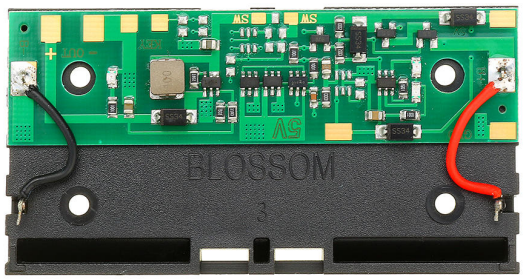
```
void setup() {
  for (int i = 0; i < 8; i++){
    pinMode(i, OUTPUT);
  }
  pinMode(A4, INPUT_PULLUP);
  pinMode(A5, INPUT_PULLUP);
  PORTD=1;
}

int l=1;
void loop() {
  char tipka_a4_is_pressed = !digitalRead(A4);
  char tipka_a5_is_pressed = !digitalRead(A5);
  if (tipka_a4_is_pressed) l = l << 1;
  if (tipka_a5_is_pressed) l = l >> 1;
  if (l < 1) l = 128;
  if (l > 255) l = 1;
  PORTD = l;
  delay(100);
}
```

2. Nato preverite delovanje obeh tipk (A4 in A5) na modulu in vrednosti izhodnih priključkov D0 .. D7.

### 2.6.7 Napajalni modul

Napajalni modul uporablja 2x Li-ion akumulatorja tipa 18650. Spodnje tiskano vezje je prikazano sl. 2.5.




**Slika 2.5:** Napajalni modul.

Dodatno smo ga opremili z:

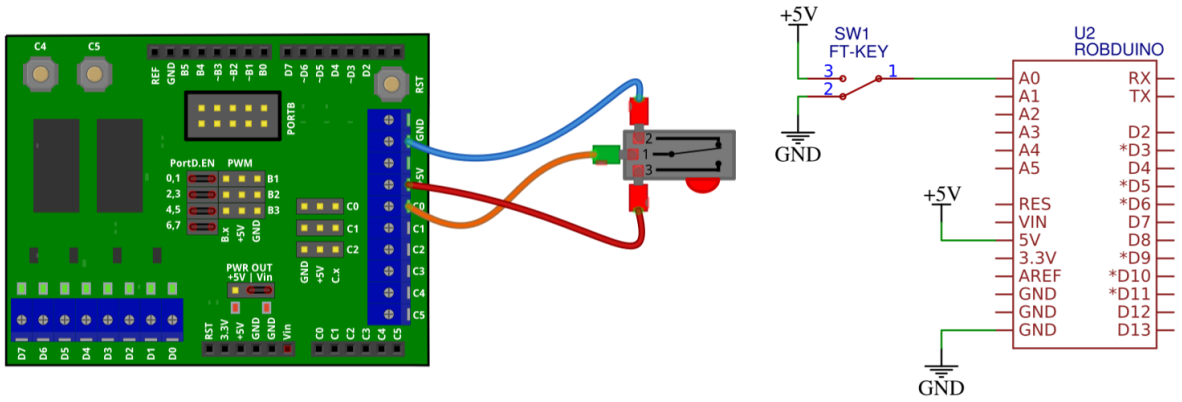
1. 2.5mm jack priključkom za napajanje,
2. 3-pinskim priključkom za napajanje,
3. preklopnim stikalom za izbiranje načina delovanja:

    1. ON - izhod za 9V je kaktiviran
    2. OFF - izključen izhod 9V napajanja in omogočeno je polnenje akumulatorjev preko 3-pinskega priključka (5V).

Pomembno: Pred prvo uporabo moramo ročno aktivirati napajalni modul tako, da povežemo GND na 3-pinskem priključku in NEGATIVNI terminal akumulatorjev.

### 2.6.8 Tipka

1. Priključite stikalo po shemi na sl. 2.6.




**Slika 2.6:** Priključitev tipke.

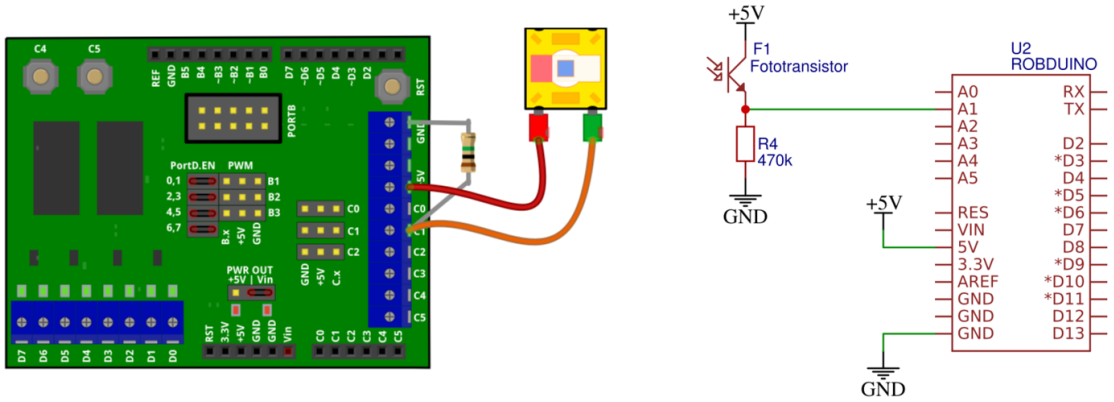
2. Nato naložite naslednji program.

```
void setup() {
    pinMode(A0, INPUT);
    pinMode(7, OUTPUT);
}

void loop() {
    char key_a0_is_pressed = digitalRead(A0);
    if (key_a0_is_pressed){
        digitalWrite(7, HIGH);
    } else{
        digitalWrite(7, LOW);
    }
    delay(100);
}
```

### 2.6.9 Svetlobni senzor

1. Priključite foto-tranzistor v delilnik napetosti z uporom, kot prikazuje sl. 2.7.




**Slika 2.7:** Priključitev foto-tranzistorja kot svetlobni senzor.

2. Nato naložite naslednji program in preverite odziv svetlobnega senzorja.

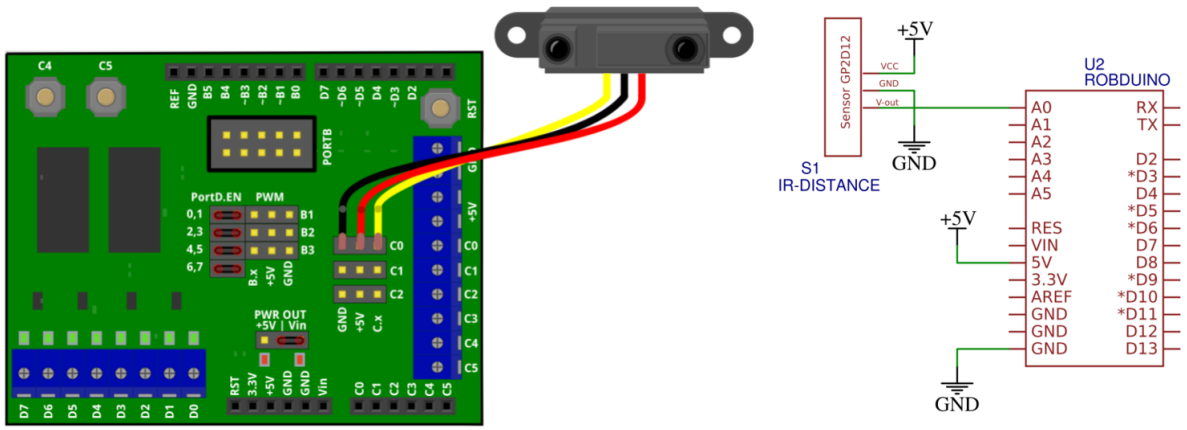
```
 void setup() {
  pinMode(A1, INPUT);
  Serial.begin(9600);
}

void loop() {
  int light_senzor_value = analogRead(A1);
  Serial.println(light_senzor_value);
  delay(100);
}
```

3. Odziv senzorja spremljajte v oknu serijske komunikacije.

### 2.6.10 IR senzor razdalje

1. IR senzor razdalje priključite na tri-pinski priključek kot je prikazano na sl. 2.8.




**Slika 2.8:** Priključitev IR senzorja razdalje.

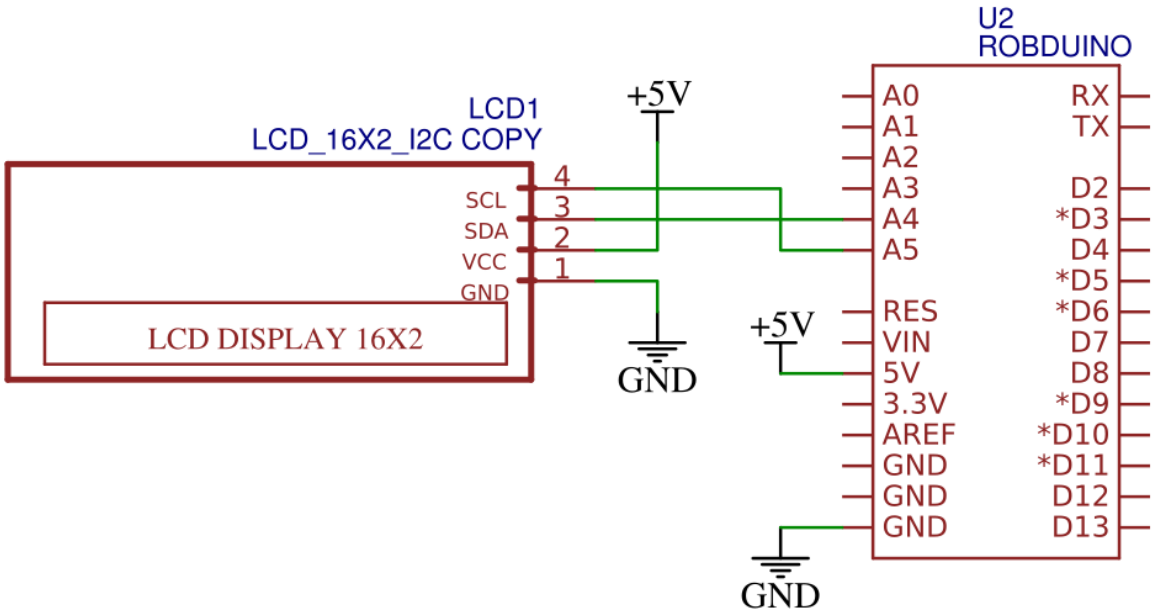
2. Delovanje senzorja preskusite z naslednjim programom, njegov odziv pa spremljajte v oknu za serijsko komunikacijo.

```
void setup() {
  pinMode(A0, INPUT);
  Serial.begin(9600);
}

void loop() {
  int distance_senzor_value = analogRead(A0);
  Serial.println(distance_senzor_value);
  delay(100);
}
```

### 2.6.11 LCD (I2C)

1. Priključite LCD na I2C vodilo kot prikazuje




**Slika 2.9:** Povezava LCD na I2C vodilo krmilnika.

2. Priskrbite si knjižnico `LiquidCristal-I2C` iz naslova:
   https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/

3. Knjižnico dodajte v Arduino IDE okolje tako, da dodate `ZIP` datoteko v :
   `Sketch >> Include Library >> Add .ZIP Library`

4. V VSC in PlatformIO vtičniku si lahko knjižnico naložite tako, da v terminalno okno vpišete ukaz
   `pio lib install "marcoschwartz/LiquidCrystal_I2C@^1.1.4"`

5. Nato preskusite naslednji program:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C Lcd(0x27, 16, 2);

void setup() {
    Lcd.init();

    Lcd.clear();
    Lcd.backlight();

    Lcd.setCursor(3,0);
    Lcd.print("Hello");
    Lcd.setCursor(6,1);
    Lcd.print("World");
}

void loop() {
}
```

Če niste prepričani kateri i2c naslov uporablja naprava na LCD-ju le tega lahko preverite s programom `I2C scanner` (https://playground.arduino.cc/Main/I2cScanner/). Običajno I2C LCD-ji, ki jih naredijo kitajski proizvajalci uporabljajo I2C naslov `0x27` , `0x3F` ali manj pogosto `0x38`.

## 2.7 Programming Arduino using C++

## 2.8 Hello world in Arduino IDE

### 2.8.1 Tasks:

1. Make a very simple program like setting the digital output bit D3 to logical state 1 or **HIGH**.

**Program 2.1:** Hello World in ArduinoIDE.

```
void setup() {
 // put your setup code here, to run once:
 pinMode(3, OUTPUT);
 digitalWrite(3, HIGH);
}

void loop() {
 // put your main code here, to run repeatedly:

}
```

2. Send the program to controller Arduino UNO .

### 2.8.2 Questions:

1. Explain the purpose of next programming characters in presented example:
    1. ;
    2. { }
    3. `pinMode(3, OUTPUT);`
    4. `digitalWrite(3, HIGH);`
    5. `//put your ...`
    6. `void setup()`
    7. `void loop()`

### 2.8.3 Summary:

#### 2.8.3.1 Using curly braces - { and }

Using curly braces in C++ is important part of riting the programming code. Imagine that you want to merge several members of programing code to a single pile. As we would separate pencils into one pile and markers to another - to be more organized. In real life we would do by elastic bundle or rope. If you have to choose single character from the keyboard to indicate that several members are combined to the same pile - which character would you choose? Probably curly braces {} are the best choice.

#### 2.8.3.2 Function Name

Function name should be stacked together from 2 - 5 short words that uniquely describing the functionality of the function. The first word should start with lower case and all the others words following should start with upper case. Some examples should be:

```
1    badname();
2    goodFunctionName();
```

#### 2.8.3.3 Function Declaration

```
1    int measre_Temperature_Avg(int temperatureSensor);
```

#### 2.8.3.4 Function Definition

```
1   void loop() {
2       //some programming
3       //code goes here...
4   }
```

#### 2.8.3.5 Function Call

```
1       digitalWrite(3, HIGH);
```

### 2.8.4 Issues:

#### 2.8.4.1 *Error: expected ';' before 'something'*

Probably you forgot to put ; (semicolon) at the end of the command. Find the row starting with "**something**" and look the row above... probably missing "**;**".

#### 2.8.4.2 *Light at the digital output D3 is not ON.*

Check if the enable switch fot the digital outputs is at the right position (ENABLE).

## 2.9 Communicating with sensors and actuators

## 2.10 Implementing control algorithms

## 2.11 Debugging and testing

# 3 ELECTRONICS FUNDAMENTALS

## 3.1 Power sources batteries AC DC adapters

## 3.2 Basic principles of electricity and electronics

## 3.3 Voltage current resistance and Ohms law

## 3.4 Circuit components and their functions resistors capacitors diodes

## 3.5 Digital output

On an Arduino Uno board, a digital output is a pin that can be used to output a digital signal, which can be either high (5 volts) or low (0 volts). Digital outputs are useful for controlling devices that are either on or off, such as LEDs, motors, and relays.

To use a digital output on an Arduino Uno board, you will need to specify which pin you want to use as an output in your code. You can do this using the pinMode function, which takes two arguments: the pin number and the mode (OUTPUT or INPUT). For example, the following code sets digital pin 13 as an output:

```
1    pinMode(13, OUTPUT);
```

Once you have set a pin as an output, you can use the digitalWrite function to set the pin to either a high or low state. For example, the following code sets digital pin 13 to a high state:

```
1    digitalWrite(13, HIGH);
```

## 3.6 Digital input

to-do

1. Push Button: a push button can be used to trigger a digital input. By connecting a push button to an Arduino digital pin and writing a sketch to register when the button is pressed, digital input can be used to trigger an action.
2. Touch Sensor: a touch sensor can be used to detect contact with a particular surface and can act as a digital input. By connecting the sensor to an Arduino digital pin and writing a sketch to listen for contact, digital input can be used to trigger an action.
3. Light Sensor: a light sensor can be used to detect light levels and can act as a digital input. By connecting the sensor to an Arduino digital pin and writing a sketch to listen for changes in light levels, digital input can be used to trigger an action."

# 4 ROBOTICS HARDWARE

## 4.1 Overview of different types of robots

## 4.2 Motor as main actuator

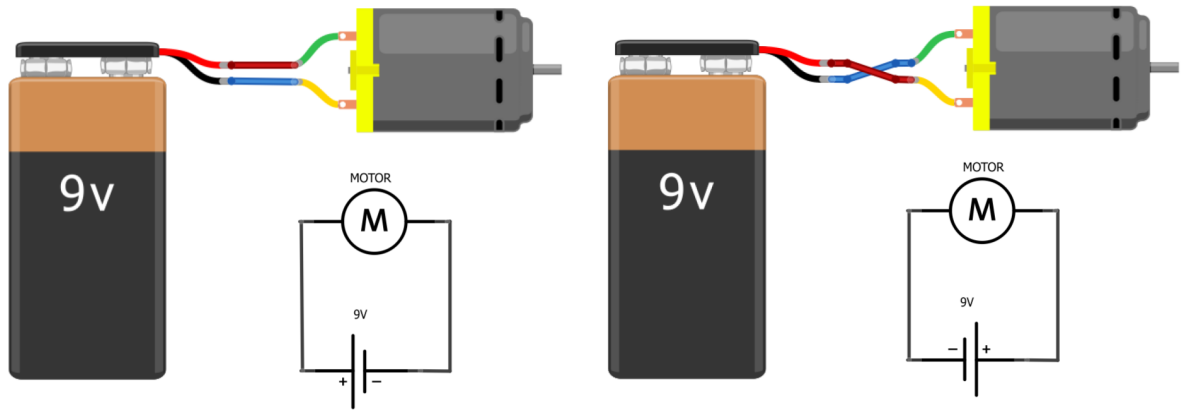
A DC motor (Direct Current motor) is an electrical machine that converts electrical energy into mechanical energy. It works by using electromagnetic principles to generate rotary motion.

Here is how a DC motor works in more detail:

- The DC motor has two main parts: the stator and the rotor. The stator is the stationary part of the motor, and the rotor is the rotating part.
- The stator consists of a coil of wire that is wound around a core. When an electric current flows through the coil, it creates a magnetic field around the core.
- The rotor consists of a permanent magnet or a coil of wire that is mounted on a shaft. When the rotor is placed inside the stator, the magnetic fields of the stator and the rotor interact with each other.
- If the stator's coil is energized with a DC current, the magnetic field it creates will rotate around the core. This causes the rotor to rotate as well, since it is attracted to the moving magnetic field.
- The speed and direction of the rotor's rotation can be controlled by adjusting the strength and polarity of the current flowing through the stator's coil. This is typically done using an H-bridge circuit, which allows the current to be reversed and the motor to run in both directions.

### 4.2.1 Task: MAKE DC MORTOT RUN

1. Connect the DC motor to the battery and make it run.
2. You can try different combinations to connect the terminals of the motor like:
    - + and -
    - - and +
    - - and -
    - + and +.




**Slika 4.1:** DC motor connection.

### 4.2.2 Questions:

1. In which direction the motor's shaft spins in different situations?
2. In which direction the electric current flow?
3. Why does motor is not spinning when both connectors are connected to + terminal of the battery?

### 4.2.3 Summary

The rotation of the DC motor depends on the direction of electric current.

### 4.2.4 Issues

#### 4.2.4.1 *When I connect the DC motor to + and - terminals of the battery the motor's shaft does not spin.*

Check the voltage of the battery… battery may be discharged.
Check the connectors of the motor… may be bad.

## 4.3 DC motor control with digital outputs

### 4.3.1 Task:

1. Connect the DC motor to Digital Output D7 and D6.

2. Write the program and check all the combinations of digital outputs; 00, 01, 10 and 11. First combination is shown in prog. 4.1

**Program 4.1:** DC Motor Control with Digital Outputs.

```
void setup()
{
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  // D7=0, D6=0
  digitalWrite(7, LOW);
  digitalWrite(0, LOW);
  delay(3000);
  // Write other combinations here...

}
void loop()
{

}
```

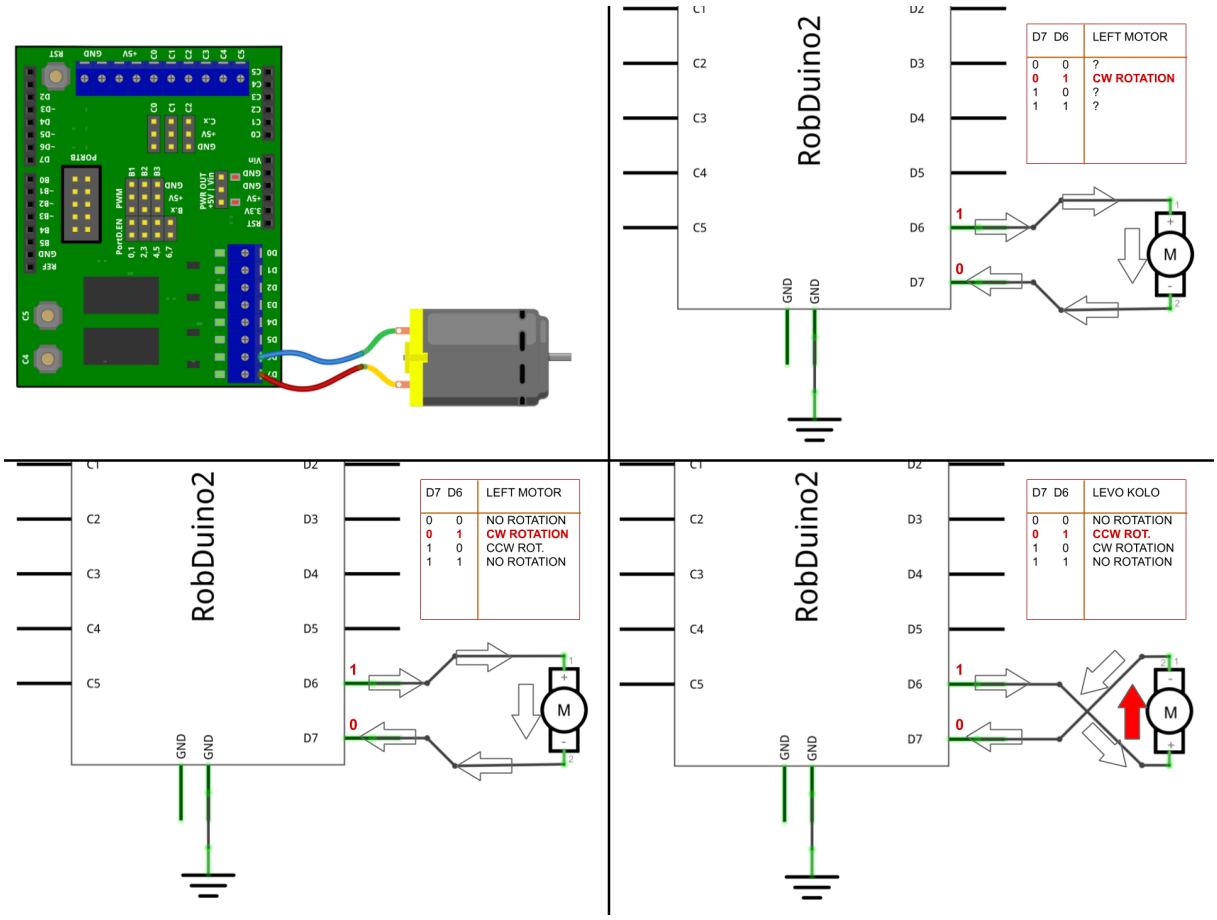
3. For each combination of digital outputs mark the state of the motor (fulfill the tbl. 4.1).

**Table 4.1:** All combinations of the states of motor's connectors.

| D7 | D6 | Motor rotation |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

### 4.3.2 Questions:

2. Try to stop the shaft of the DC motor for a short time and try to remember how difficult it is?
3. Why does motors' shaft not spinning if the digital output state are `1` and `1`.




**Slika 4.2:** Wireing the DC motor to controller.

### 4.3.3 Summary

The motor's shaft is spinning according to the direction of the electric current trough the motor. The torque is weak.

### 4.3.4 Issues

## 4.4 Gear reducer

Gear reduction is the process of using a set of gears to reduce the speed of a mechanical system while increasing the torque (rotational force). It is commonly used in robotics and other applications where it is necessary to trade speed for power.

There are several ways to achieve gear reduction, but the most common method is to use a gear train, which is a series of interconnected gears that transmit motion from one gear to another. By using gears with different sizes and ratios, it is possible to reduce the speed of the output gear while increasing the torque.

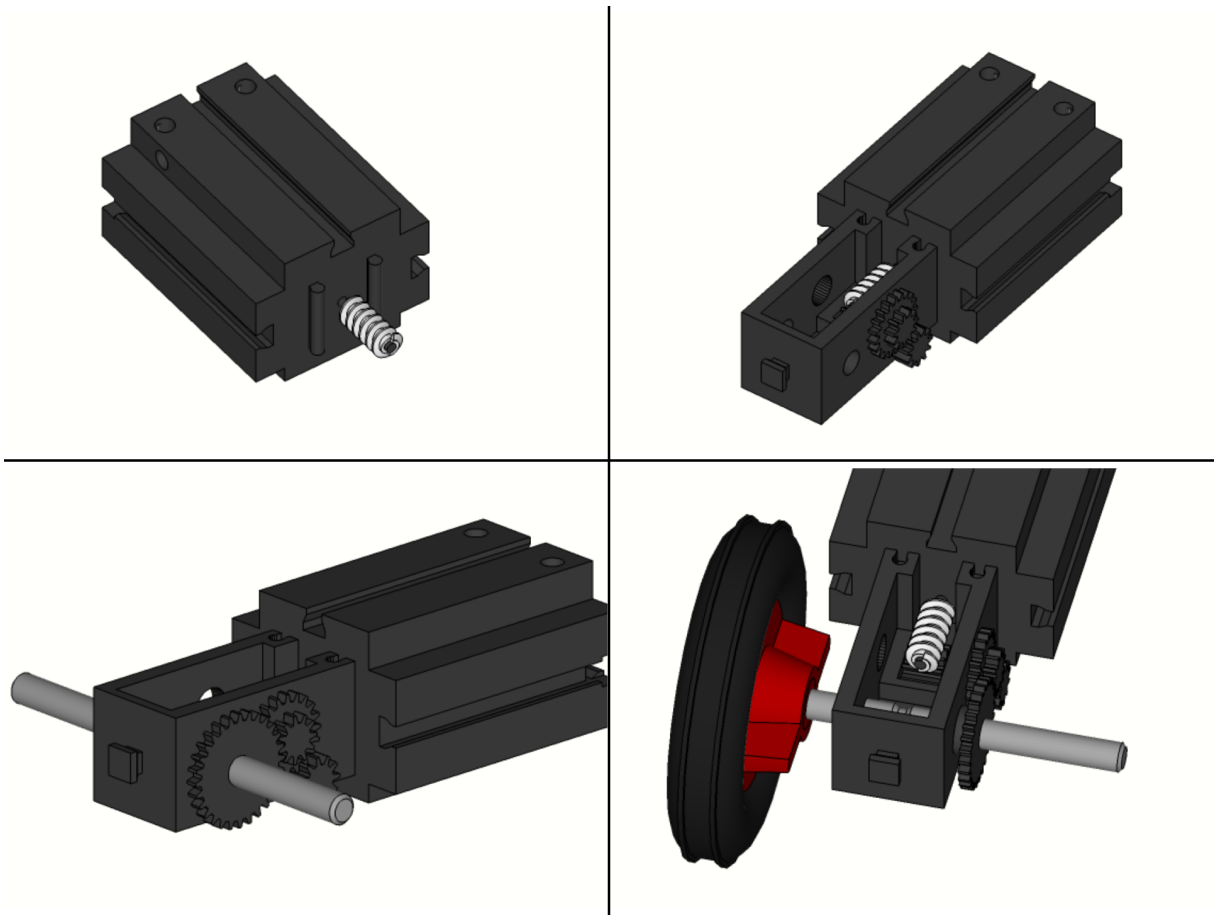
For example, consider a simple gear train with two gears: a larger driving gear (Gear A) and a smaller driven gear (Gear B). If the driving gear has 10 teeth and the driven gear has 20 teeth, the gear reduction ratio will be 2:1 (Gear B will rotate at half the speed of Gear A, but with twice the torque).

Here is the formula for calculating the gear reduction ratio:

$$Gearreductionratio = \frac{Numberofteethondrivinggear}{Numberofteethondrivengear} \tag{4.1}$$

### 4.4.1 Task:

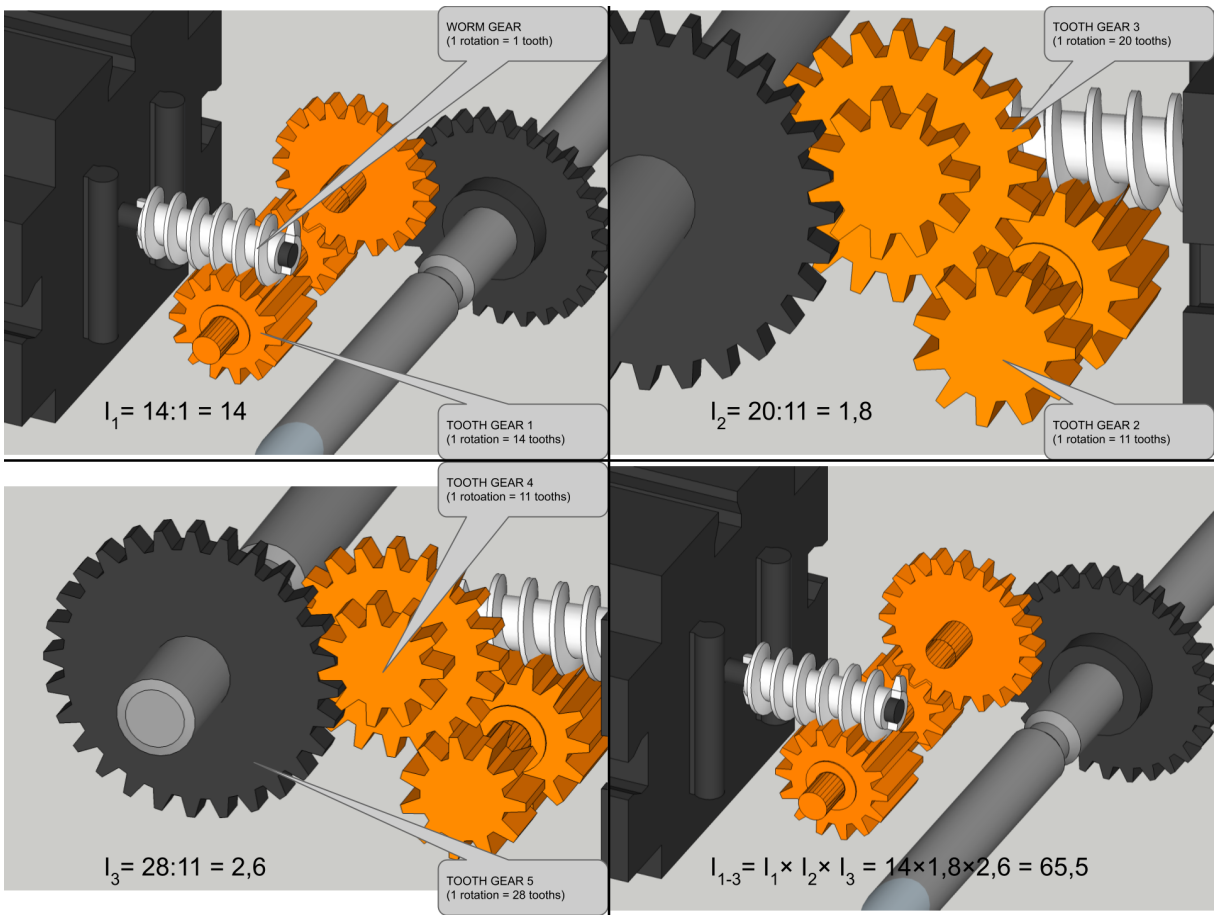
1. Add geared reductor to DC motor.
2. Try to stop the shaft of the geared reductor and compare your fillings with the stopping the motor shaft.



**Slika 4.3:** Adding the reductor to the motor.

### 4.4.2 Questions:

1. How difficult is to stop the shaft of the reductor in comparison to motor shaft?
2. How fast the shaft of the reductor is spinning in comparison to the shaft of the motor?
3. Are you able to freely rotate the shaft of the reductor by hand?
4. What happened with the produced mechanical power?
5. Try to calculate the geared ratio of the reductor.




**Slika 4.4:** Gear ration calculation.

### 4.4.3 Summary

#### 4.4.3.1 Gear ratio

The gear ratio describing the ratio between the angular velocity of input gear G1 and angular velocity of output gear G2.

$$i = \frac{\omega_1}{\omega_2}$$

Because each gear moves tooth per tooth and if two touching gears have different numbers of teeths their's angular velocity will be different. In fact the anguar velocity will be inversely proportional.

$$\frac{\omega_1}{\omega_2} = \frac{N_2}{N_1} = i$$

### 4.4.4 Issues

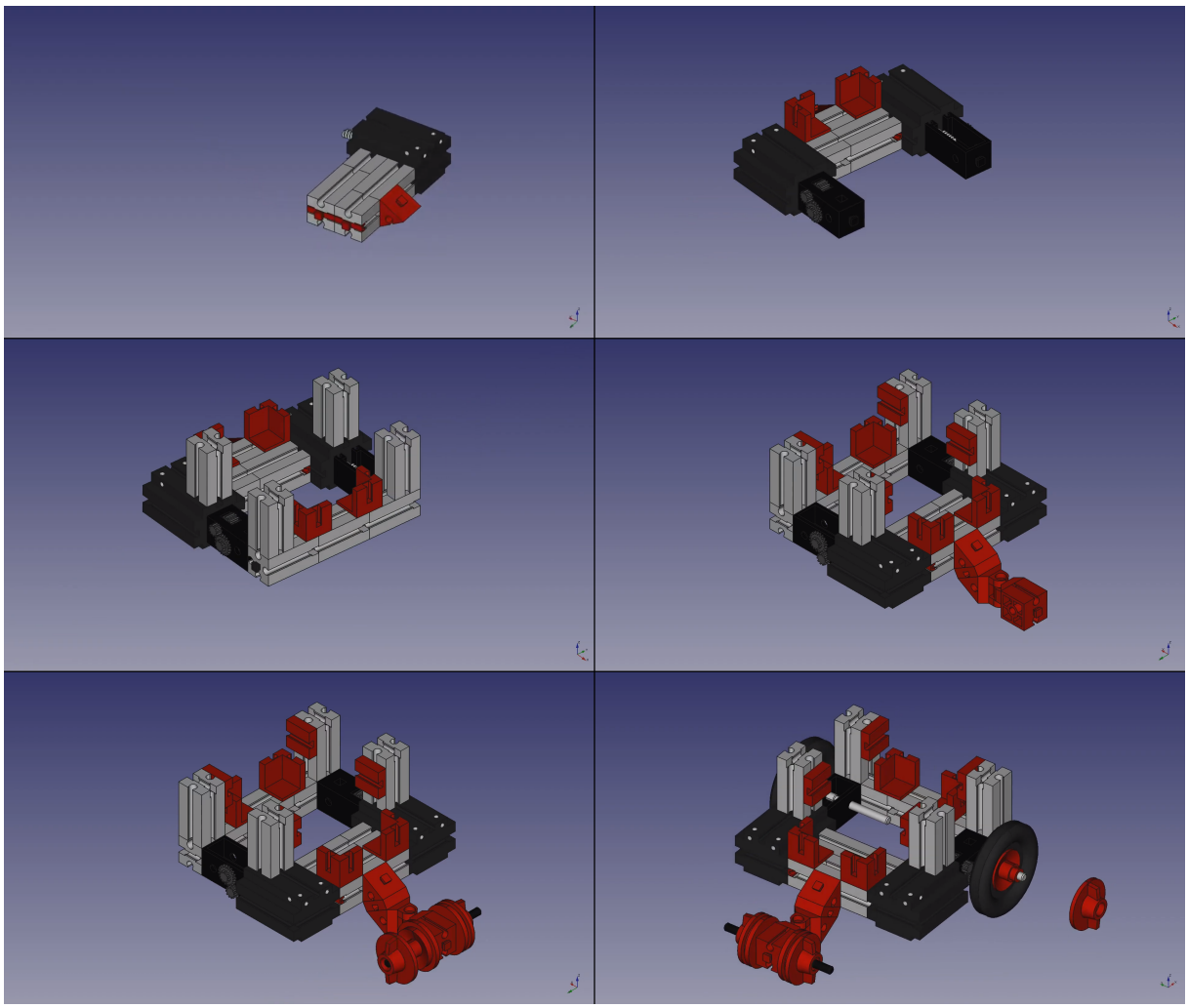
#### 4.4.4.1 *The reductor's shaft is not spinning although the DC motor is working properly.*

Check if the reductor is attached all the way to the motor. Check if the worm gear of the motor is in contact with first gear of the roductor.

## 4.5 Robot construction

### 4.5.1 Tasks:

1. Construct the mobile robot according to this sequences on the sl. **??**.



**Slika 4.5:** Construction sequences.

Or you can follow the video instructions

2. Add the battery between the red cornered bricks. The connector shuld be pointing to the back of the robot.
3. Add also the RobDuino controller. Clip the controller between the grey upstanding bricks.

### 4.5.2 Questions:

1. Where do you think is th front side of the robot?
2. Are you able to rotate the wheels freely by hand?

### 4.5.3 Summary:

<++>

### 4.5.4 Issues:

<++>

## 4.6 Understanding basic robot movement

### 4.6.1 Tasks: Make robot move

1. Connest both DC motors to RobDuino controller according to tbl. 4.2:

**Table 4.2:** Motors connections to RobDuino Output pins.

| MOTOR | RobDuino Output pins |
|---|---|
| Left DC Motor - con. A | D7 |
| Left DC Motor - con. B | D6 |
| Right DC Motor - con. A | D5 |
| Right DC Motor - con. B | D4 |

2. Write simple programming instructions to move the robot forward. Make right sequence of programming instructions (e.g. `digitalWrite()` and `delay(time_in_ms)` functions) to achive:
    1. move the robot forward,
    2. do it for 3000 ms and
    3. stop the robot.

### 4.6.2 Questions:

You probably ended up with something like prog. 4.2:

**Program 4.2:** Introduction to Programming.

```
void setup()
{
 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(6, OUTPUT);
 pinMode(7, OUTPUT);

 digitalWrite(7, HIGH);
 digitalWrite(6, LOW);
 digitalWrite(5, HIGH);
 digitalWrite(4, LOW);

  delay(3000);

  digitalWrite(7, LOW);
 digitalWrite(6, LOW);
 digitalWrite(5, LOW);
 digitalWrite(4, LOW);
}

void loop()
{

}
```

1. Is this code “easy readable”?
2. Why is readable code important?

### 4.6.3 PROGRAMMING CODE EXPLAINED

1. Zaporedje
2. Izbira
3. Ponavljanje

#### 4.6.3.1 Kako pisati pregledno kodo programa?

- Clean CODE

Proces pisanja kode je izredno NE-linearen, naše misli skačejo na različne težave in različne potrebe, ki se utegnejo pripetiti med programiranjem. Zato ni čudno, da bo prva delujoča koda zapletena in raztresena. Zato jo moramo po končnem testiranju NUJNO urediti.

```
Ko napišete delujočo kodo in ste jo stestirali, ste na pol-poti.
   Potrebno jo je še urediti in narediti berljivo! (Uncle Bob)
```

Organizacija programa naj bo podobna pisanju članka v časopisu:
1. Začnemo z naslovom, nato 2. napišete povzetek 3. nato sledijo odstavki, ki razkrivajo zgodbo v podrobnosti in 4. na koncu je zaključek z rezultati.

Taka ureditev omogoča bralcu, da besedilo lahko zapusti takoj, ko vsaj približno razume namen vsebine. Zamislite si, da berete časopis z novicami, a preberete le tiste, ki vas zanimajo, ostale pa le preletite.

#### 4.6.3.2 Manj je več

Krajše koščke programa je lažje razumeti. Zato se moramo potruditi, da vsako zaključeno celoto strnimo v podprogram ali funkcijo.

#### 4.6.3.3 Funkcije

Pri funkcijah naj bi se držali nekaj previl:

**4.6.3.3.1 Koda v funkcijah naj bo kratka** Funkcija naj naredi le eno stvar. To pomeni, da iz kode, ki je v funkciji ne moremo izvleči programske stavke in jih logično ločiti v svojo funkcijo. Seveda pa, moramo vse te majhne funkcije primerno poimenovati.

```
Imena funkcije naj bodo GLAGOLI in ne samostalniki, ker funkcije
   OPRAVLJAJO neko nalogo. (Uncle BOB)
```

**4.6.3.3.2 Oblikovanje funkcij v razrede** Pri oblikovanju funkcij lahko opazimo, da funkcije operirajo s podatki. Če se ti podatki ponavljajo ali pa so podobni moramo razmisliti o uporabi RAZREDA (callses). Naprimer krmiljenje DC motorja je tak primer. Lahko imamo več motorjev in za vsakega posebej želimo nastavljati smer in hitrost. V ta namen bi bilo smiselno pripraviti class:

```
class Motor
{
  public:
    int smer;
    int hitrost;
};
```

#### 4.6.3.4 Da je koda pregledana je verjetno bolj pomembno kot, da deluje... zakaj?

Če imamo delujočo kodo in je ta nepregledna, se lahko zgodi, da ko se bodo zahteve spremenile (posodobili bomo program) bomo skušali kodo popraviti in je ne bomo mogli. Da o možnosti, da bi nam jo popravil nekdo tretji sploh ne razmišljamo. Če pa je koda pregledna, pa ne deluje nam jo lahko pomaga rešiti bolj izkušenj programer.

Pregledno kodo lahko uporabi nekdo drug in je prenosljiva.

#### 4.6.3.5 Koda naj gre iz višjega nivoja v nižji

Med posameznimi vrsticami naj ne bo velikih prehodov med nivoji programiranja. Naprimer ne mešajmo deklaracij objektov z deklaracijami konstant.

#### 4.6.3.6 Razlagalna spremenljivka

Te spremenljivke določimo zato, da bodo if-stavki bolj berljivi. Samo spremenljivko določimo predhodno in ji damo tako ime, ki nakazuje na neko logično stanje. Izogibamo se negaciji.

```
1    stikaloJeSklenjeno = digitalRead(3);
2    if (stikaloJeSklenjeno) digitalWtire(3, HIGH);
```

#### 4.6.3.7 Kakšna so naša pričakovanja glede programske kode?

**4.6.3.7.1 Vmesne različice naj bodo delujoče** Postaviti si moramo kratke roke ob katerih bomo izdali delujočo kodo. Koda je lahko še podhranjena z uporabnimi funkcijami, a vse funkcije morajo delovati. Izdajanje vmesne različice naj vsebuje: - vse delujoče elemente kode, - njihovo dokumentacijo in - koda naj bo urejena ter - vsak njen del stestiran.

**4.6.3.7.2 Dodajanje novih funkcij v program ne sme upočasniti dela** Dodajanje novih funkcij v program ne sme upočasniti dela, če se to zgodi, je verjetno zaradi tega, ker smo pred tem naredili zmedo v programski kodi. Še en razlog več zakaj **mora** biti koda urejena.

#### 4.6.3.8 Spremembe programske kode morajo biti enostavne

Že iz besedne zveze SOFT-WARE je razvidno, da je to MEHAK - IZDELEK in ga je zato enostavno spremeniti. Zato vsaj majhne spremembe ne smejo biti težava in morajo biti hitro implementirane. K temu koraku pripomore zopet: - pregledna koda in - dobro napisan testni program

#### 4.6.3.9 Program naj bo s časom vedno boljši

#### 4.6.3.10 Popravljanje kode brez strahu

Kadar imamo občutek, da bi morali kodo izboljšati, jo dokumentirati ali narediti preglednejšo - imamo verjetno prav. Vendar se tega dela lahko ustrašimo, češ, da bomo kodo morda uničili. Tega se ne smemo nikoli ustrašiti! V veliko pomoč nam je lahko dober testni algoritem kode. Tako brez težav počasi spreminjamo kodo in jo sproti testiramo. Tak proces je zanesljiv in enostaven.

#### 4.6.3.11 Seznanjanje svojega sodelavca s kodo

Pametno je seznanjanje svojih sodelavcev z vašim delom (programiranjem) zato, da vas lahko nadomestijo, če ste vi odsotni z dela. Poleg tega pa je to dobra praksa pregleda kode in tako pogosto kodo izboljšamo z idejami sodelavcev.

#### 4.6.3.12 Testiranje kode

1. Ne napiši kode dokler nisi napisal testa zanjo in je le-ta spodletel, ker koda ne obstaja
2. Ne napiši daljšega testa kode, le toliko, da je dovolj, da spodleti.
3. Na napiši daljše kode, le toliko, da popraviš spodleteli test.

#### 4.6.3.13 Arhitektura kode

Iz arhutekture kode mora biti jasno za kakšen projekt gre. Podobno kot lahko iz tlorisa stavbe lahko povemo za kater namen je zgrajena. Enako je, če pogledamo kako je urejena arhitektura računalniške matične plošče.

Ker gre pri robotiki v najosnovnejšem primeru za S-R-A loop bi verjetno bilo primerno, da je tudi arhitektura kode taka.

#### 4.6.4 Summary:

##### 4.6.4.1 <++>

#### 4.6.5 Issues:

##### 4.6.5.1 <++>

## 4.7 Sensors and actuators

# 5 INTRODUCTION TO C++

C++ is a high-performance programming language that is widely used for building software applications. It was developed by Bjarne Stroustrup in 1979 as an extension of the C programming language. C++ is an object-oriented language, which means that it provides features for organizing and modularizing code in the form of "objects." C++ is also a compiled language, which means that the source code is converted into machine code by a compiler before it can be run on a computer.

Here are some basic concepts in C++:

**Variables**: A variable is a named location in memory that stores a value. In C++, you must specify the data type of a variable when you declare it. For example:

```
int x;      // declares a variable x of type int
float y;    // declares a variable y of type float
char c;     // declares a variable c of type char
```

**Operators**: Operators are special symbols that perform specific operations on one or more operands. C++ has a variety of operators, including arithmetic operators (e.g., +, -, *, /), comparison operators (e.g., ==, !=, >, <), and logical operators (e.g., &&, ||, !).

**Control structures**: Control structures are statements that control the flow of execution in a program. C++ has several types of control structures, including if statements, for loops, and while loops.

**Functions**: A function is a block of code that performs a specific task. C++ has a large standard library of functions, and you can also define your own functions. A function definition has the following syntax:

```
return_type function_name(parameter list) {
   // function body
}
```

**Object-oriented programming**: As I mentioned earlier, C++ is an object-oriented language, which means that it provides features for organizing and modularizing code in the form of "objects." An object is a self-contained unit of code that represents a real-world entity, such as a person, a car, or a bank account. Objects have attributes (data) and behaviors (functions). In C++, you can define classes to create objects.

## 5.1 Basic syntax and structure of a C++

The basic syntax of C++ consists of the following five elements:

• Variables: Variables are used to store data, such as numbers and strings. They are declared with a type, such as int or char, followed by an identifier, and must be initialized with a value.

• Operators: Operators are used to perform operations on variables, such as addition, subtraction, multiplication, and division.

• Expressions: Expressions are used to combine operators and variables and are evaluated by the compiler to produce a single value.

• Control Structures: Control structures are used to control the flow of the program and include if-else statements, loops, and switch statements.

• Functions: Functions are reusable blocks of code that can be used to perform tasks. A function must be declared before it is used.

Here is an example of a basic C++ program that blinks LED on a 13-th pin of an Arduino Uno controller:

**Program 5.1:** Native C++ program for ATmega328.

```
#include <avr/io.h>
#include <util/delay.h>

// Function declaration
void setup();

int main()
{
    // Variable declaration
    int time_ms = 1000;
    // Function call
    setup();
    // Main LOOP program instructions
    while (true)
    {
        PORTB |= (1<<PINB5);
        _delay_ms(time_ms);
        PORTB &= !(1<<PINB5);
        _delay_ms(time_ms);
    }
    return 0;
}
// Function definition
void setup() {
    PDDB |= (1<<PINB5);
}
```

Programming an Arduino Uno board in native C++ is much more difficult than in Arduino IDE. Arduino IDE makes it easier for users to write and debug code without having to know the details of the underlying hardware. In addition, the IDE provides many additional functions which simplify the usage of additional peripherals and actuators such as serial communication, LCDs, servo motors, step motors… This is especially true and important for beginners.

### 5.1.1 Tasks:

1. <++>
2. <++>
3. <++>

### 5.1.2 Questions:

1. <++>
2. <++>
3. <++>

### 5.1.3 Summary:

#### 5.1.3.1 <++>

### 5.1.4 Issues:

#### 5.1.4.1 Not including a semicolon at the end of each statement:

Every statement in C++ must end with a semicolon. If a semicolon is omitted, the code will not compile correctly.

#### 5.1.4.2 Not properly formatting the code:

Properly indenting and spacing code is important in C++ to make the code easier to read. Not formatting the code correctly can lead to syntactical errors.

#### 5.1.4.3 Not using correct capitalization:

C++ is a case sensitive language and therefore proper capitalization is important. If the wrong capitalization is used, it can lead to syntax errors.

## 5.2 Writing clean code

In order to make your code readable you have to clean your code regularly. This step is very important to not to slow down the programming process in the future programming. You will probably spent the same amount of time cleaning the code that you needed for writing a working version.

In general you can follow some rules:

1. Use FUNCTIONS for every single action,
2. COMMENT the code only where is necessary,
3. Use EXPLANATORY CONSTANTS and VARIABLES

to make your code clean.

Our aim is to write more readable code like in prog. 5.2:

**Program 5.2:** Writing Clean Code.

```
#include "RobotMovingFunctions.h"
void setup()
{
  setIOpins();
  moveForward();
  delay(3000);
  stopTheRobot();
}

void loop()
{

}
```

... we will do it in several steps.

### 5.2.1 Tasks:

1. Write programming functions for moving the robot in several dirrections:
    1. `moveForward()`,

2. `moveLeft()`,
3. `moveRight()`,
4. `moveBackward()`,
5. `stopTheRobot()`.

2. Save all the functions into header file: `RobotMovingFunctions.h`. An example of header file is shown in prog. 5.3

**Program 5.3:** Header file example of Robot moving functions.

```
/*****************************
 * IO pins of the Robot
 *****************************/
const int LEFT_MOTOR_PIN_1  = 7;
const int LEFT_MOTOR_PIN_2  = 6;
const int RIGHT_MOTOR_PIN_2 = 5;
const int RIGHT_MOTOR_PIN_1 = 4;
/*****************************
 * Function declarations
 *****************************/
void setIOpins();
void moveForward();
/*****************************
 * Function definitions
 *****************************/
void setIOpins(){
  pinMode( LEFT_MOTOR_PIN_1, OUTPUT);
  pinMode( LEFT_MOTOR_PIN_2, OUTPUT);
  pinMode(RIGHT_MOTOR_PIN_1, OUTPUT);
  pinMode(RIGHT_MOTOR_PIN_2, OUTPUT);
}
void moveForward(){
  digitalWrite( LEFT_MOTOR_PIN_1, LOW);
  digitalWrite( LEFT_MOTOR_PIN_2, HIGH);
  digitalWrite(RIGHT_MOTOR_PIN_1, LOW);
  digitalWrite(RIGHT_MOTOR_PIN_2, HIGH);
}
```

### 5.2.2 Questions:

1. Explain why functions contribute to more readable code.
2. Why is good to use explanatory variables?
3. <++>

### 5.2.3 CLEAN CODE EXPLAINED

#### 5.2.3.1 Comments - YES and NO

Comments are very helpful and necessary. Keep them short and meaningful whenever is needed. May also help during thinking process while beginning designing the code.

```
1   // robot will go forward
2   digitalWrite(7,HIGH);
3   digitalWrite(6,LOW);
4   digitalWrite(5,HIGH);
5   digitalWrite(4,LOW);
```

Don't use comments where the code is self-explanatory, for example:

```
1     delay(3000); //wait for 3000ms
```

#### 5.2.3.2 Functions

Concatenate programming code into meaningful functions is a must! Previous example of code for `driving a robot forward` is very difficult to understand at first sight. We can make cleaner code as is shown in nex example where is easier to understand what-is-what:

```
1   void robotForward()
2   {
3     digitalWrite(LEFT_MOTOR_PIN_1,HIGH);
4     digitalWrite(LEFT_MOTOR_PIN_2,LOW);
5     digitalWrite(RIGHT_MOTOR_PIN_1,HIGH);
6     digitalWrite(RIGHT_MOTOR_PIN_2,LOW);
7   }
```

Compact code is more understandable than large one, see next example:

```
1   void setup()
2   {
3     setIOpins();
4     robotForeward();
5     delay(3000);
6     robotStop();
7   }
```

**5.2.3.2.1 Function declaration** Function declaration is highly advisable since allow you a quick overview of available functions in a current file. It is like a table of functions with it's return type and parameters. All declarations are tipically found at the beginig of the file.

```
void moveForward();
void moveLeft();
void moveLeft_PWM(int pwm_value);
```

**5.2.3.2.2 Function Definition** A function definition provides the actual body of the function.

```
void robotForward()
{
  digitalWrite(LEFT_MOTOR_PIN_1,HIGH);
  digitalWrite(LEFT_MOTOR_PIN_2,LOW);
  digitalWrite(RIGHT_MOTOR_PIN_1,HIGH);
  digitalWrite(RIGHT_MOTOR_PIN_2,LOW);
}
```

#### 5.2.3.3 Constants

Use explanatory constants to more clearly represent unintuitive numbers or other abstract values. Use these constants instead of comments since these numbers will appear several times during programming code.

```
const int LEFT_MOTOR_PIN_1 = 7;
const int LEFT_MOTOR_PIN_2 = 6;
```

Now you can easily see why the pins are set as OUTPUT. Because there is `Left motor` attached.

```
void setIOpins()
{
  pinMode(LEFT_MOTOR_PIN_1, OUTPUT);
  pinMode(LEFT_MOTOR_PIN_2, OUTPUT);
}
```

#### 5.2.3.4 Variables

Use explanatory variables to make if-statements easily readable and thus understandable. Make `boolean` variables as short statements with no inverting logic.

For example we will use the case where the robot should stop as soon it hits the obstacle with front bumper. The worst case scenario of the program could look like this (we have all done it at some point):

```
1  void loop()
2  {
3    if (digitalRead(A0) == FALSE){
4      digitalWrite(7, HIGH);
5      digitalWrite(6, LOW);
6      digitalWrite(5, HIGH);
7      digitalWrite(4, LOW);
8    }else{
9      digitalWrite(7, LOW);
10     digitalWrite(6, LOW);
11     digitalWrite(5, LOW);
12     digitalWrite(4, LOW);
13   }
14 }
```

And more clean representation of same functionality is shown in next example of the code. Line 3 is easy readable, simple, clear and easy understandable.

```
1  void loop()
2  {
3      int front_bumper_is_pressed = digitalRead(BUMPER_INPUT);
4      if (front_bumper_is_pressed) robotStop(); else robotForward();
5  }
```

#### 5.2.3.5 Header files

To keep our main program file short and transparent as possible we can put supporting code (e.g. functions, settings, ...) into separate file and include it at the beginning of the main program. These files are called header files. We can write a function and save it into header file called “calculate.h”

```
1  int sumTwoNumbers(int A, int B)
2  {
3    return A+B;
4  }
```

In our main program we can include the header file and use the function:

```
1  #include "calculate.h"
2
3  int main()
4  {
5    int a = 5, b = 3;
6    int sum = sumTwoNumbers(a, b);
7  }
```

There are several reasons to use header files in C++:

**Code organization**: Header files allow you to organize your code into logical units, which can make it easier to understand and maintain. For example, you can use a header file to group together related function declarations, constants, and data types.

**Code reuse**: Header files can be included in multiple source files, which allows you to reuse the same code in multiple places without having to copy and paste it. This can save time and reduce the risk of errors.

**Compilation speed**: When you include a header file in a source file, the compiler does not need to recompile the code in the header file every time it compiles the source file. This can significantly improve the compilation speed of your program, especially if the header file contains a large amount of code.

**Separation of interface and implementation**: Header files can be used to separate the interface (the function declarations and data types that are visible to the rest of the program) from the implementation (the actual code that performs the tasks). This can make it easier to change the implementation of a module without affecting the rest of the program.

#### 5.2.3.6 Pre-process

The preprocessors are the directives, which give instructions to the compiler to pre-process the information before actual compilation starts (e.g. `#include` is one of them). You can easily use as such text substitutions for more clear code reading.

```
1    #define LEFT_MOTOR_PIN_1 7
2    #define LEFT_MOTOR_PIN_2 6
```

Remember! `#define` is really a simple text substitution and is not type-safe. Furthermore, we have to be certain that our definition will not interfere with other code used outside of our scope e.g. `libraries`. The last example is not the best representation of `#define` usage. In these case the `const int` is more proper way to go (allowed type checking, debugging). But `#define` has other benefits where `const` can not be used.

**5.2.3.6.1 Translations** The substitutions can be used as a translation and simplification of code. Such code can be introduced to very young children to get involved in programming.

```
#define vkljuci_led digitalWrite(13,HIGH)
#define izkljuci_led digitalWrite(13, LOW)
#define pocakaj(time) delay(time)
void loop(){
  vkljuci_led;
  pocakaj(1000);
  izkljuci_led;
  pocakaj(1000);
}
```

**5.2.3.6.2 Debugging** You can even substitute function names e.g. `debug(txt)` with `Serial.println(txt)` and easily separate debugging code lines from necessary serial print of data.

```
#define debug(txt) Serial.porintln(txt)
void setup()
{
  Serial.begin(9600);
  debug("Running...")
}
void loop()
{
  unsigned long myTime = millis();
  Serial.println(myTime);
  delay(1000);
}
```

When we are done with programming and debugging is not needed anymore we can simply change `#define` line to nothing:

```
#define debug(txt)
```

And these programming sentences will not be used. More sophisticated example is shown where programmer can switch between debugging mode (with `#define DBG 1`) and normal operation (with `#define DBG 0`) where code statement `debug("Running...")` will not even compile into program.

```
#define DBG 1
#if DBG == 1
#define debug(txt) Serial.porintln(txt)
#else
#define debug(txt)
#endif
void setup()
{
  Serial.begin(9600);
  debug("Running...")
}
```

### 5.2.4 Summary:

### 5.2.5 Issues:

#### 5.2.5.1 What is the difference between `const int` and #define?

`#define` is textual replacement, so it is as fast as it can get. Also it can save some RAM. The downside is that it's not type-safe.

`const` variables may or may not be replaced inline in the code. It is guaranteed to be type-safe though since it carries its own type with it.

## 5.3 Testing programming code

Testing code in Arduino is important because it helps to ensure that the code is working correctly and producing the desired results. Testing can help to catch bugs and errors in the code, and can also help to verify that the code is performing the tasks that it is intended to perform. By thoroughly testing code, you can improve the reliability and functionality of your Arduino projects.

The Serial.println() function is a useful tool for debugging Arduino code because it allows you to print information to the serial monitor, which can help you understand what your code is doing and troubleshoot any problems.

To use Serial.println() for debugging, you will need to include the Serial library at the top of your sketch and initialize the serial monitor using the Serial.begin() function. Then, you can use Serial.println() to print strings or variables to the serial monitor.

Here is an example of how to use Serial.println() for debugging in an Arduino sketch:

```
void setup() {
  // Initialize serial communication at a baud rate of 9600
  Serial.begin(9600);
}

void loop() {
  int x = 10;
  int y = 20;

  // Print the value of x and y to the serial monitor
  Serial.print("x = "); Serial.println(x);
  Serial.print("y = "); Serial.println(y);

  // Print the result of an operation to the serial monitor
  int sum = x + y;
  Serial.print("sum = "); Serial.println(sum);
}
```

To view the output of the Serial.println() statements, you will need to open the serial monitor in the Arduino IDE. You can do this by clicking on the “magnifying glass” icon in the top right corner of the window.

### 5.3.1 Testing mode

Since testing programming code and hardware is one of the key features in designing a robot it is recommended that testing functions are a part of your main program.

We will write a program which will be triggered by sending specific text over the serial communication. For example if we will send the string `forward` the function `moveForward()` will be executed. The function `serialEvent()` is trigged when some data are available on serial communication. A basic example of such functionality is shown in prog. 5.4:

**Program 5.4:** Testing programming code.

```
#include "RobotMovingFunctions.h"
//#include <RobDuinoSerialTesting.h>

void setup()
{
  Serial.begin(115200);
  setIOpins();
  moveForward();
  delay(3000);
  stopTheRobot();
}

void loop()
{

}

void serialEvent(){
  String test_string = Serial.readString();
  if      (test_string == "forward")  moveForward();
  else if (test_string == "stop")     stopTheRobot();
  else Serial.println("\n" + test_string + " is not valid command.");
}
```

### 5.3.2 Task: Try testing workflow

1. Explore the testing functionality of this example.
2. Complete the testing functionality with other functions available to control the robot movement.

In further lectures we will be using more advances `Testing mode` where single digital outputs can be controlled; and inputs will be measured in digital and analog manner. This testing process is available if you have installed `RobDuino Libraty` (see Program Installing chapter). The testing mode will be trigged by the command `testing`. The output will show every output state:

```
******* Testing mode ********************
Dig. Out    Dig. In.    An.In.
D0 = 1      A0 = 0      A0 = 293
D1 = 0      A1 = 0      A1 = 334
D2 = 0      A2 = 0      A2 = 353
D3 = 0      A3 = 0      A3 = 369
D4 = 0      A4 = 0      A4 = 339
D5 = 0      A5 = 0      A5 = 264
D6 = 0
D7 = 0
----------------------------------------
```

### 5.3.3 Task: RobDuino module testing

3. Delete `serialEvent()` function in your program and uncomment line 2 in prog. 5.4: `#include "RobDuinoSerialTesting.h"`.
4. Explore testing functions with command `testing` writing it into Serial Monitor and you will get this respond:

```
*** Testing mode - menu - ******************
*    help - prints this text menu
*    D5   - toggles output state of D5
*    Dx   - toggles output state of any Dx,
*           x is any num. from 0 .. 13.
*    run  - toggles monitoring od I/O pins
*    exit - exits the Testing mode.
*----------------------------------------
Type any command to continue ...
```

### 5.3.4 Questions:

1. Explain why testing is important.
2. Describe the techniques of testing.
3. What parts of the robot should be tested regularly.

### 5.3.5 Summary:

#### 5.3.5.1 Testing mode

### 5.3.6 Issues:

#### 5.3.6.1 How can I get RobDuinoSerialTesting working.

Basicaly you need to do these stps:

1. install RobDuino Library
2. put this code at the top of your porgram:
   `#include <RobDuinoSerialTesting.h>`
3. Compile and write the porgram to your Arduino UNO controller
4. Open `Serial Monitor` window
5. and write `testing` command into prompt.

## 5.4 Programming loops: FOR-NEXT & DO-WHILE

It is very often needed, that we want to repeat some part of code several times. In that case we can use programming loops where we can specify which code should be repeated. In general there are two very often situation where we are using the programming loops:

1. We know `how many times` some code should repeat and
2. The code is `repeated` **`while`** `the condition` is met.

### 5.4.1 For-Next Loop

So called `For-Next` loop is used whenever the repetition of the code can be controlled by a `counter`. Counter is a number with some **starting value** and gets incremented by each repetition of the code. When `counter` reaches the given **ending value** repetition will stop. Typical examples where `For-Next` loop is used are:

- filling an array of data,
- summarising of all the costs in the bill
- robot should turn for **8 times** with 45 degree step to complete full rotation.

### 5.4.2 Do-While Loop

`Do-While` loop is used in situations where we can not predict the numbers of repetitions in advanced. In this case we must state the `condition` that must be met to repeat the code. The repetition of the code will be terminated when the `condition` will not hold anymore. Typical examples are:

- read the content to end of file,
- divide some number by 2 while we can,
- while no obstacle is in front of the robot it should drive forward

### 5.4.3 Task: FOR-NEXT LOOP

1. For example the next prog. 5.5 repeats the functions **robotLeft()** and **robotRight()** for **10 times** and robot will do a funny "dancing" move.

**Program 5.5:** Programming Loops.

```
#include "RobotMovingFunctions.h"

void setup()
{
  setIOpins();
  // Repeating Left and Right movement
  // for 10 times to make a danging move
  for (int i = 0; i < 10; i++)
  {
    robotLeft();
    delay(100);
    robotRight();
    delay(100);
  }
  stopTheRobot();
}

void loop()
{

}
```

2. Experiment a bit more with such programming techniques and change some code:
    - value of `i`,
    - duration of `delay()` function,
    - add some other functions to the `for-next` loop…

### 5.4.4 Task: DO-WHILE LOOP

3. Change the `for-next` loop with this `do-while` loop. Can you predict the result?

```
  while ( 1 == 1 ){
    robotLeft();
     delay(100);
      robotRight();
     delay(100);
  }
```

Presented `do-while` loop is not an useful example as the condition `( 1 == 1 )` will never change and will be always `true`. So, we created an infinite loop. `Do-While` loop is far more usable if in the condition is some sensor's value, as we will see in next sections.

### 5.4.5 Questions:

1. Name the situation where `for-next` loop can be used.
2. What is the purpose of a `counter` in `for-next` loop?
3. What is the difference between `for-next` and `do-while` loops?

### 5.4.6 Summary:

#### 5.4.6.1 For-loop

<++>

### 5.4.7 Issues:

#### 5.4.7.1 *Can I measure the execution time of the loop?*

Yes, you can. You must save the time before the loop and save the time after the loop is executed. The difference in these two values is the spent in the execution of the loop. A minimal working example counld look like this:

```
  unsigned long start_time = millis();
  for (int i = 0; i<100; i++)
  {
    //some code in this loop
  }
  unsigned long stop_time = millis();
  unsigned long loop_duration = stop_time - start_time;
```

#### 5.4.7.2 *Can I exit a while loop.*

Yes, you can use the “break” statement to exit a while loop in C++. However, this is not a common practice it is advised to set appropriate condition to exit a while loop. Here is an example of using “brake” statement:

```
int x = 0;
while (x < 10) {
  Serial.println(x);
  x++;
  if (x == 5) {
    break;
  }
}
```

This code will output the following to the serial port:

```
0
1
2
3
4
```

In this example, the “break” statement is used to exit the while loop when the value of “x” becomes 5. As a result, the loop only executes 5 times, rather than 10 times.

It is also possible to use the “continue” statement to skip the remainder of the current iteration of a loop, without exiting the loop entirely. For example:

```
int x = 0;
while (x < 10) {
  x++;
  if (x % 2 == 1) {
    continue;
  }
  Serial.println(x);
}
```

This code will output the following to the serial port:

```
2
4
6
8
10
```

In this example, the “continue” statement is used to skip the remainder of the current iteration of the

loop if the value of “x” is odd. As a result, only the even values of “x” are printed.

## 5.5 Variables and data types

In earlier examples we have stored some values into `variables` (e.g counting **`for`** `loop` repetition). Variables are the containers for storing data values usually located in RAM (also in EPROM, FLASH …). In order to store different data (e.g. numbers, words …) we have to use different type of variables. The declaration of the variable ( =creation ) has next syntax:

```
type varialble_name = value;
```

With next example we will solve the problem how to make light blinking while the robot is driving in reverse.

### 5.5.1 Task: USING VARIABLES

1. Start with this example of driving the robot for 3s forward and then for 3s backward. Test program example in prog. 5.6. Then try to add some code to blink the light while the robot is driving backward.

**Program 5.6:** Variables and Data Types.

```
#include "RobotMovingFunctions.h"
void setup()
{
  setIOpins();

  moveForward();
  delay(3000);
  moveBack();
  deay(3000);
  stopTheRobot();
}
void loop()
{
}
```

2. As you probably find out you have to divide the duration of 3000 ms into smaller durations and meanwhile controlling the light output. This can be done with **`for`**`-next` loop which repeats 10 times.

   Change the $9^{th}$ line `delay(3000)` in previous example into **`for`**`-next` loop with 10 repetition, but with the same overall duration of 3000 ms.

```
...
moveBack();
for (int i = 0; i < 10; i++)
{
  delay(150);
  delay(150);
}
stopTheRobot();
...
```

3. Add some code for blinking the LED in the `for loop` during the robot is driving backward.

> Don't forget to set the REVERSE_LIGHT_PIN value and its `pinMode(...)`.

```
...
moveBack();
for (int i = 0; i < 10; i++)
{
  digitalWrite(REVERSE_LIGHT_PIN, HIGH);
  delay(150);
  digitalWrite(REVERSE_LIGHT_PIN, LOW);
  delay(150);
}
stopTheRobot();
...
```

4. More advanced way to do a time conditioned loop is shown in next example:

```
...
robotBack();
unsigned long start_time = millis();
int time_diff = 0;
while (time_diff < 3000)
{
  digitalWrite(REVERSE_LIGHT_PIN,HIGH);
  delay(150);
  digitalWrite(REVERSE_LIGHT_PIN,LOW);
  delay(150);
  unsigned long now = millis();
  time_diff = now - start_time;
}
stopTheRobot();
```

### 5.5.2 Questions:

1. Show some examples of programming assignment statement!
2. What is the operator for assign the value to the variable?

### 5.5.3 Summary:

#### 5.5.3.1 What is variable?

In computer programming, a variable is a storage location in memory that is used to hold a value. The value of a variable can be changed during the execution of a program.

Each variable has a name, which is used to refer to the variable in the code, and a data type, which determines the kind of value that the variable can hold.

There are several different data types in C++, including:

Integers: Integers are whole numbers that can be positive, negative, or zero. In C++, there are several different integer data types, including char, short, int, and long.

Floating-point numbers: Floating-point numbers are numbers with decimal points. In C++, the float and double data types are used to represent floating-point numbers.

Characters: Characters are single letters, digits, or symbols. In C++, the char data type is used to represent characters.

Booleans: Booleans are values that can either be true or false. In C++, the bool data type is used to represent booleans.

To use variables in C++, you will need to declare them and assign them values. Here is an example:

```
int x;        // Declare an integer variable called x
x = 10;       // Assign the value 10 to x

char c;       // Declare a character variable called c
c = 'A';      // Assign the value 'A' to c

double d;     // Declare a double variable called d
d = 3.14;     // Assign the value 3.14 to d
```

#### 5.5.3.2 Variable definition and initialization in C++

A variable definition means that the programmer writes some instructions to tell the compiler to create the storage in a memory location. The syntax for defining variables is:

```
data_type variable_name;
```

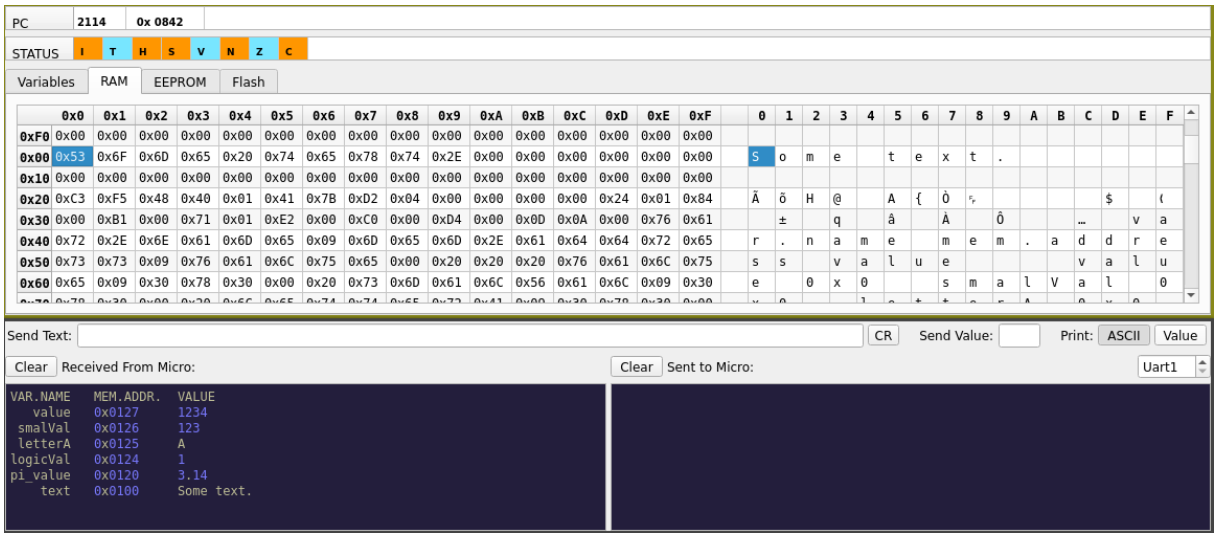
Here `data_type` means the valid C++ data type which includes int, float, double, char, wchar_t, bool and variable list is the lists of variable names to be declared which is separated by commas. Variables are declared in the above example, but none of them has been assigned any value. Variables can be initialized, and the initial value can be assigned along with their declaration.

```
    data_type variable_name = value;
```

Examples:

```
    int value = 1234;        // whole numbers from -32768 .. 32767
    char smalVal = 123;      // whole numbers from 0 .. 255
    char letterA = 'A';      // character value like !"#0123..ABC..xyz
    bool logicVal = true;    // 0 and 1 or false and true
    float pi_value = 3.14;   // from -3.4E+38 .. +3.4E+38
    char text[32] = "Some text.";
```

In next sl. 5.1 we can find previous variables stored in controllers' RAM memory (upper window of sl. 5.1). In the lover left corner of the sl. 5.1 we can find printed memory addresses of these variables. In the memory table we can first notice `text` variable from the address `0x0100` within next 32 bytes (2 rows of the memory table). Next 4 bytes are occupied by `pi_value` variable, at the memory address `0x0124` `logicVal` is stored (1 byte), following with character letter A stored in variable named `letterA` at the address `0x0125` with the HEX value of `0x41`. At the memory address `0x0126` we can find `smalVal` variable which storing the value 123 (DEC) or `0x7B` in HEX. The last 2 bytes are occupied by the integer variable named `value` where the nuber 1234 is stored or in HEX 0x04 0xD2.




**Slika 5.1:** Table of values stroed in RAM memory of Arduino UNO controller.

#### 5.5.3.3 Measuring Time with programming loops

The easiest way to measure time is to simply count the number of loop's executions. And if we know how long is one execution of the loop - we can easily determine the time lapsed for the whole process.

Example:

```
int t = 0;
while (t<10){
t++;
delay(100);
}
```

In the previous example the `while` loop is executed 10 times (t = [0 .. 9]), since each execution of the loop last 100 ms (determined by `delay(100);`) the whole `while` loop last 1 s.

#### 5.5.3.4 Time measuring with Timers

More proper way of measuring the time is by using the timer's values. More on that can be read here.

Example:

```
unsigned long start_time;
unsigned long stop_time;
start_time = millis();
// time measured process goes here
// ...
stop_time = millis();
unsigned long duration = stop_time - start_time;
```

Where the `duration` is time measured in milliseconds.

#### 5.5.3.5 Structures

n C++, a struct is a user-defined data type that groups together a collection of variables. It is similar to a class in that it can contain variables and functions, but there are a few key differences between the two.

One of the main differences between a struct and a class in C++ is that structs have public members by default, while classes have private members by default. This means that, by default, all members of a struct can be accessed directly from outside the struct, while members of a class can only be accessed through its member functions.

Another difference is that structs are often used for small, simple data structures that do not require the encapsulation and data hiding features provided by classes. Structs are commonly used for situations where you simply want to group together related data, such as representing a point in two-dimensional space, a date, or a color.

Here is an example of a simple struct in C++:

```
struct Point {
    int x;
    int y;
};
```

This struct defines a new type called Point, which contains two variables of type int, x and y, representing the coordinates of a point in a two-dimensional space.

```
Point p1;
p1.x = 3;
p1.y = 4;
```

In this example, we create a variable p1 of type Point and assign values to its members x and y.

It's also worth noting that C++ has also a keyword class which is semantically equivalent to struct except for the default access level of its members.

#### 5.5.3.6 Enumeration

In C++, an **enum** (short for "enumeration") is a user-defined data type that consists of a set of named values. It is used to create a new type with a fixed set of possible values, which can make your code more readable and maintainable.

Here's an example of an enumeration that could be used in a mobile robot program to represent the different states of the robot:

```
enum class RobotMoves{
    FORWARD,
    BACKWARD,
    MOVE_LEFT,
    MOVE_RIGHT,
    STOP
};
```

You can use this enumeration in the robot's control loop to check and update the current state of the robot:

```
1   RobotMoves currentRobotState = RobotMoves::STOP;
2
3   while (true) {
4       // Some other logic here
5       // ...
6
7       // Sampling the sensors based on the state of the robot
8       switch (currentRobotState){
9         case RobotMoves::FORWARD    : checkFrontSensors(); break;
10        case RobotMoves::BACKWARD   : checkBackSensors(); break;
11        case RobotMoves::MOVE_LEFT  : checkLeftSensors(); break;
12        case RobotMoves::MOVE_RIGHT : checkRightSensors(); break;
13        default: //nothing to do...
14      }
15  }
```

This way, it's clear and easy to understand the current state of the robot, and it can also help to implement logic and different behaviors for each state. It's also easy to add or remove states in the future if needed, without having to modify the code in many different places.

### 5.5.4 Issues:

#### 5.5.4.1 <++>

<++>

## 5.6 Flow control

Before we dive into S-R-A Loop lets take a first look to IF-statemen. `IF-statement` allows us to execute some code when the condition is **`true`**. Such navigation of execution of the code is essential in programming and as such one of the fundamental structures of the field. Lets test test the bumper push-button-switch if it is working properly…

### 5.6.1 Tasks:

1. Construct the bumper of the robot with push-button-switch as is shown in this video instructions.
2. And connect the push-button-switch (PBSW) terminals with module RobDuino according to tbl. 5.1:

**Table 5.1:** Connection of push-button-switch to the Robduino module.

| PBSW con. | RobDuino connectors |
|---|---|
| No. 1 | A0 |
| No. 2 | GND |
| No. 3 | +5V |

3. Test the push-button-switch in the bumper with next prog. 5.7:

**Program 5.7:** Conditional Statements.

```
const int BUMPER_PIN          = A0;
const int TEST_BUMPER_LED_PIN = 3;
void setup()
{
  pinMode(BUMPER_PIN, INPUT);
  pinMode(TEST_BUMPER_LED_PIN, OUTPUT);
}

void loop()
{
  bool bumperIsPressed = digitalRead(BUMPER_PIN);
  if ( bumperIsPressed ) digitalWrite(TEST_BUMPER_LED_PIN, HIGH);
}
```

2. Then... complete the program to turn OFF the LED when the bumper is not touching anything.
3. Next... Change IF statements into single one IF-THEN-ELSE statement.

### 5.6.2 Questions:

1. Check if the LED on the output terminal D3 is ON when the bumper is pressed.
2. Measure the voltage potencial at the terminal A0 when the bumper is pressed.
3. Explain when the curly braces `{}` are necessary in the if-statement.

### 5.6.3 Summary:

#### 5.6.3.1 Conditional statements

Conditional statements in C++ allow you to execute different blocks of code based on whether a condition is true or false. There are several different types of conditional statements in C++, including

if, if-else, and switch.

Here is an example of how to use an if statement in C++:

```
int x = 10;

if (x > 5) {
  // This code will be executed if x is greater than 5
  printf("x is greater than 5");
}
```

In this example, the if statement checks whether the value of x is greater than 5. If it is, the code block inside the curly braces will be executed. If it is not, the code block will be skipped.

Here is an example of how to use an if-else statement in C++:

```
int x = 10;

if (x > 5) {
  // This code will be executed if x is greater than 5
  printf("x is greater than 5");
} else {
  // This code will be executed if x is not greater than 5
  printf("x is not greater than 5");
}
```

In this example, the if-else statement first checks whether the value of x is greater than 5. If it is, the code block inside the first set of curly braces will be executed. If it is not, the code block inside the second set of curly braces will be executed.

Here is an example of how to use a switch statement in C++:

```
int x = 2;

switch (x) {
  case 1: printf("x is 1"); break;
  case 2: printf("x is 2"); break;
  case 3: printf("x is 3"); break;
  default: printf("x is something else"); break;
}
```

In this example, the switch statement checks the value of x and executes the code block corresponding to the first case label that matches the value. The break statements are used to exit the switch statement once a match is found. If no match is found, the code block for the default label is executed.

#### 5.6.3.2 IF Statement

If statement can be written in several forms. The easiest one is:

```
   if (value_one) statement1;
```

In this case the variable named `value_one` can hold some numerical number. If `value_one` is **`true`** or greater than `0` the program will execute `statement1`. But this simple example is not used so often due its simplicity. We rather use it in this form:

```
   if ( value_one == value_two ){
    statement1;
    statement2;
   }
```

In this case `value_one` can be any number and the `statement1` and `statement2` will be executed if the `value_one` will be equal to `value_two`. These command can be expanded into IF-ELSE form:

```
   if ( value_one == value_two ){
    statement1;
    statement2;
   }else{
    statement3;
   }
```

#### 5.6.3.3 Condition operators

Also other logical condition operators can be used:

- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- Equal to `a == b`
- Not Equal to: `a != b`

### 5.6.4 Issues:

#### 5.6.4.1 <++>

<++>

## 5.7 Arrays and strings

## 5.8 Pointers and references

## 5.9 Classes and objects

## 5.10 Exception handling

## 5.11 Input and output

## 5.12 Debugging and testing

## 5.13 Advanced topics threading memory management templates

### 5.13.1 Bit-field variable type

Variable bit fields are a specific type of data structure in C++ that allows a user to store multiple bit-sized values within a single variable. This can be useful for storing several different values in the same memory space or for compressing data. An example of a variable bit field in C++ is shown below:

```
struct example {
 unsigned int value1 : 4; // Use 4 bits
 unsigned int value2 : 8; // Use 8 bits
 unsigned int value3 : 12; // Use 12 bits
} myStruct;
```

In this example, we have defined a structure called ‘example’ which contains three members - ‘value1’, ‘value2’, and ‘value3’. Each of these members has been defined as a variable bit field using the ‘unsigned int’ data type and the ‘:’ syntax, which allows us to specify the number of bits that each member should use. In this case, ‘value1’ will use 4 bits, ‘value2’ will use 8 bits, and ‘value3’ will use 12 bits. To access these values, we can use the members of the structure, for example, ‘myStruct.value2’.

```
struct adc4 {
  unsigned int value1 : 10;
  unsigned int value2 : 10;
  unsigned int value3 : 10;
  unsigned int value4 : 10;
};

unsigned int adc_val[40];           //40 values
adc4 myAdc[10];                     //40 values

void setup() {
  Serial.begin(9600);
  Serial.println(sizeof(adc_val));  //print 80
  Serial.println(sizeof(myAdc));    //print 50
}

void loop() {

}
```

# 6 INPUTS AND SRA LOOP IN ROBOTICS

Robotics is a field of engineering that involves the design and operation of robotic systems. One of the most fundamental principles underlying robotic systems is the S-R-A (sensor-response-actuation) loop. This concept is at the heart of all robotic systems and is essential for understanding the behavior of robots.

The S-R-A loop involves a robot continually sensing its environment, interpreting the data, and then taking some action in response. In other words, the robot is constantly interpreting sensory input and responding with a motor action. It is a continuous cycle of sensing, reasoning, and acting.

The sensing component of the S-R-A loop generally involves the use of sensors such as cameras, ultrasound, or infrared sensors. These sensors detect the robot's surroundings and provide the robot with the data necessary to make decisions. The response component of the loop involves the robot using its artificial intelligence to interpret the data and make decisions. This decision-making process is what gives robots the ability to respond to their environment.

The actuation component of the S-R-A loop is where the robot takes action. This action may involve a physical movement, such as walking, or it may involve activating a motor to perform a task, such as picking up an object.

The S-R-A loop is the basic building block of any robotic system. All robots use this concept as it is essential for a robot to be able to interact with its environment. Without it, robots would not be able to make decisions or take action. This concept is also important for enabling robots to learn, as it allows them to continually increase their knowledge and abilities.

Overall, the S-R-A loop is the cornerstone of robotics. It is essential for robots to be able to interact with their environment and learn from it. Without the S-R-A loop, robots would be unable to take any action or make decisions. It is an integral part of any robotic system.

From the S-R-A loop, lets start at the very beggining of the loop - at reading input signals by emphasizing the importance of received input signal. In other words, it is critical that the system be able to detect and interpret input signals in order to produce the appropriate responses. Once these input signals are received, they must be accurately processed and acted upon. This is the primary task of the S-R-A loop, and is the basis for any successful input processing system.

To read an input signal on an Arduino, you can use one of the digital input pins or one of the analog

input pins. Digital input pins can only read two states: high (5 volts) or low (0 volts). They are often used to read switches or buttons, or to detect the presence or absence of a signal.

To read a digital input signal on an Arduino, you can use the digitalRead function, which takes a pin number as an argument and returns either HIGH or LOW. For example, to read the state of digital pin 2, you could use the following code:

```
int pin = 2;
int state = digitalRead(pin);
```

Analog input pins, on the other hand, can read a range of voltage levels, from 0 to 5 volts. They are often used to read sensors that output an analog signal, such as a temperature sensor or a potentiometer.

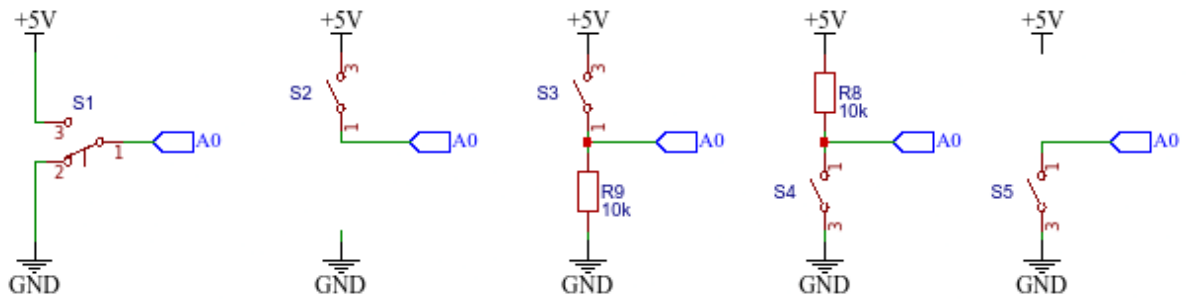
To read an analog input signal on an Arduino, you can use the analogRead function, which takes a pin number as an argument and returns a value between 0 and 1023, corresponding to the voltage level on the pin. For example, to read the voltage on analog pin 0, you could use the following code:

```
int pin = 0;
int value = analogRead(pin);
```

## 6.1 Digital input

Digital inputs can only measure 2 different values. As such they are binary inputs and it's values are represented as logical `0` and `1` or in other words **`false`** and **`true`** or `LOW` and `HIGH`. However from electrical point of view those values are basically different voltage potencials. Usually potencial `0 V` is presented as logical `0` and potencial `+5 V` is indicated as logical `1`. Digital inputs are often used for detecting state of switches, board keys and push buttons…

Lets go back to fundamentals of digital inputs and explore some options we have to connect a push-button-switch.




**Slika 6.1:** Different options of wireing the bush-button-switch.

### 6.1.1 Tasks:

1. Connect the push-button-switch according to first diagram on sl. 6.1 and test the program prog. 6.1

**Program 6.1:** Digital Input.

```
const int BUMPER_PIN = A0;
void setup()
{
 pinMode(BUMPER_PIN, INPUT);
}

void loop()
{
  bool bumperIsPressed = digitalRead(BUMPER_PIN);
 if ( bumperIsPressed ) digitalWrite(3, HIGH);else digitalWrite(3, LOW);
}
```

2. Try to connect the bush-button-switch according to second diagran on sl. 6.1

**Table 6.1:** Connection of push-button-switch with only 2 terminals.

| PBSW con. | RobDuino connectors |
|---|---|
| No. 1 | A0 |
| No. 2 | not connected |
| No. 3 | +5V |

Try to understand why this setup is not working. And test all other options in sl. 6.1

3. Solve the problem by constructing a `voltage divider` with `pull-down` resistor (third diagran on sl. 6.1).
4. Try to understand how the voltage potencial is spread among the components in electrical loop and how we can calculate this by using 2nd Kirchhoff's Rule.
5. Change the setup of PBSW and resistor to a `pull-up` setup (fourth diagran on sl. 6.1). What is changed?
6. Enable internal `pull-up` resistor (and remove external one - fifth diagran on sl. 6.1).

### 6.1.2 Questions:

1. Measure the voltage potencial on pin A0 where the bumper is in ether position.
2. Why the setup is not working properly if we connect the PBSW only to +5V voltage potencial?
3. Draw a schematic circuit of the bush-button-switch connected to controller.
4. What is determined by 2nd Kirshhoff's Rule?
5. How can we wnable `pull-up` resistor?

### 6.1.3 Summary:

#### 6.1.3.1 2nd Kirshhoff's Rule

Kirchhoffs Voltage Rule states that `in any closed loop network, the total voltage around the loop is equal to the sum of all the voltage drops within the same loop` which is also equal to zero. In other words the algebraic sum of all voltages within the loop must be equal to zero. This idea by Kirchhoff is known as the Conservation of Energy.

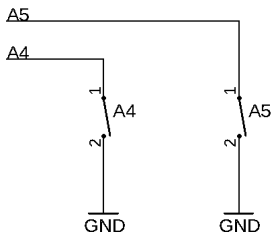
### 6.1.4 Issues:

#### 6.1.4.1 <++>

<++>

## 6.2 Pull-up resistors on digital input

On the module RobDuino we can find two "on-board push button switches". Wiring of this switches is presented in sl. 6.2, where can we noticed that both switches are connected to ground voltage potential.




**Slika 6.2:** Wiring of on-board switches.

To properly use this on-board push-button switches we must enable the `pull-up` resistors of A4 and A5 input of microcontroller.

### 6.2.1 Tasks:

1. Configure pins `A4` and `A5` as inputs with `pull-up` resistor.
2. At the end of the `setup()` function add the **`while`**`-loop` which will delay the execution of the program until we press the `A4` key - acting as a "START BUTTON".
3. Use the `A5` key to stop the robot and terminate the execution of the program.

**Program 6.2:** Pull Up Resistors on Digital Input.

```
#include "RobotMovingFunctions.h"
const int KEY_A4 = A4;
const int KEY_A5 = A5;

void setup()
{
  setIOpins();
  pinMode(KEY_A4, INPUT_PULLUP);
  // KEY_A5 setup here...
}

void loop()
{
  moveForward();
  //to-do: the key reading
  bool stopTheRobotKey = 0;
  if (stopTheRobotKey == 1)
  {
    stopTheRobot();
    exit(0);        //terminate the program
  }
}
```

### 6.2.2 Questions:

1. What is the programming instruction of reading the value form digital input?
2. Which values can be assigned to **`bool`** type variable?
3. Explain the programming instruction `exit(0)`.

### 6.2.3 Summary:

#### 6.2.3.1 <++>

<++>

### 6.2.4 Issues:

#### 6.2.4.1 <++>

<++>

## 6.3 S-R-A loop

S-R-A loop is repeating process where:

1. Seasoning,
2. Reasoning and
3. Acting

is involved during the procedure of controlling the robot. This is the most important part of software in robotics. Remember the `autonomous` control is `ability to perform intended tasks based on current state` **`and`** `sensing, without human intervention`.

The S-R-A loop is a common design pattern in robotics. It refers to the process of using sensors to gather information about the environment, processing the information to determine an appropriate response, and then executing the response using actuators.

Here is an pseudo example of how the S-R-A loop could be implemented in C++:

```
   while (true) {
     // 1. Sense the environment using sensors
     sensor_data = gatherSensorData();

     // 2. Process the sensor data to determine an appropriate response
     response = processSensorData(sensor_data);

     // 3. Execute the response using actuators
     executeResponse(response);
   }
```

In this example, the `gatherSensorData` function is used to gather data from the robot's sensors, the `processSensorData` function is used to determine an appropriate response based on the sensor data,

and the `executeResponse` function is used to execute the response using the robot's actuators. The loop is executed continuously, allowing the robot to constantly sense and respond to its environment.

### 6.3.1 Tasks:

1. Using the S-R-A loop technique you should write the program in particular order:
    1. Check the sensor. IF the bumper ...
    2. ... Is pressed the robot has to stop/go back/turn.
    3. ... Is not pressed the robot can drive forward.

    Test the prog. 6.3 and **find out why the robot does not stop.** (Such mistake is quite often - can you fix it):

**Program 6.3:** SRA Loop.

```
#include "RobotMovingFunctions.h"
const int BUMPER_PIN = A0;
void setup()
{
  setIOpins();
  pinMode(BUMPER_PIN, INPUT);

  bool bumperIsPressed = digitalRead(BUMPER_PIN);
  if ( bumperIsPressed )
  {
    stopTheRobot();
  }
  else
  {
    moveForward();
  }
}
void loop()
{
}
```

2. Hint for fixing the prog. 6.3: *S-R-A must be a loop function!*
3. Write a program to drive the robot around the class and avoid the obstacles.

### 6.3.2 Questions:

1. What for `S-A-R loop` stands for?
2. Mark all three basic S-A-R processes in previous code example.

3. Can the `line 7` of the prog. 6.3 be written outside of `loop()` function? What would happened if so?

### 6.3.3 Summary:

#### 6.3.3.1 <++>

<++>

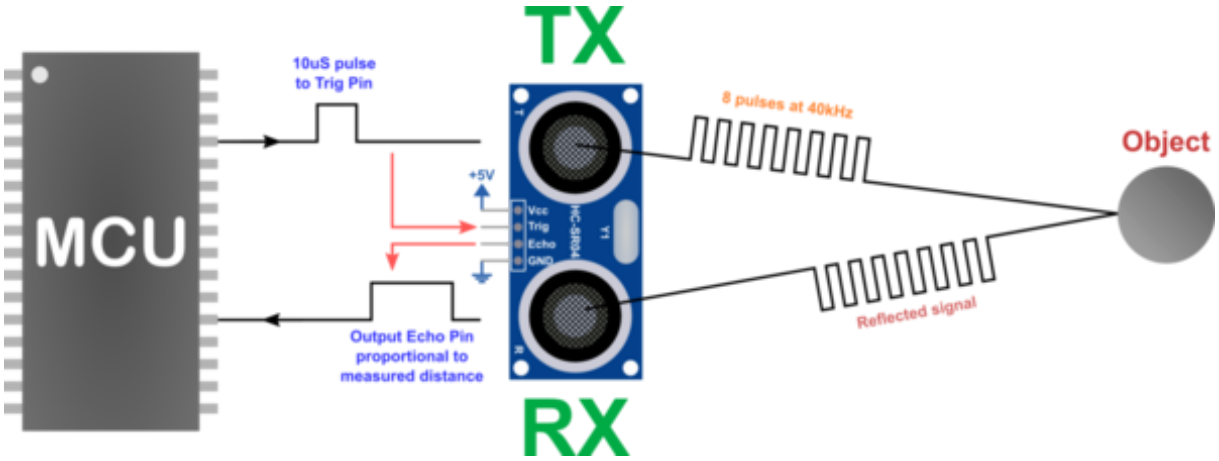
### 6.3.4 Issues:

#### 6.3.4.1 <++>

<++>

## 6.4 Pulse width as digital input

Digital input can also be used to transferee other data. One way is to modulate the data into pulse duration e.g. longer the duration of the pulse, bigger the value. This modulation of data is called **Pulse-width modulatio** or **PWM**. Such an example is ultrasonic distance sensor. Where the distance is hidden in the time duration that sound needed of travel the distance from source to object and back as presented in sl. 6.3.




**Slika 6.3:** How Ultrasonic sensor works.

Since the speed of sound in air is constant ($v_s = 340m/s$) we can easily calculate the distance according to en. 6.1.

$$distance = \frac{1}{2} v_s t_{duration} \tag{6.1}$$

### 6.4.1 Tasks:

1. Connect the ultrasonic distance sensor to module Robduino according to tbl. 6.2

**Table 6.2:** Connestion of ultrasonic distance sensor.

| HC-SR04 pins | RobDuino pins |
|---|---|
| +5V | +5V |
| Trigg. | A0 |
| Echo | A1 |
| GND | GND |

2. Test next program if you get reasonable data of time duration in `Serial` window.

**Program 6.4:** PWM as Digital Input.

```
const char TRIGGER_PIN  = A0;
const char ECHO_PIN     = A1;

void setup()
{
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  Serial.begin(9600);
}

int getPulseWidth_us()
{
  digitalWrite(TRIGGER_PIN,HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN,LOW);
  return pulseIn(ECHO_PIN, HIGH);
}

float getDistance_cm()
{
  // do distance calculation here...
  return 0
}
void loop()
{
  float distance_cm = getDistance_cm();
  int duration_us = getPulseWidth_us();
  Serial.println(duration_us);
  delay(2000);
}
```

3. Add needed code in function `getDistance_cm()` to calculate the distance in cm. Also change the `Serial.println(duration_us)` program line to output `distance_cm` value.

### 6.4.2 Questions:

1. What is PWM?
2. How are PWM data presented in digital signal?
3. What voltage is used to transmit PWM values?

### 6.4.3 Summary:

#### 6.4.3.1 <++>

### 6.4.4 Issues:

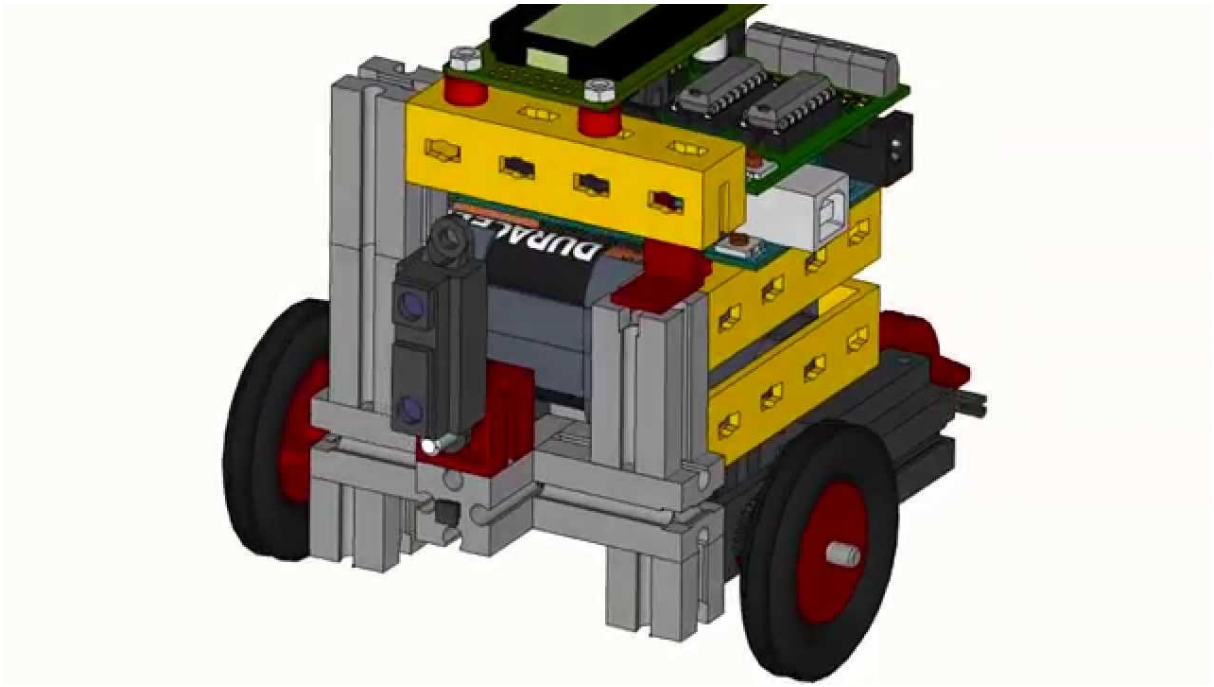
#### 6.4.4.1 <++>

## 6.5 Analog input

In general, controllers are equipped with `Analog to Digital Converters` or short `ADC`. This internal devices converts voltage potencial into numeric value which can be further used by written program. This is also the case in Arduino UNO converter by the function `analogRead(pin_number)`. In this case the voltage range $[0.0\ V .. +5.0\ V]$ is converted in to range of numbers $[0..1024]$.

### 6.5.1 Tasks:

1. Unmount robot's bumper and all connections to the switch.
2. Equip the robot with distance sensor according to video and scheme (see sl. 6.4).



**Slika 6.4:** Mounting possition of analog distance sensor.

3. Try next prog. 6.5 and check the output of distance sensor in Serial monitor.

**Program 6.5:** Analog Input.

```
const int DIST_SEN_PIN = A0;
void setup()
{
  pinMode(DIST_SEN_PIN, INPUT);
 Serial.begin(9600);
}

void loop()
{
  int adc_value = analogRead(DIST_SEN_PIN);
 Serial.println(adc_value);
 delay(1000);
}
```

4. Convert the `analog_sensor_value` into `input_voltage` and measure the input voltage potencial with volt-meter. The formula for conversion can be programmed as:

```
float input_voltage = 5.0/1024 * adc_value;
```

5. From the datasheet for the distance sensor try to code the function for measuring the distance in cm. According to documentation there is almost linear trend between output voltage and $distance^{-1}$. Thus we can get good result with en. 6.2.

$$distance^{-1}[cm] = 0.045 V_{out} \tag{6.2}$$

Next example can be your guide to code the function.

```
float getDistance_cm()
{
  int adc_value = analogRead(DIST_SEN_PIN);
  float input_voltage = 5.0/1024 * adc_value;
  float distance = 1/(0.045 * input_voltage);
  return distance;
}
```

### 6.5.2 Questions:

1. What kind of values do you getting from the reading of the distance sensor with the function `analogRead(A0)`?
2. Find the reasonable value where you should stop the robot.
3. Measure the voltage potencial of the sensor's output.

### 6.5.3 Summary:

#### 6.5.3.1 Analog to digital converter - ADC

ADC is an electronic sistem that converts analog signal (voltage) to a digitalized values. In our particular case the range of an analog voltage from 0V to 5V is converted to range of numbers from 0 to 1024.

### 6.5.4 Issues:

#### 6.5.4.1 <++>

<++>

## 6.6 Avoiding obstacles

### 6.6.1 Tasks:

Write the program to drive the robot around the class and avoid the obstacles.

1. Check the value of distance sensor. If the distance is greater than …
2. ... the robot can drive forward.
3. ...else ... the robot must to stop/go back/turn.

**Program 6.6:** Avoiding Obstacles.

```
#include "RobotMovingFunctions.h"
const int DIST_SEN_PIN   = A0;
const int DISTANCE_LIMIT = 20;
void setup()
{
  setIOpins();
  pinMode(DIST_SEN_PIN,INPUT);
}
float getDistance_cm()
{
  int adc_value  = analogRead(DIST_SEN_PIN);
  float distance = 1/(0.045 * 5.0/1024 * adc_value);
  return distance;
}
void loop()
{
  if ( getDistance_cm() > DISTANCE_LIMIT )
  {
    moveForward();
  }
  else
  {
    stopTheRobot();
  }
}
```

### 6.6.2 Questions:

1. What are the values of the distance sensor (use `Serial.println(distance)` to verify)?
2. Robot stil hits the obstacles that are not in view angle of the distance sensor. Write and use new function for moving the robot forward more carefully.

### 6.6.3 Summary:

#### 6.6.3.1 Moving the robot and checking the sensor simultaneously

The main important proces in robotics is S-R-A loop. This process is used in different situations and many times. One can be where we are moving the robot forward and at the same time observing the sensors value with the intention to stop it when the specific condition is met.

```
   void goForwardCarefully()
   {
      for (int i = 0; i < 10; i++)
      {
         robotLeft();delay(50);
         if (getDistance_cm() < DISTANCE_LIMIT) brake;
      }

      for (int i = 0; i < 10; i++)
      {
         robotRight();delay(50);
         if (getDistance_cm() < DISTANCE_LIMIT) brake;
      }
   }
```

<++>

### 6.6.4 Issues:

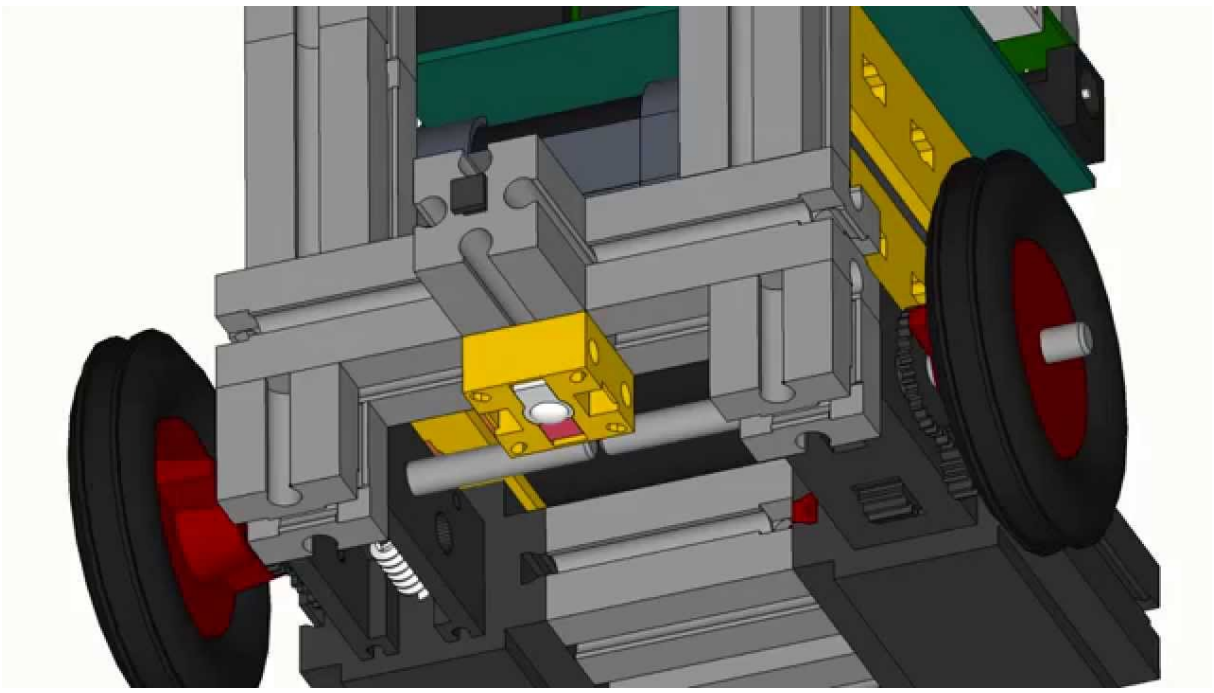
#### 6.6.4.1 <++>

<++>

## 6.7 Light sensor

### 6.7.1 Tasks:

1. Construct the light sensor according to video and scheme. Add also the light bulb which will help to lightning the area beneath the robot.



**Slika 6.5:** Mounting a light sensor.

1. To test the light sensor and light bulb test this example code and check the reported serial data.

**Program 6.7:** Ligth Sensor.

```
const int LIGHT_SENSOR_PIN = A0;

void setup()
{
  pinMode(LIGHT_SENSOR_PIN , INPUT);
  Serial.begin(9600);
}

void loop()
{
  int light_sensor_value = analogRead(LIGHT_SENSOR_PIN );
  Serial.println(light_sensor_value );
  delay(200);
}
```

2. Try different resistors (1k, 10k, 100k, 1M) and find out at which the sensitivity of the sensor is greatest.

**Table 6.3:** Testing the sensitivity of the light sensor.

| Resistance | (black) Sensor value | (whithe) Sensor value | Sensor difference |
|---|---|---|---|
| 1 kOhm | | | |
| 10 kOhm | | | |
| 100 kOhm | | | |
| 1 MOhm | | | |

### 6.7.2 Questions:

1. What is the value of the sensor when the robot is over white/black area?
    - ADC value on white:
    - ADC value on black:
2. Calculate the average between those two values.
    - Average is:

### 6.7.3 Summary:

#### 6.7.3.1 Sensors

Sensors are electronic devices which convert physical quantity into electrical quantity (usually voltage). In simplest setup, sensor can be constructed as voltage divider with two resistors - $R_1$ and $R_2$. One of the resistors is resistor with fixed resistance value (eg. $R_1 = 10k\Omega$). The second one is a bit special and it's resistance depends on some physical quantity (e.g. light, temperature, humidity...). When combining those two resistors into such voltage divider the output of the voltage divider can be calculated as:

$$U_{Out} = \frac{R_1}{R_1 + R_2} U_0$$

### 6.7.4 Issues:

#### 6.7.4.1 *Value of the sensor is very small*

If the value of the sensor is less than 100 the resistance of $R_2$ (connected to GND) is to low in comparismant to the resistance of R_1 (connected to +5V).

#### 6.7.4.2 *Value of the sensor is large*

If the value of the sensor is grater than 900 the resistance of $R_2$ (connected to GND) is to hi in comparismant to the resistance of R_1 (connected to +5V).

## 6.8 Line follower

### 6.8.1 Tasks:

1. Write the program to control the robot follow the line ( actually above the edge between black and white area ). Some programming hints you can find in prog. 6.8 :

**Program 6.8:** Line Follower.

```
#include "RobotMovingFunctions.h"
const int LIGHT_SENSOR_PIN = A0;
const int SURFACE_BRIGHTNESS_REFERENCE = 400;

void setup()
{
  setIOpins();
  pinMode(LIGHT_SENSOR_PIN , INPUT);
}

void loop()
{
  int light_sensor_value = analogRead(LIGHT_SENSOR_PIN );
  if ( light_sensor_value < treshold_value )
  {
    // do this if robot is over the black line

  }
  else
  {
    // do this if robot is over white area

  }
}
```

### 6.8.2 Questions:

1. What is the program function to get the `light_sensor_value`?
2. Determine the movements of the robot if the robot is over the black area and if the robot is over the white area.

### 6.8.3 Summary:

#### 6.8.3.1 <++>

<++>

### 6.8.4 Issues:

#### 6.8.4.1 <++>

<++>

# 7 CONTROLLING ACTUATORS

Motors and actuators are essential components of many robotic systems, as they allow robots to move and manipulate their environment. In Arduino robotics, there are several types of motors and actuators that you can use, depending on the specific needs of your application.

Some common types of motors and actuators that you can use with Arduino include:

**DC motors**: These are simple motors that rotate at a constant speed when a DC voltage is applied. They are commonly used to drive wheels or other mechanisms. To control a DC motor with an Arduino, you will need a motor driver, such as an H-bridge, which allows you to control the direction and speed of the motor.

**Stepper motors**: These motors have multiple coils that can be energized in a specific sequence, allowing them to rotate in precise increments. Stepper motors are commonly used in applications that require precise positioning, such as 3D printers or CNC machines. To control a stepper motor with an Arduino, you will need a stepper motor driver, such as a ULN2003 or L298N.

**Servo motors**: These motors have built-in feedback control and can rotate to a specific angle. They are commonly used to control the position of a mechanism, such as a robotic arm or a camera. To control a servo motor with an Arduino, you can use the Servo library and the write function, which takes an angle as an argument.

**Linear actuators**: These are motors that produce linear motion, rather than rotary motion. They are commonly used to move mechanisms or lift loads. To control a linear actuator with an Arduino, you will need a motor driver, such as an H-bridge, and you can use the analogWrite function to control the speed and direction of the actuator.

## 7.1 DC motor

## 7.2 PWM motor control

There is often the situation where the power of the motors must be controlled. One convenient way to do this is that we don't power the motor full time, but we can turn off the motor for short period of

time. For an example we can turn the motor on for 1 ms and turn it off for 1 ms. In this case the motor will not get 100% of power, but the motor's average power will be 50%.

Since we are changing the pulse width of logical 1 with the respect to width of logical 0, this technique is called `pulse width modulation` or shorter `PWM`.

This modulated output is controlled by the `analogWrite(pin, pwm)` function. Modulatio can be performed on pins: 3, 5 and 6 of the RobDuino modul. The value of `pwm` parameter can be on a scale of 0 - 255., where 0 is 0% and 255 is 100% of electrical power served.

### 7.2.1 Tasks:

1. Write new functions for driving the robot left and right with reduced power of the motors:

   - `moveLeftPWM();`
   - `moveRightPWM();`

   In one case you will might find yourself in trouble of controlling the power of the motor since both pins are not able to perform `PWM` output. In this case you can remember that the motor's power is 0 W also if both pins are in state of logical `1`.

   An example of reducing power of both motors in function `moveForwardPWM()` is here:

```
1  void robotForwardPWM()
2  {
3    digitalWrite( LEFT_MOTOR_PIN_1, LOW);
4    analogWrite(  LEFT_MOTOR_PIN_2, 150);
5    digitalWrite( RIGHT_MOTOR_PIN_1, LOW);
6    analogWrite(  RIGHT_MOTOR_PIN_2, 150);
7  }
```

Similar to this function you can write other functions to.

2. Change the functions `moveLeft()` and `moveRight()` in S-R-A loop with new ones with less power on motors.

**Program 7.1:** PWM motor control.

```
#include "RobotMovingFunctions.h"
const int LIGHT_SENSOR_PIN = A0;
const int SURFACE_BRIGHTNESS_REFERENCE = 400;

void setup()
{
  setIOpins();
  pinMode(LIGHT_SENSOR_PIN , INPUT);
}

void loop()
{
  int light_sensor_value = analogRead(LIGHT_SENSOR_PIN );
  if ( light_sensor_value < SURFACE_BRIGHTNESS_REFERENCE ){
      moveLeft();
  } else {
      moveRight();
  }
  delay(10);
}
```

3. Also add `analogWrite(LEFT_MOTOR_PIN_A, 0);` to function `stopTheRobot()` to stop the `PWM` control of the motor. And do similar code for the `right motor`.

4. Add a parameter `PWM_value` to each function to set the `duty cicle` of the controlled output.

   - `moveLeftPWM(int PWM_value)`
   - `moveRightPWM(int PWM_value)`

5. Save `moveRightPWM(int PWM_value)` and `moveLeftPWM(int PWM_value)` functions into header file `RobotMovingFunctions.h`

### 7.2.2 Questions:

1. How can we control the average power of the motor?
2. How can we control the average power of the motor in both directions if we are not able to control `PWM` both output pins of the motor?
3. Explain the purpose of programming function `analogWrite(pin, pwm)`.
4. Explain the meaning of the `pin` and `pwm` parameters in function `analogWrite`.

### 7.2.3 Summary:

#### 7.2.3.1 <++>

<++>

### 7.2.4 Issues:

#### 7.2.4.1 <++>

<++>

## 7.3 Servo motor

## 7.4 Stepper motor

Stepper motors are a type of electric motor that can precisely control a rotating shaft's angular position. They are the most commonly used type of motor in motion control applications. A stepper motor works by converting electrical pulses into mechanical shaft rotations, which can be used to move a device or position an object. Stepper motors produce precise, smooth, and repeatable motion and can be used in a variety of robotic applications. They are commonly used for positioning CNC machines, 3D printers, pick-and-place systems, and other robotic applications. Stepper motors are available in a variety of sizes and configurations, and can be used with a variety of drive systems and controllers.

In general we differ two types of Stepper motors (regarding the coil wireing):

1. Bipolar Stepper Motor - This type of stepper motor has two sets of coils, each with a single winding per phase. The coils are wired in series or in parallel depending on the application. Each winding in the motor is energized, then de-energized in order to make the motor rotate.
2. Unipolar Stepper Motor - This type of stepper motor has two sets of coils, each with multiple windings per phase. The coils are wired in series or in parallel depending on the application. Only one winding in the motor is energized at a time to make the motor rotate."

### 7.4.1 Task

Stepper motors are used in many Arduino projects to control motion, such as turning a wheel or a motor shaft. By applying pulse-width modulation (PWM) signals, the Arduino can control the speed

and direction of the motor. Below is an example of Arduino code that can be used to control a stepper motor:

```
    //Define the pins to be used for the stepper motor
    #define STEPPER_PIN_1 8
    #define STEPPER_PIN_2 9
    #define STEPPER_PIN_3 10
    #define STEPPER_PIN_4 11

    //Define the delay between steps in milliseconds
    #define STEP_DELAY 10

    //Create an array of the pins to be used
    int pins[] = {STEPPER_PIN_1,STEPPER_PIN_2,STEPPER_PIN_3,STEPPER_PIN_4};

    //Initialize the stepper motor
    void setup()
    {
     //Set each pin as an output
     for(int i=0;i<4;i++)
     {
     pinMode(pins[i], OUTPUT);
     }
    }

    //Control the stepper motor
    void loop()
    {
     //Rotate clockwise
     for(int i=0;i<4;i++)
     {
     digitalWrite(pins[i],HIGH);
     delay(STEP_DELAY);
     }
     //Rotate counter-clockwise
     for(int i=3;i>=0;i--)
     {
     digitalWrite(pins[i],HIGH);
     delay(STEP_DELAY);
     }
    }"

    ---
    grand_parent: Book
    parent: Barier Gate
    title: I2C LCD
    nav_order: 4
    ---

    ## LCD(I2C)

    ### Tasks:

    1. Priključite LCD na I2C vodilo kot prikazuje

    ![Povezava LCD na I2C vodilo krmilnika.](./slike/I2C_LCD.png){#fig:
       test_I2C_LCD}
```



```
    2. Priskrbite si knjižnico `LiquidCristal-I2C` iz naslova:
    https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/
    3. Knjižnico dodajte v Arduino IDE okolje tako, da dodate `ZIP` datoteko
        v :
    `Sketch >> Include Library >> Add .ZIP Library`
    3. V VSC in PlatformIO vtičniku si lahko knjižnico naložite tako, da v
```

Če niste prepričani kateri i2c naslov uporablja naprava na LCD-ju le tega lahko preverite s programom `I2C scanner` (https://playground.arduino.cc/Main/I2cScanner/). Običajno I2C LCD-ji, ki jih naredijo kitajski proizvajalci uporabljajo I2C naslov `0x27` , `0x3F` ali manj pogosto `0x38`.

### 7.4.2 Questions:

1. <++>
2. <++>

[ Visual instructions. ]

### 7.4.3 Summary:

#### 7.4.3.1 <++>

<++>

### 7.4.4 Issues:

#### 7.4.4.1 <++>

<++>

# 8 FUNDAMENTAL TASKS IN ROBOTICS

## 8.1 Timers and time measurement

Timers and time measurement are important concepts in Arduino programming, as they allow you to perform tasks at specific intervals, measure elapsed time, or synchronize events. The Arduino has several built-in timer modules that you can use in your programs.

Here are some common ways to use timers and measure time in Arduino:

**delay() function**: This function causes the program to pause for a specific number of milliseconds. For example, the following code will cause the LED on digital pin 13 to blink every second:

```
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);  // wait for 1 second
  digitalWrite(13, LOW);
  delay(1000);  // wait for 1 second
}
```

**millis() function**: This function returns the number of milliseconds that have elapsed since the Arduino was powered on or reset. You can use this function to measure elapsed time or to trigger events at specific intervals. For example, the following code will turn the LED on and off every 5 seconds:

```
unsigned long previous_time = 0;  // store the previous time

void loop() {
  unsigned long current_time = millis();  // get the current time
  if (current_time - previous_time >= 5000) {  // check if 5 seconds
      have passed
    digitalWrite(13, !digitalRead(13));  // toggle the LED
    previous_time = current_time;  // update the previous time
  }
}
```

**Hardware timers**: The Arduino has several hardware timers that can be used to generate periodic interrupts. You can use these timers to trigger events at specific intervals without using the delay() function. For example, the following code uses Timer 1 to toggle the LED on and off every second:

```
1   void setup() {
2     // set up Timer 1 to generate an interrupt every 1 second
3     cli();  // disable global interrupts
4     TCCR1A = 0;  // set Timer 1 to normal mode
```

## 8.2 Move to reference position

## 8.3 Navigation and mapping

### 8.3.1 Tasks:

1. Stop the robot when it reaches the end of line.
2. Detecting the end of line can be done by measuring the time that robot spend over the black and white area. E.g. if the robot is driving along the line - the time spent over black and time spent over white area will be quite the same. When line ends the robot will not detect the black area soon and the time spent over white area will increase significantly - and that is the trigger for detecting the end of line.
3. Advanced: Make a function to align (move) the robot back to the line.

### 8.3.2 Questions:

1. How can we store a data to the controller's memory?
2. How can we measure time in programming loops?
3. What is the purpose of the prog. instr. exit(0); ?

**Program 8.1:** Edn of Line Detection.

```
#include "RobotMovingFunctions.h"
const int LIGHT_SENSOR_PIN = A0;
const int SURFACE_BRIGHTNESS_REFERENCE = 400;
int time_on_black = 0;
int time_on_white = 0;

void setup()
{
  setIOpins();
  pinMode(LIGHT_SENSOR_PIN , INPUT);
}
void loop()
{
  int light_sensor_value = analogRead(LIGHT_SENSOR_PIN );
  if ( light_sensor_value < SURFACE_BRIGHTNESS_REFERENCE )
  {
    // BLACK area
    moveLeft();
    time_on_white = 0;  // reset time on white
    time_on_black++;    // meas. time on black
    delay(100);
  }
  else
  {
   // WHITE area
    moveRight();
    // Do similar meas.
    // of time on white
    delay(100) ;
    // If time is signif. longer:
    //        robotStop();exit(0);
  }
}
```

#### 8.3.3 Summary:

##### 8.3.3.1 <++>

#### 8.3.4 Issues:

##### 8.3.4.1 <++>

## 8.4 PID Control

## 8.5 Pick and place operations

## 8.6 Perception and recognition

# 9 ROBOTICS APPLICATIONS

## 9.1 Robotics projects for educational and research applications

## 9.2 Robotics in industry and everyday life

## 9.3 Robotics competitions and challenges

## 9.4 Robotics careers and future opportunities

# 10 ADVANCED ROBOTICS

## 10.1 Robotics in artificial intelligence and machine learning

## 10.2 Robotics in computer vision and image processing

## 10.3 Robotics in natural language processing

## 10.4 Robotics in swarm intelligence and multi-agent systems