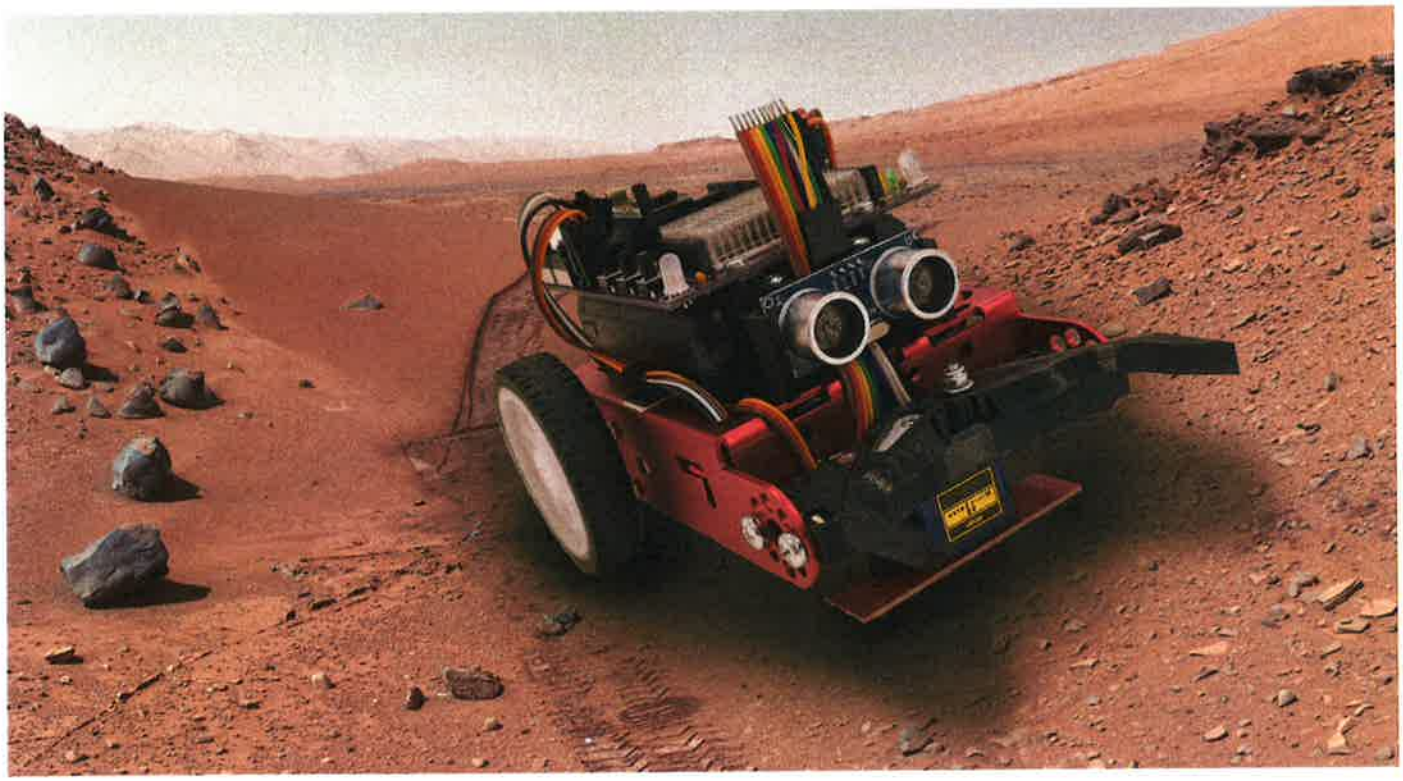
The T!CS RelayBot Guide



RelayBot

NHL STENDEN
university of
applied sciences

Preface

Congratulations on becoming the proud owner of a RelayBot! You are now part of a select group of individuals who have access to the latest in robotic technology and engineering. Your RelayBot is a formidable machine, designed to compete in the high-intensity sport of robot racing.

Handmade and 3D printed with durable materials and powerful motors, your RelayBot is ready to take on any opponent. Equipped with a variety of sensors, it has the ability to outmaneuver and outsmart its opponents. With your RelayBot, you will have the opportunity to participate in competitions, race against other robots, and showcase your engineering skills to the world.

As the owner of a RelayBot, you will have access to a community of like-minded individuals who share a passion for robotics and technology. You will be able to share tips, strategies, and experiences with other RelayBot enthusiasts. And, you will be able to take part in a community of builders, engineers, and competitors that is truly one of a kind.

We are proud to have you as part of the RelayBot community and look forward to seeing your robot on the RaceDay in action. Get ready to experience the thrill of robot racing and show the world what your RelayBot is capable of!

Table of contents

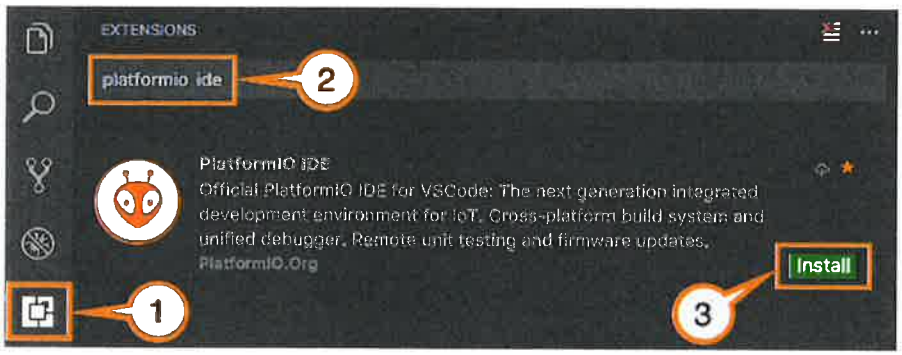
1. Quickstart

A good start is half the battle! This tutorial introduces you to the basics of PlatformIO IDE workflow and shows you a creation process of a simple "Blink" example. After finishing you will have a general understanding of how to work with projects in the IDE.

1.1 PlatformIO IDE for VSCode

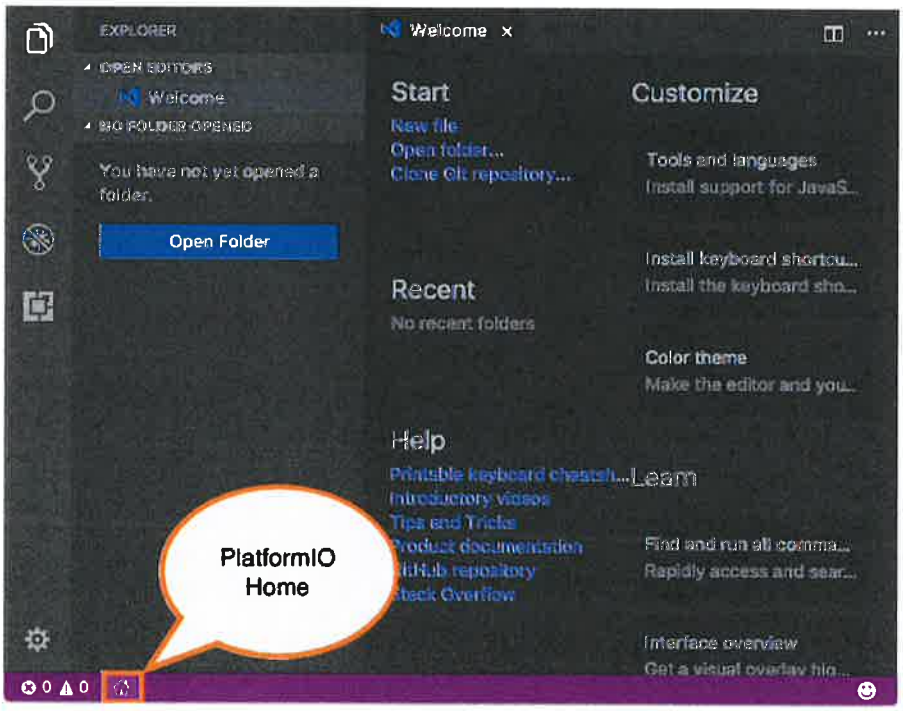
Visual Studio Code is a lightweight but powerful source code editor. It has a rich ecosystem of extensions for languages we will use, such as C++ and Python.

0. Download and install Visual Studio Code: code.visualstudio.com
1. Open VSCode Extension Manager
2. Search for official PlatformIO IDE extension
3. Install PlatformIO IDE.

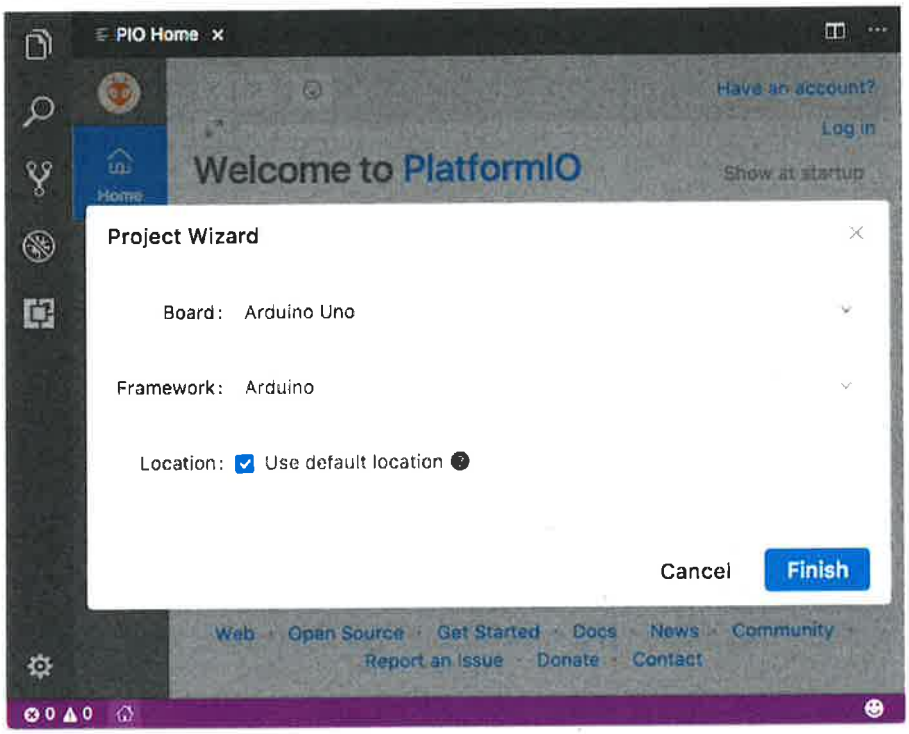


1.2 Setting up the project

1. Click on "PlatformIO Home" button on the bottom PlatformIO Toolbar

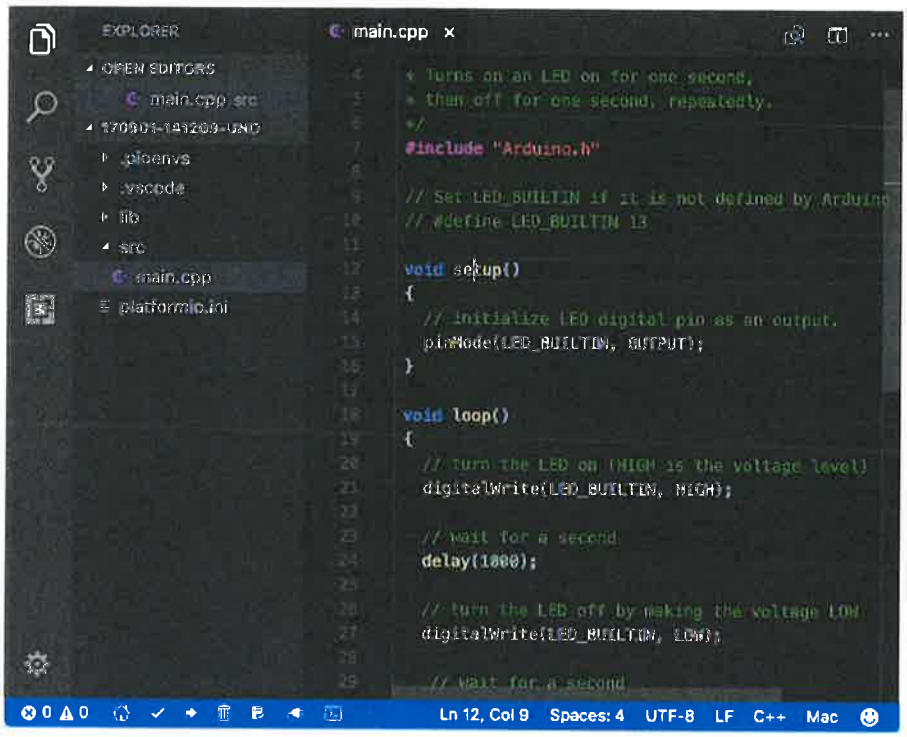


2. Click on "+ New Project", select a board and create a new PlatformIO Project. Choose Board: **Arduino Nano ATmega328** and click the "Finish" button.

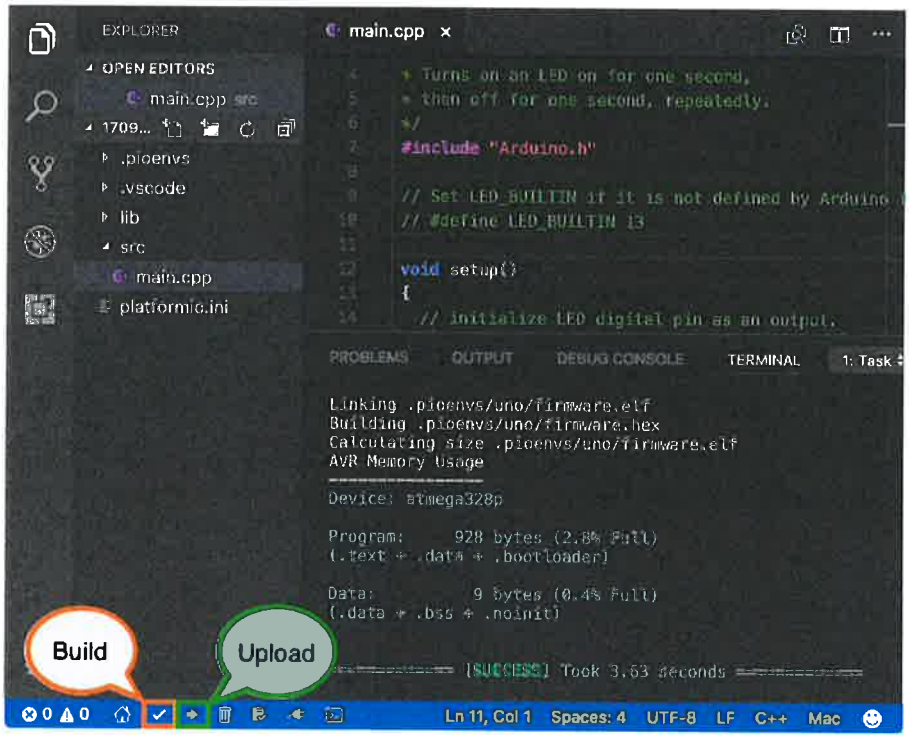


3. Open main.cpp file from src folder and replace its contents with the next:

```
#include "Arduino.h"

void setup(){
  // initialize LED digital pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop(){
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);

  // wait for a second
  delay(1000);

  // turn the LED off by making the voltage LOW
  digitalWrite(LED_BUILTIN, LOW);

  // wait for a second
  delay(1000);
}
```



4. Build your project using “Build” button on the PlatformIO toolbar



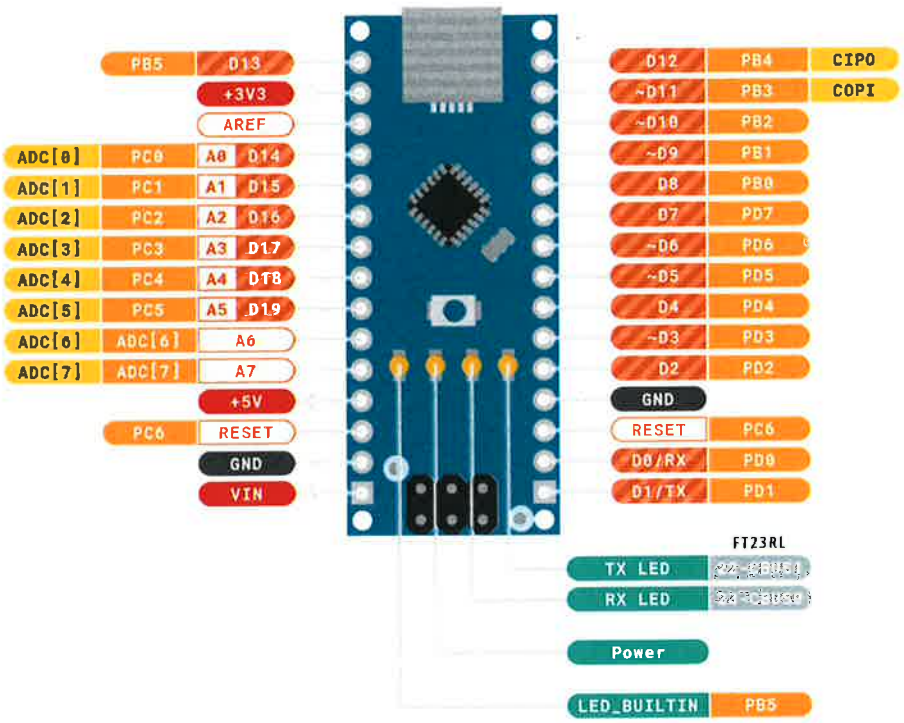
Congratulations, the RelayBot is ready to use!!!

2. Introduction to Arduino

The Arduino platform has since its start in 2005, grown to become one of the most recognizable brands in the space of electronics and embedded design. But what are the cornerstones of Arduino? What is a "board", how do I write code to it, and what are the tools needed to create my own project? The goal with this guide is to provide you with an overview of the Arduino project.

2.1 Arduino Nano

Over the years, Arduino has released hundreds of hardware designs in many shapes and forms. While all Arduino boards differ from each other, there are several key components that can be found on practically any Arduino. Let's take a look at the image of the Arduino Nano pinout below:



Most Arduino boards are designed to have a single program running on the microcontroller. This program can be designed to perform one single action, such as blinking an LED. It can also be designed to execute hundreds of actions in a cycle. The scope varies from one program to another.

The program that is loaded to the microcontroller will start execution as soon as it is powered. Every program has a function called "loop". Inside the loop function, you can for example:

- Read a sensor,
- Turn on a light,
- Check whether a condition is met,
- All of the above.

The speed of a program is incredibly fast, unless we tell it to slow down. It depends on the size of the program and how long it takes for the microcontroller to execute it, but it is generally in microseconds (one millionth of a second).

The specifications of an Arduino Nano can vary depending on the specific version, but here are the general specifications of a typical Arduino Nano:

- Microcontroller: ATmega328P
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 8
- DC Current per I/O Pin: 20mA
- Flash Memory: 32 KB (of which 2 KB used by bootloader)
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed: 16 MHz
- Size: 45mm x 18mm
- Weight: 7 g

It is also worth noting that Arduino Nano's can be powered either via USB or an external power supply. The external power supply can be connected through the Vin pin or directly to the power supply pins.

2.2 General functions

In this section, you will find a list of some of the most common elements in the standard Arduino API. This will help you get familiar with some key building blocks.

2.2.1 setup()

The **setup()** function is where you configure your program to get it ready to run, like setting up IO-pins and defining communication settings.

```
void setup() {
   //program configurations go here
}
```

2.2.2 loop()

The **loop()** function, a critical part of Arduino programming, serves as the heart of your code on the Arduino Nano. Here, you define the continuous operations your project will perform, such as reading sensors, processing data, and controlling outputs. It executes repeatedly for as long as your board is powered on, ensuring your project functions smoothly and continuously.

```
void loop() {
   //main program goes here
}
```

2.2.3 delay()

The **delay()** function pauses the program for a set number of milliseconds.
The classic blink sequence is found in the snippet below:

```
void loop() {
  digitalWrite(LED, HIGH); //turn on an LED
  delay(1000);             //as program is paused, with the LED on
  digitalWrite(LED, LOW);  //program is unpaused, the LED is turned off
  delay(1000);             //program is paused, with the LED off
}
```

The **delay()** function is an incredibly useful function, and you will find it in almost all examples. But, for efficiency of the code, it is **not** the best option, as it prevents the Arduino from doing anything for the duration of the delay.

2.2.4 millis()

The **millis()** function is a bit more advanced, but an incredibly resourceful function. It allows you to have multiple events happening simultaneously, without pausing the program. This is done by measuring time (in milliseconds) passed since the program started. Then, with the use of intervals and continuously storing the time for the last event, a simple algorithm can be made to have events happening at specific times without pausing the program.

See the example below:

```
const int ledPin = 13;          // internal led on pin 13
unsigned long timerOne = 0;     // store the next time
bool state;                     // status of the led (on/off)

void setup() {
  pinMode(ledPin, OUTPUT);      // Set pin 13 as OUTPUT
}

void loop() {
      blink(500);               // blink every 500 ms
}
//==== [ Blink Function ] ========
void blink(int INTERVAL) {
  if (millis() >= timerOne) {               // time reached?
      timerOne = millis() + INTERVAL;       // set new time

      state = !state;           // change the current led state
      digitalWrite(ledPin, state);
  }
}
```

While the **millis()** function is a more advanced concept than the **delay()** function, it is good to start practicing it early on.

2.2.5 Custom functions

You can create **custom functions** that either just executes code and returns to the program, or that returns a result.

Example of a void function that does not return:

```
int x;

void setup() {
   //program configurations go here
}

void loop() {
   thisFunction(); //execute the function
}

void thisFunction() {
   x++; //increase x by 1 each time the function is run.
}
```

Example of a type int function that returns a value.

```
int value;

void setup() {
   //program configurations go here
}

void loop() {
   value = returnFunction();
}
int returnFunction() {
   int returnValue = 5 + 2;
   return returnValue;
}
```

2.3 Variable Definition

Variables can either be created **locally** or **globally**. Variables that are defined in the **loop()** are considered local, and variables defined at the top of your sketch are considered global.

```
int sensorReading = x; //global variable

void setup(){
   //program configurations go here
}

void loop(){
   int sensorReading = x; //local variable
}
```

2.4 Data Types

There are several data types available for use, and below are some of the most common:

```
1 bool    //holds one of two values, true or false
2 byte    //range:                0 to           255
3 char    //example: "A"
4 double  //range:    1.18 x 10^-38 to  3,40 x 10^38
5 float   //example: 3.14 or -0.005 or 123.456
6 int     //range:          -32,768 to        32,767
7 long    //range:   -2,147,483,648 to 2,147,483,647
8 short   //range:          -32,768 to        32,767
9 String  //example: "This is a string!"
```

*To store data in for example an **int** (integer):*

```
int exampleNumber = 25;
```

*For numbers with a lot of decimals, we can use **float**:*

```
float exampleNumber = 22.2123002;
```

*Or to store a string, we can use the **String** function:*

```
String exampleSentence = "This is a string!";
```

*For simple switches and true/false, we use **boo**leans:*

```
bool exampleSwitch = true; // true/false
```

2.5 Serial Communication

Serial communication is essential to Arduino programming, as it is the easiest way to know what goes on on your board. For this, we can use the Serial class.

2.5.1 Serial.begin()

Initialize serial communication between board & computer. This is defined in the void setup() function, where you also specify baud rate (speed of communication).

```
void setup() {
   Serial.begin(9600);
}
```

2.5.2. Serial.print()

Print data to the serial port, which can be viewed in the Arduino IDE Serial Monitor tool.

```
void loop() {
   Serial.print();
}
```

2.5.3. Serial.read()

Read the incoming serial data.

```
void loop() {
   int incomingByte = Serial.read();
}
```

2.6 Pin Management

Configuring, controlling and reading the state of a digital/analog pin on an Arduino.

2.6.1 pinMode()

Configures a digital pin to behave as an input or output. Is configured inside the void setup() function.

```
void setup() {
   pinMode(pin, INPUT);        //configures pin as an input
   pinMode(pin, OUTPUT);       //configures pin as an output
   pinMode(pin, INPUT_PULLUP); //enables the internal pull-up resistor
}
```

2.6.2 digitalRead()

Reads the state of a digital pin. Used to for example detect a button click.

```
int state = digitalRead(pin); //store the state in the "state" variable
```

2.6.3 digitalWrite()

Writes a high or low state to a digital pin. Used to switch on or off a component.

```
digitalWrite(pin, HIGH); //writes a high (1) state to a pin
digitalWrite(pin, LOW);  //writes a low (0) state to a pin
```

2.6.4 analogRead()

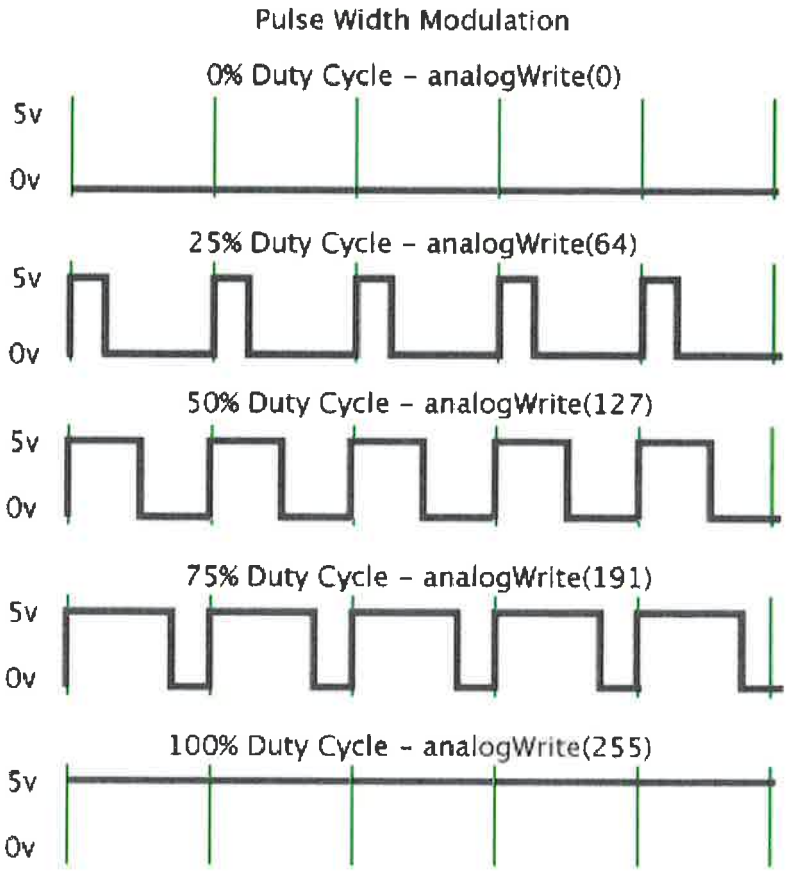
Reads the voltage of an analog pin, and returns a value between 0-1023 (10-bit resolution). Used to read analog components.

```
sensorValue = analogRead(A1); //stores reading of A1 in "sensorValue"
variable
```

2.6.5 analogWrite()

Writes a value between 0-255 (8-bit resolution). Used for dimming lights or setting the speed of a motor. Also referred to as PWM, or Pulse Width Modulation. PWM is only available on specific pins, marked with a "~" symbol.

```
analogWrite(pin, value); //write a range between 0-255 to a specific pin
```



2.7 Structure

The structure of the Arduino API is based on C++, and can be considered the building blocks of a program.

2.7.1 Conditionals

Conditionals are some of the most popular used elements in any program. In Arduino, a typical conditional consists of an **if** and **else** statement. You can make use of several **if/else** statements in your code.

```
if(variable == true){
   //do something
}
else {
   //do something else
}
```

2.7.2 Loops / Iterations

The **for** and **while loops** are commonly used in programs, to execute a block of code for a set number of times, or while a condition is met.

A basic use of a **while loop** to execute a block of code while variable is true.

```
while (variable == true) {
   //do something
}
```

A basic use of a **for loop** is to execute a block of code a custom number of times (in this case, 10).

```
for (int x = 0; x < 10; x++) {
     //do something 10 times
}
```

To break out of a loop, you can use break. In the snippet below, if a condition is met (variable is true), we break out of the loop.

```
for (int x = 0; x <= 10; x++) {
     if(variable == true) {
         break;
     }
 }
```

2.8 Operators

The Arduino programming language supports the following operators:

1. Arithmetic Operators: +, -, *, /, %
2. Comparison Operators: ==, !=, >, <, >=, <=
3. Boolean Operators: && (and), || (or), ! (not)
4. Compound Operators: =, +=, -=, *=, /=, %=
5. Bitwise Operators: & (and), | (or), ^ (xor), ~ (not), << (left shift), >> (right shift)

These operators are used to perform various operations on variables and values in the code, such as performing arithmetic calculations, comparing values, making decisions based on conditions, etc.

2.8.1 Arithmetic Operators

Arithmetic operators are used for addition, subtraction, multiplication, division and other mathematical calculations.

```
int x = 5;
int y = 2;

x + y; //result is 7
x * y; //result is 10
x - y; //result is 3
```

2.8.2 Comparison Operators

Comparison operators are used for comparing one property to another, and are a key component of a conditional statement.

There are several comparison operators:

```
1 != //not equal to
2 <  //less than
3 <= //less than or equal to
4 == //equal to
5 >  //greater than
6 >= //greater than or equal to
```

To use them in a conditional, see the following example:

```
if(value > 10) {
   //do something
}
```

2.8.3 Boolean Operators

Boolean operators (logical NOT, AND and OR) can for example be used for more advanced conditionals.

*To use the **&&** (and) operator:*

```
if(value > 10 && otherValue > 10) {
   //do something if only if *both* conditions are met
}
```

To use the || (or) operator:

```
if(value > 10 || otherValue > 10) {
   //do something if a one *or* the other condition is met
}
```

To use the ! (not) operator:

```
if(!value) {
   //do something if the value is false (!)
}
```

2.8.4 Compound Operators

Compound operators consist of two operators, which are used to perform two operations in the same statement. This can for example be to add + and assign = a value at the same time.

Here are some examples:

```
x = 5;
y = 2;

x++; //increase by one, so x is now 6
x--; //decrease by one, so x is now 4

x += y; //x is now 7 (add and assign)
x -= y; //x is now 3 (subtract and assign)
x *= y; //x is now 10 (multiply and assign)
```

These are only a few examples, for more language reference about functions, values (variables and constants), and structure, see: www.arduino.cc/reference.

2.8.5 Structure example

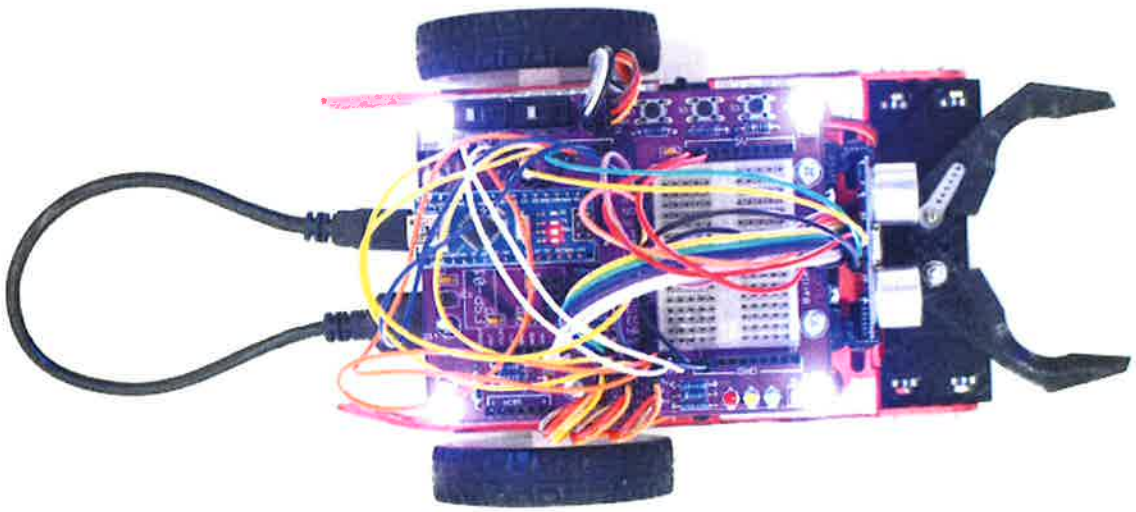
```
/************************
 *** Structure example ***       <- This is a block comment!!!
 *** 03-05-2024 - V1.1 ***
 ************************/
// ====[ LIBRARIES ]================================
#include <Arduino.h>

// ====[ PIN NUMBERS ]==============================
const int buttonPin = 3;       // Push Button
const int ledPin    = 11;      // Red Led

// ====[ VARIABLES ]================================
  // Define the variables: List all the variables that you'll
  // be using in your program, including their data types.
bool ledValue = true;          // Turn the led on or off

// ====[ SETUP ]====================================
void setup() {
// =====[ INPUT/OUTPUT ]============================
  pinMode(ledPin, OUTPUT);     // D11 -> Red Led
  pinMode(buttonPin, INPUT);   // D3  <- Push Button
  // In this section, you should initialize all the required
  // hardware, such as defining the pin modes, setting up
  // serial communication, etc.
}
// ====[ LOOP ]=====================================
void loop() {
  // The main loop should contain the logic for your
  // program. This is where you'll be using the variables
  // and control structures such as if-else statements
  // do-while loops and for loops.

  if (condition == true){      // if button is pressed
    showLed();                 // then show the led
  }
  // the rest of the program code goes here...
}
// ====[ FUNCTIONS ]================================
  // When writing functions, use clear and descriptive
  // names that describe the functionality of the function.
void showLed() {
  // Add comments to explain what each function is doing.
}
```

3. RelayBot Hardware

This section gives you detailed information about the hardware that is used for the RelayBot. The RelayBot contains the following hardware:

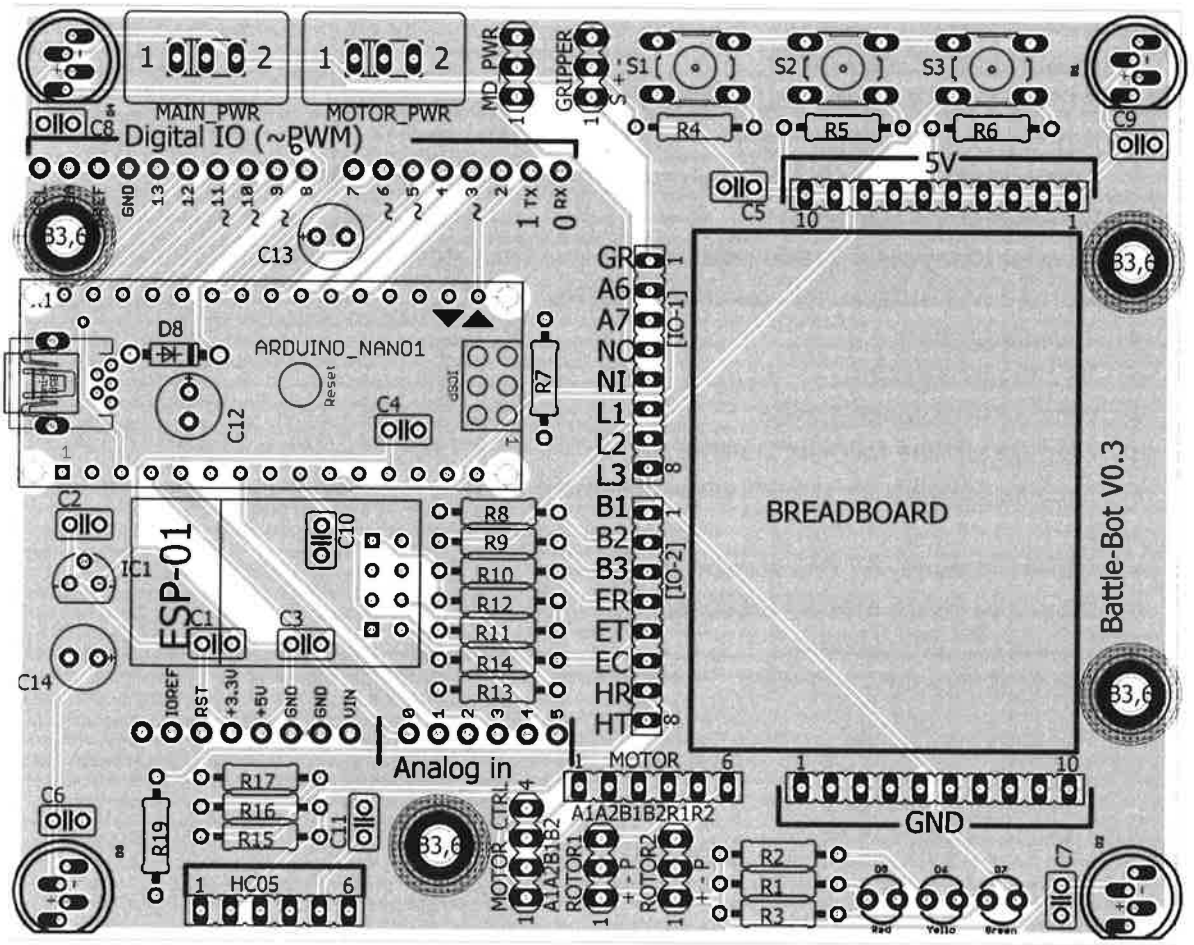
- Arduino NANO microcontroller
- PCB with an Arduino UNO footprint for additional shields
- Metal chassis
- Powerbank (10.000 mAh)
- 2 On/Off switches, for the logic circuit and the motor circuit
- A gripper with a servo motor
- 4 Neopixels (tricolor leds) for signaling (head-, blink- or brake-lights)
- 3 Pushbuttons (optional)
- 3 Leds (red, yellow, green optional).
- Motor driver suitable for two electromotors
- 2 Electromotors
- 2 Rotation sensors
- Ultrasonic distance sensor
- 8 Analogue line follower sensors
- Mini breadboard for experimental purposes

- ESP01 module, AT Wifi Modem (optional)
- Bluetooth HC05 module (optional)
- Gyroscope I2C (optional)
- Oled Display I2C (optional)



3.1 Printed Circuit Board (PCB)

The Printed Circuit Board (PCB) interconnects the electronic components using conductive tracks, pads and other features etched onto one or more thin sheets of copper laminated onto a non-conductive substrate. The PCB is mounted on top of the RelayBot and has many headers that can be used for connecting components to the Arduino Nano.

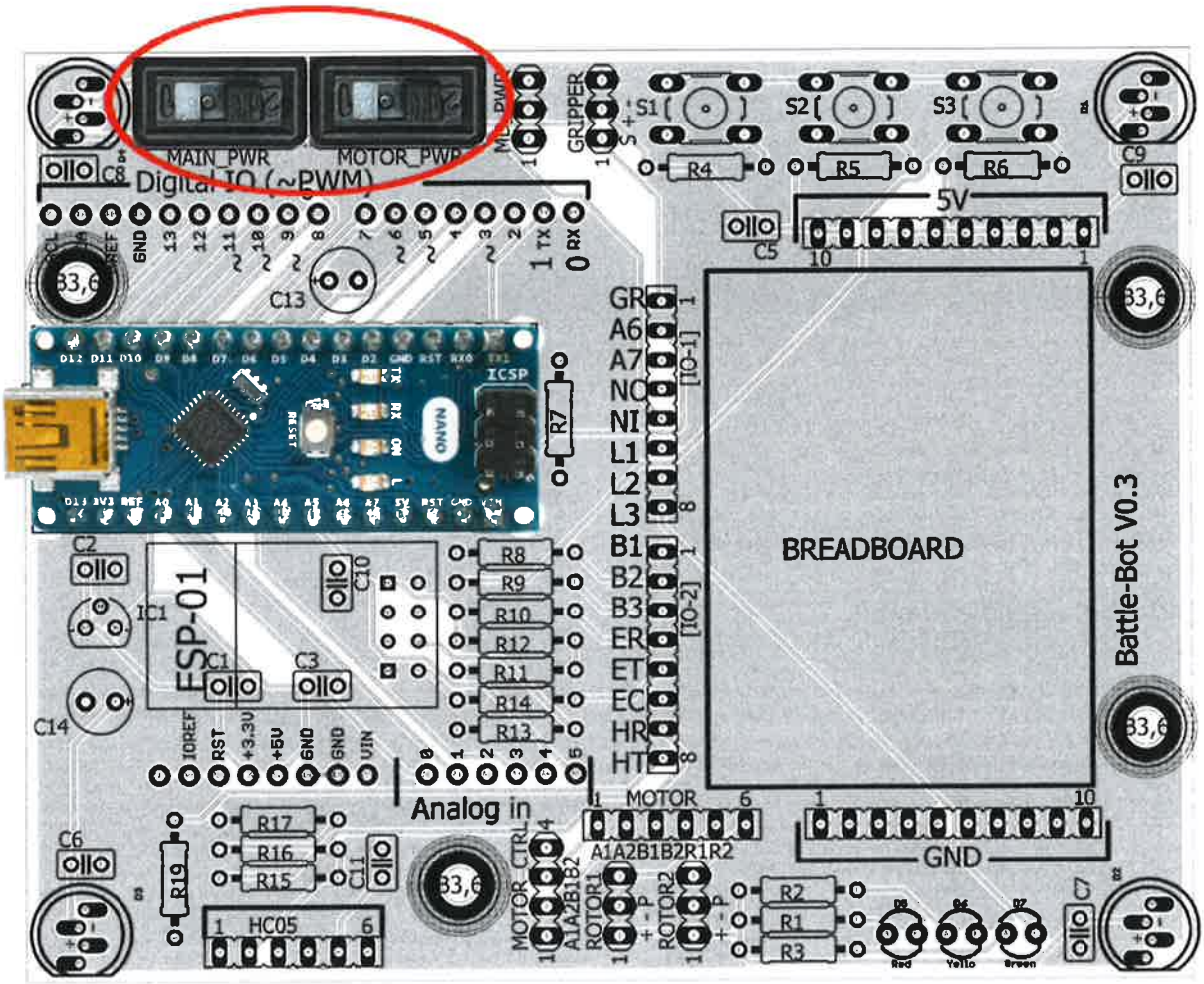


The main functions of PCB in electronic components are:

- provide mechanical support and guidance for the components
- act as a bridge between the component leads and the external connector pins
- carry current from one component to another
- protect the components from short circuits and electrical interference

3.2 Power switches

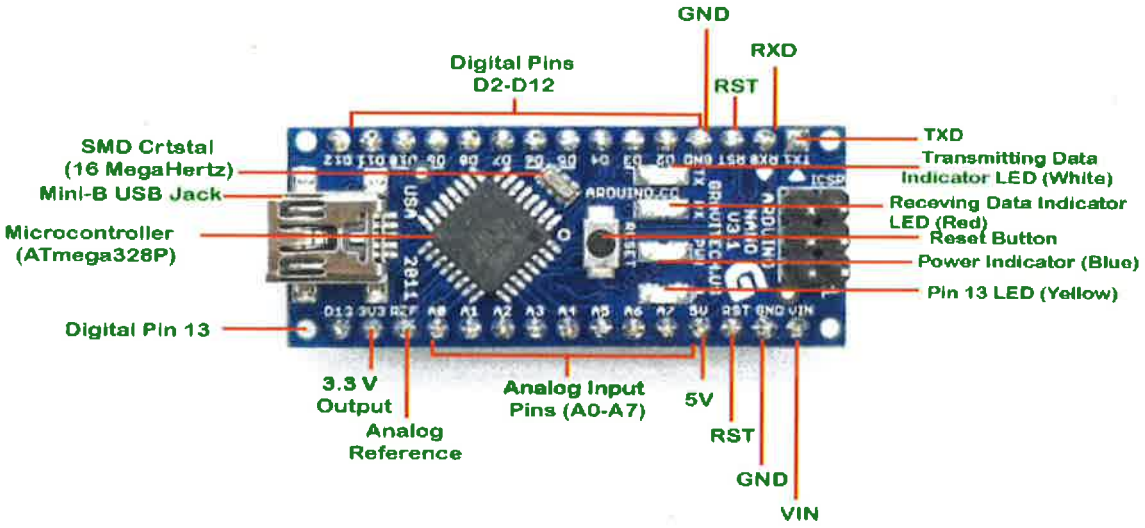
On the PCB there are 2 power switches these are to turn the Robot on and off, with the MAIN_PWR you turn the electronics of the robot on and off and with the MOTOR_PWR you turn the motors on and off, the latter is especially useful to prevent your robot from driving unexpectedly and falling off the table for example.



These switches only work with the lower USB connection and not with the USB connection of the Arduino Nano. The Arduino Nano will always power the board but never the motors.

3.3 Arduino NANO (microcontroller)

The square black 'stamp' on the Arduino Nano is the actual microcontroller (ATmega328) the rest of the components are to make it easily usable and programmable.



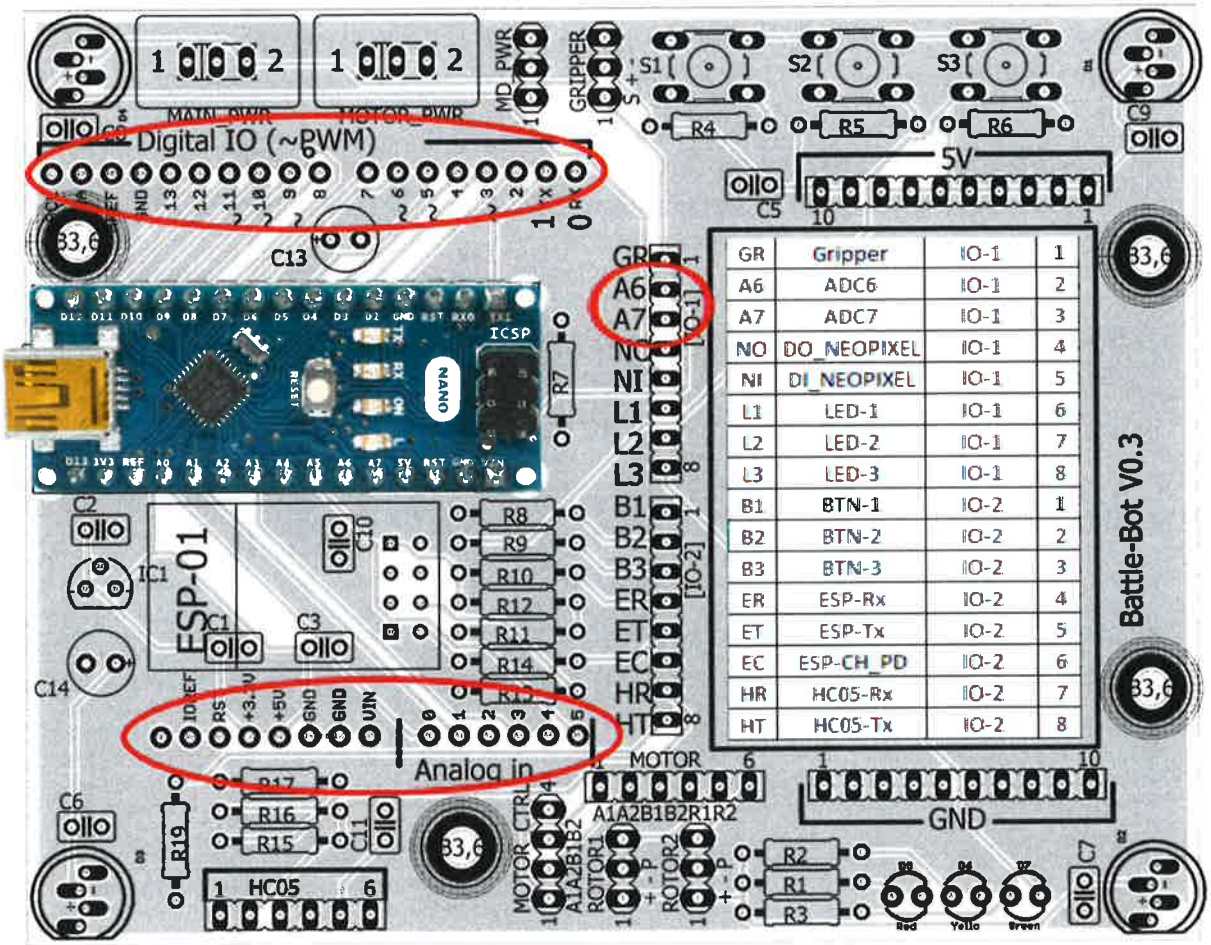
** Colors of the LEDs may differ by vendor

Technical specifications

Microcontroller	ATmega328
Architecture	8-bit AVR
Interfaces	Mini USB, ISP, I²C, SPI™, Serial, ICSP
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz
Analog IN Pins	8
EEPROM	1 KB
DC Current per I/O Pin	20 mA
Input Voltage	7-12 V
Digital I/O Pins	22 (6 of which are PWM)
PWM Output	6
Power Consumption	19 mA

3.4 I/O connectors

These connections are to connect components to the Arduino NANO, they are connected directly to the processor and have the same layout as on an Arduino UNO.

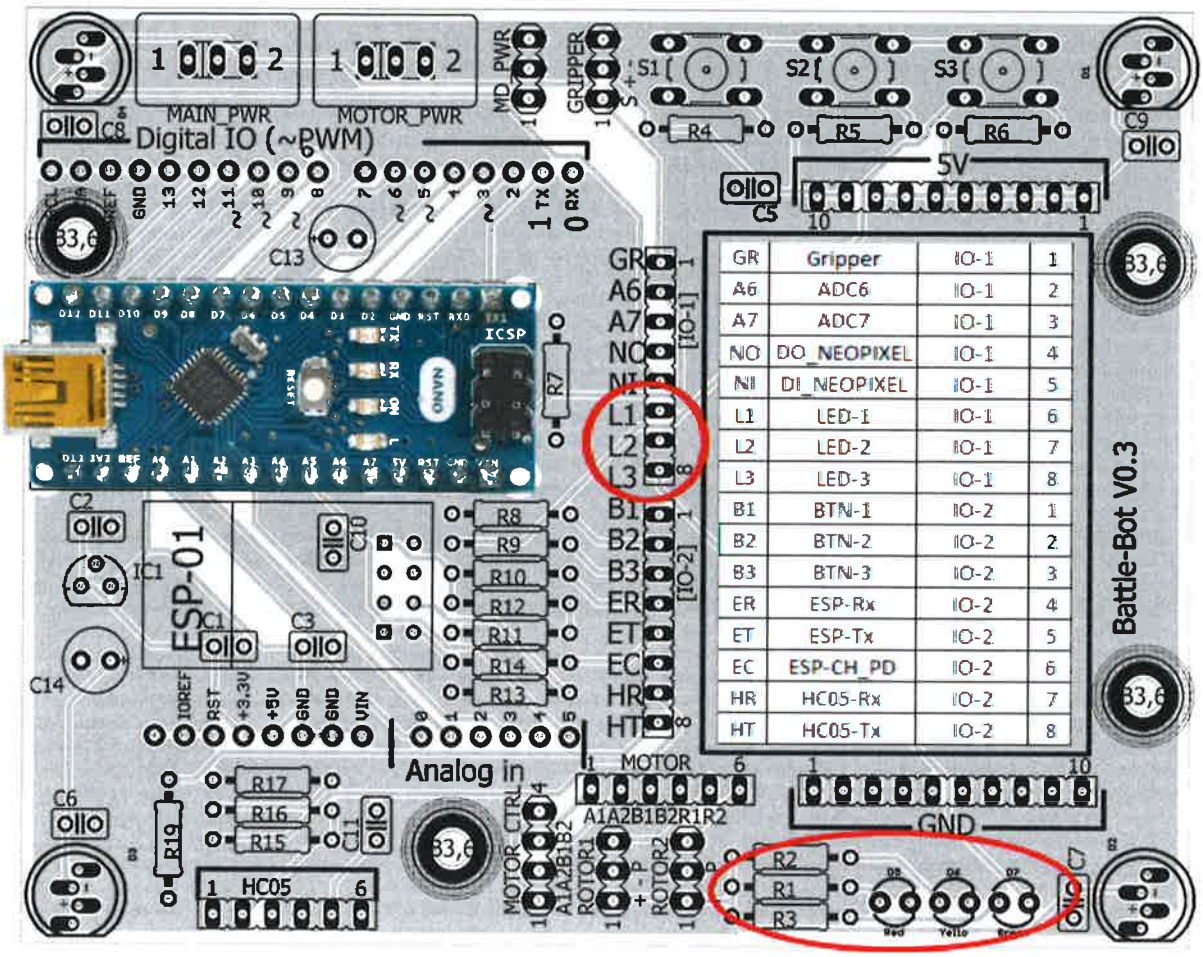


GR	Gripper	IO-1	1
A6	ADC6	IO-1	2
A7	ADC7	IO-1	3
NO	DO_NEOPIXEL	IO-1	4
NI	DI_NEOPIXEL	IO-1	5
L1	LED-1	IO-1	6
L2	LED-2	IO-1	7
L3	LED-3	IO-1	8
B1	BTN-1	IO-2	1
B2	BTN-2	IO-2	2
B3	BTN-3	IO-2	3
ER	ESP-Rx	IO-2	4
ET	ESP-Tx	IO-2	5
EC	ESP-CH_PD	IO-2	6
HR	HC05-Rx	IO-2	7
HT	HC05-Tx	IO-2	8

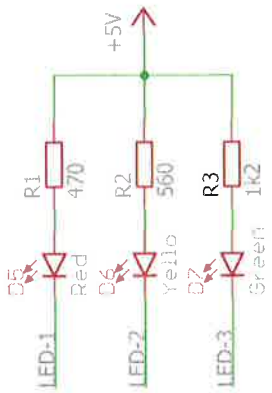
3.5 Light Emitting Diode (LED)

A light-emitting diode (LED) is a semiconductor device that emits infrared or visible light when current flows through it. Visible LEDs are used in many electronic devices as indicator lamps, in automobiles as rear- and brake lights, and on billboards and signs as alphanumeric displays or even full-color posters. Infrared LEDs are employed in autofocus cameras and television remote controls and also as light sources in fiber-optic telecommunication systems.

The red, yellow and green LED on the RelayBot are switched low active, this means they turn on with a LOW ("0") output. This is opposite of the built-in LED (on D13) which is HIGH active.



The symbol of an LED is a kind of arrow, this is because an LED is in conduction only 1 way, so flip plus and minus then the LED blocks the current and will not light up. Now each LED has a certain forward current (If = I forward) at a certain voltage (Uf = U forward), this is different for each type of LED, the resistors *R1, R2* and *R3* ensure that the current is limited (series resistor).



L1 (LED-1) = LED red
L2 (LED-2) = LED yellow
L3 (LED-3) = LED green

For the red LED applies: *Uf* = 2.5V and *If* = 5mA (0.005A) so over the resistor must now fall 5V - 2.2V = 2.8V, the current through the resistor is equal to the current through the LED (series circuit) so also 5mA, with Ohm's law (*U = I x R)* we can now calculate the resistance needed:

R = U / I
R = 2.8 / 0.005
R = 500 Ohm

Since this resistor value does not exist we choose the value closest to the one that does exist in the E12 series and that is 470 Ohm, this is not a problem because in practice an *If* of +/- 2 mA is workable.

You may also check the yellow and green LED to see if this is correct:

Yellow LED: *Uf* = 2.1 and *If* = 5mA
Green LED: *Uf* = 2.2V and If = 2mA

Resistors in the E12 range: 1.0 - 1.2 - 1.5 - 1.8 - 2.2 - 2.7 - 3.3 - 3.9 - 4.7 - 5.6 - 6.8 - 8.2 (and multiples 1x, 10x, 100x, 1000x, ... etc.)

3.5.1 Example 1 - Blink

Write the following program and upload it to the Arduino Nano. Connect digital pin 13 to the red LED using a jumper wire. Test the program and adjust the speed of the flashing LED.

```
/*****************************
 ***    Example 1 - Blink    ***
 *****************************/
int ledRed = 13;

void setup() {
  pinMode(ledRed, OUTPUT);
}

void loop() {
  digitalWrite(ledRed, LOW);
  delay(500); // Wait for 500 milliseconds
  digitalWrite(ledRed, HIGH);
  delay(500); // Wait for 500 milliseconds
}
```

Save the program after testing.

3.5.2 Exercise 1 - Traffic light

Adjust the blink program to build a traffic light. Traffic light should respond in this fashion:

- red light = 3 seconds
- green light = 4 seconds
- orange/yellow light = 1 second
- and repeat...

Save this program, you will need this later.

3.5.3 Example 2 - Fade in

Write the following program and connect digital pin ~11 to the red LED using a jumper wire. For more information see: 2.6.5 **analogWrite()**

```
/*****************************
 ***   Example 2 - Fade in   ***
 *****************************/
int ledRed = 11;
int brightness = 0;

void setup() {
  pinMode(ledRed, OUTPUT);
}

void loop() {
  for (brightness = 0; brightness <= 255; brightness += 5) {
    analogWrite(ledRed, brightness);
    delay(30); // Wait for 30 milliseconds
  }
}
```

Test the program and adjust the speed.

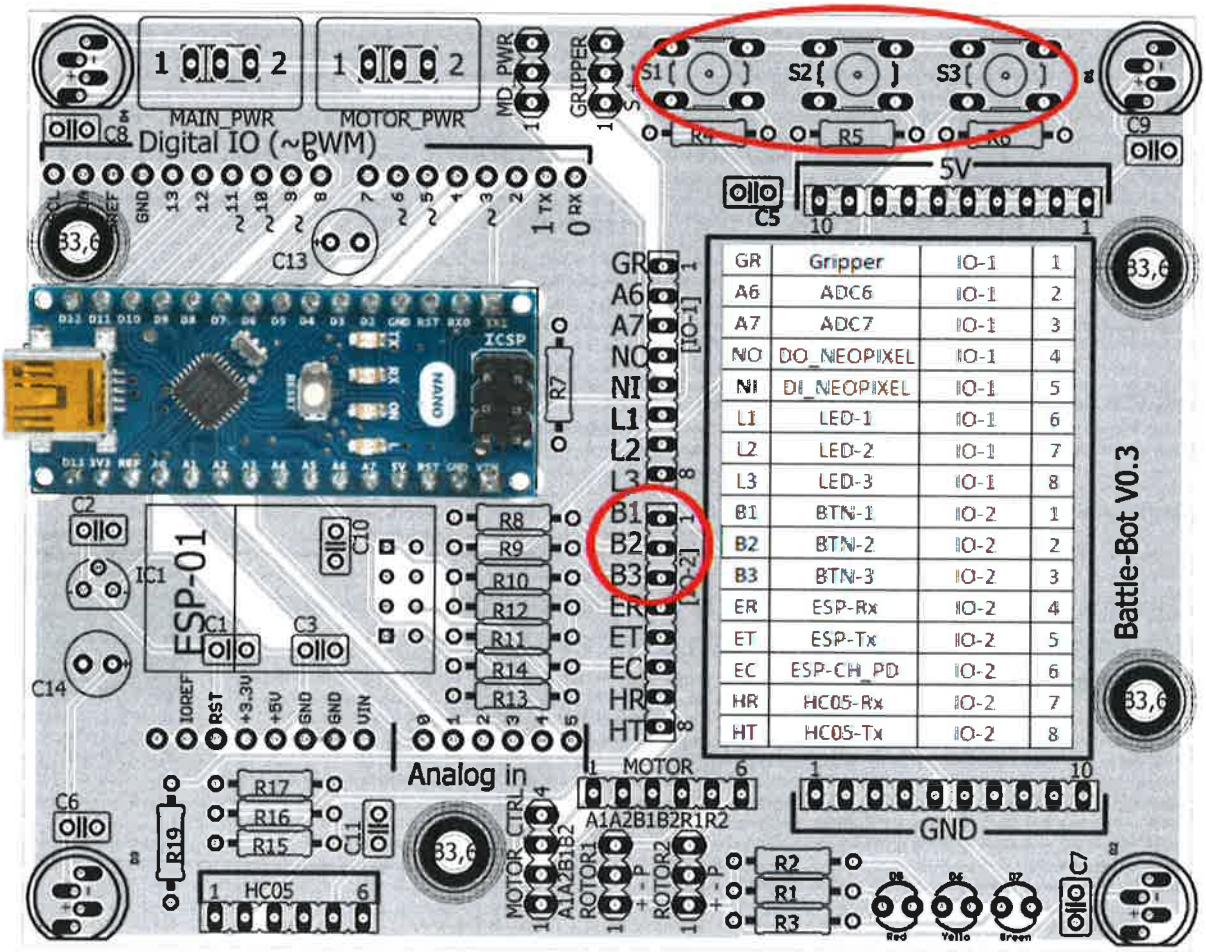
3.5.2 Exercise 2 - Fade in/out

Adjust the program so the LED will fade in and out and save the program after testing. Tip: the **analogWrite()** function is very useful for adjusting the RelayBot's motor speed.

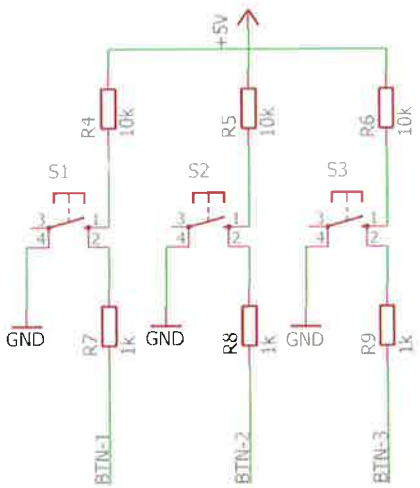
3.6 Switches (Buttons)

These switches are active low, this means that when pressed the input becomes LOW ("0"). The circuit with the resistors *R4, R5, R6* ensures that the the signal is HIGH when the switch is not pressed, it is pulled up to the 5V ("1") hence the name pull-up resistors, without these resistors the input would start to "float" when the switch is not pressed and it becomes unpredictable what the input value is.

When you push the switch the input will become "0" because it is then connected to ground (ground, 0V "0"), without the pull-up resistors a short circuit would now occur. For the same reason the resistors *R7, R8, R9* are added, these prevent that if the switches are connected incorrectly (directly to the 5V e.g.) a short circuit will occur when pressing.



Since the current through the input of the processor is enormously low (negligible), almost no current will flow through the resistors during normal operation, nor will the voltage drop at the input. $U = I \times R$ where I = 0A so U = 0V regardless of the resistor.



B1 (BTN-1) = Switch 1
B2 (BTN-2) = Switch 2
B3 (BTN-3) = Switch 3

3.6.1 Example 3 - Button

Open the example-sketch 02.Digital > Button as shown. Examine the code and connect button S1 to the buttonPin and the ledPin to the yellow led. Change the if-statement, because our buttons and leds are LOW-active.

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin
// variables will change:
int buttonState = 0;         // variable for reading the pushbutton
status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  if (buttonState == LOW) {
    digitalWrite(ledPin, LOW);  // turn LED on:
  } else {
    digitalWrite(ledPin, HIGH); // turn LED off:
  }
}
```

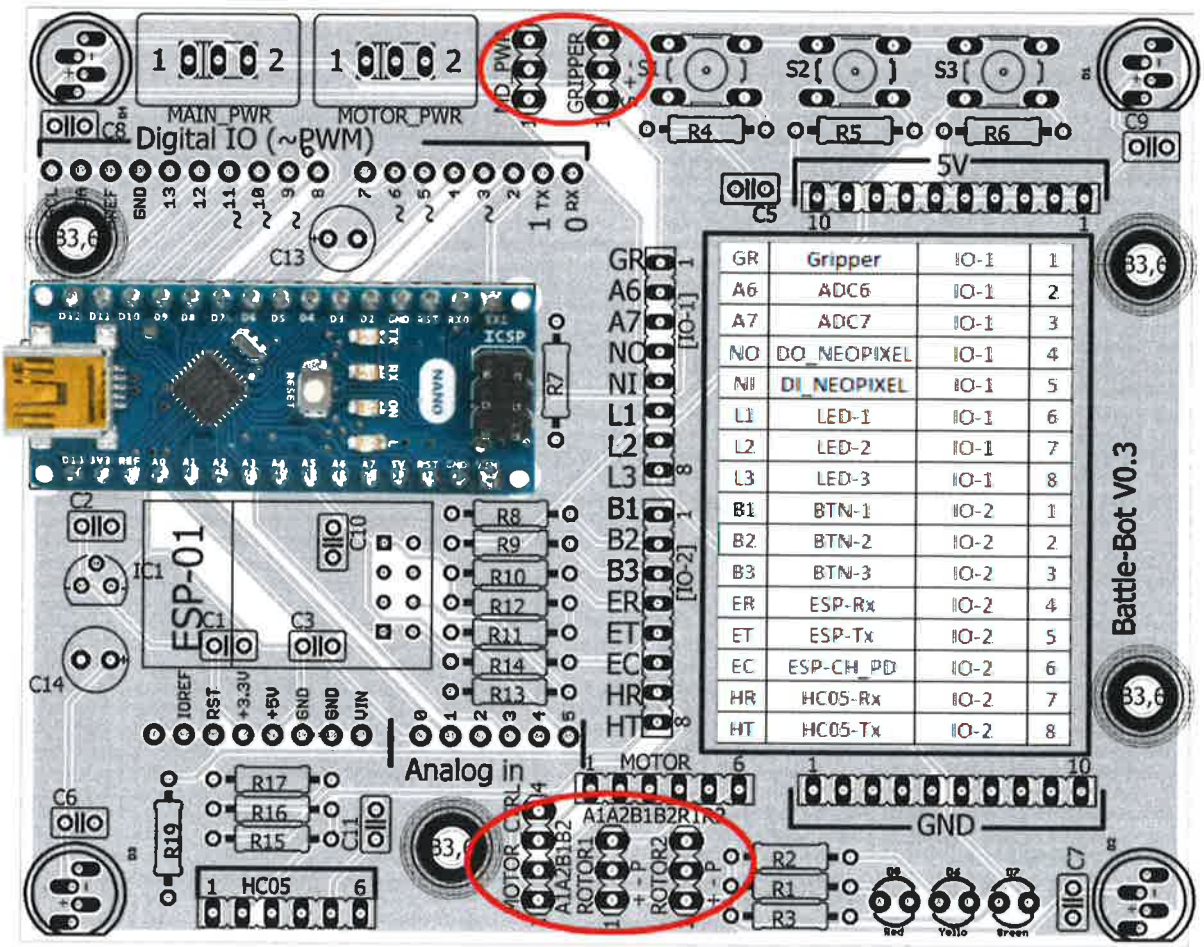
3.6.2 Exercise 3 - Traffic Light with button

Adjust this program so the traffic light (exercise 1) will activate when the button is pushed.

3.6.3 Exercise 4 - Blink with button

Write a program that combines a red led flashing every second (like example 1 - Blink) The green led goes off when at least one of the three buttons is pushed and goes on again when all buttons are released. This is a tricky one!!! For more information read section 2.2.3 delay() and 2.2.4 millis().

3.7 Robot Frame connections

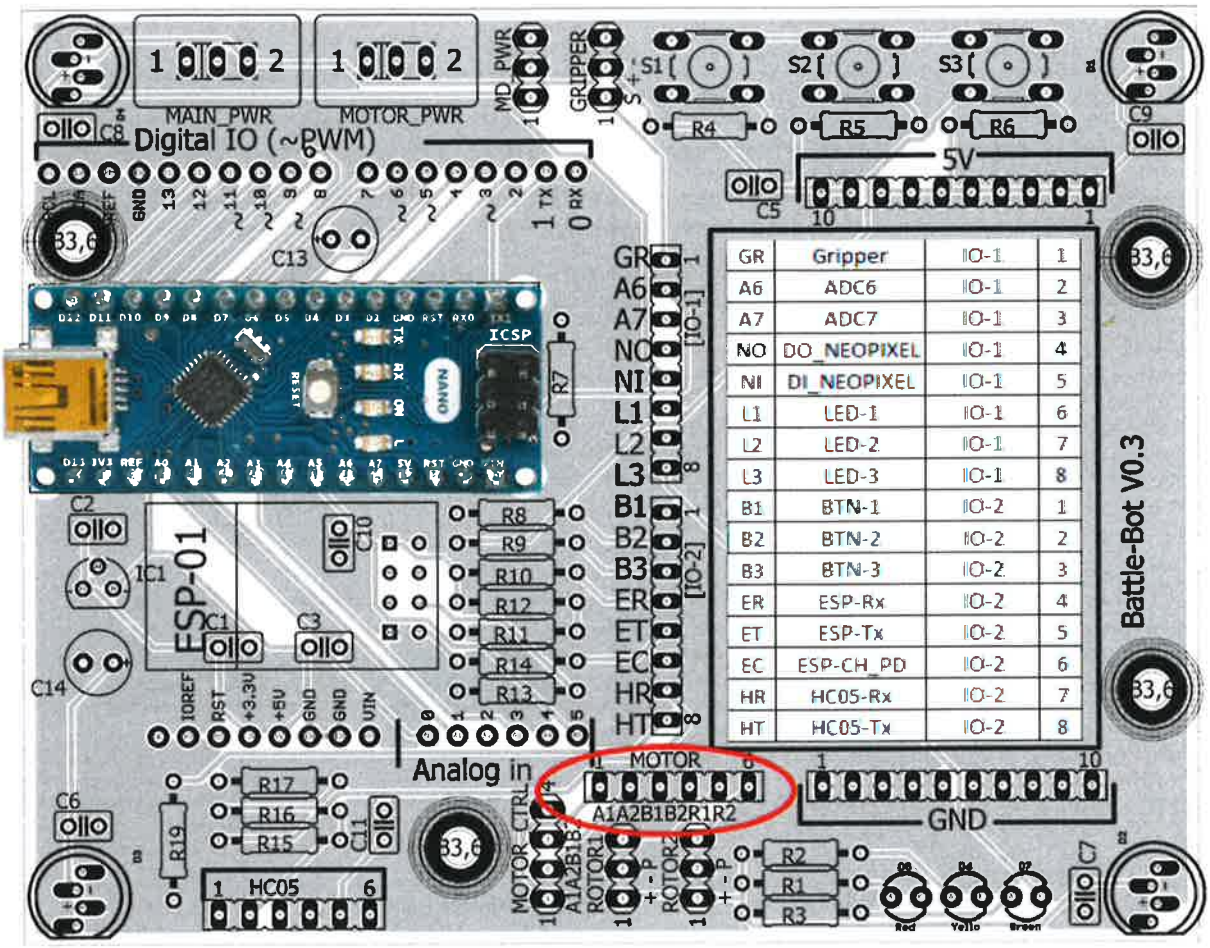


These connections connect the PCB to the motor driver, rotation pulsers and the servo of the front gripper. You don't have to do anything with these, and if connected incorrectly, the electronics may even fail.

- MOTOR_CTRL : motor control
- MD_PWR : motor driver power supply
- ROTOR1 and ROTOR2 : Connection of the rotation sensor.

3.8 Motor control

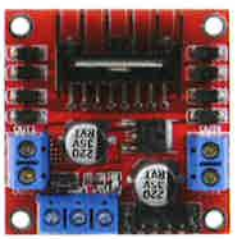
These connections are used for controlling the speed of the motors and reading out the wheel pulses.



- A1 and A2 are for motor A,
- B1 and B2 are for motor B,
- R1 and R2 are the rotation sensors and give 20 pulses per rotation, per motor.

It is up to you to figure out which motor controls which wheel and in which direction and which pulse is on which wheel, REMEMBER not all RelayBots are wired the same.

The motor driver is an extension board that is mounted underneath the RelayBot and it powers the two DC motors. The board allows control with a constant voltage between 5V and 35V.

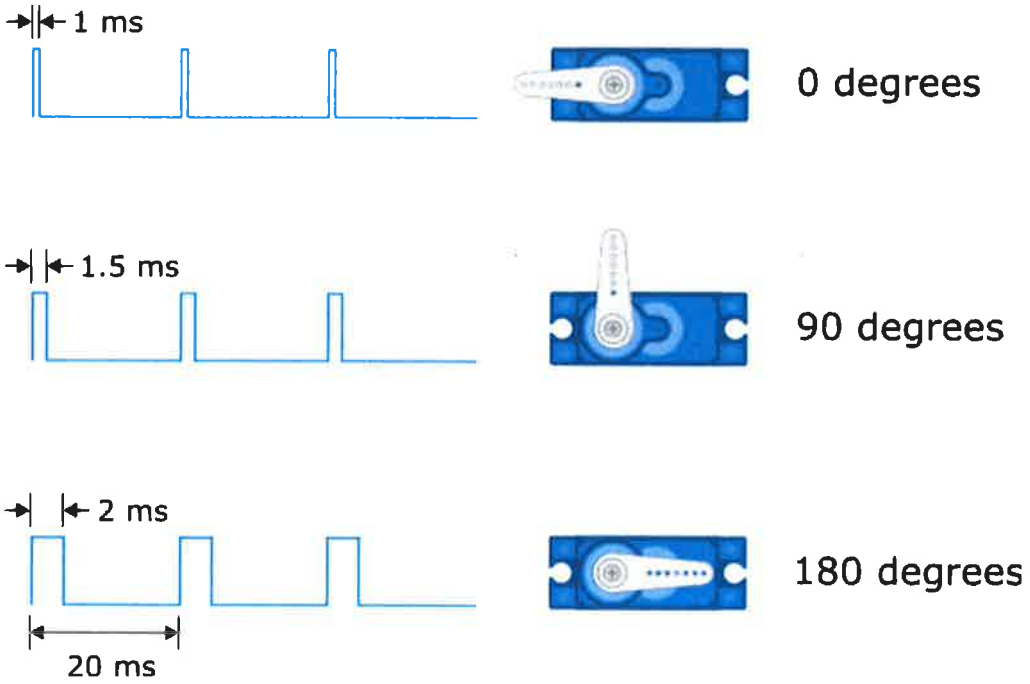


Make use of PWM signals to control the speed of the motors.

3.9 Gripper

The gripper on the front of the RelayBot is using a servo motor and is connected to the PCB. A servo motor is a type of motor that is commonly used in robotics and control systems for precise positioning and control. It consists of a motor, a gear train, and a feedback mechanism, which allows it to rotate to a specific angular position and hold that position with high accuracy.

Servo motors are controlled by sending a pulse to the signal line of the servo. The width of the pulses determines the position of the output shaft. When you send the servo a signal with a pulse width of 1.5 milliseconds (ms), the servo will move to the neutral position (90 degrees). The min (0 degrees) and max (180 degrees) position typically correspond to a pulse width of 1 ms and 2 ms respectively. Note this can vary slightly between different types and brands of servo motors (e.g. 0.5 and 2.5 ms). Many servos only rotate through about 170 degrees (or even only 90) but the middle position is almost always at 1.5 ms.



Servo motors generally expect a pulse every 20 milliseconds or 50 Hz but many RC servos work fine in a range of 40 to 200 Hz.

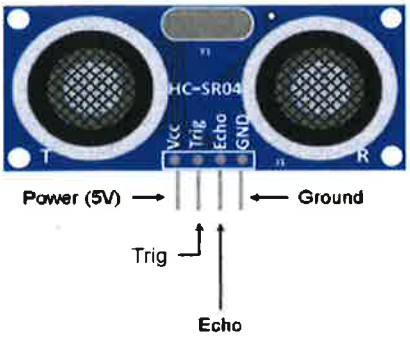
Servo motors have three wires: power, ground, and signal.

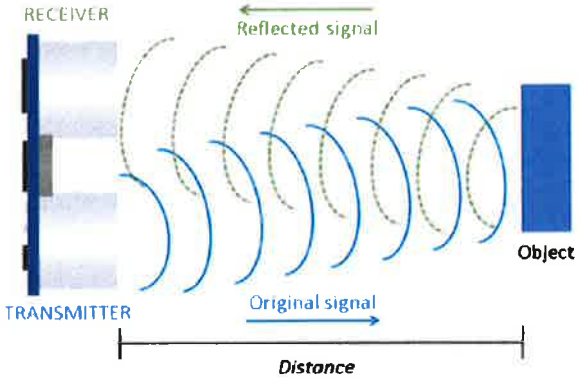
- The power wire is typically red, and should be connected to **+5V**.
- The ground wire is typically black or brown and should be connected to **ground**.
- The signal wire is typically yellow or orange and should be connected to a **digital pin** of the Arduino.

Our servo motor can rotate approximately 180 degrees (90 in each direction). Example sketches using a servo motor are available within the Arduino IDE. If you make use of a servo motor library, then you should read the available documentation carefully!

3.10 Ultrasonic distance sensor

HC-SR04 is a type of ultrasonic sensor that is commonly used for distance measurement. The sensor uses high-frequency sound waves to determine the distance to an object.





It sends out an ultrasonic pulse and measures the time it takes for the echo to return. Based on this time measurement, the distance to the object can be calculated. It is commonly used in projects such as obstacle avoidance robots, distance measuring devices, and level sensors.

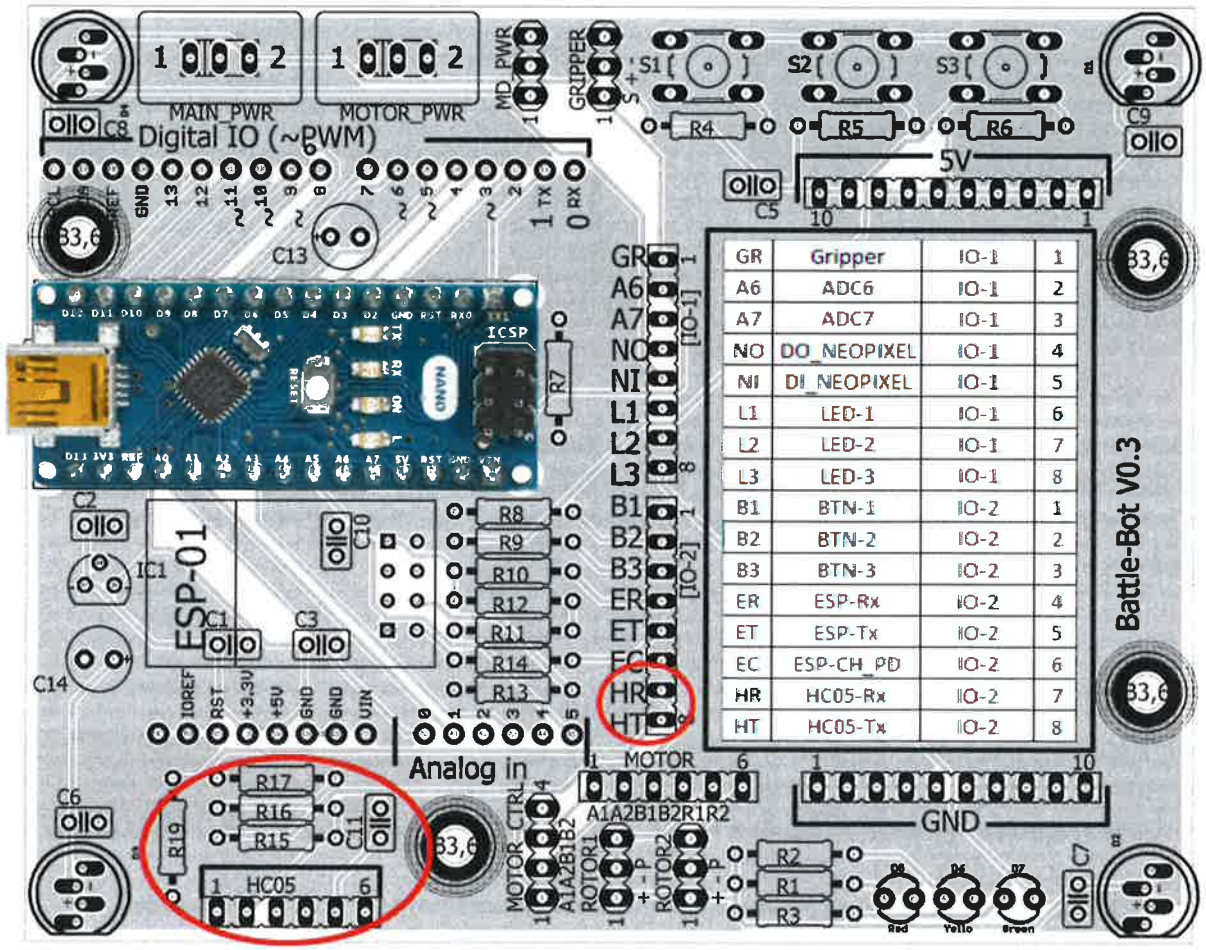
3.11 Line sensor

The RelayBot uses an analogue line sensor array that outputs an analogue signal proportional to the intensity of light being sensed by each individual sensor.

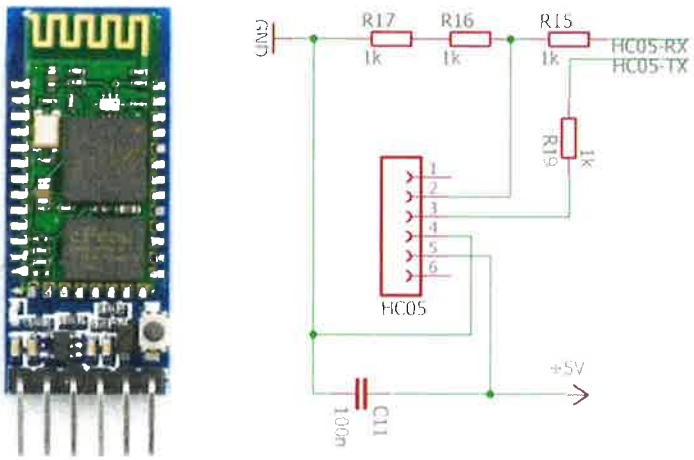
The 8 analogue outputs allow for a more precise measurement of the light intensity, compared to digital line sensor arrays which typically only output binary on/off signals. Analogue line sensor arrays are commonly used in applications that require high precision and accuracy in line detection and tracking, such as high-speed material handling, edge detection in machine vision, and precise line following in robotics.

3.12 Bluetooth module

The HC-05 bluetooth module in the kit can be plugged in here to communicate via bluetooth with e.g. your laptop, you still need to connect the Receive (HR) and Transmit (HT) pins to the suitable I/O pins of the Arduino.

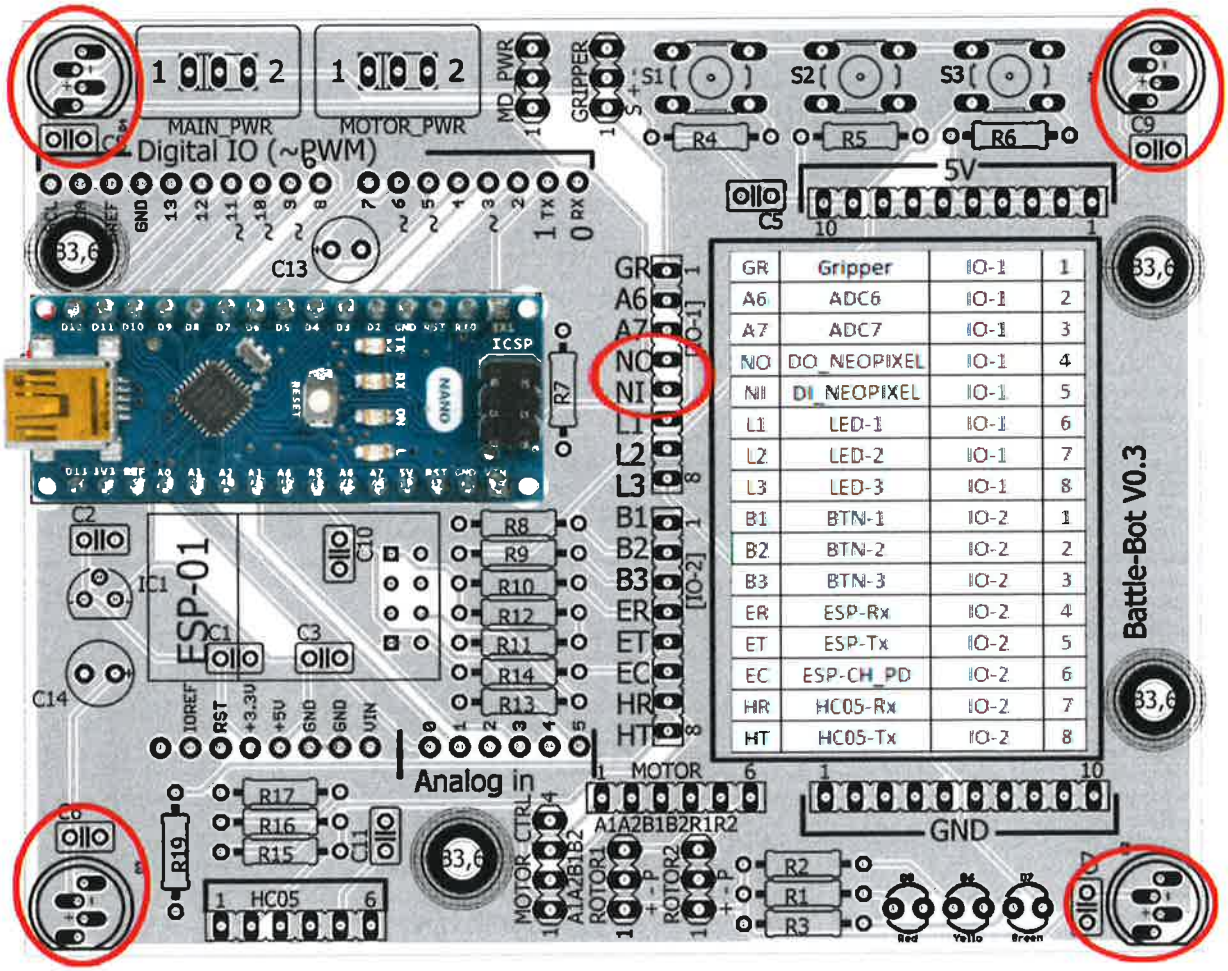


Because the HC-05 module does not work with 5V but with 3.3V the resistors R15-17 are added to form a so-called voltage divider.



3.13 NeoPixels

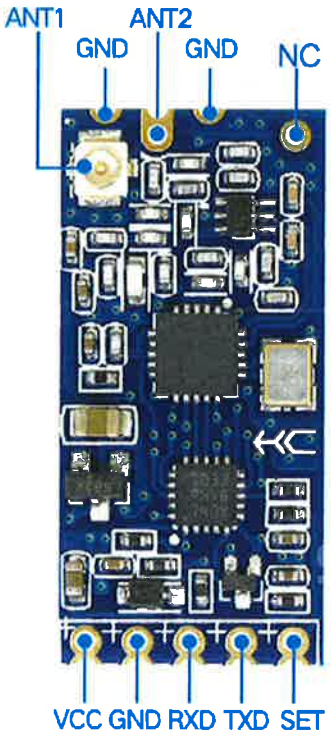
NeoPixel is a brand name of an addressable RGB LED, also known as an WS2812B LED. These LEDs are controlled individually and can be set to display a specific color and brightness. NeoPixels are popular in projects such as DIY lighting, wearable electronics, and interactive displays because of their ease of use, compact size, and the ability to chain multiple NeoPixels together to form longer strips or arrays. NeoPixels use a single data line to control multiple LEDs, and the color of each LED is determined by the data sent over this line. With an appropriate controller, such as our Arduino Nano, you can easily create complex lighting effects and animations with NeoPixels.



Each LED has a Data IN and a Data OUT pin and the 4 NeoPixels on the PCB are already connected in series. The Data IN (NI) of the first NeoPixel and the Data OUT (NO) of the last NeoPixel are available. When you connect the Data IN (NI) to an I/O pin of the Arduino Nano you can control all 4 LEDs with Adafruit's NeoPixel library.

3.14 Wireless Serial Module

The HC-12 is a wireless serial communication module.



#	Pin	Details
1	VCC	Power Input 3.3~5V DC
2	GND	Common Ground
3	RXD	Receive Input-UART TTL
4	TXD	Transmit Output-UART TTL
5	SET	LOW=Enter Config mode (AT)
6	ANT2	Spring Antenna/PCB
7	GND	
8	GND	
9	NC	
ANT1	ANT1	IPX External Antenna

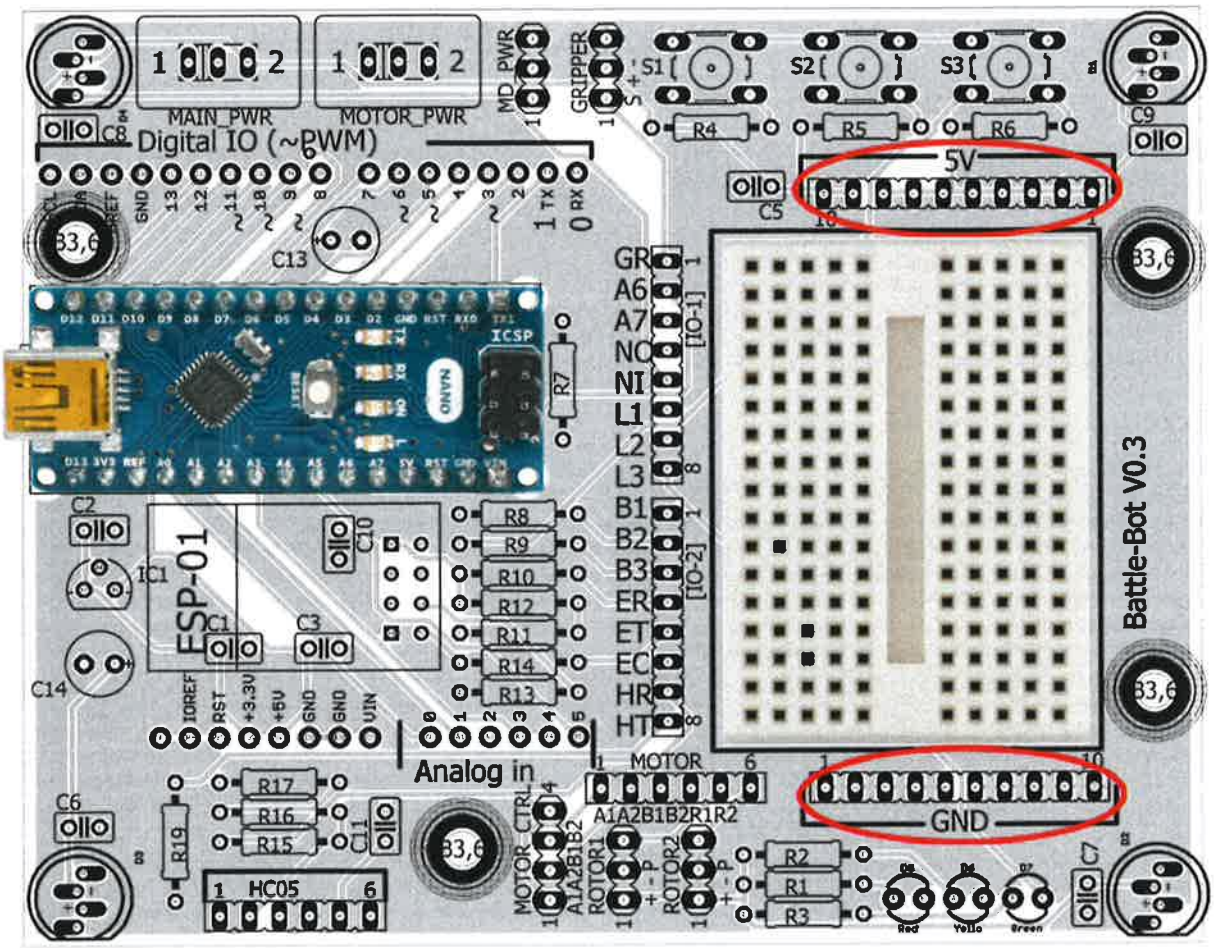
This module is **preprogrammed by the school** to work on legal frequencies and transmitting power in groups of 4 modules. Only modules in the same group will receive each other's transmissions.

Serial settings: 9600-8-N-1

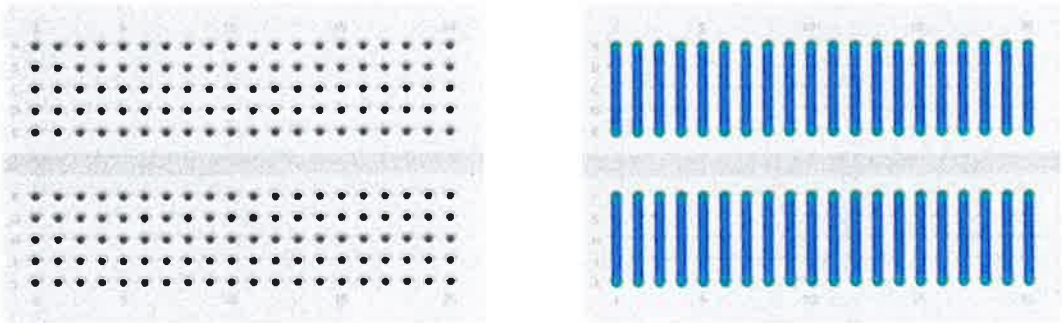
A diode and an electrolytic capacitor are needed to connect the module to the Arduino.

4. Testing

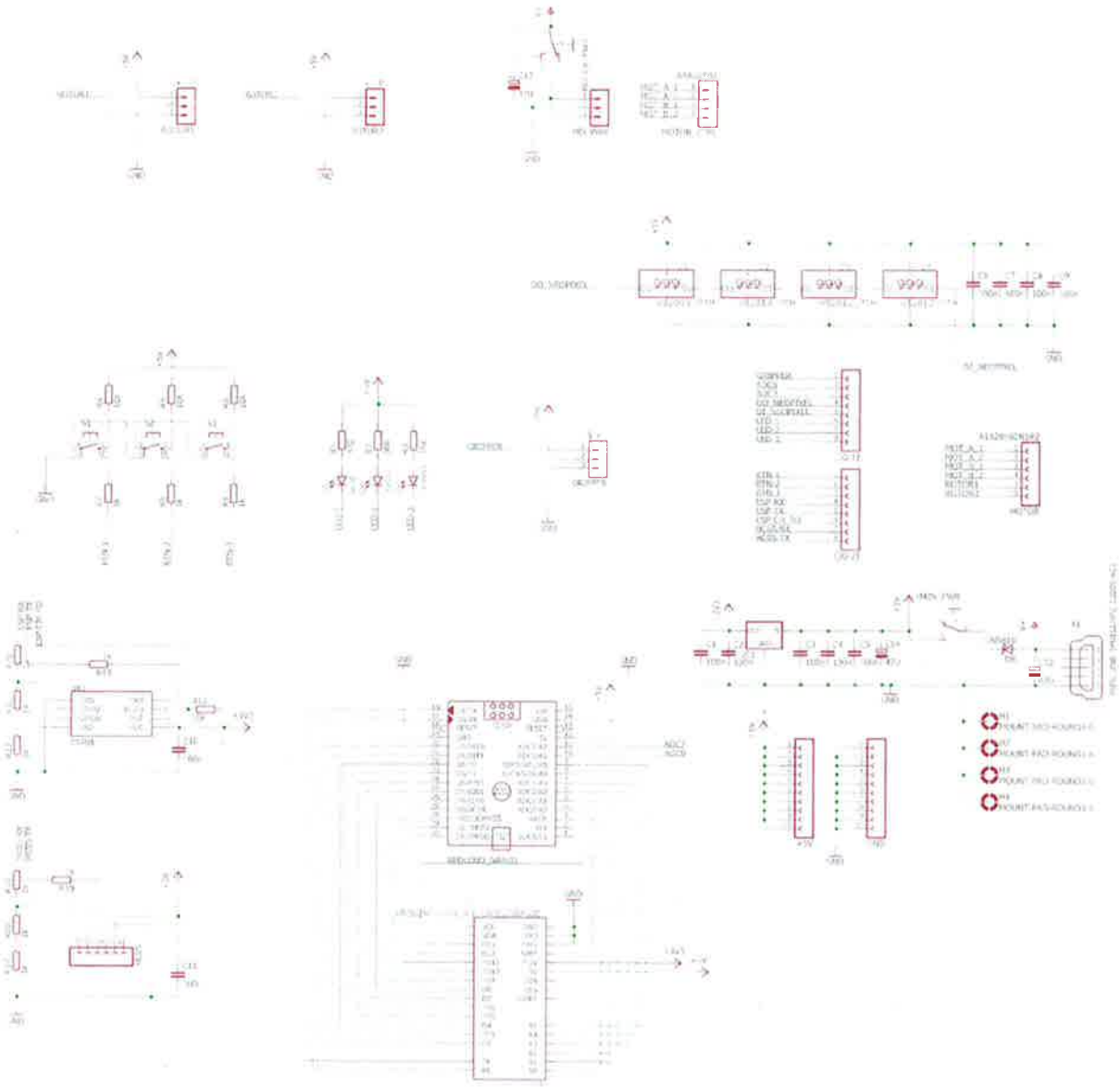
Testing with a breadboard is a common method used in electronics to prototype and experiment with circuit designs. A breadboard allows you to easily connect and disconnect components, using the jumper wires, to build and test your circuits without the need for soldering. With a breadboard, you can quickly test your ideas and iterate on your designs until you arrive at the desired outcome. By using a breadboard, you can save time and resources compared to traditional circuit building methods, making it a valuable tool.



To make it more easy, the RelayBot has power (+5V) and ground (GND) headers mounted near the breadboard.



5. Schematic



6. Pinout Arduino Nano