**REVIEW REPORT-3**

**PROGRAM CODE:**

#include <Servo.h>

#include <DFRobot\_DHT11.h>

DFRobot\_DHT11 DHT;

Servo plg;

Servo seed;

#define DHT11\_PIN 7

int m1=A2;

int m2=A3;

int m3=A4;

int m4=A5;

int tr=A1;

int ec=A0;

int i=0;

int x,dst;

int sp=4;

int sd=0;

void setup() {

// put your setup code here, to run once:

pinMode(m1,OUTPUT);

pinMode(m2,OUTPUT);

pinMode(m3,OUTPUT);

pinMode(m4,OUTPUT);

pinMode(tr,OUTPUT);

pinMode(sp,OUTPUT);

pinMode(ec,INPUT);

plg.attach(5);

seed.attach(6);

plg.write(60);

seed.write(0);

Serial.begin(9600);

digitalWrite(sp,1);

}

void loop() {

// put your main code here, to run repeatedly:

DHT.read(DHT11\_PIN);

int t=DHT.temperature;

int h=DHT.humidity;

digitalWrite(tr,0);

delayMicroseconds(2);

digitalWrite(tr,1);

delayMicroseconds(10);

digitalWrite(tr,0);

delayMicroseconds(2);

dst=pulseIn(ec,1)/58.2;

Serial.println("D:" +String(dst) + " T:"+String(t)+ " H:"+String(h));

if(Serial.available())

{

x=Serial.read();

if(x=='0')

{

digitalWrite(sp,1-digitalRead(sp));

if(digitalRead(sp)==0)

Serial.println("Spray ON");

else

Serial.println("Spray OFF");

}

}

if(x=='1')

{

if(dst<20)

{

digitalWrite(m1,0);

digitalWrite(m2,0);

digitalWrite(m3,0);

digitalWrite(m4,0);

Serial.println("Obstacle");

delay(500);

}

else

{

digitalWrite(m1,1);

digitalWrite(m2,0);

digitalWrite(m3,1);

digitalWrite(m4,0);

}

}

if(x=='2')

{

digitalWrite(m1,0);

digitalWrite(m2,1);

digitalWrite(m3,0);

digitalWrite(m4,1);

}

if(x=='3')

{

digitalWrite(m1,0);

digitalWrite(m2,1);

digitalWrite(m3,1);

digitalWrite(m4,0);

}

if(x=='4')

{

digitalWrite(m1,1);

digitalWrite(m2,0);

digitalWrite(m3,0);

digitalWrite(m4,1);

}

if(x=='5')

{

digitalWrite(m1,0);

digitalWrite(m2,0);

digitalWrite(m3,0);

digitalWrite(m4,0);

}

if(x=='7')

{

plg.write(60);

delay(300);

}

if(x=='6')

{

plg.write(20);

delay(300);

}

if(x=='8' || sd==1)

{

sd=1;

seed.write(90);

delay(500);

seed.write(0);

delay(500);

}

if(x=='9')

{

sd=0;

}

delay(100);

}

**SERIAL BLUETOOTH TERMINAL PROGRAM:**

{

"package": "de.kai\_morich.serial\_bluetooth\_terminal",

"date": " 20231115-113156",

"config": {

"pref\_auto\_scroll": true,

"pref\_char\_delay": "0",

"pref\_charset": "UTF-8",

"pref\_clear\_send": false,

"pref\_device\_address": "00:22:09:01:26:9A",

"pref\_device\_le": false,

"pref\_device\_name": "HC-05",

"pref\_device\_tab": 0,

"pref\_font\_family": "0",

"pref\_font\_size": "14",

"pref\_keep\_screen\_on": false,

"pref\_line\_delay": "0",

"pref\_macro\_rows": "1",

"pref\_receive\_buffer\_size": "10000",

"pref\_receive\_display\_mode": "0",

"pref\_receive\_newline": "CR+LF",

"pref\_send\_display\_mode": "0",

"pref\_send\_newline": "Auto",

"pref\_show\_send": true,

"pref\_show\_status": true,

"pref\_show\_timestamp": true,

"pref\_timestamp\_format": "HH:mm:ss.SSS"

}

}

Using temperature and humidity sensors in a seed spraying and seed sowing robot can be valuable for monitoring environmental conditions in the field. These sensors provide data that can be used to make informed decisions about planting and spraying. Here's some information on temperature and humidity sensors for use with Arduino:

**Temperature Sensors:**

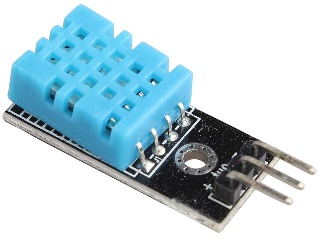
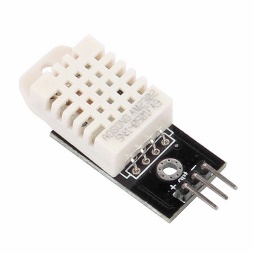
1. **DS18B20 Temperature Sensor:**
   * **Description:** This is a popular digital temperature sensor that uses the One-Wire protocol. It is accurate and relatively simple to use.
   * **Features:**
     + Wide temperature range (-55°C to +125°C).
     + High accuracy.
     + Can be easily interfaced with Arduino.
2. **DHT Series (DHT11, DHT22) Humidity and Temperature Sensors:**
   * **Description:** DHT sensors are commonly used for measuring both temperature and humidity. The DHT22 is more accurate than the DHT11.
   * **Features:**
     + Measures temperature and humidity.
     + DHT22 has a wider operating range and higher accuracy than DHT11.
     + Single-wire communication.
3. **LM35 Temperature Sensor:**
   * **Description:** An analog temperature sensor that provides a linear output proportional to the Celsius temperature.
   * **Features:**
     + Wide temperature range.
     + Calibrated directly in Celsius.
     + Simple analog interface.

**Humidity Sensors:**

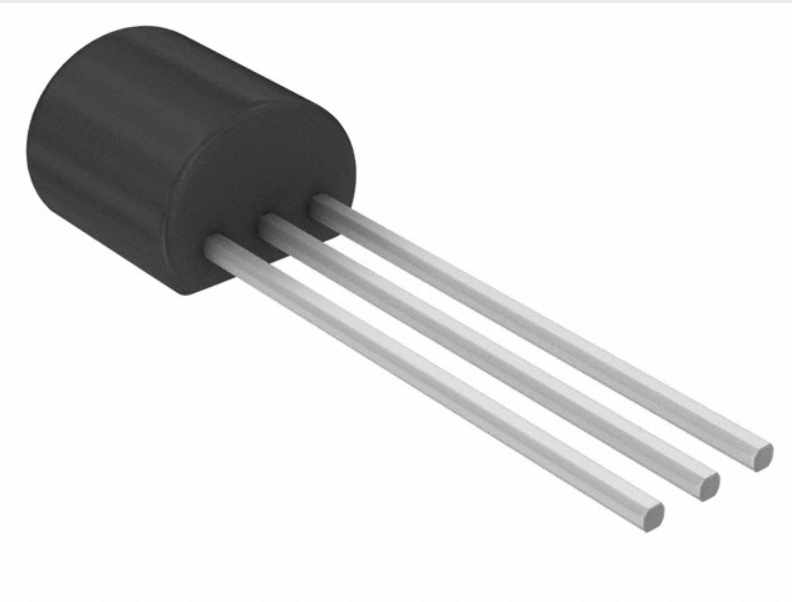
1. **DHT Series (DHT11, DHT22) Humidity and Temperature Sensors:**
   * **Description:** As mentioned earlier, these sensors measure both temperature and humidity.
   * **Features:**
     + Measures humidity and temperature.
     + DHT22 has higher accuracy compared to DHT11.
     + Single-wire communication.
2. **HTU21D Humidity and Temperature Sensor:**
   * **Description:** A digital humidity sensor with integrated temperature sensor.
   * **Features:**
     + High accuracy.
     + I2C communication.
     + Low power consumption.
3. **SHT Series (SHT15, SHT31) Humidity and Temperature Sensors:**
   * **Description:** SHT series sensors are known for their accuracy and reliability.
   * **Features:**
     + Measures humidity and temperature.
     + Digital output with I2C or serial communication.
     + High precision and long-term stability.

**Connecting Sensors to Arduino:**

1. **DS18B20:**
   * Connects to the digital pins of Arduino. Requires a pull-up resistor.
   * Library: OneWire and DallasTemperature libraries can be used for interfacing.
2. **DHT Series:**
   * Connects to digital pins. DHT11 requires a 5K ohm pull-up resistor, while DHT22 has a built-in pull-up resistor.
   * Library: Adafruit DHT sensor library or DHT library.
3. **LM35:**
   * Connects to analog pins. No additional components are required.
   * Reading is directly proportional to temperature.
4. **HTU21D and SHT Series:**
   * Connect via I2C communication.
   * Libraries: Adafruit\_SHT31 and Adafruit\_HTU21DF for SHT and HTU21D respectively.

**DHT11 DHT22 DS18B20**

**LM35 HTU21D SHT15**

#include <DHT.h>

#define DHTPIN 2 // Digital pin connected to the DHT sensor

#define DHTTYPE DHT22 // DHT 22 (AM2302) sensor type

DHT dht(DHTPIN, DHTTYPE);

void setup() {

Serial.begin(9600);

dht.begin();

}

void loop() {

delay(2000);

float temperature = dht.readTemperature();

float humidity = dht.readHumidity();

Serial.print("Temperature: ");

Serial.print(temperature);

Serial.print(" °C, Humidity: ");

Serial.print(humidity);

Serial.println(" %");}

**COMMUNICATION:**

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, an \*.inf file is required..

The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-toserial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. To use the SPI communication, please see the ATmega328 datasheet.

**Programming** :

The Arduino Uno can be programmed with the Arduino software. The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C headerfiles). You can also bypass the bootloader and program the microcontroller through the ICSP (InCircuit Serial Programming) header; see these instructions for details. The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by: On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. 10 On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

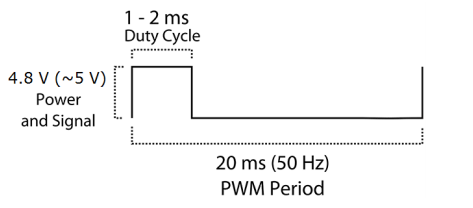
**SERIAL COMMUNICATION:-**

To transfer to a device located many meters away, the serial method is used. The data is sent one bit at a time. Here not 8bit data is send 2 extra bit are send along with it .this two bit are called start bit and stop bit. These tow bit are used so synchronization can be done between transmitter and receiver.



### ****How to use a Servo Motor:****

After selecting the right Servo motor for the project, comes the question how to use it. As we know there are three wires coming out of this motor. The description of the same is given on top of this page. To make this motor rotate, we have to power the motor with +5V using the Red and Brown wire and send PWM signals to the Orange colour wire. Hence we need something that could generate PWM signals to make this motor work, this something could be anything like a 555 Timer or other Microcontroller platforms like Arduino, PIC, ARM or even a microprocessor like Raspberry Pie. Now, how to control the direction of the motor? To understand that let us a look at the picture given in the datasheet.



From the picture we can understand that the PWM signal produced should have a frequency of 50Hz that is the PWM period should be 20ms. Out of which the On-Time can vary from 1ms to 2ms. So when the on-time is 1ms the motor will be in 0° and when 1.5ms the motor will be 90°, similarly when it is 2ms it will be 180°. So, by varying the on-time from 1ms to 2ms the motor can be controlled from 0° to 180°

**ARDUINO IDE:**

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

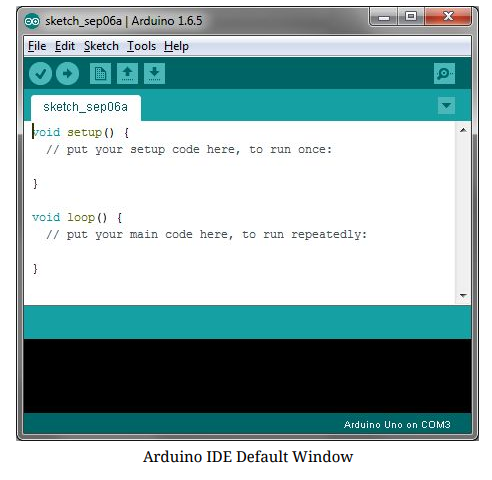
### Writing Sketches

Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

**NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.**

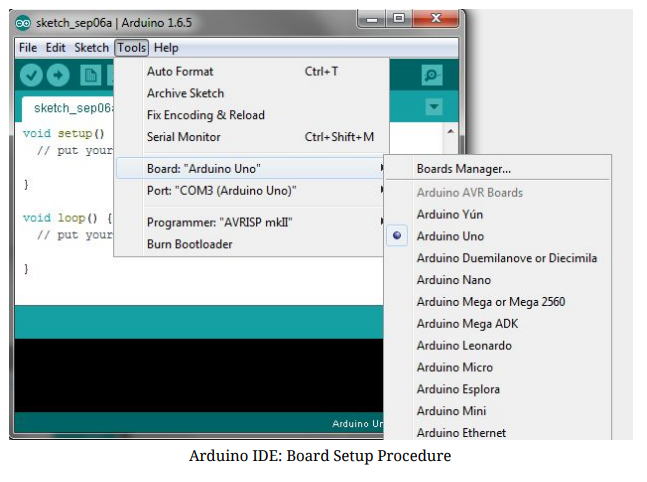
**Arduino IDE: Initial Setup**

This is the Arduino IDE once it’s been opened. It opens into a blank sketch where you can start programming immediately. First, we should configure the board and port settings to allow us to upload code. Connect your Arduino board to the PC via the USB cable.

****

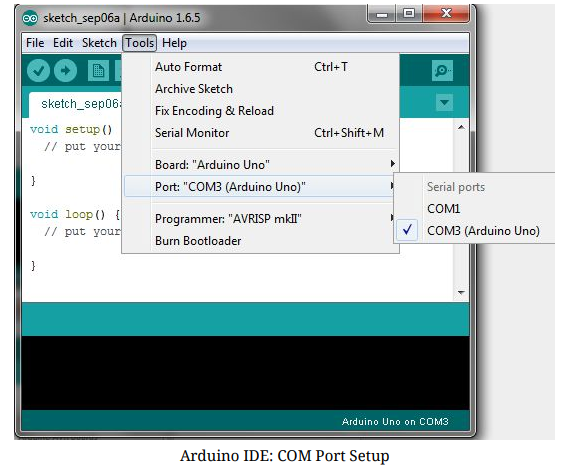
**IDE: Board Setup**

You have to tell the Arduino IDE what board you are uploading to. Select the Tools​pulldown menu and go to Board.​This list is populated by default with the currently available Arduino Boards that are developed by Arduino. If you are using an Uno or an Uno-Compatible Clone (ex. Funduino, SainSmart, IEIK, etc.), select Arduino Uno. If you are using another board/clone, select that board.



**IDE: COM Port Setup**

If you downloaded the Arduino IDE before plugging in your Arduino board, when you plugged in the board, the USB drivers should have installed automatically. The most recent Arduino IDE should recognize connected boards and label them with which COM port they are using. Select the Tools pulldown menu and then Port.​Here it should list all open COM ports, and if there is a recognized Arduino Board, it will also give it’s name. Select the Arduino board that you have connected to the PC. If the setup was successful, in the bottom right of the Arduino IDE, you should see the board type and COM number of the board you plan to program. Note: the Arduino Uno occupies the next available COM port; it will not always be COM3.



**Testing Your Settings:**

Uploading Blink One common procedure to test whether the board you are using is properly set up is to upload the “Blink” sketch. This sketch is included with all Arduino IDE releases and can be accessed by the File​pull-down menu and going to Examples, 01.Basics, ​and then select Blink​. Standard Arduino Boards include a surface-mounted LED labeled “L” or “LED” next to the “RX” and “TX” LEDs, that is connected to digital pin 13. This sketch will blink the LED at a regular interval, and is an easy way to confirm if your board is set up properly and you were successful in uploading code. Open the “Blink” sketch and press the “Upload” button in the upper-left corner to upload “Blink” to the board.