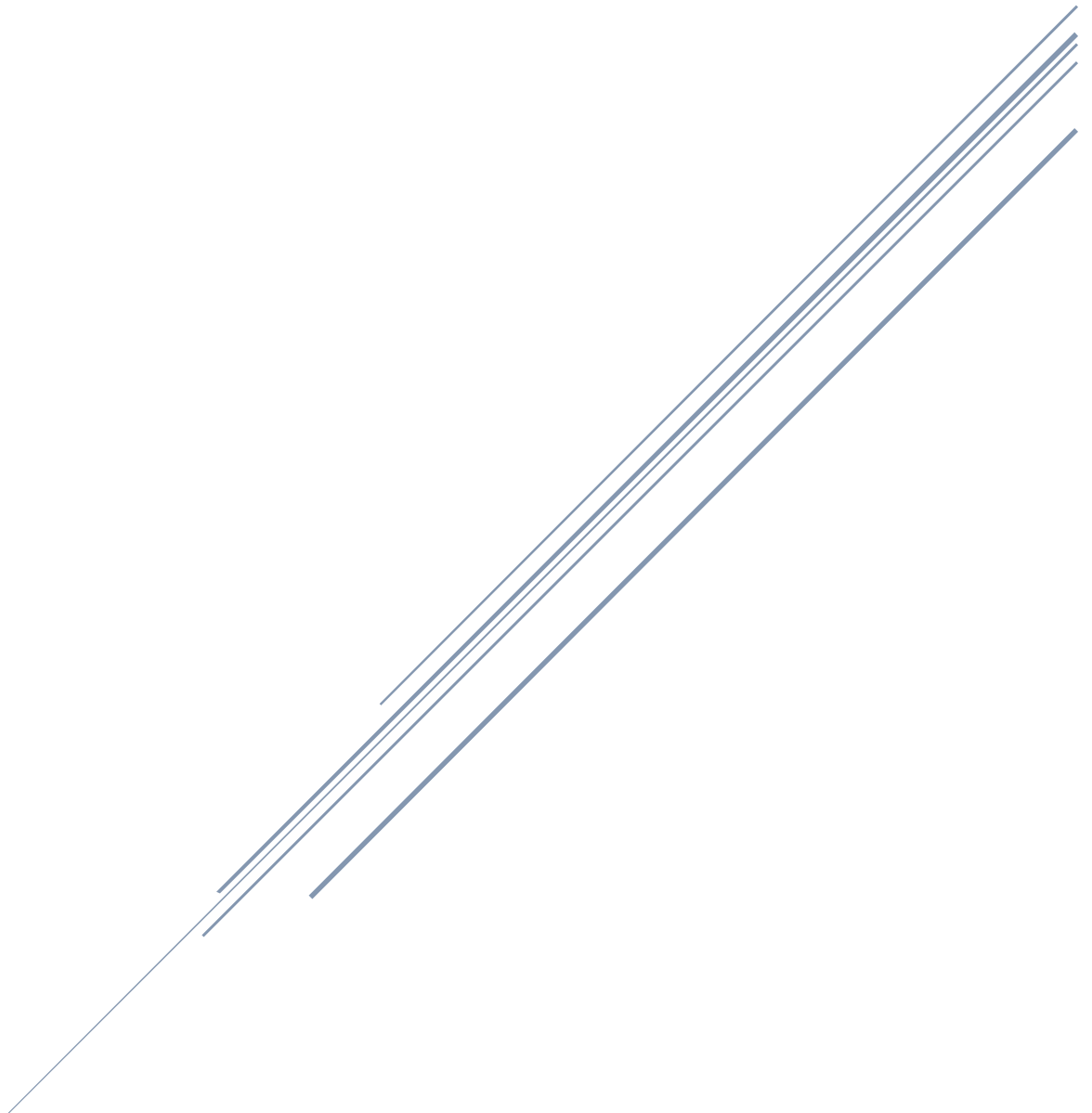# PROGRAMMING WITH ARDUINO - 1

IESL RoboGames 2014

University of Moratuwa
Department of Computer Science & Engineering

Table of Contents

Introduction

The purpose of this self-learning document is to provide you the knowledge in robotics of how to use development boards, positioning actuators and measuring distance using micro controllers.

This self-learning document contains

- How to use Arduino development board
- Reading inputs
- Controlling servo motors
- Interface with ultrasonic distance measuring

Arduino Development Board

You will find that there are lot of work to do if you are using a PIC micro controller. You have to design a power supply unit, oscillator unit for the circuit that you are going to build and also you should have a separate programmer or a PIC development board.

Arduino is an open-source electronic prototyping platform based on Atmel micro controllers. It is not required to remove the micro controller from development board. You can directly program an Arduino development board if you have Arduino software using a USB cable. Also you can power it using USB cable while you are testing the circuit.

There are several types of Arduino development boards available in the market. You can use Arduino UNO board for this project. Also you can use any Arduino compatible UNO board instead of original Arduino board.

e.g.: Lakduino UNO, Funduino UNO

How to Setup Arduino Development Board

Step 1: Download and install Arduino software. You will be prompted to install USB driver. Install that as well.

Step 2: Connect Arduino to the PC using USB cable.

Step 3: Go to device manager and double click on "Ports (COM & LPT)", find the port number of your Arduino as follows.
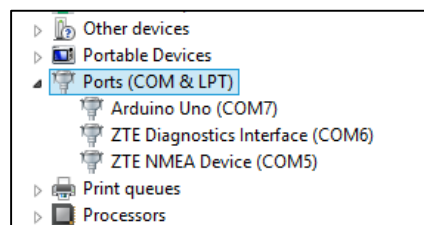
Step 4: Open Arduino software and set the port as follows.
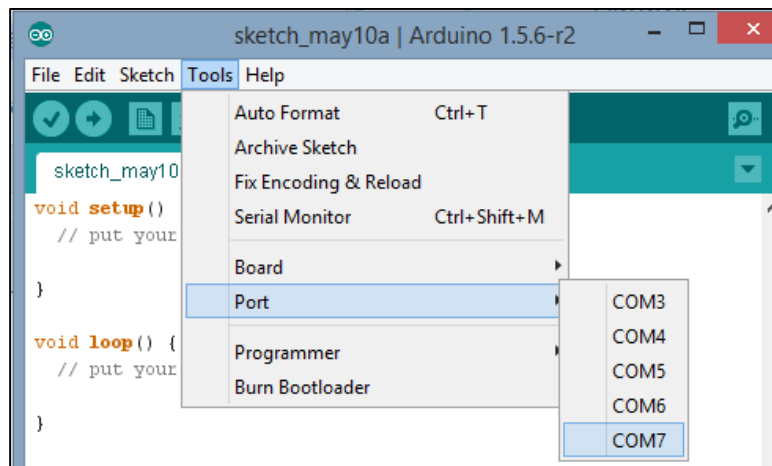
Step 4: Now choose the type Arduino board you are using as follows.
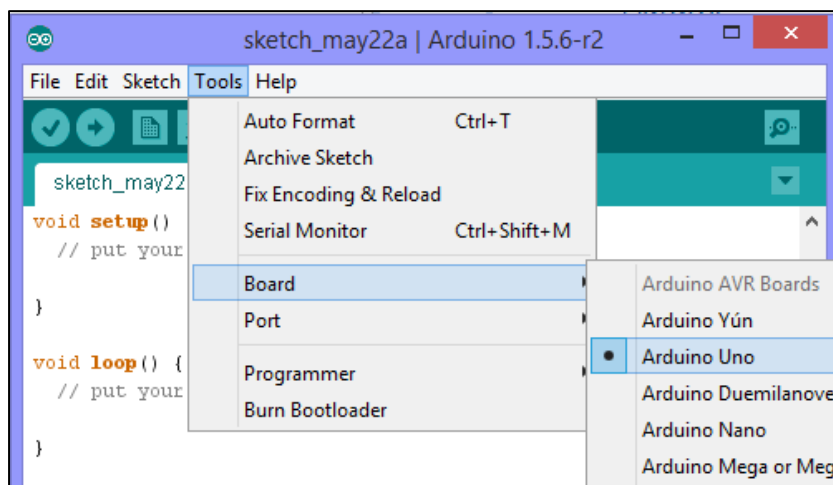
Now you are ready to program the Arduino board!!

Try Your First Arduino Code – LED Blink

Open the Arduino software and go to, file -> Examples -> 01.Basics -> Blink.

The following code will be opened.

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

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

Here you can see the Arduino program for blinking an LED. Gray colour text is called comments. Those are not a part of the program. But they will help you to understand the program.

Let's understand the program.

- `int led = 13;`
  You have to connect an LED to Arduino. Therefore, you can use any I/O pin. Here it is 13th pin because Arduino comes with an LED which is directly connected to that pin. Therefore you do not need an external LED.
  This program code defines a name for the 13th I/O pin of Arduino board. Now onwards, name of the pin 13 is "led".

- ```
  void setup(){
  }
  ```
  This is a function. Arduino executes this function only once. You can initialize your I/O pins, set initial output values, and initialize communication protocols (Discuss later) within the curly braces of this function.

- ```
  pinMode(led, OUTPUT);
  ```
  You should define selected pin as an output pin to on/off the LED. This code does that. Here, "led" is the name of the pin you selected.

- ```
  void loop(){
  }
  ```
  This is the main function of your program. This function is called repeatedly as long as the Arduino is powered on.

- ```
  digitalWrite(led, HIGH);
  ```
  The output of 13th pin becomes logical high voltage level after executing this instruction. It means that voltage level of 13th pin becomes 5V. Then the LED lights up.

- ```
  delay(1000);
  ```
  This instruction delays the next instruction by a given time. Here it is 1000 milliseconds.

- ```
  digitalWrite(led, LOW);
  ```
  The output of 13th pin becomes logical low voltage level after executing this instruction. It means that voltage level of 13th pin becomes 0V. This switches the LED off.

Now you can upload this to your Arduino using Ctrl + U command or clicking second icon (Upload) below the menu bar. If there are any errors in your code or port is wrong, you will receive an error message mentioning the type of error.

If everything is OK, your program will be uploaded to the Arduino and you can see the LED is blinking.

Note: If you are using an external LED, connect it to Arduino using 1kΩ resistor. This will protect your LED because maximum supply voltage of an LED is 3V.

Exercise: Write a program to perform following task.
- ✓ Connect 3 LEDs to Arduino. (red, green and yellow)
- ✓ Write a program to simulate traffic lights.

Reading inputs

Open the Arduino software and go to, file -> Examples -> 02.Digital -> Button

You will see that two variable names are defined here. One is for LED and other one is for reading button inputs.

- `pinMode(ledPin, OUTPUT);`
  For defining an input, you only have to use above code segment in the setup function.

- `digitalRead(buttonPin);`
  After defining the input, you can read input values using "`digitalRead(<pin_name>);`" function. This gives the input of the pin you defined. Returning value is either "HIGH" or "LOW". You can save it using a variable as in the Arduino example.

- ```
  if (buttonState == HIGH) {
       digitalWrite(ledPin, HIGH);
  }
 else{
       digitalWrite(ledPin, LOW);
  }
  ```
  This is called condition checking. Here condition checking has two options.
  If value of the variable "buttonState" is "HIGH", then instructions within curly braces of "if" condition checking are executed and instructions within curly braces of "else" are not executed.
  If value of the variable "buttonState" is not "HIGH" (if value of variable is "LOW"), then only the instuctions within curly braces of else are executed.

  Note:   There are several types of condition checking instructions are available.
  e.g.:   -   if
          -   if else (described above)
          -   if else if
          -   switch case

To test this program, you should connect a push button to your Arduino board. You can use push button switches in two ways. That can be done according to following diagrams.
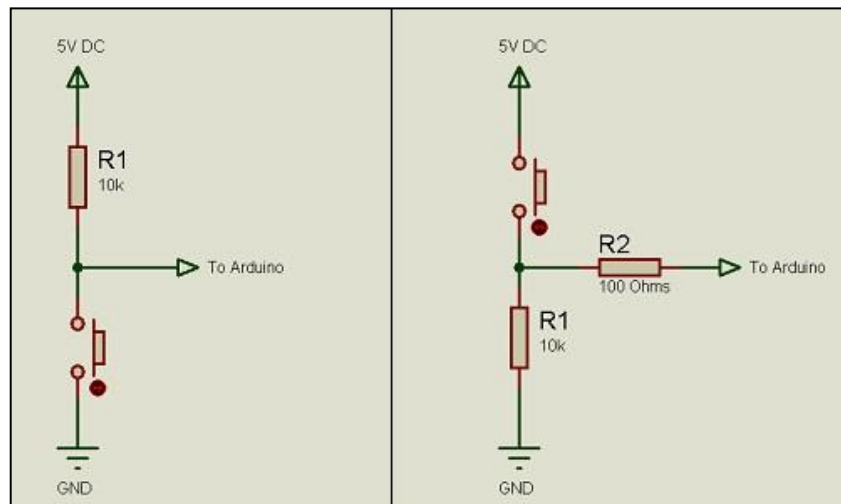


In left side circuit, Arduino input becomes high state when the push button is released. In right side one, input becomes high when the button is pressed. You can choose circuit type according to your requirement.

**Exercise:** Write a program to perform following task.
- ✓ Connect a reflective IR optocoupler and an LED to Arduino.
- ✓ Read the input of the optocoupler and use it to detect black and white backgrounds separately.
- ✓ Turn the LED on if optocoupler is in white background and turn off the LED if optocoupler is in black background.

# Servo Motors

Servo motors are unlike usual DC motors you have seen. Typically they rotate within 180 degrees. Servos are specially used for positioning such as rotate something for a known angle between 0° and 180°. Servos are widely used in robot arms. All the servo motors contains a gear set inside it. Therefore you can obtain a higher torque using servo motors.
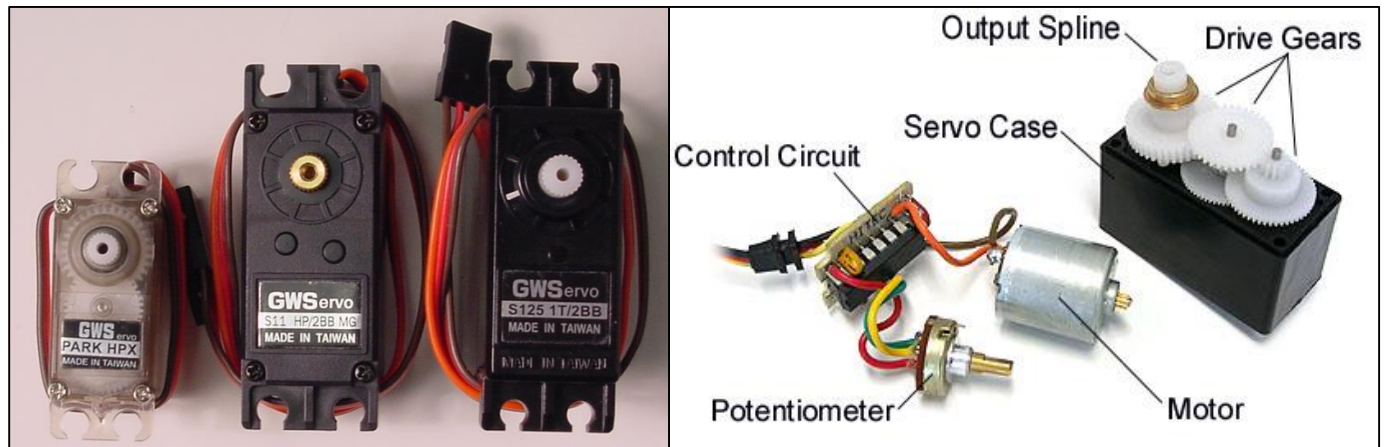


Different sized servo motors

Inside of a servo motor

Servo motors comes with several different servo arms. Servo arms can be attached to the spindle of servo. You can choose the shape and size of the servo arm according to you requirement.

You can see that there are 3 wires comes out from a servo. Two of the wired are used to supply power to the servo and the other wire is used to send the signal to the servo. Servo is positioned according to that signal. Following figure will help you in identifying wires of servos.



Different servo arms
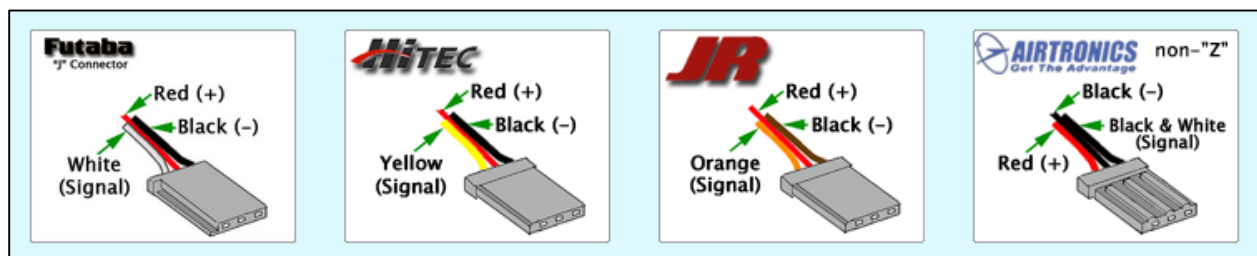
- Standard supply voltage for servo is between 4.8 - 6.0 V. Exceeding the voltage will damage the internal circuit of servo.
- Polarity of supply voltage is important. Do not supply voltage in reverse direction.

Servos are controlled using a specific signal which is sent to the signal wire which is called PWM signal. There is a minimum pulse, a maximum pulse, and a repetition rate. Neutral position of servo is defined as the position where the servo can rotate same amount of angle in both clockwise and counter-clockwise direction. The PWM sent to the servo determines position of the shaft, and based on the duration of the pulse sent via the control wire, the rotor will turn to the desired position.

The servo motor expects to see a pulse every 20 milliseconds and the width of the pulse width will determine the stopping angle. For an example, 1.5ms pulse will turn servo to $90^{\circ}$ position (Neutral position). 1ms pulse will turn servo to $0^{\circ}$ position and 2ms pulse will turn servo to $180^{\circ}$ position.

Until you keep giving the signal to the servo, it maintains the position. Suppose you are giving signal with 1.5ms pulse. Servo will rotate and stop at $90^{\circ}$ position. If an external force pushes against the servo while the servo is holding a position, the servo will resist from moving out of that position. The maximum amount of force the servo can exert is called the torque rating of the servo.

Following diagram shows the standard servo controlling signal.

## Controlling Servos using Arduino

Open the Arduino software and go to, file -> Examples -> Servo -> Sweep

You can see the following program.

```
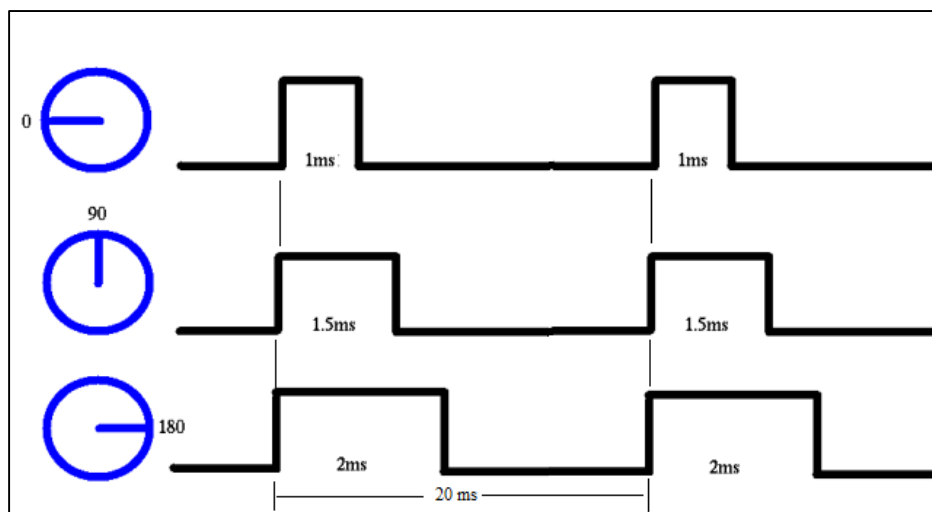#include <Servo.h>

Servo myservo;  // create servo object to control a servo
                // twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop()
{
  for(pos = 0; pos <= 180; pos += 1) // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=0; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

- `#include <Servo.h>`
  Arduino contains a library called "Servo.h" which reduces the workload of programmers. In the first line of this program, servo library is included to the project.

- `Servo myservo;`
  A variable called "myservo" is defined for the servo motor. Now onwards, name of the servo motor is "myservo".

- `int pos;`
  An integer variable is defined to store the position of servo.

- `myservo.attach(9);`
  In setup function, you should give the pin number of Arduino development board where you connect the signal line of the servo.

- ```
  for(pos = 0; pos <= 180; pos += 1) {
  }
  ```
  This is called "for" loop. Any code segment that is written between curly braces of the loop executed repeatedly until looping condition is true.

  Let's figure out the loop condition. You can see that there are three code segments within brackets of the loop.

    1. `pos = 0;`
       This initialize the variable "pos" to zero.

    2. `pos <= 180;`
       This is the loop condition. Meaning of this condition is "pos less than or equal to 180". The loop continues until this condition is true. As initial value of "pos" is zero, it is less than 180. Therefore, the loop continues. This condition is checked before starting each loop cycle.

    3. `pos += 1`
       Here, value of "pos" is incremented by 1. This instruction is same as following instructions.
       ```
       pos = pos +1
       ++pos
       ```
       Each time the loop is repeated, this statement executes. When the value of "pos" becomes 181, the loop condition becomes false and then the loop ends.

- `myservo.write(pos);`
  This instruction sends the relevant PWM signal to the servo which is required to stop the servo at given angle.

Exercise 1: Write a program to perform following task.
- ✓ Start at 0° position.
- ✓ Rotate and stop at 90° in steps of 15 degrees.
- ✓ Time between each step should be 2 seconds.

**Exercise 2:** Write a program to perform following task.
- ✓ Start at 90° position.
- ✓ Rotate in steps of 15 degrees clockwise according to a push button input.
- ✓ Time between two button presses should be greater than 1 seconds.

Ultrasonic Transducers

Ultrasonic transducers work on a principle similar to sonar which evaluate attributes of a target by interpreting the echoes from sound waves. Ultrasonic sensors generate high frequency sound waves and evaluate the echo which is received back by the sensor. We can calculate the time interval between sending the signal and receiving the echo to determine the distance to an object.

Ultrasonic sensors can be used to detect objects within 2cm to 4 meters. Also they can be used to measure distance to the object which is located within above distance.

Here we uses HC-SR04 ultrasonic ranging module for explaining. You can download the datasheet by clicking above link. It contains all technical specification and functionality of the module.
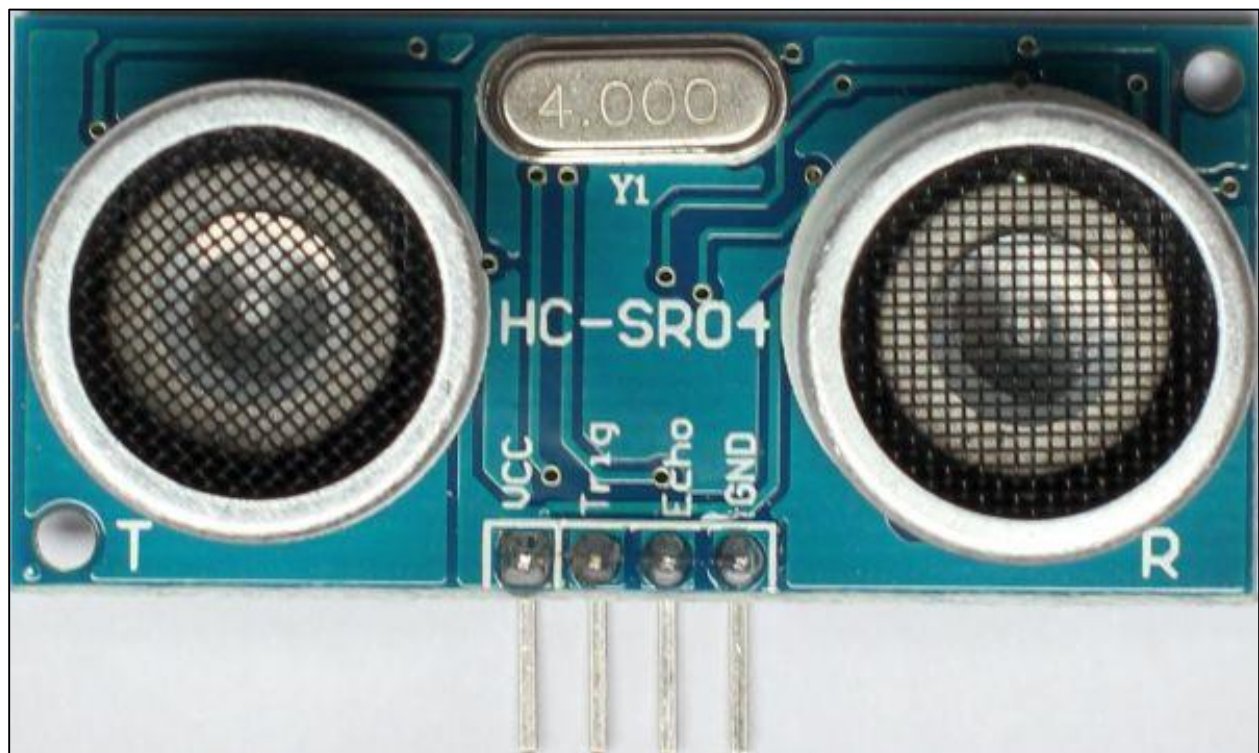
You can see that there are 4 pins in the module. Two of them are used to supply power to the module. Other two pins are used to trigger the module and receive the echo.

- ➢ VCC : 5V power supply
- ➢ Trig : Trigger input pin
- ➢ Echo : Receive pin
- ➢ GND : Power ground

Operating principle

- Before start measuring, set the Trig pin low when initializing the module.
- Then, transmit at least 10 µs high level pulse to the Trig pin. The module will automatically sends eight 40 kHz square wave.
- Then wait to capture the rising edge output by echo pin, at the same time, open the timer to start timing. Next, once again capture the falling edge output by echo pin, at the same time, read the time of the counter, which is the ultrasonic running time in the air.
- Finally distance can be calculated using following formula.

$$Distance = (high\ level\ time \times velocity\ of\ ultrasonic\ in\ air)\ /\ 2$$

Velocity of ultrasonic in air = 340 ms^{-1}

$$Distance\ in\ cm = \frac{high\ level\ time\ in\ \mu s}{58}$$

Interfacing HC-SR04 with Arduino

Open the Arduino software and go to, file -> Examples -> 06.Sensors -> Ping

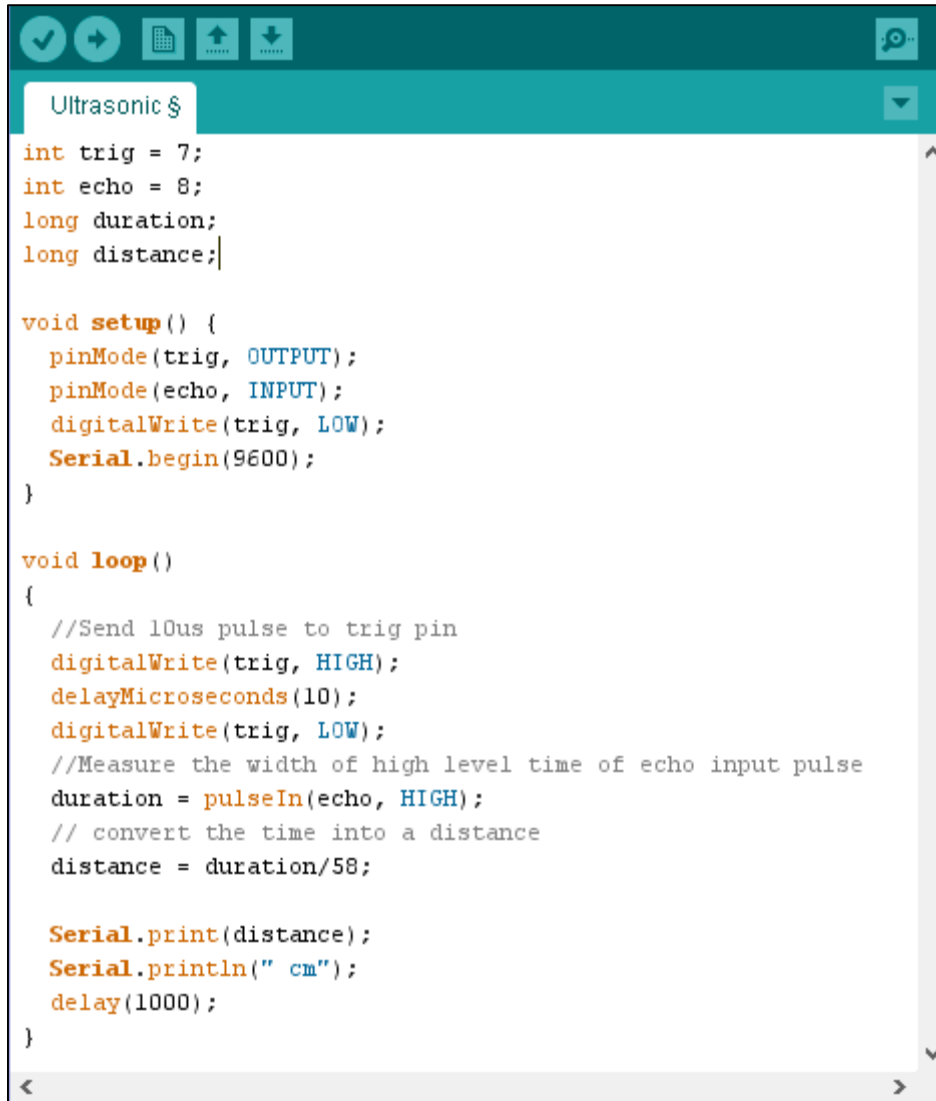We cannot use this program directly. Therefore change the program as follows.

- ➢ Define two variables for trigger pin and echo pin.
- ➢ Define variables to store duration and distance.
- ➢ Initialize trigger pin as an output and echo pin as an input.
- ➢ Set initial output of trigger pin to low.
- ➢ In loop function, set trigger pin high for 10 microseconds.
- ➢ Read the echo pin using "`pulseIn()`" function and store the result in "duration".
- ➢ Calculate the distance using the formula described above.
- ➢ Send the distance value to "`Serial.println()`" function.
- ➢ Set time gap between measurements to 1000 milliseconds.

Figure on next page shows the completed Arduino program to measure distance using ultrasonic module.

You can see that type of "duration" and "distance" variables are declared as "long". Because long type variables can store 32bit number. "int" type variable can store only 16 bit number.

As we have to read the distance calculated by Arduino, calculated distance is sent to computer through USB cable which is connected to Arduino.

You can read the values using your computer by clicking "Serial Monitor" icon (icon at top right corner in the following figure) in your Arduino software.

```
Ultrasonic §
int trig = 7;
int echo = 8;
long duration;
long distance;

void setup() {
  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);
  digitalWrite(trig, LOW);
  Serial.begin(9600);
}

void loop()
{
  //Send 10us pulse to trig pin
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  //Measure the width of high level time of echo input pulse
  duration = pulseIn(echo, HIGH);
  // convert the time into a distance
  distance = duration/58;

  Serial.print(distance);
  Serial.println(" cm");
  delay(1000);
}
```

- Serial.begin(9600);
  This function initializes the Arduino to send data to the computer. 9600 is called as "baud rate".
  You should select same baud rate in this function as the baud rate of Serial Monitor.

- pulseIn(echo, High);
  This function calculates how much time the output of echo pin was in high state.

- Serial.print(distance);
  Using this function, you can send the value of "distance" variable to the computer.

- Serial.println(" cm");
  This function sends the string within inverted commas to the computer. Difference between Serial.print() and Serial.println() is, Serial.println() function send data to computer with a new line. Therefore, each measured distance will be shown in the Serial Monitor in a new line.

Following figure shows the sample output of the program on Serial Monitor.
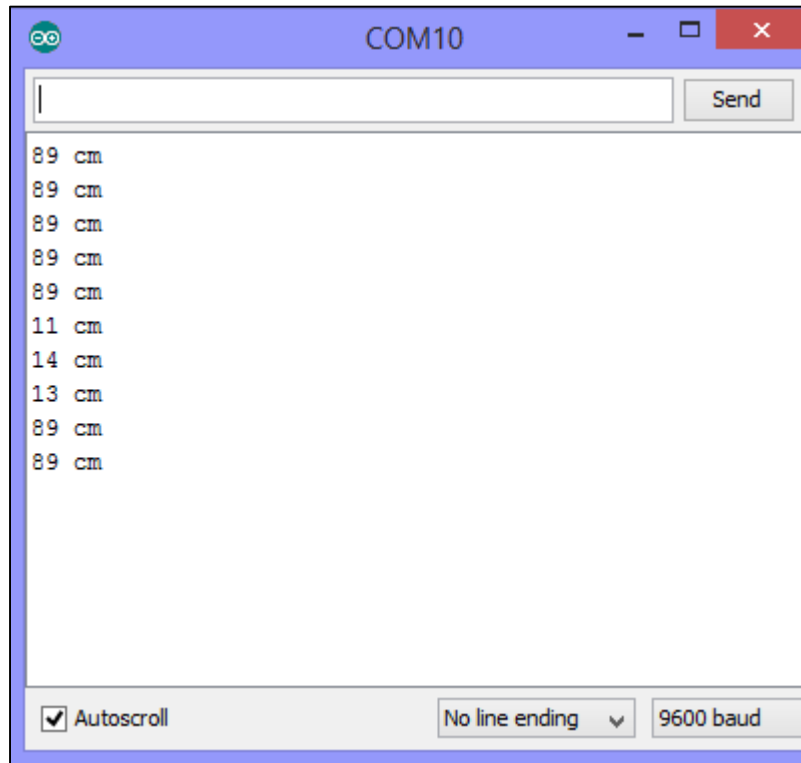
Serial.print(), Serial.println() functions are very useful when you need to check the calculated values, readings of the inputs and outputs as you can read them directly using a computer. That also a major advantage of using Arduino.

Exercise 1: Write a program to perform following task.
- ✓ Measure distance to an obstacle which is located within 2cm to 300 cm.
- ✓ Use two LEDs (red and green).
- ✓ Turn the red LED on and use Serial monitor to display "Warning!" if distance to the obstacle is less than 10 cm.
- ✓ Turn the green LED on and use Serial monitor to display "OK…" if distance to the obstacle is greater than 20 cm.

Exercise 2: Write a program to perform following task.
- ✓ Connect a servo and an ultrasonic module to Arduino.
- ✓ Measure distance to an obstacle which is located within 2cm to 300 cm.
- ✓ Rotate servo from 0° to 180° according to the distance.
- ✓ Hint: use map function in Arduino.