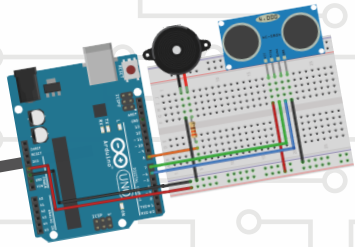
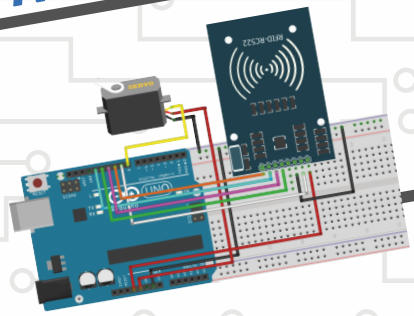
E-Book

Electronic Kits Application Book

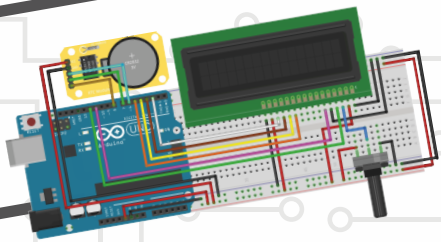
Compatible With Arduino UNO



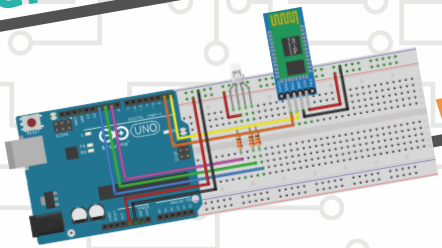
PARKING SENSOR



AUTOMATIC DOOR



DIGITAL CLOCK



WITH SMART PHONE REMOTE CONTROL

shop.robotistan

Welcome to the world of Electronics and Coding. Now that you've opened this book, you're eager to swim in the sea of wonder and learn new things. Although it may seem difficult to learn new things on these subjects, you will realize that it is very simple if you proceed step by step and with the right practices. In the first stages, there will be places that do not fit and seem meaningless. You will overcome this problem as you practice. You just need a little patience ...With an easy and accurate roadmap, you can learn how to program in Arduino, starting from easy and proceeding towards the more complex.

If you want to watch more detailed video explanations of the applications, you can visit our YouTube channel by scanning the QR code on the first pages of applications. If you want to access applications digitally, they are also available on our blog page at: http://maker.robotistan.com. You can access the codes written in the booklet both from the description section of the related videos and from our blog page.

This booklet was prepared by Robotistan Elektronik A.Ş. The aim of this booklet is to guide those who want to start Arduino in the easy and right way. We hope that these contents will be beneficial to everyone and that they will facilitate your learning process and allow you to make projects quickly.

For set contents, applications, videos and any kind of suggestions and questions, you can contact us via our e-mail address: info@robotistan.com

Robotistan Team

Contents

01

What is Arduino? How to Install and What to Do?

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

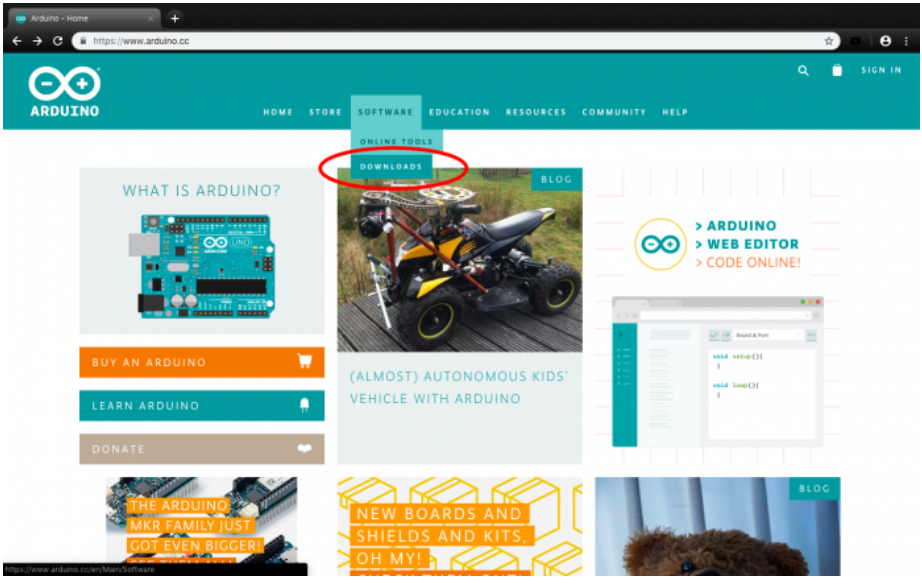
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

With this book you will enter the world of Arduino and go further to advanced applications. Arduino drivers and software must be installed on your computer before starting the applications. We recommend you to install the software before you connect the Arduino board to your computer with a USB cable. In this way, the Arduino drivers that come with the software are installed on your computer so that you can easily introduce the board and start using it immediately.

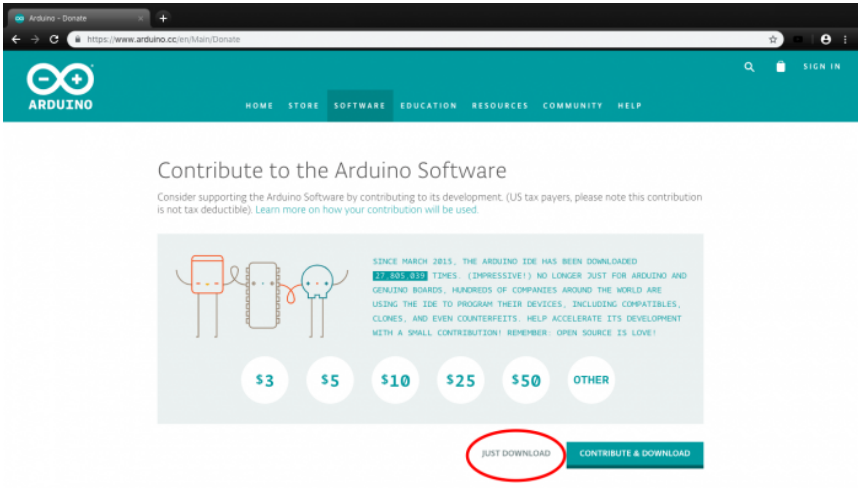
Downloading Arduino Software

To download Arduino, enter "Downloads" tab on **www.arduino.cc.**



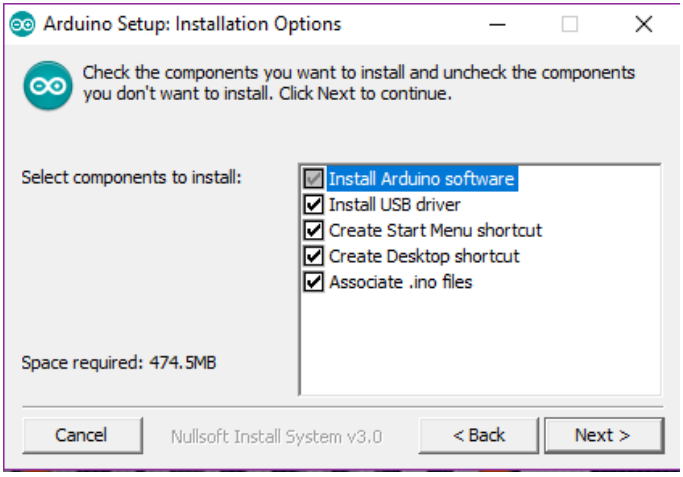
After clicking the Downloads tab, a screen appears where you will download the file according to your operating system. The latest version of the Arduino software at the time of writing of this guide was 1.8.7. Windows users can click **"Windows Installer"** option. The installation files for other operating systems are provided below.

Then a page opens asking us to donate. You can donate as you wish or download the software without donating with **"Just Download"** option.



Installing Arduino Drivers

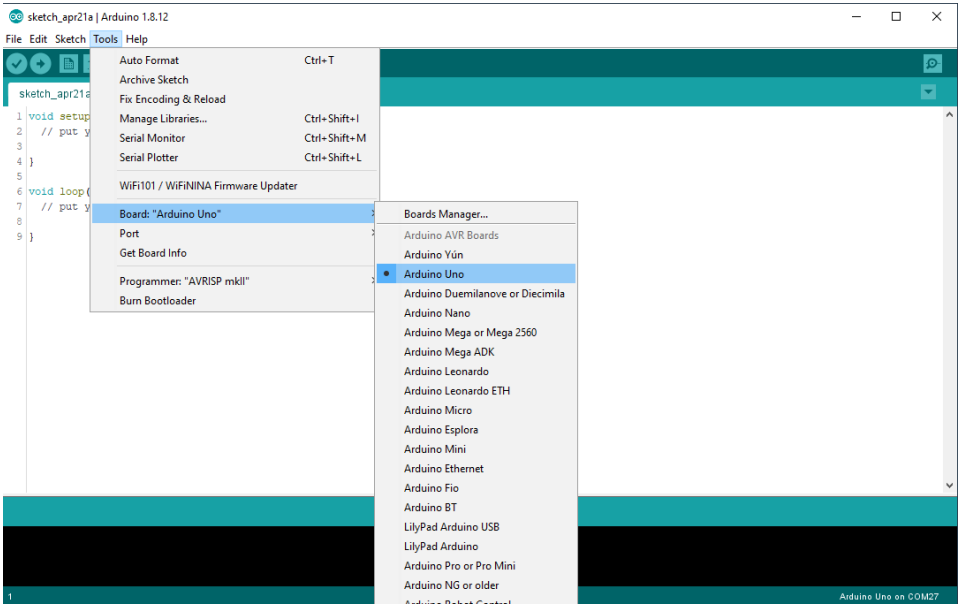
After that, the software installation file starts to download. After the download process, open the file and start the installation process. Make sure that the "Install USB driver" option, which appears during the installation process is selected.



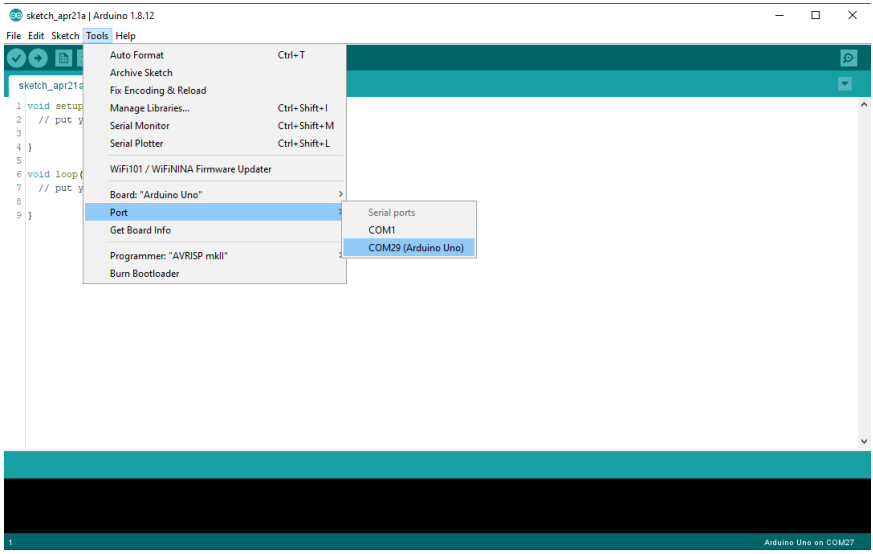
After the installation process, connect the board to the computer with the USB cable. The "New hardware found" window appears on our computer. If the drivers are installed with the software, the automatic installation option will automatically install the drivers of our Arduino.

Starting Arduino Program on Your Computer for the First Time

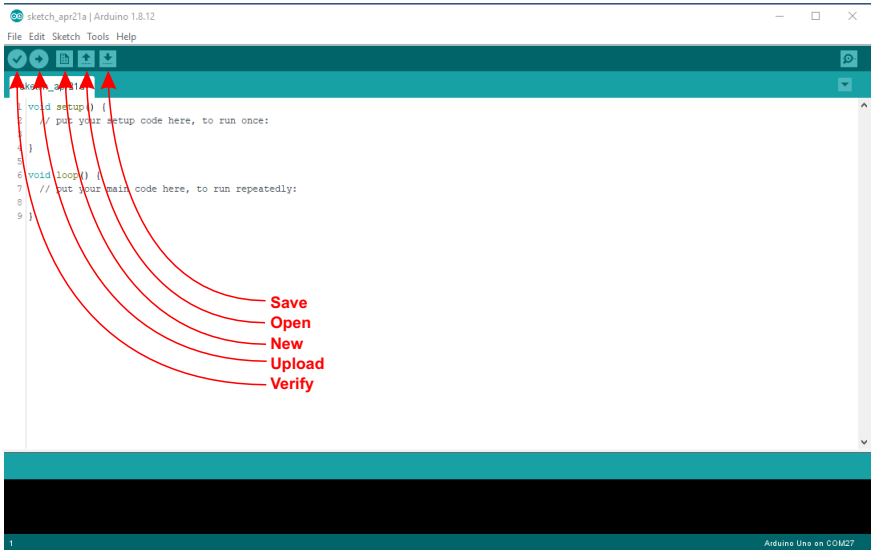
Now, you can open your Arduino program. The first thing you need to do after opening the program is to set the program to run with your Arduino UNO board. Click on the Arduino UNO option on **Tools> Board** menu.



Then, select the port to which Arduino is connected on the “Tools” menu, under the “Port” submenu. This port number may be different on each computer.

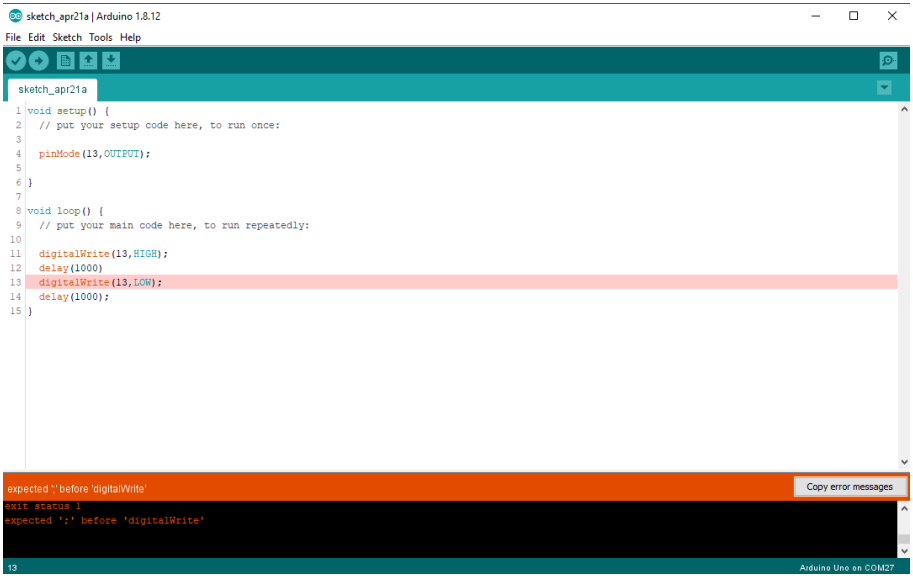


Now,”you”have”an”Arduino”program”ready”to”use.”



The function written in the "void setup ()" section in the program will only work once when the board is powered up. Input / output pins, serial port configuration etc. settings are adjusted in this section. The "void loop ()" section contains functions that will run continuously until the board is de-energized after running the commands in the "void setup ()" function.

After writing the program, first click the "Check" option before installing it on the board. The program first asks you to save the code into a folder on your computer, then compiles the code and notifies if there is any error.



For example, there is an error message about this line as we forgot to add a semicolon (;) after writing **"delay"**, the command before the **"digital Write"** function in this code.

If there is no error in the code you have written and your Arduino board is connected to the computer with USB,
You can upload our code to your board by clicking the **"Install"** option.

02

LED Lighting with Arduino Blink Application

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

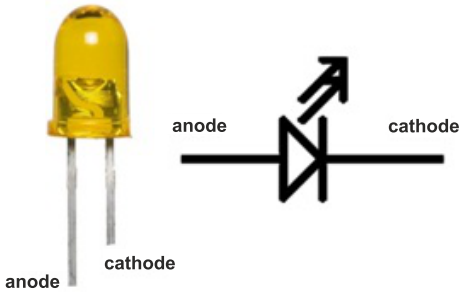
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Red LED
- 330 Ohm Resistor (Amber - Amber - Brown)
- 2 Pcs Male-Male Jumper Wire

What is LED?

LED is an abbreviation consisting of the initial letters of "Light Emitting Diode". Unlike the small 6V bulbs that are we are familiar with and used in most of our projects, LEDs have two different legs as anode and cathode. Of these, the anode must be connected to the positive voltage, the "+" terminal, and the cathode to the negative voltage, the "-" terminal or the ground (GND).



Voltage, Current, and Ohm's Law

Various circuit elements operate at different voltages. Arduino board operates with 5V voltage. However, the situation is slightly different for LED. The maximum current to pass through the LED should not exceed 15 mA (milliamps = 1/1000 amp). Remember that Arduino works with 5V. 5V value indicates the output voltage of the board. But the LED needs 15 mA current. Things are getting a little complicated. No need to be afraid! Everything has a solution.

LED is an abbreviation consisting of the initial letters of "Light Emitting Diode". Unlike the small 6V bulbs that are we are familiar with and used in most of our projects, LEDs have two different legs as anode and cathode. Of these, the anode must be connected to the positive voltage, the "+" terminal, and the cathode to the negative voltage, the "-" terminal or the ground (GND).

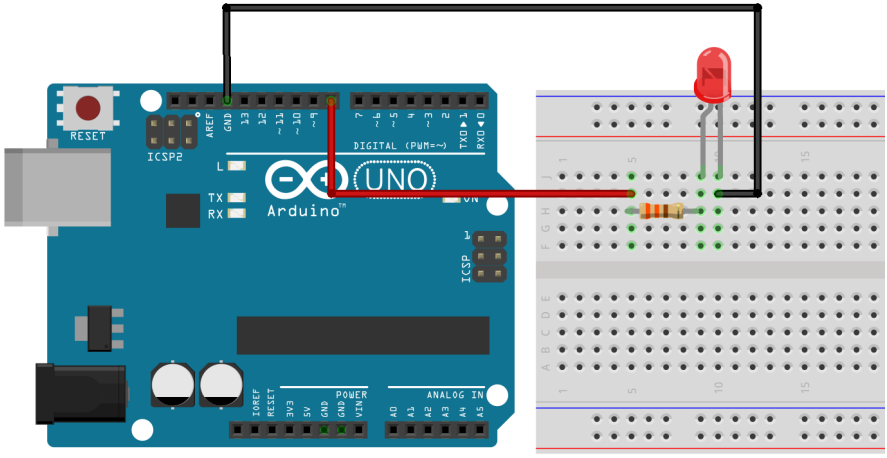
$$V = i x R$$

In this equation, "V" represents voltage, "i" represents current, "and "R represents resistance. If the LED, which needs 15 mA current, is connected to one of our Arduino's 5V output pins;

$$5V = 0{,}015A \times R$$

is the equation we get. If we take "R" out of this equation, we find the result as 333. This means that we need a resistance of 333 Ω (ohms) to use the LED with 5V. It is not crucial to find the exact value, we can use the 330 Ω resistor available.

Let's set up the circuit and then start writing our project code.



After setting up the circuit, open the Arduino IDE to write the code and open a new program page by selecting "File" from the tabs above and then "New". On the page that opens, you can delete the lines with "//" followed by comments.
On this page, you will write the codes in curly brackets "{}" in the lines starting with "void setup" and "void loop".

```
Arduino
1  void setup() {
2    pinMode(8,OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(8,HIGH);
7    delay(500);
8    digitalWrite(8,LOW);
9    delay(500);
10 }
```

In Setup, pin 8 on the board is set to output. If the pin to be used is not determined as output or input, the input or output functions that you will write in the continuation of the program cannot use that pin. Now that you have set the pin, let's write the code for LED lighting.

In the "loop" section, it first sets pin 8 to the HIGH logic level, i.e. 5V, waits for 500 milliseconds (equals half a second) without any action, and this time sets pin 8 to the LOW logic, i.e. 0V or ground level. After this process, the microcontroller waits for half a second without any operation thanks to the "delay" function.

By changing the time periods of the "delay" commands in this code, you can change the duration of the periods in which the LED is on or off. If you want to use another pin, all you need to do is replace the pin number in the "pinMode" and "digitalWrite "functions with the pin number you want to use. Do not forget to connect a 330 Ω resistor in series to our LED!

03

LED lighting with Button

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

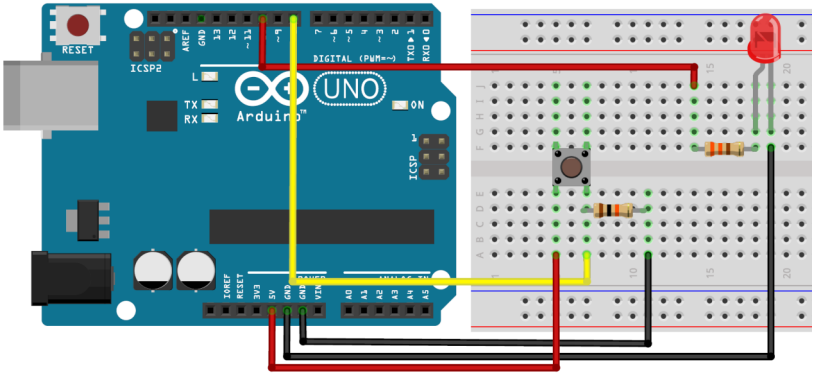
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Red LED
- Push Buton
- 330 Ohm Resistor (Amber - Amber - Brown)
- 1 0k Ohm Resistor (Brown - Black - Amber)
- 5 Pcs Male-Male Jumper Wire

In this application, you will learn to use pins on Arduino as input. In this way, you can ensure to be notified in Arduino when a button is pressed remotely. The LED circuit may be the same as the previous application. Only the leg to which the LED is connected will be #10 in this application. Let's set up the circuit and then continue with coding.



In order to read the data from the button, it is required to use it together with 10k Ohm resistor. You need to use "pull-up" or "pull-down" resistance to prevent interference on the pin and to detect wrong signals caused by the interference when the button is not pressed. In this application, we will use "pull-down" resistance. In this project, the pin reads the value of 0V, i.e. LOW logic level, when the button is not pressed. The pull-down resistance ensures that the voltage on this pin remains constant at 0V unless the button is pressed and the value is HIGH. Now that you have learned the logic of the circuit, let's continue with coding.

```arduino
#define button 8
#define led 10

int button_state = 0;

void setup() {
  pinMode(button, INPUT);
  pinMode(led, OUTPUT);
}

void loop() {
  button_state = digitalRead(button);
  if(button_state == 1){
    digitalWrite(led, HIGH);
  }
  else{
    digitalWrite(led, LOW);
  }
}
```

With "#define" line, we name the pin 8 as "Button" so that we can write much more memorable and easier code by writing "Button" instead of 8 to necessary places. We make the similar definition in the LED for the pin 10. The variables are used to access the data read in the software or the information we want to store later. The data stored differs according to the type of variables. The most commonly used variable is int, the abbreviation of "integer". This variable can hold numbers from -32767 to 32767. These numbers must be integers. If you want to assign a comma separated value, it will round it to an integer. In this code, we define the "button_state" variable and set the initial value as zero. The initial value does not necessarily have to be zero, but when you define an integer, it can be a random number in the variable. We set the initial value to zero in order not to cause any inconvenience in the software.

Set the" Button "pin (designated as number 8) as the input with the "pinMode" command. On the line below, set the LED pin (designated as number 10) as input.

When setting the input-output, it is enough to type "INPUT" for the buttons of input and" OUTPUT" for the pins of output. If you do not define the pins that you will use as input-output, these pins will not work as you want or stably. In the "loop" section, you will read the data coming from the button and evaluate this data with "if-else" command. In the end, the LED will be on or off according to the data being "1 "or "0".

04

Analogue Reading and Serial Communication with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

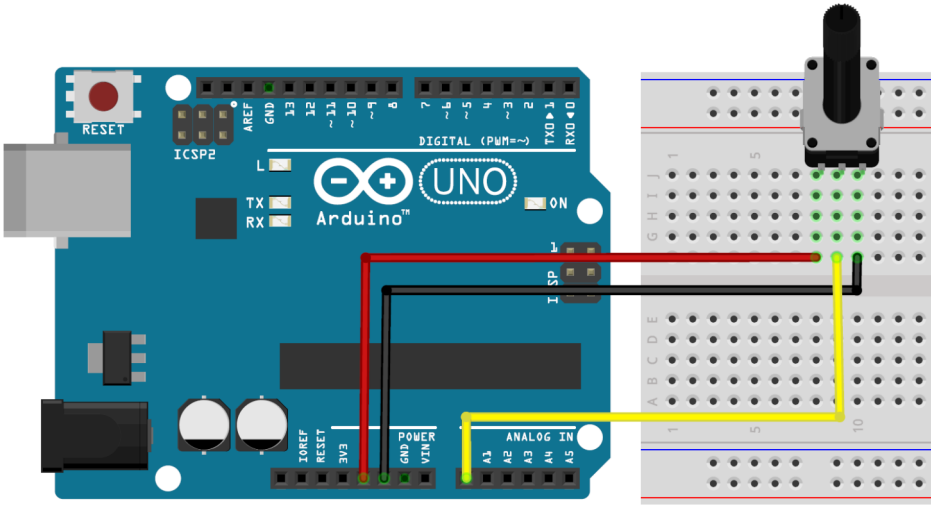
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 10k Ohm Potentiometer
- 3 Pcs Male-Male Jumper Wire

When you look at the Arduino board, you will see the "Analog Input" pins. It is possible to read the voltage on this pin by converting from digital to analog signal using these pins. The Arduino is capable of reading 0V (zero) and 5V digitally. If there are intermediate values between these two extremes, it cannot detect it and accept the incoming voltage as 0V or 5V according to the threshold value. Thanks to analog pins, you can detect intermediate voltage values from 0V to 5V and convert them to digital. You will use a adjustable resistor (potentiometer) to obtain signals at intermediate values. In the application, you will read the numerical value of the voltage coming from the analog input pin on the serial port. Let's set up the circuit and continue with coding.



```arduino
#define pot_pin A0

int value = 0;

void setup() {
  Serial.begin(9600);
  Serial.println("Reading Pot Value");
}

void loop() {
  value = analogRead(pot_pin);
  Serial.println(value);
  delay(300);
}
```

Define, as you did before for the previous codes, the "A0" pin as "potpin". In the next line, define the variable in"integer" type and the value name to store the values that the analog pin reads.

In the "setup" part, as we used the digital input-output in the previous software, the pins were adjusted according to their use. There is no need to define input / output for analog reading, you will not use the "pinMode" command in this software.
You need to initiate serial communication to send the data to the computer. This serial communication allows Arduino to communicate with the computer via USB connection and enabling to transfer the data to the computer.

Start this communication with "Serial.begin(9600);" line When Arduino starts running the code, it will first start communication with the computer. After the communication established, "Serial.println ("Pot Value Reading");" line and "Pot Value Reading" text will displayed on the serial monitor on the computer. "Serial.print" and "serial.println" commands will be explained in detail below.

05

LED Lighting with Potentiometer

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

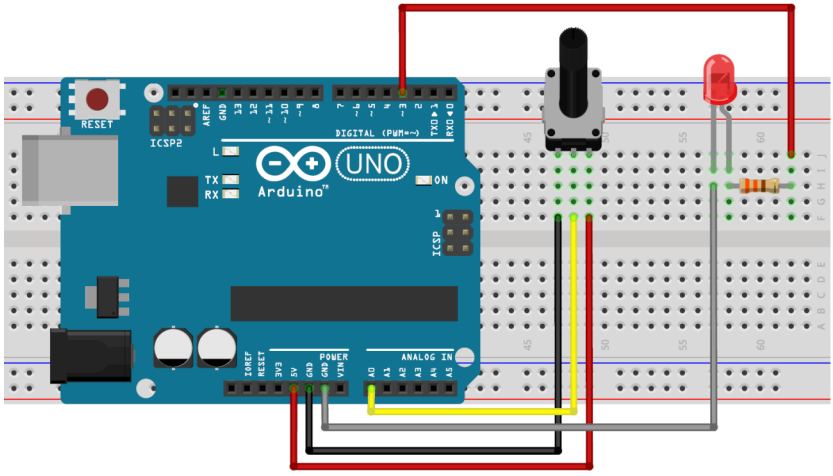
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino UNO
- Breadboard
- 10k Ohm Potentiometer
- Red LED
- 330 Ohm Resistor (Amber - Amber - Brown)
- 5 Pcs Male-Male Jumper Wire

In our previous application, you read the voltage value from the analog pin. In this application, you will check the brightness of the LED again according to the value received from the analog pin. In the first LED lighting application, it was possible to send 0V or 5V to the LED as we used a digital output. So, the led was either off or on. Using a new feature of Arduino, you will be able to send voltage to the LED at intermediate values in the range of 0-5 V. Voltage control enables to adjust the brightness of the LED. Until this application, you have learned about digital input-output and analog input. With this application, you will learn the analog output, i.e. the PWM feature. Now, let's set up the circuit and continue with coding.



PWM (Pulse with Modulation) is an abbreviation for Signal Width Modulation. You can watch the video of this application by scanning the square code at the end of the application for detailed information about PWM from 6 of the pins (~ 3,5,6,9,10 and 11) with the tilde (~) on Arduino Uno. After setting up the circuit, let's continue with coding.

```
Arduino
1  #define pot_pin A0
2  #define led 3
3
4  void setup() {
5    pinMode(led,OUTPUT);
6  }
7  void loop() {
8    int value = analogRead(pot_pin);
9    value = map(value,0,1023,0,255);
10   analogWrite(led,value);
11 }
```

Since you will not use digital input-output in this application, you will not make any adjustments in the "setup" section.

The aim is to read the data from the POT in the main program cycle and to send this data to the LED. First, read the data from the potentiometer with the command "analogRead". Write this value to the "value" variable. In the second line, compare the value from 0 to 1023 between 0 and 255 using the "map" command.

While analog reading is possible at 10 bit (2 ^ 10 = 1024) resolution, analog writing is possible at 8 bit (2 ^ 8 = 256) resolution. Now, you need to rate and print out the data read. Instead of "map" command, you can divide it directly into 4. After comparing, you can set PWM pins as output with "analogWrite "command.

06

Knigt Rider Application with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

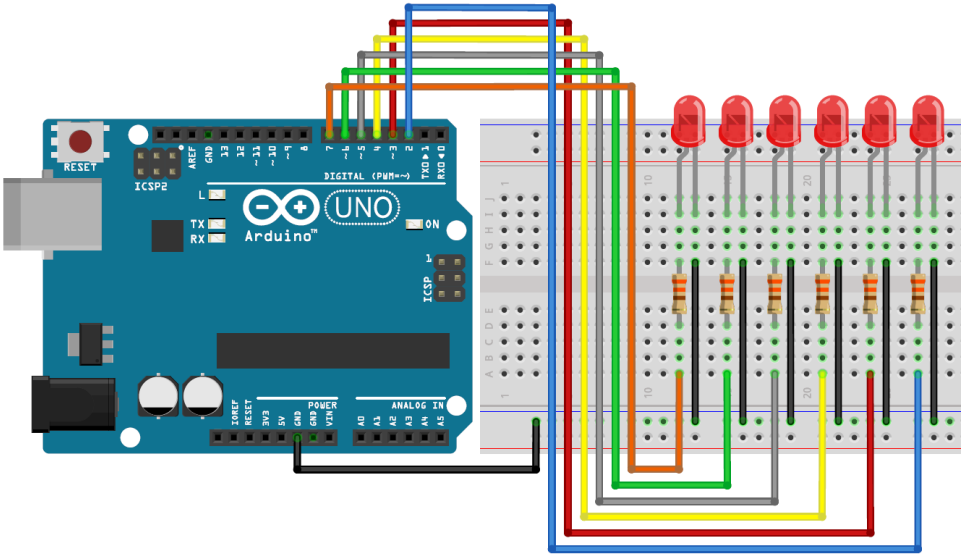
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino UNO
- Breadboard
- 13 Pcs Male-Male Jumper Wire
- 6 Pcs LED
- 6 Pcs 330 Ohm Resistor (Amber - Amber - Brown)

In this application you will see the use of the "for" loop. The "for" loop can be used for successive operations. In order to turn on 6 LEDs in the application, you will need to set output for them all and turn on respectively. This can be easily implemented with the output identification and LED lighting commands that you have learned before. If you want to make changes to the written code without using the "for" loop, you will have to make changes to each line one by one, however, in the "for" loop you can proceed much faster both in understanding and writing the code. After setting up the circuit, let's continue with coding.



```
int leds[] = {2,3,4,5,6,7};

void setup() {
  for(int i=0; i<6; i++){
    pinMode(leds[i], OUTPUT);
  }
}

void loop() {

  for(int i=0; i<6; i++){
    digitalWrite(leds[i],HIGH);
    delay(80);
    digitalWrite(leds[i],LOW);
  }

  for(int j=0; j<6; j--){
    digitalWrite(leds[j],HIGH);
    delay(80);
    digitalWrite(leds[j],LOW);
  }
}
```

When defining the digital pins one by one, you used “int” or “#define” to name the outputs before. This you will use an array in order to use 6 outlets. You can think of arrays as a set of variables. At the top of the code, define an array whose variable types are “int” (integer) and that are named “leds”. Separate the array elements with commas. Since we will use numbers from 2 to 7 from digital outputs, we have determined our elements in this way.

You can duplicate the elements of the array. When defining the array, you also can write in “leds []” the quantity of elements in the array. For example, when calling elements from an Array, such as “int leds [6] = {2,3,4,5,6,7};”, the number of the first element starts at 0 (zero). You can use “leds[*array element sequence number*]” in “setup” or “loop” to call the desired element. So, if you type “leds[0]” to call the zeroth element of the array, it will be equal to 2. 5. If you type “leds[5]” to call the array element, this will be equal to 7.

In this application, we want to turn on 6 LEDs, and in this case we need to set the output from 6 digital pins. The part of the "for" loop up to the first semicolon in parentheses is defined as the variable of the condition to be used for the loop.

In this software, we have defined it in parentheses because it will only be used for the "for" loop. If you wish, you can define a different variable or define the variable on the software and then use it here.

"İ <6" determines the condition of the "for" loop. As long as the variable "i" is less than 6, it will repeat the lines of code within the "for" loop. If the variable "i" is equal to or greater than 6, it will continue to run the code from where the loop ends instead of executing the "for" loop.

We want the variable "i" to increase its value by 1 each time we do the "for" loop with "++ i + + uz". Thus, the first value of the variable is 0 (zero). Also, in the commands you give digital output, when you write the "i" variable, it calls the element from the array. Hence, when you write "pinMode(ledler[0], OUTPUT)", the software calls the zeroth element from the "leds" array and perceives as "pinMode(2, OUTPUT)". In this way, we define the pin 2 as output. Since the "for" loop will do this from 0 to 5, we define 6 pins as output with a single line.

In the "loop" section, the use of the "for" loop proceeds with the same logic as in the "setup" section. At this time, instead of defining the output, we turn 6 LEDs on and off successively with "digitalWrite" command. You can watch the detailed video description of the "for" loop by scanning the following qr code.

07

Automatic Lamp Application with LDR

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

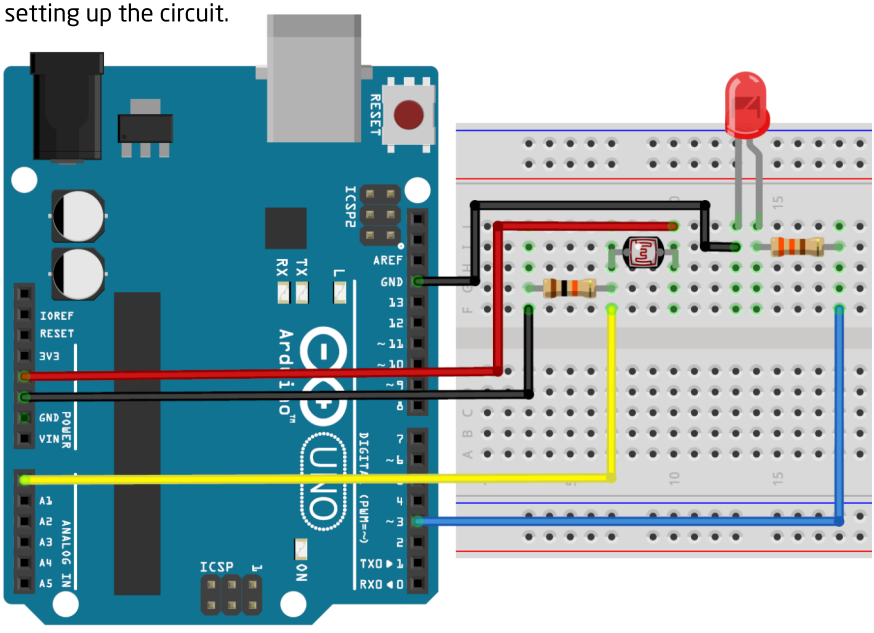
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 5 Pcs Male-Male Jumper Wire
- 330 Ohm Resistor (Amber - Amber - Brown)
- 10k Ohm Resistor (Brown - Black - Amber)
- 5mm Red LED
- 5mm LDR

In this application, you will learn to read the data from LDR, which can detect the light in the environment and according to this data, and turn the LED on and off. LDR (Light Dependent Resistance), i.e photoresistor, changes the resistance according to the amount of light in the environment. This resistance change can be detected by the Arduino board. In this way, since it is possible to know the amount of light in the environment, you can make an automatic lamp by turning on the LED when the environment is dark and turning it off when there is light. You will also send the data to the computer and display it on the serial monitor. Let's start by setting up the circuit.



```
#define led 3

void setup() {
  pinMode(led,OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int light_value = analogRead(A0);
  Serial.println(light_value);
  delay(50);
  if(light_value > 900){
    digitalWrite(led,LOW);
  }
  if(light_value < 850){
    digitalWrite(led,HIGH);
  }
}
```

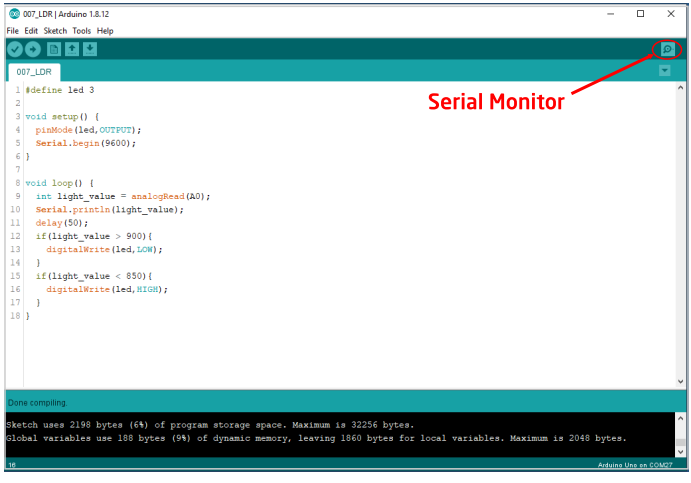
In coding part, name the pin to which you will connect the LED in the first line. You will do this with "#define" command. After this process, instead of writing 3 when needed, we will simplify the process by writing "led".

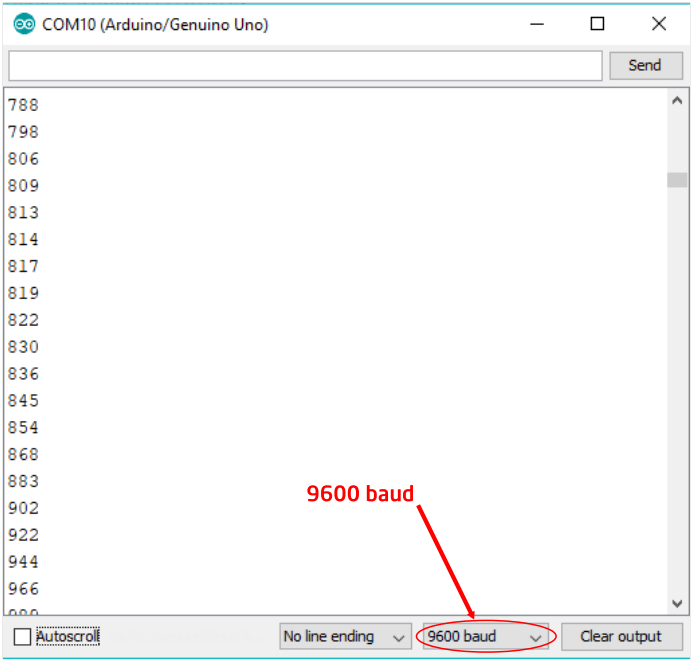
You need to output the pin to which the LED is connected in the "setup" part of the code, and start serial communication. We started serial communication at 9600 baudrate. This number determines how fast the computer and Arduino board communicate. This number can't be randomized. It is required to use pre-determined speeds.

You canuse300,600,1200,2400,4800,9600,14400,19200,28800,38400 or 115200 baudrates. The baudrate value that you type in the Arduino code must be the same as the speed in the lower right corner of the serial monitor that you will turn on the computer.

Define an "int" type variable named "light" in the loop where the main algorithm will run and print the value the LDR reads. Send this value to the computer via serial communication. After waiting for 50 ms, evaluate whether the value received with "if" command is below or above the desired values to decide. If there is low light in the environment, the value on LDR will decrease. In our environment, we want the LED to turn on after the value of 850. Since the value will increase when the environment starts to be lightened, we want it to turn off after 900.

After sending the code to the board, you can see the data by clicking on the "Serial Monitor" button in Arduino IDE. When you put our hand over the LDR, the readings will change as the light intensity changes.





08

RGB LED Application

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

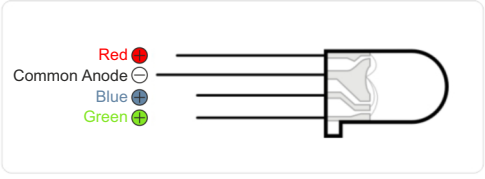
YouTube

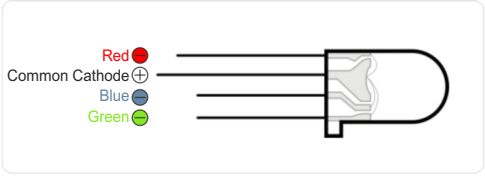
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 330 Ohm Resistor (Amber - Amber - Brown)
- RGB LED
- 10k Ohm Potentiometer
- 9 Pcs Male-Male Jumper Wire

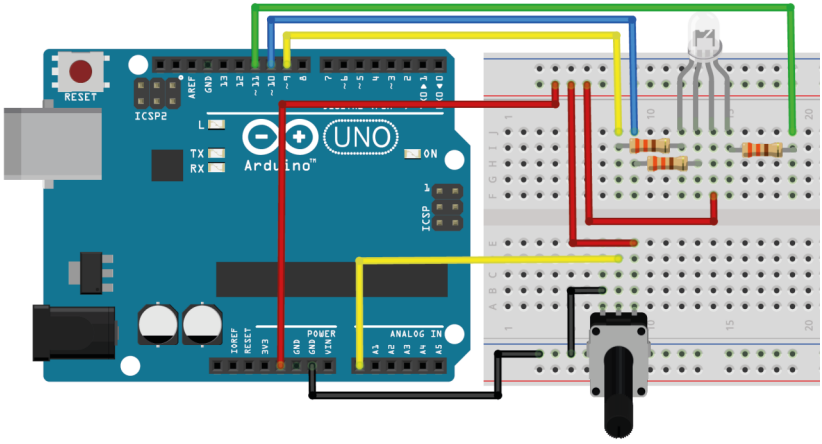
RGB LED has 3 led structures including red, green and blue colors. RGB (Red, Green, Blue) name is formed by combining the initial letters of the colors it contains. When you consider 3 LEDs, there must be a total of 6 legs, each with a plus and a minus. The RGB LED has 4 legs. It uses 3 colors inside plus legs commonly. When energized from the plus leg, the corresponding LED will turn on when a color is connected to its negative leg. There are also RGB LEDs that have common negative legs instead of a common positive legs. In this case, the corresponding LED will turn on when you give the positive signal. You can use the terms of common anode (positive) and common cathode (negative) to describe LEDs. In this case, the LED we will use will be the common anode.





In this application, we want to light up intermediate colors besides full brightness. We need to use PWM to turn the LED on to the desired brightness level. Since PWM feature is present in certain legs (3,5,6,9,10,11), we will pay attention to use these legs when making connections. Let's start by setting up the circuit.

In this application, we want to light up intermediate colors besides full brightness. We need to use PWM to turn the LED on to the desired brightness level. Since PWM feature is present in certain legs (3,5,6,9,10,11), we will pay attention to use these legs when making connections. Let's start by setting up the circuit.



In the code section, like in other software, you will assign names and create the variables. When assigning names, it can assign variables as shown here. You can create an integer type "potPin" variable and write A3 in it. In this case, everywhere you write "potPin", it will be like you have written number A3. You can also use the "define" command when making this definition. The command must be used as "#define potPin A3". There is no need to put the equal sign and a semicolon at the end of the line.

Arduino

```
1  int pot_pin = A0;
2  int pot_value = 0;
3
4  int red_pin = 9;
5  int green_pin = 10;
6  int blue_pin =11;
7
8  int red_value = 0;
9  int green_value = 0;
10 int blue_value = 0;
11
```

```
Arduino
12 void setup() {
13   pinMode(red_pin,OUTPUT);
14   pinMode(green_pin,OUTPUT);
15   pinMode(blue_pin,OUTPUT);
16 }
17
18 void loop() {
19   pot_value = analogRead(pot_pin);
20
21   if(pot_value < 341){
22     pot_value = (pot_value*3) / 4;
23
24     red_value = 255 - pot_value;
25     green_value = pot_value;
26     blue_value = 0;
27   }
28   else if(pot_value < 682){
29     pot_value = ((pot_value-341)*3)/4;
30
31     red_value = 0;
32     green_value = 255- pot_value;
33     blue_value = pot_value;
34   }
35   else{
36     pot_value = ((pot_value-683)*3)/4;
37
38     red_value = pot_value;
39     green_value = 0;
40     blue_value = 255-pot_value;
41   }
42   analogWrite(red_pin, 255-red_value);
43   analogWrite(green_pin, 255-green_value);
44   analogWrite(blue_pin, 255-blue_value);
45 }
```

In the “setup” part of the project, it is enough to determine the pins to be output. You will need 3 outputs for the negative legs of the red, green and blue LEDs.

When you start the main program loop, you will continue by evaluating the value read from the “potPin” pin. After the reading, you will use the “if”, “ifelse” and “else” commands to make the necessary LEDs according to the corresponding values. If the value in the first “if” command is less than 341, we want it to perform operations under the “if” parenthesis.

In "if", we divide the value in "potDeger" by 4 to multiply by 3 to compare between 0-255 (values with possible PWM outputs). In this way, we find 255 with the value of 340 a result of this calculation

The analog pin can read from 0 to 1023. This value is divided into 3 different regions because there are 3 LEDs. These regions are identified as 0-341, 342-681, 682-1023. We use the "if-else" structure to determine in which of these regions is the incoming value. The incoming value is evaluated with the "if" command; if it is the desired value, it is set to "if" and skipped in other conditions (if else, else). If the "if" condition cannot be met, "if else", i.e. the second region is evaluated. Depending on whether this is provided, either apply the commands in it or pass to the "else" line. You can specify multiple digits instead of 3 by duplicating the "if else" lines.

Similar commands are applied in each step. Only the assigned value is in different colors. For example, when we look inside "if", the incoming value is between 0 and 255. Here, it is needed to clarify that when a LED is controlled by PWM, it is lit at full brightness when you give 255. However, in this circuit, there will be a reverse effect as we have connected the negative leg of the LED to the PWM , instead of positive. When the PWM output is set to 0 (zero), the LED will be lit at full brightness and will turn off at 255 . We will solve this problem by subtracting the values from 255 before sending them to the LEDs to overcome the reverse situation. In this case, it will be possible to write the code as if a normal LED were connected within the "if" conditions. In the first "if", subtract the "potValue" from 255 and send it to the red LED. And send the potValue directly to the green LED. Set the value 0 (zero) to turn off the blue LED. To switch between colors, completely turn off one LED in each of the 3 steps and ensure that the sum of the values sent to the other LEDs is 255. When you turn the potentiometer with this method, one color increases while the other decreases and color transitions occur.

09

Temperature Measurement with NTC

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

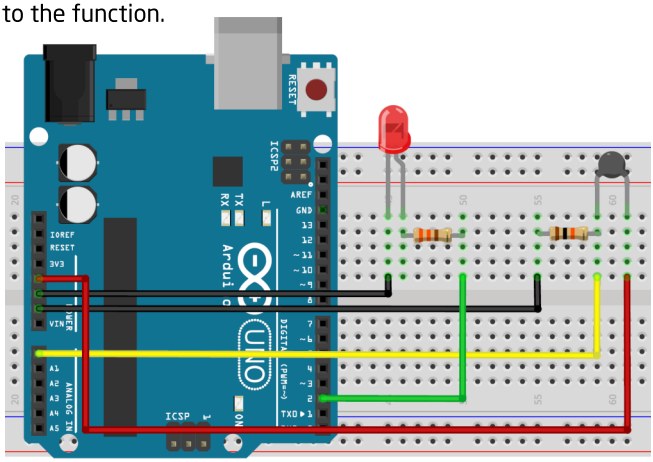
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 5 Pcs Male-Male Jumper Wire
- NTC Temperature Sensor
- 5mm Red LED
- 330 Ohm Resistor (Amber - Amber - Brown)
- 10K Ohm Resistor (Brown - Black - Amber)

As in the LDR application, we will ensure that what we want is done by reading and interpreting the analog signal while reading the temperature. NTC (Negative Temperature Coeficient) is an element that reduces its internal resistance against the rise in temperature. One of the elements commonly used instead of NTC is PTC. PTC reacts by increasing its internal resistance against temperature rise. In order to convert the data read from NTC into a temperature unit, it needs to be processed. At the same time, because the variable resistance value according to the temperature increase of the NTC sensor is not constant, it is necessary to pass through logarithmic functions. You do not need to examine these functions in detail at this time, it is just enough to know the processes. After learning the logic of using the NTC sensor, you can examine this part in detail. Let's start by setting up our circuit first. In this application, you will create a different function and do some operations by going to the function.



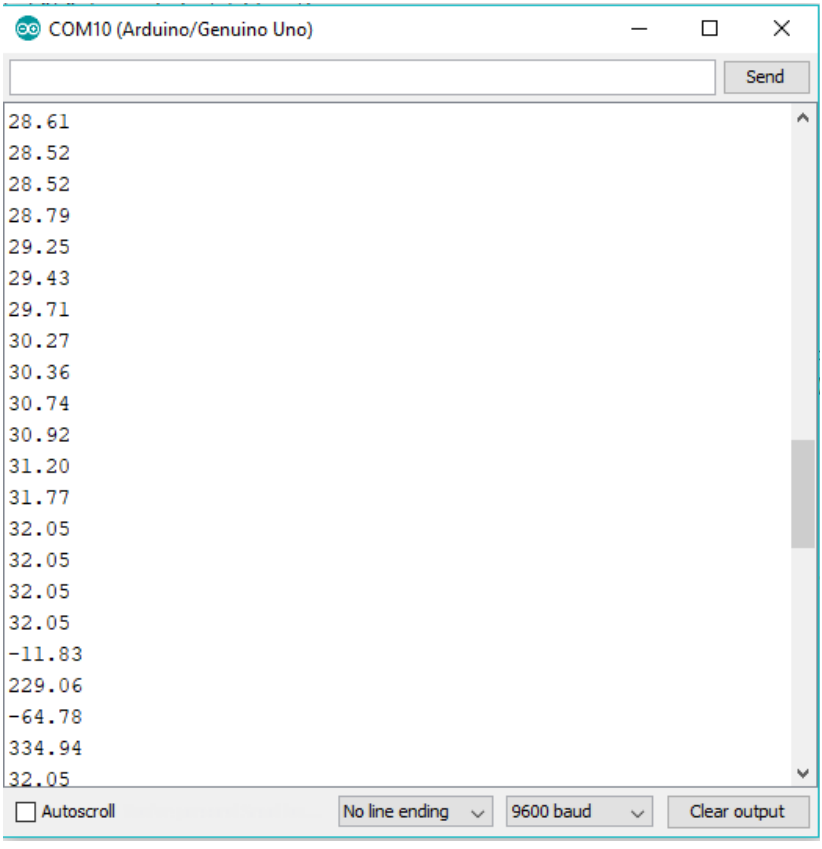
```arduino
#include <math.h>

#define led 2

void setup() {
  Serial.begin(9600);
  pinMode(led,OUTPUT);
}

double Termistor (int analogOkuma){
  double temp;
  temp = log(((10240000 / abalogOkuma) - 100000));
  temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * temp * temp)) * temp);
  temp = temp - 273.15;
  return temp;
}

void loop() {
  int value = analogRead(A0);
  double temp = Termistor(value);
  Serial.println(temp);
  if(temp > 30){
    digitalWrite(led,HIGH);
  }
  else{
    digitalWrite(led,LOW);
  }
  delay(250);
}
```

Since you will use logarithmic functions in the code section, you need to include the math library. Include the math library with the "#include <math.h>" line. At the bottom, name the 2nd pin with the "#define led 2" line.

Since you will send the data to the computer in the "setup" section, start a serial communication and define the pin named "led" as the output.
In the main loop, read analog data on the NTC connected to the pin "A0". Write this data in the "variable" value. You will send this data to the "Thermistor" function and convert it to a temperature value. You can also use it in the main loop by typing the codes in the "Thermistor" function without defining a function. When you define a function, you get rid of the complex structure of the code and divide it into sections. In this way, the main code will be simpler and easier to write and understand later when read.

You have written the mathematical operations into the "double Termistor (intanalog Reading)" function located between the" setup "and" loop "functions. Since you specify the function name, you can assign any other name.

When you send the read value to the function, the return value of the function will be the temperature value, and you will display this variable on the serial monitor on your computer with serial communication. Evaluate the temperature value with the "if" condition, and turn on the LED if it is above 30 degrees. If it is below 30 degrees, turn off the LED. In this way, we have made a temperature alarm system based on the temperature value. We put a 250 ms wait at the end to prevent the software from repeating itself very fast. You van open the serial monitor via Arduino IDE and observe the temperature values.



10

Making Parking Sensors with Ultrasonic Sensor

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

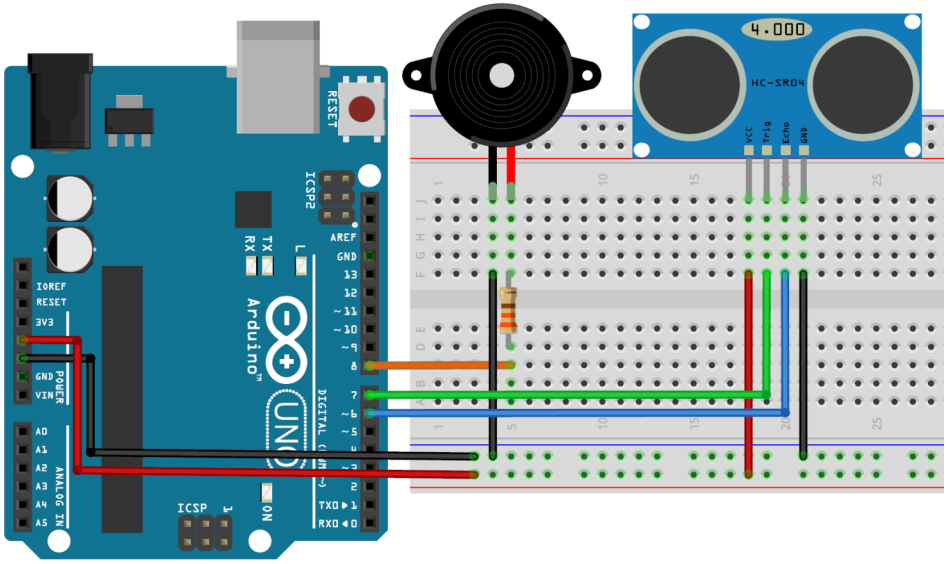
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 8 Pcs Male-Male Jumper Wire
- Buzzer
- 330 Ohm Resistor (Amber - Amber - Brown)
- HC-SR04 Ultrasonic Sensor

You will meet a new application model in this ultrasonic sensor application. The HC-SR04 ultrasonic sensor has metal parts that can send and detect sounds. After the sound is sent from the sensor, if it is reflected from any object or obstacle available and comes back to the sensor. The receiver can detect the reflected signal and make a measurement.

The duration between the time when the sensor is sent and received is measured in order to calculate the distance. Since we know the speed of sound in the air, we can calculate the total distance by time and speed. Let's continue with setting up the circuit.



```arduino
#define echoPin 6
#define trigPin 7
#define buzzerPin 8

int maximumRange = 50;
int minimumRange = 0;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  int value = distance_func(maximumRange, minimumRange);
  melody(value*10);
}

int distance_func(int maxrange, int minrange)
{
  long duration, distance;

  digitalWrite(trigPin,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = duration / 58.2;
  delay(50);

  if(distance >= maxrange || distance <= minrange)
  return 0;
  return distance;
}

int melody(int dly)
{
  tone(buzzerPin, 440);
  delay(dly);
  noTone(buzzerPin);
  delay(dly);
}
```

Name the pins that you will use with "#define" commands in the software section. Define integer type variables named "maximumRange" and "minimumRange". In the "setup" section, set the pins to be input and output.

The main program cycle seems too short. In this section, first go to the distance function. Define the variables “duration” and “distance in ”long” type. ”long” is a variable like ”integer” you used before. It can hold much larger numbers than the integer variable. Numbers from +2,147,483,647 to -2,147,483,647 can be assigned in it. When it is defined by the number volume it can hold, it uses 2 times more memory than the “integer” variable. After defining the variable, make the sensor’s “trig” pin high and low, so that the sensor sends a sound wave to the physical environment. After sending the sound wave, we wait for the sound wave come back echoing from the object with the ”pulseIn(echoPin,HIGH)” command. This time can be measured with the ”pulseIn” command. Print this value to the ”duration” variable. Now that the time has been measured, you need to calculate the distance. Divide the measured time by “58.2” to convert it to distance according to the speed of sound. When you get the distance value, if this value is not between the minimum (2 cm) and maximum (400 cm) values that the sensor can measure, command return with 0 (zero) value. If it is in the desired interval, return to the main function and write data into the “measurement” variable.

The value in the measurement variable is multiplied by 10 and sent from the main function to the “melody” function. This value will be used for the waiting periods in the ”melody” to determine the time between 2 bip sounds. The sensor beeps in short intervals if it measures short distance and in long intervals if it detects an object in long distance.

11

Motor Control with Sound

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

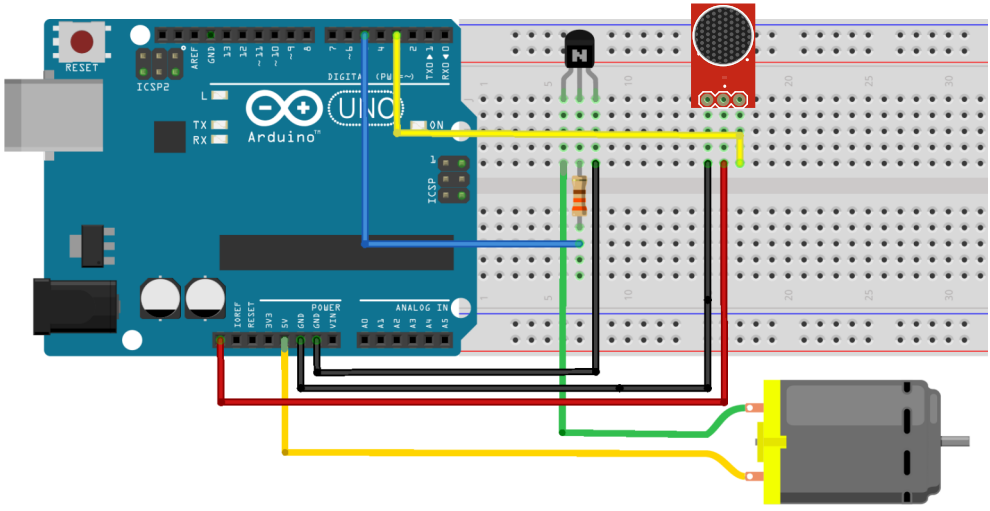
Materials Required:

- Arduino Uno
- Breadboard
- 5 Pcs Male-Male Jumper Wire
- DC motor
- 330 Ohm Resistor (Amber - Amber - Brown)
- BC547 NPN Transistor
- Sound Sensor Board

In this example, you will provide motor movement according to the value read by the sound sensor. The sound sensor provides a digital output by measuring the ambient sound level with a microphone. The sensor circuit amplifies the audio signal received from the microphone and converts the analog audio signal into a digital signal according to the threshold level.

Since the motor draws excessive current, a motor driver board is used in such circuits. The motor drive powers the motor according to the signal it receives from the Arduino. This way you can safely control the motor without damaging Arduino.

The threshold value of the sound level can be adjusted with the potentiometer on the sound sensor. This adjustment can be made with the help of a screwdriver.



```arduino
#define sensor_pin 3
#define motor_pin 5
int motor_state = LOW;

void setup() {
  pinMode(sensor_pin, INPUT);
  pinMode(motor_pin, OUTPUT);
}
void loop() {
  if( digitalRead(sensor_pin) ){
    if(motor_state == LOW){
      motor_state = HIGH;
    }
    else{
      motor_state = LOW;
    }
    digitalWrite(motor_pin, motor_state);
  }
  delay(50);
}
```

First, define the variable named sound. This variable enables to keep the value read by the sensor in memory.

In the "setup()" section, set the signal pin from the sound sensor as input. We then set pin 5 to the motor drive as output.

In the loop () section, read the data from the sound sensor with the "digitalRead()" function and evaluate it in "if". If the sound comes back, write "HIGH" in" MotorStatus "variable. Otherwise, write "LOW. When the "if-else" loop is over, send the "MotorStat" variable as the digital output in the "digitalWrite(MotorPin, MotorStat)" line. You can also use the motor without using variables, with "digitalWrite (MotorPin, HIGH)" or "digitalWrite (MotorPin, LOW)" commands in "if" and "else". But, remember to put a 50 ms wait at the end of the code so that the main loop does not start again too quickly.

When the sound level exceeds the threshold value, the sound variable takes the value HIGH. In this case, the HIGH value is sent to the motor drive connected to pin 7 and the motor rotates for 300 milliseconds.

When the sound level is below the threshold value, the sound variable takes the value LOW. In this case, the LOW value is sent to the motor drive connected to pin 7 and the motor is stopped.

12

Servo Motor Control with Joystick

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

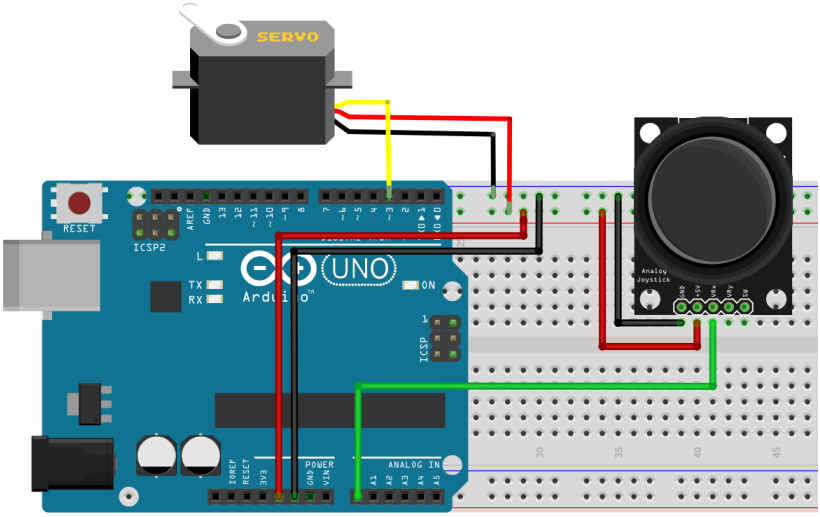
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 1 Pcs Servo Motor
- 1 Pcs Joystick
- 8 Pcs Male-Male Jumper Wire

The joystick outputs position as an analog signal depending on whether the lever is moved forward-backward or right-left.

The servo is a motor that positions itself at an angle according to the signal received from the data pin.



When the lever on the joystick is moved back and forth, the voltage values on the VRX pin change and the voltage values on the VRY pin changes when it is moved to the right or left. When you click on the joystick, the SW pin gives 5V output. In this example, the VRX pin will be used as we will only use one servo. Since the data from the VRX pin is analogue data between 0 and 5V, connect it to the A0 pin on Arduino. Connect the data pin of servo motor to pin 3 which can give analog output. The sample code allows the servo motor to rotate between 0 and 180 degrees with the data from the joystick.

```
#include <Servo.h>
Servo motor;
int value;
int degree;

void setup() {
  motor.attach(3);
}

void loop() {
  value = analogRead(A0);
  degree = map(value, 0,1023,0,180);
  motor.write(degree);
}
```

First, add the “Servo.h” library to the code. Then create a servo motor called “motor”. In the “setup ()” section, introduce to the software the “motor” connected to pin 3.

In “loop () section, first make the reading from the joystick connected to the A0 pin with the ”analogRead () ” function and equalize the variable named ”reading” to the value read with the “analogRead()” function.

Arduino Uno provides data from 0 to 1023 via analog reading pins. However, the servo motor can move between 0 and 180 degrees. Therefore, you need to use the “map ()” function to control the servo motor with the value read from pin A0. The “map ()” function allows the input variable to be proportional to the desired range.

The reading value, which is proportional to the ”map()” function, is equalized to the degree value. Finally, the degree value is printed on the servo motor so that the servo reaches the desired degree.

13

LED Control with IR Controller

robotistan BLOG

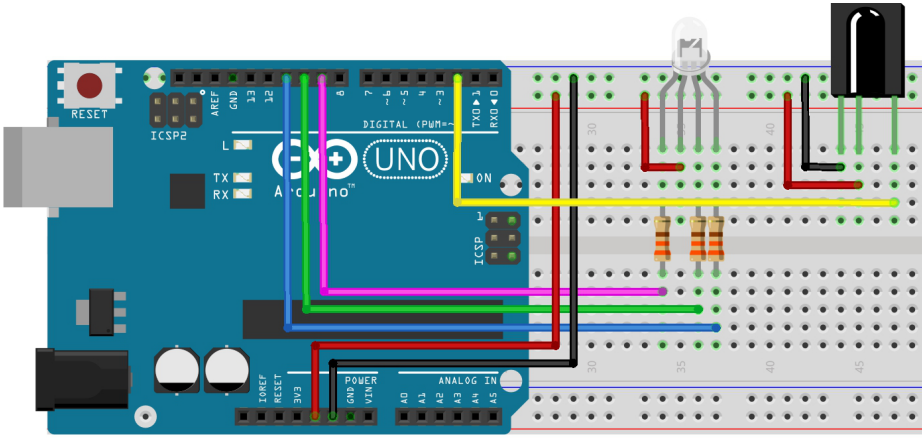
You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 9 Pcs Male-Male Jumper Wire
- RGB LED
- IR Controller and Module
- 3 Ohm Resistor (Amber - Amber - Brown)



The application enables the LED to produce light in the desired color based on the infrared signals received from the IR controller. Red LEDs turn on when we press 1, green LEDs turn on when we press 2, and blue LEDs turn on when we press 3. When we press 4, all LEDs produce white light flashing at the same time. When we press 0, all "LEDs" turn off. No feature was assigned to other buttons.

There is an infrared LED on the IR control. The infrared LED blinks at a certain frequency according to the address of the button pressed and sends the signal to the IR receiver. Usually this infrared LED has a flashing frequency of 38kHz. The frequency of the remote control used in this example was 38kHz.

Arduino

```
#include <IRremote.h> // You can download IR library from: https://github.com/z3t0/Arduino-IRremote
int RECV_PIN = 2;
IRrecv irrecv(RECV_PIN);
decode_results results;
#define CH1 0xFFA25D
#define CH 0xFF629D
#define CH2 0xFFE21D
#define PREV 0xFF22DD
#define NEXT 0xFF02FD
#define PLAYPAUSE 0xFFC23D
#define VOL1 0xFFE01F
#define VOL2 0xFFA857
#define EQ 0xFF906F
#define BUTTON0 0xFF6897
#define BUTTON100 0xFF9867
#define BUTTON200 0xFFB04F
#define BUTTON1 0xFF30CF
#define BUTTON2 0xFF18E7
#define BUTTON3 0xFF7A85
#define BUTTON4 0xFF10EF
#define BUTTON5 0xFF38C7
#define BUTTON6 0xFF5AA5
#define BUTTON7 0xFF42BD
#define BUTTON8 0xFF4AB5
#define BUTTON9 0xFF52AD
int red_led_pin = 9;
int green_led_pin = 10;
int blue_led_pin = 11;

void setup(){
  pinMode(red_led_pin, OUTPUT);
  pinMode(green_led_pin, OUTPUT);
  pinMode(blue_led_pin, OUTPUT);
  Serial.begin(9600);
  irrecv.enableIRIn();
}
void loop() {
  if (irrecv.decode(&results)){
    if (results.value == BUTTON2){
      digitalWrite(red_led_pin, !digitalRead(red_led_pin));
      if (digitalRead(red_led_pin) == HIGH){
        Serial.println("Red LED ON");
      }
      else{
        Serial.println("Red LED OFF");
      }
    }
    if (results.value == BUTTON3){
      digitalWrite(green_led_pin, !digitalRead(green_led_pin));
      if (digitalRead(green_led_pin) == HIGH){
        Serial.println("Green LED ON");
      }
```

```
      else{
        Serial.println("Green LED OFF");
      }
    }
    if (results.value == BUTTON4){
      digitalWrite(blue_led_pin, !digitalRead(blue_led_pin));
      if (digitalRead(blue_led_pin) == HIGH){
        Serial.println("Blue LED ON");
      }
      else{
        Serial.println("Blue LED OFF");
      }
    }
    if (results.value == BUTTON0){
      digitalWrite(red_led_pin, LOW);
      digitalWrite(green_led_pin, LOW);
      digitalWrite(blue_led_pin, LOW);
      Serial.println("LED OFF");
    }
    if (results.value == BUTTON5){
      digitalWrite(red_led_pin, HIGH);
      digitalWrite(green_led_pin, HIGH);
      digitalWrite(blue_led_pin, HIGH);
      Serial.println("LED ON White");
    }
    irrecv.resume();
  }
}
```

First of all, add to the code the "IRremote.h" library, which contains the functions required to use the IR module. Then, define the necessary variables. Then introduce the "receive "pin to the library.

The variables defined in this section are the addresses of the buttons on the controller. Not all buttons will be used in this example. Only use the buttons numbered 0, 1, 2, 3, 4 will be used.

In the "setup" section, set the LED pins as outputs. Start serial communication at 9600 communication speed and then start IR communication.

In the loop section, read the data coming from the controller with the "irrecv.decode(&results)" function and assign them to the "results" variable. Compare the "results" variable with the defined addresses. Turn on the red LED if the "results" variable is equal to" BUTTON1, we turn on the green LED if it is equal to "BUTTON2", and turn on the blue LED if it is equal to "BUTTON3". Turn off all LEDs if it is equal to "BUTTON0" and turn on all the LEDs and ensure RGB LED to be white if it is equal to "BUTTON4".

14

Making a Digital Meter with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

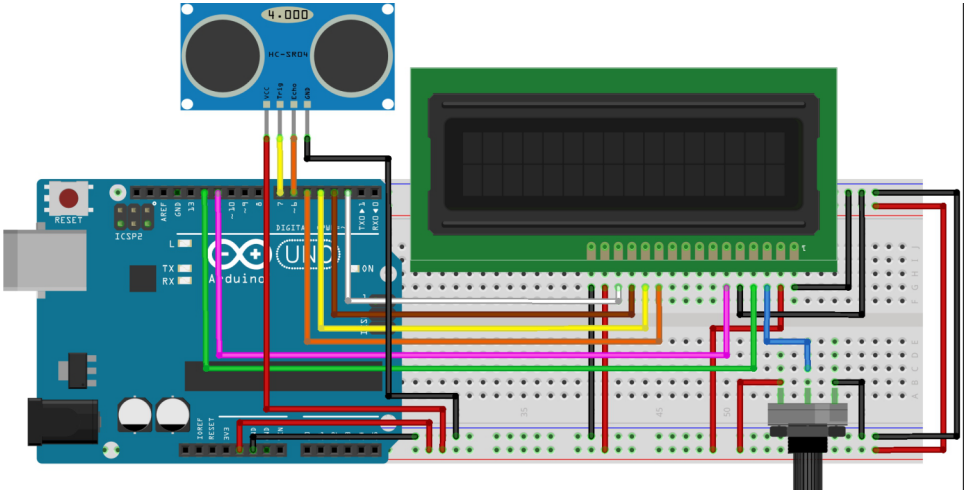
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 16x2 Character LCD
- HC-SR04 Ultrasonic Sensor
- 10k Ohm Potentiometer
- 4 Pcs Male-Female Jumper Wire
- 16 Pcs Male-Male Jumper Wire

The ultrasonic distance sensor is a device that can send a sound wave and detect the reflected sound wave. LCD is an element that displays characters on the screen according to the data provided. The LCD screen consists of 2 lines and can display 16 characters per line. Each character consists of 5x7 pixels.

The potentiometer is an adjustable resistor. In this circuit we used the potentiometer as a voltage divider. When the voltage is sent to two different resistors connected in series, voltages proportional to the resistance values on the resistors are obtained. And the potentiometer can be used as a voltage divider. When the potentiometer is turned, the voltage of the middle pin changes. This changing voltage allows adjusting the Contrast of the LCD screen. Let's continue with setting up the circuit.



```
#include <LiquidCrystal.h>
int trigPin = 7;
int echoPin = 6;
int pulsein_time;
int distance;
int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin,INPUT);
  lcd.begin(16, 2);
}
void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  pulsein_time = pulseIn(echoPin, HIGH, 11600);
  distance = pulsein_time*0.0345/2;
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Distance:");
  lcd.setCursor(0, 1);
  lcd.print(distance);
  lcd.print("cm");
}
```

You will take distance information from ultrasonic sensor in the code section and display it on LCD screen. The "LiquidCrystal.h" library is used to use the LCD. This library contains the functions necessary to use the LCD. Add the LCD library to the code. Then define the necessary variables. Define the LCD pins and make necessary pin settings.

In the "setup" section, define the echo pin as the input and the trig pin as the output. Then, set the LCD row-column length with "lcd.begin()" function.

In the "loop" section, first make the ultrasonic sensor ready for measurement by bringing the trig pin to the LOW level. Then the trig pin is set to HIGH and then to LOW to send the sound wave.

Measure the total round trip time of the sound wave with the "pulseIn()" function. The round duration is multiplied by 0.0345 to calculate the distance. The number 0.0345 is the distance the sound wave takes in 1 microsecond. The calculated distance value must be divided into two, as the sound wave travels back and forth. In the parking sensor construction application, the measured value was divided by "58.2". There is no difference between these two processes. With the "lcd.clear()" function, the remaining text is deleted from the screen. With the "lcd.setCursor()" function is used to define the line and column of the text to be displayed on the LCD screen. Finally, distance value is displayed on the LCD with the "lcd.print()" function.

15

Servo Motor Control with Motion Sensor (PIR)

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

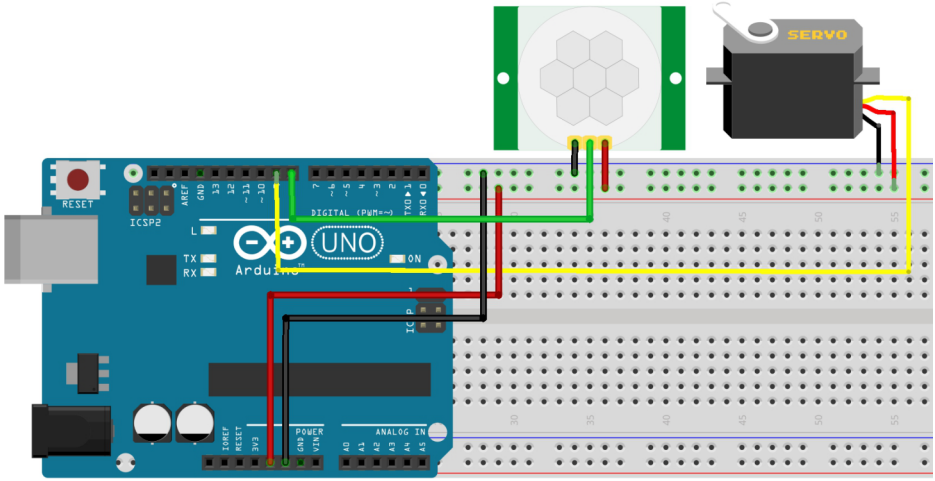
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Servo Motor
- Motion Sensor
- 5 Pcs Male-Male Jumper Wire
- 3 Male-Female Jumper Wire

In this example, you will use a motion detector to set up a system where you can move the motor when motion is detected in the environment.



The motion sensor works by detecting infrared light waves in the environment. Materials naturally emit infrared waves because of heat. The motion sensor detects the changes in the infrared waves emitted by the materials and produces the signal output.

In this sample code, you will read the data from the motion detector and ensure the movement of the servo motor when a motion is detected in the environment.

```arduino
#include <Servo.h>

int pirPin = 8;
int servoPin = 9;
int motion_state;
Servo motor;

void setup() {
  motor.attach(servoPin);
  pinMode(pirPin, INPUT);

}

void loop() {
  motion_state = digitalRead(pirPin);

  if(motion_state == HIGH){
    motor.write(150);
    delay(250);
    motor.write(30);
    delay(250);
    motor.write(150);
    delay(250);
    motor.write(30);
    delay(250);
    motor.write(150);
    delay(250);
    motor.write(30);
    delay(250);
    motor.write(90);
  }
  else{
    motor.write(90);
  }
}
```

First, add the servo motor library "Servo.h" to the code. Then, make the necessary pin definitions. Then define a variable called motion to store the data received from the sensor. And define a servo motor called "motor". The "motor" variable is a variable that allows controlling the servo motor.

In the "setup" section, associate the pin with which the servo motor is connected to the motor variable. Then set the pin to which the sensor is connected as input.

In the "loop" section, read the motion data from the sensor with the "digitalRead ()" function. Send the required angle command to the servo motor according to the value of the motion pin with "if-else" structure.

16

RGB LED Control with Bluetooth

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

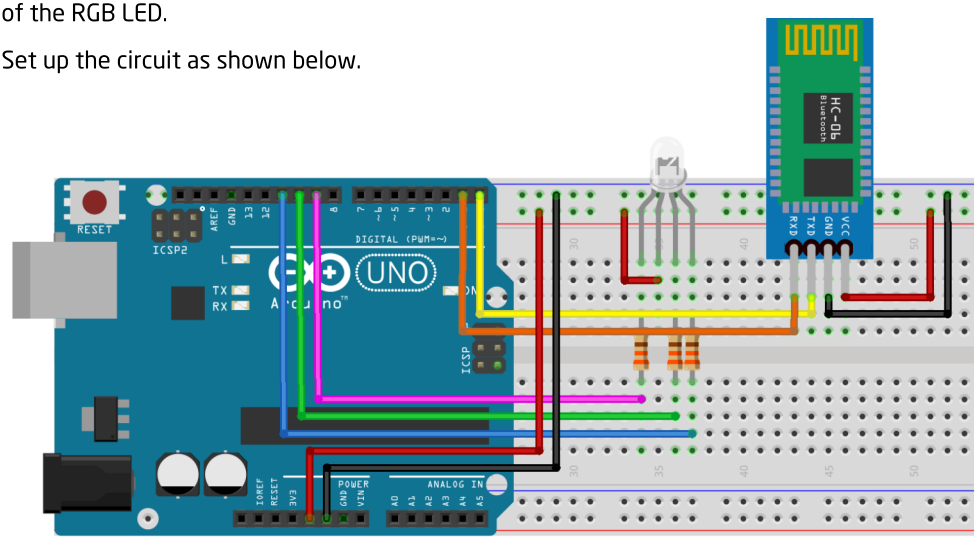
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- 9 Pcs Male-Male Jumper Wire
- Bluetooth module
- RGB LED
- 330 Ohm Resistor (Amber - Amber - Brown)

In this example, you will receive the data sent via the Bluetooth module and check the color of the RGB LED.

Set up the circuit as shown below.



RGB is a term formed by combining the initials of Red, Green and Blue. RGB led is a component consisted of the combination of 3 LEDs of main colors. It allows the creation of color combinations by giving different voltages to the LEDs of different colors inside. There are two kinds of RGB LED types. These are the common anode and common cathode. The common anode RGB LED is made by combining the positive pins of the 3 LEDs inside. The common cathode RGB LED is made by combining the negative pins of the 3 LEDs inside. In this example, common anode RGB LED was used.

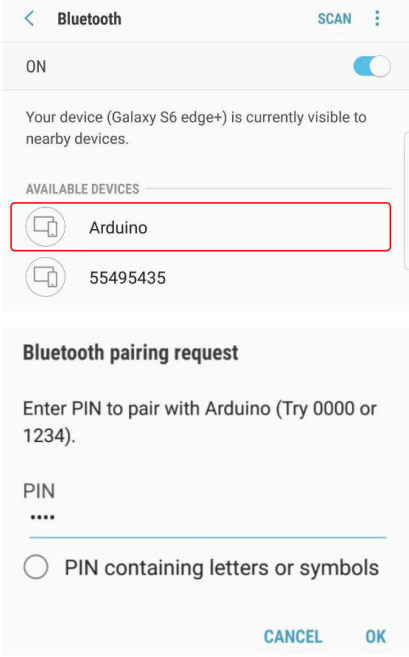
Arduino uses the UART (Universal Asynchronous Receiver Transmitter) protocol to communicate with the Bluetooth module. This is a serial communication protocol. In order to use the UART protocol, the Baud Rate (Communication speed) on both sides must be the same. 4800, 9600, 57600, 115200 are the most commonly used communication speeds. In this example, we will use 9600 Baud Rate. The Bluetooth module uses this communication speed by default.

```arduino
int received_data;
int red_pin = 9;
int green_pin = 10;
int blue_pin = 11;
void setup() {
  Serial.begin(9600);
  pinMode(red_pin,OUTPUT);
  pinMode(green_pin,OUTPUT);
  pinMode(blue_pin,OUTPUT);
}
void loop() {
  if(Serial.available()>0){
    received_data = Serial.read();
  }
  if(received_data == 'k'){
    digitalWrite(red_pin,LOW);
    digitalWrite(green_pin,HIGH);
    digitalWrite(blue_pin,HIGH);
  }
  else if(received_data == 'y'){
    digitalWrite(red_pin,HIGH);
    digitalWrite(green_pin,LOW);
    digitalWrite(blue_pin,HIGH);
  }
  else if(received_data == 'm'){
    digitalWrite(red_pin,HIGH);
    digitalWrite(green_pin,HIGH);
    digitalWrite(blue_pin,LOW);
  }
  else{
    digitalWrite(red_pin,HIGH);
    digitalWrite(green_pin,HIGH);
    digitalWrite(blue_pin,HIGH);
  }
}
```

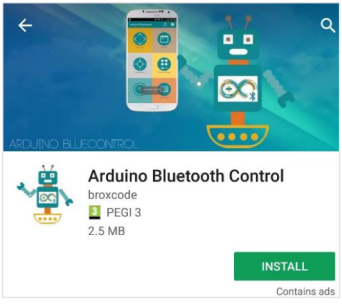
First, define the variables to use.

Start serial communication in the “setup” section. Then define output pins for RGB LEDs.

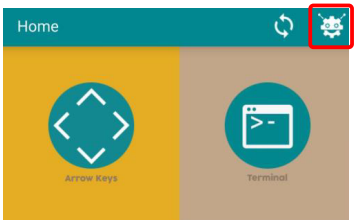
In the ”loop” section, read the data received through serial communication and equalize it to the variable named data. With “if else,“ else ”and“ if-else ”structures, turn on the RGB LED according to the value of the data variable. After installing the code to Arduino, you can do RGB LED control with an Arduino Bluetooth terminal application that you will install on our phone.



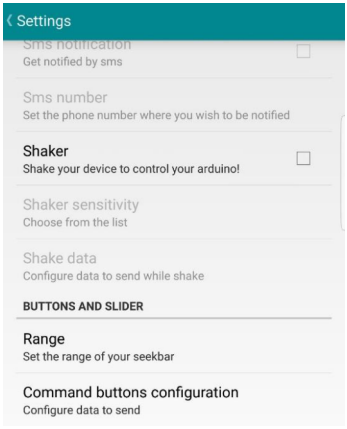
To connect to Bluetooth from your phone, first, you need to activate Bluetooth. Then enter the Bluetooth settings. A screen similar to the picture on the left should appear. This screen may appear differently on different phones. A device with the name defined when setting up Bluetooth appears on the screen. Click on the device and select match. In the Arduino code, enter the password defined in the settings section and make the connection.



Downloading the application called Arduino Bluetooth Control to your phone via Google Play. With this application you will send data to the Bluetooth module.

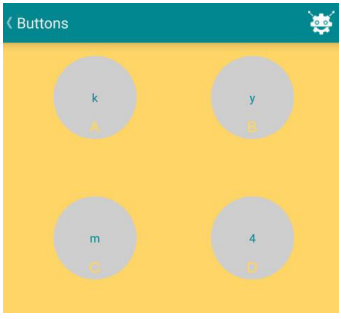


Then, enter the Arduino Bluetooth Control application that you have installed on your phone and click the settings button at the top right.



In the settings, click on the "Command buttons configuration" tab and define the data it will send via Bluetooth when you click the buttons. Click Button A and type "k" on the screen that appears. Write "y" for Button B and "m" for Button C.

Then leave the settings section back to the main screen. Click on the Buttons & Slider tab on the main screen to open the buttons to control the RGB LED.



Click on the A, B and C buttons to send the necessary commands to the Arduino via Bluetooth.

When you press "k", the RGB led is red; When you press "y", it is green; when you press the "m, it is blue.

17

Making a Digital Clock with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

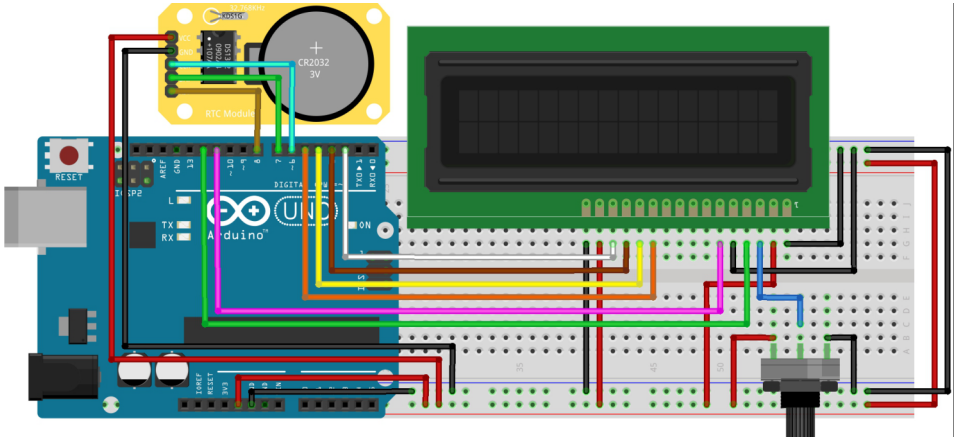
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Gerekli malzemeler:

- Arduino Uno
- Breadboard
- DS1302 RTC Module
- 16x2 LCD Screen
- 10k Ohm Potentiometer
- 17 Pcs Male-Male Jumper Wire
- 5 Pcs Male-Female Jumper Wir

When you try to make a clock on devices like Arduino, the biggest problem you will face is that in case of a power failure, Arduino stops counting the time. This causes deviations in time and prevents you from learning the right time. RTC modules are built to keep time synchronous. This module, which can operate even with very little power, counts with the crystal on it without deviation in time for many years thanks to its battery. This crystal produces 32000 signals per second. The RTC reads these signals and counts one second forward in every 32000 steps.

In order to use the watch, you must first adjust the time to the current time. Therefore, you need to install our setting code first.



```arduino
#include <virtuabotixRTC.h>
int CLK_PIN = 6;
int DAT_PIN = 7;
int RST_PIN = 8;
virtuabotixRTC myRTC(CLK_PIN, DAT_PIN, RST_PIN);
void setup() {
 Serial.begin(9600);
 myRTC.setDS1302Time(10, 10, 14, 4, 13, 9, 2018);
}
void loop() {
 myRTC.updateTime();
 Serial.print("Tarih / Saat: ");
 Serial.print(myRTC.dayofmonth);
 Serial.print("/");
 Serial.print(myRTC.month);
 Serial.print("/");
 Serial.print(myRTC.year);
 Serial.print(" ");
 Serial.print(myRTC.hours);
 Serial.print(":");
 Serial.print(myRTC.minutes);
 Serial.print(":");
 Serial.println(myRTC.seconds);
 delay(1000);
}
```

Define the necessary variables. Determine which RTC pins you will assign to the Arduino pins you will use. In the “setup” section, start serial communication at 9600 communication speed. Then set the time as seconds, minutes, hours, day of the week, day of the month, month and year. In the loop section, read the time from the RTC and print it to the serial port screen. Check the readings on the serial port screen. After making sure there are no errors, you can start the display code on the LCD.

```
#include <LiquidCrystal.h>
#include <virtuabotixRTC.h>
int CLK_PIN = 6;
int DAT_PIN = 7;
int RST_PIN = 8;
virtuabotixRTC myRTC(CLK_PIN, DAT_PIN, RST_PIN);
int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
void setup() {
  lcd.begin(16, 2);
}
void loop() {
  myRTC.updateTime();
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(myRTC.dayofmonth);
  lcd.print("/");
  lcd.print(myRTC.month);
  lcd.print("/");
  lcd.print(myRTC.year);
  lcd.setCursor(0, 1);
  lcd.print(myRTC.hours);
  lcd.print(":");
  lcd.print(myRTC.minutes);
  lcd.print(":");
  lcd.print(myRTC.seconds);
  delay(1000);
}
```

You can download this library from: https://github.com/chrisfryer78/ArduinoRTClibrary

First, include the necessary libraries in the code. Then assign the pins to the variables. Introduce the assigned variables to the library.

In setup() section, adjust the aspect ratio of the LCD. In this example, since we used a 16x2 LCD, we set the aspect ratio to 2 by 16.

In loop() section, first read the time data from the RTC. Then clear the characters on the LCD screen. Otherwise the characters may overlap, leading to obtain incorrect data. Starting from the first column and the first row, print the date as day, month, and year. From the first column of the second row, print the time in hours, minutes, seconds and add a waiting time of 1 second.

18

Using Soil Moisture with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

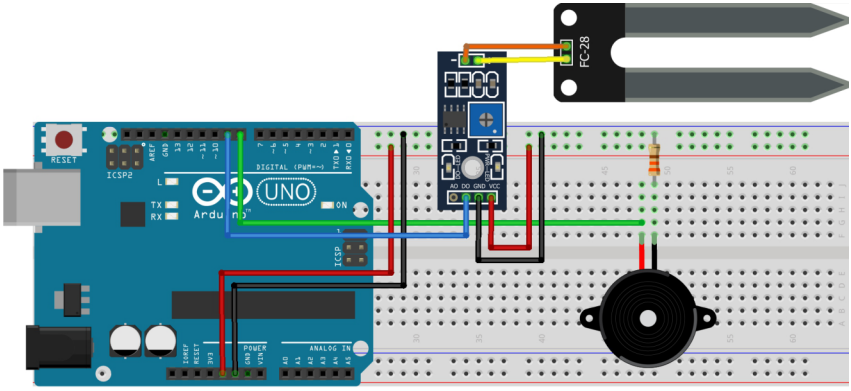
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Soil Moisture Sensor
- 3 Pcs Male-Male Jumper Wire
- 3 Pcs Male-Female Jumper Wire
- Buzzer
- 1 Pcs 330 Ohm Resistor (Amber - Amber - Brown)

The soil moisture sensor consists of two electrodes. It measures the conductivity between the electrodes and gives information about how moist the soil is. If the soil is moist, the conductivity between the electrodes increases. Since the resistance decreases as the conductivity increases, less voltage starts to come from the voltage divider inside the sensor. If the soil is dry, a higher voltage is obtained as the resistance between electrodes will increase, and this data is processed analogously to learn the amount of moisture in the soil.



In this example, when the amount of moisture in the soil exceeds a certain threshold, it sends a signal and the buzzer on the circuit starts to make sound. In this circuit, we set the threshold value with the potentiometer on the sensor. Therefore, if the threshold exceeds the value set, it will give 0V output from pin D0. Likewise, you can also prepare analogue data from pin A0 in the range of 0V-5V to operate when a certain threshold value is exceeded in the code.

In the code, you will read the data from the soil moisture sensor and ensure that the Arduino sounds alarm with the buzzer if the soil moisture decreases.

```
int sensorPin = 9;
int buzzerPin = 8;
int data;

void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
}
void loop() {
  data = digitalRead(sensorPin);
  if(data == true){
    digitalWrite(buzzerPin, HIGH);
    delay(100);
    digitalWrite(buzzerPin, LOW);
    delay(100);
  }
  else{
    digitalWrite(buzzerPin, LOW);
  }
}
```

First, define the necessary variables. Make input-output settings in the "setup" section. In the "loop" section, the value received from the sensor is equalized to the data variable with the "digitalRead ()" function. The status of the data variable is then checked by the "if-else" structure.

When the soil moisture sensor detects moisture, it gives 0V signal from digital output pin. The data variable LOW indicates that moisture has been detected by the soil moisture sensor. In this case, the data variable takes the value HIGH and the commands inside the else structure are executed.

When no moisture is detected by the soil moisture sensor, 5V output is given from the digital output pin of the sensor. In this case, the buzzer starts to make sound.

19

Using Rain Sensor with Arduino.

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

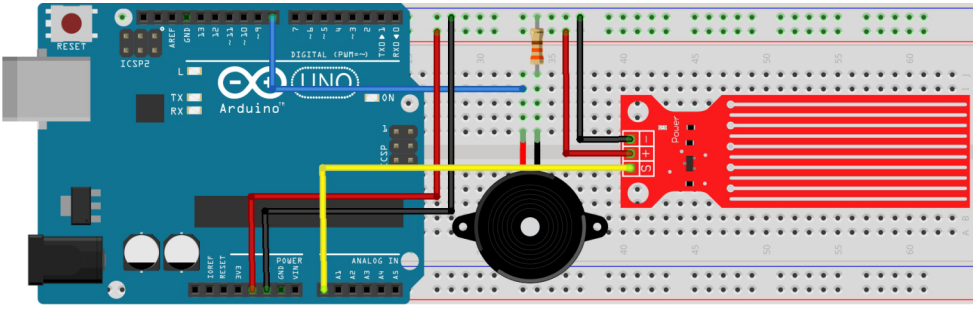
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Rain Sensor
- 3 Pcs Male-Male Jumper Wire
- 3 Pcs Male-Female Jumper Wire
- Buzzer
- 1 Pcs 330 Ohm Resistor (Amber - Amber - Brown)

If the weather is rainy, when the amount of water above the rain sensor exceeds a certain threshold, it will send a signal and the buzzer on the circuit will start to make sound. We prepared this circuit to give an alarm according to the signal received via the analog pin.
The rain sensor consists of two electrodes. It informs us by measuring the conductivity between the electrodes. Water drops on the sensor increase the conductivity between the electrodes. The sensor sends this data as analog and digital outputs.
You will read the data from the rain sensor in the code and ensure that it will sound alarm with the buzzer when it rains.



```
int sensorPin = A0;
int threshold_value = 100;
int buzzerPin = 8;
int data;

void setup() {
  pinMode(buzzerPin, OUTPUT);
}
void loop() {
  data = analogRead(sensorPin);
  if(data > threshold_value){
    digitalWrite(buzzerPin, HIGH);
    delay(100);
    digitalWrite(buzzerPin, LOW);
    delay(100);
  }
  else{
    digitalWrite(buzzerPin, LOW);
  }
}
```

First, define necessary variables. In the “setup” section, define the pin to be connected to the buzzer as the output. In the “loop” section, the value obtained from the sensor is equalized to the data variable by the "analogRead()" function. The status of the data variable is then checked by the "if-else" structure.

If the data is greater than the threshold value read from the rain sensor, it indicates that the weather is rainy. If there is moisture on the sensor, the commands in “if" are executed and the alarm sounds.

If the data read from the rain sensor is below the threshold level, it means that the weather is dry. If the weather is dry, the commands in "else" are executed and the alarm does not sound.

29

Using Gas Sensor with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

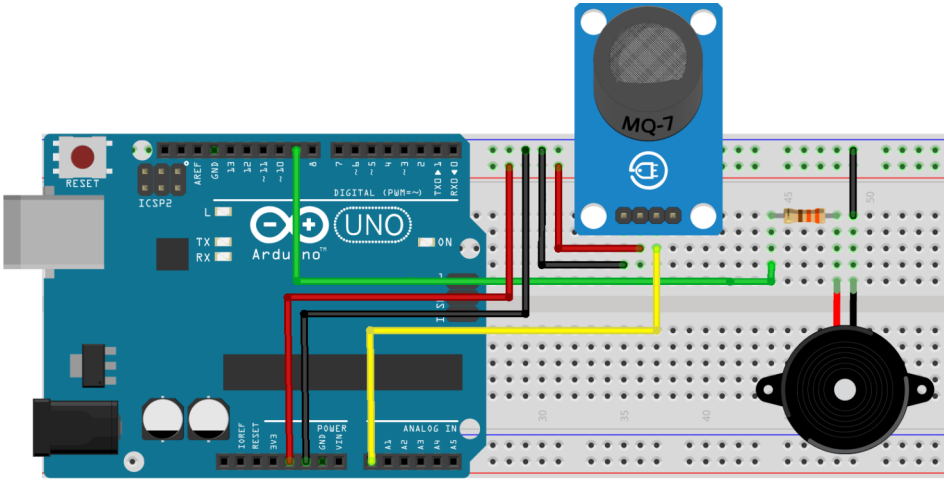
You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Gas Sensor
- Buzzer
- 7 Pcs Male-Male Jumper Wire
- 1 Pcs 330 Ohm Resistor (Amber - Amber - Brown)

In this circuit, we made an alarm with gas sensor. When the amount of natural gas exceeds a certain threshold, it will send a signal and the buzzer on the circuit will start to sound. Similarly, it can be done by determining the threshold value with the potentiometer on the sensor and receiving digital data from pin D0.

There are a number of resistors inside the gas sensor. The gas in the environment interacts with the resistors inside the sensor and changes the resistance values. By means of a voltage divider, the sensor gives different voltage values according to the gas density and, the amount of gas in the environment is measured with analog reading of these voltage values.



The gas sensor can provide analog and digital outputs. In this application, you will set up the sample circuit using the analog output.

In our sample code, you will read the value from the analog pin of the gas sensor and compare it with the threshold value. When the value you read is greater than the threshold value in the code, it will sound an alarm with the help of buzzer.

```
Arduino
1  int threshold_value = 400;
2  int buzzerPin = 9;
3  int value;
4
5  void setup() {
6    pinMode(buzzerPin, OUTPUT);
7  }
8
9  void loop() {
10   value = analogRead(A0);
11   if(value > threshold_value){
12     digitalWrite(buzzerPin, HIGH);
13     delay(100);
14     digitalWrite(buzzerPin, LOW);
15     delay(100);
16   }
17   else{
18     digitalWrite(buzzerPin, LOW);
19   }
20 }
```

First, define the necessary variables. The threshold value of the sensor can be adjusted by changing the value of the variable “thresholdValue”. In the “setup” section, set the pin to which the buzzer is connected as the output. In the “loop” section, read the analog value from the sensor with the “analogRead()” function and equalize it to the variable named ”value”. Then, using the if-else structure, compare the variable named “value” with “thresholdValue”. When the analog data that read from the sensor is greater than the threshold variable, open and close the buzzer pin to generate an alarm with the buzzer. When the data read from the sensor is below the threshold value, give LOW value to the buzzer pin so that the buzzer does not sound the alarm.

21

Using RFID Sensor with Arduino

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

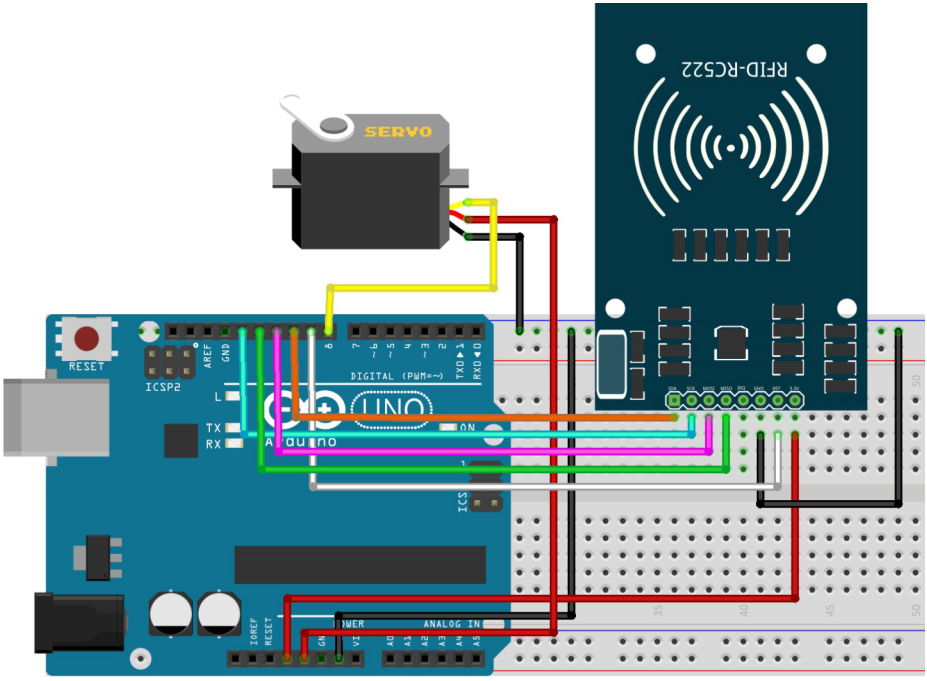
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- Servo Motor
- RC522 RFID module
- RFID Card
- 12 Pcs Male-Male Jumper Wire

In this application, the RFID card reader will read the ID number of the card and sent it to Arduino via SPI (Serial Perhiperal Interface) protocol. If the ID number is registered in the system, it activates the servo and opens the door. If it is not registered, the door remains closed.



SPI is a serial communication protocol based on Master-Slave logic. That is, they need a clock signal to work synchronously with each other. In this way, communication is provided more reliably than asynchronous protocols such as UART. For SPI, you will need at least 4 pins. SCK is used for the clock signal. MOSI (Master Out Slave In) is used to send data from the master device to the slave. MISO (Master In Slave Out) is used to send data from the slave device to the master. The SS (Slave Select) pin determines which device the master device communicates with. The RC522 RFID module used in this application also communicates via the SPI protocol.

Now, let's write the code.

```
#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>

int RST_PIN = 9;
int SS_PIN = 10;
int servoPin = 8;

Servo motor;
MFRC522 rfid(SS_PIN, RST_PIN);
byte ID[4] = {97, 76, 67, 9};

void setup() {
  motor.attach(servoPin);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
}

void loop() {

  if ( ! rfid.PICC_IsNewCardPresent())
    return;

  if ( ! rfid.PICC_ReadCardSerial())
    return;
```

```
Arduino
27
28   if (rfid.uid.uidByte[0] == ID[0] &&
29     rfid.uid.uidByte[1] == ID[1] &&
30     rfid.uid.uidByte[2] == ID[2] &&
31     rfid.uid.uidByte[3] == ID[3] ) {
32         Serial.println("Door is OPEN");
33         Print_Screen();
34         motor.write(180);
35         delay(3000);
36         motor.write(0);
37         delay(1000);
38     }
39     else{
40       Serial.println("Unauthorized card");
41       Print_Screen();
42     }
43   rfid.PICC_HaltA();
44 }
45 void Print_Screen(){
46   Serial.print("ID Number: ");
47   for(int sayac = 0; sayac < 4; sayac++){
48     Serial.print(rfid.uid.uidByte[sayac]);
49     Serial.print(" ");
50   }
51   Serial.println("");
52 }
```

First, define the libraries that you will use in the code. The MFRC522 library contains the functions required to use the RC522 module.

RST_PIN and SS_PIN variables are the pins used by the RC522 module. The servoPin variable is the pin to which the servo motor is connected.

Then, define the “Servo” variable named motor. After making the necessary settings for the RFID module, create an array called “ID”. The elements of this array are the ID number of the RFID card you want to register. After writing all the code, you can change the ”ID” variable by reading your own card through serial port. In this way, you can authorize access to any card.

In the “setup” section, the pin to which the servo motor is connected is associated with the motor variable. Then, start the serial communication and SPI. The RC522 module is started with the “rfid.PCD_Init()” function.

In the loop section, wait until a new card is read from the RC522 module. When the new card is read, with the "if-else" structure, compare the registered card number and the card number read. If the read card is authorized, servo motor movement is provided and the card number is printed to the serial port with "Printtoscreen()" function. This function will be explained in the next chapter. If the read card number is not authorized, an unauthorized card warning is printed on the serial port.

Parts that are repeated too often in the code are converted to functions. In this way, the complexity of the code is minimized so that the memory of the processor is not used unnecessarily.

The "printtoscreen()" function is created by us. Where this function is used, commands in the function are executed. This function allows the read RFID card information to be written to the serial port.

Temperature and Humidity Measurement with ESP8266

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

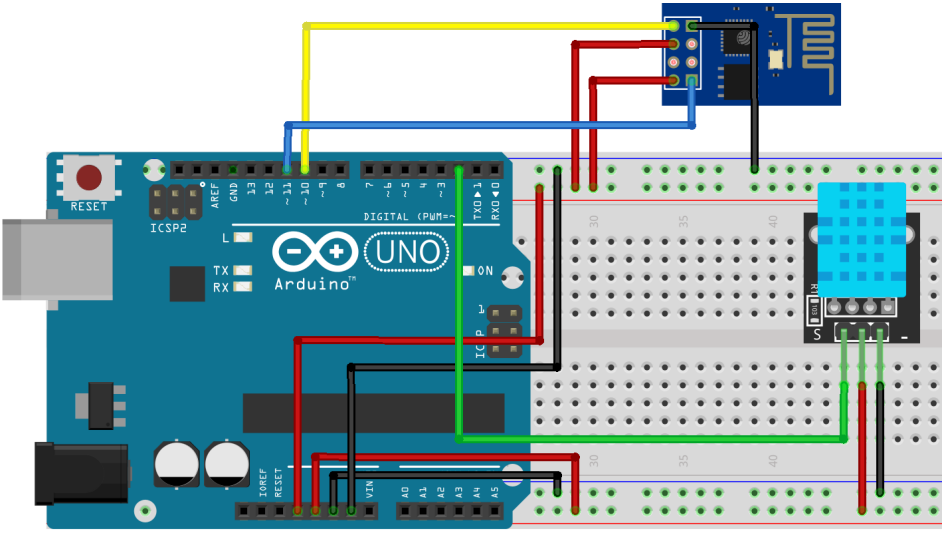
Materials Required:

- Arduino Uno
- Breadboard
- ESP8266 WiFi Module
- DHT11 Humidity and Temperature Sensor
- 8 Pcs Male-Female Jumper Wire
- 3 Pcs Male-Male Jumper Wire

In this application, you will send the temperature and humidity data received via DHT11 to Thingspeak platform using ESP8266 WiFi module.

The UART protocol is used when communicating with ESP8266. In this example, we will use the 115200 Baud rate (Communication speed).

Thingspeak is an open source IoT (Internet of Things) application. Users send data to the site via HTTP and make their own applications visually better and easy to understand thanks to the graphical interfaces on the site. In this example, time-humidity and time-temperature graphs will be generated in Thingspeak with data from the DHT11 sensor.



ESP8266, thanks to the wireless communication circuit on the ethernet protocol, allows us to connect to the wireless internet. Devices such as Arduino do not have any hardware that allows them to use the ethernet protocol. Therefore, modules are utilized to use the ethernet protocol. These modules convert the ethernet protocol into communication protocols that provide easier communication. ESP8266 is an example of these modules. It acts as an interpreter, which simplifies and converts the Ethernet protocol into a UART protocol.

The DHT11 is a sensor for reading humidity and temperature data. As with the sensors used before, this sensor allows us to measure the humidity and temperature of the air by measuring the conductivity. There is an NTC (Negative Temperature Coefient) inside the sensor. NTC is a kind of resistor, and as the ambient temperature increases, its conductivity increases and the resistance value decreases. There are 2 electrodes inside the sensor and a surface that holds the moisture in the air between the electrodes. This surface changes the conductivity between the electrodes as the amount of moisture in the air increases. In this way, the humidity level in the air can be measured.

Arduino

```
#include <SoftwareSerial.h>
#include <dht11.h>
String Wifi_Name = "Your Wifi Name";
String password = "Wifi Password";
int rxPin = 10;
int txPin = 11;
int dht11Pin = 2;
String ip = "184.106.153.149";
float temp, huminity;
dht11 DHT11;
SoftwareSerial esp(rxPin, txPin);
void setup() {
  Serial.begin(9600);
  esp.begin(115200);
  esp.println("AT");
  while(!esp.find("OK")){
    esp.println("AT");
    Serial.println("No ESP8266 found.");
  }
  esp.println("AT+CWMODE=1");
  while(!esp.find("OK")){
    esp.println("AT+CWMODE=1");
  }
  Serial.println("Connecting Wifi");
```

```
Arduino
  esp.println("AT+CWJAP=\""+Wifi_Name+"\",\""+password+"\"");
  while(!esp.find("OK"));
  Serial.println("Connected Wifi");
  delay(1000);
}
void loop() {
  esp.println("AT+CIPSTART=\"TCP\",\""+ip+"\",80");
  if(esp.find("Error")){
    Serial.println("AT+CIPSTART Error");
  }
  DHT11.read(dht11Pin);
  temp = (float)DHT11.temperature;
  huminity = (float)DHT11.humidity;
  String data = "GET https://api.thingspeak.com/update?api_key=2F55993RWVDCTSUS'
  data += "&field1=";
  data += String(temp);
  data += "&field2=";
  data += String(huminity);
  data += "\r\n\r\n";
  esp.print("AT+CIPSEND=");
  esp.println(data.length()+2);
  delay(2000);
  if(esp.find(">")){
    esp.print(data);
    Serial.println(data);
    Serial.println("Data Sent");
    delay(1000);
  }
  Serial.println("Connection closed");
  esp.println("AT+CIPCLOSE");
  delay(1000);
}
```

The ESP8266 module communicates with Arduino via serial communication.

This communication can be provided with Arduino Uno pins 0 and 1, however, it can be done on software basis from other pins with the SoftwareSerial library.

In this example, you will communicate with the ESP8266 module using the SoftwareSerial library.

First, add the SoftwareSerial library to the code. Then add the “dht11.h” library of the temperature and humidity sensor to be used.

Keep the name and password of the network to be connected as variable in memory.

Change the “agName” and “agPassword” variables according to the network name and password.

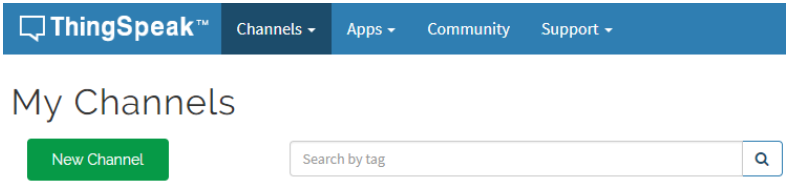
Create a variable for the RX and TX pins to which you will connect the ESP8266 module.

Then, create a variable for the pin to which you will connect the DHT11 sensor.

Create the variable called “ip“ where you will keep the IP number of the Thingspeak platform that will be used in the application. With DHT11, create ”temperature” and ”humidity” variables to keep the temperature and humidity data to be read. Since these data will be consisted of comma seperated values, create them in ”float” type. The ”float” is a variable where you can store decimal numbers. Set the pins to which you will connect the ESP8266 module. In the “setup” section, start the communication with the computer. Use 9600 for communication speed with the computer. Then, start the communication with the ESP8266 module with the “esp.begin” function. The speed of serial communication must be the same with the communication speed of the ESP8266 module. That’s why we used 115200. If the sample code does not work, you can correct this problem by updating the firmware of ESP8266 module. You can find detailed information about the firmware update on:
https://maker.robotistan.com/esp8266-ile-iot-dersleri-1-esp8266-modulunu-guncelleme/
Send an AT command via serial communication to see if the ESP8266 module is ready. Wait until the ESP8266 module is ready. Set the module as “client by sending “AT+CWMODE=1” command. Then, connect to the network with “AT+CWJAP” command. “

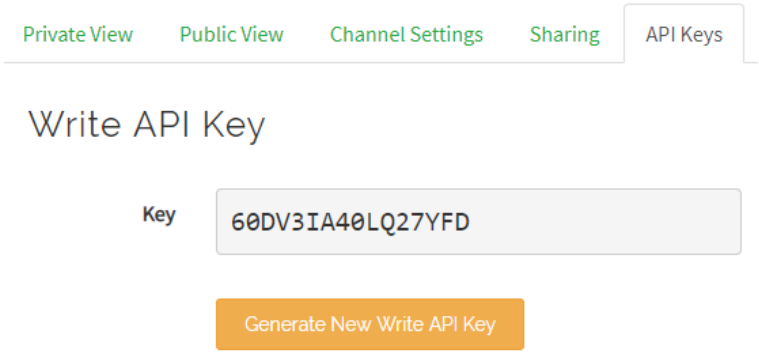
In the “loop” section, connect to Thingspeak with the ”AT+CIPSTART” command. Check whether there is an error in connection setup. Then, read the temperature and humidity values with the DHT11 sensor. Load the readings into variables named “temperature” and “humidity”.
Create the command named data to be sent to Thingspeak. Write your own key in the “key” section here. It will be explained later how to learn your own key.
Create the data and send the data length to the ESP8266 module with the “AT+CIPSEND” command.
The ESP8266 module sends the “>” icon when ready. Wait for this icon to appear. When the icon appears, send the connection data to the module. Terminate the connection with “AT+CIPCLOSE” command when the data transmission is completed.

In this sample application, you will use the Thingspeak platform to display data on the Internet. Sign up to Thingspeak platform from https://www.thingspeak.com .



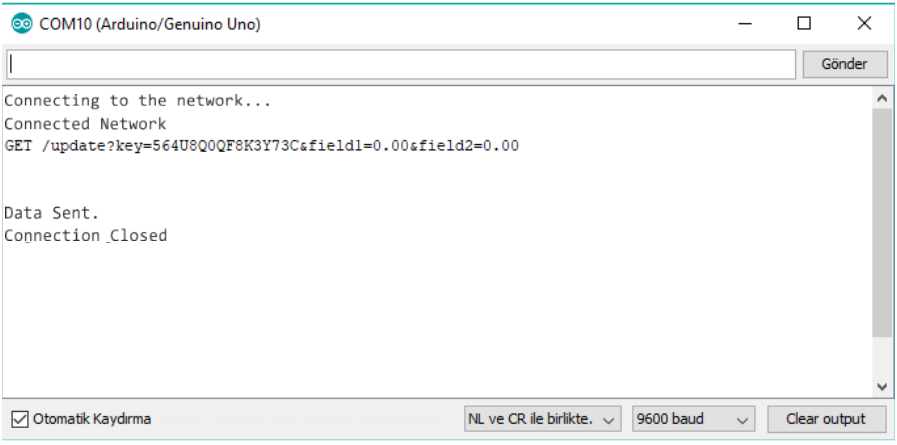
After signing up, enter "My Channels" section under "Channels" tab. Then click "New Channel" button.

Make the necessary settings on the incoming page as shown above. Then click "Save Channel" button at the bottom of the page to create your new channel.

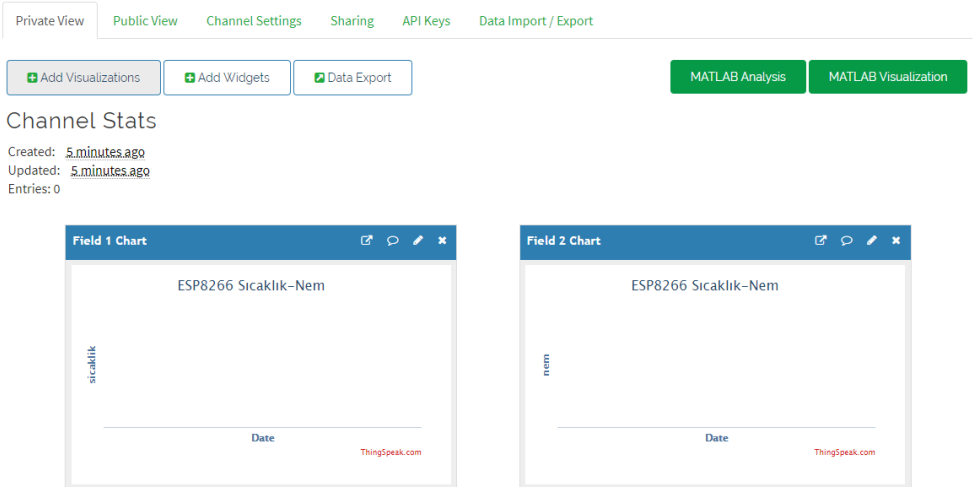


The "api key" is required for Arduino to communicate with Thingspeak. Arduino connects to your Thingspeak account with the "api key" and saves the data to your channel. Click on the "Api Keys" tab to access the "Api key". Copy the "Key" in the "Write API Key" pane on the incoming page and paste it to the required place in the sample Arduino code.

After installing the code in Arduino, open the serial port via Arduino IDE.



After setting the serial port speed to 9600, display the results coming from the Arduino. After the message sent text appears in the serial port, view the data via thingspeak.



By clicking the "Private View" tab, you can also view the data sent by Arduino on graphics.

23

Step Motor Control with ESP2866

robotistan BLOG

You can access the blog post of the application from the link below.
http://bit.ly/arduinodersleri

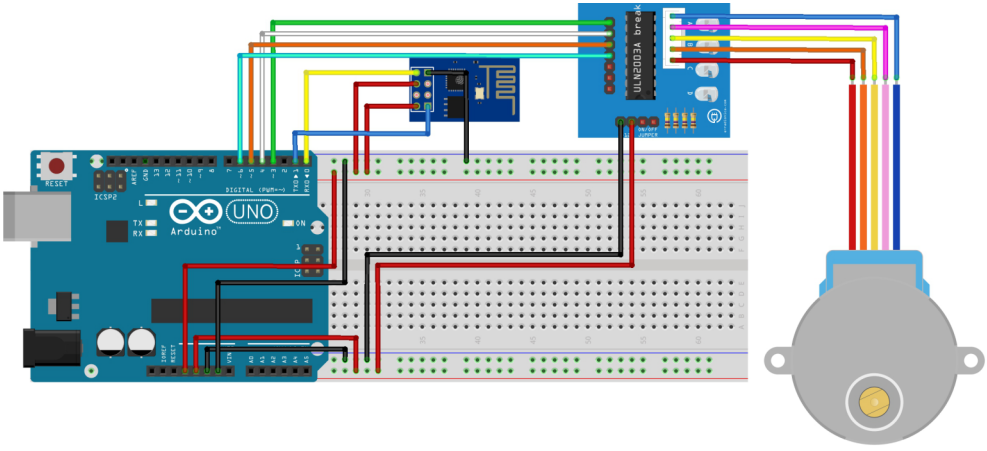
YouTube

You can access the video of the application from the link below.
http://bit.ly/arduinovideodersler

Materials Required:

- Arduino Uno
- Breadboard
- ESP8266 WiFi Module
- Step Motor
- ULN2003A Motor Driver
- 11 Pcs Male-Female Jumper Wire
- 2 Pcs Male-Male Jumper Wire

In this application, you will control the step motor via the WEB Server opened using ESP8266. Step motors are electromechanical devices that convert electrical energy into physical energy with rotary motion. As the name suggests, these motors run step by step. They continue their movement by opening and closing of the coils in order. In this example, the step motor used was 4-phase with 4 coils inside. Step motors are controlled by very high speed switching motor drives and motor control boards. The step motor driver we used drives the motor according to the signals it receives from Arduino.



First, define the variables. Write “agName” and “agScrypter” variables according to your Wi-Fi name and password. Define the motor pins. These variables are the pins to which the step motor will be connected.

```
String Wifi_Name = "Your Wifi Name";
String Wifi_Password = "Wifi Password";
int motorPin1 = 3, motorPin2 = 4, motorPin3 = 5, motorPin4 = 6;

void setup(){
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
  Serial.begin(115200);
  Serial.println("AT");
  while(!Serial.find("OK")){
    Serial.println("AT");
  }
  delay(1000);
  Serial.println("AT+RST");
  delay(1000);
  while(!Serial.find("ready"))
  delay(1000);
  Serial.println("AT+CWMODE=1");
  while(!Serial.find("OK"));
  Serial.println("AT+CWJAP=\""+Wifi_Name+"\",\""+Wifi_Password+"\"");
  while(!Serial.find("OK"));
  Serial.print("AT+CIFSR\r\n");
  Serial.print(read_esp(1000));
  clear_serial(2000);
  Serial.print("AT+CIPMUX=1\r\n");
  clear_serial(2000);
  Serial.print("AT+CIPSERVER=1,80\r\n");
  clear_serial(2000);
}
```

In the “setup” section, set the LED pins as outputs. Then, start the serial communication. The ESP8266 module is connected via serial communication. The speed of serial communication must be the same with the communication speed of the ESP8266 module. If the sample code does not work, you can correct this problem by updating the firmware of ESP8266 module. You can find detailed information about the firmware update on: https://maker.robotistan.com/esp8266-ile-iot-dersleri-1-esp8266-modulunu-guncelleme/ .

Send an AT command via serial communication to see if the ESP8266 module is ready. Wait until the ESP8266 module is ready. Then, reset the module with "AT+RST" command. Wait until the reset process is completed.

Set the module as client by sending "AT+CWMODE=1" command. Then we connect to our network with "AT + CWJAP" command. After waiting to connect to the network, read the IP and MAC addresses with the "AT+CIFSR" command. Then print this data to the serial port.

Since Arduino's communication pins with the computer and the esp8266 module are on the same pins, prevent the communication from being interrupted with the "serialClean" function after printing data to the serial port. This function allows you to delete data that is not used in serial communication. This function will be explained later.

```
void loop(){
  if(Serial.available()){
    if(Serial.find("+IPD,")){
      delay(200);
      int connectionId = Serial.read() - 48;
      String command = read_esp(1000);
      if(command.indexOf("step=ileri") != -1){
        for(int adim = 0; adim < 5; adim++){
          stepForward(50);
        }
      }
      else if(command.indexOf("step=geri") != -1){
        for(int adim = 0; adim < 5; adim++){
          stepBackward(50);
        }
      }
      String page = "<h1>Step Motor Control</h1><br>";
      page+="<br><a href=\"?step=Forward\"><button><h1>Ileri</h1></button></a>";
      page+="<br><br><a href=\"?step=Backward\"><button><h1>Geri</h1></button></a>";
      command = "AT+CIPSEND=";
      command += connectionId;
      command += ",";
      command +=page.length();
      command +="\r\n";
      Serial.print(command);
      delay(1000);
      Serial.print(page);
      delay(1000);
      command = "AT+CIPCLOSE=";
      command+=connectionId;
      command+="\r\n";
      Serial.print(command);
    }
  }
}
```

With the "AT+CIPMUX=1" command, set the module to allow multiple connections. Then, create a server with the "AT+CIPSERVER=1" 80 command and start listening on port 80. The ESP8266 module transmits incoming connection requests via serial communication. In the "loop" section, check whether data comes from serial communication. The wait for the IPD text to appear. The data after this text is the communication number. Synchronize this number to the "connectionId" variable. You will use this number when sending page data and terminating the connection.

The connection data from the ESP module are received by using the "espRead" function. Then, with the "if-else if" structure, check whether there is a step motor control command in this data. To do this, use "indexOf" function in "if". This function searches the position of the given variable in an array of texts. It returns the -1 value if the given variable does not exist in this array.

After the command processing sections are completed, start transmitting the site data to the client. First, generate the site codes to be displayed on the client screen. These codes are written in html language. With "AT+CIPSEND "command, send to the ESP8266 module the length of these codes and which client they will go. Then, send the site commands to the module with "AT+CIPSEND" command. Finally, terminate the connection with "AT+CIPCLOSE" command.

```
Arduino
67 String read_esp (long int timeout){
68   long int start_time = millis();
69   String received;
70   while(millis() - start_time < timeout){
71     if(Serial.available()>0){
72         char c = Serial.read();
73         received += c;
74     }
75   }
76   received.replace("AT+","");
77   return received;
78 }
79 void clear_serial (long int timeout){
80   long int start_time = millis();
81   while(millis() - start_time < timeout){
82     if(Serial.available()>0){
83         Serial.read();
84     }
85   }
86 }
```

The "espRead" function allows reading of commands from esp via serial communication. It takes the variable named "timeOut" as input. Data is read in milliseconds for the time according to the data in the "timeOut" variable. First, the start value of the function is stored in the memory with the "start" variable. Then, with the "while()" structure, data is read by serial communication as long as the running time of the total function is less than the "timeOut" variable. This data is provided as function output.

The "serialClean" function works just like the "espRead" function, but it does not generate any value as output. In this way, unused serial communication bytes are deleted from the memory of Arduino.

```
Arduino
87  void stepForward(int delay_time){
88    digitalWrite(motorPin1, HIGH);
89    digitalWrite(motorPin2, LOW);
90    digitalWrite(motorPin3, LOW);
91    digitalWrite(motorPin4, LOW);
92    delay(delay_time);
93    digitalWrite(motorPin1, LOW);
94    digitalWrite(motorPin2, HIGH);
95    digitalWrite(motorPin3, LOW);
96    digitalWrite(motorPin4, LOW);
97    delay(delay_time);
98    digitalWrite(motorPin1, LOW);
99    digitalWrite(motorPin2, LOW);
100   digitalWrite(motorPin3, HIGH);
101   digitalWrite(motorPin4, LOW);
102   delay(delay_time);
103   digitalWrite(motorPin1, LOW);
104   digitalWrite(motorPin2, LOW);
105   digitalWrite(motorPin3, LOW);
106   digitalWrite(motorPin4, HIGH);
107   delay(delay_time);
108 }
```

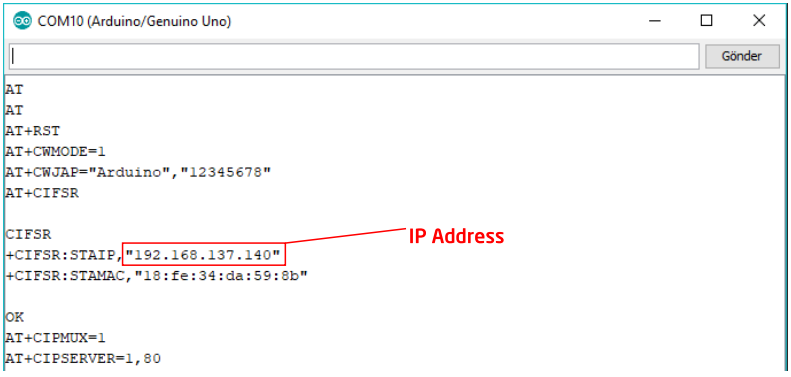
The "stepForward()" function takes the "waitTime" variable as an input. It gives HIGH value to the pins of the step motor in order to move the steper motor forward. The time between 2 steps is set with the "waitTime" variable.

```
Arduino
109 void stepBackward(int delay_time){
110   digitalWrite(motorPin1, LOW);
111   digitalWrite(motorPin2, LOW);
112   digitalWrite(motorPin3, LOW);
113   digitalWrite(motorPin4, HIGH);
114   delay(delay_time);
115   digitalWrite(motorPin1, LOW);
116   digitalWrite(motorPin2, LOW);
117   digitalWrite(motorPin3, HIGH);
118   digitalWrite(motorPin4, LOW);
119   delay(delay_time);
120   digitalWrite(motorPin1, LOW);
121   digitalWrite(motorPin2, HIGH);
122   digitalWrite(motorPin3, LOW);
123   digitalWrite(motorPin4, LOW);
124   delay(delay_time);
125   digitalWrite(motorPin1, HIGH);
126   digitalWrite(motorPin2, LOW);
127   digitalWrite(motorPin3, LOW);
128   digitalWrite(motorPin4, LOW);
129   delay(delay_time);
130 }
```

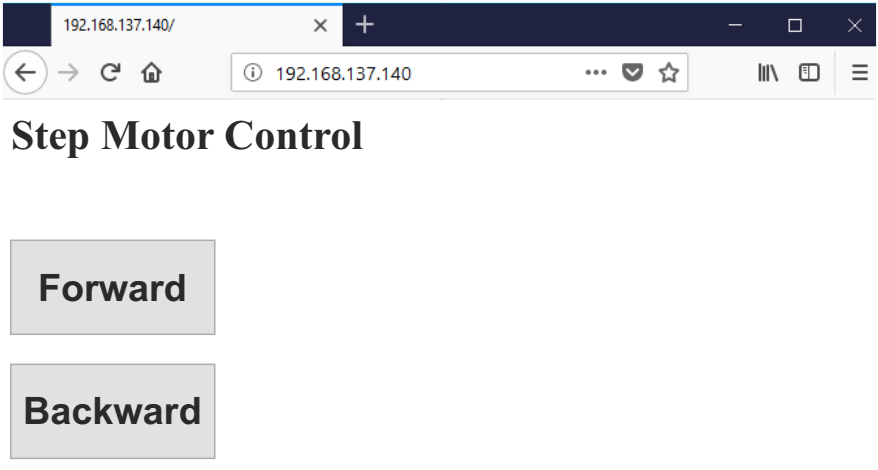
The “stepBack” function works just like the “stepForward” function, but the HIGH values given to the motor pins are given in the reverse order of the “stepForward ”function. This allows the step motor to rotate in reverse.

Before assigning the codes to Arduino, the TX and RX pins of the ESP8266 module must be removed. When programming Arduino, if the TX and RX pins of the ESP8266 module are connected they prevent communication between the Arduino and the Computer, and thus the programming of Arduino.

After writing the sample code and setting up the sample circuit, let’s run the system.



Open the serial port via Arduino IDE and set the data rate to 115200. Communication with the ESP8266 module can be seen here. When the above settings are made, connect to the IP address of the serial port with a web browser.



Rotate the step motor by clicking the forward-back buttons in the interface that appears in the web browser.

robotistan BLOG

You can access the blog posts of applications from the link below.
http://bit.ly/arduinodersleri

YouTube

You can access the videos of applications from the link below.
http://bit.ly/arduinovideodersler

robotistan.com

Robotistan Elektronik Ticaret A.Ş.

Prepared by: İlge İPEK (Editor - Editor) - Yasir ÇİÇEK (Editor) - Mehmet Nasır KARAER (Graphics)
info@robotistan.com - www.robotistan.com
Phone: 0850 766 425