**ASSIGNMENT**

**PROGRAMMING WITH ARDUINO AND RASPBERRY PI**

**16-12-2021**

**Submitted to,**

**Sangeetha Jose**

**Submitted by,**

**Anandu P R**

**Roll No: 9**

**S1 MCA**

**Arduino**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards senses the environment by receiving inputs from many sensors, and affects their surroundings by controlling lights, motors, and other actuators. Arduino boards are the microcontroller development platform that will be at the heart of your projects. When making something you will be building the circuits and interfaces for interaction, and telling the microcontroller how to interface with other components.

[Arduino boards](https://www.arduino.cc/en/Main/Products)are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the [Arduino programming language](https://www.arduino.cc/en/Reference/HomePage)(based on [Wiring](http://wiring.org.co/)), and [the Arduino Software (IDE)](https://www.arduino.cc/en/Main/Software), based on [Processing](https://processing.org/).

Arduino board designs use a variety of[microprocessors](https://en.wikipedia.org/wiki/Microprocessor)and controllers. The boards are equipped with sets of digital and analog[input/output](https://en.wikipedia.org/wiki/Input/output)(I/O) pins that may be interfaced to various expansion boards ('shields') or[breadboards](https://en.wikipedia.org/wiki/Breadboards)(for prototyping) and other circuits. The boards feature serial communications interfaces, including[Universal Serial Bus](https://en.wikipedia.org/wiki/Universal_Serial_Bus)(USB) on some models, which are also used for loading programs. The microcontrollers can be programmed using the[C](https://en.wikipedia.org/wiki/C_(programming_language))and[C++](https://en.wikipedia.org/wiki/C%2B%2B)[programming languages](https://en.wikipedia.org/wiki/Programming_language), using a standard API which is also known as the Arduino language, originated from the[Processing language](https://en.wikipedia.org/wiki/Processing_(programming_language)). In addition to using traditional[compiler](https://en.wikipedia.org/wiki/Compiler)[toolchains](https://en.wikipedia.org/wiki/Toolchains), the Arduino project provides an[integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment)(IDE) and a command line tool developed in[Go](https://en.wikipedia.org/wiki/Go_(programming_language)).

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of [accessible knowledge](http://forum.arduino.cc/)that can be of great help to novices and experts alike.

**Arduino UNO**

Arduino UNO is a low-cost, flexible, and easy-to-use programmable open-source microcontroller board that can be integrated into a variety of electronic projects. This board can be interfaced with other Arduino boards, Arduino shields, Raspberry Pi boards and can control relays, LEDs, servos, and motors as an output.

Arduino UNO features AVR microcontroller Atmega328, 6 analogue input pins, and 14 digital I/O pins out of which 6 are used as PWM output.

**Arduino Uno Configuration**

As we discussed we know that Arduino Uno is the most standard board available and probably the best choice for a beginner.

We can directly connect the board to the computer via a USB Cable which performs the function of supplying the power as well as acting as a serial port.

**Vin:** This is the input voltage pin of the Arduino board used to provide input supply from an external power source.

**5V:** This pin of the Arduino board is used as a regulated power supply voltage and it is used to give supply to the board as well as onboard components.

**3.3V:**This pin of the board is used to provide a supply of 3.3V which is generated from a voltage regulator on the board

**GND:**This pin of the board is used to ground the Arduino board.

**Reset:**This pin of the board is used to reset the microcontroller. It is used to Resets the microcontroller.

**Analog Pins:** The pins A0 to A5 are used as an analog input and it is in the range of 0-5V.

**Digital Pins:** The pins 0 to 13 are used as a digital input or output for the Arduino board.

**Serial Pins:**These pins are also known as a UART pin. It is used for communication between the Arduino board and a computer or other devices. The transmitter pin number 1 and receiver pin number 0 is used to transmit and receive the data resp.

**External Interrupt Pins:**This pin of the Arduino board is used to produce the External interrupt and it is done by pin numbers 2 and 3.

**PWM Pins:**This pins of the board is used to convert the digital signal into an analog by varying the width of the Pulse. The pin numbers 3,5,6,9,10 and 11 are used as a PWM pin.

**SPI Pins:**This is the Serial Peripheral Interface pin, it is used to maintain SPI communication with the help of the SPI library. SPI pins include:

SS: Pin number 10 is used as a Slave Select

MOSI: Pin number 11 is used as a Master Out Slave In

MISO: Pin number 12 is used as a Master In Slave Out

SCK: Pin number 13 is used as a Serial Clock

**LED Pin:** The board has an inbuilt LED using digital pin-13. The LED glows only when the digital pin becomes high.

**AREF Pin:**This is an analog reference pin of the Arduino board. It is used to provide a reference voltage from an external power supply.

**Arduino light blink program:**

const int LED = 13;

void setup()

{

pinMode(LED,OUTPUT);

}

void loop()

{

digitalWrite(LED,HIGH);

delay(1000);

digitalWrite(LED,LOW);

delay(1000);

}

**Debounce program:**

Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program. This example demonstrates how to debounce an input, which means checking twice in a short period of time to make sure the pushbutton is definitely pressed. Without debouncing, pressing the button once may cause unpredictable results.

const int buttonPin = 2; // the number of the pushbutton pin

const int ledPin = 13; // the number of the LED pin

// Variables will change:

int ledState = HIGH; // the current state of the output pin

int buttonState; // the current reading from the input pin

int lastButtonState = LOW; // the previous reading from the input pin

// the following variables are unsigned longs because the time, measured in

// milliseconds, will quickly become a bigger number than can be stored in an int.

unsigned long lastDebounceTime = 0; // the last time the output pin was toggled

unsigned long debounceDelay = 50; // the debounce time; increase if the output flickers

void setup() {

pinMode(buttonPin, INPUT);

pinMode(ledPin, OUTPUT);

// set initial LED state

digitalWrite(ledPin, ledState);

}

void loop() {

// read the state of the switch into a local variable:

int reading = digitalRead(buttonPin);

// check to see if you just pressed the button

// (i.e. the input went from LOW to HIGH), and you've waited long enough

// since the last press to ignore any noise:

// If the switch changed, due to noise or pressing:

if (reading != lastButtonState) {

// reset the debouncing timer

lastDebounceTime = millis();

}

if ((millis() - lastDebounceTime) > debounceDelay) {

// whatever the reading is at, it's been there for longer than the debounce

// delay, so take it as the actual current state:

// if the button state has changed:

if (reading != buttonState) {

buttonState = reading;

// only toggle the LED if the new button state is HIGH

if (buttonState == HIGH) {

ledState = !ledState;

}

}

}

// set the LED:

digitalWrite(ledPin, ledState);

// save the reading. Next time through the loop, it'll be the lastButtonState:

lastButtonState = reading;

}

**Raspberry Pi**

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What’s more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

**Installing the Operating System**

Raspberry Pi recommend the use of Raspberry Pi Imager to install an operating system on your SD card. You will need another computer with an SD card reader to install the image.

**Download the Raspberry Pi Imager for your operating system and follow the installation instructions.**

**Launch Raspberry Pi Imager**

There are 3 versions of Raspberry Pi OS available. We will briefly touch on each one but we will be using **Raspberry Pi OS (32-bit)** for this guide.

Raspberry Pi OS (32-bit) Lite: This version provides the bare essentials to get you up and running. There is no Graphical User Interface (GUI) with this image and the size is around 0.4 GB.

Raspberry Pi OS (32-bit): This version includes a GUI and has basic software installed. The size of this image is around 1.1 GB.

Raspberry Pi OS Full (32-bit): This version includes a GUI and more software installed than the non-full version. The size of this image is around 2.5 GB

**Select Raspberry Pi OS (32-bit)**

Step 3: Choose SD Card

You’ll now need your SD card connected to your computer to copy over the OS you chose.

**Select CHOOSE SD CARD and select the SD card you have connected to your computer.**

Select SD Card

You’re now ready to begin writing the OS to your SD card.

OS and SD Card selected

**Step 4: Write to SD Card**

This step will write the selected OS to the SD card and run a verification that the copy was successful.

**Select WRITE**

The writing process will vary depending on the OS you selected. Typically, this will only take a few minutes.

The OS has now been copied to your SD card. You are now ready to move on to booting your Raspberry Pi.

**Select CONTINUE**

Step 5: Booting Your Raspberry Pi

Insert your microSDHC card into your Raspberry Pi. Then, hook up your Raspberry Pi to power, keyboard, mouse, and monitor.

Note: There is a way to configure the Raspberry Pi OS with only power and a network connection. This is called a headless install and is a more advanced technique that will not be covered in this guide.

You should be brought directly into the Raspberry Pi OS with a**Welcome to Raspberry Pi** dialog on the display.

Welcome to Raspberry Pi

The next screen will be to set your country.

**Choose Your Country**

**Choose Your Language**

**Choose Your Timezone**

**Select the checkboxes if you are using the English language and an English keyboard**

**Click Next**

Set Country

The next screen will prompt you to change your password. The default username is “pi” and the default password is “raspberry”

It is best practice to enable strong passwords for your accounts.

**Enter a new password**

**Confirm the new password**

**Press Next**

Change Password

The next screen will ask you if there is a black border around the desktop. The desktop should take up the entire screen. If it doesn’t, Raspberry Pi OS can make the adjustment to fill the black space. This change will take effect when the Raspberry Pi is restarted.

**Select the checkbox if the screen shows a black border around the desktop.**

**Press Next**

Set Up Screen

The next screen will prompt you to connect the Raspberry Pi to a wireless network. If no internet connection is available you can press the skip button to proceed past this step. If you have a wired connection you will not see this step.

**Select a Wireless Network**

**Press Next**

Select Wireless Network

The next screen will prompt you to enter the wireless network password. You can uncheck “hide characters” to see the characters you are typing.

**Enter Wireless Network Password**

**Press Next**

Wireless Network Password

The next screen will ask you if you’d like to have the operating system and applications checked and updated if necessary. An internet connection is required to complete this step. Press “Skip” if you’d like to continue without checking.

Note: The time to complete this step will vary based on your internet connection and specs of your Raspberry Pi.

**Press Next**

Update Software

A popup will indicate when your system is up to date.

System is up to date

The last screen will let you know that setup is complete and your Raspberry Pi is good to go. Feel free to restart your Raspberry Pi for any system changes to take effect.

Setup Complete

If the wiring looks good, then you’re ready to write some Python to get the LED blinking. Start by creating a file for this circuit inside of the python-projects directory. Call this file led.py:

pi@raspberrypi:~/python-projects $ touch led.py

CODE

from gpiozero import LED

from signal import pause

led = LED(4)

led.blink()

pause()

Save the file and run it to see the LED blink on and off:

pi@raspberrypi:~/python-projects $ python3 led.py

The LED should now be blinking on and off every second. When you’re done admiring your Python code in action, stop the program with Ctrl+Cor *Stop*in M

**Raspberry Pi program using ultrasonic sensor**

For many (outdoor) projects a distance measurement is necessary or advantageous. These small modules are available starting at 1-2 bucks and can measure the distance up to 4-5 meters by ultrasound and are suprisingly accurate. This tutorial shows the connection and control.

**Hardware**

HC-SR04 Module ([US](https://www.amazon.com/s/ref=nb_sb_noss_2?tag=754u-20&url=search-alias%3Daps&field-keywords=HC-SR04)\* / [UK](https://www.amazon.co.uk/s/ref=nb_sb_noss/?tag=755-21&url=search-alias%3Daps&field-keywords=HC-SR04)\*)

Resistors: 330Ω and 470Ω ([US](https://www.amazon.com/s/ref=nb_sb_noss_2?tag=754u-20&url=search-alias%3Daps&field-keywords=restistors+set)\* / [UK](https://www.amazon.co.uk/s/ref=nb_sb_noss/?tag=755-21&url=search-alias%3Daps&field-keywords=restistors+set)\*)

Jumper wire ([US](https://www.amazon.com/s/ref=nb_sb_noss_2?tag=754u-20&url=search-alias%3Daps&field-keywords=jumper+wire)\* / [UK](https://www.amazon.co.uk/s/ref=nb_sb_noss/?tag=755-21&url=search-alias%3Daps&field-keywords=jumper+wire)\*)

**Wiring**

There are four pins on the ultrasound module that are connected to the Raspberry:

VCC to Pin 2 (VCC)

GND to Pin 6 (GND)

TRIG to Pin 12 (GPIO18)

connect the 330Ω resistor to ECHO. On its end you connect it to Pin 18 (GPIO24) and through a 470Ω resistor you connect it also to Pin6 (GND).

We do this because the GPIO pins only tolerate maximal 3.3V. The connection to GND is to have a obvious signal on GPIO24. If no pulse is sent, the signal is 0 (through the connection with GND), else it is 1. If there would be no connection to GND, the input would be undefined if no signal is sent (randomly 0 or 1), so ambiguous.

Here is the structure as a circuit diagram:

**Script for controlling**

First of all, the Python GPIO library should be installed

To use the module, we create a new script

sudo nano ultrasonic\_distance.py

with the following content:

#Libraries

import RPi.GPIO as GPIO

import time

#GPIO Mode (BOARD / BCM)

GPIO.setmode(GPIO.BCM)

#set GPIO Pins

GPIO\_TRIGGER = 18

GPIO\_ECHO = 24

#set GPIO direction (IN / OUT)

GPIO.setup(GPIO\_TRIGGER, GPIO.OUT)

GPIO.setup(GPIO\_ECHO, GPIO.IN)

def distance():

# set Trigger to HIGH

GPIO.output(GPIO\_TRIGGER, True)

# set Trigger after 0.01ms to LOW

time.sleep(0.00001)

GPIO.output(GPIO\_TRIGGER, False)

StartTime = time.time()

StopTime = time.time()

# save StartTime

while GPIO.input(GPIO\_ECHO) == 0:

StartTime = time.time()

# save time of arrival

while GPIO.input(GPIO\_ECHO) == 1:

StopTime = time.time()

# time difference between start and arrival

TimeElapsed = StopTime - StartTime

# multiply with the sonic speed (34300 cm/s)

# and divide by 2, because there and back

distance = (TimeElapsed \* 34300) / 2

return distance

if \_\_name\_\_ == '\_\_main\_\_':

try:

while True:

dist = distance()

print ("Measured Distance = %.1f cm" % dist)

time.sleep(1)

# Reset by pressing CTRL + C

except KeyboardInterrupt:

print("Measurement stopped by User")

GPIO.cleanup()

After that we run:

sudo python ultrasonic\_distance.py

So every second, the distance will be measured until the script is cancelled by pressing CTRL + C.