

Learn the basics of robotics

"SumoBoy" v 2.0



**ROBOT
NEST**

Learning material and worksheets should be used together with robot prototyping kit “SumoBoy” for learning basics of robotics pack.

Please send any suggestions that has come up while using the pack to:

karlis@robot-nest.com

Follow updates on the learning material at our webpage www.robot-nest.com

Learning material published by RTU

Editors Anita Vēciņa, Lilita Vīksna

Graphic designer Jekaterina Lukina

Photo Elīna Karaseva

CONTENTS

Introduction	5
1. Elements of Electronics	6
1.1. Worksheet. Turning on Light-emitting diode	9
1.2. Worksheet. Switching Light-emitting diode	10
1.3. Worksheet. Series circuit with resistors	11
1.4. Worksheet. Parallel circuit with resistors	12
1.5. Worksheet. Light-emitting diode with capacitor	13
1.6. Worksheet. Series and parallel circuit with capacitor	14
2. Preparing programming environment	16
3. Beginning Programming	20
3.1. Worksheet. Button and light-emitting diode	26
3.2. Worksheet. Staircase lighting	28
3.3. Worksheet. Light-emitting diodes brightness	30
3.4. Worksheet. Light-emitting diodes blinking speed change	32
Chapters Annex	34
4. Semiconductors	37
4.1. Worksheet. Diodes circuit	43
4.2. Worksheet. Light-emitting diodes voltage	45
4.3. Worksheet. RGB Light-emitting diode	46
4.4. Worksheet. Operating RGB Light-emitting diodes with program	48
4.5. Worksheet. Using transistors	50
4.6. Worksheet. Multivibrator	52
4.7. Worksheet. Multivibrator with program	53
5. Sensors	55
5.1. Worksheet. Angle sensor	58
5.2. Worksheet. Temperature sensor	60
5.3. Worksheet. Light sensor	62
5.4. Worksheet. Obstacle and light reflection sensor	64
5.5. Worksheet. Capacitive sensor	66

6. Motors and Their Management	68
6.1. Worksheet. Operating a motor using a program	71
6.2. Worksheet. Regulating motors speed using a program	73
6.3. Worksheet. Servomotors management	75
6.4. Worksheet. Using <i>H</i> bridge	77
7. Mini-sumo robot	79
7.1. Worksheet. Robots light-emitting diodes	80
7.2. Worksheet. Robots buttons	81
7.3. Worksheet. Robots DIP switches	82
7.4. Worksheet. Robots line sensors	84
7.5. Worksheet. Robots distance sensors	86
7.6. Worksheet. Robots motors	88
7.7. Worksheet. Robots screen	90
8. Minisumo competitions	91
8.1. Worksheet. Minisumo settings	92
8.2. Worksheet. Robot that avoids obstacles	93
8.3. Worksheet. Program for not driving out of the ring	94
8.4. Worksheet. Program for sumo battles	98
Annex No1. List of parts and their images	102
Annex No2. Robots input and output devices	104
Annex No3. Robots overall scheme	106
Annex No4. Learn soldering	108
P4.1. Worksheet	110

INTRODUCTION

Learning pack for learning the basics of robotics “SumoBoy” is meant for one school year (Form 7-12, upon schools preference). Classes are held once a week, every lessons length is three academic hours. After learning the basics of robotics students can use this pack for further development of their sumo robot for competitions as well as applying their skills in modern field of robotics. The pack is complemented by curriculum and worksheets for every lesson. Curriculum can be carried out as an individual subject or as extra-curricular lessons.

During the school year students learn about the electronic elements that are needed for constructing a robot, management of electric motor for direct current, processing of data obtained by sensors all while doing more and more complicated tasks. Another set of tasks is meant for learning basics for programming a robot. The “SumoBoy’ robot that is included in the learning pack complies with the regulations for international minisumo robot competitions. Because of this students will be able to participate in national or international competitions defending the name of their school or region.

Overview of the learning pack “SumoBoy”



Aim of the Program

Promote students interest in robotics as one of the most fast-growing field of engineering science; motivate students to connect their further education with engineering science.

Tasks of the Program

Gain the ability to combine the elements of electrical engineering, electronics and computer science to create a robot by using the ingredients included in the learning pack so the robot would be able to accomplish the actions embedded in its software.

For students attending schools for general education the physics program does not include lesson dedicated to learning soldering, which is why this program for the basics of robotics includes a separated lesson to teach it. The worksheets include exercises for learning technical safety and the use of multimeter. Sumo robot competition is held by the end of the school year to motivate students to learn.

Expected results after learning the program:

- Have learned the basics in technical safety and soldering; have skill in using multimeter.
- Have gained understanding of basic elements of electronics and their usage.
- Have gained understanding in the management of electric motor and ability to process sensor data.
- Have learned basics in programing.

The Structure of Learning Material

The learning material is separated in topics which view individual aspect of robot’s structure, for example, elements of electronics, management of electronic motor, etc. Each topic has general description and tasks. All topics are numbered throughout the learning material, but worksheets (independent work) for each topic start with worksheet No1.

Worksheets are made so they could be printed and kept beside as a ‘cheat-sheet’ while doing the task.

1. ELEMENTS OF ELECTRONICS

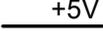
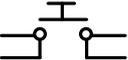
1.1. Aim

The aim of this topic is to introduce elements and possibilities of constructor prototyping plate as well as give general knowledge about elements of electrical circuits.

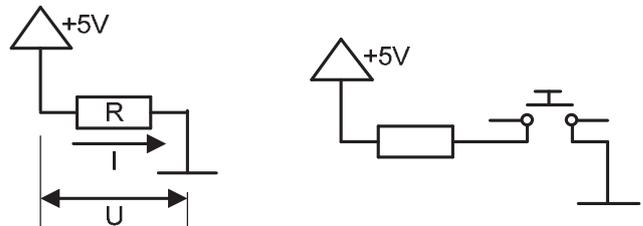
1.2. Theoretical part

vienkārša elektriskā ķēde

A robot is made up of mechanical, electronic and programming part. In this chapter we look at electric part- electric circuits. Every electric circuit has source of energy (in this constructor it is an accumulator), connector (wires) and resistor which consumes power to accomplish an action - do calculations, operate the engine or read sensor data. Schematically it is labeled as follows:

Denotation		Schemes symbol
		Positive port of the source of energy (battery, accumulator). Electrical current in circuit flows from positive to negative connection or pole.
		Negative or shared port of the source of energy (battery, accumulator). Sometimes it is also called ground or mass (car circuits).
		Resistor.
		Wire or connector without important resistance.
		Button.
		One and two pole switch.

Example for simple circuit:



terms

- **Resistance** is the counteraction of the connector for the flow of the current. It is measured in ohms (Ω). Resistance is indicated with **R**.
- **Amperage** is the flow of particles that can carry electric charge. It is measured in amperes (A). It can be compared to the flow of water in the pipes- the more you open the tap the stronger the flow of the water. Current is indicated with **I**. Current flows from circuits positive port to its negative port as it is shown in the example circuits.
- **Voltage** is the difference in electric potential energy. It is measured in volts (V) and indicated with **U**.

The three elements mentioned are connected by the Ohm's law:

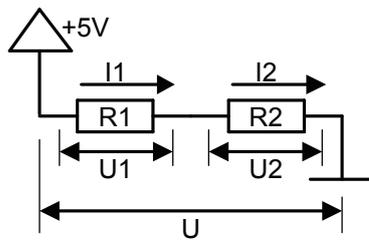
$$I = \frac{U}{R} \text{ jeb } U = IR$$

Fun-fact. This law is named after the German scientist Georg Simon Ohm who formulated it in 1827 after proving that in electric circuit there is coherence between voltage in a part of a circuit and its current.

Series and parallel circuits

Sometimes in more complex circuits it is necessary to form a circuit with several resistors, which are closed one after the other - series connection or next to each other - parallel connection:

Series circuit



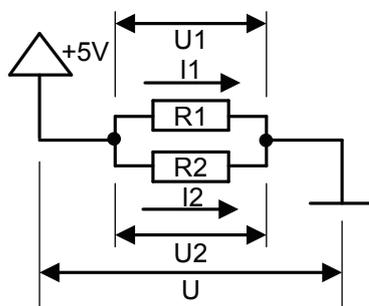
$$U_1 = I_1 R_1 \quad U_2 = I_2 R_2 \quad U = IR$$

$$U = U_1 + U_2$$

$$R = R_1 + R_2$$

$$I = I_1 = I_2$$

Parallel circuit



$$U_1 = I_1 R_1 \quad U_2 = I_2 R_2 \quad U = IR$$

$$U = U_1 = U_2$$

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$$

$$I = I_1 + I_2$$

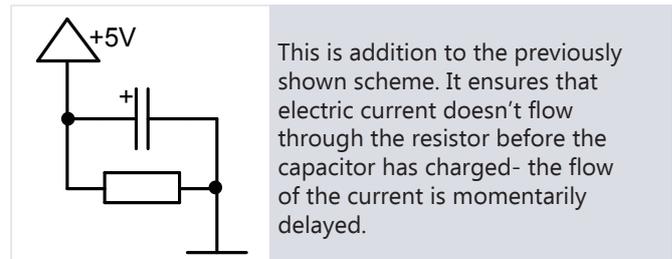
Capacitors

To have the electric and electronic circuits working in a preferred way one important element is the capacitor which main task is to store electric charge. In this way its function is similar to accumulator which can store electric charge for a period of time and when necessary use it in the circuit. In this topic we look only at the electrolytic capacitor which is usually made up of at least two decors of connectors with output for connecting to circuit and with dielectric (form material that is non-conducting or electrical isolator).

Schematic denotation of the capacitor:

Denotation	Schemes symbol
	Different symbols for electrolytic capacitor where with '+' is shown the direction of the positive port for the circuit.
	The display of the capacitor clearly shows the two decors of connectors that are separated by non-conducting material - isolator.

Simple example of the scheme:



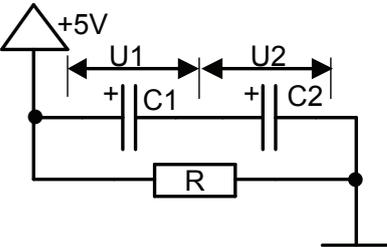
Terms

- The amount of electric charge that can be stored by a capacitor is called **capacitance** and indicated with **C**. The bigger the size of capacitor the bigger its capacitance.
- The measurement for capacitance is **farad (F)**. Usually when using capacitors their capacitance is from few picofarads (pF) up to ten's and hundreds of microfarads (μF). In a few special usages there can be used a capacitor with especially large capacitance, up to couple farads.
- Capacitor can also be characterised by **breakdown voltage (V)** by which its properties stay constant. It is advised to use smaller voltage because bigger voltage could damage the capacitor.

***Fun-fact.** The first capacitor was used in 1745 by German physicist Ewald Georg von Kleist and Dutch physicist Pieter van Musschenbroek. It happened in Leiden and the device was made from glass container with decors of metal foil on the inside and outside of them. It was named the "Leyden jar". It could be charged with electrostatic generator and it could store quite large high-voltage charge.*

Similar to the resistors, in more complicated circuits it is possible to create a circuit with many capacitors connected one after the other (Series circuit) or one next to the other (Parallel circuit):

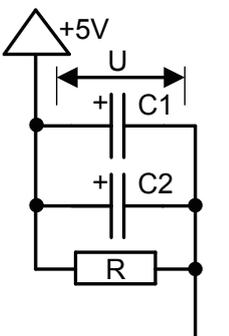
Series circuit



$\frac{1}{C} = \frac{1}{C_1} + \frac{1}{C_2}$

Allows adjusting the capacitance to individual needs.

Parallel circuit



$C = C_1 + C_2$

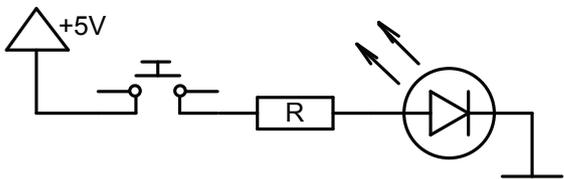
Allows increasing the general capacitance.

Light-emitting diode (LED)

There is another element added to the scheme to make it more understandable for practical tasks- Light-emitting diode (LED) which works similar to a light bulb and ensures that the current flows in one direction; unlike the bulb it won't work if it is connected in the wrong direction. In the topic "Semiconductors" we will talk about diodes and other elements in more detail. Schematic symbol for LED is this:

Denotation	Schemes symbol
	Light-emitting diode; the arrow shows the direction of the current.

Simple example of scheme:



This type of scheme ensures that the LED shines when the button is pushed.

It should be emphasized that the LED will work only if the positive and negative ports will be aligned with the current as it shown in the scheme. LED has quite small resistance so it requires a resistor to be added before or after it.

1.1. WORKSHEET. Turning on Light-emitting diode

Aim

To make your first electrical circuit as shown in the theoretical part.

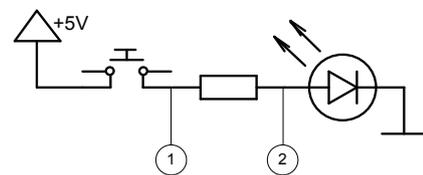
Steps of work

1. Make the scheme that is shown in the image.
2. Connect the scheme to a source of power and push the button. Let there be light!
3. Disconnect the scheme from the source of power.
4. Use multimeter to measure resistance between the points 1 and 2. How big is the resistance in ohms?
5. Connect the scheme to the source of power.
6. Lock down the switch or replace it with a wire so that the light-emitting diode would constantly shine.
7. Use multimeter to measure voltage between the points 1 and 2. How big is the voltage in volts?

Materials needed

Material/ part	Amount
Resistor (330 Ω)	1
Light-emitting diode	1
Button	1
Mounting cord	4

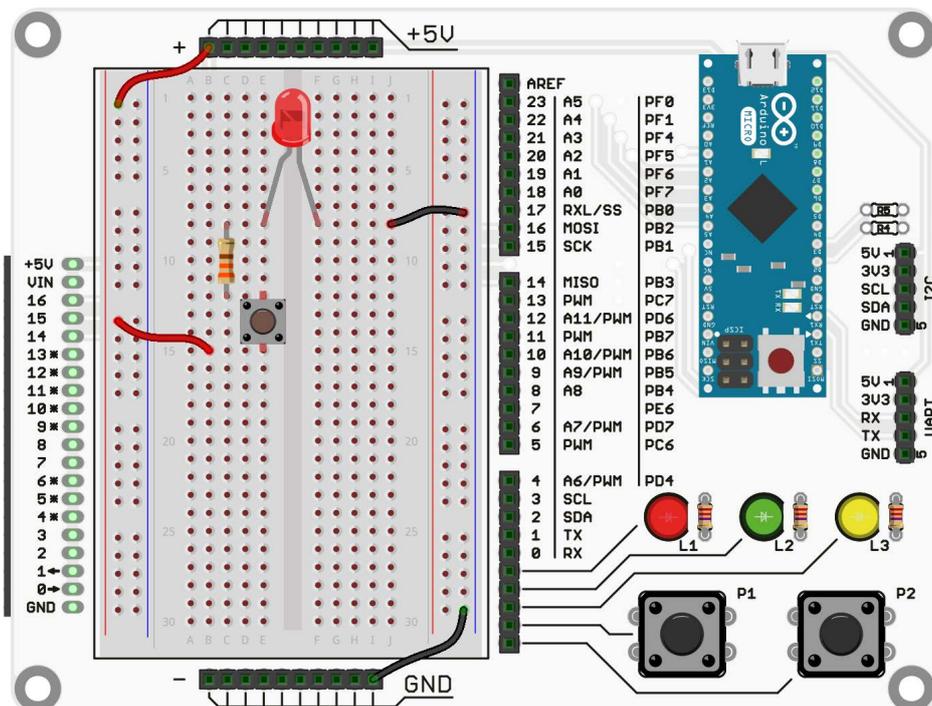
Scheme to be created



This scheme shows how to make a circuit that ensures that the light-emitting diode shines after pressing the button.

The two points (1 and 2) are meant for measuring voltage (U) and resistance (R).

Turning on LED



fritzing

1.2.

WORKSHEET. Switching Light-emitting diode

Aim

Learn how to use switches for interesting exercises.

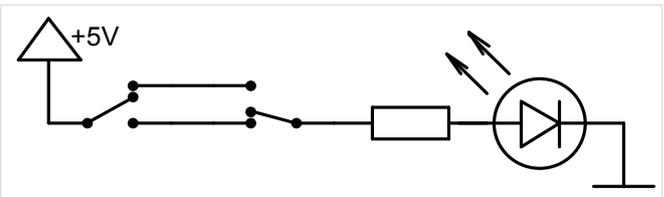
Steps of work

1. Make the scheme that is shown in the image.
2. Connect the scheme to the source of power and turn on one of the switches.

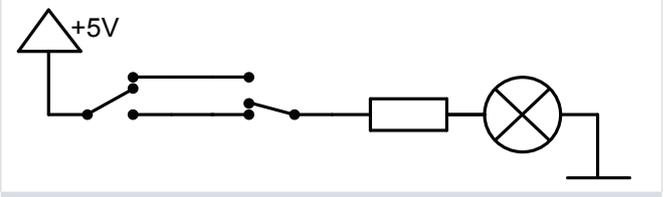
Materials needed

Material/ part	Amount
Resistor (330 Ω)	1
Light-emitting diode	1
Mounting cord	6
Switch	2

Schemes to be created



Switch for the staircase where the light can be turned on and off from different places. The light-emitting diode can be replaced with a light bulb as seen below.



It should be emphasized that the light bulb has its own resistance. Because of this the resistor can be removed.

Connecting LED

1.3. WORKSHEET. Series circuit with resistors

Aim

Learn how to use series circuit with resistors.

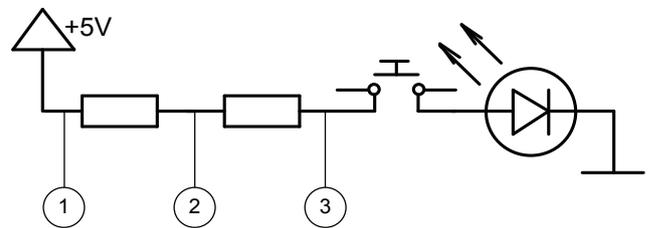
Steps of work

1. Make the scheme that is shown in the image.
2. Connect the scheme to the source of power and push the button.
3. Disconnect the scheme from source of power.
4. Use multimeter to measure resistance between the points 1 and 2. How big is the resistance in ohms?
5. Use multimeter to measure resistance between the points 2 and 3. How big is the resistance in ohms?
6. Use multimeter to measure resistance between the points 1 and 3. How big is the resistance in ohms?
7. Check if the measured resistance matches the calculated one by using formula for calculating resistance for series circuit.

Materials needed

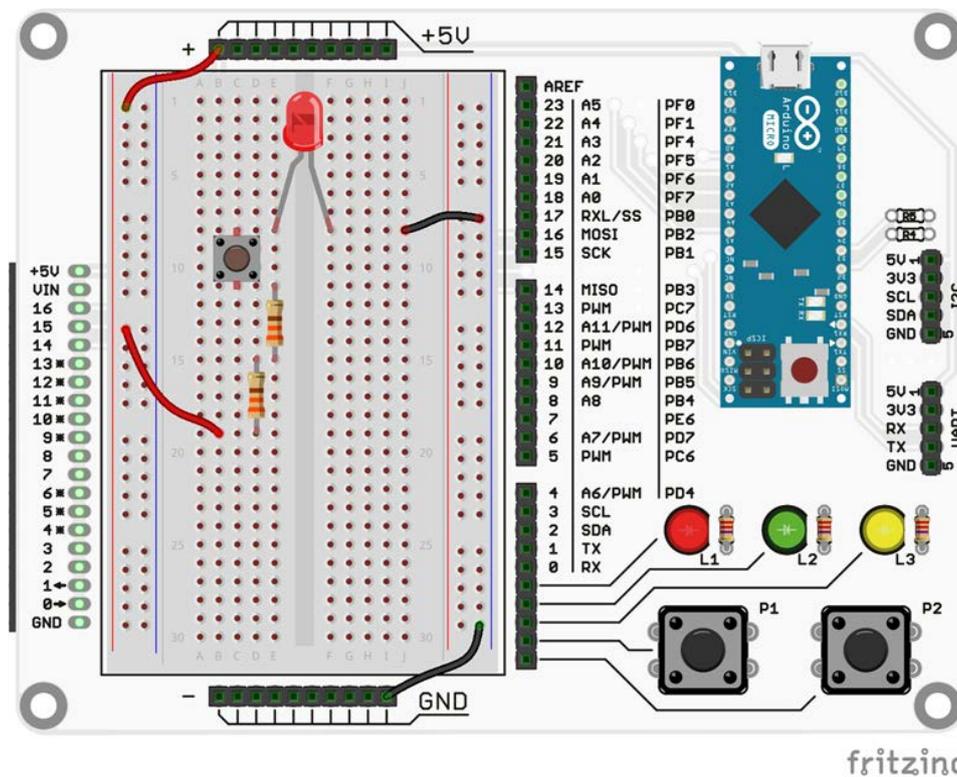
Material/ part	Amount
Resistors (330 Ω)	2
Light-emitting diode	1
Button	1
Mounting cord	4

Scheme to be created



The previously shown scheme with light-emitting diode with added resistor creating series circuit with resistors.

Resistors series circuit



1.4. WORKSHEET. Parallel circuit with resistors

Aim

Learn how to use parallel circuit with resistors.

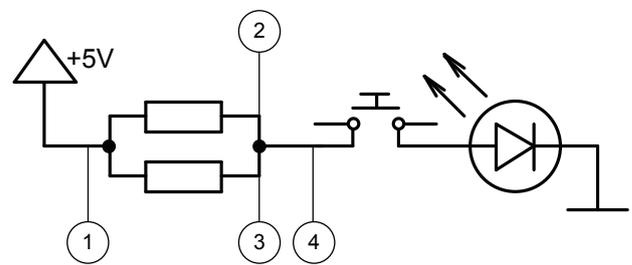
Steps of work

1. Make the scheme that is shown in the image.
2. Connect the scheme to the source of power and push the button.
3. Disconnect the scheme from source of power.
4. Use multimeter to measure resistance between the points 1 and 2. How big is the resistance in ohms?
5. Use multimeter to measure resistance between the points 1 and 3. How big is the resistance in ohms?
6. Use multimeter to measure resistance between the points 1 and 4. How big is the resistance in ohms?
7. Check if the measured resistance matches the calculated one by using formula for calculating resistance for parallel circuit.

Materials needed

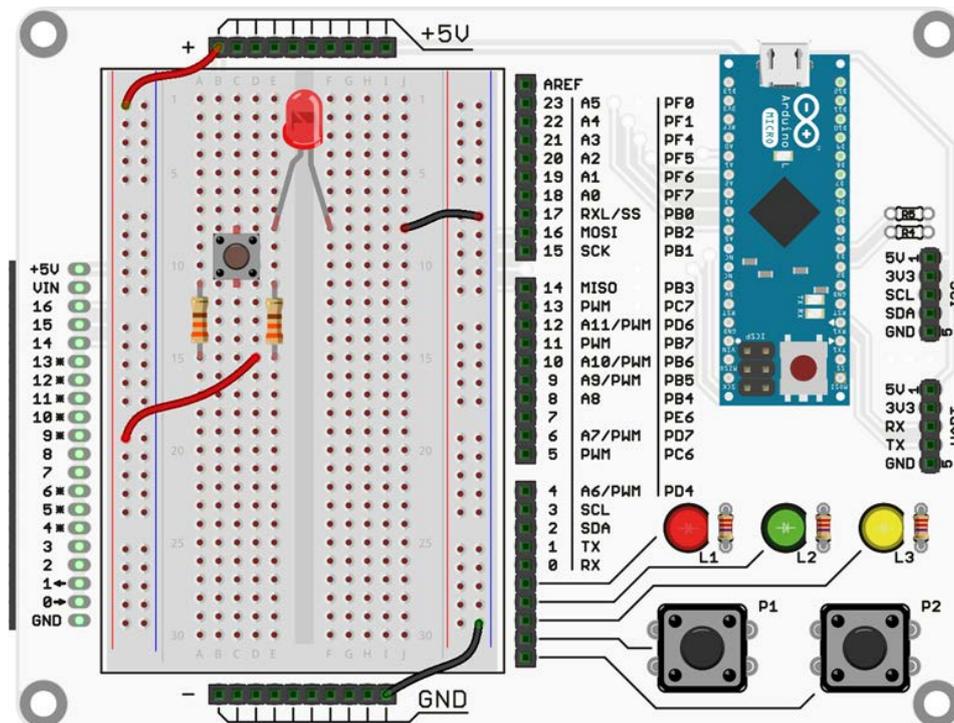
Material/ part	Amount
Resistors (330 Ω)	2
Light-emitting diode	1
Button	1
Mounting cord	4

Scheme to be created



The previously shown scheme with light-emitting diode with added resistor creating parallel circuit with resistors.

Resistors parallel circuit



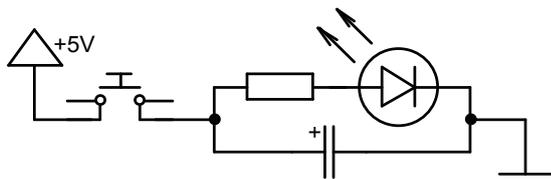
fritzing

1.5. WORKSHEET. Light-emitting diode with capacitor

Aim

Learn how to use capacitor.

Scheme to be created



Scheme ensures that the light-emitting diode extinguishes evenly after disconnecting the power; after turning the power off the capacitor ensures the energy needed for the light-emitting diode for a certain amount of time.

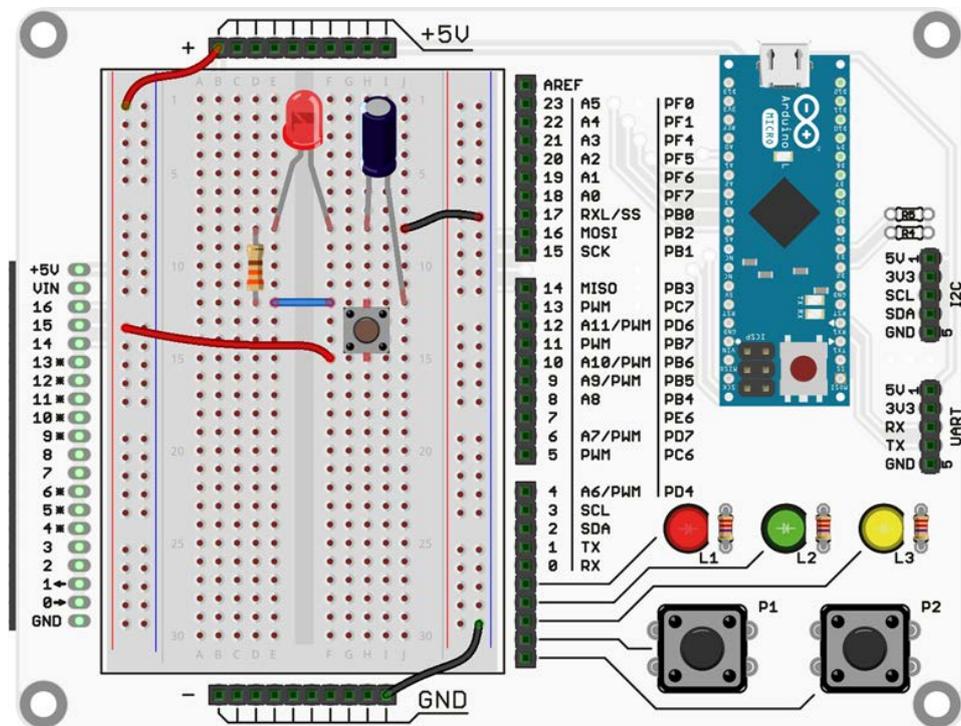
Steps of work

1. Make the scheme that is shown in the image.
2. Connect the scheme to the source of power and push the button. How many seconds does it take for the light-emitting diode to turn on?

Materials needed

Material/ part	Amount
Resistor (330 Ω)	1
Light-emitting diode	1
Electrolytic capacitor (2200 μF, 10 V)	1
Button	2
Mounting cord	2

LED with capacitor



1.6. WORKSHEET. Series and parallel circuit with capacitor

Aim

Learn how to use series and parallel circuit with capacitor.

Steps of work

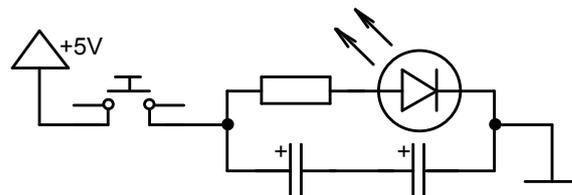
1. Make the scheme that is shown in the first image.
2. Connect the scheme to the source of power and push the button. How many seconds does it take for the light-emitting diode to turn on?
3. Make the scheme that is shown in the second image.
4. Connect the scheme to the source of power and push the button. How many seconds does it take for the light-emitting diode to turn on?
5. How did the time change from the series circuit.

Materials needed

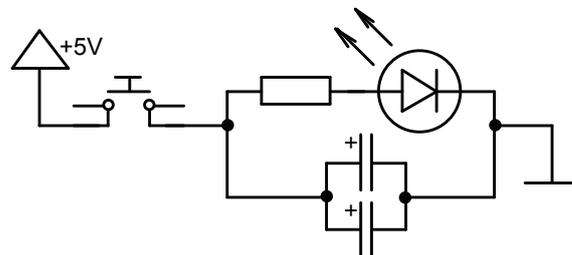
Material/ part	Amount
Resistor (330 Ω)	1
Light-emitting diode	1
Electrolytic capacitor (2200 μF , 10 V)	2
Button	1
Mounting cord	2

Scheme to be created

Series circuit

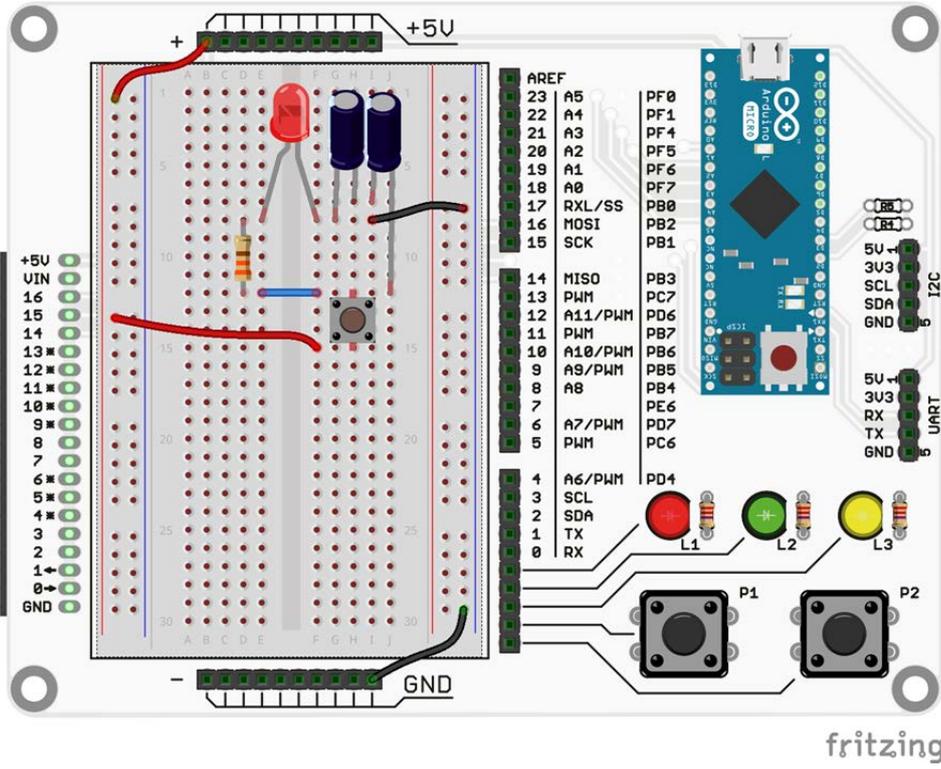


Parallel circuit

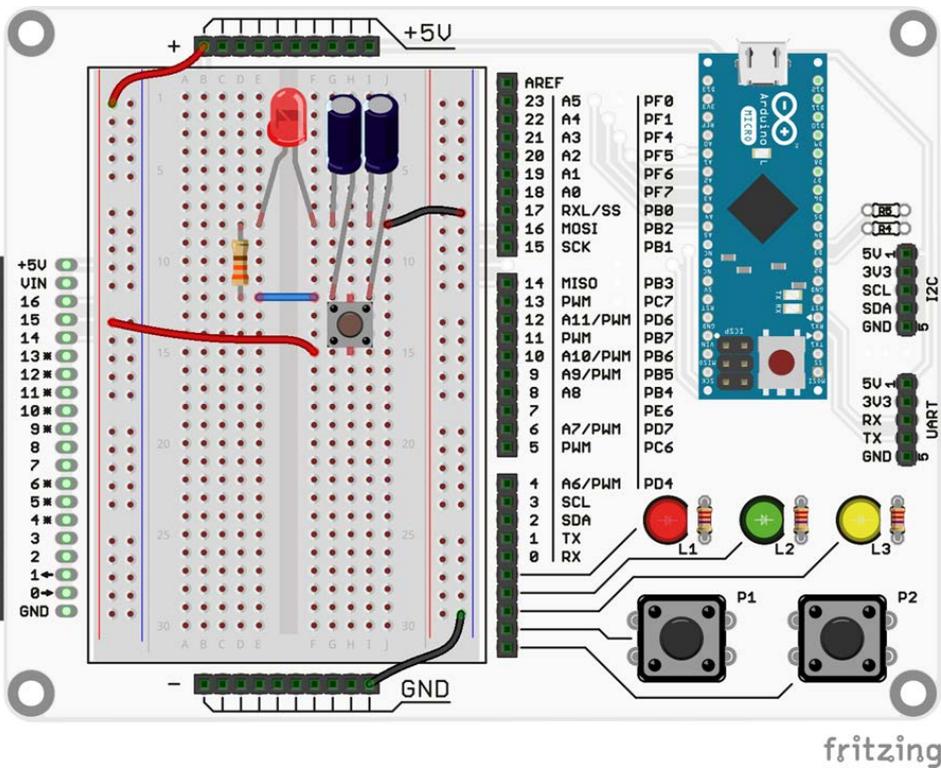


Schemes ensures that the light-emitting diode extinguishes evenly after disconnecting the power; after turning the power off the capacitors ensures the energy needed for the light-emitting diode for a certain amount of time. The delay varies with different type of circuit.

Capacitors series circuit



Capacitors parallel circuit



2. PREPARING PROGRAMMING ENVIRONMENT

2.1. Aim

The aim for this topic is to create programming environment, configure it make your first program by using the Arduino Micro® microcontroller that is included in the learning pack.

2.2. Theoretical part

Over all description

The learning constructor includes Arduino Micro type microcontroller which will help in learning the basics of programming as well as fully prepare you for programming your minisumo robot and prepare it for competition. Microcontroller is based on ATmega32u4 microprocessor base in collaboration with Italian corporation "Adafruit". It has 20 input/output pins; part of witch can be used for doing different specific tasks, for example managing electric motors or acquiring sensory data.

The Microcontroller package includes all that is necessary for creating and operating a program. It simply has to be connected to your computers USB (Universal Serial Bus) port while using the appropriate cable (more on it later). Microcontroller can be used together with prototyping plate or with robot. During the more simple programming exercises it can be used as separate device.

Taking in account the similarity with other Arduino type microcontrollers, and you already have experience in using them, it will help in doing the exercises in this book.

Over all information about the microcontroller

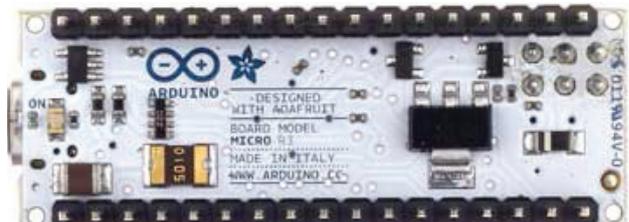
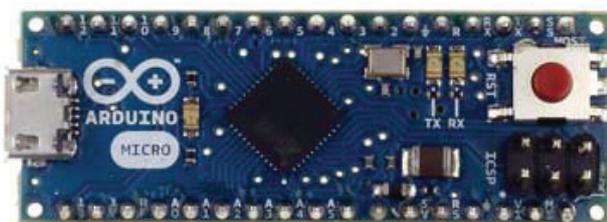
Parameter	Value
Needed microprocessor	ATmega32u4
Working voltage	5 V
Power voltage (recommended)	7–12 V
Digital input/output	20
PWM canals	7
Analog signal input/output	12
Input/output maximum amperage	40 mA
Programs available memory	32 KB (kilobyte), form which 4 KB are occupied by the source code
Clock frequency	16 MHz
Length/width	48 mm/18 mm
Weight	13 g

Detailed description of scheme can be found in: <http://arduino.cc/en/uploads/Main/arduino-micro-schematic.pdf>.

Power

To ensure that the microcontroller will work it can be connected to outer power source or USB port. Microcontroller determines its power source automatically. If outer power sources are used (not USB) then you have to use GND or VIN ports. Manufacturer advises to use 7-12 V voltage to ensure normal microcontrollers workability. If it is exceeded without reaching 20 V, there is a possibility that the microcontrollers feeding circuits can overheat. If the voltage is lover than 7 V then there is a possibility of unstable activity of microcontroller and the result will be unpredictable.

Microcontroller outer appearance



In addition to before mentioned, the microcontroller can ensure small outer circuit power by connecting them to microcontroller's legs (from now on MC legs).

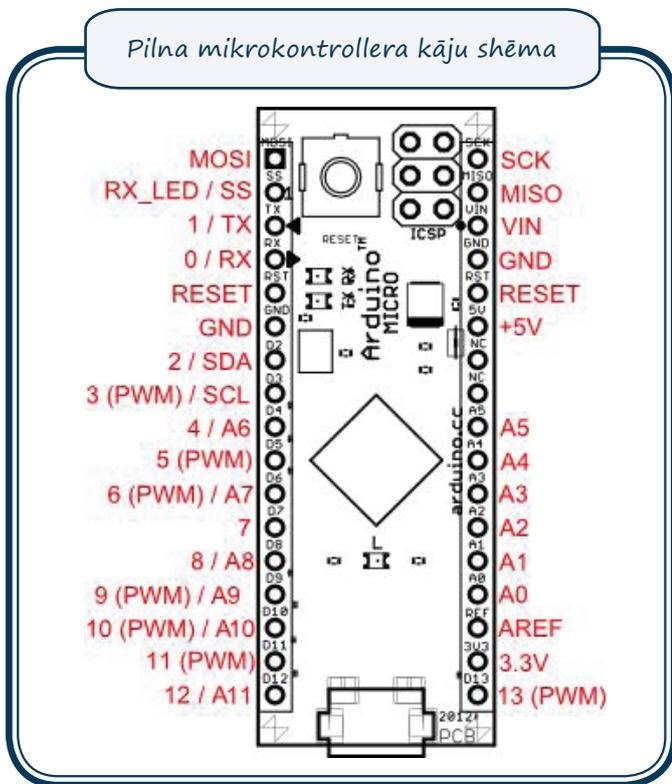
Denotation	Explanation
VIN	Power voltages input, if USB is not used; using outer power source.
5 V	5 V regulated feeding voltage which can be ensured by using both USB port and VIN.
3 V	3.3 V power voltage for outer circuit power. This output can ensure maximum amperage of 50 mA. If it is exceeded then the microcontroller's circuits can be permanently damaged.
GND	Ground or port 0

Below there are explanations for other microcontroller's legs denotations and their specific applications.

In addition to the previously mentioned pin functions the microcontroller can also ensure other functions that will be looked at in specific exercises.

Denotation	Explanation
1/TX un 0/RX	Pins needed for series communication connection. RX- for receiving data; TX- for transmitting data to outer devices. For transmitting and receiving data the signal level (voltage) cannot exceed 5 V.
2/SDA un 3/SCL	Ports are used for TWI (Two Wire Interface) data exchange which is an alternative for series communication transferring and receiving.
0 (RX), 1(TX),2,3	In addition to previously mentioned functions this pin can be used for receiving outer interrupts. Interrupts are used to signal to the main program for necessity to stop its work and do some special actions. This functionality is used with command <code>attachInterrupt()</code> .
PWM: 3,5,6,9,10,11 un 13	These pins ensure 8 bit resolution pulse-width modulation to port which is normally used in robotics for managing motors as well as other specific applications. This functionality is used with command <code>analogWrite()</code> .
LED 13	In addition to previously mentioned functionality this pin is connected to red LED which can be operated by the program.
A0–A5, A6–11	The microcontroller has 12 analogue pins with 10 bit resolution. That means that voltage level from 0 to 5 V is coded with a number from 0 to 1023. However, these settings can be changed with the command <code>analogReference()</code> .

Pilna mikrokontrollera kāju shēma



Input and output pins

Every one of the microcontrollers 20 pins can be used for emitting or receiving signals by using commands `pinMode()`, `digitalWrite()` and `digitalRead()`. These will be closer looked at in chapters about learning programming basics. All pins work in the range of 0 to 5 V. every pin can receive or give maximum 40 mA current. Each have inner load resistors in the range of 20-50 kΩ.

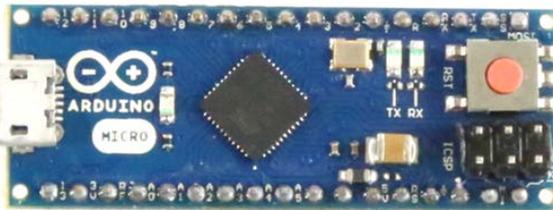
2.3. Installing programming environment

To begin developing software for the microcontroller you have to install and appropriately configure the environment of development which consists of programs editor and Arduino Micro driver. Down below there are all the steps to prepare programming environment for operating system Windows 8.

Step No.1. Prepare Arduino Micro and USB cable

To install development environment you need to prepare Arduino Micro as well as USB micro cable.

Arduino Micro and USB micro cable



USB micro cable usually is a part of modern mobile phone charging tools.

Step No.2. Download Arduino software development environment

First you need to download files that are required for installing the environment: <http://arduino.cc/en/Main/Software>.

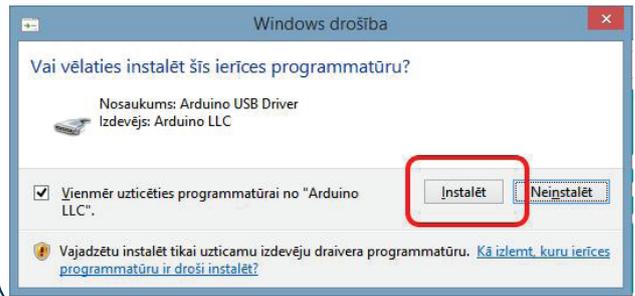
Choose installation files that are appropriate to the operating system and restriction laws.

Choice of installation files for download



If you downloaded the installer, run it after the download is completed. If you downloaded the ZIP archive then unpack it and run the installer. Follow the installer's instructions. If the operating system asks for permission to install the drive for the device, give the permission.

Safety question about installing the drive



Step No.3. Connecting Arduino

Connect Arduino to one of the free USB ports of your computer. Blue colour LED starts continuously shining. This means that Arduino microcontroller is working. Green LED will blink; this shows that developer's test program is working. If the green LED is not blinking it does not mean that microcontroller is malfunctioning.

Step No.4. Starting up the programming environment

With double-click on programming environments desktop shortcut start up Arduino programming environment.

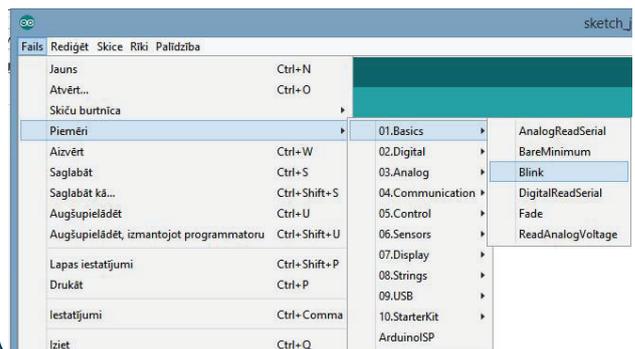
Programming language will be the same as your operating systems language; if it is Latvian then the programming environments menu will be in Latvian. If it is not the same than follow the instructions in:

<http://arduino.cc/en/Guide/Environment#languages>.

Step No.5. Open example program

Choose LED example program.

Opening example program



Example program will be shown in a new window; it is of the green LED turning on and off with one second interval.

Example program

```

Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.

Most Arduinos have an on-board LED you can control. On the Uno and
Leonardo, it is attached to digital pin 13. If you're unsure what
pin the on-board LED is connected to on your Arduino model, check
the documentation at http://arduino.cc

This example code is in the public domain.

modified 8 May 2014
by Scott Fitzgerald
*/

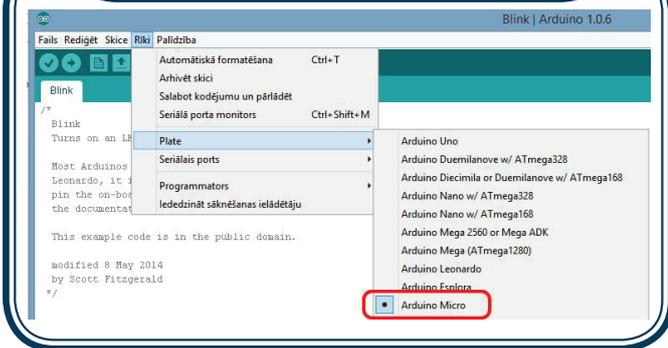
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
    
```

Step No.6. Choosing microcontroller

Choose menu Tools > Plate > Arduino Micro as it is shown in the image.

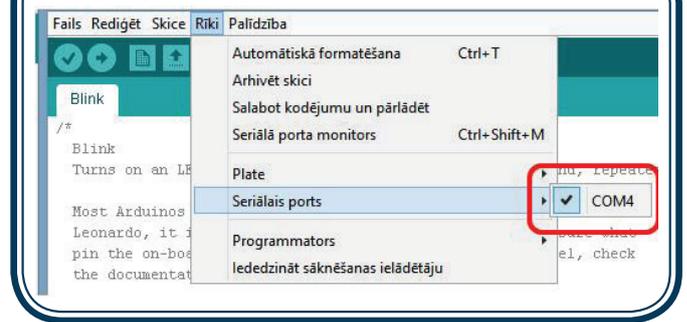
Setting up microcontroller's type



Step No.7. Setting up COM port

To ensure transmitting and receiving of data for the controller you need to set up series communication port – COM port. They are usually in sequential order and for Arduino microcontrollers they are usually bigger than COM3; as in COM4, COM5 etc. (in the image it is COM4).

Setting up microcontrollers COM port

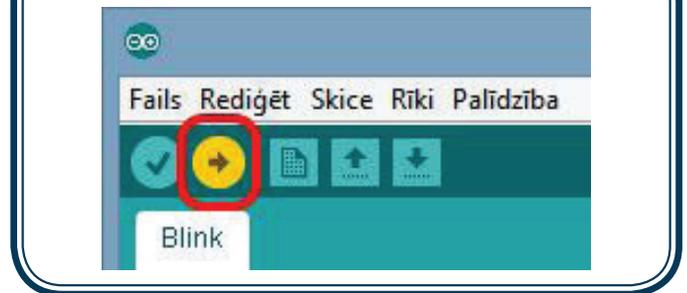


If you're not sure about the ports number, disconnect the microcontroller and check the port list again. The right port won't be in the list any more.

Step No.8. Sending example program for execution

Now all that's left is to push the button for uploading; wait for a few seconds in which you can see the indication of data being sent – Quick blinking of the LED (shows that data are being transmitted or received) – and wait for the notice "uploading completed".

Setting up microcontrollers COM port



After a few seconds the green LED will start blinking with one second interval. If it is so, then you have done everything to be able to start learning basics of programming!

If everything doesn't happen as it is described then please look over the advices that can be found in: <http://arduino.cc/en/Guide/Troubleshooting>.

If you wish to learn about possibilities of using microcontroller or basics of programming see one of these sources:

- Examples for performing exercises of varied difficulty: <http://arduino.cc/en/Tutorial/HomePage>.
- Explanations for programming language used: <http://arduino.cc/en/Reference/HomePage>.

3. Beginning programming

3.1. Aim

The aim of the topic is to introduce with the possibilities of programming that are offered by the programmable microcontroller (from now on MC) Arduino Micro used in the constructor. The task of this topic is not to show all programming possibilities, but to concentrate on achieving specific programming goals. That is why it is advised to see some of Arduino additional programming learning materials for example "Arduino Programming Notebook" by B. Evans which can be found in: http://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf.

In this chapter it is considered that you have successfully installed the programming environment as well as completed 1st chapters "Elements of electronics" exercises which can be referenced here.

3.2. Theoretical part

3.2.1. What does the MC Arduino program consist of

Every Arduino program consists of these parts:

1. **Global definition part** - in it variables, constants and other definitions that are valid in the entire program, are set; they are valid in both program parts mentioned below.
2. **Initialization part** - it is carried out only once, before carrying out the main program. Usually in this part all the constant variables, like MC leg meaning (input or output), constants and others, are defined. The most important aspect that is defined in this part is the input or output pins that will be needed for the program and their use. Example of initialization:

```
void setup()  
//the data type and name of the functional  
result  
{ //beginning of the function  
  //body of the function - all the  
  executable commands  
} //end of the function
```

As before mentioned this is a function that will be carried out only once. Function is characterised with its data type (number, a series of symbols or something else). In this example the key word **void** means that the initializations function does not have a result as in it is carried out only once. The title has to be **setup** so the MC embedded programs execution subsystem would be able to distinguish between initialization and other parts of the program.

3. **Cycle part** - it is executed all the time; it reads the output, does calculations and emits output signals. After executing the last command the program returns to the cycle parts first command and starts executing all the commands again. This is the main part of the program which includes the course and so called logic of execution.

```
void loop()  
//the data type and name of the functional  
result  
{ //beginning of the function  
  //cycle commands  
  //body of the function - all the  
  executable commands  
} //end of the function
```

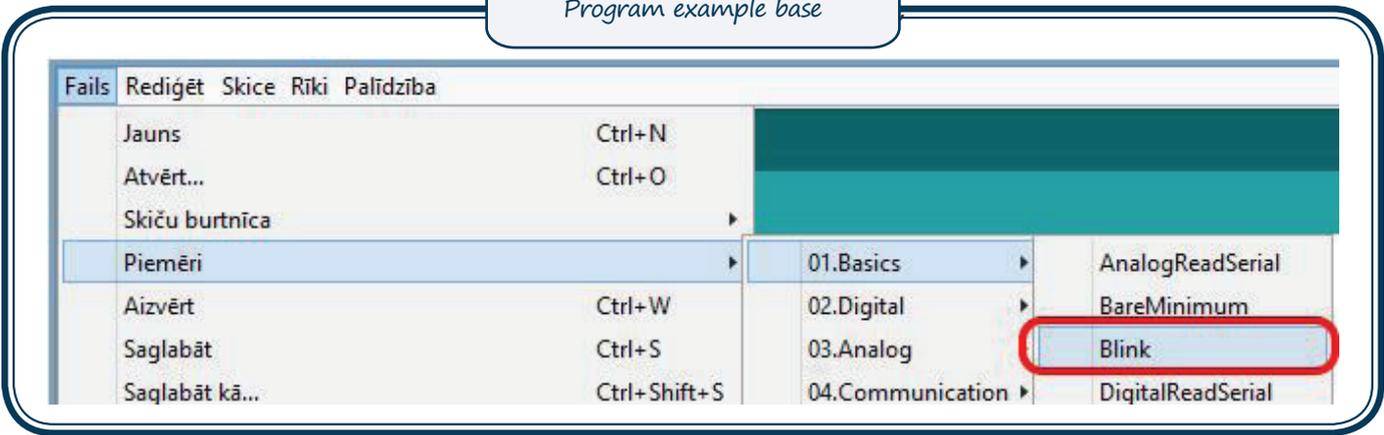
The type of this function is also void which shows that it has no result as in its executed cyclically during the programs operation.

As you can see, both parts are structured in the same way because both are functions with the only difference being its name, setup and loop, and meaning. To better understand the meaning of every function we will look at a simple example in the programming environments example section **Fails → Examples → 01.Basics → Blink.**

This is the program code is shown:

```
void setup()  
//the data type and name of the functional  
result  
{ //beginning of the function  
  pinMode(13,OUTPUT);  
  //defines 13 MC leg as output  
  //as in it can be used for emitting a  
  signal  
  //functions body - all executed  
  //commands  
} // end of the function
```

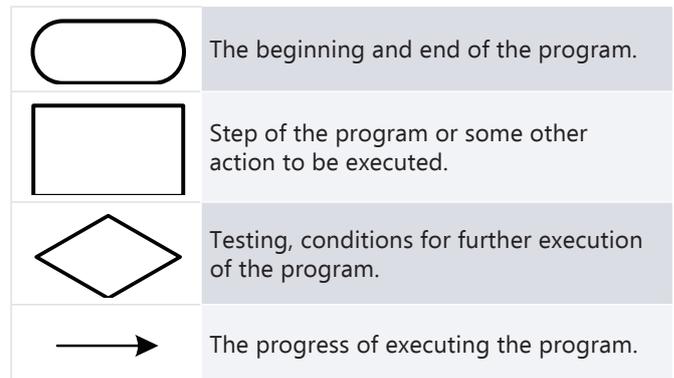
Program example base



```
void setup()
//the data type and name of the functional
//result
{ //beginning of the function
digitalWrite(13,HIGH); //output signal is
//set up on output pin No13
//logical level 1 as in +5 V
delay(1000); //the execution of the program
//is stopped for 1000 milliseconds
//or 1 second
digitalWrite(13,LOW); //output signal is set
//up on output pin No13
//logical level 0 as in 0 V
delay(1000); // again the execution is
//stopped for 1000 milliseconds
//or 1 second
//cycle commands
}
//functions body- all executed
//~↑↑á^äb
cÁÁÐÐæ^ää~ää\áæÁä|^'^\↔~^
```

3.2.1.1. Display of the program in diagram

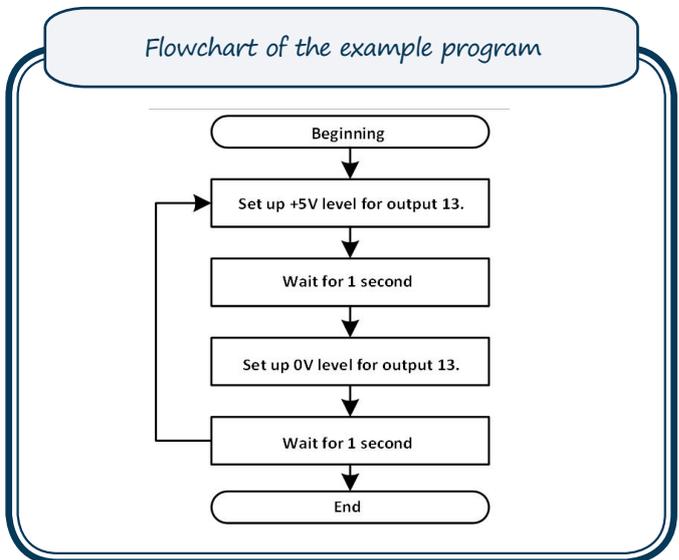
Sometimes it is easier to display difficult algorithms as a graphical diagram. One of the most popular types of these diagrams is the flowchart. Its base elements look like this.



In this program code you can see:

1. The MC leg 13 is defined as an output. With it you can emit signal. In this example that program is set to periodically determine the output signal in logical level 1 (+5V) and 0 (0V). Taking in account that the 13 pin is connected to MC Arduino LED, by changing the signal in this way the LED gets periodically turned on or off.
2. A continuous function **loop()** has been created which allows 1 second or 1000 milliseconds for every signal level. It is achieved by stopping the execution of the program. As long as the program doesn't change the input/output mode the function doesn't change. That way after emitting signal level +5V and then stopping the execution the LED will continue to shine until the signal level will be lowered to 0V.
3. The last row defines that the program with 0 V level is also stopped for 1 second. That way the periods of turning on and off of the LED are the same. After this command the program returns to first command of the **loop()** and begins its execution all over again.

By using this flowcharts elements the example function **loop()** can be shown as a diagram in the image below. In the following tasks the functions will be displayed as flowcharts to better and easier show more difficult processes of executing programs.



3.2.1.2. Programs basic functions

In this subchapter we will look at MC Arduino programs main constructions.

Functions

Function is the set of commands that is executed every time it's called. The two structures are already known - **setup()** and **loop()**. The programmer usually tries to put the main idea of the program in one or many self-created functions which are called from **setup()** or **loop()**. Structure looks like this:

```
type structure name (arguments)
{
  commands;
}
```

We can create our own example function which allows turning on and off the LED (similarly to the previously shown example):

```
void exampleFunction()
{
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13,LOW);
  delay(1000);
}
```

To call it we need to slightly modify the **loop()** function:

```
void loop()
{
  exampleFunction();
}
```

The program code in Ardiono environment looks like this:

```
void loop()
{
  exampleFunction();
}

void exampleFunction()
{
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13,LOW);
  delay(1000);
}
```

If we want to get a specific result from our function then we need to show the returning type of the function:

```
int sumOfTwoFigures(int x, int y)
{
  return (x+y);
}
```

Variables

Variables are a way how to temporarily save or give name to specific digital or other type of data. The values of variables can be changed during the execution of the program whenever it is necessary. Every variable has its data type (see below), name and value. In the example below you can see how new variable is defined with starting value and then it is changed:

```
int myExampleVariable = 0;
//assigning variables definition and
//starting value
mansPiemeraMainigais = 32;
//change of variables value
```

Attention! The names of the variables have to be meaningful so others would be able to understand the program code.

You have to remember that variable is similar to storage; it reflects the contents of MC memory. The contents are always available and it doesn't change if it is not changed. In the next example the previously shown value is increased by 10.

```
myExampleVariable = myExampleVariable + 10;
//10 is added to the variables value
```

Attention! Remember that '=' is not the 'equals' sign; it is the assigning of value. The command in the previous example is not comparing as in math; it is the changing of the existing value. The basic idea of the action is this: get the value of the variable **myExampleVariable** which is 32, and increase it by 10. Save the result in the variable **myExampleVariable**. Now the value of the variable **myExampleVariable** will be 42.

Variables have to often be checked to make a decision about further execution of the program. In the example below the value from the MC leg 2 is read and compared. If the value is bigger than 100 then the variable is assigned the value 100 (example from: http://play-ground.arduino.cc/uploads/Main/arduino_note-book_v1-1.pdf):

```

int inputVariable = 0
//defines the variable and its starting value
inputVariable = analogRead(2);
  //reads the analog signal value from the
  //MC lag 2 and records the value in the
  //variable}
if (inputVariable > 100)
  //the variables value is compared to 100
  {
inputVariable = 100;
  //if the value is bigger than 100 the
  //variable is assigned the value 100;
  //variables value
  //is restricted by upper threshold 100
  }
delay(inputVariable);

```

The activity area of variable

Every variable has its activity area (visibility) where the value can be accessed. Variable can be defined in the very beginning of the program (global variables), in functions body or sometimes even in command block, for example in the cycle. The place where you define the variable determines its visibility. In the next example you can see many variables with different visibility:

```

int globalVariable;
//this variable is visable in whole program
void setup()
{
  int setupVariable;
//this vatiabile is accasible only in setup()
}
void loop()
{
  int loopVariable;
//this vatiabile is accasible only in loop()
//functions
//body
for(int i = 0; i < 20; i = i + 1)
//in this cycle new variable I is defined
//which is accessible only in this cycles
//body. That is why these variables are
//sometimes called cycle variables
  {
    //other commands
  }
}

```

Data types

Every variable can have its own data type which determines its place in MCs memory as well as the way it is used. Here we will look only at the main data types, but there are many more.

Data type	Explanation
<i>byte</i>	8 byte numeric type which can store numbers from 0 to 255 <i>byte</i> exampleVariable = 123;
<i>int</i>	16 byte integer which can store values from -32 767 to 32 768 <i>int</i> exampleVariable = 12 300;
<i>float</i>	Real numbers data type which take up 32 bytes and is capable in storing numbers from around $3,4 \cdot 10^{38}$ to $-3,4 \cdot 10^{38}$. <i>float</i> exampleVariable = 12300.546;
<i>Array</i>	Arrays are a set of the same data types which can be accessed with order number or index. Index for the first element is always 0. Array's values can be assigned to it all at once at the beginning or over the course of programs execution. <i>int</i> exampleArray[] = {12,-3,8,15}; Here you can see array with the name exampleArray and data type int . There are initial values assigned to it. This means that elements No0 value is 12 and elements No3 value is 15. <i>int</i> otherVariable = exampleArray[1]; After executing this command the value of the variable otherVariable is -3. Arrays are very useful for processing in cycle part. In the next example you can see how necessary values from analog signal input are automatically saved in previously defined array. for(int i = 0; i < 4; i = i +1) { exampleArray[i] = analogRead(2); } Example cycle is started with 0 index (i = 0) which is increased by 1 as long as it is lower than 4. This means that the cycle will be executed for the last time when i value will be 3, because if i is equal to 4 then equation i < 4 will be false and cycles action will be stopped.
<i>Bool</i>	This type of variable can take the value of <i>true</i> or <i>false</i> . Accordingly Arduino programming environment allows the variables to take these values: True TRUE False FALSE Logical 1 (+5 V) HIGH Logical 0 (0 V) LOW

Logical operators and comparing

Comparing was already shown in example programs with the help of operator if. However, it is important to look at different operators for fast and easy coding.

```

a==b //variable a is equal to variable b
a!=b //variable a is not equal to variable b
a<b //a is smaller than b
a>b //a is bigger than b
a<=b //a is smaller or equal to b
a>=b //a is bigger or equal to b

```

All logical equations can be *true* or *false*. These logical equations can be combined with logical operators to create more difficult conditions.

```

if(a>b && a<45)
// '&&' operator translates as 'and';
//for all of the values of the equation to be
//true the equation
// a > b as well as a < 45 has to be true.

```

```

if(a>b || a<45)
// '||' operation translates as 'or';
//for all of the values of the equation to be
//true one of the equations
//a > b or a < 45 has to be true.

```

```

if(!a>b)
// '!' operation translates as 'not';
//for all of the values of the equation to be
//true the equations
//a > b has to be false

```

Condition operator

Condition operator helps to make a decision about further action of the program. For example, if you have to make a decision about processing a specific input signal you can use this construction:

```

int inputVariable = analogRead(2);

if(inputVariable > 100)
{
    //do signal processing or other actions
}
else
{
    //do alternative sequence of actions
}

```

As you can see, alternative section **else** is added to the operator **if**. This allows making some other actions if the signals value is not 100 as shown in the example. One condition operator can include others, creating complicated program action scenarios, for example:

```

int inputVariable = analogRead(2);
if(inputVariable > 100)
{
    //do signal processing or other actions
    if(ieejasMainigais < 130)
    {
        //special processing of input value
        //when it is between 100 and 130
    }
}
else if (inputVariable < 30)
{
    //do signal processing or other
    //actions, if input signal value is
    //less than 30
}
else
{
    //do alternative sequence of actions,
    //if non of the conditions work
}

```

FOR cycle's operator

Cycle operator allows executing the same actions for a specific amount of times. This is where cycles are similar to **loop()** function, but it allows to control cycles execution. Every time all the actions in the cycle have been executed there is one cycle's iteration step. Because of this cycle is one of the fundamental techniques of programming which is the basis for all programs and automations as a whole. FOR cycle construction is this:

```

for (initialization;condition;
action with cycles variable)
{
    //cycles bodies operations
}

```

- Initialization usually is used to initialize cycle's variables value which can be equal or not equal to 0.
- Condition allows managing the amount of cycle's iteration steps because it determines how every next iteration step is carried out.
- Actions with cycle's variable allow determining the amount of iteration steps.

Typical FOR cycles example:

```

for (int i = 0; i < 4; i = i + 1)
{
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
}

```

By the end of executing this cycle the LED on the MC will have turned on and off 4 times.

WHILE cycle's operator

This cycle's operator is similar to FOR, however, it doesn't have cycle's variable so it allows executing unknown amount of iteration step. Cycle's management is carried out with the condition that the cycle is true for the interval to be executed. This is WHILE cycles construction:

```
while (condition that is true)
{
    //cycles body
}
```

WHILE cycle can serve as a good tool for executing previously unpredictable program. For example if you need to wait before input signal is in a specific level of voltage you can use this construction:

```
int inputVariable = 0;
while (inputVariable < 100)
{
    digitalWrite(13,HIGH);
    delay(10);
    digitalWrite(13,LOW);
    delay(10);
    inputVariable = analogRead(2);
}
```

Cycle ensures LED to blink until the input signal is in specific level. With this the user is given an indication that the input signals level has to be increased.

3.1. WORKSHEET. Button and Light-emitting diode

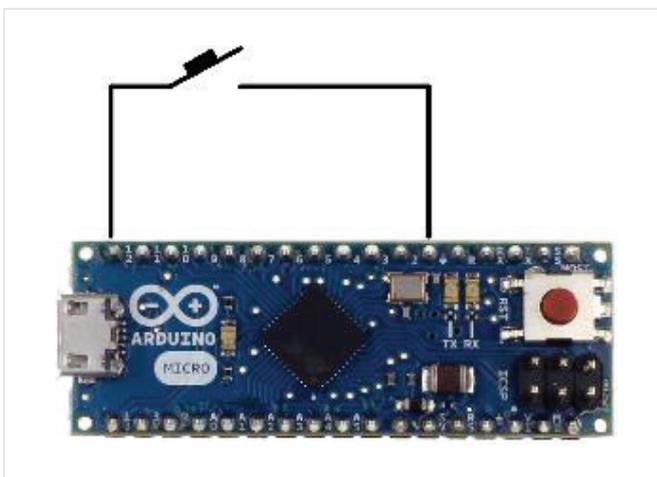
Aim

Create your first program which will turn on the LED diode after reacting to a pressed button.

Materials needed

Material/part	Amount
Mounting cord	1

Scheme to be created



This scheme allows creating a circuit that ensures that the LED shines when the button is pushed. Reaction to the pressure on the button is ensured by the MC Arduino, unlike the examples shown in chapter about basics of electronics. In the constructor, the buttons **P1** and **P2** are connected to the collective output – 0 pole. That means that the scheme can be simplified as it is shown in the image "Needed materials and circuit", there's no need to use extra details.

Steps of work

1. Create the scheme of the circuit.
2. Write the given program.
3. Upload the program to MC Arduino memory as it was shown in chapter for preparing programming environment.
4. Push the button.

```
void setup()
{
  pinMode(13, OUTPUT);
  //set MC leg No13 as an output
  pinMode(12, INPUT);
  // set MC leg No12 as an input
  //which will read button position
  digitalWrite(12, HIGH);
  //connect inner resistor
  //so we wouldn't have to use extra
  schemes
}
void loop()
{
  bool ButtonPosition = digitalRead(12);
  //read position of MC leg No 12
  digitalWrite(13, not(ButtonPosition));
  //set up MC legs No13 position
  //according to read value
}
```

Short explanation:

Initializations part:

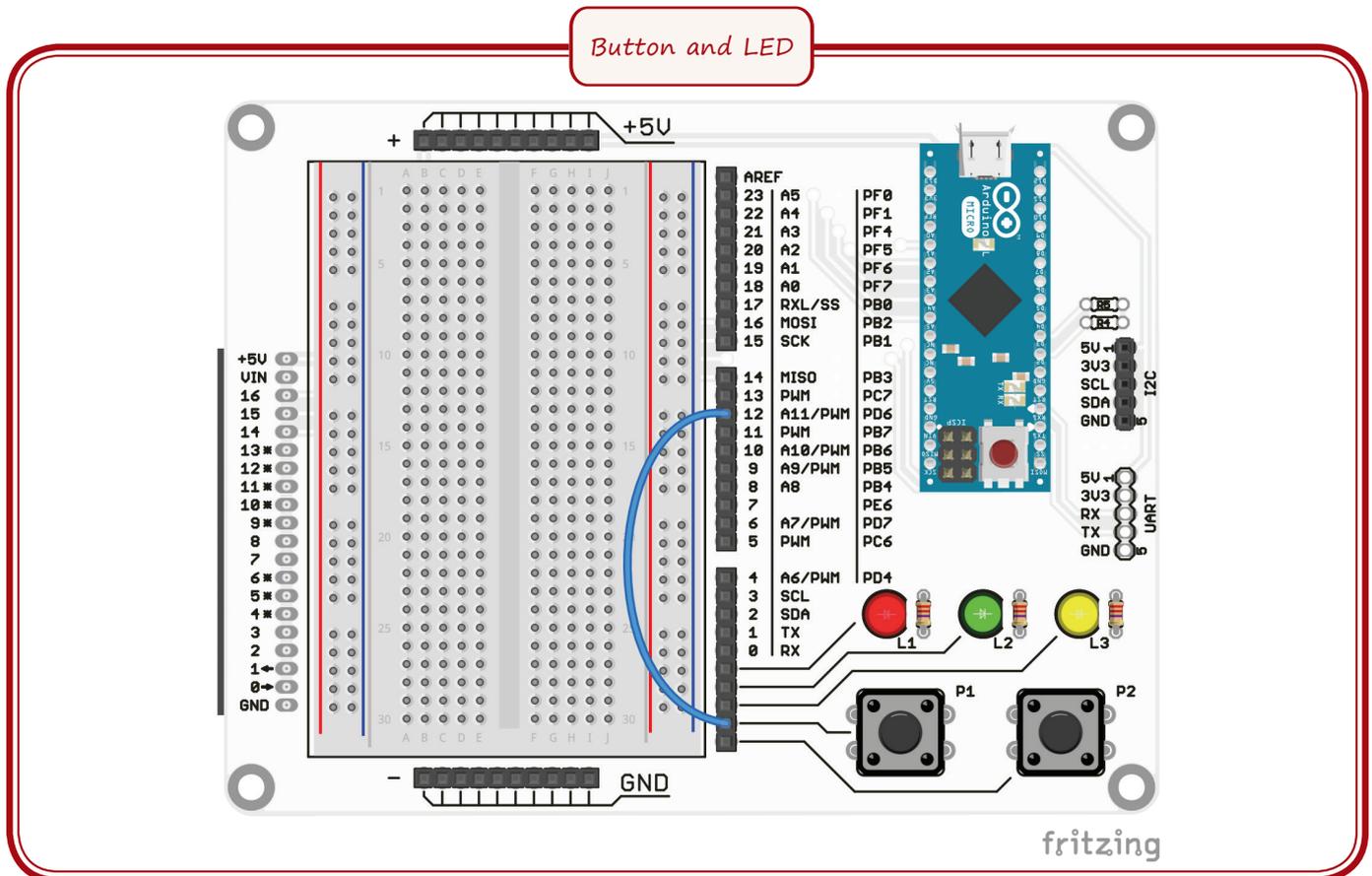
- Similar as before the MC leg No13 has to be defined as an output;
- MC leg No12 is defined as input which is required for reading the position of the button.
- Resistor is connected to MC leg No12 so there would be no need for outer circuits. This resistor ensures MC leg No 12 to be in logical 1 level (+5V level) all the time until there are no other sources of signal.

Cycles part:

- Position variable **ButtonPosition** is created with the logical type **bool**. It can take two values – true (logical 1 or +5V) or false (logical 0 or 0V). Variables are used for short-term storage of information. As long as their values are not changed, it doesn't change no matter the process of execution. In this example if during one execution of the cycle the variable has an assigned value then the value doesn't change in any of the other times of execution of the cycle unless it is changed by some event or command in the program.

- Command **digitalRead(12)** allows to read the logical value (1 or 0) of MC leg No12. Read value is saved in the variable **buttonPosition**.
- Command **digitalWrite(13,...)** allows to install the level of MC leg No13.
- In this command you need to also use **not()** which reverses the logical value written in the brackets; if logical 0 is shown it is turned into logical 1.

This is required because we use inner resistor; if there are no other signals the MC legs No12 level will be +5V or logical1. If we don't use **not()** operator LED will turn on when the button isn't pushed; reversed to the usual.



3.2. WORKSHEET. Staircase lighting

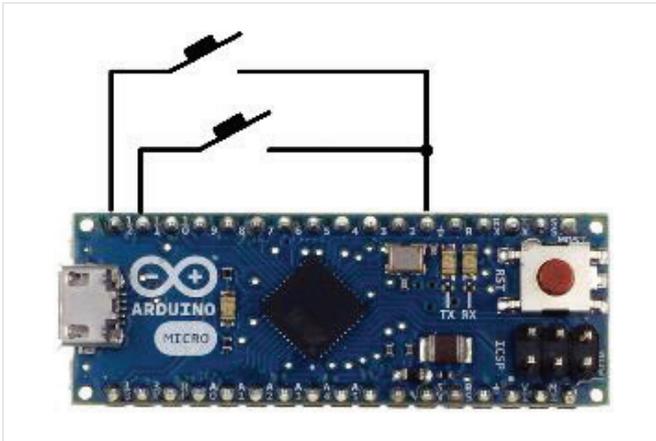
Aim

Create a program that allows turning on or off the LED by using two buttons, imitating staircase by turning on the light with one button but turning off the light with another.

Materials needed

Material/part	Amount
Mounting cord	2

Scheme to be created



This scheme allows creating circuit that ensures that the LED shines when the button is pushed. Reaction to the pressure on the button is ensured by MC Arduino, unlike the examples shown in chapter about basics of electronics. In the constructor the buttons **P1** and **P2** are connected to collective output – 0 pole. That means that the scheme can be simplified as it is shown in the image "Needed materials and circuit", there's no need to use extra details.

Steps of work

1. Create the scheme of the circuit.
2. Write the given program.
3. Upload the program to MC Arduino memory as it was shown in chapter for preparing programming environment.
4. Enjoy your staircase lighting system.

```
bool LEDposition = false;
//variable will save LEDs
//value
void setup()
{
  pinMode(13,OUTPUT);
  //set MC leg No 13 as an output
  pinMode(12,INPUT);
  //set MC leg No12 as an input
  //which will read buttons position
  digitalWrite(12,HIGH);
  //connect inner resistor
  //so we wouldn't have to use extra
  //schemes
  pinMode(11,INPUT);
  //set MC leg No11 as an input
  //which will read buttons position
  digitalWrite(11,HIGH);
  //connect inner resistor so we
  //wouldn't have to use extra schemes
}
void loop()
{
  bool ButtonPosition_1 = digitalRead(12);
  // read position of MC leg No12
  bool ButtonPosition_2 = digitalRead(11);
  // read position of MC leg No11
  if((ButtonPosition_1 && ButtonPosition_2)
      == false)
    //check if any of the buttons is pushed
    {
      LEDposition =
        not(LEDposition);
      //if is then LED position is changed
      //to oposite
    }
  digitalWrite(13,LEDposition);
  //set up new LEDs position
  delay(300);
}
```

Short explanation:

Global variables:

- One variable is added which is accessible in both functions – **setup()** and **loop()**, and also is meant to keep the LEDs position outside every execution of functions cycle.

Initialization part:

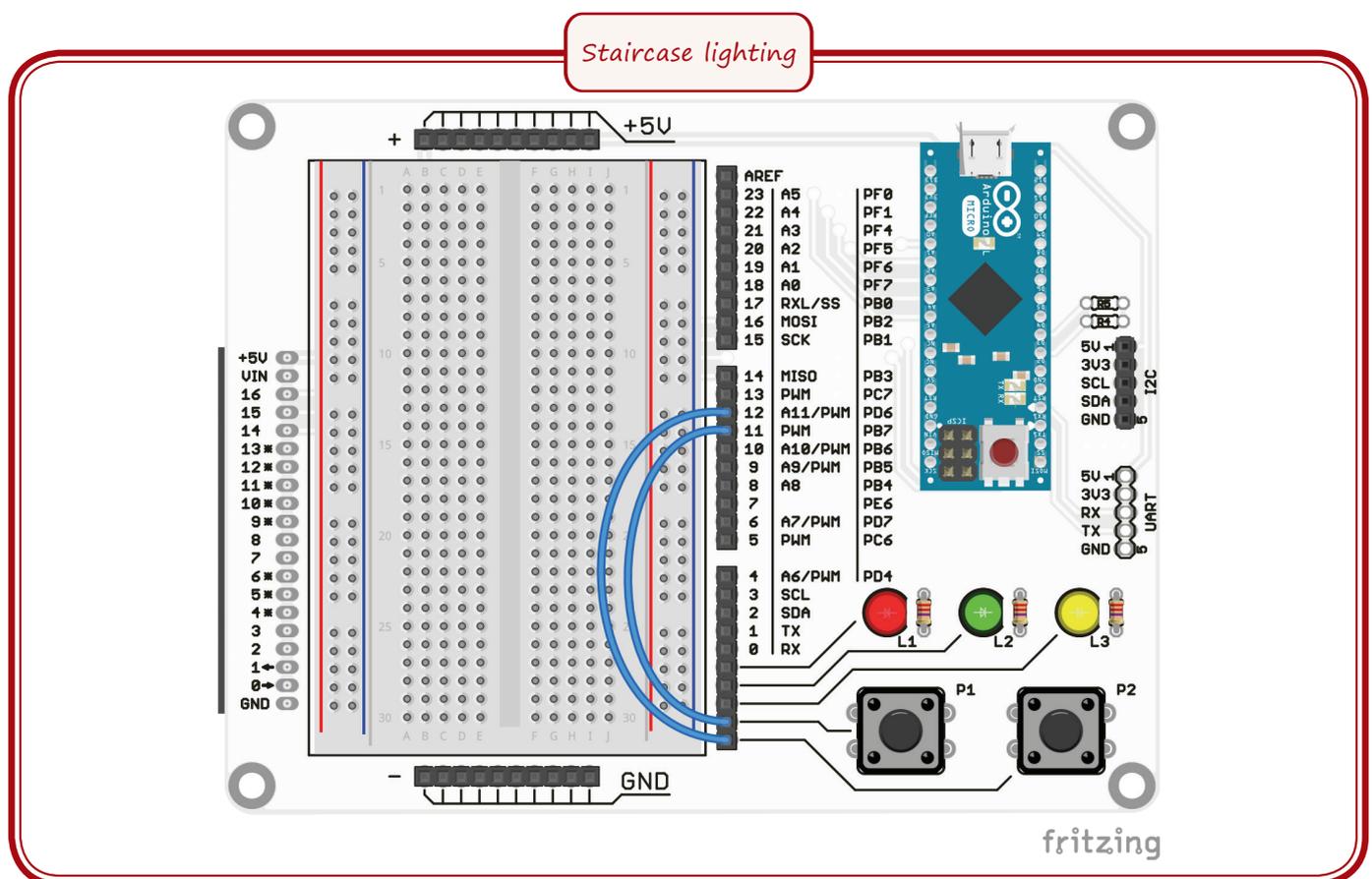
- Another buttons input is added to initialization form the previous task – MC leg No11 which is precisely defined as MC leg No12.

Cycle part:

- Similarly to the previous task two positions of the buttons are defined – **Buttons_Position_1** & **Buttons_Position_2** which will accordingly keep MC legs No11 and No12 position for one cycle.
- There is an if condition which allows to evaluate positions of the buttons. Here it checks whether any of the buttons are pushed. For this purpose we use logical operator **&&** - logical AND.

In this case it will give the value true only if both of the operands (buttons positions) are in logical level 1; none of the buttons are pressed and inner resistor ensures that they have logical voltage level 1.

- If one of the buttons is pushed then the value of the condition is logical 0 or **false**. In this case the position of the LED has to be changed to the opposite. If the LED is on it has to be turned off and vice versa.
- Then the new position of the LED is saved.
- Small pause of the programs execution has to be added so the programs actions speed would be in sync with humans actions speed. This is ensured by using the command **delay(300)** – stop the execution for 300 milliseconds.



3.3. WORKSHEET. Light-emitting diodes brightness

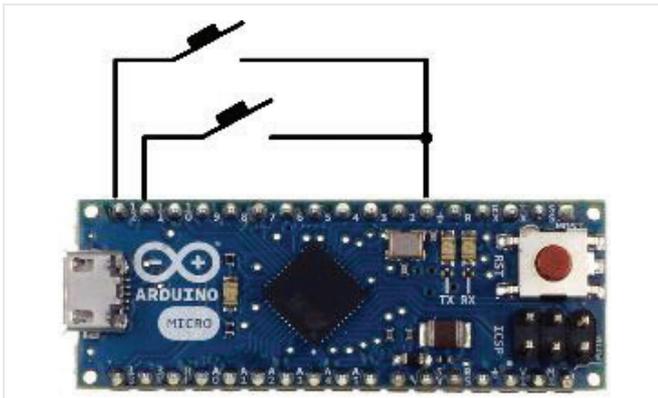
Aim

Create a program which allows changing the brightness of the LED using two buttons where one makes the LED become brighter, but the other dims it until it turns off.

Materials needed

Material/part	Amount
Mounting cord	2

Scheme to be created



This scheme allows creating circuit that ensures that the LED shines when the button is pushed. Reaction to the pressure on the button is ensured by MC Arduino, unlike the examples shown in chapter about basics of electronics. In the constructor the buttons **P1** and **P2** are connected to collective output – 0 pole. That means that the scheme can be simplified as it is shown in the image "Needed materials and circuit", there's no need to use extra details.

Steps of work

1. Create the scheme of the circuit.
2. Write the given program.
3. Upload the program to MC Arduino memory as it was shown in chapter for preparing programming environment.
4. Enjoy your brightness changer at work.

```
int ledLeg = 13;
int increase = 0;
int brightness = 127;

void setup()
{
    pinMode(ledLeg, OUTPUT);
    analogWrite(ledLeg, brightness);
    pinMode(11, INPUT);
    pinMode(12, INPUT);
}

void loop()
{
    bool ButtonPosition_1 = digitalRead(12);
    // read position of MC leg No12
    bool ButtonPosition_2 = digitalRead(11);
    // read position of MC leg No11
    if(ButtonPosition_1 == true)
        { increase = 5; }
    if(ButtonPosition_2 == true)
        { increase = -5; }
    spozums = spozums + pieaugums;
    if(brightness >= 255) { brightness = 255; }
    if(brightness <= 0) { brightness = 0; }
    analogWrite(ledLeg, brightness);
    delay(30);
}
```

Short explanation:

Global variables:

- Here we define three variables:
 - ledLeg – shows to LED built-in into the MC.
 - Increase – increase in brightness. If it is positive then the brightness slowly reaches its maximum, but if negative then slowly extinguishes.
 - Brightness – brightness which is saved for determining the brightness of the LED. This variable can take values from 0 to 255.

Initialization part:

- As before, inputs and outputs are determined but also the starting brightness of the LED is determined.

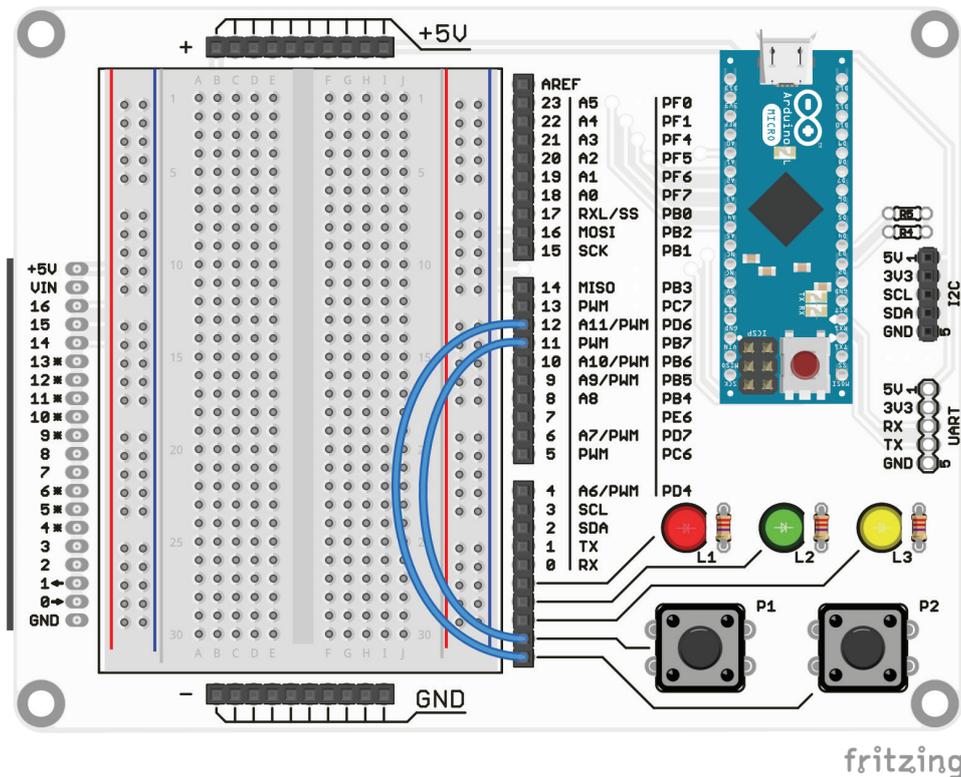
Cycle part:

- Similarly to the previous task two positions of the buttons are defined – **Buttons_Position_1** & **Buttons_Position_2** which will accordingly keep MC legs No11 and No12 position for one cycle.
- We add a condition which allows changing the brightness of the LED to 5 or -5 according to which button is pushed. If none of the buttons are pushed the brightness doesn't change.
- After this we change the brightness of the LED according to the set increase.

- Extra check-ups need to be set up so the new brightness value would be kept in the range of the set interval.
- At the end of the cycle there is a small delay so we could see the change in brightness.

Interesting! For those who think this task is too simple, you can add to the program to make the brightness stop changing when both buttons are pushed.

LEDs brightness



3.4. WORKSHEET. Changing LEDs blinking speed

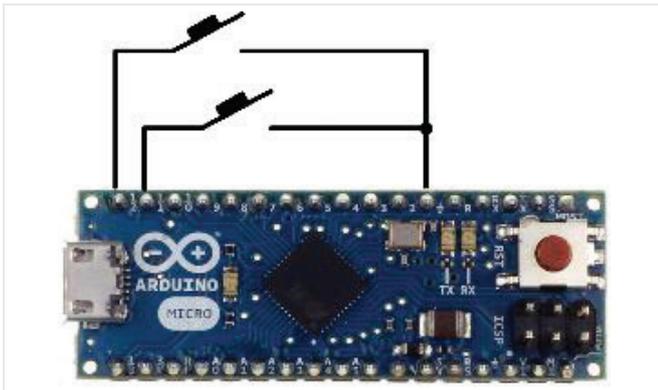
Aim

Create a program which allows to regulate the speed of LEDs brightness change using two buttons where one makes the speed increase, but the other slows it.

Materials needed

Material/part	Amount
Mounting cord	2

Scheme to be created



This scheme allows creating circuit that ensures that the LED shines when the button is pushed. Reaction to the pressure on the button is ensured by MC Arduino, unlike the examples shown in chapter about basics of electronics. In the constructor the buttons **P1** and **P2** are connected to collective output – 0 pole. That means that the scheme can be simplified as it is shown in the image “Needed materials and circuit”, there’s no need to use extra details.

Steps of work

1. Create the scheme of the circuit.
2. Write the given program.
3. Upload the program to MC Arduino memory as it was shown in chapter for preparing programming environment.
4. Enjoy your brightness changer at work.

```
int ledLeg = 13;
int time = 500;

void setup()
{
  pinMode(ledLeg, OUTPUT);
  pinMode(11, INPUT);
  pinMode(12, INPUT);
}

void loop()
{
  bool ButtonPosition_1 = digitalRead(12);
  // read position of MC leg No12
  bool ButtonPosition_2 = digitalRead(11);
  // read position of MC leg No11
  if (ButtonPosition_1 == true)
  { time = time + 50; }
  if (ButtonPosition_2 == true)
  { time = time - 50; }
  if (time >= 1000) { time = 1000; }
  if (time < 0) { time = 0; }
  digitalWrite(ledLeg, HIGH);
  delay(time);
  digitalWrite(ledLeg, LOW);
  delay(time);
}
```

Short explanation:

Global variables:

- Two variables are defined:
 - ledLeg - shows to LED built-in into the MC.
 - time - time in milliseconds with which the position of the LED is changed.

Initialization part:

- Inputs and output are determined.

Cycle part:

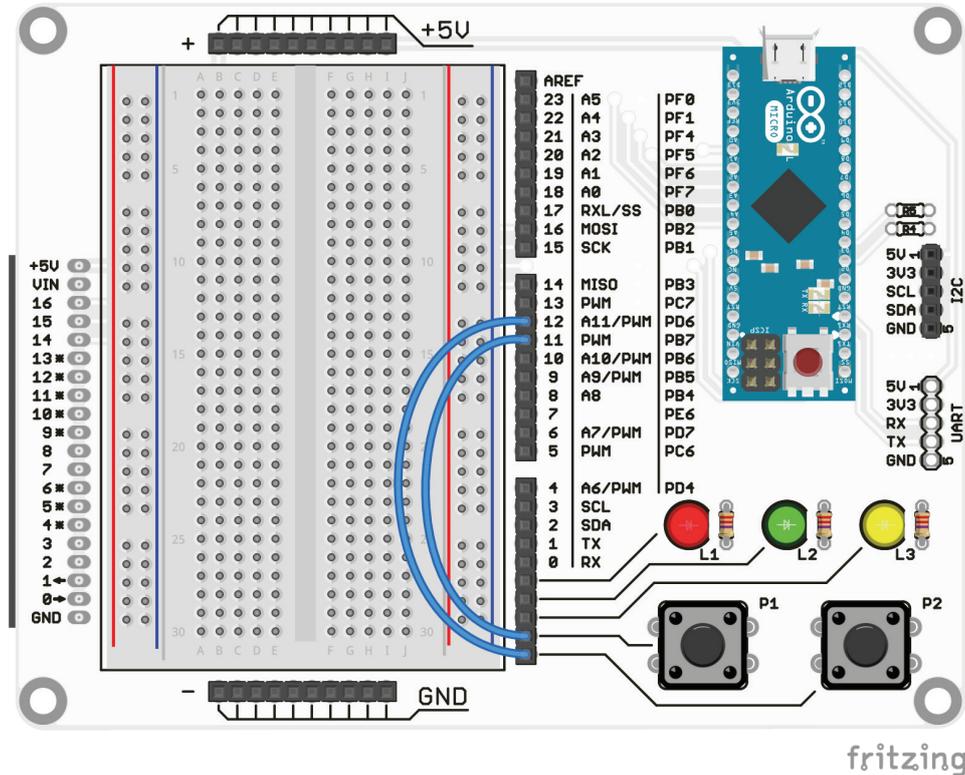
- Two positions of the buttons are defined – **Buttons_Position_1** and **Buttons_Position_2** which will accordingly keep MC legs No11 and No12 position for one cycle.

- A condition which allows regulating the intervals of turning on and off the LED is added. The intervals with a step of 50 milliseconds.
- Extra check-ups need to be set up so the new value of time is kept in the interval of 0 to 1000 milliseconds.

- After that the LED is turned on or off with the new delay value.

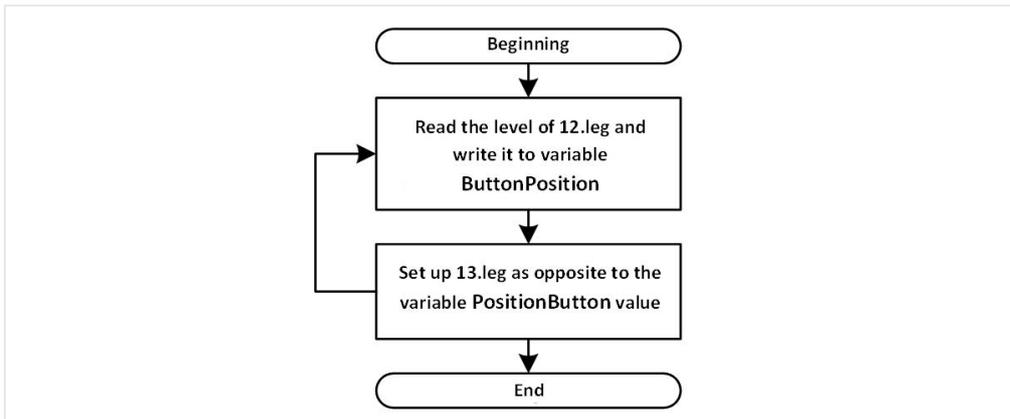
Interesting! If the task seems simple you can add to the program so the blinking speed doesn't change when both buttons are pushed.

Changing the blinking speed of the LED

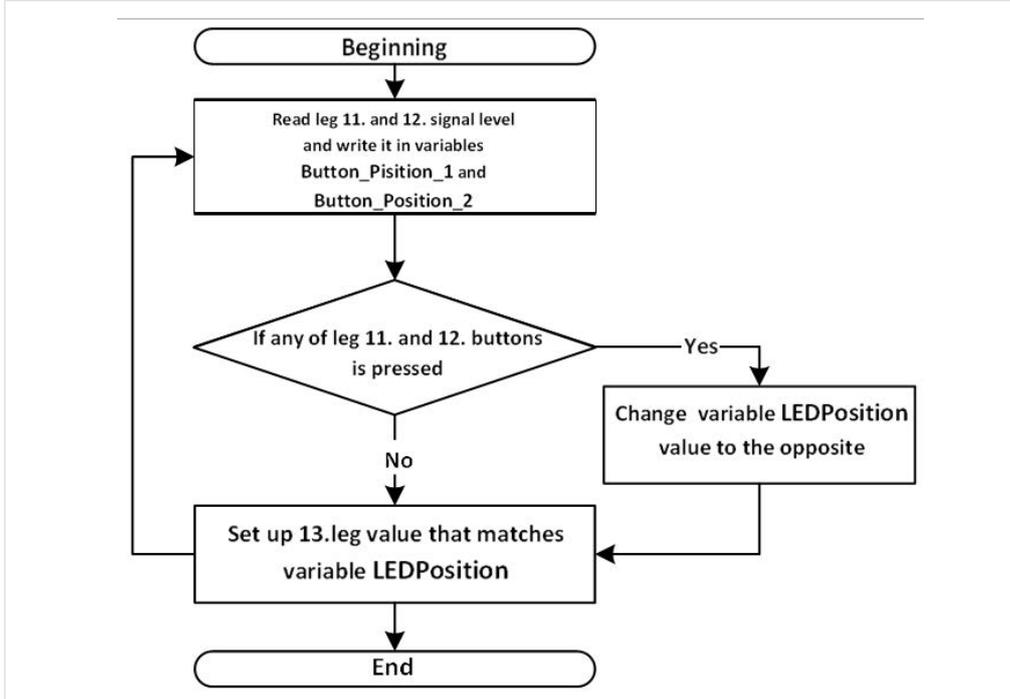


3rd chapters annex

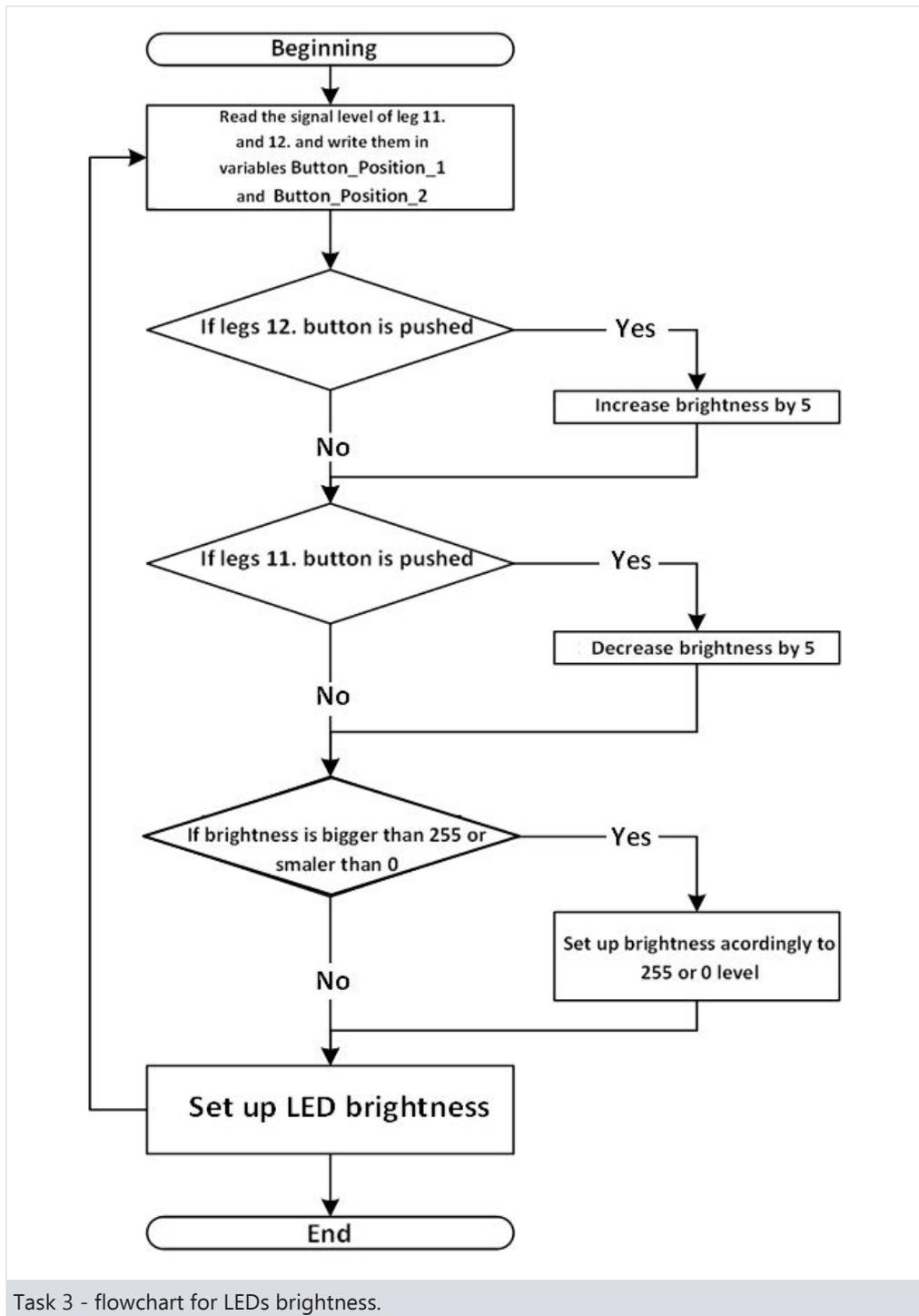
You can look at flowcharts of the programs from this chapter.

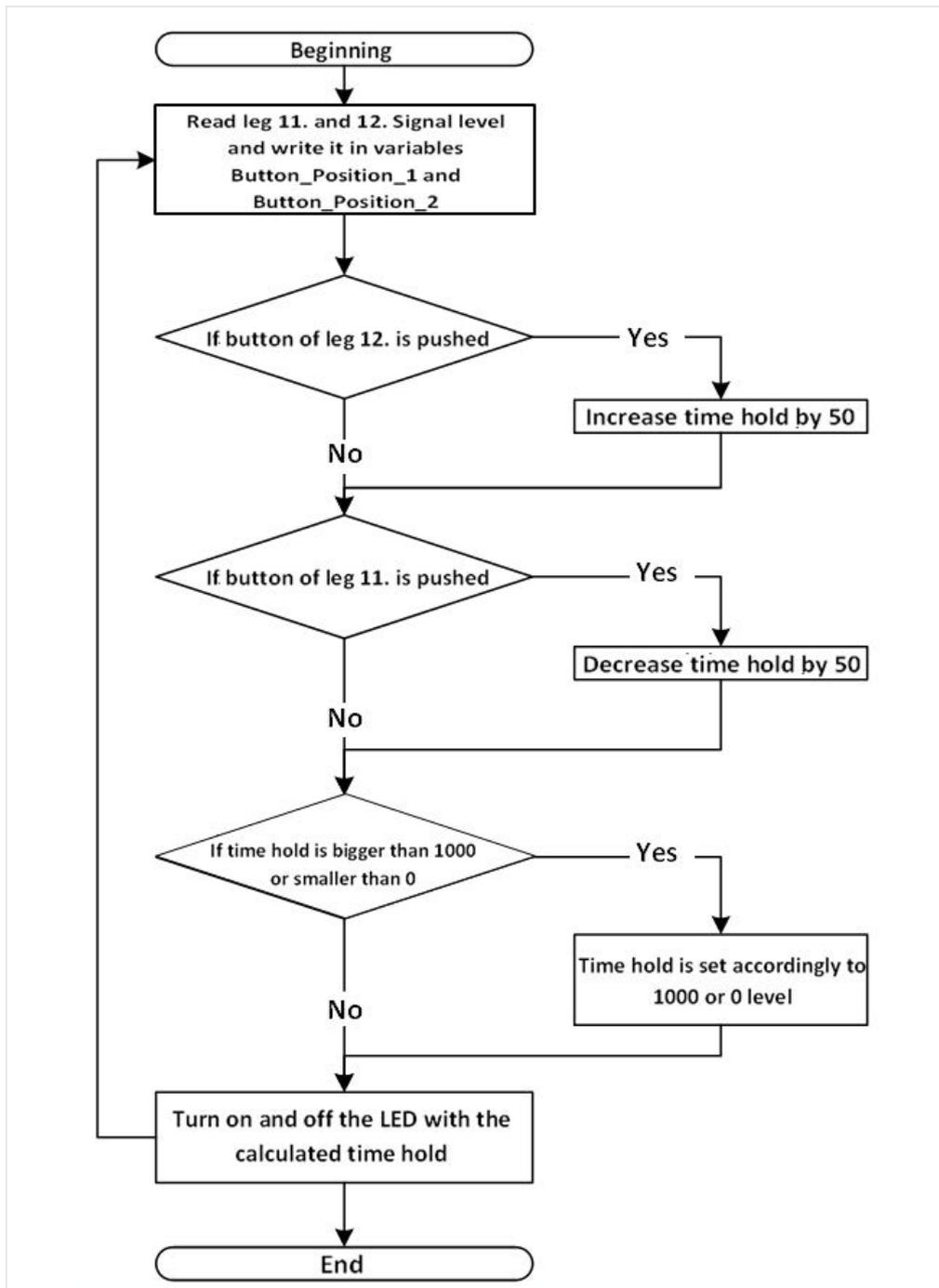


Task 1 - flowchart for turning on the LED.



Task 2 - flowchart for staircase lighting system.





Task 4 - flowchart for change in blinking speed of the LED.

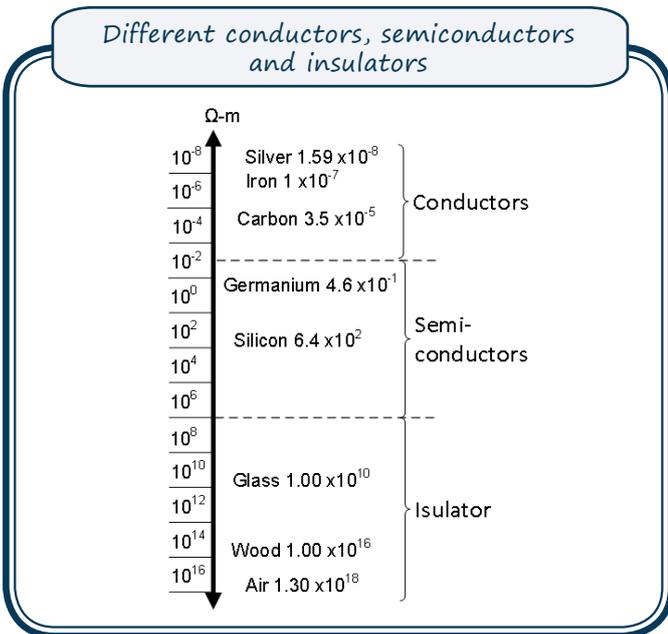
4. SEMICONDUCTORS

4.1. Aim

The aim of this topic is to be introduced to the most used elements of semiconductors in electronics which ensure the execution of different tasks. For example, semiconductors allow creating an electronic switch which then serves a specific function like turning the device on and off.

4.2. Theoretical part

Semiconductors are not especially good conductor of current and not especially good insulator, this is where the name semiconductor comes from. Semiconductors are Silicon (Si) and Germanium (Ge).

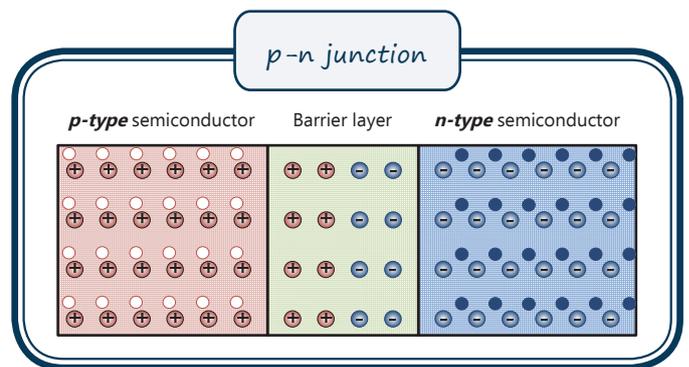


To increase semiconductors ability to conduct electricity they are mixed with different admixtures - donors and acceptors. If we add donors we will get negatively charged *n-type* semiconductor which will have a lot of free electrons. If we add acceptors we will get positively charged *p-type* semiconductor with a lot of free holes which can be filled by electrons.

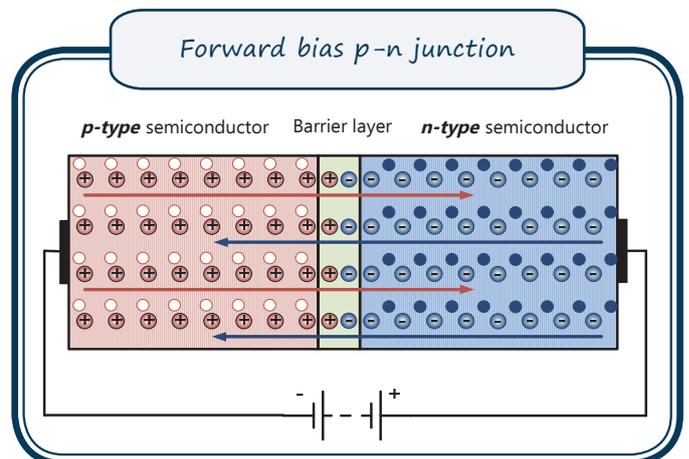
n and *p* type semiconductors are used to create electronic elements like diodes, transistors, sun batteries, microchips and others. Most of the modern devices couldn't exist without semiconductors.

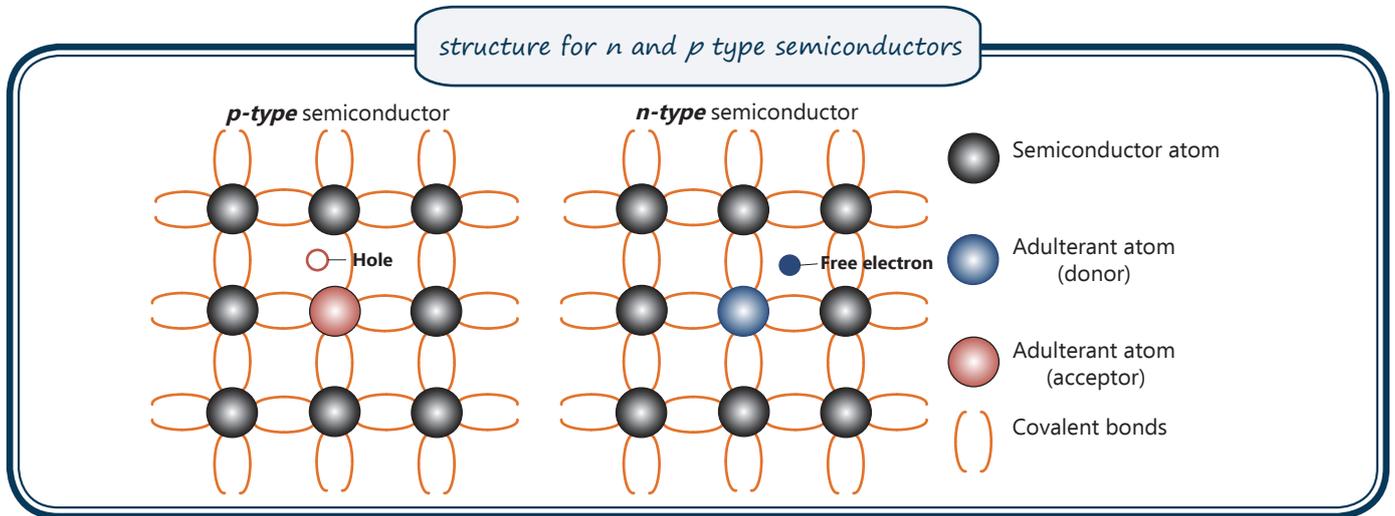
4.2.1. p-n junction

By connecting *n* and *p* type semiconductors we create *p-n* junction. As we previously found out *n-type* semiconductors have a lot of electrons and *p-type* semiconductors have a lot of holes. By connecting *p* and *n* type semiconductors the electrons from *n* region start to move to the holes in *p* region, neutralizing each other. This process continues until a barrier layer is formed at the point where the two materials collide, preventing electrons from entering the holes.

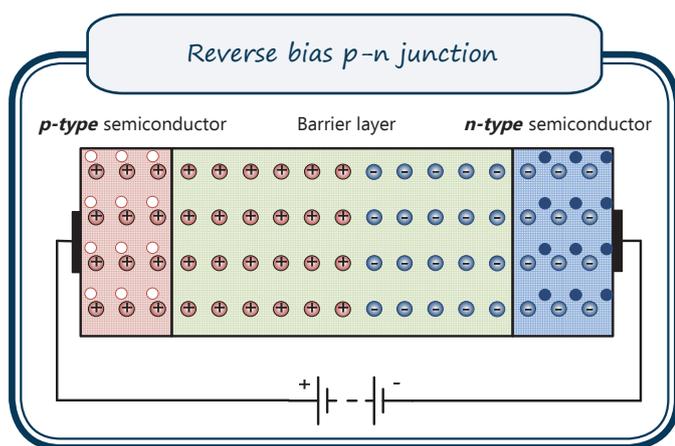


If we add an additional source of voltage to the *p-n* junction where *p-type* is connected to the positive end and *n-type* is connected to the negative end and the voltage will be at least 0.7 V then the barrier layer will decrease and electrons will start to flow through the *p-n* junction. This is called the forward bias *p-n* junction.

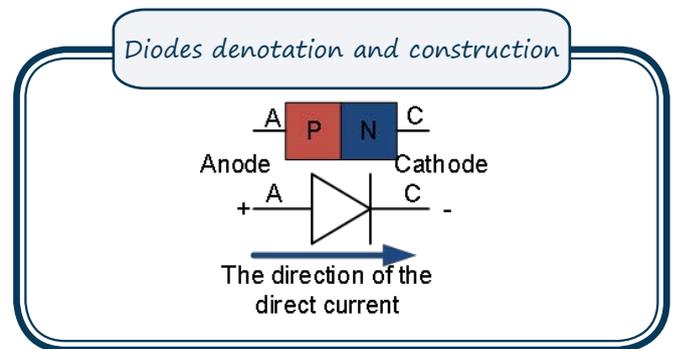




If the added voltage will be connected to the *p-n* junction in a way where *p-type* is connected to the negative end and *n-type* is connected to the positive end then the barrier layer will increase and the current won't flow through this layer. This is called the reverse bias *p-n* junction.



Denotation	Schemes symbol
	Diode is indicated by an arrow with a straight line at the end of it. The arrow indicates in which direction does the diode conduct current.



By connecting the diode in reverse bias where the cathode is more positive than anode the current won't flow through the diode. Every diode in meant to withstand a certain voltage in reverse bias. If the voltage will be exceeded then the diode will be passed it will start conducting current and will be damaged.

4.3. Diodes

4.3.1. Diodes properties

Simpler devise which can be created by putting together *p* and *n* type semiconductors is a diode. Its main properties are that it can let the current go through in only one direction.

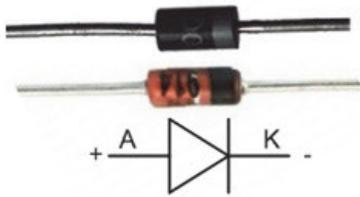
Diode starts conducting current when its anode is at least 0.7 V more positive than its cathode. That means, if the voltage is less than 0.7 V the diode won't conduct current. Because diode is the active component it will sustain its voltage at 0.7 V between its terminals while the current flowing through the diode will quickly increase. Diode is meant for certain amount of current, if the meant amount is exceeded then the diode can start to heat up and burn out.

4.3.3. Diodes parameters, types, uses

There are many different diodes types and uses but we will be looking at the simplest signal diodes parameters, types and uses.

Signal diodes can be different; they are used in the same way but their appearance and parameters are different. Most commonly you will see two types of signal diode bodies- black with a grey stripe at the end or red glass with a black stripe at the end. The stripes on diodes body usually show the location of the cathode; with this you can determine in which direction you should connect the diode to the scheme.

Most commonly seen bodies of signal diodes



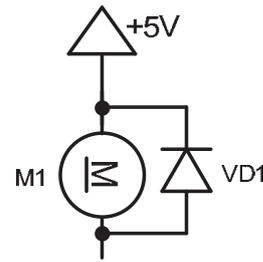
Diodes are different with their parameters. Diodes parameters can be found out by looking at the manufacturers technical specifications. Manufacturer of the device has created a datasheet for each electrical component; in it the manufacturer describes all that the user of the device needs to know. To find the diodes datasheet you have to know the name of the diode; usually it is written in small letters on the body of the diode. You can find diodes datasheet by writing its number in internet browser and adding "datasheet". Most of the datasheets are in English. Diodes are characterised by many different parameters but here we will look at the two most important:

- **Maximum forward current I_F** determines the amperage of the current that flows through the diode. Diode has a small resistance so by Ohm's law it releases power in the form of heat. If the allowed amperage is exceeded then the diode overheats and burns out. Resistors are often used in series with diodes to limit the current.
- **Maximum inverted voltage U_R** determines the voltage allowed to be put on diodes terminals in inverted direction before crossover occurs. If this voltage is exceeded the diode gets crossover starts conducting current and gets damaged.

Diodes uses:

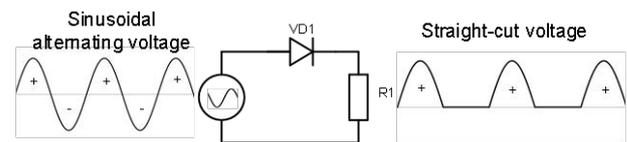
- **Components protection.** Diodes can be used to protect components from different inverted voltages and currents. Diode can be used to protect scheme if the battery is inserted in the wrong way because diode allows the current to flow in only one direction. It protects schemes from elements with inductive elements in them (coil, motor). Coils resist changes in current and, if we turn on a motor it can create dangerously large current which flows in the opposite direction of the normal current. By inserting a diode this current flows through the coil itself and doesn't damage the scheme.

Diode as a protection in motor



- Attaining direct current. Diodes are used to change alternating current into direct current; this process is called the rectification. Alternating current polarity changes regularly. Diode allows it to flow in only one direction rectifying it into direct current.

Alternating currents rectification into direct current

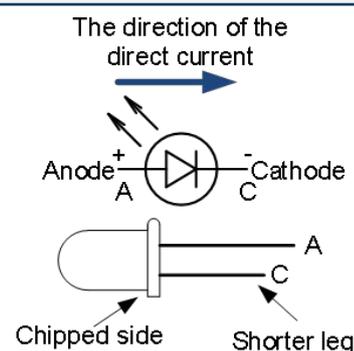


4.4. Light-emitting diodes

In the previous schemes we already looked at light-emitting diodes (LED). LED is a special type of diodes which emits light unlike the other diodes. LED has completely different body which is made of transparent plastic that protects the diode and lets it emit light. Like the other diodes LED conducts the current in only one way so it is very important to connect it to the scheme correctly. There are two safe ways how to determine the direction of the diode:

- In the cathodes side of the diode its side is chipped.
- Anodes leg usually is longer than the cathodes leg.

Determining LEDs cathode and anode



LED is one of the best light sources. Unlike incandescent light bulb LED transforms most of the power into light not warmth; it is more durable, works for a longer period of time and can be manufactured in a smaller size.

LED colour is determined by the semiconductors material. If normally diodes are made from silicon then LEDs are made from elements like gallium phosphate, silicon carbide and others. Because the semiconductors used are different the voltage needed for the LED to shine is also different. In the table you can see with which semiconductor you can get a specific colour and the voltage needed to turn on the LED.

LED parameters for insulators

Diodes parameters			
Semiconductor	Wave length (nm)	Colour	Voltage (V) for LEDs U_D (20 mA)
GaAs	850–940	Infrared	1,2
GaAsP	630–660	Red	1,8
GaAsP	605–620	Orange	2,0
GaAsP:N	585–595	Yellow	2,2
AlGaP	550–570	Green	3,5
SiC	430–505	Blue	3,6
GaInN	450	White	4,0

When LED is connected to voltage and turned on a very large current starts to flow through it and it can damage the diode. That is why all LEDs **have to be connected to current limiting resistor**.

Current limiting resistors resistance is determined by three parameters:

- Current that can flow through the LED - I_D .
- Voltage that is needed to turn on the LED - U_D .
- Combined voltage for LED and resistor - U .

To calculate the resistance needed for a diode this is what you have to do:

1. Find out the voltage needed for the diode to work U_D ; you can find it in the diodes parameters table.
2. Find out the amperage needed for the LED to shine I_D ; it can be found in the LEDs datasheet but if you can't find it then 20 mA current is usually correct and safe choice.
3. Find out the combined voltage for the LED and resistor, usually it is the feeding voltage for the scheme.
4. Insert all the values in this equation:

$$R = \frac{U - U_D}{I_D}$$

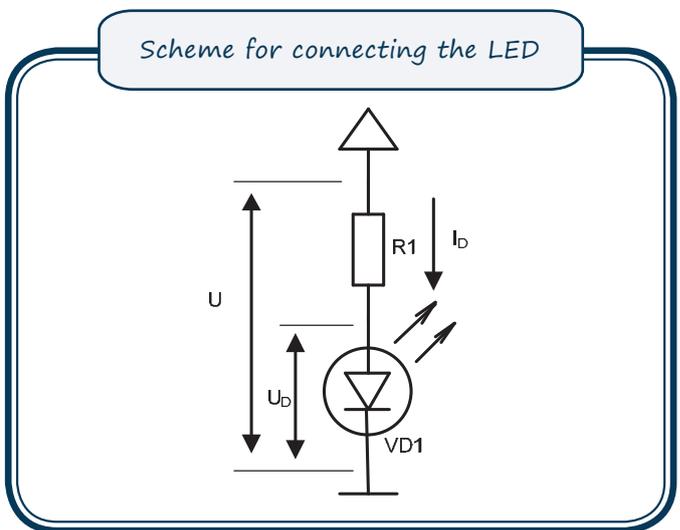
5. You get the resistance for the resistor for safe use of the LED.
6. Find resistors nominal that is the same or bigger than the calculated resistance.

Example

We will use orange LED. Orange LED needs voltage $U_D = 2$ V. Current flowing through the LED will be $I_D = 20$ mA. Voltage for the whole scheme will be $U = 5$ V. insert all the values in the equation:

$$R = \frac{U - U_D}{I_D} = \frac{5 - 2}{0,02} = \frac{3}{0,02} = 150 \Omega$$

To turn on an orange LED you need resistor with 150Ω or bigger resistance.

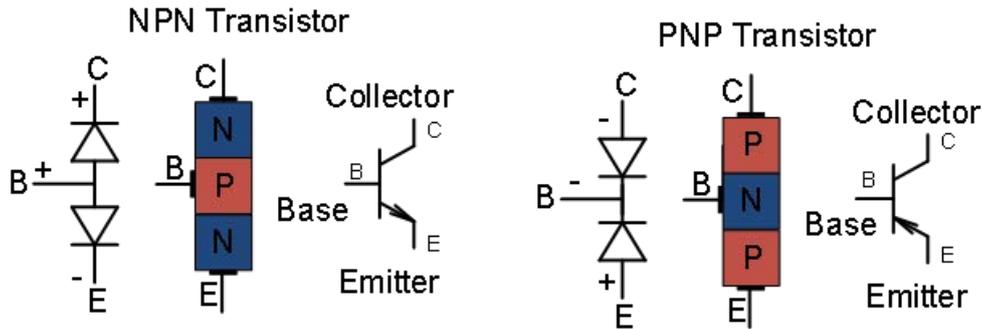


4.5. Transistor

Bipolar transistor is created by connecting two $p-n$ junctions. Transistor usually is a device with three terminals: collector (C), base (B) and emitter (E). Transistors are used in two ways: as non-contact switch and as voltage and currents amplifier.

Apzīmējums	Shēmas simbols
	NPN transistor – emitter arrow is in the direction away from the base; it shows that the signal's current flows from base to emitter.
	PNP transistor – emitter arrow is in the direction towards the base; it shows that the signal's current flows from emitter to base.

Comparison of PNP and NPN transistors construction



4.5.1. Transistors construction

Transistor is made by adding an extra semiconductor p-n layer to junction's diode. You can think of a transistor as two diodes put together. Bipolar transistor is constructed from two *p-n* junctions creating three terminals: collector, base and emitter. *p-n* junction can be put together in two ways: *n-p-n* and *p-n-p*, that creates two types of transistors - NPN and PNP. Both transistors work in the same way; the difference is with connecting the transistor to the scheme and its feeding polarities.

4.5.2. PNP and NPN operations

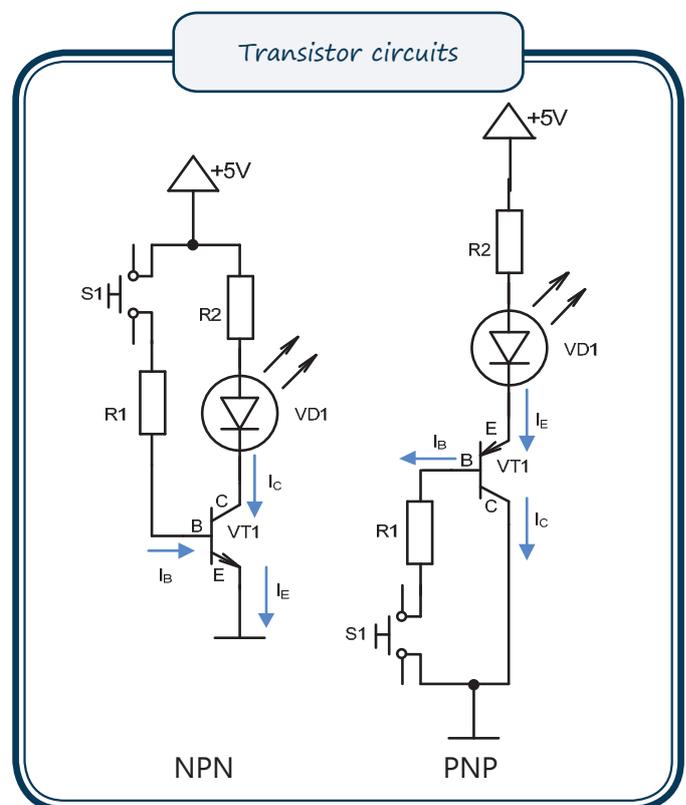
Transistor is a device which is managed with current. The amount of the current at transistors base determines its position. If transistors base has no current it will be turned off. If transistors base has a small current, then it will be turned on and work as an amplifier. The stronger the current at the base the stronger will be the current flowing through collector and emitter. If the current at the base is strong enough the transistor will be completely open and the current between the collector and emitter will flow in the same way as in a turned on switch.

Transistor can be:

- Turned off - there is no currents between collector and emitter.
- Active - transistor is working as an amplifier.
- Saturated - transistor is completely opened and works as a turned on switch.

For turning on the NPN transistor the base has to have a more positive voltage than the emitter; that is why the base has to be connected to positive voltage. With PNP transistors it is the opposite - emitter has to have more positive voltage than base so the emitter has to be connected to the positive terminal but the base to the negative.

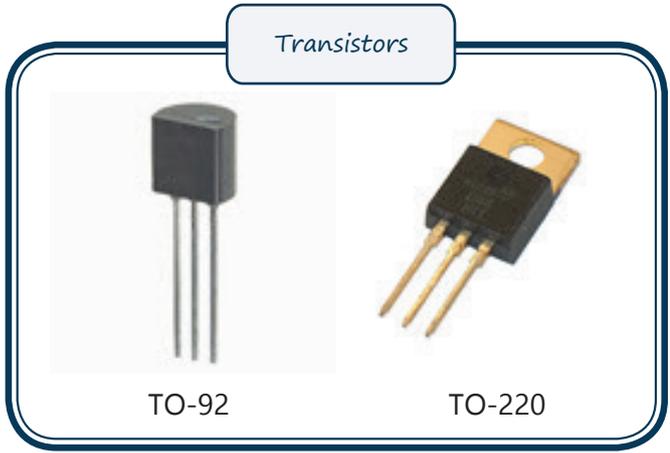
This also changes the currents direction accordingly. Transistors base can have only very small current flowing through it so **it is important to use current limiting resistor R1 so the transistor wouldn't be damaged.**



4.5.3. Transistors body and parameters

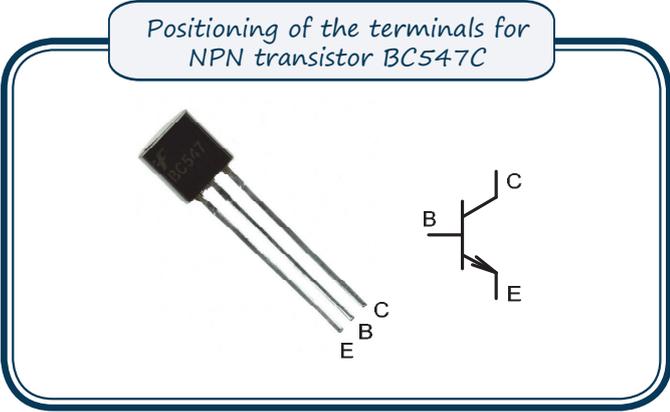
The same as with diodes, transistors also have different parameters. One of the most popular is the TO-92 and TO-220. You can see that these bodies are different with their size and appearance. The size of the body is determined by the need to disperse power so the transistor wouldn't overheat and burn out while in use. The bigger the body of the transistor the more power can be dispersed and stronger current can be conducted.

Transistors



While using transistor it is important to know the position of its terminals (collector, base, and emitter). However, it can be different for each resistor so it is important to look at the transistor data. The same as with diodes transistor's name can be found on its body and by writing it in the internet browser you can find its datasheet where you will see the positioning of its terminals.

Positioning of the terminals for NPN transistor BC547C



While using a transistor it's important to know a couple of parameters that determine transistors operations:

1. **Maximum voltage for collector and emitter** U_{CE0} - determines how strong voltage can be used while using the transistor for it to not be damaged.
2. **Maximum collectors current** I_C - determines how strong current can flow through the transistor for it to not be damaged. After exceeding this amount the transistor overheats and burns out.
3. **Currents amplification coefficient** β - determines by how many times will the collectors current change when the base current is changed $I_C = I_B \beta$.
4. **Maximum voltage for the collector and base** U_{CBO} - determines how strong base current I_B flows through the transistor; to not damage the transistor this value should be kept under the maximum.

Transistors longevity will be ensured by following the maximum parameters set by the manufacturer.

4.1. WORKSHEET. Diodes circuit

Aim

Novērot diodes pamatīpašību – strāvas laišanu vienā virzienā.

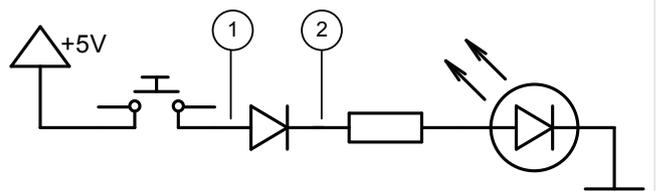
Steps of work

1. Connect the circuit shown in the image “Diode connected in the direction of the current”.
2. Connect the circuit to a source of voltage and push the button.
3. See if the light-emitting diode shines.
4. Measure the voltage on the diode and compare it with the voltage that theoretically should be on the diode. How does it differ?
5. Connect the circuit shown in the image “Diode connected in the opposite direction of the current” (diode changes its direction).
6. Connect the circuit to a source of voltage and push the button.
7. See if the light-emitting diode doesn't shine.

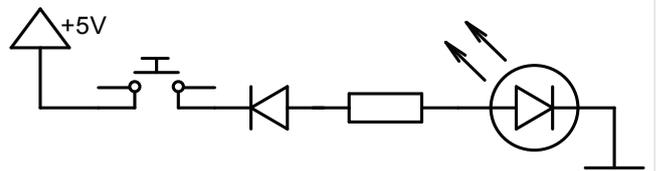
Materials needed

Material/part	Amount
Button	1
Diode PH4148	1
Resistor (330Ω)	1
Light-emitting diode	1
Mounting cord	3

Schemes to be created

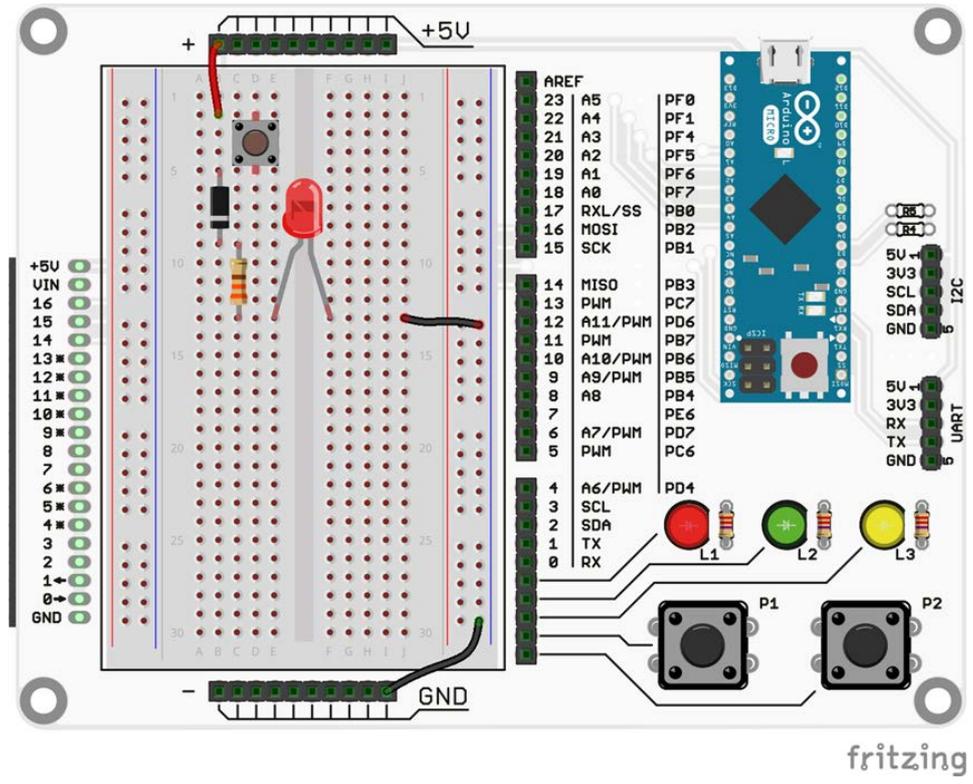


Elements in series circuit with diode in the direction of the current

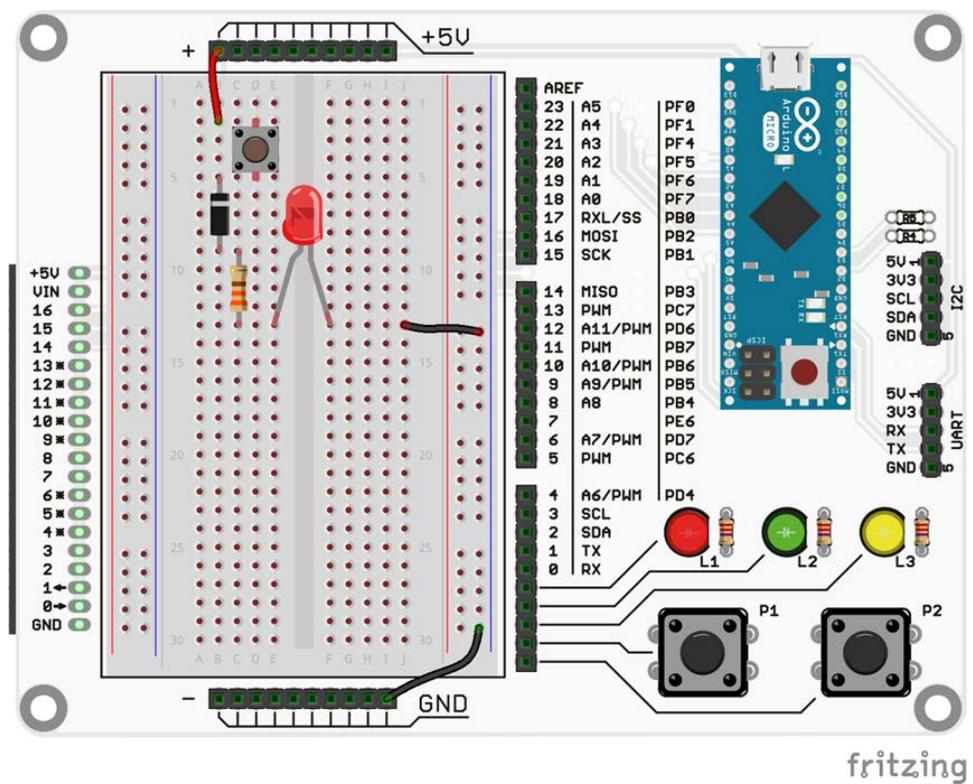


Elements in series circuit with diode in the opposite direction of the current

Diode connected in the direction of the current



Diode connected in the opposite direction of the current



4.2. WORKSHEET. Light-emitting diodes voltage

Aim

Observe the difference between different light-emitting diodes.

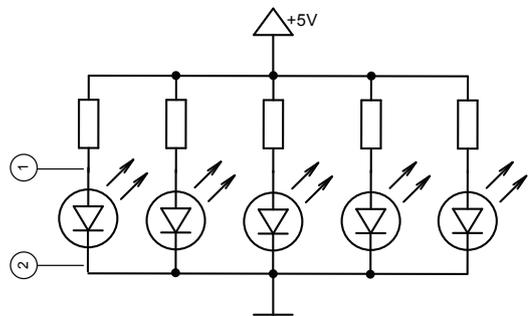
Steps of work

1. Connect the circuit shown in the image.
2. Connect the circuit to a source of voltage.
3. See if every diode shines in different brightness.
4. Using multimeter for measuring direct current and measure voltage for every diode in the points 1 and 2.
5. Compare with theoretical voltages of LED. How do they differ?

Materials needed

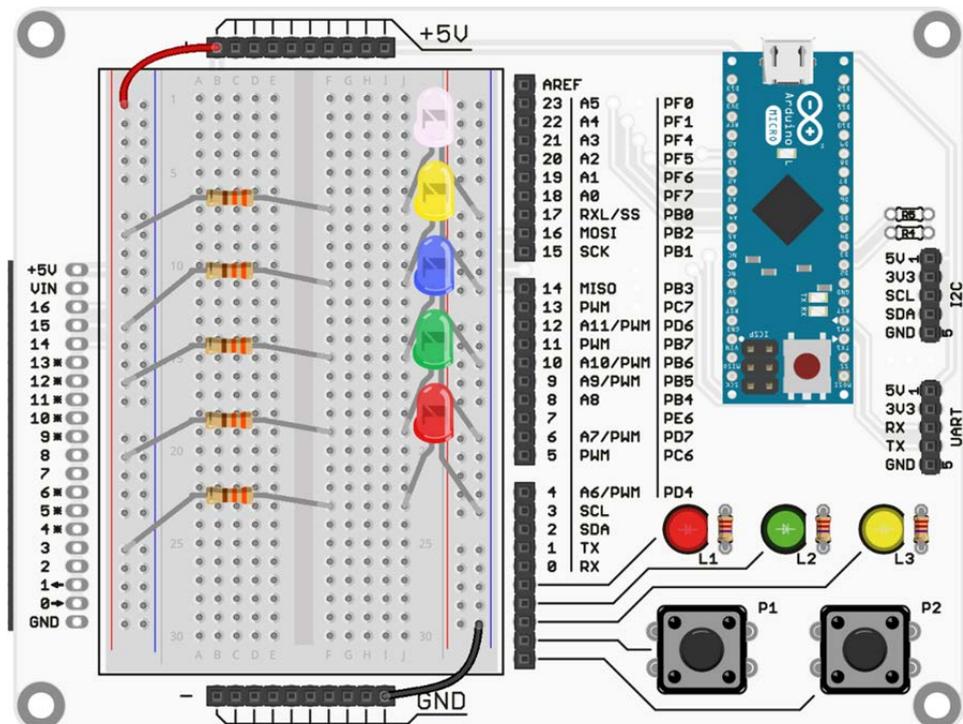
Material/part	Amount
Resistor (330Ω)	5
LED – white, yellow, blue, green, red	5
Mounting cord	2

Scheme to be created



Parallel circuit with multi coloured LEDs

LEDs voltage



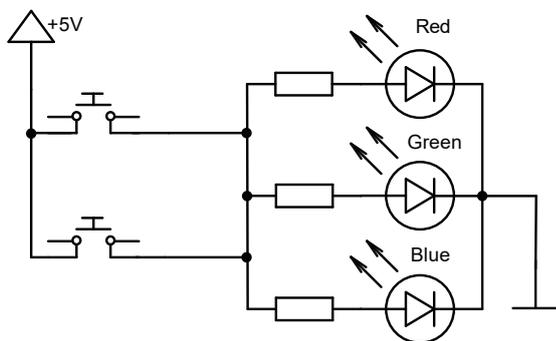
fritzing

4.3. WORKSHEET. RGB Light-emitting diode

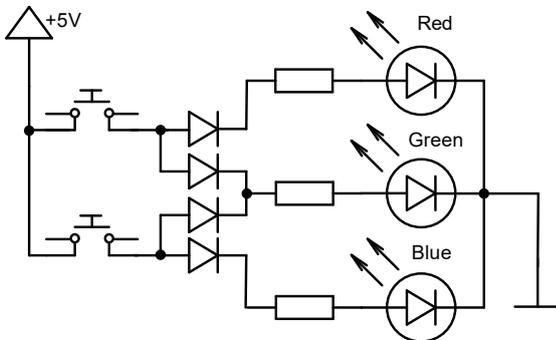
Aim

Learn RGB LED use and main properties of diodes.

Scheme to be created



RGB Diode will shine in white colour no matter which button is pushed



Diode will shine in either yellow or light blue colour depending on which button is pushed.

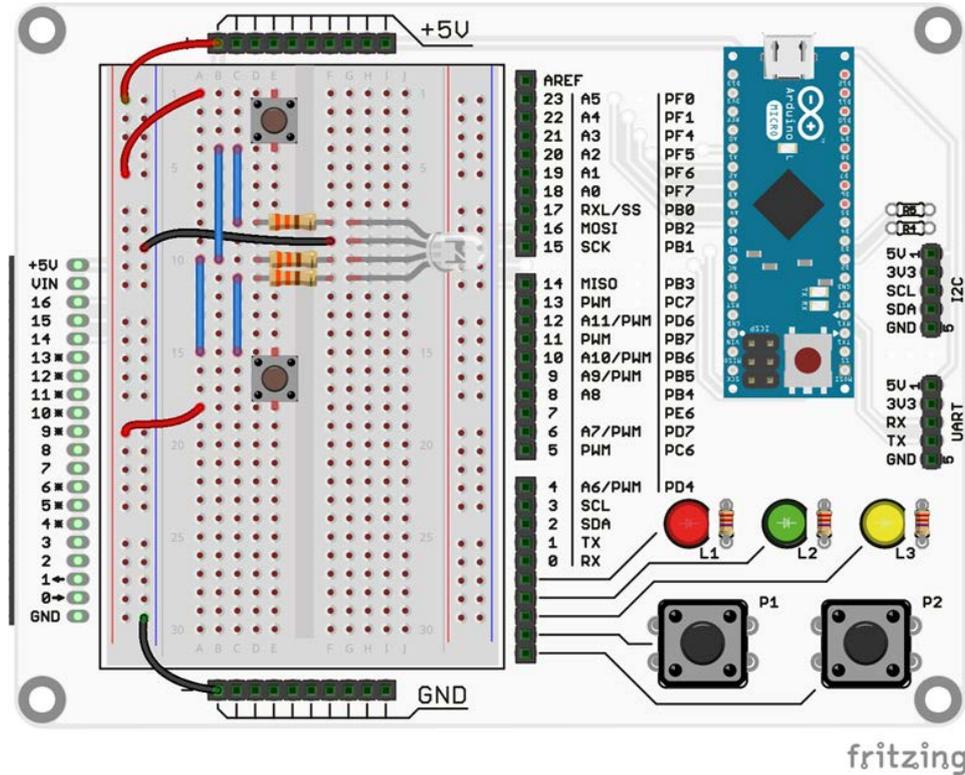
Materials needed

Materials/detaja	Skaits
Diode PH4148	4
Button	2
Resistor (330Ω)	3
RGB LED	1
Mounting cord	As much as needed

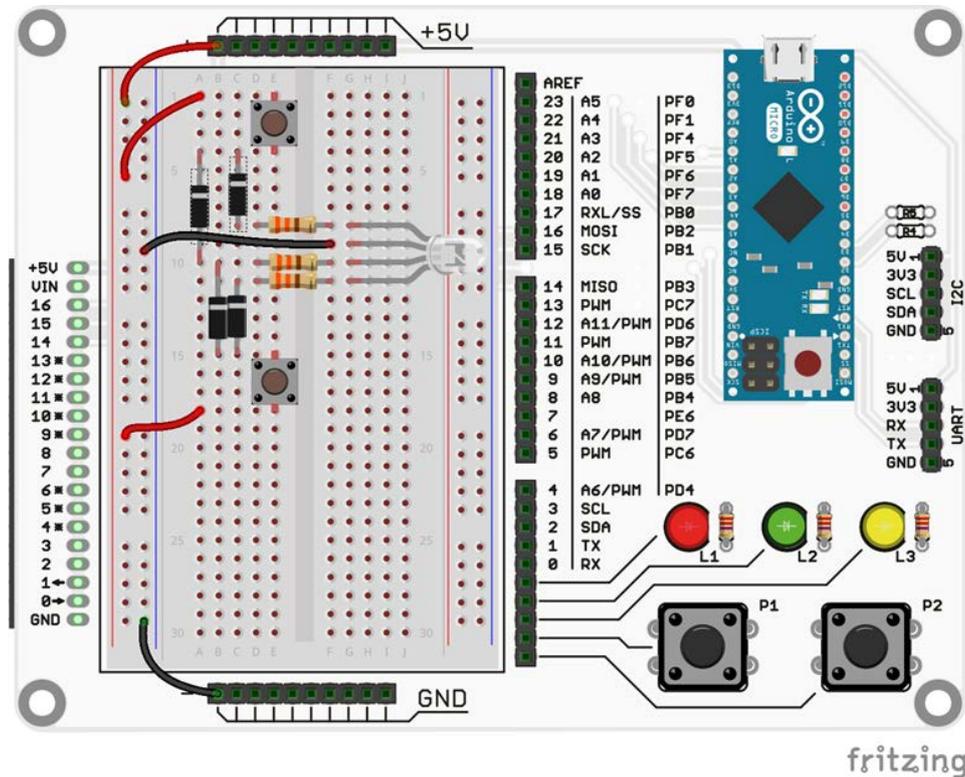
Steps of work

1. Connect the circuit shown in the image "RGB diode's managing without diodes".
2. Connect the circuit to a source of voltage.
3. Push one of the buttons and then the other button.
4. See if the RGB diode shines in white colour after pushing both buttons.
5. Connect the circuit shown in the image below the previous.
6. Connect the circuit to a source of voltage.
7. Push one of the buttons and then the other button.
8. See if the RGB diode shines in yellow colour after pushing the first button and in light blue colour after pushing the other button.
9. Pushing both buttons at the same time will create white colour.

RGB diode's managing without diode



RGB diode's managing with diodes



4.4.

WORKSHEET. RGB Light-emitting diodes operating with program

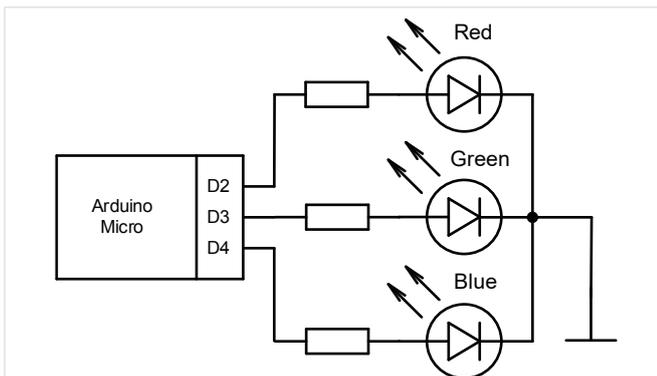
Aim

Learn how to operate RGB diode with program.

Materials needed

Material/part	Amount
Resistor (330Ω)	3
RGB LED	1
Mounting cord	As much as needed

Scheme to be created



Scheme to operate RGB diode with program

Steps of work

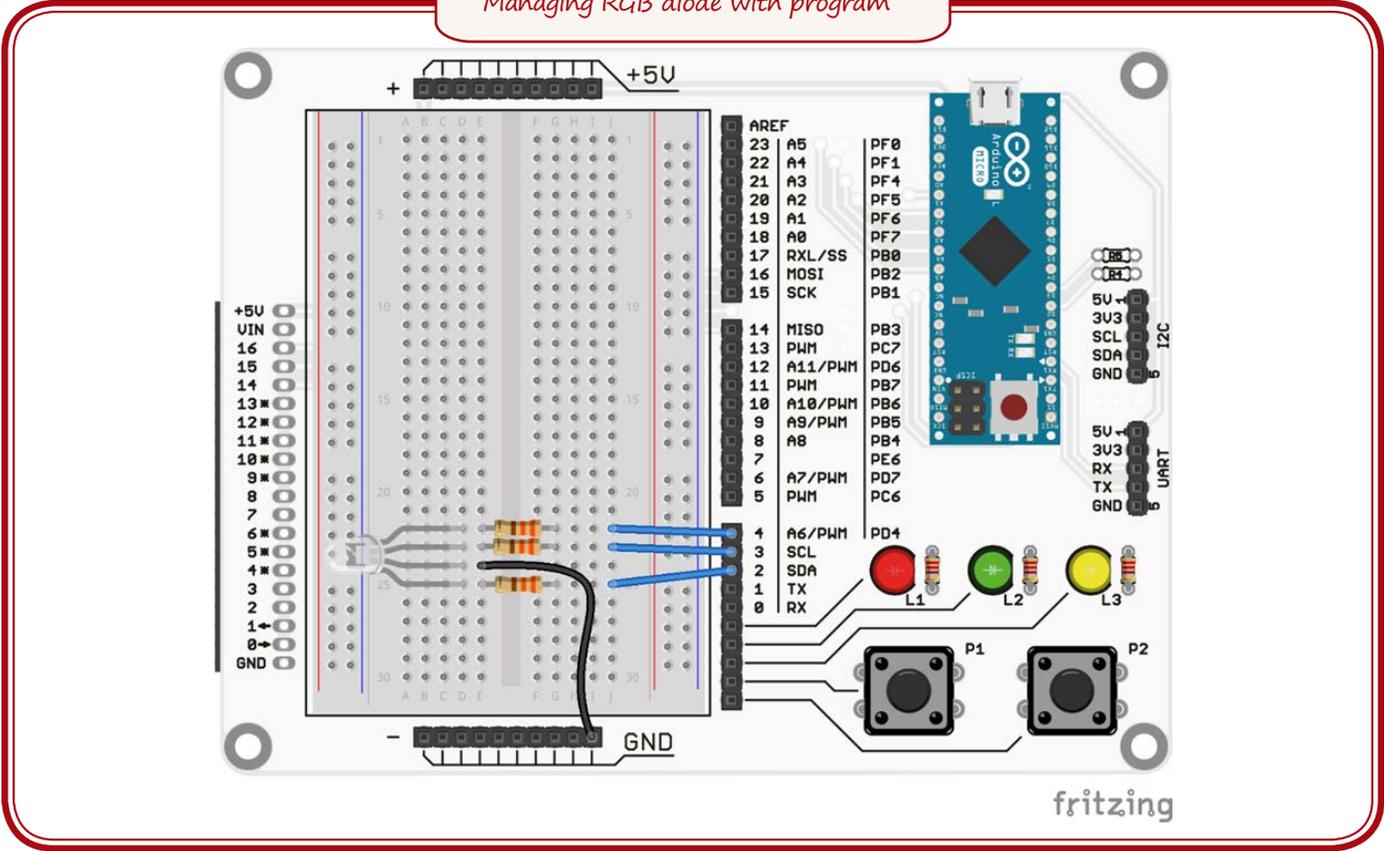
1. Connect the circuit shown in the image.
2. Write the given operating program.
3. Connect the circuit to a source of voltage.
4. Upload the program to Arduino memory.
5. See how after every half a second the diodes colour changes.

```
void setup ()
{
  pinMode(4,OUTPUT)
  //set up MC leg 4 as output
  #define greenLEDON digitalWrite(4,HIGH)
  //define how to turn on green diode
  #define greenLEDOFF digitalWrite(4,LOW)
  // define how to turn off green diode
  pinMode(3,OUTPUT);
  #define blueLEDON digitalWrite(3,HIGH)
  #define blueLEDOFF digitalWrite(3,LOW)
  pinMode(2,OUTPUT);
  #define redLEDON digitalWrite(2,HIGH)
  #define redLEDOFF digitalWrite(2,LOW)
}

void RGBColor (boolean red,boolean green,
                boolean blue)
//create function for operating RGB diode
{
  if (red){redLEDON;} else {redLEDOFF;}
  //if red == 1, then turn on the red
  //diode, if not diode turns off
  if (green){greenLEDON;} else {greenLEDOFF;}
  if (blue){blueLEDON;} else {blueLEDOFF;}
}

void loop ()
{
  //turn on 7 colours for RGB diode
  RGBColor(1,0,0);
  //Red
  delay(500); //wait for half a second
  RGBColor(0,1,0);
  //Green
  delay(500);
  RGBColor(0,0,1);
  //Blue
  delay(500);
  RGBColor(1,1,0);
  //Yellow
  delay(500);
  RGBColor(0,1,1);
  //Light Blue
  delay(500);
  RGBColor(1,0,1);
  //Violet
  delay(500);
  RGBColor(1,1,1);
  //White
  delay(500);
}
```

Managing RGB diode with program



4.5. WORKSHEET. Transistors usage

Aim

Learn transistors operations.

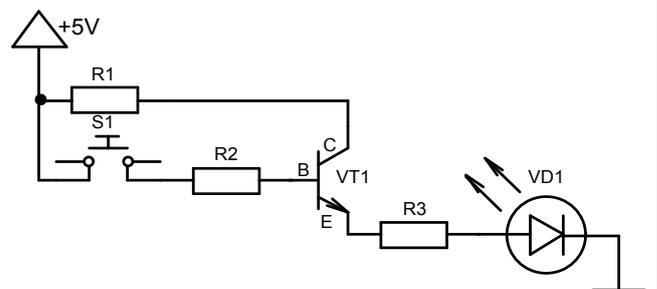
Steps of work

1. Connect circuit shown in the image “Turning on the transistor with a button”.
2. Connect circuit to a source of voltage.
3. Push the button.
4. See if the LED turns on.
5. Connect the circuit shown in the image below.
6. Connect circuit to source of power.
7. **DO NOT connect the wires under no circumstances.**
8. Put a finger to each wire.
9. See if the diode lightly shines.

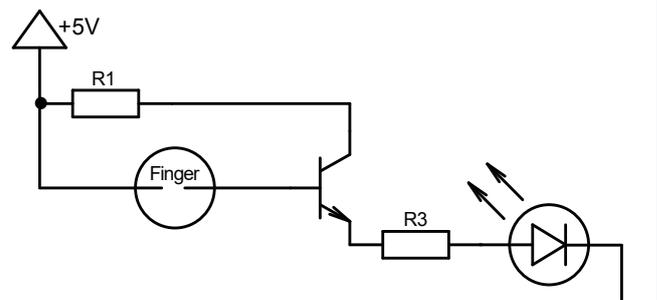
Materials needed

Material/part	Amount
Button	1
Resistor R2 (10 k Ω)	1
Resistor R1 (220 Ω)	1
LED	1
NPN transistor BC517	1
Mounting cord	As much as needed

Scheme to be created

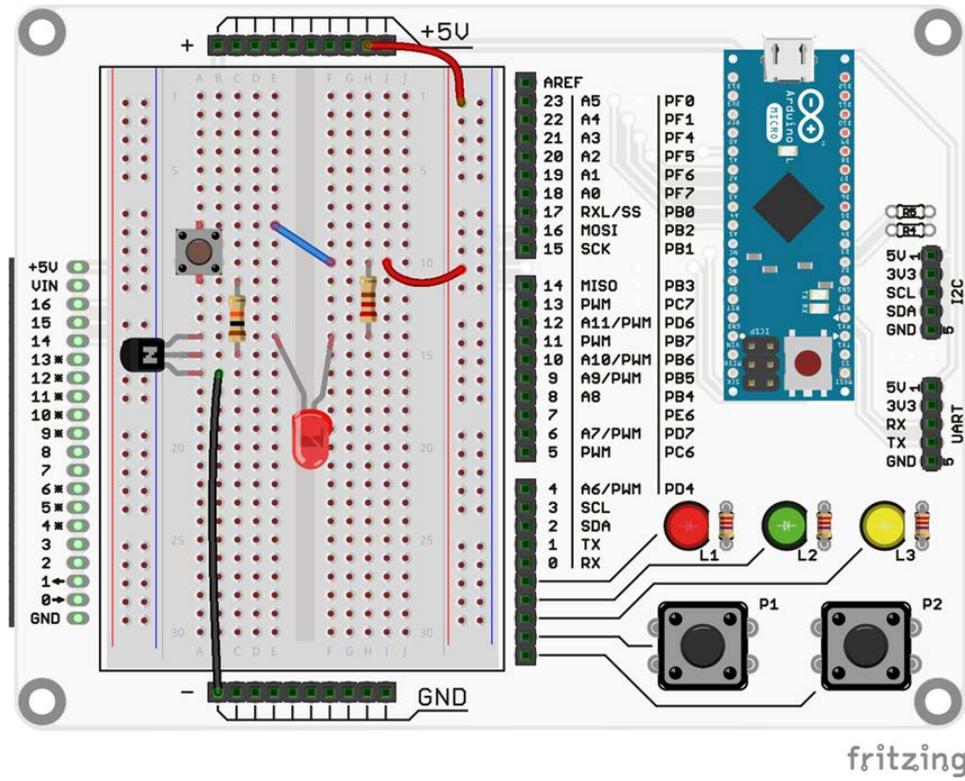


After pushing the button transistor will turn on and the LED will start to shine

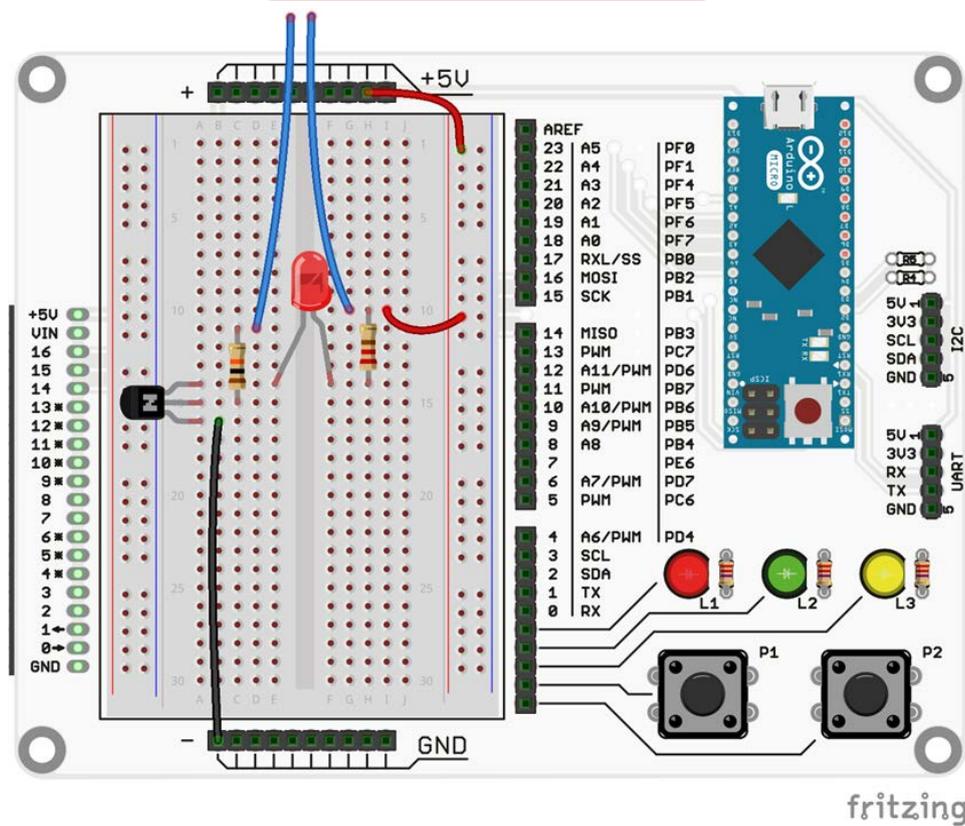


Putting finger at the place shown the LED will start to shine

Turning on the transistor with a button



Turning on the transistor with a finger



4.6. WORKSHEET. Multivibrator

Aim

Learn about interactions of different components.

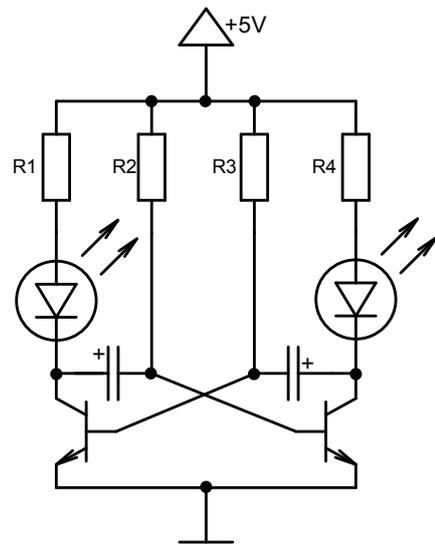
Materials needed

Material/part	Amount
Resistor R1, R4 (330 Ω)	2
Resistor R2, R3 (10 k Ω)	2
NPN transistor BC517	2
LED	2
Capacitor (100 μ F)	2
Mounting cord	As much as needed

Steps of work

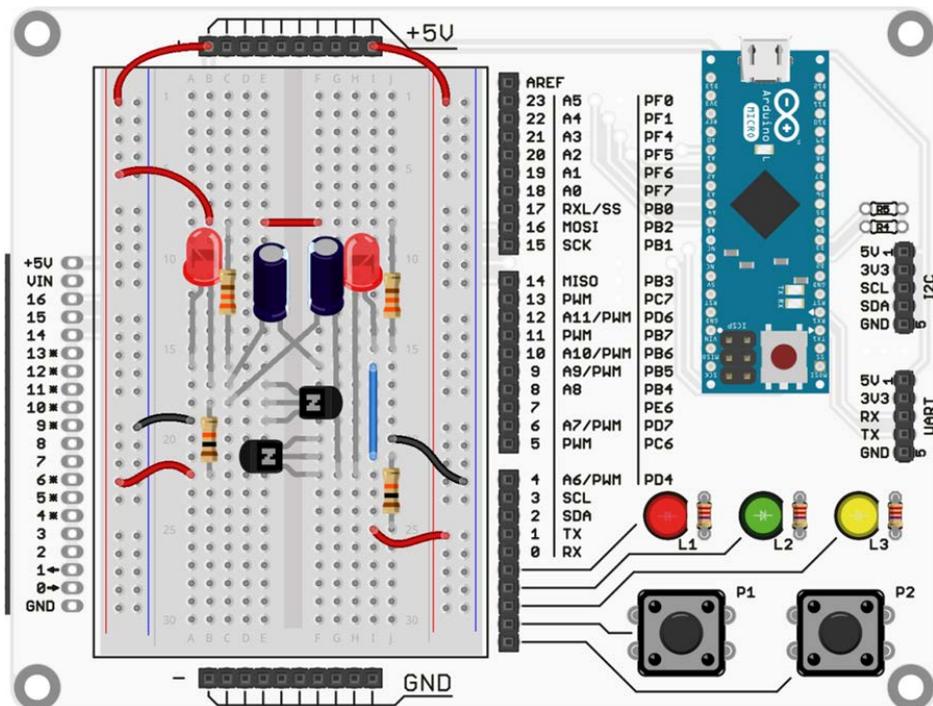
1. Connect the circuit shown in the image "Multivibrators circuit".
2. Connect circuit to a source of voltage.
3. See that the diodes blink alternating.
4. Switch the resistors R2, R3 or capacitor; the blinking speed will change.

Scheme to be created



Correctly connected circuit will create alternating blinking lights

Multivibrator



fritzing

4.7. WORKSHEET. Multivibrator with program

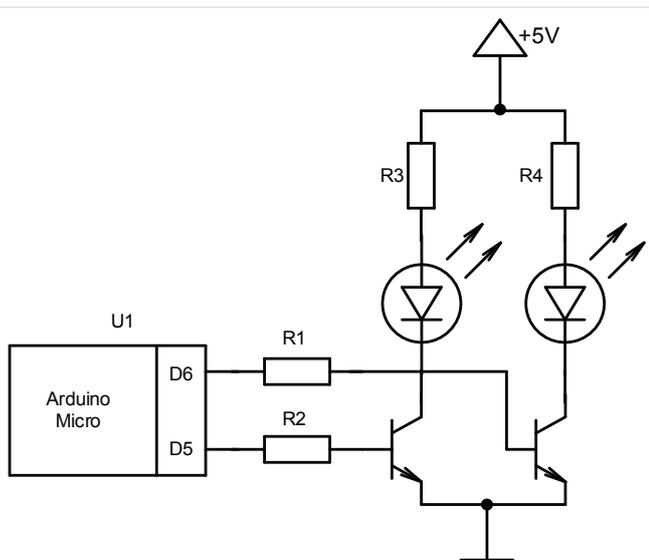
Aim

Learn how to operate transistor with program.

Materials needed

Material/part	Amount
Resistor R3, R4 (330Ω)	2
Resistor R1, R2 (1 kΩ)	2
NPN transistor BC517C	2
LED	2
Mounting cord	As much as needed

Scheme to be created



Scheme how to make multivibrator with program.

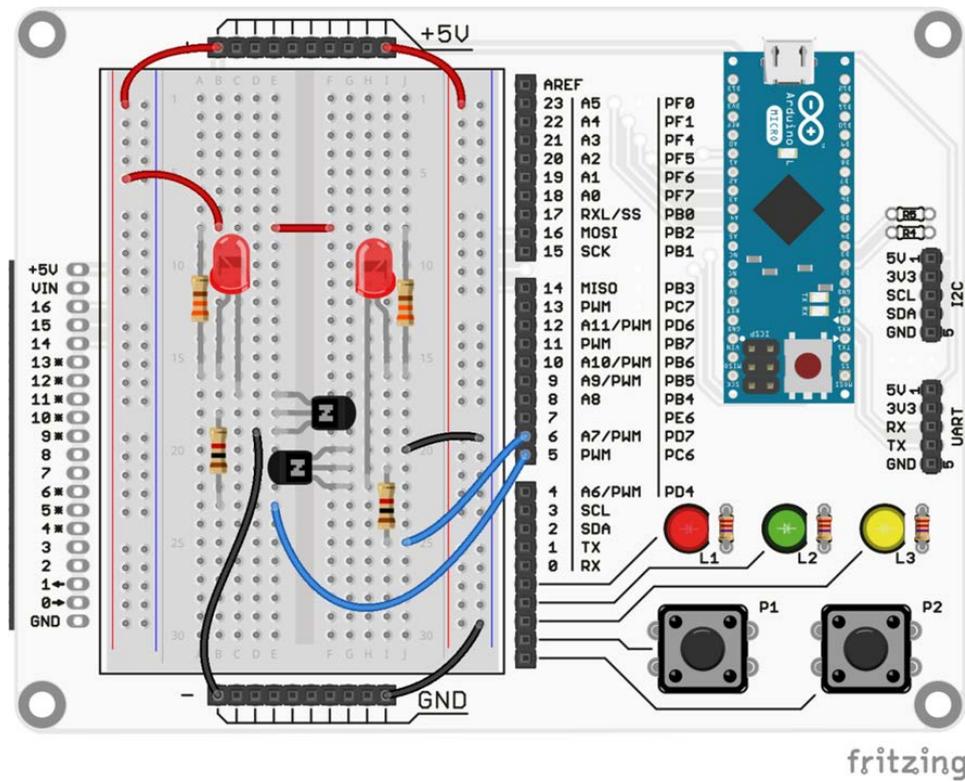
Steps of work

1. Connect circuit shown in the image.
2. Connect circuit to a source of voltage.
3. Write the given program.
4. Upload the program to Arduino and see the alternating blinking lights.

```
void setup ()
{
  pinMode(5,OUTPUT);
  //set MC leg 5 as an output
  digitalWrite(5,HIGH)
  //define how to turn on diode No1
  //make definition for turning on diode No1

  led1OFF digitalWrite(5,LOW)
  //define how to turn off diode No1
  //make definition for turning off diode
  //No1
  pinMode(6,OUTPUT);
  led2ON digitalWrite(6,HIGH)
  led2OFF digitalWrite(6,LOW)
}
void loop ()
{
  led1ON;led2OFF;
  //turn on diode No1 turn off diode No2
  delay(500); //wait for half a second
  led1OFF;led2ON;
  //turn off diode No1 turn on diode No2
  delay(500); //wait half second
}
```

Multivibrator with program



5. SENSORS

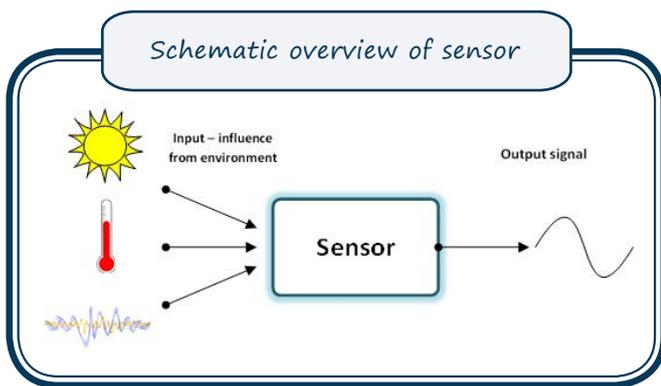
5.1. Aim

Aim for this topic is to give an overall knowledge about sensors, their types, most relevant operating principles and potential uses in robotics.

5.2. Theoretical part

Sensor is an element which can turn a physical outer stimulus into an output signal which then can be used for further analyses, management or decision making. People also use sensors like eyes, ears and skin for gaining information about the outer world and act accordingly to their aims and needs.

From the view of Norbert Wiener, the originator of cybernetics – robot is a manageable system which uses its senses or input and changes its actions accordingly to reach a certain goal. This is why robot's input is made up of one or more sensors which give information used in managing the robot. A schematic overview of sensors operation is shown in the image.



Usually every natural phenomenon – temperature, weight, speed, etc. – needs specially customized sensors which can change every phenomenon into electronic signals that could be used by microprocessors or other devices. Sensors can be divided into many groups according to the physical nature of their operations:

Influence type	Measurement – what can be quantitatively measured
Optical – light influence	Decrease in light intensity – the amount of radiated light an irradiated object absorbs (keeps in itself and doesn't reflect). This type of sensor is digital photo-camera and distance meter and other similar sensors.
Mechanical influence	These sensors are widely spread and used to measure weight, pressure, speed, acceleration, position, tensile strength. It is important to remember that even a button is a sensor with which, for example, you can determine collision with an obstacle.
Electric influence	Electric charges size, voltage, current, alternating currents phase, polarization, electrical conductivity.
Magnetic influence	Magnetic fields polarization, flow and strength (similar to a compass).
Thermic influence or heat influence	Height of temperature, its changes and conductivity – how well a certain material conducts heat.
Sound – acoustic influence	Properties of a sound wave – amplitude, frequency, polarization, frequency spectrum, speed of sound etc. sound amplitude (loudness) and frequency (tone) is enough for simpler applications.
Biological and chemical influence	The concentration of gas or liquid in a particular environment. Used in gas analyzers and different type of leak detectors for chemical substances.

Specific regularities and material properties are used in order to make these sensors work. Here are only a few of those:

1. **Faraday's law of induction** – conductive coil resists changes in magnetic field by generating electric current and voltage in the opposite direction of magnetic field.
2. **Photoconductivity** – certain semiconductors become better conductors after absorbing electromagnetic radiation because its conductivity or electrical resistance decreases.

These regularities and their wide use in everyday life is a good reason for the reader to find practical interest in physics during their time in school or facilities for higher education. Sensors can be found everywhere – radios, cars, refrigerators, vacuums, medical machinery and many others. If these regularities were not discovered and engineers hadn't found their practical use, automation wouldn't exist.

5.2.1. How to choose a sensor

The choice of sensor is usually determined by practical measures, size, weight, and price.

However, for some occasions where some specific properties of sensor or observable object are important it is useful to know certain characteristics.

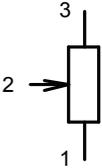
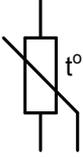
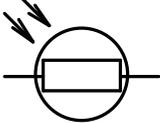
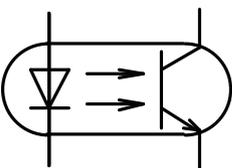
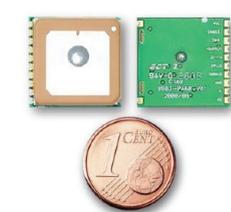
Remember! *There are no universal sensors that would be able to serve all purposes. Before you choose a sensor you have to know the main characteristic you need and only then you can start looking for specific sensor.*

The most important characteristics for choosing a sensor are shown in the table.

Characteristics group	Characteristic	Short explanation
Environmental factors	Performance temperature range in which sensor doesn't lose its performance quality	If work mode takes place in specific conditions, for example Latvia's winters can reach -30 °C, you should choose a sensor with the appropriate indications for its use in the technical documentation.
	Humidity	Resistance to humidity is very important if the sensor has to operate outdoors.
	Corrosion resistance	Resistance to corrosion is also connected to operating outdoors.
	Size	Sometimes sensors size can influence its practicality in specific product.
	Overly intensive exposure	This indicator is important for a temperature sensor such as temporary exceedances of temperature would continue to operate.
	Resistance to mechanical exposure – vibrations, falling from a height	Usually this characteristic is important for mobile devices – mobile phones, cars and other environments that are connected to vibrations.
	Energy consumption	Especially important characteristic for devices that work on batteries (devices with limited energy). In these cases sensors precision and other properties are sacrificed to save energy.
	Self-test	Characteristic is important in autonomous devices which have to ensure self-sufficient work when human interference is impossible.
Economic factors	Price	Important in the case of limited financial resources.
	Accessibility	Important in mass production when you have to be sure that you can access the needed amount.
	Work life	Work life determines the duration according to technical documentation; during this time sensors actions are predictable.
Sensor operation factors	Sensitivity	The smallest change in measured aspect that can be determined.
	Working range	The interval of measured aspect that can be determined.
	Stability	Resistance to noises.
	Repeatability	Difference in measurements in the same conditions.
	Linearity	Determines measurements dependence on changes in conditions if they are continuously and evenly changed.
	Error	Measurements deviation from its fixed value.
	Response time	Time needed to make every next measurement.
	Frequency	The amount of measurements that can be made in a specific amount of time.

5.2.2.Examples of sensors

Different popular sensors used by robot enthusiasts. (Images from www.robotshop.eu)

Sensor	Symbol in scheme	Short description
		Potentiometer – usually a three terminal device. It consists of resistor with added sliding contact and joystick. By turning the joystick the resistance between the middle and the side terminals changes. It is often used for regulation or as a tilt angle sensor, for example, in servomotors.
		Thermistor – resistor which resistance changes depending on the temperature. It consists of special ceramics or polymers and it is used to accurately measure temperature in small range.
		Photo-resistor – makes the electrons in it much more mobile by changing its resistance depending on the intensity of the light.
	No general symbol	"Parallax Ping" Ultrasonic distance sensor – measures the time in which the ultrasound echoes. Knowing the speed of the ultrasound used it is possible to calculate the distance to closest object or obstacle.
	No general symbol	"Lynxmotion" Quadrature motor encoder – creates many impulses in every motors rotation field; by making one full rotation a small piece of metal or magnet situated on special plate creates an impulse which is detected by microprocessor that calculates the real rotation speed of the motor.
		Optocoupler – device used as reflected lights sensor. Optocoupler consists of infrared light diode and phototransistor. The light emitted by the diode reflects from a surface and goes to phototransistors base. If there is a current in transistors base it begins to conduct current between collector and emitter. the amount of reflected light determines the amount of the current; this affects voltage on the transistor which is measured
	No general symbol	SHARP distance sensor – uses infrared light which is emitted in the form of impulses and the reflected impulse is received in miniature photoelectric array. The distance from the sensor to the object from which the impulse reflected can be calculated depending on the specific photocell that has turned on. The distance is coded using voltage in the range of 0-5 V; this allows the microprocessor to interpret the measurement.
	No general symbol	"MediaTek AGPS" sensor – much more complicated than previous sensors that determined GPS coordinates. They are obtained by using Data exchange protocol and appropriate software microcontroller. These sensors can be found in mobile phones and other portable devices.

Nowadays the variations of sensors are so big that it is impossible to include everything in this learnig material.

However, the reader can easily go to one of the internet stores meant for robotics enthuziasts and learn more about the available sensors and their characteristics.

5.1. WORKSHEET. Angle sensor

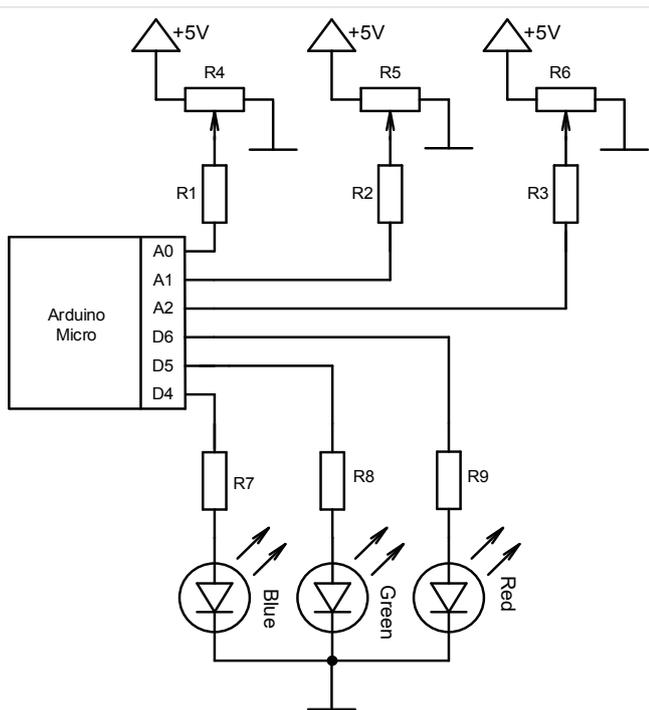
Aim

Learn the use of potentiometer.

Materials needed

Material/part	Amount
Resistors R1, R2, R3 (1 kΩ)	3
Potentiometer R4, R5, R6 (50 kΩ)	3
Resistors R7, R8, R9 (330 Ω)	3
RGB LED	1
Mounting cord	As much as needed

Scheme to be created



Changing the colour of RGB LED by using three potentiometers

Steps of work

1. Connect the circuit shown in the image.
2. Connect the circuit to a source of voltage.
3. Upload the given program.
4. Turn potentiometers joysticks.
5. See how the RGB LED colour changes.

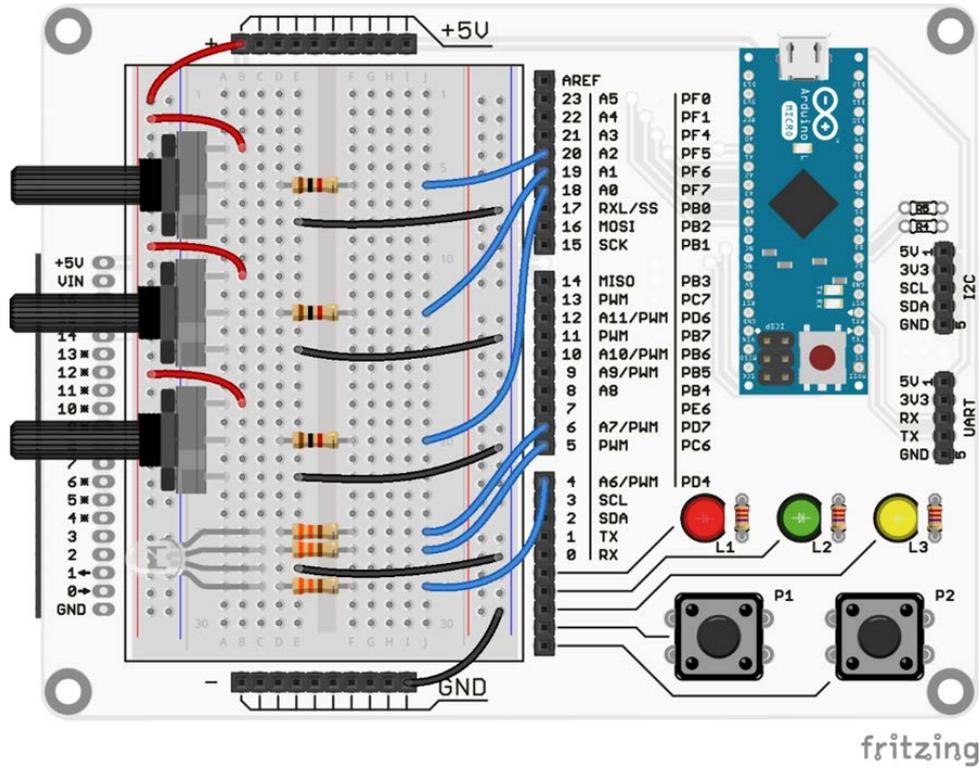
```
int greenPin = 6; //Green diodes pin
int bluePin = 5; //Blue diodes pin
int redPin = 4; //Red diodes pin
int pot1Pin = A0; //First potentiometer
int pot2Pin = A1; //Second potentiometer
int pot3Pin = A2; //Third potentiometer

void setup ()
{
    redRead = analogRead(pot3Pin)/4
    //reads the value of the potentiometer
    //and divides by 4a
    greenRead = analogRead(pot2Pin)/4
    //reads the value of the potentiometer
    //and divides by 4
    #define blueRead = analogRead(pot1Pin)/4
    //reads the value of the potentiometer
    //and divides by 4
}

void RGBLED (int red,int green,int blue)
//function for managing the RGB diode
{
    digitalWrite(redPin,red);
    //shows the red colours identity
    digitalWrite(greenPin,green);
    //shows the green colours identity
    digitalWrite(bluePin,blue);
    //shows the blue colours identity
}

void loop ()
{
    RGBLED (redRead,greenRead,blueRead);
    //turns on the RGB LED colour setup
    //function
}
```

Angle sensor



5.2. WORKSHEET. Temperature sensor

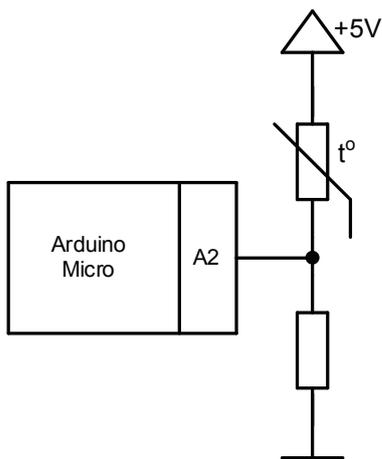
Aim

Learn how to use thermo-resistor and show the gained information on the monitor.

Materials needed

Material/part	Amount
Thermistor	1
Resistor (1 k Ω)	1
Mounting cord	As much as needed

Scheme to be created



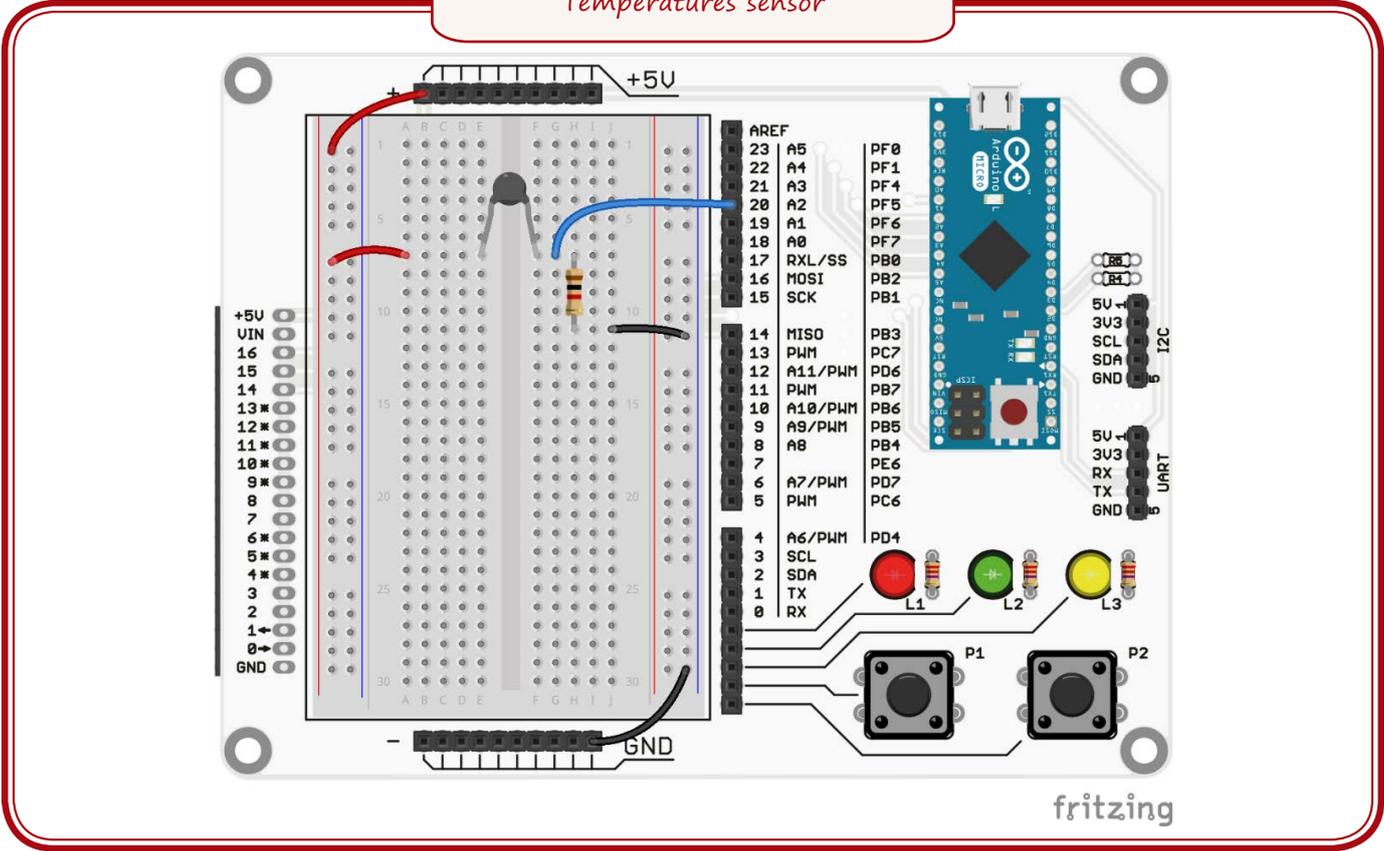
Scheme of connecting thermo-resistor to measure temperature.

Steps of work

1. Connect the circuit shown in the image.
2. Connect the circuit to a source of voltage.
3. Upload the given program.
4. Open series port monitor Tools → Series port monitor
5. See the emitted temperature in series port monitor.
6. Squeeze the sensor in your fingers and see the change in temperature.

```
int termoPin = A2; //thermo-resistors pin
float tempcoef = 0.07;
//resistors coefficient for changes
//in voltage
void setup ()
{
  #define termoReading analogRead(termoPin)
  // termo-resistors value reading
  Serial.begin(9600);
  //turning on communications with computer
}
void loop ()
{
  float temperature = termoReading*tempcoef;
  //calculating temperature
  Serial.print ("Temperatura ir: ");
  Serial.println(temperature);
  //outputs information to series
  //port monitor
  delay(20);
}
```

Temperatures sensor



5.3. WORKSHEET. Light sensor

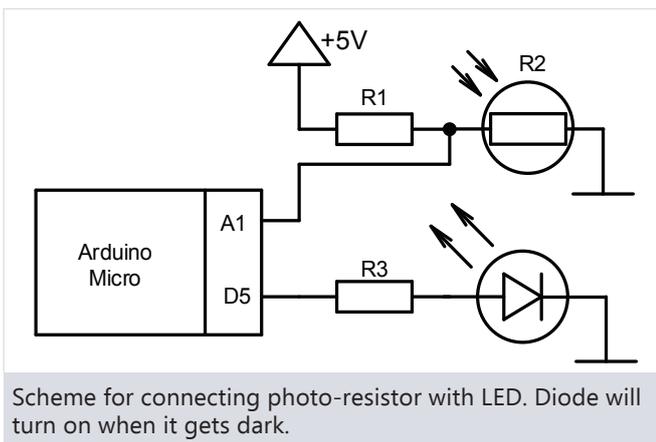
Aim

Learn how to use photo-resistor.

Materials needed

Material/part	Amount
Resistor R1 (1 k Ω)	1
Photo-resistor R2 (20 k Ω)	1
Resistor R3 (330 Ω)	1
LED	1
Mounting cord	As much as needed

Scheme to be created



Steps of work

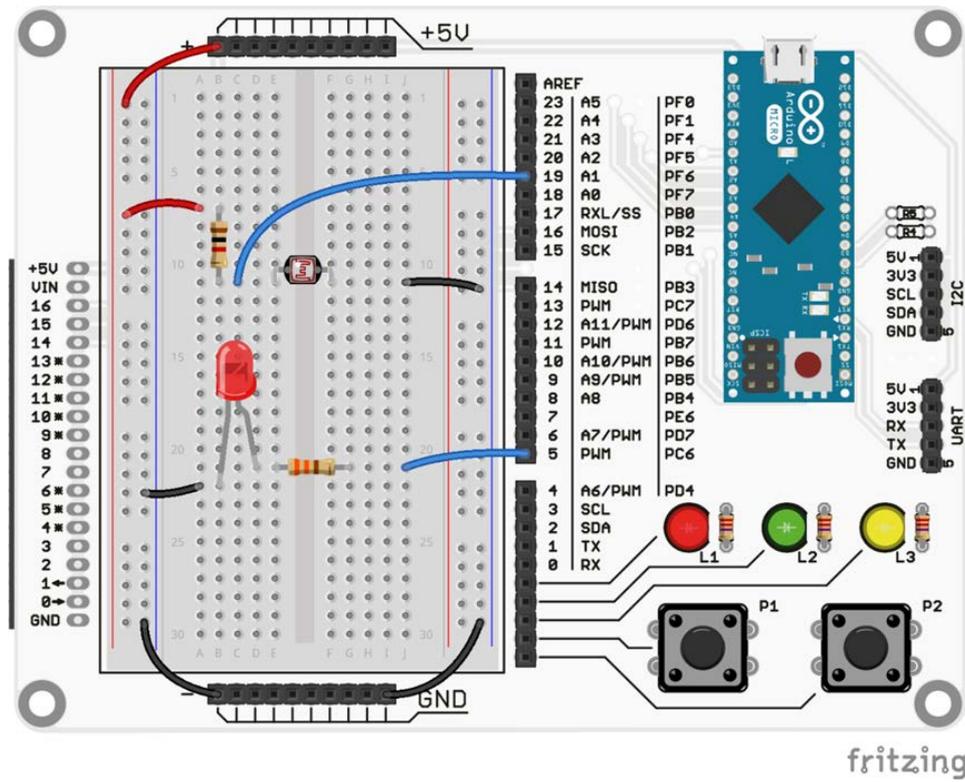
1. Connect the circuit shown in the image.
2. Connect the circuit to a source of voltage.
3. Upload the given program.
4. Open series port monitor Tools → Series port monitor
5. See the emitted light level in series port monitor.
6. Cover the photo-resistor with your palm.
7. See whether the LED turns on.
8. If the LED doesn't turn on, change 'darktreshold' value to lower than shown in the series port monitor after covering the diode.

```
int fotoPin = A1; //photo-resistors pin
int ledPin = 5; //LED pin
int darktreshold = 900; //darkness threshold

void setup ()
{
  pinMode(ledPin,OUTPUT);
  //sets up LEDs leg as an output
#define fotoReading analogRead(fotoPin)
  //pfoto-resistors value reading
#define ledON digitalWrite(ledPin,HIGH)
  //define the turning off of the LED
#define ledOFF digitalWrite(ledPin,LOW)
  //define the turning on of the LED
  Serial.begin(9600);
  //turns on the communication with
  //the computer
}

void loop ()
{
  Serial.print ("Light value is: ");
  Serial.println(fotoReading);
  //outputs information to series
  //port monitor
  if (fotoReading > darktreshold)
  //true if the photo-read is bigger
  //than darkness threshold
  {
    ledON; //if its dark turn on the LED
  }
  else
  {
    ledOFF;
    // if its light turn off the LED
  }
}
```

Light sensor



5.4. WORKSHEET. Obstacle and light reflection sensor

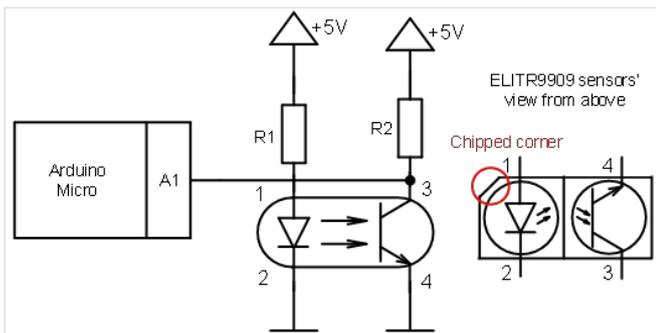
Aim

Learn how to use optocoupler.

Materials needed

Material/part	Amount
Resistor R1 (330 Ω)	1
Resistor R2 (10 k Ω)	1
Optopoupler ELITR9909	1
Mounting cord	As much as needed

Scheme to be created



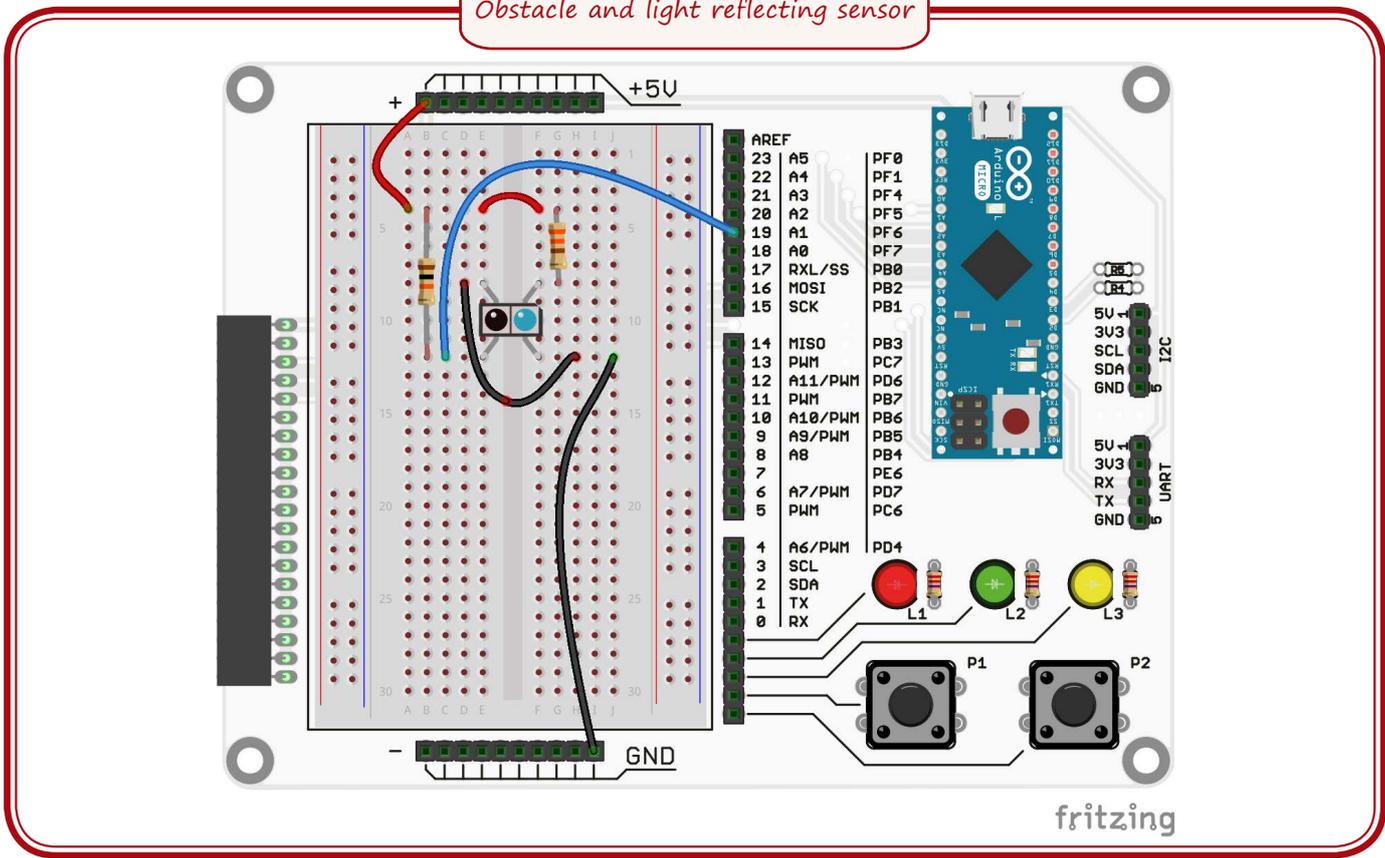
Connecting optocoupler and reading sensory data.

Steps of work

1. Connect the circuit shown in the image.
2. **Attention - pay attention to correctly connecting the sensor and the location of MC legs!**
3. Connect the circuit to a source of voltage.
4. Upload the given program.
5. Open series port monitor Tools → Series port monitor
6. See the information shown in series port monitor.
7. Put a white sheet of paper 4 mm from the sensor.
8. See the sensors value.
9. Put a black sheet of paper 4mm from the sensor.
10. See the sensors value.
11. Put a sheet of paper in front of the sensor in multiple different distances.
12. See the results.

```
int optoPin = A1; //optocouplers sensors pin
int objecttreshold = 1000;
    //threshold of the object
int whitetreshold = 150;
    //white colours threshold
void setup ()
{ #define optoReading analogRead(optoPin)
    //optocouplers signal reading
  Serial.begin(9600);
    //turns on the communication with
    //computer
}
void loop ()
{
  Serial.print ("sensors reading is: ");
  Serial.println(optoReading);
    //shows sensors readings
  if (optoReading < objecttreshold)
    //true if the reading is lower than
    //objects threshold
  {
    Serial.println ("object is in front of
the sensor!");
    if (optoReading < whitetreshold)
      //true if the reading is lower
      //than the white colours threshold
    {
      Serial.println ("object has a light
colour!");
    }
    else
      //if the reading is bigger than the
      white colours threshold
    {
      Serial.println ("object has a dark
colour");
    }
  }
  else
    // if the reading is bigger than
    //the objects threshold
  {
    Serial.println ("there is no object in
front of the sensor!");
  }
  delay(500);
}
```

Obstacle and light reflecting sensor



5.5. WORKSHEET. Capacitive sensor

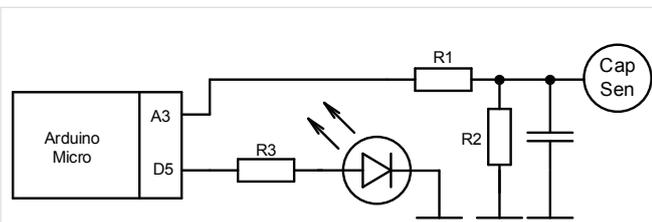
Aim

Create simple capacitive sensor.

Materials needed

Material/part	Amount
Resistor R1 (4.7 k Ω)	1
Resistor R2 (47 k Ω)	1
Resistor R3 (330 Ω)	1
Capacitor (15 pF)	1
LED	1
Capacitive button from aluminium or copper foil	1
Mounting cord	As much as needed

Scheme to be created



Scheme for capacitive button. The button is made out of aluminium or copper foil and connected to the scheme with a cord.

Steps of work

1. Connect the circuit shown in the image.
2. Connect the circuit to a source of voltage.
3. Connect aluminium or copper plate.
4. Upload the given program.
5. Touch the plate with your finger.
6. See if the diode turns on.
7. Open series port monitor Tools → Series port monitor.
8. See the sensors readings in series port monitor depending on whether or not you are touching the button.

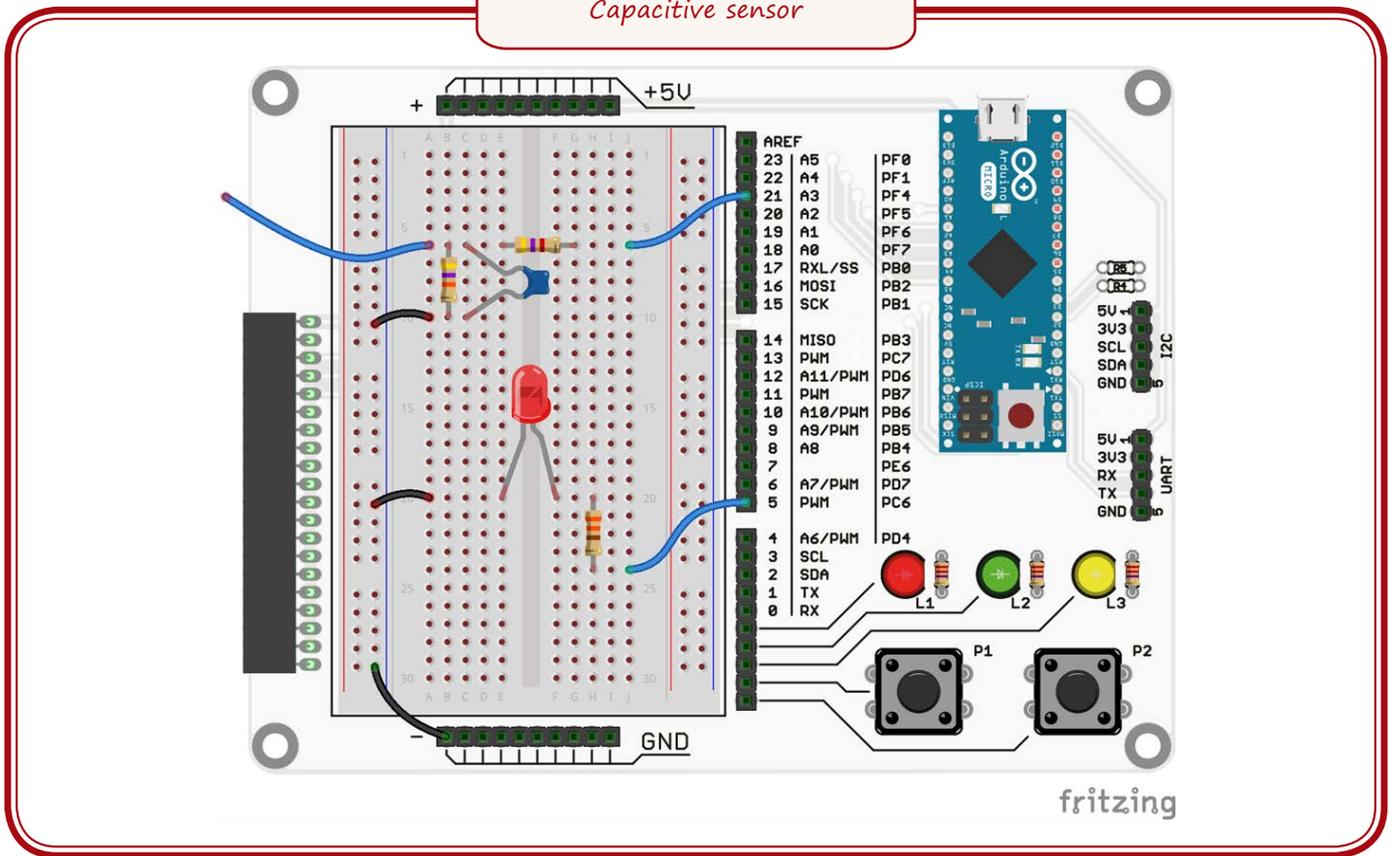
```
int capPin = A3; //capacitive sensors pin
int ledPin = 5; //diodes pin

void setup ()
{
    analogRead(capPin)
    //capacitive sensors readings
    pinMode(ledPin,OUTPUT);
    //set up LED leg as an output
    #define ledON digitalWrite(ledPin,HIGH)
    //turn on LED
    #define ledOFF digitalWrite(ledPin,LOW)
    //turn off LED
    Serial.begin(9600);
    //turn on communication with the
} //computer

int CapacitivSen ()
//function for capacitive sensors readings
{
    pinMode(capPin,OUTPUT); //set up the MC
    //leg for readings as an output
    digitalWrite(capPin,HIGH);
    //charge the sensor
    pinMode(capPin,INPUT);
    //set up the MC leg for readings as an
    //input
    int cap = capReading;
    //read the sensors value
    return cap;
    //function returns the read value
}

void loop ()
{
    Serial.println(CapacitivSen());
    //reads the value on the screen
    if (CapacitivSen(>2){
        //turn on the diode if the value
        //is bigger than 2
        ledON;
    }
    else {
        ledOFF;
    }
    delay(10);
}
```

Capacitive sensor



6.

MOTORS AND THEIR MANAGEMENT

6.1. Aim

The aim of this chapter is to introduce different types of motors and the basics of their management. This chapter does not give extensive description of all electric motors, but only the ones accessible in the constructor – permanent-magnet DC motor and servomotor.

6.2. Theoretical part

Electric motor is an electro-technical device which can change electrical energy into mechanical energy; motor turns because there is electricity flowing in its windings. Electric motors have seen many technical solutions over the year from which the simplest is the permanent-magnet DC motor. Its construction and simplified scheme of operation is shown in the image.

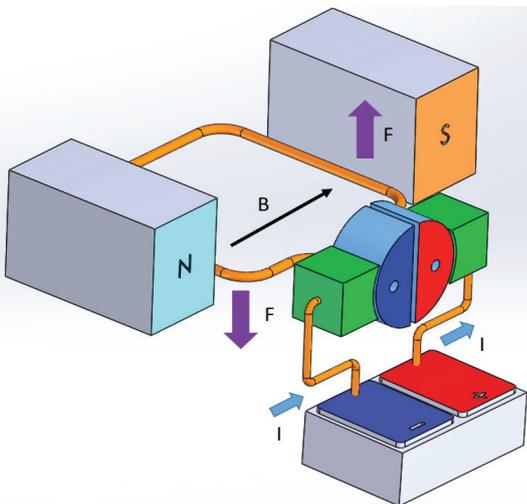
Without diving into details you have to know that the spinning force F is directly proportional to current I and magnetic field strength B . Because of this – the stronger the magnets the bigger the current bigger the motors force.

In the last 20 years there have appeared small but very powerful motors because of the especially powerful neodymium magnets. However, when talking about use in robotics the management of the motor is important aspect; how can robots managements mechanisms influence the motor to change its rotations direction and speed to the one that's needed.

6.2.1. Electro-motors speed change

In this part we will look at one specific way of managing the speed of permanent-magnet DC motor which is based on determining the width of voltage impulse. This technique sometimes is also used with obtaining analog signals with digital techniques methods. This technique basically means turning on and off the power and creating an impulse with specific length (the time when the power was on). By changing the proportion of time when the power is on and when it is off we change the speed of the motor. By changing this proportion we modulate the signal (see the image - "Impulse width modulated signal by using command `analogWrite()`"). This is how the technical name of impulse width modulation was created.

Permanent-magnet DC motors scheme of construction and operation.



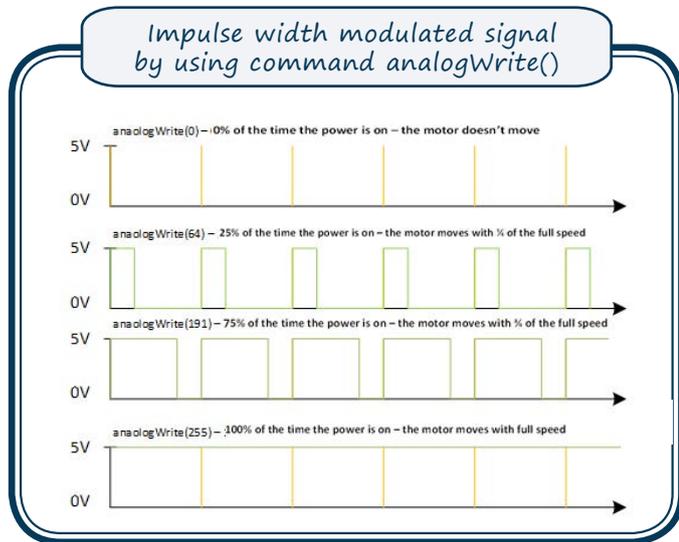
I – The direction of the current; from the battery's positive pole to the negative.

B – The direction of magnetic field created by permanent magnets.

F – The direction of current and the force created by magnetic field.

As shown in the image, current and magnetic field creates force that turns the motors winding – the rotor – in a specific direction. However, this force maintains only for a while; until the rotor is turned by 180 degrees. That is why there are many windings that can ensure continuous operation of the motor with the help of contactors (shown in green) which helps them switch while keeping as big of a force F as possible.

By changing the poles of the circuit the force will change its direction and the motor will turn in the opposite direction.



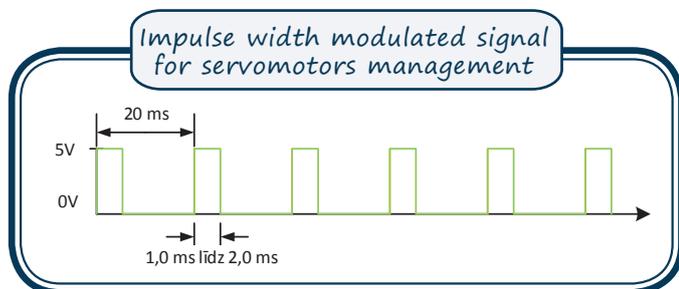
In this image the orange lines show time intervals that are dependent on the specific microprocessor. With Arduino they are 50 Hz. The green line shows time in which power is turned on or off – the width of the impulse.

By using the program `analogWrite()` it is possible to change the speed of the motor or the brightness of the LED as it was shown in the previous chapter.

6.2.2. Managing servomotor

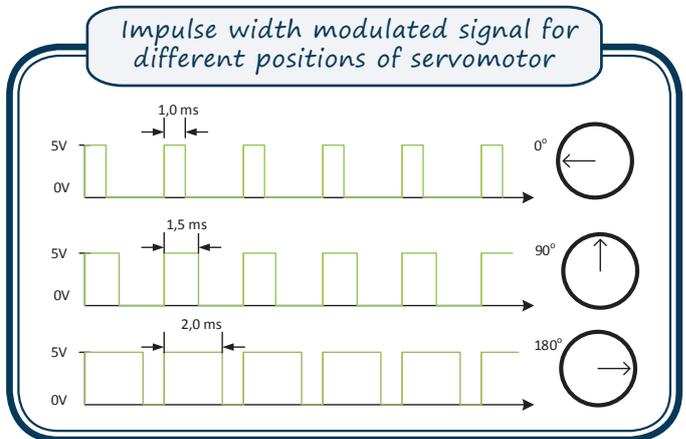
Unlike the simple DC motor, servomotor is a special management chain which allows very easy control over motors speed or position in which the motors axle should be situated. The management of the engine is realised by using three connections – currents positive (usually red) and negative connection (brown or black) as well as management connection (orange or yellow).

The same impulse width technique is used for the management, but it is used with different cycle length (see the image - "Impulse width modulated signal for servomotors management").



As you can see the servomotors impulse's cycle length is 20 milliseconds but the impulse length is 1 or 2 milliseconds. These signal characteristics are true for most enthusiast level servomotors, but it should be checked for each individual module in its manufacturer's specifications.

Servomotors management chain expects the impulse every 20 milliseconds, but the width of the impulse shows the position the servomotor hat to reach. For example, 1ms (millisecond) shows to 0 degree position but 2 ms to 180 degree position in relation to the starting point. When in this position the servomotor will stay in it; resist any outer forces from trying to change its position. The graph of it is shown in the image.



Servomotors, the same as other motors, have different parameters, the most important being working time; the time that's needed to change the position to specific angle. For best enthusiast level servomotors it is 0.09 seconds to turn 60 degrees.

Servomotors deviation:

- Positional rotation servomotor – most widely used type of servomotor. It was described above and with the help of management signal it can determine the position of the axel from its starting position.
- Continuous rotation servomotor – this type of motor allows setting the speed and direction of the rotation using the management signal. If the position is less than 90 degrees it turns in one direction, but if more than 90 degrees it turns in the opposite direction. The speed is determined by the difference in value from 90 degrees; 0 or 180 degrees will turn the motor at its maximum speed while 91 or 89 degrees at its minimum speed.
- Linear servomotor – with the help of extra transfers it allows moving forward or backward, it doesn't rotate.

Sadly, Arduino is not as easily manageable as DC motor. That is why there is a special servomotor management library.

6.2.3.H bridge

The H bridge has earned its name because of its resemblance to the capital 'H' where in all the corners it has switches and in the middle – electric motor.

This bridge is usually used for operating permanent-magnet DC motor, electro magnets and other similar elements because it allows operating significantly bigger currents devices with a small current.

By switching the switches you can change the motors direction. You have to keep in mind that the switches need to be turned on and off in pairs. It is shown in the image below.

If both of the positive or negative switches are turned on in the top or the bottom then the engine stops, it can't freely rotate so it is slowed down. The management can be reflected in this table below.

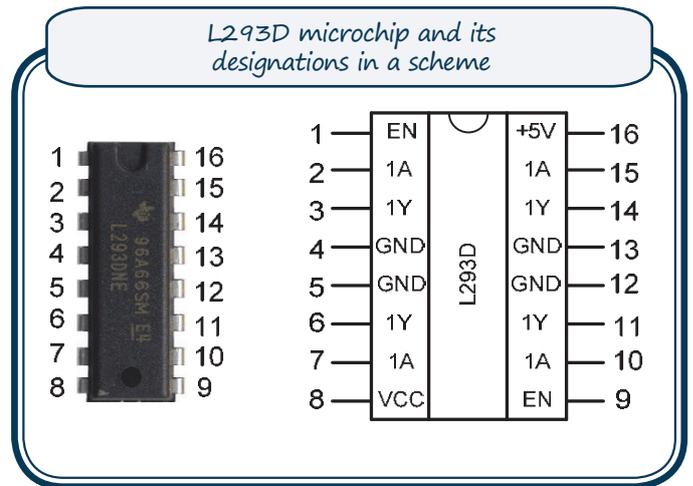
If all of the switches are turned off then the engine is in free movement. In robotics it is not always enough so sometimes the H bridge is turned on in the opposite direction to slow it down faster; it quickly turned in reverse.

Remember! Neither of these braking mechanisms is good for the H bridge or the power source used. That is why this action is unacceptable without a special reason because it can damage the switches or the power source.

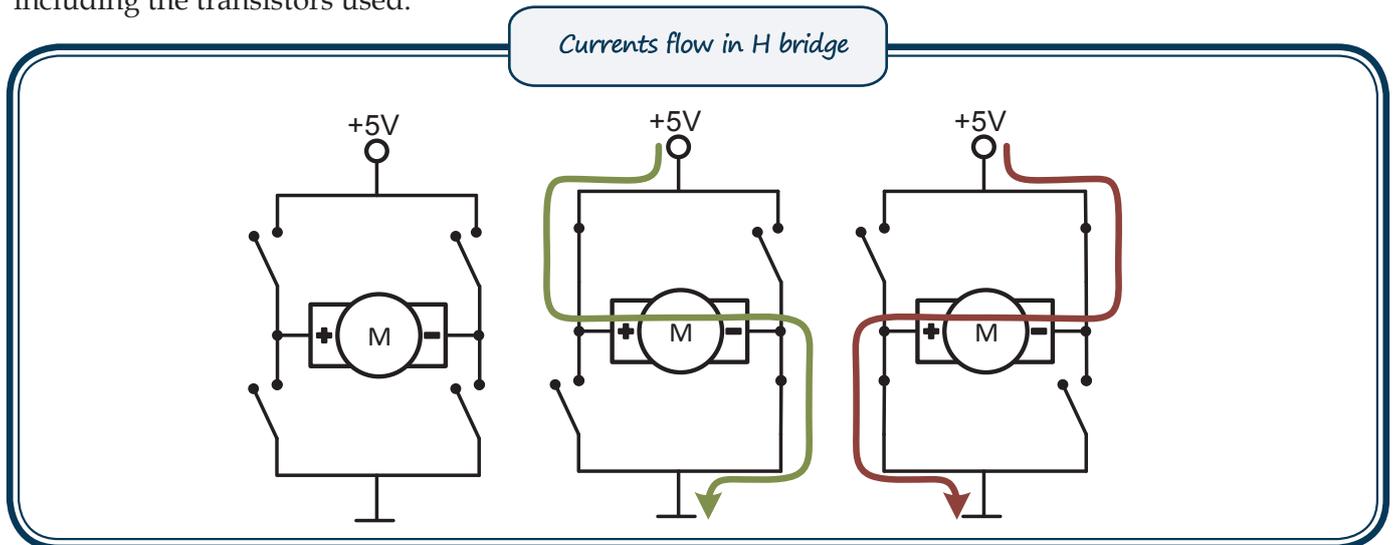
This far everything is fairly simple the complicated part is with realising the switches - if the switches don't work usually relays or appropriate power transistors are used. The biggest drawback for relays is that they can only turn the engine on or off. If the rotation speed needs to be regulated using the impulse width modulation then transistors have to be used. MOSFET type transistors should be used for ensuring large amount of power.

Specific circuit depends on multiple factors including the transistors used.

Nowadays to ensure stable operation of the bridge, extra elements are added. Manufactured bridges have one body, for example, the one that's included in the constructor - L293D.



L293D microchip consists of two H bridges and is meant for managing two motors. Every microchips leg has its own function that's why it is very important not to mix them up, otherwise the microchip can be damaged. All microchips have numbered legs. The numeration begins with mark on the body: chipped surface or side, dot or some other, and continues counter clockwise. While creating a scheme it is important to take notice of these numbers and the numbers shown in the scheme. If you need to find additional information about the microchip it can be found in its datasheet. Remember that the datasheet can be found by writing the number of the device (written on the body) and adding the word 'datasheet' in the browser.



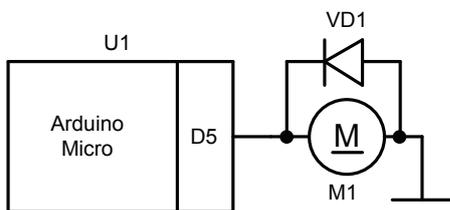
Upper left	Upper right	Lower left	Lower right	Motor work mode
On	Off	Off	On	Turns in one direction
Off	On	On	Off	Turns in the other direction
On	On	Off	Off	Is slowed
Off	Off	On	On	Is slowed

6.1. WORKSHEET. Operating a motor using a program

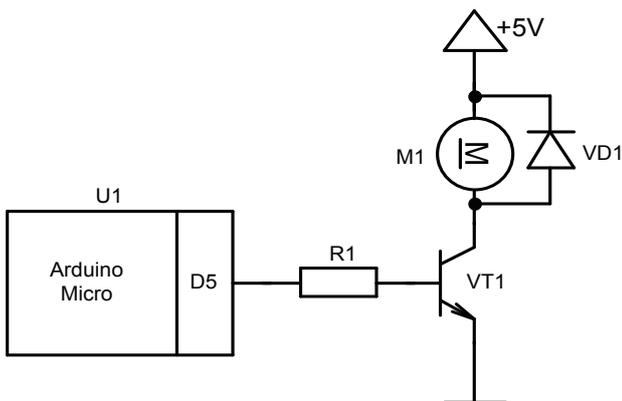
Aim

Learn how to use thermo-resistor and show the gained information on the monitor.

Scheme to be created



If the motor will be connected to Arduino without a transistor it will turn very slowly or not turn at all.



Motor will turn faster if it will be managed through a transistor.

Materials needed

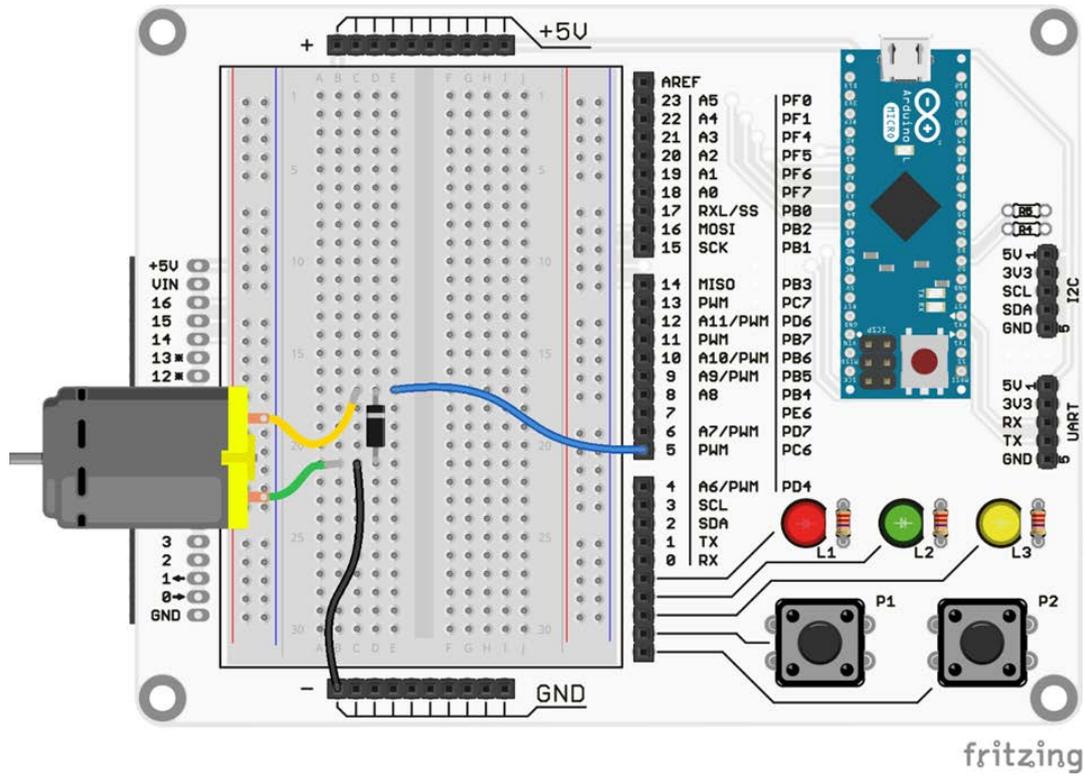
Material/part	Amount
Resistor (10 kΩ)	1
DC motor (3-6 V)	1
Diode PH4148	1
NPN transistor BC517	1
Mounting cord	As much as needed

darba izpildes soļi

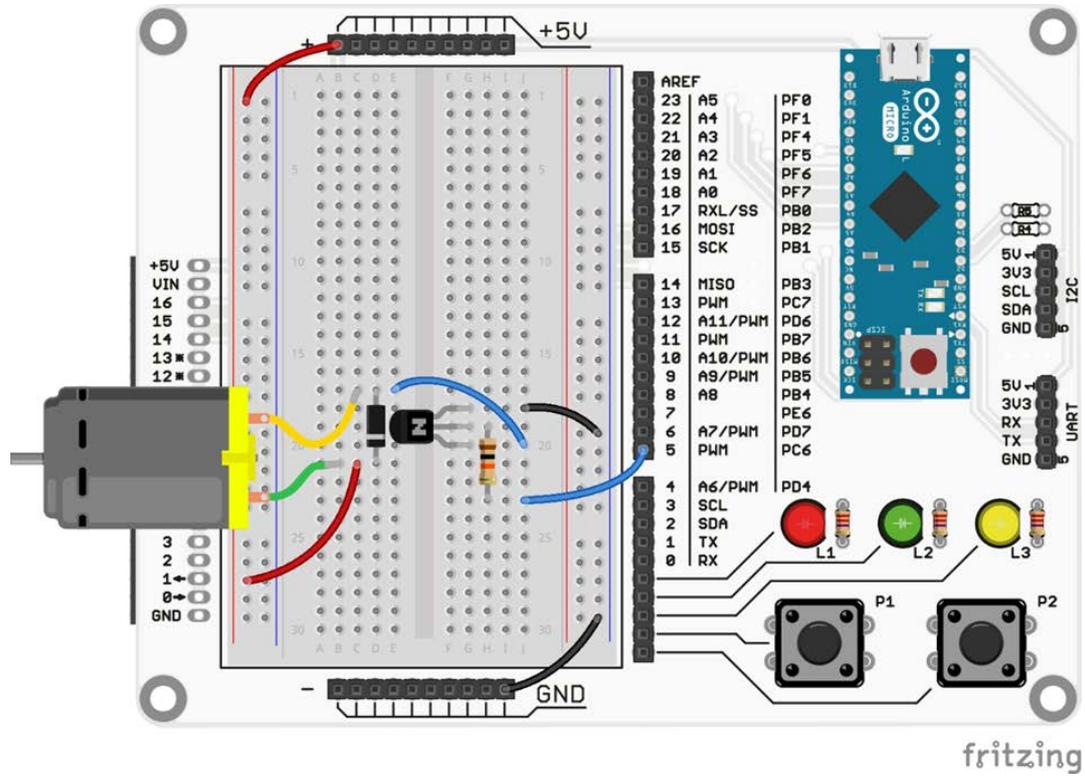
1. Connect the circuit shown in the image "Managing motor with program".
2. Connect the circuit to a source of voltage. Upload the given program.
3. See that the motor doesn't turn.
4. Connect the circuit shown in the image "Managing motor with program and transistor".
5. Connect the circuit to a source of voltage.
6. See that the motor turns.

```
void setup ()
{
  pinMode(5, OUTPUT);
  //set up MC leg No 5 as an output
  #define motON digitalWrite(5, HIGH)
  //define how to turn on the first diode
  #define motOFF digitalWrite(5, LOW)
  //define how to turn off the first diode
}
void loop ()
{
  motON;
}
```

Managing motor with program



Managing motor with program and transistor

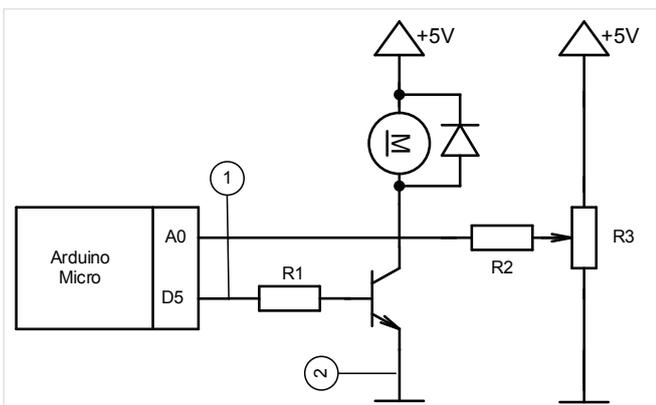


6.2. WORKSHEET. Regulating motors speed using a program

Aim

Learn how to use PWM to regulate the speed of the motor.

Scheme to be created



Scheme from the previous exercise is upgraded with potentiometer which will determine the speed of the motor.

Steps of work

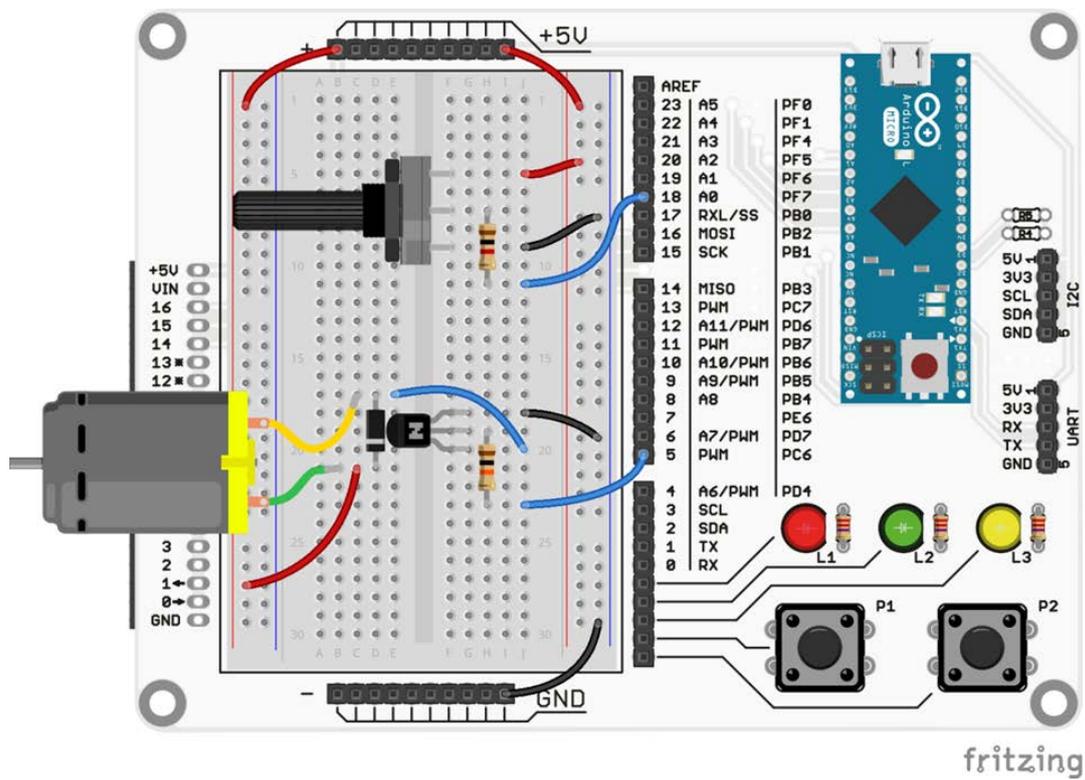
1. Connect the circuit shown in the image.
2. Write the given program.
3. Connect the circuit to a source of voltage.
4. Upload the given program.
5. Turn the potentiometers lever.
6. See if the motors speed changes after turning the lever.
7. Open series port monitor Tools → Series port monitor.
8. See how the speed of the motor changes depending on the angle in which the lever is turned.
9. Turn the lever so the port monitor would show around 200.
10. Measure voltage between the points 1 and 2.
11. Turn the lever so the port monitor would show around 50.
12. Measure voltage between the points 1 and 2.
13. Calculate what voltage corresponds to what values on the series port monitor.

```
int motPin = 5; //motor management output
int potPin = A0;
//potentiometers readings input
void setup ()
{
  pinMode(motPin,OUTPUT);
  //set up motor management leg as an
  //output
  /*analog input reads values from 0 to
  1023, but motors management happens in
  the range of 0 to 255, that is why the
  read value is divided by 4*/
  #define potReading analogRead(potPin)/4
  //definition of potentiometers reading
  //comand
  #define motDrive analogWrite(motPin,
  potReading);
  //definition of managing the motor
  Serial.begin(9600);
  //turn on communication with the computer
}
void loop ()
{
  motDrive;
  //turn on the motor according to the
  //potentiometers readings
  Serial.println(potReading);
  //information about the speed of the motor
}
```

Materials needed

Material/part	Amount
Resistor R1 (10kΩ)	1
Resistor R2 (1 kΩ)	1
Potentiometer R3 (50 kΩ)	1
DC motor (3-6 V)	1
Diode IN5819	1
NPN transistor BC517	1
Mounting cord	As much as needed

Regulating motors speed with a program



6.3. WORKSHEET. Servomotors management

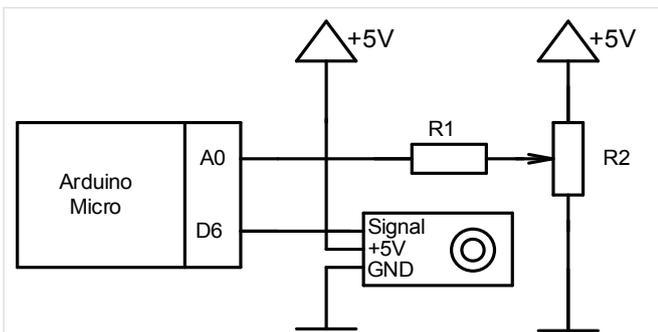
Aim

Learn how to use servomotors and its management with program.

Materials needed

Material/part	Amount
Resistor R1 (1 k Ω)	1
Potentiometer R2 (50 k Ω)	1
DC servomotor (5 V)	1
Mounting cord	As much as needed

Scheme to be created



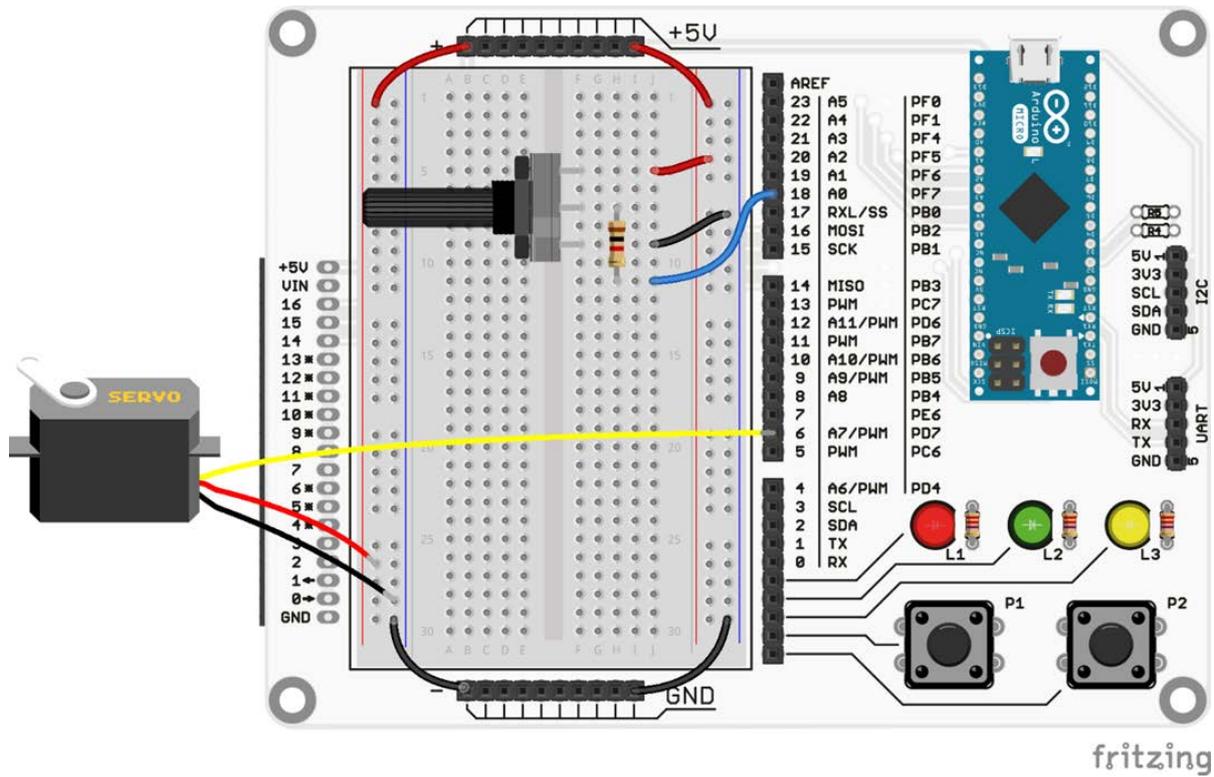
Scheme for connecting servomotor. It is important to correctly connect the servomotors cords. Voltage can't be greater than 5 V.

```
#include <Servo.h>
//add the library to servomotors management
int servoPin = 6;
    //servomotors management output
int potPin = A0;
    //potentiometers input
int servoAngle;
    //servomotors turning angle
Servo servoMotor;
    //create new servomotors object
void setup ()
{
    servoMotor.attach(servoPin);
        //add servo pin to servo object
#define potReading analogRead(potPin)
        //definition of potentiometer
        //reading comand
    Serial.begin(9600);
        //turn on communication with the computer
}
void loop ()
{
    servoAngle = map(potReading,0,1023,0,179);
        //changes potentiometers read
        //values into degrees
    servoMotor.write(servoAngle);
        //sets up turning angle on servo
    Serial.println(servoAngle);
        //information about servomotors angle
    delay(15);
        //delay so servo could return to its
        //starting position
}
```

darba izpildes soļi

1. Connect the circuit shown in the image. Write the given program.
2. Connect the circuit to a source of voltage.
3. Upload the given program.
4. Turn the potentiometers lever.
5. See if by turning the lever the servomotors turning angle changes.
6. Open series port monitor Tools → Series port monitor.
7. See how the motors angle change in correspondence to turning the lever.

Servomotors management



6.4. WORKSHEET. Using H bridge

Aim

Learn how to use H bridge microchip.

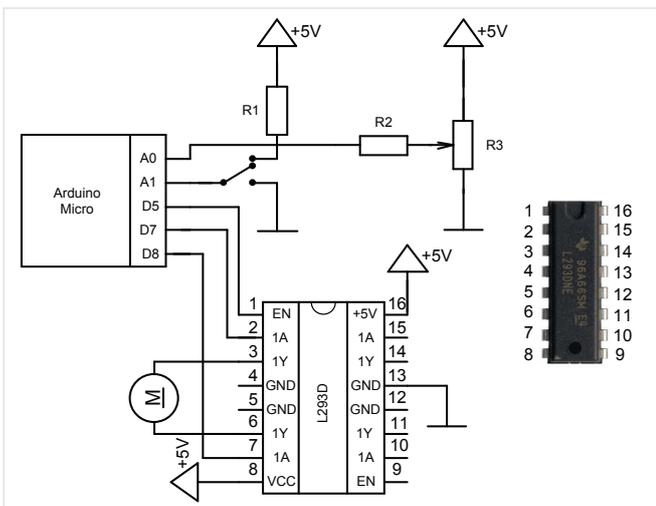
Steps of work

1. Connect the circuit shown in the image.
2. Write the given program.
3. Upload the given program.
4. Turn the potentiometers lever until the motor starts to turn.
5. Shift the motors direction switch.
6. See the motor direction changing.
7. Connect the circuit to a source of voltage.

Materials needed

Material/part	Amount
H bridge microchip L293D	1
Resistor R1 (10 kΩ)	1
Resistor R2 (1 kΩ)	1
Potentiometer R3 (50 kΩ)	1
Switch ON-ON	1
DC motor (3-6 V)	1
Mounting cord	As much as needed

Scheme to be created



Connecting H bridge microchip for managing single motor.

```

int dirPin1 =7;    //direction pin 1
int dirPin2 =8;    //direction pin 2
int speedPin = 5;  //motors speed output
int potPin = A0;
    //potentiometer reading input
int switchPin = A1;
    //direction switch reading pin

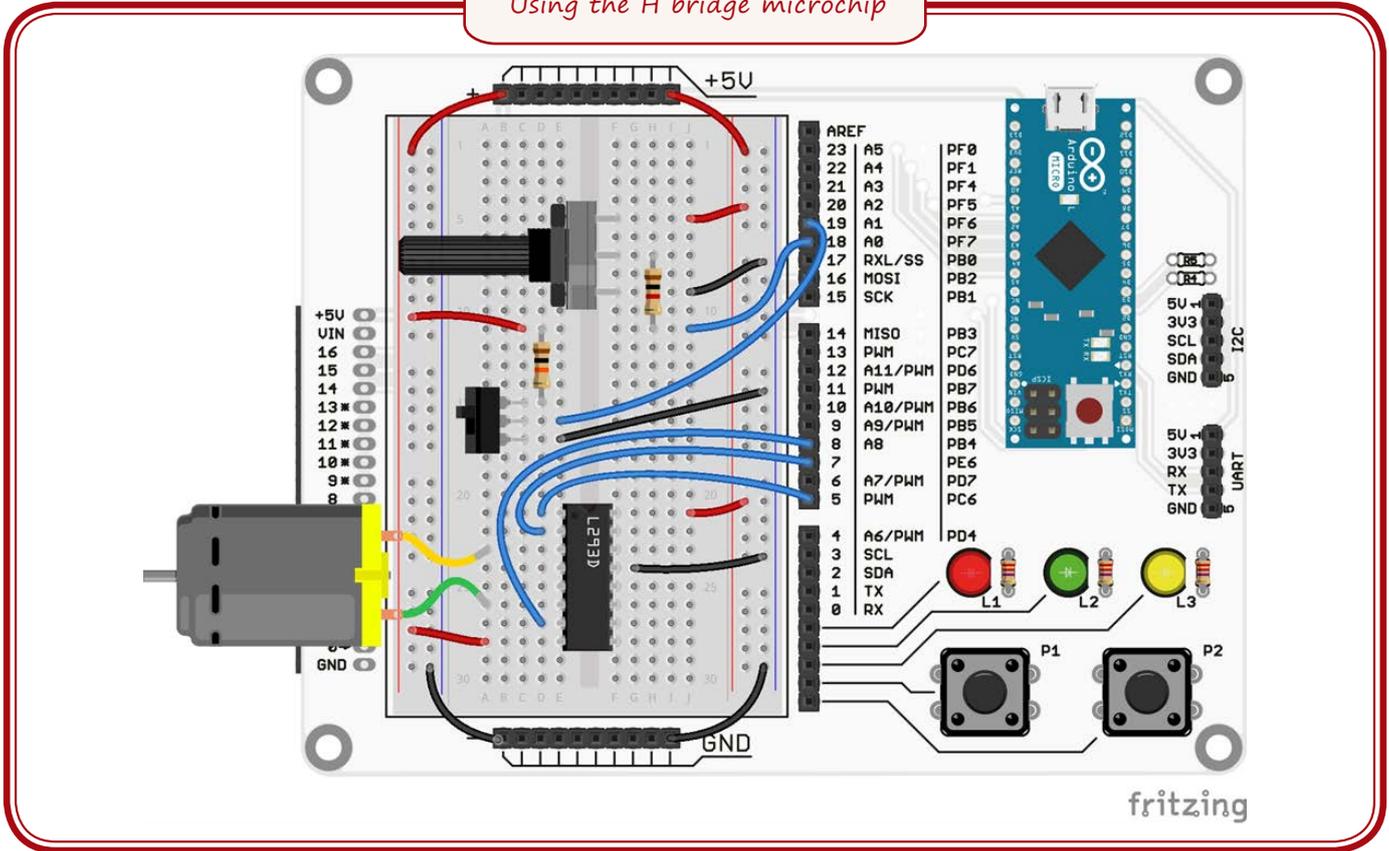
void setup ()
{
    pinMode(dirPin1,OUTPUT);
    //set un direction pin No 1 as output
    pinMode(dirPin2,OUTPUT);
    // set un direction pin No 2 as output
    pinMode(speedPin,OUTPUT);
    //set up speed pin as an output
    /*analog input reads values from 0
    to 1023, but motors management
    happens in the range of 0 to 255,
    that is why the read value is
    divided by 4*/
    #define potReading analogRead(potPin)/4
    //definition for potentiometers reading
    //command
    #define switchReading digitalRead(switchPin)
    //definition for switches reading command
    digitalWrite(switchPin,HIGH);
    //set up inner pull-up resistor for the
    //switches
}

void MotorDrive (int motSpeed,
                 int motDirection)
    //function for managing the motor
{
    analogWrite (speedPin,motSpeed);
    //setting the speed of the motor
    if (motDirection)
        //setting the direction of the motor
    {
        digitalWrite (dirPin1,HIGH);
        digitalWrite (dirPin2,LOW);
    }
    else
    {
        digitalWrite (dirPin1,LOW);
        digitalWrite (dirPin2,HIGH);
    }
}

void loop ()
{
    MotorDrive (potReading,switchReading);
    //calls motors management function
}

```

Using the H bridge microchip



7. MINI-SUMO ROBOT

7.1. Aim

The aim of this chapter is to provide an inside into the operations and programming peculiarities of the robot which has been included in the constructor while moving towards development on complete mini-sumo robot and creation of qualitative software. That is why we will look at robots construction and the meaning of its base elements. We will create an example or test program for testing the operation of robots nodes which will serve as a base for further more complicated programs.

7.2. Theory

Most robots are made up of three main parts which can ensure high performance and reaching the set goals if they work in agreement with each other. These parts are:

- **Mechanics** – all of the physical in the robot; everything we can take in our hands – body, printed circuit board, motor, wheels, battery and others. While creating mechanical part you have to follow the rules of mini-sumo competitions so you could participate in international competitions in Latvia or other countries. According to the mini-sumo competition rules a robot can't be bigger than 10x10cm or heavier than 500 g. Its components can't damage the surface of the field, come off, or influence opponents operations. If the rules are followed you can make whatever construction you want. However, you should keep in mind that there are robots meant for other uses; making their mechanical components requires a lot of time and attention because only complete cooperation of all these parts ensure successfully working robot.
- **Electronics** – part that connects the robot in a single system. It brings data from sensors to program and according to programs commands moves robots mechanics.

It directly influences how the robot should be programmed and what will it be able to do. This constructor's robots electronics is shown in scheme.

- **Software** – part that allows realising self-imagined operation of the robot. That is why in competitions often wins the smartest not the strongest robot. It processes sensor information and gives commands to robots output devices as well as motors and LED. Robot will do only what is written in its program. That is why if the robot doesn't do its task most likely the mistake will be in its program. The entire robots program is stored in microcontroller's memory which is located on Arduino plate.

If all of the parts work in agreement and correctly the robot moves and does its tasks as planned. Making a correct and effective program is not simple because you have to take in count both mechanic and electronic peculiarities. In this chapter you can see the basics of this type of program that will allow you to create your own program; one so you could win.

Mini-sumo robot



7.1. WORKSHEET. Robots LEDs

Aim

Learn the managements of “SumoBoy” mini-sumo’s LED.

Devices used in robot

Number in the diagram	Name in the program	Arduino pin
3	led1	12
7	led2	6
12	led3	13
14	led4	17

Steps of work

1. Make a blinking diodes program for led3 diode.

```
void setup() { //setting up
  pinMode(13,OUTPUT);
  //defines leg No13 as output
#define led3ON digitalWrite(13,HIGH)
  //definition for turning on led3
#define led3OFF digitalWrite(13,LOW)
  //definition for turning off led3
}
void loop() { //neverending cycle
  led3ON; //turns on led3
  delay(1000); //waits for 1000 milliseconds
  led3OFF; //turns off led3
  delay(1000); //waits for 1000 milliseconds
}
```

2. Connect the robot to the computers USB port.
3. Upload the program to the robot.
4. See how the led3 diode is blinking.
5. Change the delay time so the diode blinks faster.
6. Save the program, from now on we will upgrade and change the existing code.
7. Create a program where all the diodes of the robot turn on and off one by one by adding to the previous program. The additions are accented. Upload the program to the robot.
8. See how the diodes turn on and off one by one.

```
void setup() { //setting up
  //LED definitions
  pinMode(12,OUTPUT); //Left LED
#define led1ON digitalWrite(12,HIGH)
#define led1OFF digitalWrite(12,LOW)
  pinMode(6,OUTPUT); //Right LED
#define led2ON digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
  pinMode(11,OUTPUT); //Back LED
#define led3ON digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
  pinMode(2,OUTPUT); //Orange LED
#define led4ON digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
}
void loop() { //neverending cycle
  led1ON; //turns on led1
  delay(1000); //waits for 1000 ms
  led1OFF; //turns off led1
  led2ON;
  delay(1000);
  led2OFF;
  led3ON;
  delay(1000);
  led3OFF;
  led4ON;
  delay(1000);
  led4OFF;
}
```



7.2. WORKSHEET. Robots buttons

Aim

Learn mini-sumo robots button operations and use of series port monitor.

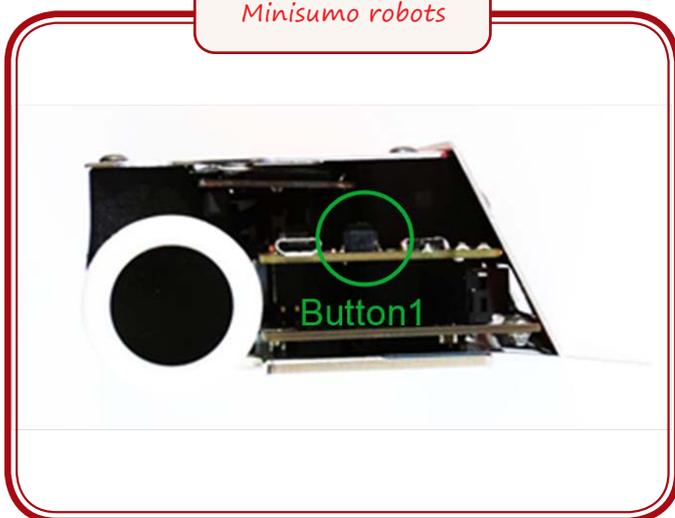
Devices used in robot

Number in the diagram	Name in the program	Arduino MC leg
10	Button1	A2

Steps of work

1. Add to the program so the BUTTON1 turns on and off the blinking of the LED as well as output information about the position of the button to the computer.
2. Connect the robot to the computers USB port.
3. Upload the program to the robot.
4. Push the button and check if the blinking of the LED can be turned on and off.
5. Turn on series circuits monitor and check if there is a notification about the buttons being pushed.
6. Save the program.

Minisumo robots



```
void setup() { //setting up
               //LED definitions
    pinMode(12,OUTPUT); //Left LED
#define led1ON digitalWrite(12,HIGH)
#define led1OFF digitalWrite(12,LOW)
    pinMode(6,OUTPUT); //Right LED
#define led2ON digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
    pinMode(13,OUTPUT); //Back LED
#define led3ON digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
    pinMode(17,OUTPUT); //Orange
#define led4ON digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
    //button definitions
    pinMode(A2,INPUT);
    digitalWrite(A2,HIGH);
#define button1 !digitalRead(A2)
    //turning on series port monitor
    Serial.begin(9600);
}
void loop() { //neverending cycle
    led1OFF;
    led2OFF;
    led3OFF;
    //turns off al diodes in the beginning of
    //the program
    if (button1) //turns on blinking diodes
    { //if button1 is pushed
        while (button1){
            //cycle while button is pushed
            Serial.println("Poga 1 ir nospiesta");
            //outputs when button1 is pushed
        }
        while(!button1)
            //blinking diodes
            //until button isn't pushed again
        {
            led1ON;
            led2ON;
            led3ON; //turns on 3 diodes
            delay(1000); //waits for 1000 ms
            led1OFF;
            led2OFF;
            led3OFF; //turns off 3 diodes
            delay(1000); //waits for 1000 ms
        }
        while (button1){
            //cycle while button1 is pushed
            Serial.println("Poga 1 ir nospiesta");
            //outputs when button1 is pushed
        }
    }
}
```

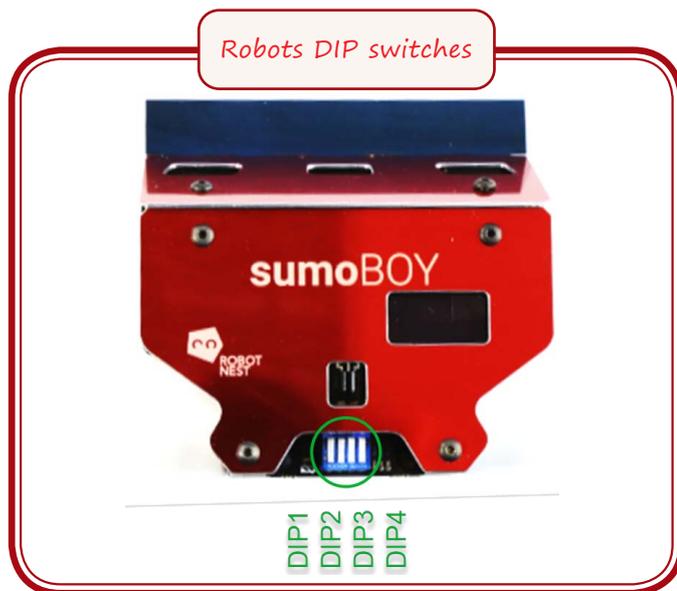
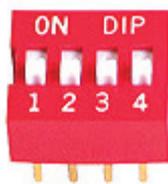
7.3. WORKSHEET. Robots DIP switches

Aim

Learn the use of robot DIP switches which allow turning on different robots strategies of action.

Devices used in robot

Number in the diagram	Name in the program	Arduino pin
21	DIP1	7
21	DIP2	15
21	DIP3	16
21	DIP4	14



Steps of work

1. Add to the program so the blinking speed would change according to the combinations of the DIP switches.
2. Connect the robot to the computers USB port.
3. Upload the program to the robot.
4. Push the button and check if with them the LEDs blinking can be turned on and off.
5. Try out some of the DIP switch combinations and check the change in the blinking speed.
6. Save the program, from now on we will add to the existing code.

DIP1	DIP2	DIP3	DIP4	Speeds multiplier
oFF	oFF	oFF	oFF	× 0
oFF	oFF	oFF	on	× 1
oFF	oFF	on	oFF	× 2
oFF	oFF	on	on	× 3
oFF	on	oFF	oFF	× 4
oFF	on	oFF	on	× 5
oFF	on	on	oFF	× 6
oFF	on	on	on	× 7
on	oFF	oFF	oFF	× 8
on	oFF	oFF	on	× 9
on	oFF	on	oFF	× 10
on	oFF	on	on	× 11
on	on	oFF	oFF	× 12
on	on	oFF	on	× 13
on	on	on	oFF	× 14
on	on	on	on	× 15

```

void setup() { //setting up
    //LED definitions
    pinMode(12,OUTPUT); //Left LED
#define led1ON digitalWrite(12,HIGH)
#define led1OFF digitalWrite(12,LOW)
    pinMode(6,OUTPUT); //Right LED
#define led2ON digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
    pinMode(13,OUTPUT); //Back LED
#define led3ON digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
    pinMode(17,OUTPUT); //Orange LED
#define led4ON digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
    //button definitions
    pinMode(A2,INPUT);
    digitalWrite(A2,HIGH);
#define button1 !digitalRead(A2)
    //turning on series port monitor
    Serial.begin(9600);
    //DIP switch definition
    pinMode(7,INPUT);
    digitalWrite(7,HIGH);
#define DIP1 digitalRead(7)
    pinMode(15,INPUT);
    digitalWrite(15,HIGH);
#define DIP2 digitalRead(15)
    pinMode(16,INPUT);
    digitalWrite(16,HIGH);
    digitalRead(16)
    pinMode(14,INPUT);
    digitalWrite(14,HIGH);
#define DIP4 digitalRead(14)
}
int switchCounter =0;
//creates variable
//for counting pressed switches
int blinkingSpeed =0;
//creates variable for blinking speed
void loop() { //neverending cycle
digitalWrite(14,HIGH);
//shows high level
//so switch 4 could be read
led1OFF;
led2OFF;
led3OFF; //turns off all diodes in
//the beginning of the program

```

```

if (button1)
//turns on blinking diodes
//if button1 is pushed
{
while (button1){
//cycle while the button is pushed
Serial.println("Button 1 is pushed");
//outputs that button1 is pushed
}
while(!button1)
//blinking diodes until
//button1 isn't pushed
{
switchCounter = 0;
//changes switch calculation to 0
if(DIP1){
switchCounter =switchCounter+8;
} //if switch 1 is on adds 8
if(DIP2){
switchCounter =switchCounter+4;
}
if(DIP3){
switchCounter =switchCounter+2;
}
if(DIP4){
switchCounter =switchCounter+1;
}
blinkingSpeed =(switchCounter*100)+100;
//calculates blinking speed
led1ON;
led2ON;
led3ON; //turns on 3 diodes
delay(blinkingSpeed);
//waits as long as the blinking speed
led1OFF;
led2OFF;
led3OFF; //turns off 3 diodes
delay(blinkingSpeed);
//waits as long as the blinking speed
}
while (button1){
//cycle while button1 is pushed
Serial.println("Button 1 is pushed");
//outputs that button1 is pushed
}
}
}

```

7.4. WORKSHEET. Robots line sensors

Aim

Learn the use of mini-sumo's line sensors.

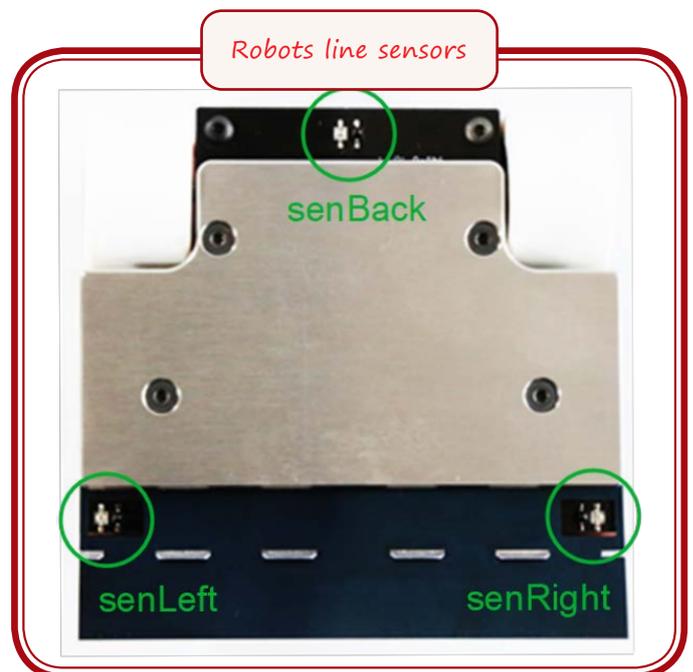
Devices used in robot

Number in the diagram	Name in the program	Arduino MC leg
17	senRight	A3
18	senLeft	A4
19	senBack	A5

Steps of work

1. Add to the program so one of the diodes would turn on when the line sensor is on a white surface.
2. Connect the robot to the computers USB port.
3. Upload the program to the robot.
4. Put the robot on a white surface and check if all of the diodes turn on.
5. If any of the diodes doesn't turn on you have to change the threshold value.
6. Put every sensor on a dark surface and check if the diode turns off.
7. Save the program.

Diode	Line sensor
led1	senLeft
led2	senRight
led3	senBack



```

void setup() { //setting up
    //LED definitions
    pinMode(12,OUTPUT); //Left LED
#define led1ON digitalWrite(12,HIGH)
#define led1OFF digitalWrite(12,LOW)
    pinMode(6,OUTPUT); //Right LED
#define led2ON digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
    pinMode(13,OUTPUT); //Back LED
#define led3ON digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
    pinMode(17,OUTPUT); //Orange LED
#define led4ON digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
    //button definitions
    pinMode(A2,INPUT);
    digitalWrite(A2,HIGH);
#define button1 !digitalRead(A2)
    //turnig on series port monitor
    Serial.begin(9600);
    //DIP switch definitions
    pinMode(7,INPUT);
    digitalWrite(7,HIGH);
#define DIP1 digitalRead(7)
    pinMode(15,INPUT);
    digitalWrite(15,HIGH);
#define DIP2 digitalRead(15)
    pinMode(16,INPUT);
    digitalWrite(16,HIGH);
    digitalRead(16)
    pinMode(14,INPUT);
    digitalWrite(14,HIGH);
#define DIP4 digitalRead(14)
    //line sensor definitions
    pinMode(A4,INPUT);
#define senLeft analogRead(A4)<37
    pinMode(A3,INPUT);
#define senRight analogRead(A3)<37

```

```

    pinMode(A5,INPUT);
#define senBack analogRead(A5)<37
}
void loop() { //neverending cycle
    led1OFF;
    led2OFF;
    led3OFF;
    //Turns off diodes in program beginning
    if (button1)
    //turns on blinking diodes if
    //button1 is pushed
    {
        while (button1){
            // cycle while the button is pushed
            Serial.println("Button 1 is pushed");
            // outputs when button1 is pushed
        }
        while(!button1)
        //blinking diodes until
        //button1 isn't pushed
        {
            if (senLeft){led1ON;}
            //if left sensor sees white
            //diode turns on
            else {led1OFF;} //otherwise the
            //diode turns off
            if (senRight){led2ON;}
            else {led2OFF;}
            if (senBack){led3ON;}
            else {led3OFF;}
        }
        while (button1){
            //cycle while the button is pushed
            Serial.println("Button 1 is pushed");
            //outputs when button1 is pushed
        }
    }
}

```

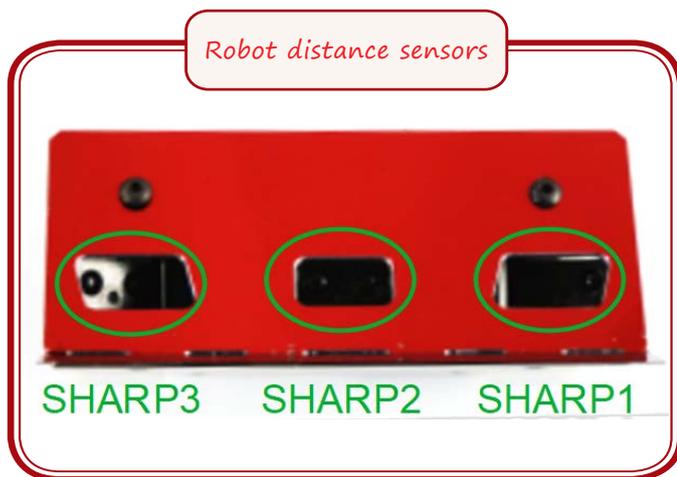
7.5. WORKSHEET. Robots distance sensors

Aim

Learn the use of mini-sumo's SHARP distance sensors.

Devices used in robot

Number in the diagram	Name in the program	Arduino pin
2	SHARP1	A0
5	SHARP2	A1
6	SHARP3	4



Steps of work

1. Add to the program so when the SHARP sensors turn on a notification shows up on the monitor.
2. Connect the robot to the computers USB port.
3. See if the diodes turn on when an object is put close to the SHARP sensor even without the program running.
4. Upload the program to the robot.
5. Turn on the series circuit monitor.
6. Check if the appropriate notification shows up after putting your hand to the specific sensor.
7. Save the program.

```

void setup() { //setting up
    //LED definitions
    pinMode(12,OUTPUT); //Left LED
#define led1ON digitalWrite(12,HIGH)
#define led1OFF digitalWrite(12,LOW)
    pinMode(6,OUTPUT); //Right LED
#define led2ON digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
    pinMode(13,OUTPUT); //Back LED
#define led3ON digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
    pinMode(17,OUTPUT); //Orange LED
#define led4ON digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
    //button definitions
    pinMode(A2,INPUT);
    digitalWrite(A2,HIGH);
#define button1 !digitalRead(A2)
    //turning on series port monitor
    Serial.begin(9600);
    //DIP switch definitions
    pinMode(7,INPUT);
    digitalWrite(7,HIGH);
#define DIP1 digitalRead(7)
    pinMode(15,INPUT);
    digitalWrite(15,HIGH);
#define DIP2 digitalRead(15)
    pinMode(16,INPUT);
    digitalWrite(16,HIGH);
#define DIP3 digitalRead(16)
    pinMode(14,INPUT);
    digitalWrite(14,HIGH);
#define DIP4 digitalRead(14)
    //line sensors definitions
    pinMode(A4,INPUT);
#define senLeft analogRead(A4)<37
    pinMode(A3,INPUT);
#define senRight analogRead(A3)<37
    pinMode(A5,INPUT);
#define senBack analogRead(A5)<37
    //SHARP sensors definitions
    pinMode(A0,INPUT); //SHARP1 left middle

```

```

#define SHARP1 !digitalRead(A0)
    pinMode(A1,INPUT); //SHARP2 middle
#define SHARP2 !digitalRead(A1)
    pinMode(4,INPUT); //SHARP3 right middle
#define SHARP3 !digitalRead(4)
}
void loop() { //neverending cycle
    led1OFF;
    led2OFF;
    led3OFF;
    //Turns off diodes in program beginning
    if (button1)
    //turns on blinking diodes
    //if button1 is pushed
    {
        while (button1){
            //cycle until button1 is pushed
            Serial.println("Button 1 is pushed");
            //outputs when button 1 is pushed
        }
        while(!button1)
            //blinking diodes while
            //button1 isn't pushed
        { //if there is an obstacle
            //a notification is shown
            if (SHARP1){Serial.println("Obstacle is
                in front at right side");}
            if (SHARP2){Serial.println("Obstacle is
                in front at middle");}
            if (SHARP3){Serial.println("Obstacle is
                in front at left side");}
            delay(500);
            //delays the program for half a second
            //so you would be able to read the text
        }
        while (button1){
            //cycle until button1 is pushed
            Serial.println("Button 1 is pushed");
            //outputs when button 1 is pushed
        }
    }
}

```

7.6. WORKSHEET. Robots motors

Aim

Learn the use of mini-sumo motors.

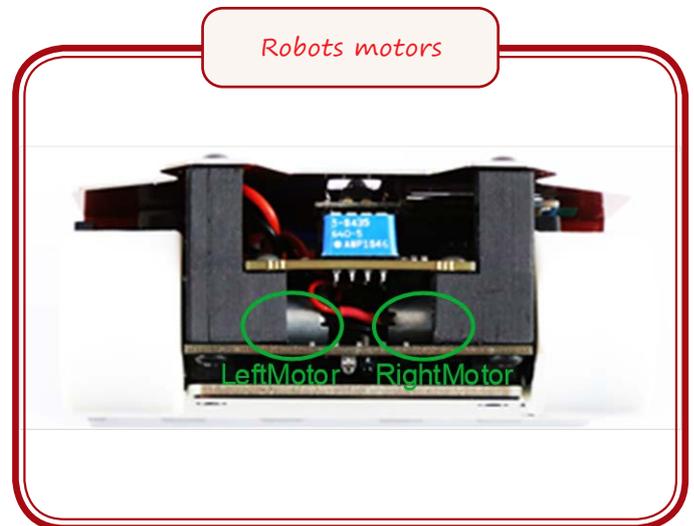
Devices used in robot

Number in the diagram	Name	Arduino pin
15	Right motors speed management pin	9
15	Right motors direction management pin	10
20	Left motors speed management pin	5
20	Left motors direction management pin	11

Steps of work

1. Add to the program so turning on a specific switch robot would go in specific direction.
2. Connect the robot to the computers USB port.
3. Upload the program to the robot.
4. Turn on only one of the switches.
5. Push button No1.
6. See if the motors are turning.
7. Check all the driving directions of the robot.
8. Check if robot will do the action of the last button when many of them are pushed.
9. Save the program.

Motors operations	Turning direction of the left motor	Turning direction of the right motor
Forward	↑	↑
Backward	↓	↓
Left	↓	↑
Right	↑	↓



```

void setup() { //setting up
                //LED definitions
    pinMode(12,OUTPUT); //Left LED
#define led1ON  digitalWrite(12,HIGH)
#define led1OFF digitalWrite(12,LOW)
    pinMode(6,OUTPUT); //Right LED
#define led2ON  digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
    pinMode(13,OUTPUT); //Back LED
#define led3ON  digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
    pinMode(17,OUTPUT); //Orange LED
#define led4ON  digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
    //button definitions
    pinMode(A2,INPUT);
    digitalWrite(A2,HIGH);
#define button1 !digitalRead(A2)
    //turning on series port monitor
    Serial.begin(9600);
    //DIP switch definitions
    pinMode(7,INPUT);
    digitalWrite(7,HIGH);
#define DIP1  digitalRead(7)
    pinMode(15,INPUT);
    digitalWrite(15,HIGH);
#define DIP2  digitalRead(15)
    pinMode(16,INPUT);
    digitalWrite(16,HIGH);
                digitalWrite(16)
    pinMode(14,INPUT);
    digitalWrite(14,HIGH);
#define DIP4  digitalRead(14)
    //line sensors definitions
    pinMode(A4,INPUT);
#define senLeft  analogRead(A4)<37
    pinMode(A3,INPUT);
#define senRight analogRead(A3)<37
    pinMode(A5,INPUT);
#define senBack  analogRead(A5)<37
    //SHARP sensors definitions
    pinMode(A0,INPUT);//SHARP1 left middle
#define SHARP1 !digitalRead(A0)
    pinMode(A1,INPUT);//SHARP2 middle
#define SHARP2 !digitalRead(A1)
    pinMode(4,INPUT);//SHARP3 right middle
#define SHARP3 !digitalRead(4)
    //motors definition
    pinMode(5, OUTPUT);
    pinMode(11, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
}

//motors managements function definitions
//function for driving forward
void Forward (int leftSpeed, int rightSpeed)
{
    analogWrite(5, leftSpeed); //left speed
    digitalWrite(11, LOW);
    //left sides direction
    analogWrite(9, rightSpeed); //right speed
    digitalWrite(10, LOW);
    //right sides direction
}
void Backward (int leftSpeed, int rightSpeed)
{
    analogWrite(11, leftSpeed);
    digitalWrite(5, LOW);
    analogWrite(10, rightSpeed);
    digitalWrite(9, LOW);
}
void Left (int leftSpeed, int rightSpeed)
{
    analogWrite(11, leftSpeed);
    digitalWrite(5, LOW);
    analogWrite(9, rightSpeed);
    digitalWrite(10, LOW);
}
void Right (int leftSpeed, int rightSpeed)
{
    analogWrite(5, leftSpeed);
    digitalWrite(11, LOW);
    analogWrite(10, rightSpeed);
    digitalWrite(9, LOW);
}
int senState = 0; //variable for robots
operations
void loop() { //neverending cycle
    //turns off all the diodes in
    //the beginning of the program
    led1OFF;
    led2OFF;
    led3OFF;
    Forward(0,0); //stops motors
    //turning on appropriate DIP switch
    //motors change turning direction
    while(DIP1) {
        Forward(100, 100);
    }
    while(DIP2) {
        Backward(100, 100);
    }
    while(DIP3) {
        Left(100, 100);
    }
    while(DIP4) {
        Right(100, 100);
    }
}

```

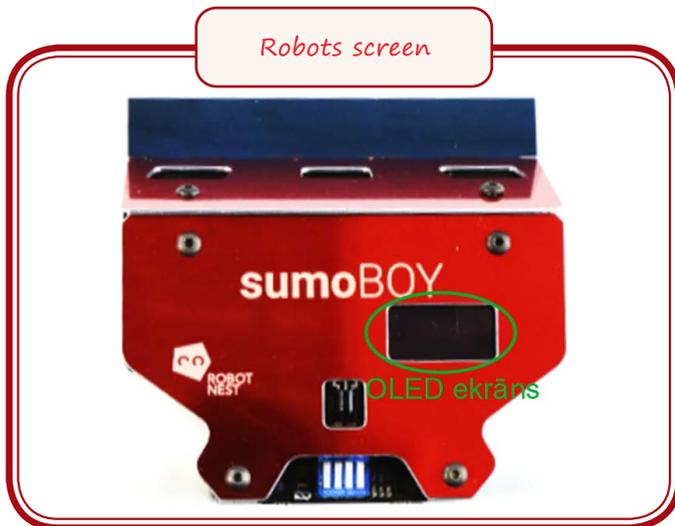
7.7. WORKSHEET. Robots screen

Aim

Learn the use of robots OLED screen.

Devices used in robot

Number in the diagram	Name in the program	Arduino pin
23	OLED SDA	2
24	OLED SCL	3



Steps of work

1. Add libraries to the program to ensure that the screen works.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
```

2. Write the screens definition to the other definitions meanwhile also setting up the colour and size of the output letters.

```
void setup() { //setting up
//setting up the screen
display.begin(SSD1306_SWITCHCAPVCC, 0x3C,
false);
display.setTextSize(1); //text size
display.setTextColor(WHITE); //text colour
// - white
```

3. Add to the program so when the line sensors turn on the screen shown values read by the sensors.

```
void loop() { //neverending cycle
// output sensor values to the screen
display.setCursor(0, 0);
display.clearDisplay();
display.print("SenLeft: ");
display.println(analogRead(A4));
display.print("SenRight: ");
display.println(analogRead(A3));
display.print("SenBack: ");
display.println(analogRead(A5));
display.display();
delay(100);
display.clearDisplay();
```

4. Connect the robot to the computers USB port.
5. Upload the program to the robot.
6. See if the values read by the line sensors show up on the screen.
7. Save the program.

8. MINI-SUMO COMPETITION

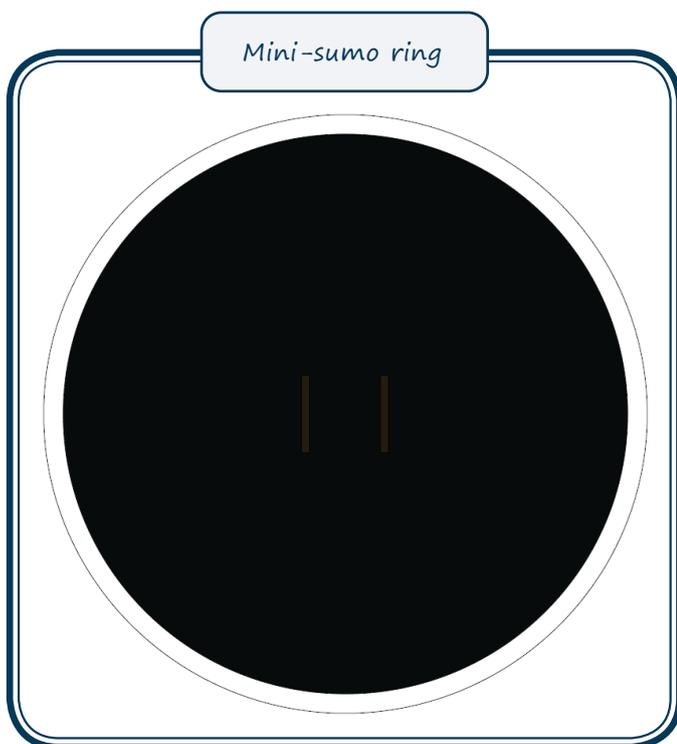
8.1. Aim

In this chapter we will briefly look at the rules for mini-sumo competitions. We will create a simple fighting program based on these rules with which you will be able to compete in both regional and international mini-sumo competitions.

8.2. Rules for mini-sumo competitions

Mini-sumo is a robot competition discipline which is based on the Japanese sumo battles. Normally during a robot sumo battle two robots compete in a round ring. Robots independently battle in the ring until one gets pushed out of it. The winner is the robot still in the ring.

Mini-sumo battle ring is a black circle with the diameter of 72 cm and 2.5 cm wide white line around it. In the centre of the ring there are two parallel dark brown stripes.



Parameters for mini-sumo robots - they determine conditions that have to be met for the robot to be allowed to compete. If any of the parameters aren't met then the robot isn't allowed to compete. These parameters are:

- Weight up to 500g.
- Size 10 cm x10 cm.
- Weight is unlimited.
- Robot has to be independent (while the robot operates human is forbidden to intervene in any way).
- Robot activates 5 seconds after pushing the turn-on button or receiving a remote signal.
- Robot isn't allowed to damage the ring.
- Robot isn't allowed to affect the operations of its opponent.
- No parts are allowed to separate from the robot.

Mini-sumo competitions process - competitions in different countries are very similar with some minor differences. The process of the competition can be overall explained with these actions:

- Two competitors head to a ring with their robots after a judge has called them (spectators have to be further away from the ring so they wouldn't disturb the robots and would be safe).
- After the judge's signal both competitors put their robots in the ring on their side of their line.
- After the start signal both competitors push the START button and step away from the ring.
- After 5 seconds of the judge's remote signal robots start the battle.
- The first robot that is pushed out of the ring and touches the surface outside the ring loses.
- If the robot doesn't move or the battle gets too long the judge can stop the battle and determine the winner.
- These battles happen three times to determine the best robot.

Every competition has different rules so it is important to read the regulations and abide by them.

8.1. WORKSHEET. Mini-sumo settings

Aim

Get ready for developing mini-sumo's management program.

In the following chapters the code will be changed only in the main cycle (function loop()) and different variables will be added.

Steps of work

1. Write or use void setup() functions section motors functions from the previous chapter.
2. Copy the programs code and check its correctness.

```
//to use the screen you have
//to use Adafruit libraries:
//https://learn.adafruit.com/monochrome-
oled-breakouts/arduino-library-and-examples
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
void setup() { //setting up
  //setting up the screen
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C,
false);
  display.setTextSize(1);
  display.setTextColor(WHITE);
  //LED definitions
  pinMode(12, OUTPUT); //Left LED
#define led1ON digitalWrite(12,HIGH);
#define led1OFF digitalWrite(12,LOW);
  pinMode(6, OUTPUT); //Right LED
#define led2ON digitalWrite(6,HIGH);
#define led2OFF digitalWrite(6,LOW);
  pinMode(13, OUTPUT); //Back LED
#define led3ON digitalWrite(13,HIGH);
#define led3OFF digitalWrite(13,LOW);
  pinMode(17, OUTPUT); //Orange_LED
#define led4ON digitalWrite(17,HIGH);
#define led4OFF digitalWrite(17,LOW);
  //buttons definitions
  pinMode(A2, INPUT);
  digitalWrite(A2, HIGH);
#define button1 !digitalRead(A2)
  //turning on series port monitor
  Serial.begin(9600);
  //DIP switch definitions
  pinMode(7, INPUT);
  digitalWrite(7, HIGH);
#define DIP1 !digitalRead(7)
```

```
  pinMode(15, INPUT);
  digitalWrite(15, HIGH);
#define DIP2 !digitalRead(15)
  pinMode(16, INPUT);
  digitalWrite(16, HIGH);
#define DIP3 !digitalRead(16)
  pinMode(14, INPUT);
  digitalWrite(14, HIGH);
#define DIP4 !digitalRead(14)
  //Linijas sensoru definicijas
  pinMode(A4, INPUT);
#define senLeft analogRead(A4)<35
  pinMode(A3, INPUT);
#define senRight analogRead(A3)<35
  pinMode(A5, INPUT);
#define senBack analogRead(A5)<35
  //SHARP sensors definitions
  pinMode(A0, INPUT); //SHARP1 left middle
#define SHARP1 !digitalRead(A0)
  pinMode(A1, INPUT); //SHARP2 middle
#define SHARP2 !digitalRead(A1)
  pinMode(4, INPUT); //SHARP3 right middle
#define SHARP3 !digitalRead(4)
  //Motoru definicijas
  pinMode(5, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
}
//motors management function definitions
void Forward (int leftSpeed, int rightSpeed){
  analogWrite(5,leftSpeed);
  digitalWrite(11,LOW);
  analogWrite(9,rightSpeed);
  digitalWrite(10,LOW);
}
void Backward (int leftSpeed, int rightSpeed){
  analogWrite(11,leftSpeed);
  digitalWrite(5,LOW);
  analogWrite(10,rightSpeed);
  digitalWrite(9,LOW);
}
void Left (int leftSpeed, int rightSpeed){
  analogWrite(11,leftSpeed);
  digitalWrite(5,LOW);
  analogWrite(9,rightSpeed);
  digitalWrite(10,LOW);
}
void Right (int leftSpeed, int rightSpeed){
  analogWrite(5,leftSpeed);
  digitalWrite(11,LOW);
  analogWrite(10,rightSpeed);
  digitalWrite(9,LOW);
}
void loop() { //Mūžīgais cikls
}
```

8.2. WORKSHEET. Robot that avoids obstacles

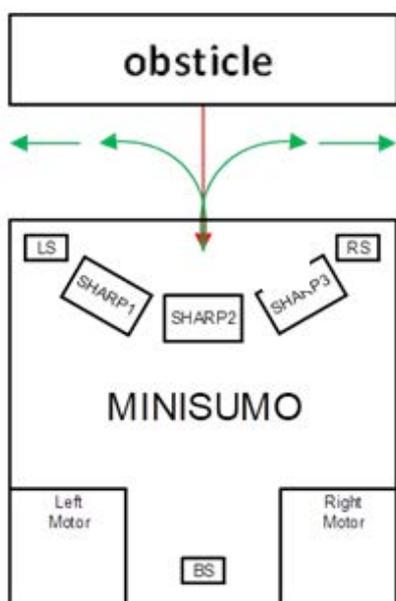
Aim

Create a managing program which allows the robot to avoid obstacles in front of it.

Steps of work

1. Create robots operation plan.
After noticing an obstacle with front sensor SHARP3 the robot must drive backwards and turn.
2. Create management program.
3. Connect the robot to the computers USB port.
4. Upload the program to the robot.
5. Put a robot on a clear surface so it would fall.
6. Turn on the robot with the button.
7. Check if the robot drives backward and turns left after spotting an obstacle.
8. You have to remember to be cautious with `delay(200)` because while delayed the robot doesn't respond to sensors signals and blindly does the last action.

Robots action plan after spotting an obstacle



```
//programs settings from first worksheet

int motState =0; //variable in which we will
                //show motors operations

void loop() { //neverending cycle
  Forward(0,0);
  //call function for stopping the motors
  if (button1)
  //robot starts working after button1 is
  {
                                //pushed

    while (button1){ //cycle while the
                    //button is pushed
    }
    while (!button1)
    //robot stops after button1 is pushed
    {
      if (SHARP3) {
        motState=1;
      } // if previous sensor activates
        //variable is put as 1
      else {
        motState=0;
      } //if previous sensor doesn't
        //activate variable is put as 0
      switch (motState){
        //Switch structure reads variable
        case 0: //if variable is 0 robot
              //drives forward
          Forward(60,60);
          break;
        case 1: //if the variable is 1
              //robot drives backward and turns
          Backward(60,60);
          delay(200);
          // 200 ms goes backwards
          Left(60,60);
          delay(200); //200 ms - turns
          break;
      }
    }
    while (button1){ //cycle while the
                    //button is pushed
    }
  }
}
```

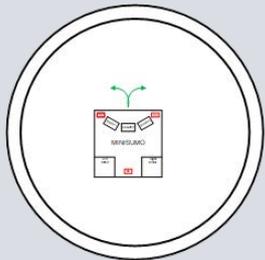
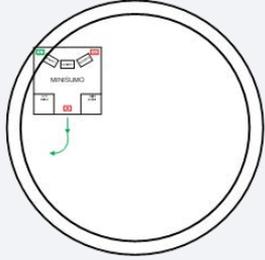
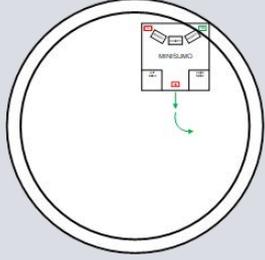
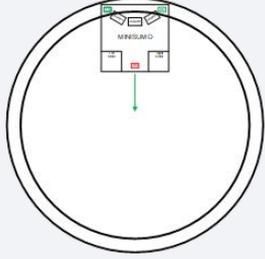
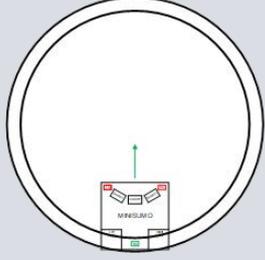
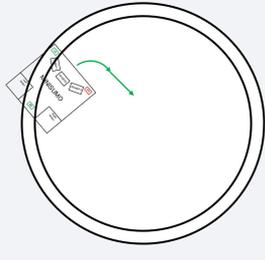
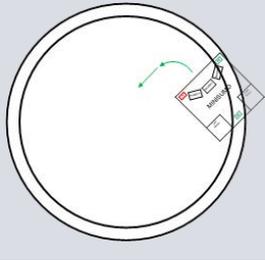
8.3. WORKSHEET. Program for not driving out of the ring

Aim

Create a management program that doesn't allow the robot to drive off the ring.

Steps of work

1. Put the robot on the ring and check in how many different combinations can the line sensors work.
Sensors work after spotting the white line.
2. Think of different strategies to avoid these situations.
3. Create a management program while keeping in mind the observed movements.
4. Connect the robot to the computers USB port.
5. Upload the program to the robot.
6. Place the robot on the ring.
7. Turn on the robots program.
8. See how the robot works in the ring; topping at the white boundary and changing its trajectory.
9. If the robot doesn't stop at the white boundary you have to change the delay values for motors speed and time.

No	Image	senBack	senRight	senLeft	Action
0.		oFF	oFF	oFF	Turn right, turn left, drive forward.
1.		oFF	oFF	on	Drive backwards, turn right while driving backwards.
2.		oFF	on	oFF	Drive backwards, turn left while driving backwards.
3.		oFF	on	on	Drive backwards.
4.		on	oFF	oFF	Drive forward.
5.		on	oFF	on	Turn right, drive forward.
6.		on	on	oFF	Turn left, drive forward.
7.	Impossible on the ring	on	on	on	Stop.

```

// beginning of the program from the
//previously made program
int motState =0;
//variable in which we will show robots
//operations
void loop() { //neverending cycle
    //outputs sensors values on the screen
    display.setCursor(0, 0);
    display.clearDisplay();
    display.print("SenLeft: ");
    display.println(analogRead(A4));
    display.print("SenRight: ");
    display.println(analogRead(A3));
    display.print("SenBack: ");
    display.println(analogRead(A5));
    display.display();
    delay(100);
    display.clearDisplay();
    //motors change their turningdirection
    //after turning the appropriate switch
    while (DIP1) {
        Forward(45,45);
    }
    while (DIP2) {
        Backward(45,45);
    }
    while (DIP3) {
        Left(45,45);
    }
    while (DIP4) {
        Right(45,45);
    }
    if (senLeft) {
        led1ON; //add sensor binary value to
    } else { //motors position
        led1OFF;
    }
    if (senRight) {
        led2ON;
    } else {
        led2OFF;
    }
    if (senBack){
        led3ON;
    } else {
        led3OFF;
    }
    Forward(0, 0); //calls function for
    //stopping the motors
    //outputs sensors values to series monitor
    Serial.print("SenLeft: ");
    Serial.println(analogRead(A4));
    Serial.print("SenRight: ");
    Serial.println(analogRead(A3));
    Serial.print("SenBack: ");
    Serial.println(analogRead(A5));
    Serial.print("S1: ");
    Serial.println(digitalRead(A0));
    Serial.print("S2: ");
    Serial.println(digitalRead(A1));
    Serial.print("S3: ");
    Serial.println(digitalRead(4));

```

```

    if (button1) //robot starts working
    //after pushing Button1
    {
        while (button1) {
            //cycle while the button is pushed
            }
            led1ON;
            delay(350);
            led2ON;
            delay(350);
            led3ON;
            delay(350);
            led1OFF;
            delay(350);
            led2OFF;
            delay(350);
            led3OFF;
            while (!button1)
            //robot stops after pushing button1
            {
                senState = 0;
                //makes variable zero
                if (senLeft) {
                    led1ON;
                    senState = senState + 1;
                } else {
                    led1OFF;
                    //adds to the motors position
                    //sensors binary value
                }
                if (senRight) {
                    led2ON;
                    senState = senState + 2;
                } else {
                    led2OFF;
                }
                if (senBack) {
                    led3ON;
                    senState = senState + 4;
                } else {
                    led3OFF;
                }
                if (senState == 0) {
                    //looks at the SHARP sensor if
                    //no line sensor has activated
                    if (SHARP1) {
                        senState = +8;
                    }
                    if (SHARP2) {
                        senState += 16;
                    }
                    if (SHARP3) {
                        senState += 32;
                    }
                }
            }
            display.setCursor(0, 0);
            display.clearDisplay();
            display.print("SenLeft: ");
            display.println(senLeft);
            display.print("SenRight: ");
            display.println(senRight);
            display.print("SenBack: ");
            display.println(senBack);

```

```

display.print("SenState: ");
display.println(senState); d
isplay.display();
display.clearDisplay();

//"senState" glabā sensoru stāvokļa
//vērtību
switch (senState) {
//Switch struktūra nolasa mainīgo
  case 0:
    //no sensor has activated
    Forward(35,35);
    break;
  case 1: //left sensor
    Backward(45,45);
    delay(250);
    Right(45,45);
    delay(150);
    break;
  case 2://right sensor
    Backward(45,45);
    delay(250);
    Left(45,45);
    delay(150);
    break;
  case 3://both front
    Backward(45,45);
    delay(350);
    break;
  case 4://back
    Forward(45,45);
    delay(350);
    break;
  case 5://back and left
    Right(45,45);
    delay(350);
    Forward(45,45);
    delay(250);
    break;
  case 6://back and right
    Left(45,45);
    delay(350);
    Forward(45,45);
    delay(250);
    break;
  case 7:
    //all sensors have activated
    Forward(0,0);
    break;
}
}
while (button1) {
//cycle while the button is pushed
}
}
}

```

8.4. WORKSHEET. Program for sumo battles

Aim

Create robots fighting program using the three front SHARP sensors.

Steps of work

1. Turn on the robot.
2. Think of ways how the robot should act after spotting the opponent with its front SHARP sensors (use the diodes that light up after sensor is activated).
3. Think of strategies that would successfully push the opponent out of the ring after it's been spotted.
4. Create a management program while keeping in mind the observed movements.
5. Connect the robot to the computers USB port.
6. Upload the program to the robot.
7. Put the robot on the ring.
8. Turn on the robots program.
9. Wait 5 seconds for the robot to start working.
10. Put a cardboard box or another robot in the ring and let your robot push it out.
11. Add the other SHARP sensors to the program code and work out your strategy.
12. Tweak with the motors speed values.
13. Participate in the mini-sumo competition!

No	Image	SHARP3	SHARP2	SHARP1	Action
8.		on	oFF	oFF	Turns left
16.		oFF	on	oFF	Goes forward
24.		on	on	oFF	Goes forward and slightly left
32.		oFF	oFF	on	Turns right
40.		on	oFF	on	Goes forward
48.		oFF	on	on	Goes forward and slightly right
56.		on	on	on	Quickly goes forward

```
// beginning of the program from the //
previously made program
int motState =0;
//variable in which we will show robots //
operations
void loop() { //neverending cycle

//outputs sensor values to the screen
display.setCursor(0, 0);
display.clearDisplay();
display.print("SenLeft: ");
display.println(analogRead(A4));
display.print("SenRight: ");
display.println(analogRead(A3));
display.print("SenBack: ");
display.println(analogRead(A5));
display.display();
delay(100);
display.clearDisplay();
//motors change direction after turning
//on appropriate DIP
while (DIP1) {
    Forward(45,45);
}
while (DIP2) {
    Backward(45,45);
}
while (DIP3) {
    Left(45,45);
}
while (DIP4) {
    Right(45,45);
}
if (senLeft) {
    led1ON; //adds sensors binary value
                //to motors position
} else {
    led1OFF;
}
if (senRight) {
    led2ON;
} else {
    led2OFF;
}
if (senBack) {
    led3ON;
} else {
    led3OFF;
}
    Forward(0, 0); //call function for
                //stopping the motors

//outputs sensors values to series monitor
Serial.print("SenLeft: ");
Serial.println(analogRead(A4));
Serial.print("SenRight: ");
Serial.println(analogRead(A3));
Serial.print("SenBack: ");
Serial.println(analogRead(A5));
Serial.print("S1: ");
Serial.println(digitalRead(A0));
Serial.print("S2: ");
```

```
Serial.println(digitalRead(A1));
Serial.print("S3: ");
Serial.println(digitalRead(4));
if (button1)
//robot starts working with pushing Button1
{
    while (button1) {
        //cycle while the button is pushed
        }
        led1ON;
        delay(350);
        led2ON;
        delay(350);
        led3ON;
        delay(350);
        led1OFF;
        delay(350);
        led2OFF;
        delay(350);
        led3OFF;

        while (!button1)
//robot stops with pushing button1
        {
            senState = 0;
            //set variable to zero
            if (senLeft) {
                led1ON;
                senState = senState + 1;
            } else {
                led1OFF;
                //adds to motors position
                //sensors binary value
            }
            if (senRight) {
                led2ON;
                senState = senState + 2;
            } else {
                led2OFF;
            }
            if (senBack) {
                led3ON;
                senState = senState + 4;
            } else {
                led3OFF;
            }
            if (senState == 0) {
                // looks at SHARP sensor, if no
                //line sensor has activated
                if (SHARP1) {
                    senState = +8;
                }
                if (SHARP2) {
                    senState += 16;
                }
                if (SHARP3) {
                    senState += 32;
                }
            }
        }
        display.setCursor(0, 0);
        display.clearDisplay();
        display.print("SenLeft: ");
        display.println(senLeft);
```

```

display.print("SenRight: ");
display.println(senRight);
display.print("SenBack: ");
display.println(senBack);
display.print("SenState: ");
display.println(senState);
display.display();
display.clearDisplay();
// "senState" saves sensors
// positions value
switch (senState) {
// Switch structure reads
// variable
  case 0:
    // no sensor has activated
    Forward(35, 35);
    delay(150);
    Right(45, 45);
    break;
  case 1: // left sensor
    Backward(45, 45);
    delay(250);
    Right(45, 45);
    delay(150);
    break;
  case 2: // right sensor
    Backward(45, 45);
    delay(250);
    Left(45, 45);
    delay(150);
    break;
  case 3: // both front
    Backward(45, 45);
    delay(350);
    break;
  case 4: // back
    Forward(45, 45);
    delay(350);
    break;
  case 5:
    // back and left
    Right(45, 45);
    delay(350);
    Forward(45, 45);
    delay(250);
    break;

```

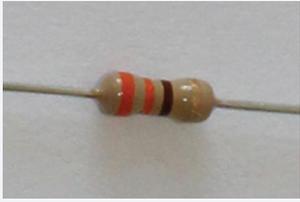
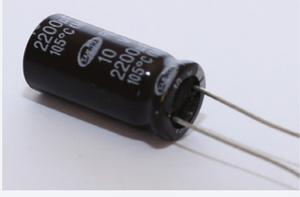
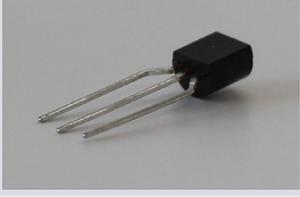
```

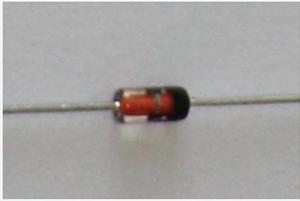
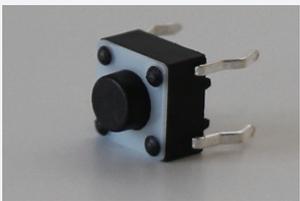
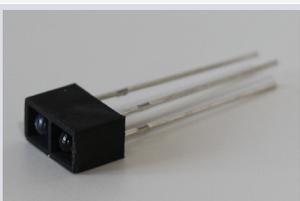
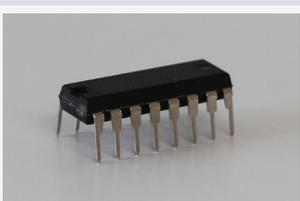
  case 6: // back and right
    Left(45, 45);
    delay(350);
    Forward(45, 45);
    delay(250);
    break;
  case 7:
    // all sensors have
    // activated
    Forward(0, 0);
    break;
  // if robots SHARP sensors
  // have activated
  case 8: // SHARP1
    Right(80, 80);
    break;
  case 16: // SHARP2
    Forward(100, 100);
    break;
  case 24: // SHARP1 & SHARP2
    Forward(100, 80);
    break;
  case 32: // SHARP3
    Left(80, 80);
    break;
  case 40: // SHARP1 & SHARP3
    Forward(100, 100);
    break;
  case 48: // SHARP2 & SHARP3
    Forward(80, 100);
    break;
  case 56:
    // SHARP1 & SHARP2 & SHARP3
    Forward(150, 150);
    break;
}
}
while (button1) {
  // cycle while the button is pushed
}
}

```

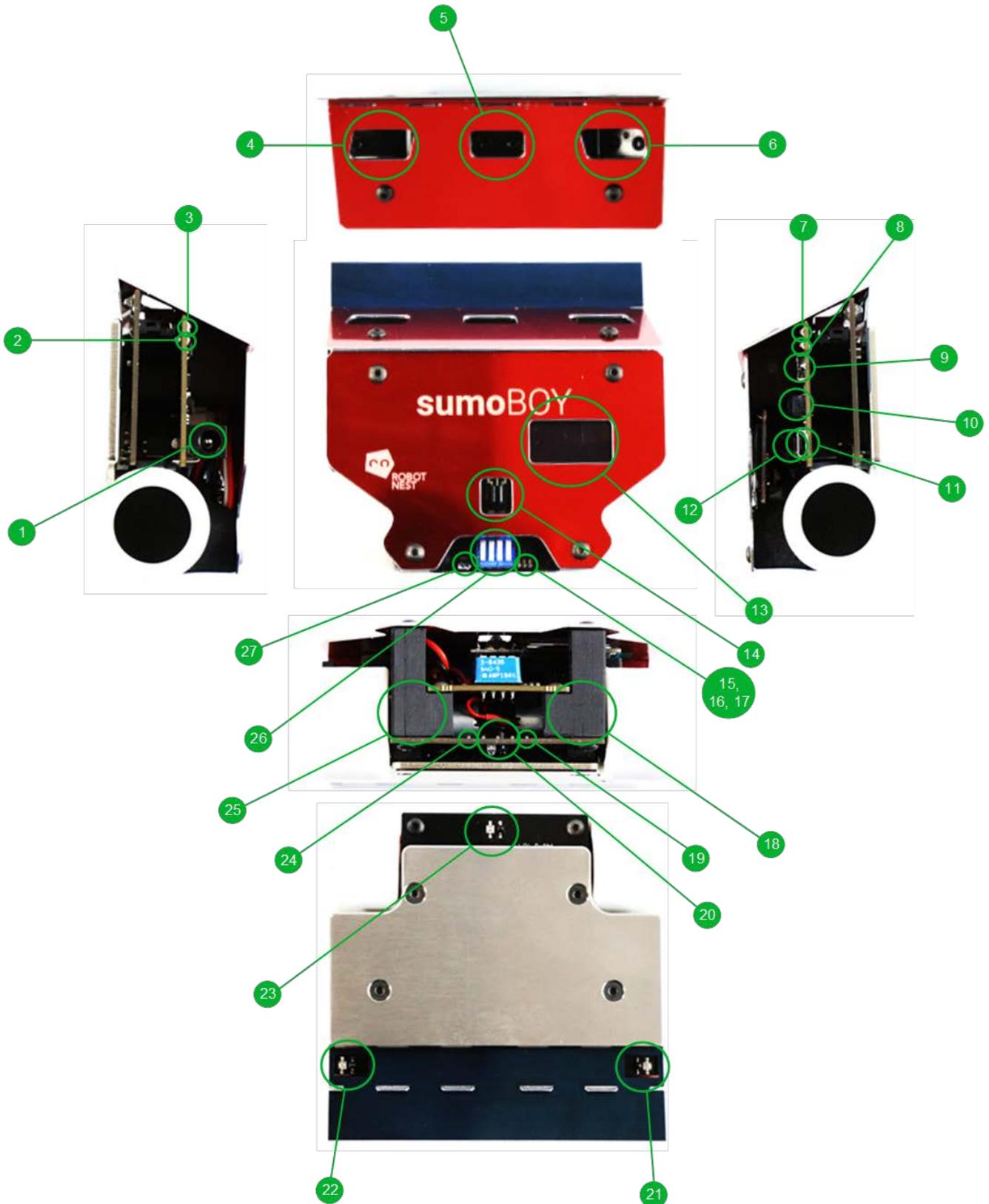
1. Annex

List of parts and their images

Material/part	Image	Amount	Material/part	Image	Amount
Resistor (200 Ω)		5	Potentiometer (50 k Ω)		3
Resistor (330 Ω)		10	Termistor NTC (2.2 k Ω)		1
Resistor (1 k Ω)		5	Capacitor (15 pF)		2
Resistor (4.7 k Ω)		5	Electrolytic capacitor (100 μ F, 63 V)		2
Resistor (10 k Ω)		5	Electrolytic capacitor (2200 μ F, 10 V)		2
Resistor (47 k Ω)		5	Transistor NPN BC517		3
Photo-resistor (20 k Ω)		1			

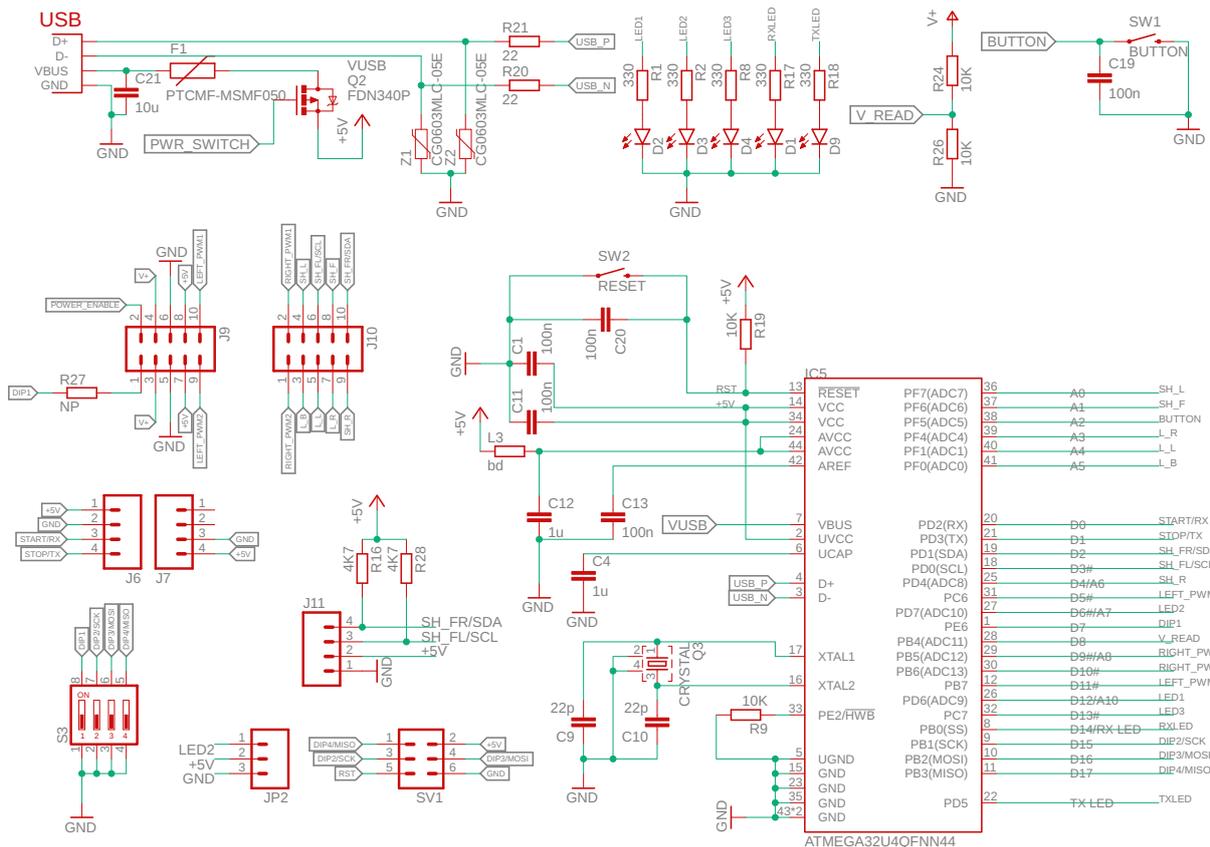
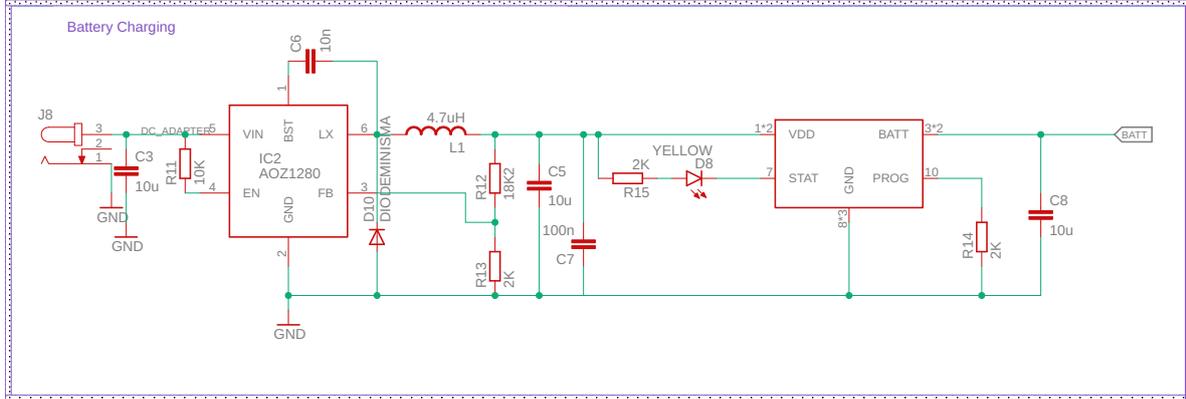
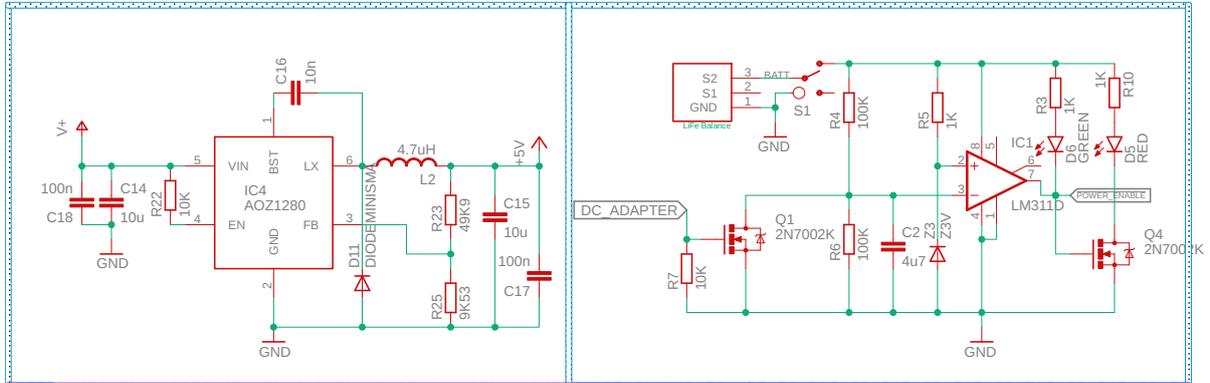
Material/part	Image	Amount	Material/part	Image	Amount
Diode PH4148		5	Mounting cords		2 packs
LED		6	Button		2
RGB LED		1	Switch		2
Optocoupler ELITR9909		1	Permanent-magnet DC motor (3-6 V)		1
H gridge microchip L203D		1	Srevomotor		1

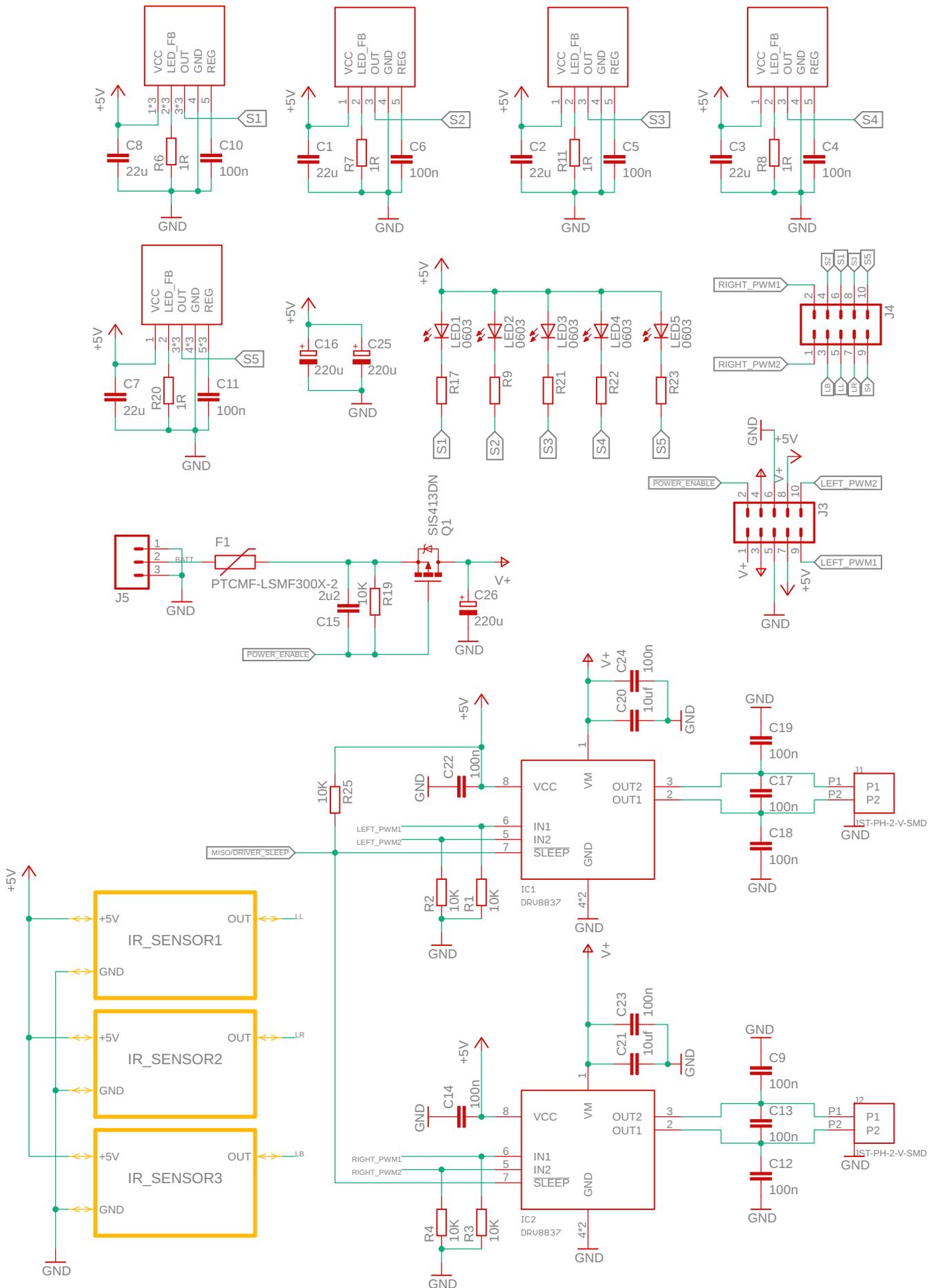
2. Annex Robots input and output devices



No	Name	Use	Arduino leg
1.	POWER SUPPLY	Port for connecting charger	–
2.	CHERGING LED	Shows that the robot is charging	–
3.	LED1	Programmable diode for information output	12
4.	SHARP1	Looking for the opponnet	A0
5.	SHARP2	Looking for the opponnet	A1
6.	SHARP3	Looking for the opponnet	4
7.	LED2	Programmable diode for information output	6
8.	POWERLED	Shows that the robot is on and working on battery	–
9.	ON/OFF SWITCH	On and off switch for the battery	–
10.	BUTTON1	Programmable button	A2
11.	MICRO USB	Arduino controllers USB cord for uploading program, power, series communication	–
12.	RESET	Button for resetting the program	–
13.	OLED screen	Programmable screen for informations output (SDA 2, SCL 3)	2,3
14.	IR RECIEVER	IR module for starting 0(HIGHT) START and stoping 1(LOW) STOP the robot with a remote	–
15.	LED4	Programmable diode for information output	17
16.	LED3	Programmable diode for information output	13
17.	TX LED	Series port indicator	–
18.	MOTOR2	Controllable 6V direct currents motor. Motors speed 9, direction 10	9,10
19.	LED SDA	I2C SDA indicator	0
20.	SHARPLED	Light up if something comes in the SHARP field of vision. SHARP1-SHARP3	0
21.	SENRIGHT	Optocoupler for detecting mini-sumo rings edge	A3
22.	SENLRFT	Optocoupler for detecting mini-sumo rings edge	A4
23.	SENBACK	Optocoupler for detecting mini-sumo rings edge	A5
24.	LED SCL	I2C SCL indicator	–
25.	MOTOR1	Controllable 6V direct currents motor. Motors speed 5, direction 11	5,11
26.	DIPSWITCH1,2,3,4	Programmable DIP switch for robots strategies DIP1 (7), DIP2 (15), DIP3 (16), DIP4 (14)	7,15,16,14
27.	EMPTY BATTERY	Diode that singals low battery	–

3. Annex Robots overall scheme





4. Annex

Learn soldering

P4.1. Aim

The aim for this topic is to teach soldering basics and safety measures so the soldering would be safe and effective.

P4.2. Theoretical part

Ievads

Soldering is one of the most important skills in the world of electronics. The basics of electronics can be learned without this skill, but soldering gives opportunity to make more interesting projects and join the electronics enthusiast group. More and more of electric and electrotechnical devices are used that is why this skill is so important. A big part of electronics enthusiast's everyday life is not only to understand, but also to make, repair and add to electric device.

In this chapter we will learn the basics of soldering concentrating on plated through-hole soldering - PTH. We will look at the materials and tools as well as repairing already soldered plate.

Materials used

The main soldering material is solder. This is a mixture of different soft materials which usually resembles a metal wire. It's usually rolled up in a spool or in other easy-to-use packages.

In this chapter we will learn the basics of soldering concentrating on plated through-hole soldering - PTH. We will look at the materials and tools as well as repairing already soldered plate.

You could choose the diameter and chemical contents of your solder according to your task. Without diving in too deep the solder is divided in containing lead and not containing lead. Historically lead (Pb) mixed with tin (Sn) is used to create solder with lower melting temperature and better flow which is important for good connection between parts.

Since 2006 many countries have forbidden using lead containing solder, because of nature's protection and human health. With direct contact lead accumulates in human bodies and in big amounts can be poisonous. That is why it is important to remember - after using lead containing solder you should **carefully wash your hands**.

To avoid the risk of getting lead in your body you can use solder with no lead in it. But you have to take in account that its melting temperature is higher and grip to other materials is lower. For improving the grip you can use fluxes. There are solders with fluxes in their core so you don't have to buy them.

The diameter of solder is determined by the parts you want to solder. The bigger the part, the bigger the diameter. For these tasks the best is solder with its diameter no bigger than 1.0 mm.

Solder packages



Tools used

For soldering you will need a soldering iron, soldering irons stand, as well as different tools for taking of solder and holding parts.

Soldering iron is an electrical heater which heats the solder to its melting temperature. As with every electrical device you should follow its instructions provided by its manufacturer. There are hot-air and gas soldering irons for specific tasks, but we will look only at the electric ones.

Soldering irons are very diverse to effectively do the task at hand. They have many tips, power settings, temperatures and many adjustment possibilities. However, the main condition is the comfort of its use. That is why you should choose a comfortable soldering iron for you. If the soldering iron is too big you won't be able to solder small details. If it is too small it most likely won't be able to properly heat-up the parts and the solder won't be able to stick to them. It is advised to use a soldering iron with a conical tip, because it will be easier to use for beginner.

Soldering iron with a conical tip



Solder wick - allows easily taking off the extra solder from a plate or part because it sucks up all of the melted solder. It consists of fine copper wires that are twisted together.

Solder wick



Solder vacuum - allows taking of the extra solder using vacuum.

Solder vacuum



Third arm - a part and other tool holder which is especially useful for a beginner.

Third arm



Soldering

Is it hard to explain soldering in written form because the process is highly connected to individual skills and attention. However, there are some conditions for a good soldering job.

1. Be careful with the hot soldering iron.
2. Keep your workspace clean, don't eat in your workspace (remember about the lead intake).
3. Use the third arm when ever it is useful.
4. If it is possible keep the soldering irons temperature around 350°
5. If you see smoke coming from the soldering iron, lower its temperature or turn it off.
6. Use a special soldering irons cleaner (a wet sponge or special paste for the tips cleaning) before every new soldering.
7. Use the sides of the soldering iron not the very tip.
8. To ensure a better connection try to heat up both of the parts.
9. After the parts have been soldered first take away the soldering wire and only then the soldering iron.
10. A good soldering job looks like a volcano not a ball or a pile of solder.

P4.1 WORKSHEET

Aim

Solder your first part in the supervision of your teacher.

Materials needed

Material/part	Amount
Soldering iron	1
Solder wire	1
Flux	1
Soldering irons cleaner	1
Parts to be soldered	As much as needed

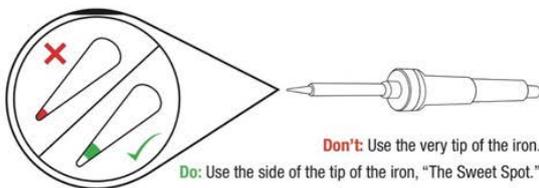
Materials needed for the task



Steps of work

1. Prepare the workspace so the heated soldering iron wouldn't be able to cause burns and you would have enough free space to use both hands.
2. Prepare the part to be soldered - small pieces of wire, old electronics parts or other materials that can be soldered which the teacher has provided.
3. Clean off the oxide layer from the parts. This can be done with the help of the flux (if it is separated from the solder) or abrasive material, for example, fine sandpaper. You don't need to clean the parts if you solder has flux in it.
4. Solder according to your teachers instructions.
5. Let the parts cool down and see if the soldering is done correctly.

Lodšanas tehnika



Do: Touch the iron to the component leg and metal ring at the same time.



Do: While continuing to hold the iron in contact with the leg and metal ring, feed solder into the joint.



Don't: Glob the solder straight onto the iron and try to apply the solder with the iron.



Do: Use a sponge to clean your iron whenever black oxidation builds up on the tip.



A Solder flows around the leg and fills the hole - forming a volcano-shaped mound of solder.



B Error: Solder balls up on the leg, not connecting the leg to the metal ring. Solution: Add flux, then touch up with iron.



C Error: Bad Connection (i.e. it doesn't look like a volcano) Solution: Flux then add solder.



D Error: Bad Connection...and ugly...oh so ugly. Solution: Flux then add solder.



E Error: Too much solder connecting adjacent legs (aka a solder jumper). Solution: Wick off excess solder.

