A

PROJECT REPORT

on

"AUTOMATED CURTAINS"

Submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

Session: - Jan- June 2022

Under Guidance of
Dr. Vivek Jain
Head Of Department
ECE & Techno India NJR Institute Of Technology

Submitted by
Amisha Bolia 18ETCEC300
Kushbu Suthar 18ETCEC005
Apex Sharma 18ETCEC002
8th Sem (ECE)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

TECHNO INDIA NJR INSTITUTE OF TECHNOLOGY, UDAIPUR-313001

MAY – 2022

Department of Electronics and Communication Engineering
Techno India NJR Institute of Technology, Udaipur-313001

Certificate

This is to certify that project work titled “AUTOMATED CURTAINS” by AMISHA BOLIA, KUSHBU SUTHAR, APEX SHARMA was successfully carried out in the Department of Electronics and Communication Engineering, TINJRIT and the report is approved for submission in the partial fulfilment of the requirements for award of degree of Bachelor of Technology in Electronics and Communication. The work has been completed in all the respects during session 2018-2022.

Prof. Yogendra Singh Solanki
Assistant Professor
Dept. of E.C.E TINJRIT, Udaipur
Date: 28 May 2022

Dr. Vivek Jain
Head of Department
Dept. of E.C.E TINJRIT, Udaipur
Date: 28 May 2022

Department of Electronics and Communication Engineering
Techno India NJR Institute of Technology, Udaipur-313001

Examiner Certificate

This is to certify that the following students

AMISHA BOLIA

KUSHBU SUTHAR

APEX SHARMA

of final year B.Tech. (Electronics & Communication Engineering), were examined for the project work titled

"AUTOMATED CURTAINS"

during the academic year 2018 – 2022 at Techno India NJR Institute of Technology, Udaipur

Remarks: Satifactory

Date: 24-5-2022

Vivekjain
Signature
(**Internal Examiner**)
Name :- Dr. Vivek Jain
Designation:- Associate Proff.
Department: - E.C.E.
Organization:- T.I.N.J.R.I.T

Hitesh Sen
Signature
(**External Examiner**)
Name :- HITESH SEN
Designation:- Assistant Professor
Department: - ECE
Organization:- ……………

ACKNOWLEDGMENT

We take this opportunity to record our sincere thanks to all who helped us to successfully complete this work. Firstly, we are grateful to **Dr. Vivek Jain** for his invaluable guidance and constant encouragement, support and most importantly for giving us the opportunity to carry out this work.

We would like to express our deepest sense of gratitude and humble regards to our

Head of Department Dr. Vivek Jain for giving invariable encouragement in our endeavours and providing necessary facility for the same. We are very thankful to **Prof. Yogendra Solanki, Director (Research & New Initiatives)** for encouraging us in this project work. Also, a sincere thanks to all faculty members of ECE, TINJRIT for their help in the project directly or indirectly.

Finally, we would like to thank my friends for their support and discussions that have proved very valuable for us. We are indebted to our parents for providing constant support, love and encouragement. We thank them for the sacrifices they made so that we could grow up in a learning environment. They have always stood by us in everything we have done, providing constant support, encouragement and love.

Amisha Bolia 18ETCEC300
Kushbu Suthar 18ETCEC005
Apex Sharma 18ETCEC003

CONTENTS

LIST OF FIGURES

Introduction

Efficiency, convenience, comfort, and security are the four pillars on which smart home technologies rest. If planning to live in such a home, one of the first things to get is an automatic curtain opener.

Convenience

Remember all the times one's forgot to draw the curtains before leaving home or, worse, after getting into bed? With smart motorized curtains in place, one won't have to worry about that anymore. Stating the obvious, automated curtains can be opened and closed through your smartphone. One can even automate your curtains' opening and closing times with smart home automation to enjoy outside views while you laze around on Sunday mornings.

Save Energy

Letting in more natural light can help save energy costs in one's home or office. However, opening and closing curtains or blinds all the time may seem like a chore. But with automated curtain controllers in place, one can regulate the interior lighting with ease. Moreover, one will be able to cut down the unnatural glare of fluorescent lights and enjoy healthier interior lighting.

Automated curtains also allow one to regulate the temperature inside one's home during cold weather. Most Indian homeowners fail to recognize the importance of letting in sunlight during winters. But with automated curtains, one can warm up the interiors when the sun is out and dial down the use of room heaters to save energy and cost.

Safety

Smart curtain controllers can be lifesaving in case of fires, allowing smoke to leave the interior, alerting onlookers, and letting emergency response teams look in for rescue assessments. Automated curtain controllers in tandem with smart lights can be used to ward off burglars conducting recon on an empty home or office.

So, smart automatic curtain system can make one's life better, healthier, and safer.

So this Arduino Curtain Automation project will let us automate our curtain blinds using just an Arduino and a stepper motor.

If one's dorm is very dark, they can brighten up their room when they wanted to.

So one can decide to make this DIY Arduino Curtain Automation system.

Using regular curtain blinds, one could control the amount of light in the room and suit it according to their needs.

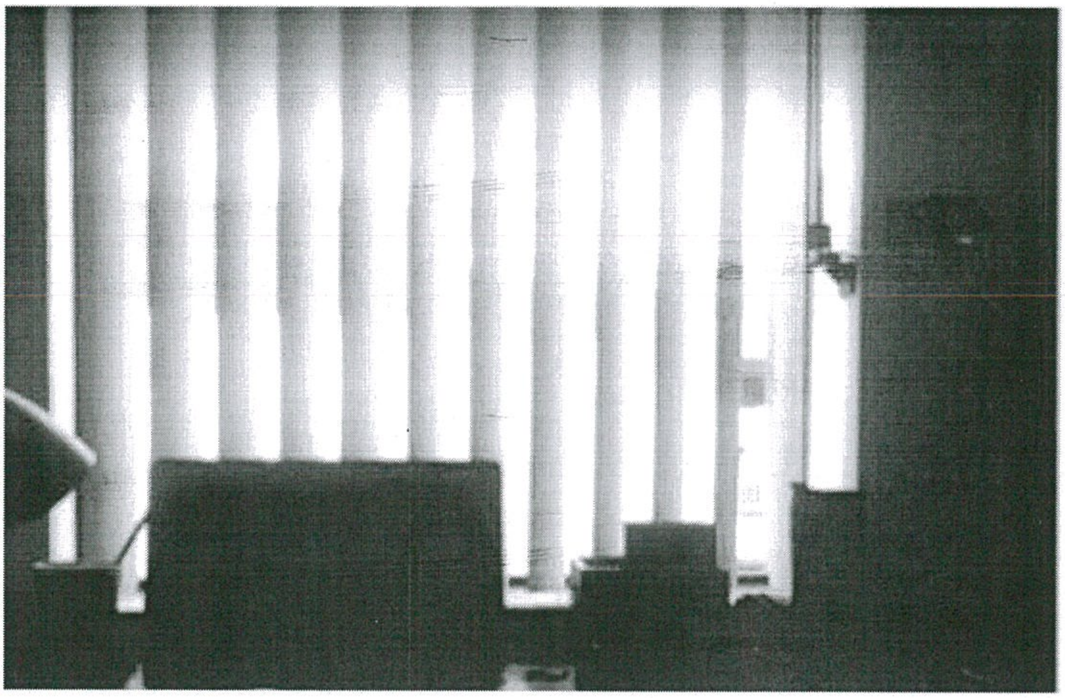


Fig. 1

Materials Required -

- Arduino
- Adafruit Sound Sensor (optional)
- Stepper Motor + Darlington array driver
- Buttons
- LED for indication
- Jumper Wires
- Breadboard

In this Arduino Curtain Automation system there are two ways to activate the curtains:

- Using a sound sensor (mic) to control it using my claps.
- Using buttons to move the shades of the curtain.

Information about the components

ARDUINO –

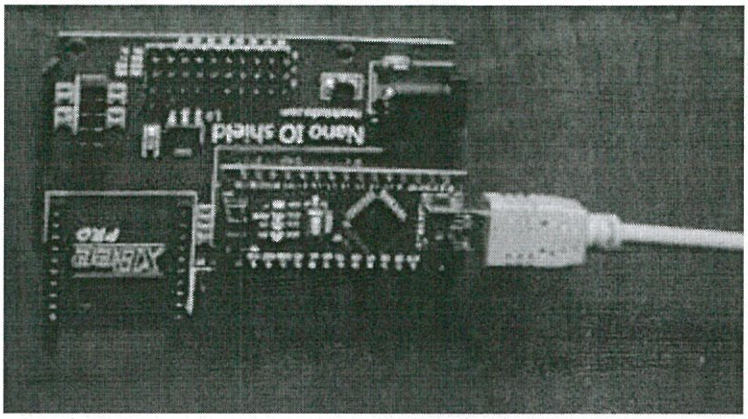
The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328P released in 2008.

It offers the same connectivity and specs of the Arduino Uno board in a smaller form factor.

The Arduino Nano is equipped with 30 male I/O headers, in a DIP-30-like configuration, which can be programmed using the Arduino Software integrated development environment (IDE), which is common to all Arduino boards and running both online and offline.

The board can be powered through a type-B mini-USB cable or from a 9 V battery.

In 2019, Arduino released the **Arduino Nano Every**, a pin-equivalent evolution of the Nano. It features a more powerful ATmega4809 processor and twice the RAM.



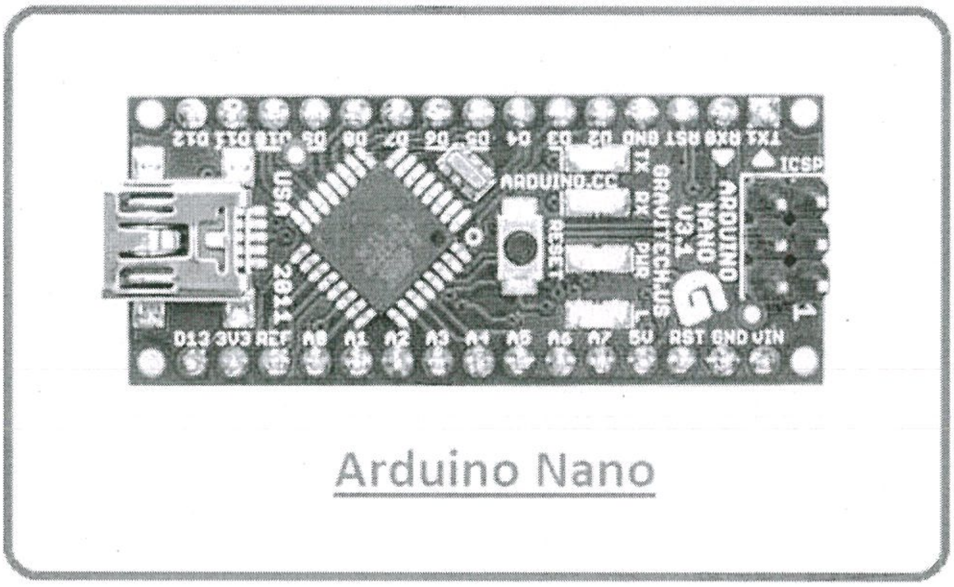

D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
ICSP
ARDUINO.CC
ARDUINO NANO V3.1
GRAVITECH.US
TX
RX
PWR
L
RESET
USA
2011
1
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
Arduino Nano

Fig. 2

ARDUINO NANO PINOUT DESCRIPTION –

In this section, we'll cover the Arduino Nano Pinout, we will discuss pin description of each pin integrated on the board.

Digital Pins: There are 14 digital pins on board which is used to connect external component.

Analog Pins: 6 analog pins on board that is used to measure voltage in a range from 0 to 5V.

LED: The unit comes with a built-in LED connected to pin 13 on the board.

VIN: This is an input voltage to the Arduino board when using an external power source (6-12V).

3.3V: It is a minimum voltage produced by the voltage regulator on the board.

5V: Regulated power supply used to power up the controller and other components on board.

AREF: It is an Analog Reference that is applied to the unit as a reference voltage from an external power supply.

GND: Two ground pins are available on the board.

Reset: Two reset pins are integrated on the board. These pins are used to reset the controller internally through software.

External Interrupts: Pin 2 and 3 are used to trigger external interrupts. These pins are used in case of emergency.

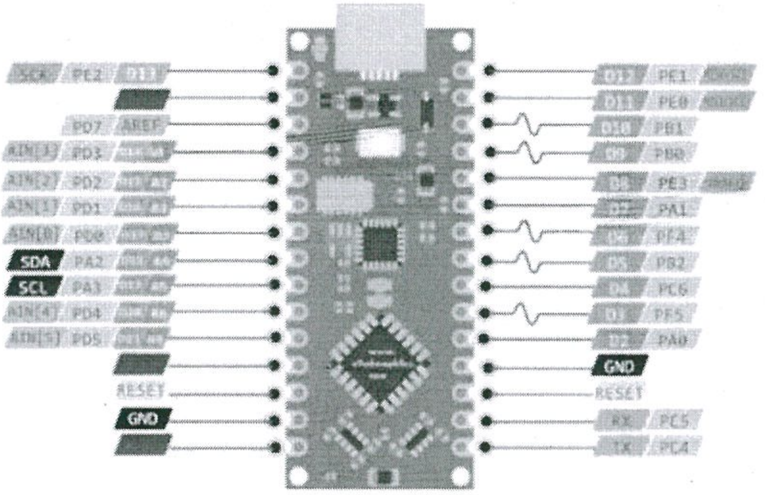
USART: The board supports USART serial communication that carries two pins i.e. Rx which is used for receiving the serial data and Tx which is a transmission pin used to transmit serial data.

I2C: The unit comes with an I2C communication protocol where two pins SDA and SCL are used to support this communication. SDA is a serial data line that carries the data while SCL is a serial clock line used for data synchronization between the devices on the I2C bus. The Wire Library of Arduino Software can be accessed to use the I2C bus.

SPI: The device also supports SPI (serial peripheral interface) communication protocol where four pins (SS, MISO, MOSI, SCK) are used for this communication. This protocol is used to transfer data between the microcontroller and other peripheral devices.

ARDUINO NANO PINOUT –

The following figure shows the pinout diagram of the Arduino Nano board –



Arduino Nano Pinout

TECHNICAL SPECIFICATION –

- Microcontroller: Microchip ATmega328P
- Operating voltage: 5 volts
- Input voltage: 6 to 20 volts

- Digital I/O pins: 14 (6 optional PWM outputs)
- Analog input pins: 8
- DC per I/O pin: 40 mA
- DC for 3.3 V pin: 50 mA
- Flash memory: 32 KB, of which 0.5 KB is used by bootloader
- SRAM: 2 KB
- EEPROM: 1 KB
- Clock speed: 16 MHz
- Length: 45 mm
- Width: 18 mm
- Mass: 7 g
- USB: Mini-USB Type-B
- ICSP Header: Yes
- DC Power Jack: No

COMMUNICATION –

The Arduino Nano has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers.

The ATmega328 provide UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An FTDI FT232RL on the board channels this serial communication over USB and the FTDI drivers (included with the Arduino software) provide a virtual com port to software on the computer.

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 FTDI chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A Software Serial library allows for serial communication on any of the Nano's digital pins. The ATmega328 also support I2C and SPI communication.

The Arduino software includes a Wire library to simplify use of the I2C bus.

AUTOMATIC (SOFTWARE) RESET –

Rather than requiring a physical press of the reset button before an upload, the Arduino Nano is designed in a way that allows it to be reset by software running on a connected computer.

One of the hardware flow control lines (DTR) of the FT232RL is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor.

When this line is asserted (taken low), the reset line drops long enough to reset the chip.

This setup has other implications.

When the Nano is connected to a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB).

For the following half-second or so, the bootloader is running on the Nano.

While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened.

STEPPER MOTOR –

Stepper Motor Basics -

A stepper motor is an electric motor whose main feature is that its shaft rotates by performing steps, that is, by moving by a fixed amount of degrees.

This feature is obtained thanks to the internal structure of the motor, and allows to know the exact angular position of the shaft by simply counting how may steps have been performed, with no need for a sensor.

This feature also makes it fit for a wide range of applications.

Stepper Motor Working Principles -

As all with electric motors, stepper motors have a stationary part (the stator) and a moving part (the rotor).

On the stator, there are teeth on which coils are wired, while the rotor is either a permanent magnet or a variable reluctance iron core.

We will dive deeper into the different rotor structures later.

Figure 3 shows a drawing representing the section of the motor is shown, where the rotor is a variable-reluctance iron core.

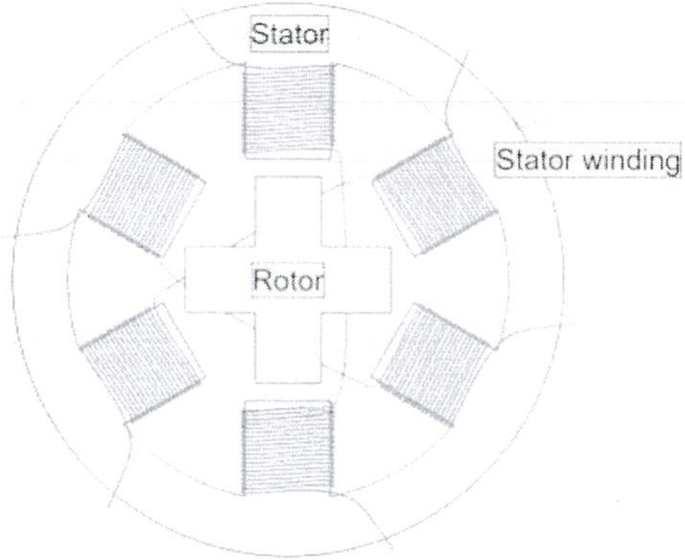


Fig. 3 Cross-Section of a Stepper Motor

The basic working principle of the stepper motor is the following: By energizing one or more of the stator phases, a magnetic field is generated by the current flowing in the coil and the rotor aligns with this field.

By supplying different phases in sequence, the rotor can be rotated by a specific amount to reach the desired final position.

Figure 4 shows a representation of the working principle. At the beginning, coil A is energized and the rotor is aligned with the magnetic field it produces.

When coil B is energized, the rotor rotates clockwise by 60° to align with the new magnetic field.

The same happens when coil C is energized. In the pictures, the colors of the stator teeth indicate the direction of the magnetic field generated by the stator winding.

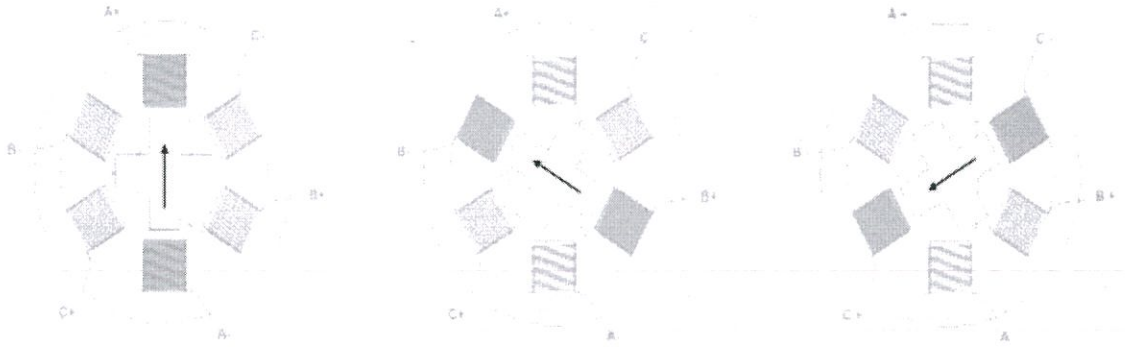


Fig. 4 Stepper Motor Steps

Stepper Motor Types and Construction -

The performance of a stepper motor — both in terms of resolution (or step size), speed, and torque — is influenced by construction details, which at the same time may also affect how the motor can be controlled.

As a matter of fact, not all stepper motors have the same internal structure (or construction), as there are different rotor and stator configurations.

Rotor –

For a stepper motor, there are basically three types of rotors:

Permanent magnet rotor: The rotor is a permanent magnet that aligns with the magnetic field generated by the stator circuit. This solution guarantees a good torque and also a detent torque. This means the motor will resist, even if not very strongly, to a change of position regardless of whether a coil is energized. The drawbacks of this solution is that it has a lower speed and a lower resolution compared to the other types. **Figure 5** shows a representation of a section of a permanent magnet stepper motor.

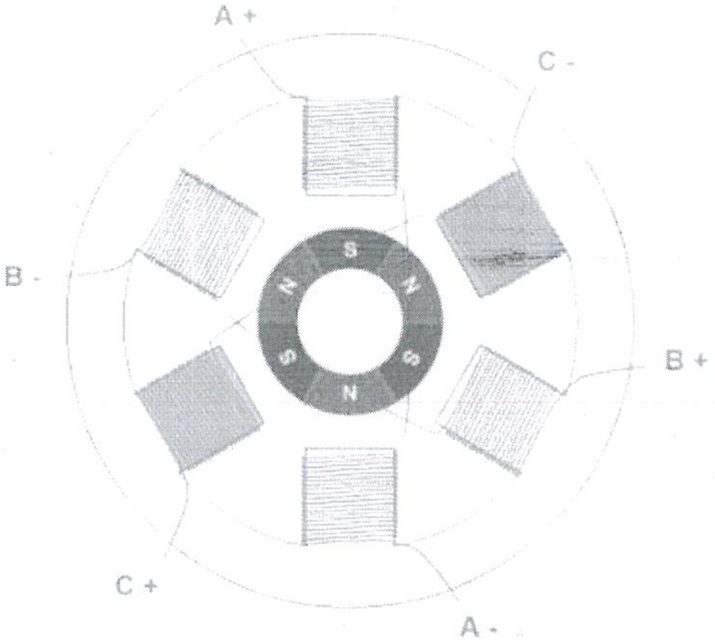


Figure 5: Permanent Magnet Stepper Motor

Variable reluctance rotor: The rotor is made of an iron core, and has a specific shape that allows it to align with the magnetic field (see Figure 3 and Figure 4).

With this solution it is easier to reach a higher speed and resolution, but the torque it develops is often lower and it has no detent torque.

Hybrid rotor: This kind of rotor has a specific construction, and is a hybrid between permanent magnet and variable reluctance versions. The rotor has two caps with alternating teeth, and is magnetized axially. This configuration allows the motor to have the advantages of both the permanent magnet and variable reluctance versions, specifically high resolution, speed, and torque. This higher performance requires a more complex construction, and therefore a higher cost. Figure 5 shows a simplified example of the structure of this motor. When coil A is energized, a tooth of the N-magnetized cap aligns with the S-magnetized tooth of the stator. At the same time, due to the rotor structure, the S-magnetized tooth aligns with the N-magnetized tooth of the stator. Real motors have a more complex structure, with a higher number of teeth than the one shown in the picture, though the working principle of the stepper motor is the same. The high number of teeth allows the motor to achieve a small step size, down to 0.9°.

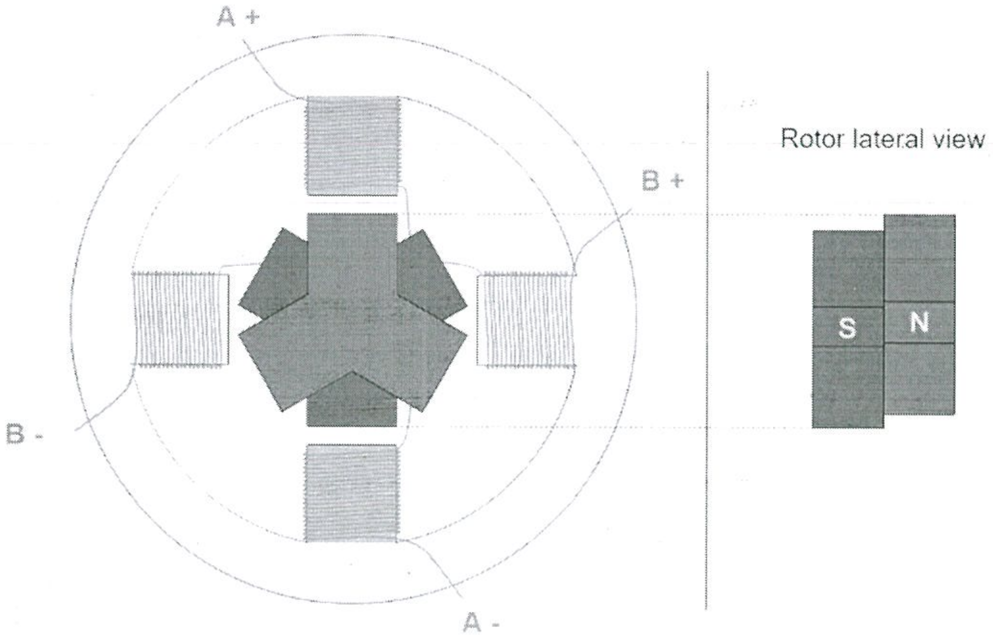


Figure 6: Hybrid Stepper Motor

Stator- The stator is the part of the motor responsible for creating the magnetic field with which the rotor is going to align. The main characteristics of the stator circuit include its number of phases and pole pairs, as well as the wire configuration. The number of phases is the number of independent coils, while the number of pole pairs indicates how main pairs of teeth are occupied by each phase. Two-phase stepper motors are the most commonly used, while three-phase and five-phase motors are less common (see Figure 7 and Figure 8).

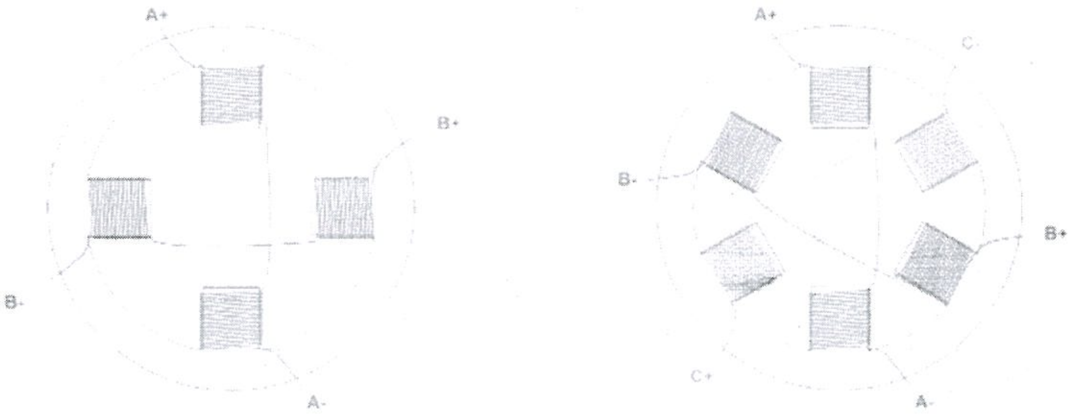


Figure 7: Two-Phase Stator Winding (Left), Three-Phase Stator Winding (Right)

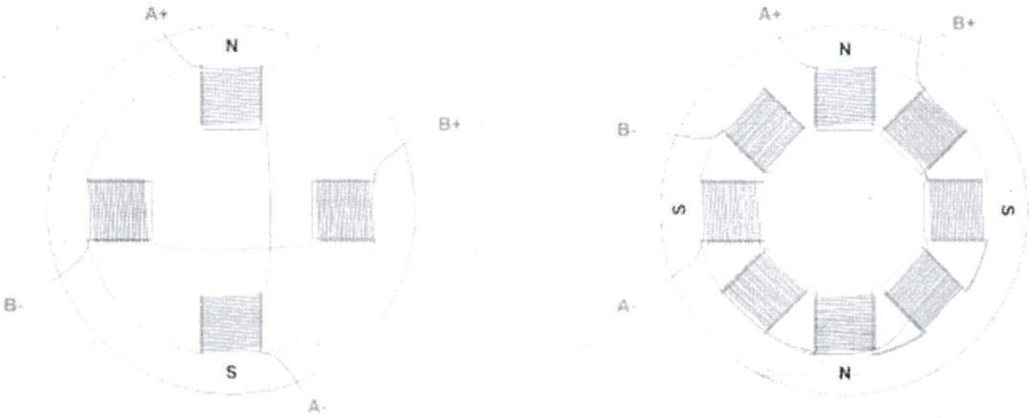


Figure 8: Two-Phase, Single-Pole Pair Stator (Left) and Two-Phase, Dipole Pair Stator (Right). The Letters Show the Magnetic Field Generated when Positive Voltage is Applied between A+ and A-

Stepper Motor Control- We have seen previously that the motor coils need to be energized, in a specific sequence, to generate the magnetic field with which the rotor is going to align. Several devices are used to supply the necessary voltage to the coils, and thus allow the motor to function properly. Starting from the devices that are closer to the motor we have:

- A transistor bridge is the device physically controlling the electrical connection of the motor coils. Transistors can be seen as electrically controlled interrupters, which, when closed allow the connection of a coil to the electrical supply and thus the flow of current in the coil. One transistor bridge is needed for each motor phase.
- A pre-driver is a device that controls the activation of the transistors, providing the required voltage and current, it is in turn controlled by an MCU.
- An MCU is a microcontroller unit, which is usually programmed by the motor user and generates specific signals for the pre-driver to obtain the desired motor behaviour.

Figure 9 shows a simple representation of a stepper motor control scheme. The pre-driver and the transistor bridge may be contained in a single device, called a **driver**.

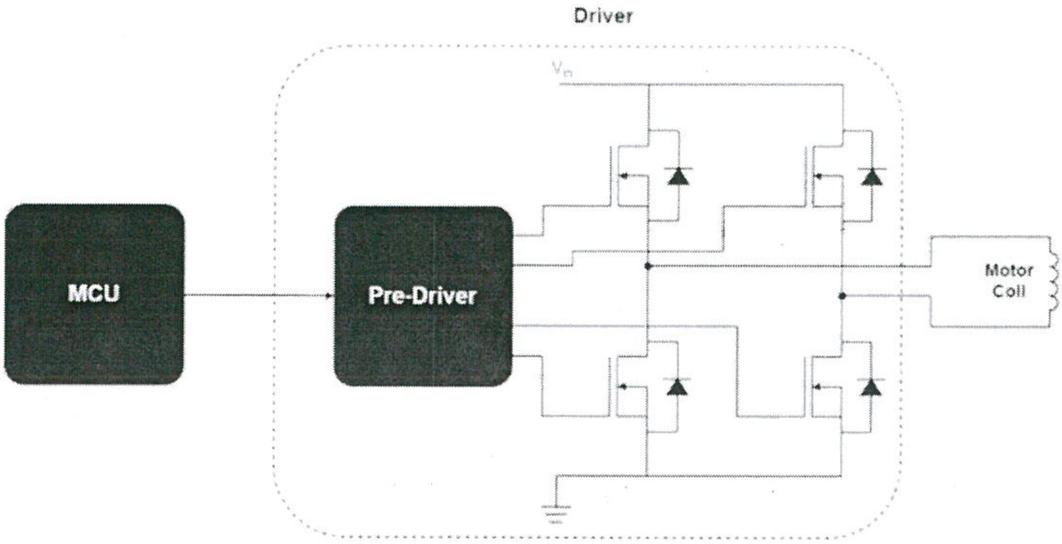


Figure 9: Motor Control Basic Scheme

Stepper Motor Driver Types -

There are different stepper motor drivers available on the market, which showcase different features for specific applications. The most important charactreristics include the input interface. The most common options are:

- Step/Direction –

  By sending a pulse on the Step pin, the driver changes its output such that the motor will perform a step, the direction of which is determined by the level on the Direction pin.

- Phase/Enable –

  For each stator winding phase, Phase determines the current direction and triggers Enable if the phase is energized.

- PWM –

  Directly controls the gate signals of the low-side and high-side FETs.

Another important feature of a stepper motor driver is if it is only able to control the voltage across the winding, or also the current flowing through it:

- With voltage control, the driver only regulates the voltage across the winding. The torque developed and the speed with which the steps are executed only depend on motor and load characteristics.
- Current control drivers are more advanced, as they regulate the current flowing through the active coil in order to have better control over the torque produced, and thus the dynamic behavior of the whole system.

Unipolar/Bipolar Motors

Another feature of the motor that also affects control is the arrangement of the stator coils that determine how the current direction is changed. To achieve the motion of the rotor, it is necessary not only to energize the coils, but also to control the direction of the current, which determines the direction of the magnetic field generated by the coil itself (see **Figure 10**).

In stepper motors, the issue of controlling the current direction is solved with two different approaches.

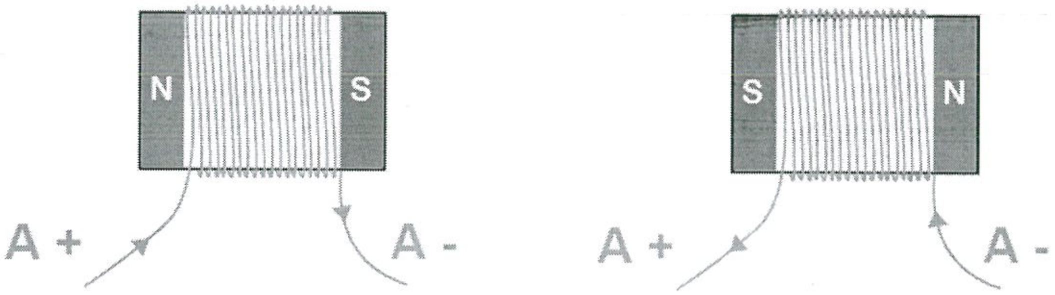


Figure 10: Direction of the Magnetic Field based on the Direction of the Coil Current

In **unipolar stepper motors**, one of the leads is connected to the central point of the coil (see **Figure 11**). This allows to control the direction of the current using relatively simple circuit and components. The central lead (A_M) is connected to the input voltage V_{IN} (see **Figure 10**). If MOSFET 1 is active, the current flows from A_M to A+. If MOSFET 2 is active, current flows from A_M to A-, generating a magnetic field in the opposite direction. As pointed out above, this approach allows a simpler driving circuit (only two semiconductors needed), but the drawback is that only half of the copper used in the motor is used at a time, this means that for the same current flowing in the coil, the magnetic field has half the intensity compared if all the copper were used. In addition, these motors are more difficult to construct since more leads have to be available as motor inputs.

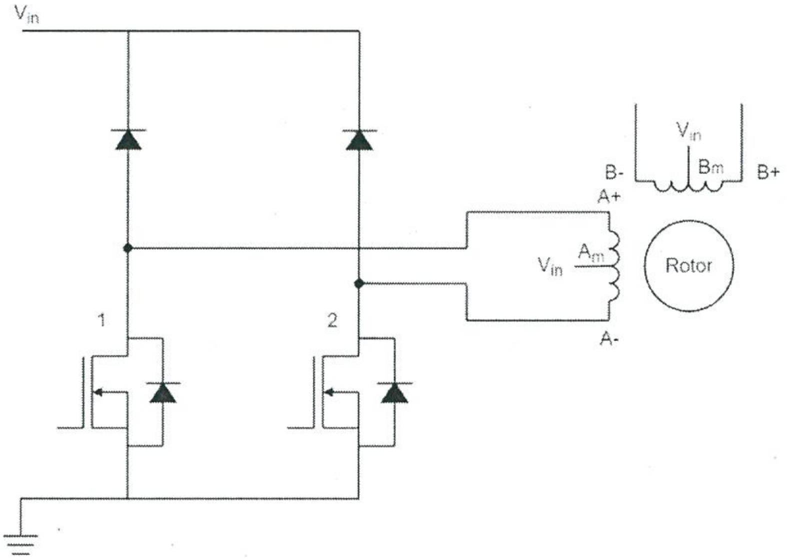


Figure 11: Unipolar Stepper Motor Driving Circuit

In **bipolar stepper motors**, each coil has only two leads available, and to control the direction it is necessary to use an H-bridge (see **Figure 12**). As shown in **Figure 10**, if MOSFETs 1 and 4 are active, the current flows from A+ to A-, while if MOSFETs 2 and 3 are active, current flows from A- to A+, generating a magnetic field in the opposite direction. This solution requires a more complex driving circuit, but allows the motor to achieve the maximum torque for the amount of copper that is used.

With technology progress, the advantages of unipolar are becoming less relevant, and bipolar steppers are currently the most popular.

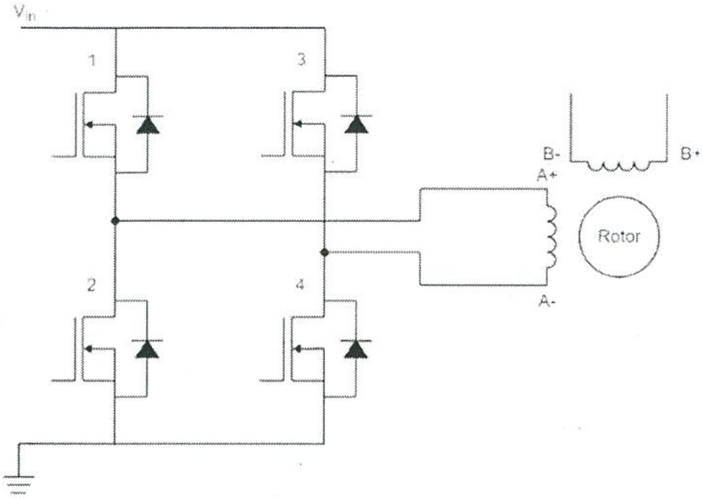


Fig. 12 Bipolar Stepper Motor Driving Circuit

Stepper Motor Driving Techniques -

There are four different driving techniques for a stepper motor:

- In **wave mode**, only one phase at a time is energized. For simplicity, we will say that the current is flowing in a positive direction if it is going from the + lead to the - lead of a phase (e.g. from A+ to A-); otherwise, the direction is negative. Starting from the left, the current is flowing only in phase A in the positive direction and the rotor, represented by a magnet, is aligned with the magnetic field generated by it. In the next step, it flows only in phase B in the positive direction, and the rotor spins 90° clockwise to align with the magnetic field generated by phase B. Later, phase A is energized again, but the current flows in the negative direction, and the rotor spins again by 90°. In the last step, the current flows negatively in phase B and the rotor spins again by 90°.

- In **full-step mode**, two phases are always energized at the same time. This shows the different steps of this driving mode. The steps are similar to the wave mode ones, the most significant difference being that with this mode, the motor is able to produce a higher torque since more current is flowing in the motor and a stronger magnetic field is generated.

- **Half-step mode** is a combination of wave and full-step modes. Using this combination allows for the step size to be reduced by half (in this case, 45° instead of 90°). The only drawback is that the torque produced by the motor is not constant, since it is higher when both phases are energized, and weaker when only one phase is energized.

- **Micro stepping** can be seen as a further enhancement of half-step mode, because it allows to reduce even further the step size and to have a constant torque output. This is achieved by controlling the intensity of the current flowing in each phase. Using this mode requires a more complex motor driver compared to the previous solutions. This shows how micro stepping works. If I_{MAX} is the maximum current that can flow in a phase, starting from the left, in the first figure $I_A = I_{MAX}$ and $I_B = 0$. In the next step, the currents are controlled to achieve $I_A = 0.92 \times I_{MAX}$ and $I_B = 0.38 \times I_{MAX}$, which generates a magnetic field that is rotated by 22.5° clockwise compared to the previous one. This step is repeated with different current values to reach the 45°, 67.5°, and 90° positions. This provides the ability to reduce by half the size of the step, compared to the half-step mode; but it is possible to go even further. Using micro stepping helps reaching very high position resolution, but this advantage comes at the cost of a more complex device to control the motor, and a smaller torque generated with each step. Indeed, the torque is proportional to the sine of the angle between the stator magnetic field and the rotor magnetic field; therefore, when the steps are smaller, the torque is smaller. This may lead to missing some steps, meaning the rotor position does not change even if the current in the stator winding has.

DARLINGTON ARRAY DRIVER –

Darlington devices are high-voltage, high-current switch arrays containing multiple open-collector Darlington pairs or multiple Darlington transistors with common emitters, and integral suppression diodes for inductive loads.

The nominal current rating of each output is 500 mA for the 7- and 8-output devices and up to 1.5 A for quad-output devices.

The inputs are pinned opposite the outputs to simplify the application board layout. These devices interface standard logic families.

ADAFRUIT SOUND SENSOR –

A thin MEMS microphone can be used to detect audio levels and even perform basic FFT functions.

One can read the analog voltage corresponding to the audio on analog pin **#A4**. Note that this is the raw analog audio waveform!

When it's silent there will be a reading of ~330 and when loud the audio will read between 0 and 800 or so.

Averaging and smoothing must be done to convert this to sound-pressure-level.

The microphone is sensitive to 100 Hz - 10,000 Hz audio frequencies.
This microphone is a bit different than some used in Arduino projects.
Instead of an analog microphone, which requires an external op-amp and level management, the Circuit Playground Express uses a PDM microphone.
This is a digital mic which is a lot smaller and less expensive.
We will need to use MakeCode/CircuitPython/Arduino support libraries to read the audio, one cannot read it like an analog voltage.

LED –

A **light-emitting diode** (**LED**) is a semiconductor light source that emits light when current flows through it.

Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons.

The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the band gap of the semiconductor.

White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.

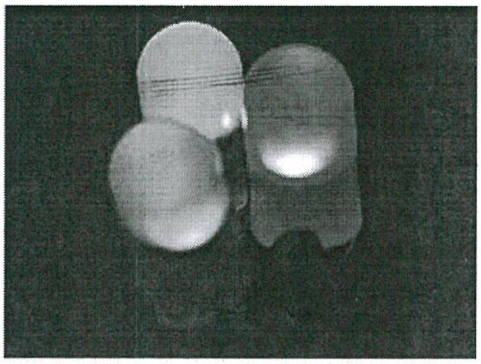


Fig. 13 LED

The design for this Arduino Curtain Automation system is quite simple and there are two ways to activate the curtains:

Using a sound sensor (mic) to control it using my claps.

Using buttons to move the shades of the curtain.

So let's get started with our DIY Arduino Curtain Automation system, following the simple instructions to replicate this project.

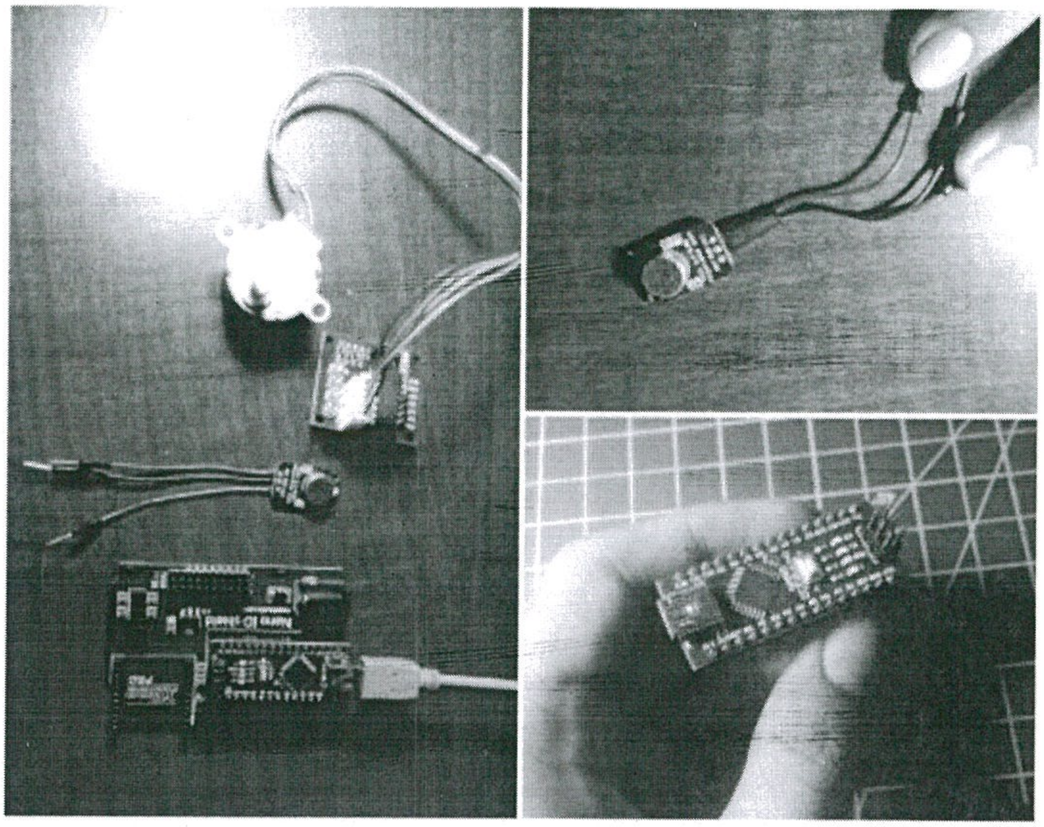


Fig. 14

Let us see how does the Arduino Curtain Automation Work.

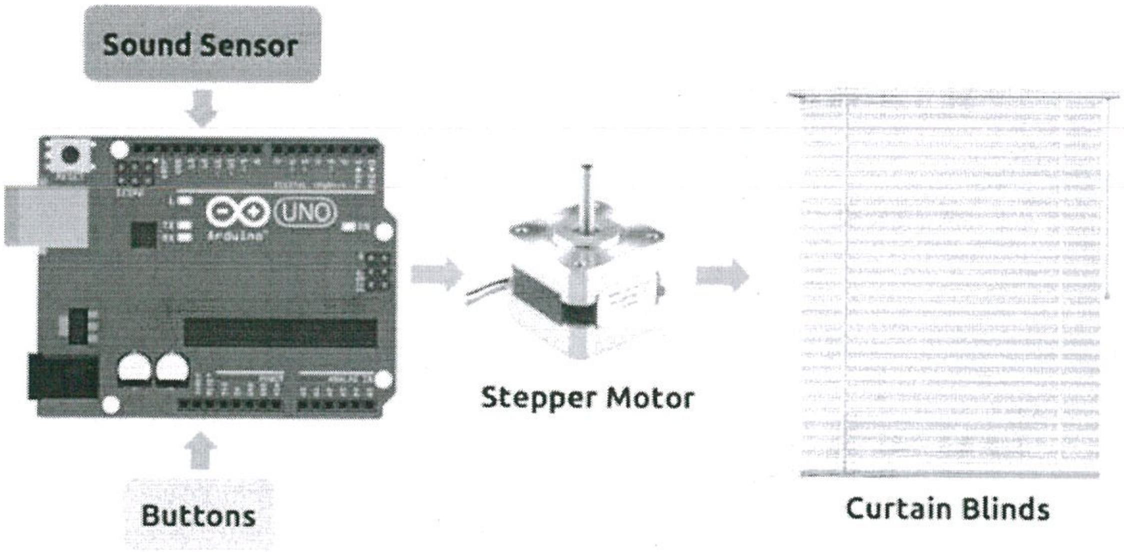


Fig. 15

The working of this DIY Arduino Curtain Automation system is very simple.

The Arduino takes input from either the sound sensor (mic) or buttons.

It then correspondingly controls the stepper motor through the Darlington array driver for the motor.

The stepper motor is attached to the control stick of the curtain blinds and thus on rotation, opens/closes the blinds.

Interfacing the Stepper Motor –

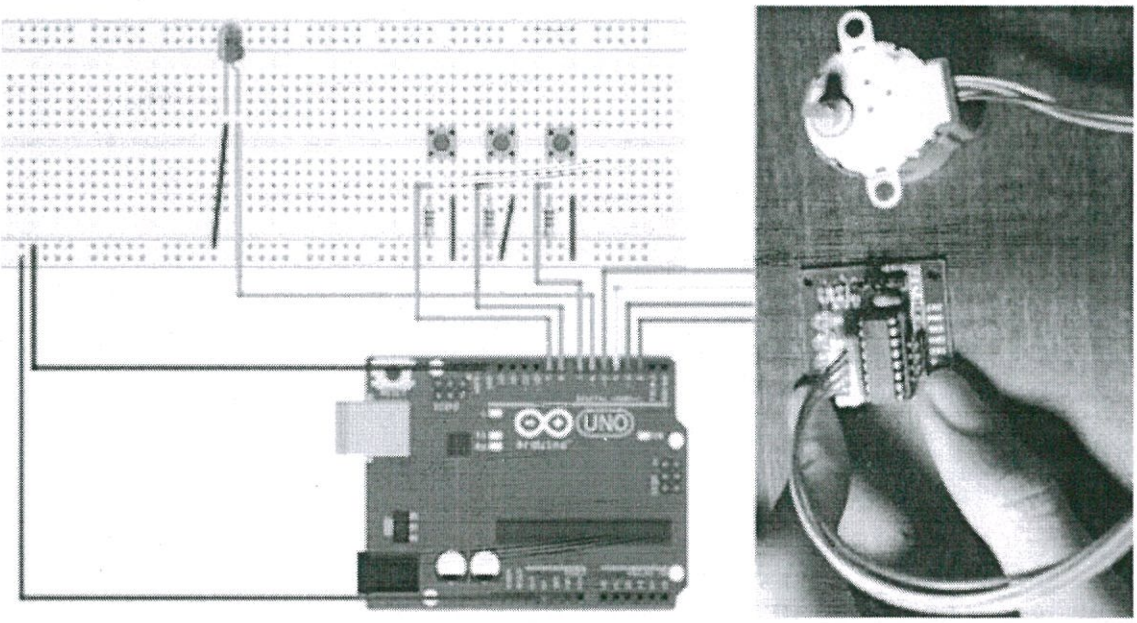


Fig. 16

The first step I did was testing the design for the Arduino curtain automation system.

The motor is connected to the 4 wires of the driver as shown in the diagram. Depending on the direction of rotation for one's curtain, one need to plug the driver accordingly.

Basically, the first button from the right will activate the motor to spin a particular degree (one can change in the code).

Press the button 4 times and it will return to the original place since it will spin around 90 degrees during each press.

The middle button will lock the circuit so that the first button cannot activate the motor.

The LED will turn ON when the motor is locked.

The final button will return the motor to the original place no matter where is it, at the time pressed.

Code is given below-

```
int pin[8]={2,3,4,5,6,7,8,9};
int steps[][4] =
{
{HIGH,HIGH,LOW,LOW},
{HIGH,LOW,LOW,HIGH},
{LOW,LOW,HIGH,HIGH},
{LOW,HIGH,HIGH,LOW},
} ;

int numofroun=1;

int current=1;
int type=3;
int place=0;

int lastLockState = LOW;
long lastLockTime = 0;
int LockState;
int Lockreading;
bool lock=true;

int lastPauseState = LOW;
long lastPauseTime = 0;
int PauseState;
int Pausereading;
bool Pauseled=false;
bool pause=false;

int lastReturnState = LOW;
long lastReturnTime = 0;
int ReturnState;
int Returnreading;
```

```
void setup() {
for (int num=0; num<5; num++) pinMode(pin[num],OUTPUT);
for (int num=5; num<8; num++) pinMode(pin[num],INPUT);
}

void reset(){
for(int num=0;num<4;num++) digitalWrite(pin[num],LOW);
}

void stepper()
{
for (int num=0; num<4;num++) { digitalWrite(pin[num],steps[abs(type-
current)][num]);} if(type==0) {++place;} if(type==3) {--place;} delay(2); } void
button1() { Lockreading = digitalRead(pin[5]); if (Lockreading != lastLockState) {
lastLockTime = millis(); } if ((millis() - lastLockTime) > 50)
{
if (Lockreading != LockState) {
LockState = Lockreading;
if (LockState == HIGH) {
lock=false;
if ((place!=1536*numofroun)&&(place!=1024*numofroun)&&(place!=512*numofroun))
{type=abs(type-3);}
}
}
}
lastLockState = Lockreading;
}

void button2()
{
Pausereading = digitalRead(pin[6]);
if (Pausereading != lastPauseState)
{
lastPauseTime = millis();
}
if ((millis() - lastPauseTime) > 50)
{
if (Pausereading != PauseState) {
PauseState = Pausereading;
if (PauseState == HIGH) {
Pauseled=!Pauseled;
pause=!pause;
if (Pauseled) {digitalWrite(pin[4],HIGH);}
if (!Pauseled) {digitalWrite(pin[4],LOW);}
}
}
}
lastPauseState = Pausereading;
}

void button3()
{
Returnreading = digitalRead(pin[7]);
if (Returnreading != lastReturnState)
{
lastReturnTime = millis();
}
if ((millis() - lastReturnTime) > 50)
{
if (Returnreading != ReturnState) {
ReturnState = Returnreading;
if (ReturnState == HIGH) {
type=3;
while (place>0)
{
for (int num=0; num<4;num++) {
digitalWrite(pin[num],steps[3-current][num]);}
--place;
if (current==3) {current=0;}
else ++current;
delay(2);
}
reset();
}
}
}
lastReturnState = Returnreading;
}

void loop() {
if (lock==true) {button2();button3();}
if (!pause)
{
if (lock==true) {button1();}
if (lock==false) {stepper();}
if
((place==2048)or(place==0)or(((place==1536*numofroun)or(place==1024*numofroun)or(place==512*numofroun))&&(type==3)))
{lock=true;reset();}
if (current==3) {current=0;}
else ++current;
}
        }
```

Setting Up the Arduino Curtain Automation System –

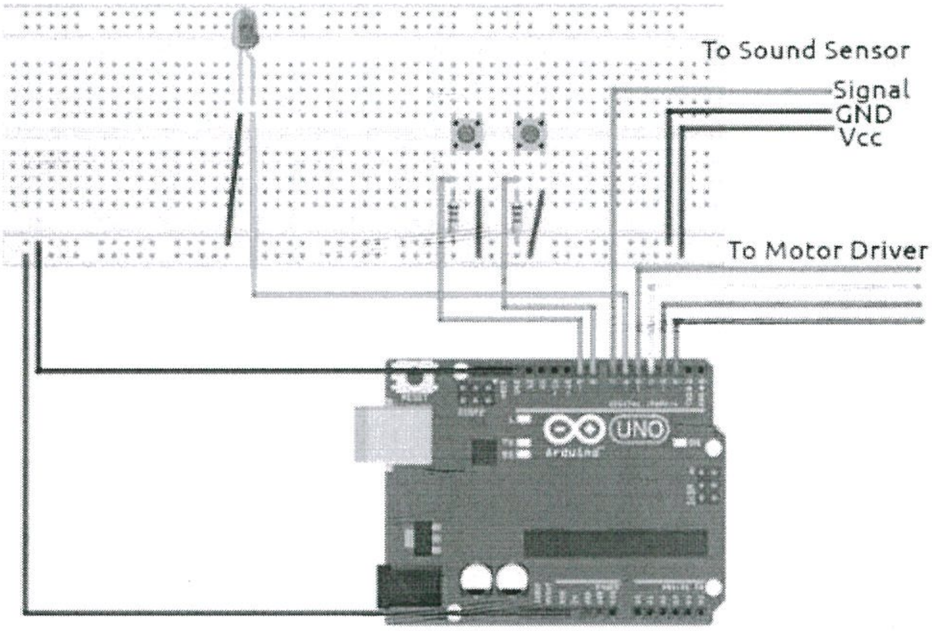


Fig. 17

After testing the stepper motor, one can use the connection diagram above to make the final prototype. After finished making, simply change the input (button) into a sound sensor.

The code is given below.

```
int pin[8]={2,3,4,5,6,7,8,9};

int steps[][4] =
{
{HIGH,HIGH,LOW,LOW},
{HIGH,LOW,LOW,HIGH},
{LOW,LOW,HIGH,HIGH},
{LOW,HIGH,HIGH,LOW},
} ;

float numofroun=4.5;

int current=1;
int type=3;
int place=0;
int claps = 0;
long detectionSpanInitial = 0;
long detectionSpan = 0;
long spancondition;
bool spanconditioncheck=false;

bool lock=true;

int lastPauseState = LOW;
long lastPauseTime = 0;
int PauseState;
int Pausereading;
bool Pauseled=false;
bool pause=false;

int lastReturnState = LOW;
long lastReturnTime = 0;
int ReturnState;
int Returnreading;

void setup() {
for (int num=0; num<5; num++) pinMode(pin[num],OUTPUT);
for (int num=5; num<8; num++) pinMode(pin[num],INPUT);
}

void reset(){
for(int num=0;num<4;num++) digitalWrite(pin[num],LOW);
}


void stepper()
{
for (int num=0; num<4;num++) { digitalWrite(pin[num],steps[abs(type-
current)][num]);} if(type==0) {++place;} if(type==3) {--place;} delay(2); } void sound()
{ int sensorState = digitalRead(pin[5]); if (sensorState == 0){ if (claps == 0){
detectionSpanInitial = detectionSpan = millis(); claps++; } else if (claps > 0 &&
millis()-detectionSpan >= 50){
detectionSpan = millis();
claps++;
}
}
if (millis()-detectionSpanInitial >= 400)
{
if (claps == 2)
{
lock=false;
if ((place!=1024*numofroun)&&(place!=512*numofroun)) {type=abs(type-3);}
spancondition=millis();
}
claps = 0;
}
}

void button1()
{
Pausereading = digitalRead(pin[6]);
if (Pausereading != lastPauseState)
{
lastPauseTime = millis();
}
if ((millis() - lastPauseTime) > 50)
{
if (Pausereading != PauseState) {
PauseState = Pausereading;
if (PauseState == HIGH) {
Pauseled=!Pauseled;
pause=!pause;
spancondition=millis();
if (Pauseled) {digitalWrite(pin[4],HIGH);}
if (!Pauseled) {digitalWrite(pin[4],LOW);}
}
}
}
lastPauseState = Pausereading;
}

void button2()
{
Returnreading = digitalRead(pin[7]);
if (Returnreading != lastReturnState)
{
lastReturnTime = millis();
}
if ((millis() - lastReturnTime) > 50)
{
if (Returnreading != ReturnState) {
ReturnState = Returnreading;
if (ReturnState == HIGH) {
type=3;
while (place>0)
{
for (int num=0; num<4;num++) { digitalWrite(pin[num],steps[3-current][num]);} --
place; if (current==3) {current=0;} else ++current; delay(2); } reset();
spancondition=millis(); } } } lastReturnState = Returnreading; } void loop() { if
```

```
(lock==true) {button1();button2();} if (!pause) { if ((lock==true)&&(millis()-
spancondition>1000)) {sound();}
if (lock==false) {stepper();spanconditioncheck=false; }
if
((place==2048*numofroun)or(place==0)or(((place==1024*numofroun)or(place==512*
numofroun))&&(type==3))){
lock=true;
reset();
if (!spanconditioncheck){
spancondition=millis();
spanconditioncheck=true;
}
}
if (current==3) {current=0;}
else ++current;
}
}
```

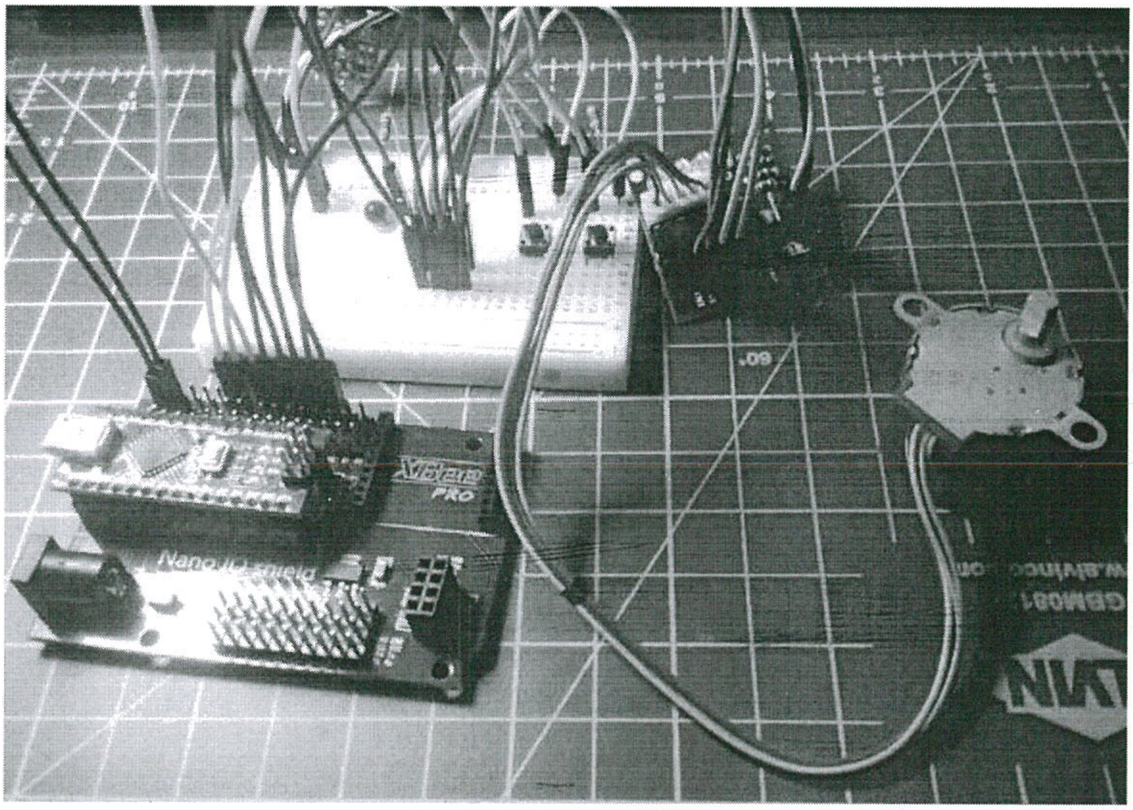


Fig. 18

We used foam board to make the holder for the motor (1, 2) and curtain handle (3) on the Arduino curtain automation system.

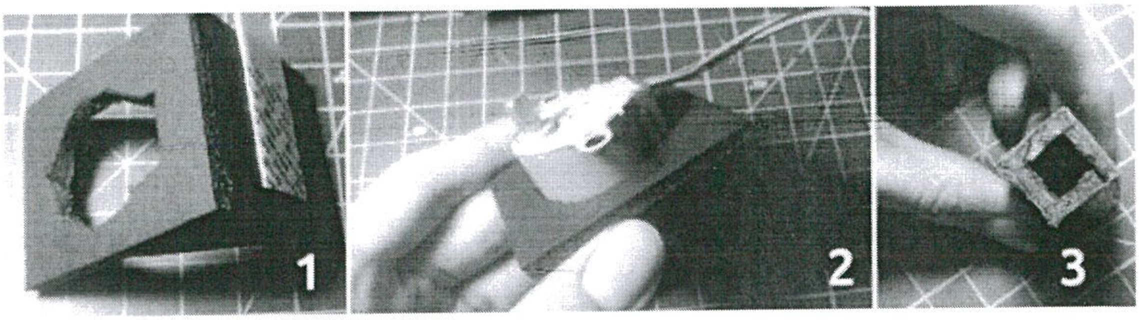


Fig. 19

The pause button is there so that if the room gets too loud, one can lock it so that the blinds won't go crazy.

We might have to adjust the potentiometer on the sound sensor to adjust the sensitivity.

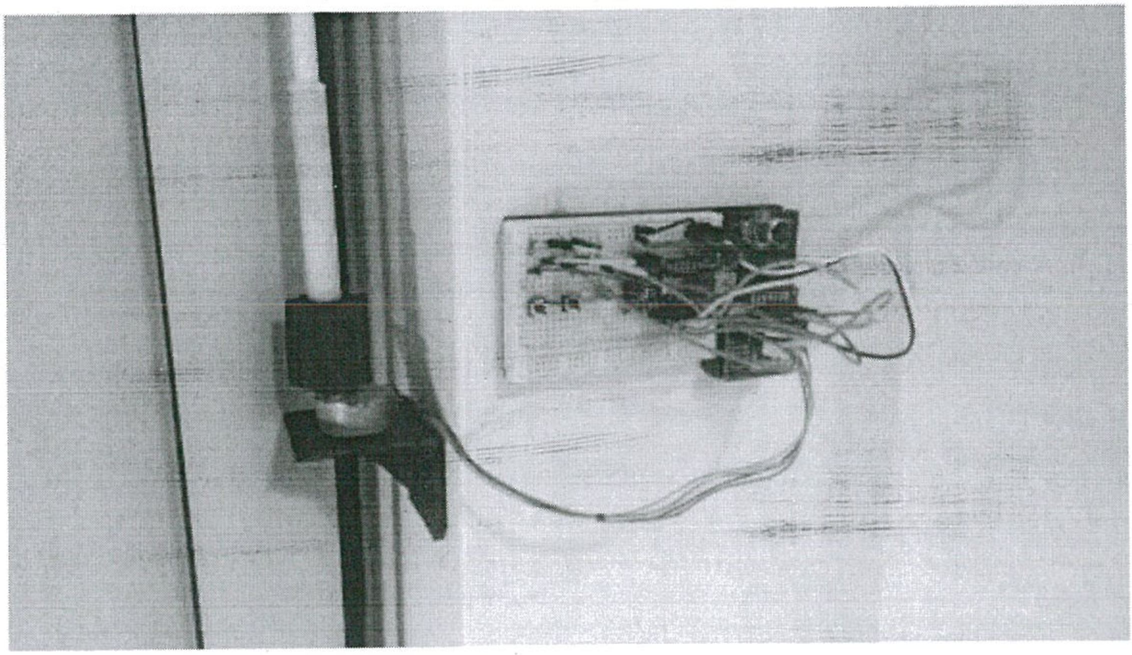


Fig. 20

Alternate Version Using Buttons –

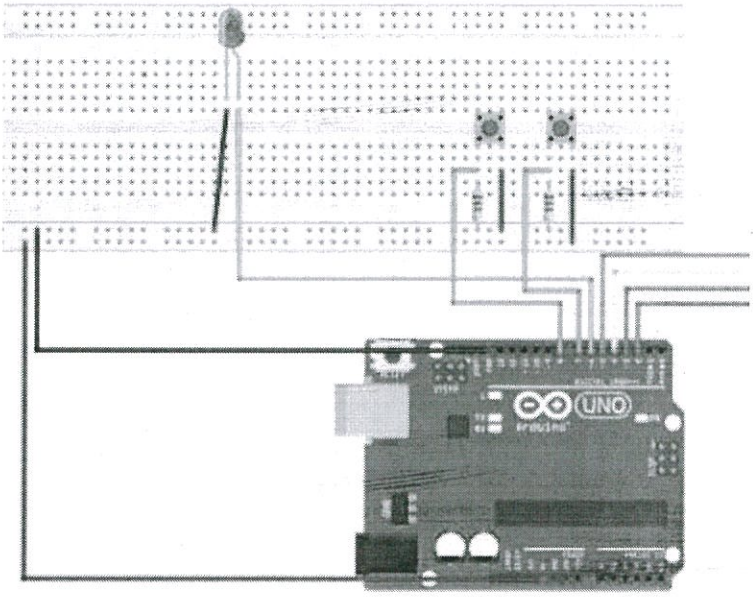


Fig. 21

Instead of sound control, we can simply use a button.

In this Arduino curtain automation design, there are only two buttons: activate and return (since the pause isn't needed).

The activate is the same as before; the reset button will return the curtain back to the original place while activating the LED.

The code for this version is here.

```
int pin[8]={2,3,4,5,6,7,8};

int steps[][4] =
{
{HIGH,HIGH,LOW,LOW},
{HIGH,LOW,LOW,HIGH},
{LOW,LOW,HIGH,HIGH},
{LOW,HIGH,HIGH,LOW},
} ;
int current=1;
int type=3;
int place=0;

int lastLockState = LOW;
```

```
long lastLockTime = 0;
int LockState;
int Lockreading;
bool lock=true;

int lastReturnState = LOW;
long lastReturnTime = 0;
int ReturnState;
int Returnreading;

void setup() {
for (int num=0; num<5; num++) pinMode(pin[num],OUTPUT);
for (int num=5; num<7; num++) pinMode(pin[num],INPUT);
}

void reset(){
for(int num=0;num<4;num++) digitalWrite(pin[num],LOW);
}

void stepper()
{
for (int num=0; num<4;num++) { digitalWrite(pin[num],steps[abs(type-
current)][num]);} if(type==0) {++place;} if(type==3) {--place;} delay(2); } void button1()
{ Lockreading = digitalRead(pin[5]); if (Lockreading != lastLockState) { lastLockTime =
millis(); } if ((millis() - lastLockTime) > 50)
{
if (Lockreading != LockState) {
LockState = Lockreading;
if (LockState == HIGH) {
lock=false;
if ((place!=1024*4)&&(place!=512*4)) {type=abs(type-3);}
}
}
}
lastLockState = Lockreading;
}

void button2()
{
Returnreading = digitalRead(pin[6]);
if (Returnreading != lastReturnState)
{
lastReturnTime = millis();
}
if ((millis() - lastReturnTime) > 50)
{
if (Returnreading != ReturnState) {
ReturnState = Returnreading;
if (ReturnState == HIGH) {
type=3;
digitalWrite(pin[4],HIGH);
while (place>0)
{
for (int num=0; num<4;num++) {
digitalWrite(pin[num],steps[3-current][num]);}
--place;
if (current==3) {current=0;}
else ++current;
delay(2);
}
digitalWrite(pin[4],LOW);
reset();
}
}
}
lastReturnState = Returnreading;
}

void loop() {
if (lock==true) {button1();button2();}
if (lock==false) {stepper();}
if
((place==2048*4)or(place==0)or(((place==1024*4)or(place==512*4))and(type==3))){
lock=true;reset();}
if (current==3) {current=0;}
else ++current;
        }
```

CONCLUSION

The project is done and completed so now one can automate the curtains using the buttons or by using the sound of a clap.