An-Najah National University

Faculty Of Engineering And Information Technology

Computer Engineering Department

# Oxygen Ventilation System

Prepared by:

Yousef Hanbali

Raoof Fares

Supervised by:

Dr.Raed Qadi

May 28, 2023

# Acknowledgements

*"We would like to express our heartfelt gratitude to Dr. Raed Qadi, our supervisor, for his dedication and continuous support throughout our project. His valuable scientific guidance has been invaluable to our success. We would also like to thank the academics in the Department of Computer Engineering for their expertise and assistance, as well as our friends and family for their belief in our abilities. We are deeply grateful for their contributions, support, and belief in our capabilities."*

# Disclaimer

The following report has been authored by students from the Computer Engineering Department, Faculty of Engineering, An-Najah National University. The report has undergone minimal modifications, limited to editorial corrections, and may still contain errors in language and content. It is important to note that the opinions expressed within the report, including any conclusions and recommendations, solely belong to the students. An-Najah National University bears no responsibility or liability for any consequences arising from the utilization of this report for purposes other than its intended commission.

# Contents

# List of Figures

**Abstract**

During COVID Pandemic in 2020, the health system in Palestine collapsed due to the high number of COVID Patients that needed oxygen ventilators and the health system couldn't provide the demanded number of ventilators, which led to a high number of deaths. So we came up with an affordable solution to continue the intensive care in their houses using Oxyilator.

Oxyilator delivers oxygen to the patients as a normal ventilator, also it monitors patient vitals such as SpO2 in the blood, ECG and other important vitals for any patient.

The Breaths per minute can be adjusted manually or using some of the patient characteristics (10-30 BPM). Also, the air volume pushed in each breath and the breath length that is related to the inhalation-to-exhalation ratio. The blood oxygen sensor in the system and ECG measures help the ventilator to operate suitable to the patient's case (regarding the 3 parameters above).

The medical professional assigned can manage the system manually and can get all data related to the patient displayed on a special mobile application.

The ventilator information is displayed on a touch screen attached to the system.

This project was only a prototype during the pandemic, and it was developed as an emergency solution, and it wasn't a professional solution. The system developed was not customizable and can't be used by everyone and also made without a medical reference.

# 1 Introduction

In 2020 when the COVID-19 pandemic spread out along the country, all hospitals and intensive care units were full of people who suffered from COVID-19 and its effects, which led to a collapse in the medical system especially in oxygen ventilators, so that led to many deaths of patients and COVID-19 sick people. So after the pandemic we thought on how to make a ventilation device more accessible to people, to avoid deaths if a pandemic happens again, or even though if a pandemic does not happen, many patients can't reach hospitals, so those also need to access ventilation devices. So we thought to make a portable ventilation device, that can be accessed by anyone, and be used by low to medium medical cases.

## 1.1 Problem Statement

The main problem is the lack of ventilation devices during the pandemic, and even though after the pandemic, the ventilation devices are too expensive and necessary for some patients that can't get to the hospital.

## 1.2 Objectives

Oxylator ventilation device aims to be easy to use, and patients can use it inside their houses, and provide a plug-and-play ventilation device so that no setup is required, and provide reliable medical care to low-medium risk medical cases.

## 1.3 Scope of work

1. **Oxygen and Beats per minute monitor system:** There will be modules to monitor SpO2 in blood and beats per minute for the patient.
2. **Ventilation System:**System responsible for providing ventilation for the patient.
3. **Control System:**System responsible for controlling the variables of the ventilation process. Can be done through a touch screen.
4. **Mobile Application:** There will be a mobile application that can control all variables.

## 1.4 Significance

Oxylator mixes simplicity, technology and ventilation in one place, so that it provides medical care, with simplicity of use, and technologies of IoT and connections to mobile apps, along with a variable controllable ventilation, all of that can be used from home.

# 2 Constraints and Earlier Coursework

## 2.1 Constraints & Limitations

1. Motors torque was a very huge constraint on the project, because it needed high torque to press more on the ambubag, so air volume was not enough.
2. ESP8266 and Arduino MEGA had some problems in serial communication between the two, because ESP8266 only had one serial bus.
3. ECG sensor was not functioning accurately.
4. MAX30100 was not giving accurate results, and even sometimes giving out zeros.

## 2.2 Earlier Coursework

1. Electronics course that provides instruction in various aspects of electronic systems and technologies.
2. Microcontrollers Course covering their fundamental principles and practical applications.
3. Microcontrollers Lab which includes hands-on experience with Arduino and its functionalities, and topics like controlling stepper motor.
4. Critical thinking and scientific research teaching students skills such as reading scientific publications and utilizing modern technologies like LaTeX to produce research papers.

# 3 Literature Review

Oxygen ventilation devices have undergone a significant evolution throughout history. Early concepts of mechanical ventilation[2] led to the invention of positive pressure ventilation devices, which revolutionized respiratory care. Advancements such as microprocessor-controlled ventilators, the introduction of positive end-expiratory pressure (PEEP), and the development of pressure support ventilation (PSV) and non-invasive ventilation (NIV) devices have improved patient outcomes and expanded the applications of ventilators.

The COVID-19 pandemic further highlighted the crucial role of ventilators. Studies[4] after the COVID-19 pandemic showed that 4% of the families that needed medical care, could not get it, which had 11% of them were afraid to leave the house, and 23% couldn't pay for the medical services.

Our system implemented a hybrid ventilation system, using mechanical ventilation as the source of positive pressure, and controlled by microcontrollers, in which we added the integration of new technologies such as IoT, so that it can a cheap ventilation device, with integrated new technologies, which overcomes the problems mentioned previously.

# 4 Methodology

## 4.1 System Architecture

### 4.1.1 Wooden Body

We designed a 50cm*25cm*25cm wooden box, without filling any of it's walls except for one wall, which will be used later to attach the screen.

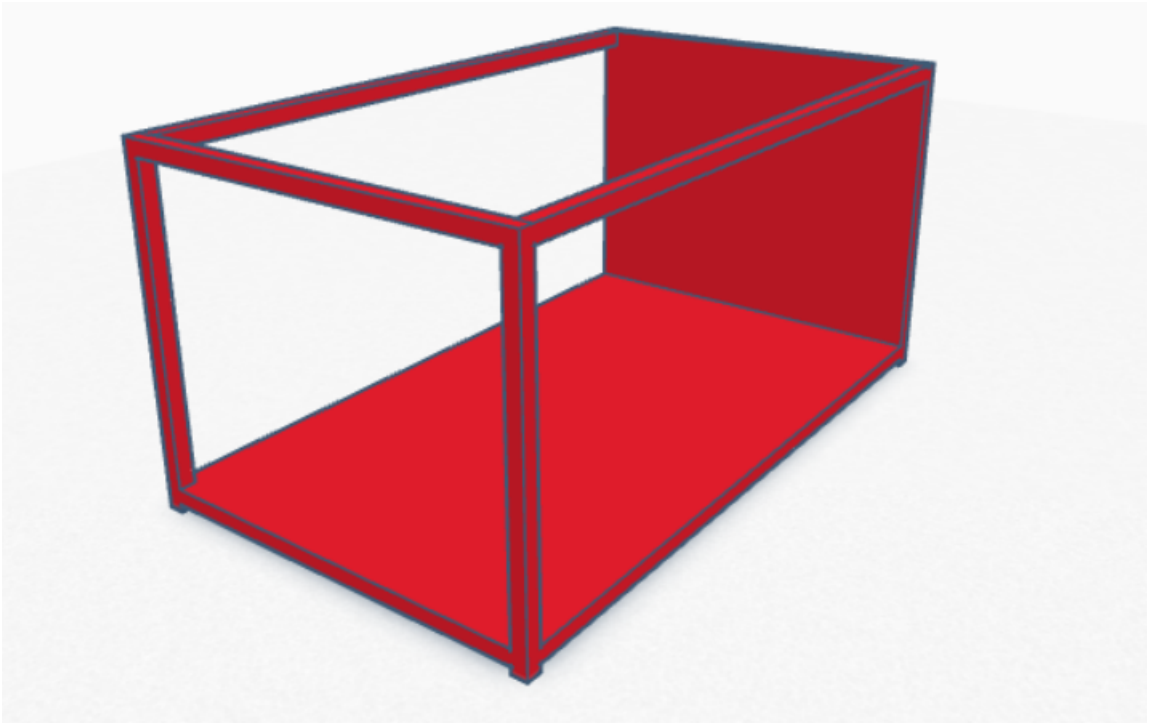

Figure 1: Initial Design for wooden box

### 4.1.2 Wooden Box

We added a wooden box with a 10cm*25cm*25cm dimensions, so we can user it to cover up as much wiring as possible. And also it was used as a support for the ventilation mechanism we used along with the stepper motor.

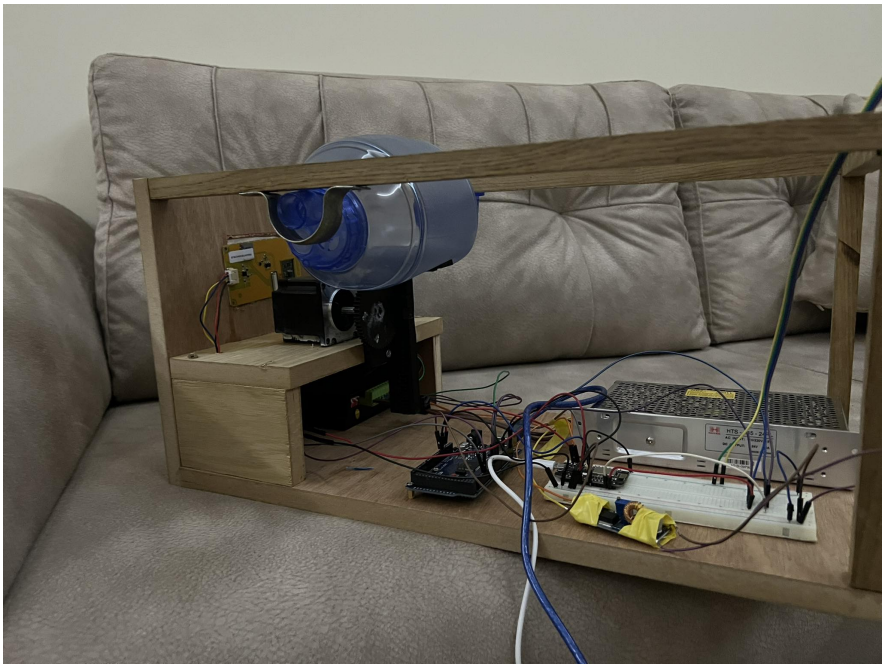

Figure 2: Final Result for the Wooden body & box

### 4.1.3 Ventilation Mechanism

The ventilation mechanism is a linear actuatur, which is 3D printed using carbon fiber, and the design was pulled out from ThingiVerse[6], which has 4 main components: Static part, Linear Actuator, Gear and flat part.

**Static part of the ventilation mechanism**

We used this part to contain the linear actuator, providing a track for the linear actuator, and this part was supported by the wooden box with screws.

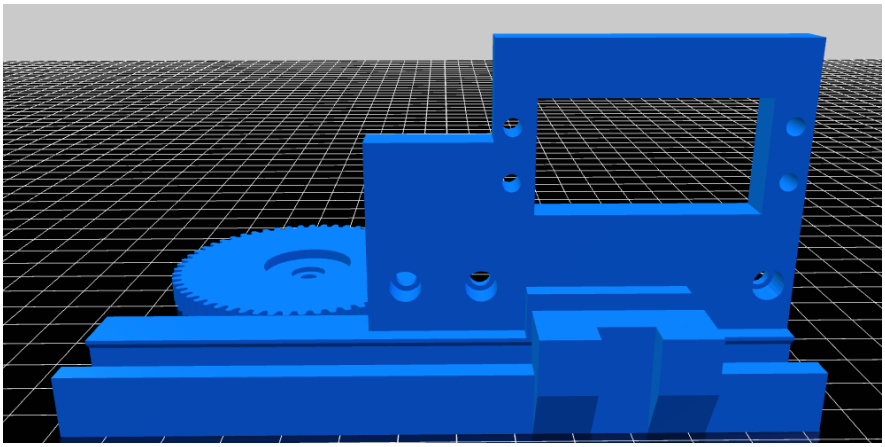

Figure 3: Static part of the mechanism

**Linear Actuator**

We used the linear actuator to move the pressing part on the ambu bag up and down.

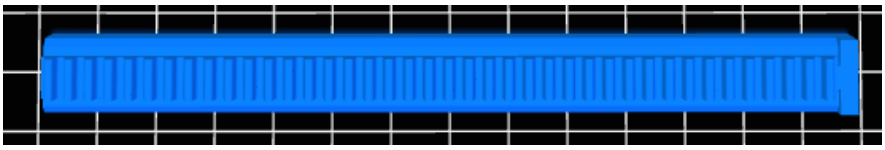

Figure 4: Linear Actuator in Ventilation Mechanism

**Gear**
We used the gear to interface between the stepper motor and the linear actuator, so it can move.

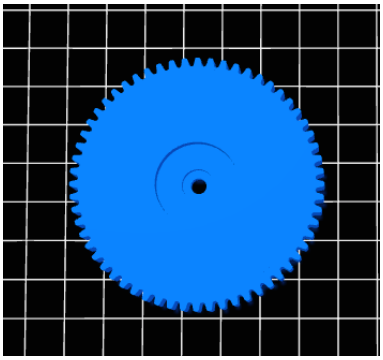

Figure 5: Gear of the mechanism

**Pulley**
We used this pulley to interface between the stepper motor shaft and the gear.

Figure 6: Pulley

**Flat Part**
This was set on the on top of the linear actuator, to make a positive pressure from the ambu bag.

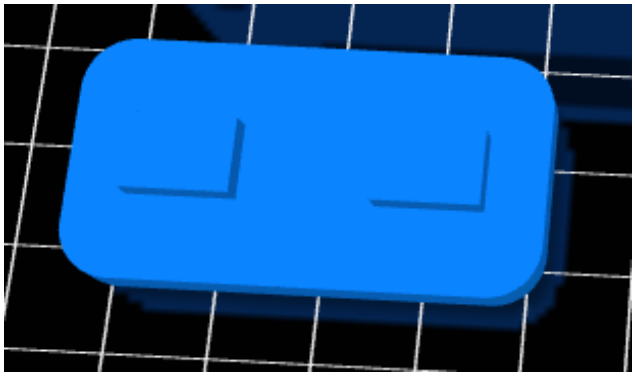

Figure 7: Flat part of the mechanism

**Final Result**

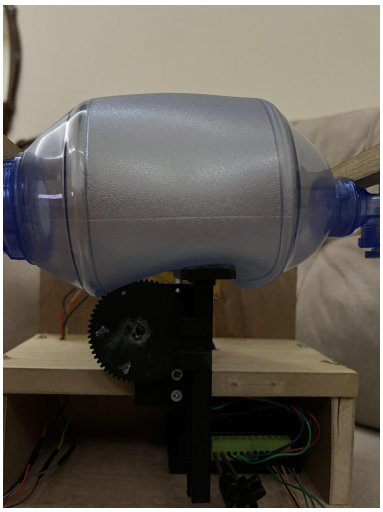

Figure 8: Final result ventilation mechanism

## 4.2 Processing Units and Used Devices

### 4.2.1 Arduino MEGA

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560[1]. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller.
We utilized one Arduino MEGA which was responsible for driving the Nextion Screen, AD8232 ECG Sensor, Stepper motor, and was driven by the ESP8266.

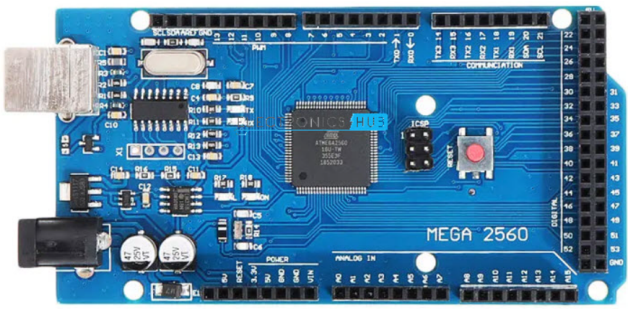


Figure 9: Arduino MEGA

### 4.2.2 ESP8266 NodeMCU

The NodeMCU (Node MicroController Unit) is an open-source software[3] and hardware development environment built around an inexpensive System-on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (WiFi), and even a modern operating system and SDK. That makes it an excellent choice for Internet of Things (IoT) projects of all kinds.
We utilized ESP8266 as the main chip to commuincate with the internet, and also to control the pulse oximetry sensor along with controlling the Arduino MEGA.

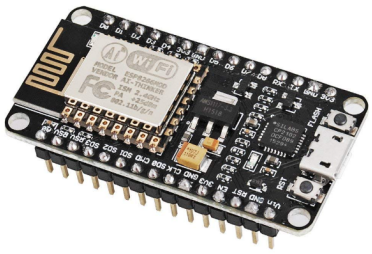

Figure 10: ESP8266 NodeMCU

### 4.2.3 Power Supply

A 24V 6A power supply was utilized in this project in order to provide high current to the stepper motor which we will explain later.

Figure 11: Power Supply

### 4.2.4 Buck Converter

Buck converter is a 4-38V to 1.25-36V 5A stepdown converter we utilized in this project to transform our voltage from the power supply to 5V in order to provide power to the project.


Figure 12: Buck Converter

### 4.2.5 NEMA 17 Stepper Motor

The NEMA 17 stepper motor is widely acclaimed for its compact dimensions and impressive torque output, rendering it a ubiquitous choice in various applications. Operating with a precise step angle of 1.8 degrees per step, it necessitates 200 steps to complete a full revolution. With two windings, it can handle currents up to 3.5 A, while accommodating voltage inputs ranging from 3 to 12 volts.

Within our system, the stepper motor assumes a pivotal role, facilitating the operation of pressing the bump to administer oxygen to patients. By actuating the aforementioned mechanical component, it successfully compresses the bump to deliver air. To achieve the desired inhalation/exhalation ratio, we employ half-revolution rotations in both clockwise and counterclockwise directions. Additionally, manipulating the rotation degree and the inter-step delay allows us to modulate air pressure and determine breaths per minute (BPMS).

In summary, the Nema17 stepper motor's compact design and high torque capabilities make it an invaluable component in our system, enabling precise control over the crucial task of administering oxygen to patients.

Figure 13: NEMA17 Stepper Motor

### 4.2.6 HY-DIV268N-5A Stepper motor driver

We integrated this driver to our system as an additional component layer between the Arduino Microcontroller and the Motor to provide many features: Current feed and regulation, protection of overheating, smaller steps to ensure more precision and the necessary torque. We set the driver to 3.3 A current and with 1/8 step. The motor is connected and controlled through 3 pins: Enable, direction and step pin (PUL). The driver is supplied with voltage from our power supply.

Figure 14: HY-DIV268N-5A Stepper motor driver

### 4.2.7 Nextion Resistive Touch Screen

The Nextion Touch Screen is positioned at the front of our system, and it is utilized in our project to present crucial information regarding the system's functionality as well as the patient's vitals. This display plays a significant role in the manual operation of monitoring the oxygen ventilator by the doctor. It provides real-time data such as BPM (heart rate), air pressure level, Spo2 (blood oxygen saturation) level, and other pertinent features. The touchscreen interface allows for easy adjustment and interaction.

This type of screen, the Nextion resistive touch screen, facilitates touch interaction by applying pressure to multiple layers. It supports input from fingers, stylus, or any object capable of applying pressure.

Nextion offers a comprehensive Integrated Development Environment (IDE) known as Nextion Editor. This IDE simplifies the process of designing graphical user interfaces (GUIs) by providing drag-and-drop functionality for elements such as sliders, labels, buttons, and text. Additionally, the Nextion board integrates a microcontroller. By utilizing a straightforward SD Card interface, we can execute our GUI design and program it using Arduino. This is achieved through serial communication between the Nextion device and our microcontroller, allowing for event registration and handling.

Figure 15: HY-DIV268N-5A Stepper motor driver

### 4.2.8 MAX30100 Pulse Oximeter

This module serves as the core of the Automatic Mode integrated into our system. The sensor plays a vital role in providing us with the Spo2 percentage, which determines the Oxygen Saturation Level in the patient's blood. This information is utilized to establish the required BPM (beats per minute) for the patient, ensuring their stability in terms of oxygen levels without the need for direct doctor intervention. The sensor operates by extracting the Spo2 level from the fingerprint using the principle of photoplethysmography (PPG). It achieves this by emitting and capturing infrared and red light from an LED.

The module establishes a connection with the system through I2C communication. The read values are then utilized in Arduino to implement the logic of our design, allowing us to control and stimulate the ambu bag remotely. It is important to note that the Spo2 level is displayed not only on the system's touchscreen but also on the accompanying Mobile App. This ensures that the oxygen saturation level is conveniently monitored and accessible from multiple interfaces.


Figure 16: MAX30100 Pulse Oximeter

### 4.2.9 Ambu Bag

This is a manual resucitator that is made with a flexible material such as plastic or rubber. Is it used to provide possitive oxygen to the patient, without any external source. It only need a trained assistant tat knows how to use id and press it the correct way and with the corresponding Timing. Our system replaces this function by making this procces automated by the stepper motor and the mechanisim connected to it. The breaths per minute are provided in a correct and uniform waywithout the need of any human effort.

Figure 17: Ambu bag ventilator

## 4.3 How the system works?

### 4.3.1 Boot up process

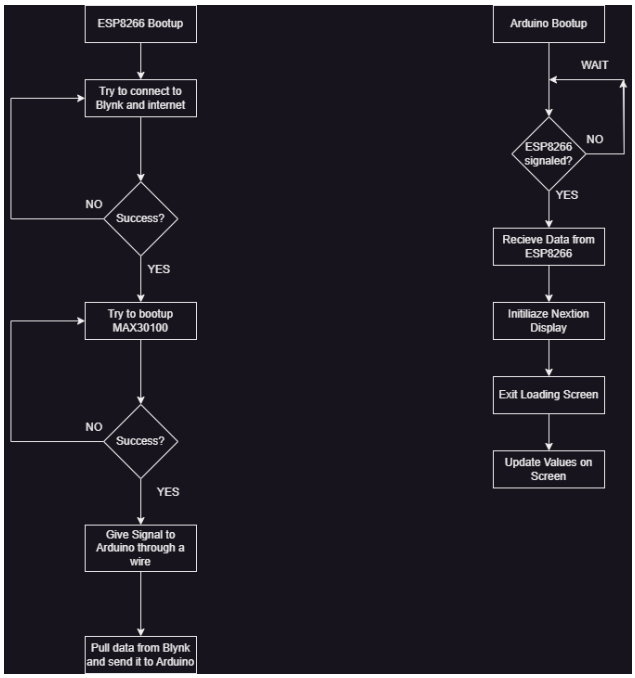


Figure 18: Bootup flowchart

When the Arduino MEGA and the ESP8266 NodeMCU is turned on, the ESP8266 tries to connect to the internet and connect to Blynk servers. At the Arduino side, the Arduino will be waiting for the ESP8266 to connect to Blynk. A loading screen will be displayed on the Nextion.

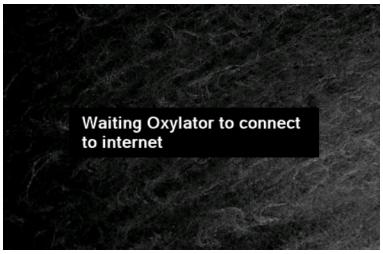


Figure 19: Loading Screen

After ESP8266 connects to the internet, the ESP8266 sends data of BPM (Breaths

Per minute), Air Volume, Manual/Auto, Emergency Switch state to Arduino MEGA through I2C, configured ESP8266 as master and Arduino MEGA as slave, and then update the data displayed on the second page of the screen which will be like Figure 20

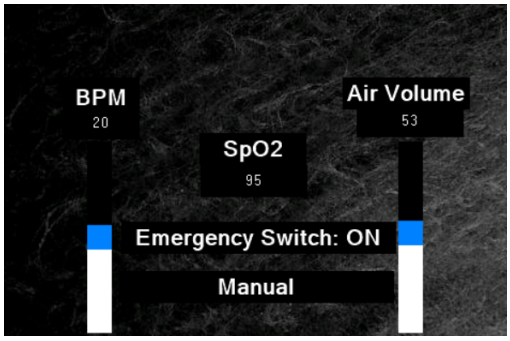


Figure 20: Main Screen

### 4.3.2 Operation Process

**Arduino & Stepper Motor**

The core of our ventilator is the stepper motor, which we will use to make a positive pressure using the ambu bag.

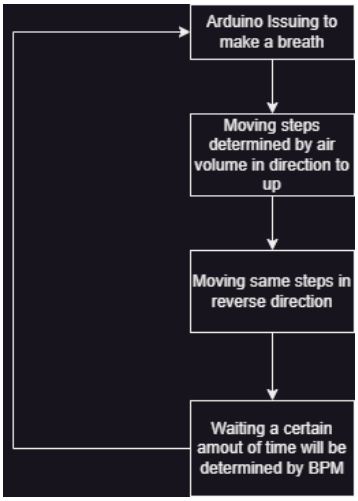


Figure 21: Stepper control flowchart

The wait time in each breath will be determined by this equation:

$$\text{Wait} = \frac{60000}{\text{BPM}} - \frac{\text{Steps} \times 2 \times \text{Speed Delay} \times 2}{1000}$$

Which speed delay is a constant for how fast the motor moves, steps will be determined by air volume, in which we determined the maximum pressure on ambubag will be 600 steps, and minimum is 300 steps. The equation for the steps according to air volume:

$$\text{Steps} = 300 + (600 - 300) \times \frac{\text{Air Volume}}{100}$$

**Arduino & Auto/Manual Mode**

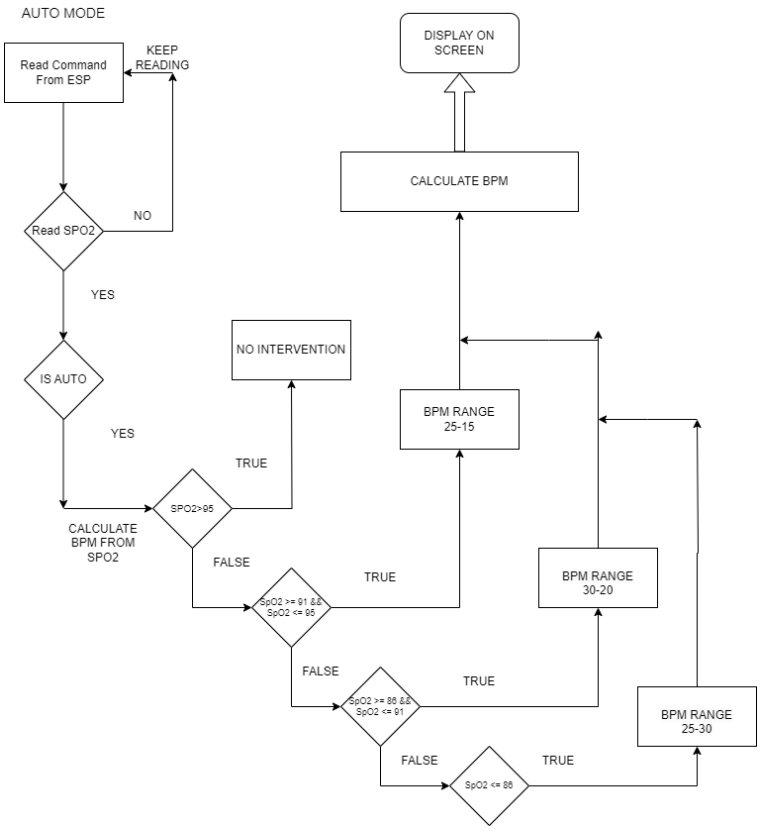


Figure 22: Auto mode control flowchart

In this automode, we read the value of the SPo2 level from the ESP and calculate the

BPM according to the range of BPM[5] needed depending on the range of the SPO2 level as showed on the figure above using the following equation:

$$\text{BPM} = (\text{High BPM} - \text{Low BPM}) \times \frac{\text{High SpO2} - SpO2}{\text{High SpO2} - \text{Low SpO2}} + \text{Low BPM}$$

**Arduino & Screen controlling**
As showed in Figure 20, the main screen can be used to control BPM and Air volume manually, and display state of the emergency switch, and the Auto/Manual Mode status. If the values is updated in the screen, it will also apply to the Arduino and the parameters will keep going.
**Emergency Switch**

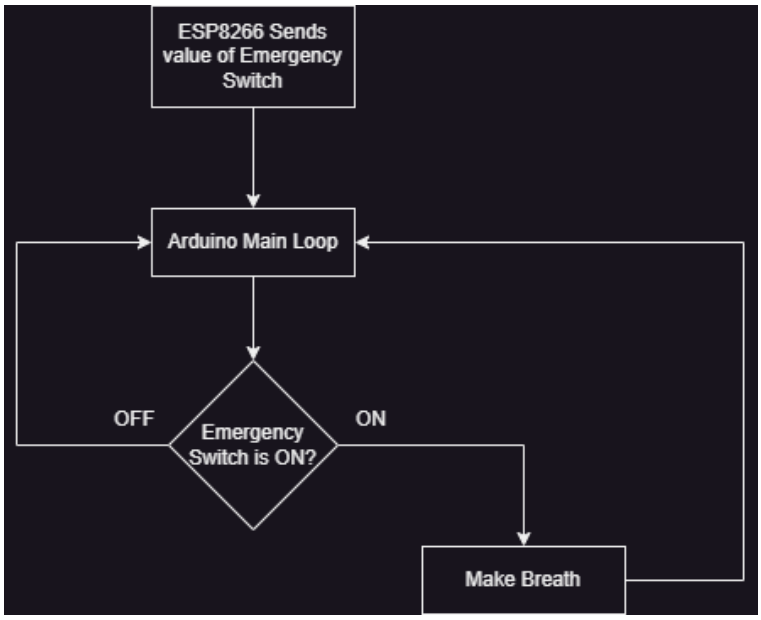


Figure 23: Emergency control flowchart

**ESP8266 & Blynk**

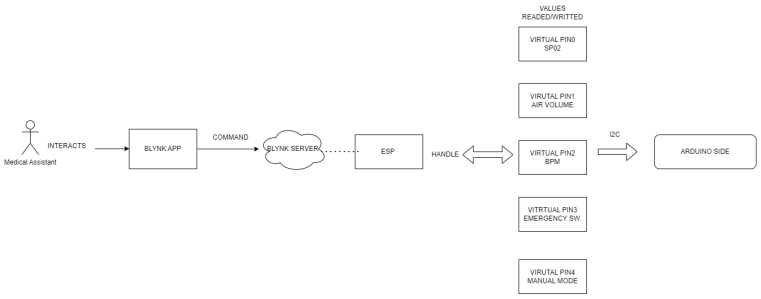


Figure 24: Blynk usecase

And the Blynk application will have this interface:

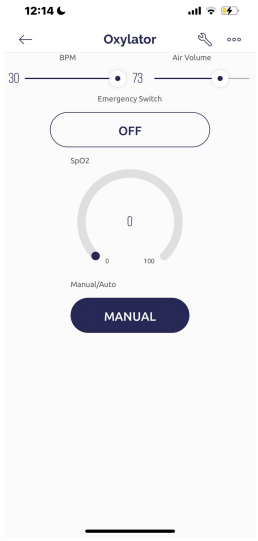


Figure 25: Blynk Interface

**ESP8266 & MAX30100 Pulse Oximeter**

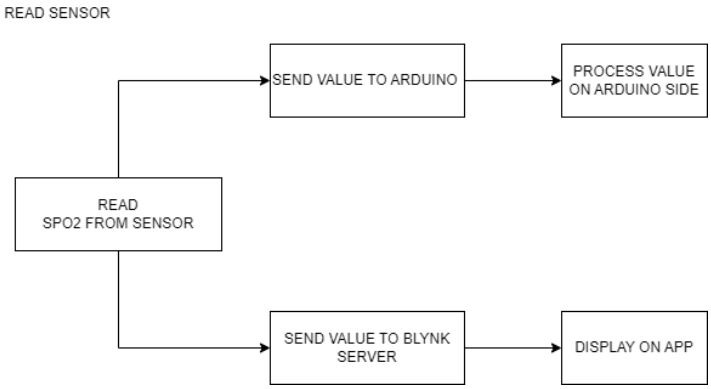


Figure 26: Pulse Oximeter control flowchart

**Arduino & ESP8266 Communication**

The communication between the Arduino and the ESP8266 is done by I2C, where ESP8266 is the master, and the Arduino is the slave, and to send data from the ESP8266, where it can be a BPM, Air volume, Emergency Switch change or either Auto/Manual mode change, we send a character indicating what we are sending to the Arduino before we send the actual data, like 'S' for SpO2, and etc..

# 5 Results & Discussion

## 5.1 Results

The system provides an efficient and automatic Oxygen Ventilator that can be manually adjusted as well. Additionally, the system is equipped with a Mobile App, which serves as a remote control panel for doctors to handle any events or gather real-time information about the patient within the healthcare facility.

## 5.2 Discussion

The system introduces new and innovative features compared to traditional ideas and methods. While it is still in the deployment phase, it offers a solution that has the potential to save millions of lives in the future.

# 6 Conclusion & Future Work

## 6.1 Conclusion

In conclusion, we believe that this idea has the potential to have a significant impact on the healthcare system and could become a highly sought-after product in the future. Its simplicity and user-friendly nature will make it accessible to a wide range of individuals. This has been the primary goal and motivation behind our project from the outset.

## 6.2 Future Work

- Air filter system
- More sophisticated design
- Adding more medical features depending on doctors and specialists feedback
- Develop the app page and make it more detailed
- Add a new type of technologies like ML and analytics
- Integrate the project with newer healthcare systems
- Use special materials instead of wood to make the system more robust and portable

# A Main Code for Arduino Mega

```
#include <Wire.h>

const int dirPin  = 3;
const int stepPin = 4;
const int enPin   = 5;
const int esp32Ready = 10;

int stepsNum = 600;
int speedDelay = 600;
int holdTime = 100;
int bpm = 10;
int air = 100;
int SpO2 = 50;
int wait;
int emergencySwitch = 1;
int manual = 1;

bool mChange = true;
bool bpmChange = true;
bool airChange = true;
bool esChange = true;
bool spChange = true;

#include "Nextion.h"

NexSlider bpmslider = NexSlider(1, 8, "BPMSlider");
NexText bpmValue = NexText(1, 3, "BPMValue");
NexSlider airslider = NexSlider(1,9,"AIRSlider");
NexText airValue = NexText(1,7,"AIRValue");
NexText spValue = NexText(1,5,"SPOValue");
NexText emergencyState = NexText(1, 10, "t0");
NexText manualState = NexText(1, 11, "t1");

NexTouch *nex_listen_list[] = {
  &bpmslider,
  &airslider,
  NULL
};


void exitLoadingScreen(){
  nexSerial.write("page 1");
```

```
  nexSerial.write(0xFF);
  nexSerial.write(0xFF);
  nexSerial.write(0xFF);
}

void bpmsliderPopCallback(void *ptr) {
  uint32_t number = 0;
  char temp[10] = {0};
  // change text with the current slider value
  bpmslider.getValue(&number);
  Serial.println(String(number));
  utoa(number, temp, 10);
  bpm = number;
  bpmValue.setText(temp);
}

void airsliderPopCallback(void *ptr) {
  uint32_t number = 0;
  char temp[10] = {0};
  // change text with the current slider value
  airslider.getValue(&number);
  Serial.println(String(number));
  utoa(number, temp, 10);
  air = number;
  airValue.setText(temp);
}



void moveSteps(int steps, int dir){
  digitalWrite(dirPin,dir);
  wait = 60000/bpm - (steps*2*speedDelay*2)/1000 - 300;
  for(int x = 0; x < steps; x++) {
      digitalWrite(stepPin,HIGH);
      delayMicroseconds(speedDelay);
      digitalWrite(stepPin,LOW);
      delayMicroseconds(speedDelay);
  }
}


void stepperMakeBreath(){
  int steps = 300 + (600-300)*air/100;
  moveSteps(steps,0);
  //delay(holdTime); // One second delay
  // 600 max pump
```

```
  // 300 min
  moveSteps(steps,1);

  delay(wait);
}

int calculateAutoBPM(int low, int high, int lowerR, int higherR, int R){
  return (int)(high-low)*(higherR - R)/(higherR-lowerR)+low;
}

void setup() {
  Wire.begin(20);                /* join i2c bus with address 8 */
  Wire.onReceive(receiveEvent); /* register receive event */
  Wire.onRequest(requestEvent); /* register request event */
  Serial.begin(9600);
  pinMode(stepPin,OUTPUT);
  pinMode(dirPin,OUTPUT);
  pinMode(enPin,OUTPUT);
  pinMode(esp32Ready, INPUT_PULLUP);
  digitalWrite(enPin,LOW);
  int value = digitalRead(esp32Ready);
  while(digitalRead(esp32Ready) == 1);
  nexSerial.begin(9600);
  nexInit();
  exitLoadingScreen();
  bpmslider.attachPop(bpmsliderPopCallback);
  airslider.attachPop(airsliderPopCallback);
  // on initiate request to master to get values of bpm and air volume from blynk
}

void changeNexText(int num,NexText label){
  char temp[10] = {0};
  utoa(num, temp, 10);
  label.setText(temp);
}

void changeStateES(){
  if(emergencySwitch == 1){
    emergencyState.setText("Emergency Switch: ON");
  }else emergencyState.setText("Emergency Switch: OFF");
}

void changeStateManual(){
    if(manual == 1){
    manualState.setText("Manual");
  }else manualState.setText("Auto");
```

```
}
// }
// B 30
void loop() {
  if(bpmChange){
    bpmslider.setValue(bpm);
    changeNexText(bpm,bpmValue);
    bpmChange = false;
  }
  if(airChange){
    airslider.setValue(air);
    changeNexText(air,airValue);
    airChange = false;
  }
  if(esChange){
    changeStateES();
    esChange = false;
  }
  if(spChange){
    changeNexText(SpO2,spValue);
    spChange = false;
  }
  if(mChange){
    changeStateManual();
    mChange = false;
  }

  nexLoop(nex_listen_list);
  if(bpm != 0 && emergencySwitch)
    stepperMakeBreath();
  // Send data from MAX30100
  }

// function that executes whenever data is received from master
void receiveEvent(int howMany) {
 if(Wire.available() > 0){
    char c = Wire.read();
    Serial.println(c);
    if(Wire.available()){

      int value = Wire.read();
      Serial.println(String(value));   /* receive byte as a character */
      if(c == 'B'){
        //Bpm switch
        if(manual){
          bpm = value;
```

```
          bpmChange = true;
        }
      }else if(c == 'V'){
        // Air volume switch ( we edit number of steps)
        if(manual){
          air = value;
          airChange = true;
        }
      }else if(c == 'E'){
        // Emergency Switch
        emergencySwitch = value;
        esChange = true;
      }else if(c == 'S'){
        SpO2 = value;
        spChange = true;
        if(manual == 0 && SpO2 != 0){
          // calculate bpm and air volume from spo2
          if(SpO2 >= 95){
            // No intervention needed, stay at the previous state
          }else if(SpO2 >= 91 && SpO2 <= 95){
            bpm = calculateAutoBPM(15,25,91,95,SpO2);
            bpmChange = true;
          }else if(SpO2 >= 86 && SpO2 <= 91){
            bpm = calculateAutoBPM(20,30,86,91,SpO2);
            bpmChange = true;
          }else if(SpO2 <= 86){
            bpm = calculateAutoBPM(25,30,80,86,SpO2);
            bpmChange = true;
          }
        }
      }
      else if(c == 'M'){
        manual = value;
        mChange = true;
      }
    }           /* print the character */
  }            /* to newline */
}

// function that executes whenever data is requested from master
void requestEvent() {
 Wire.write("Hello NodeMCU");  /*send string on request */
}
```

# B Main code for ESP8266 NodeMCU

```
#define BLYNK_TEMPLATE_ID "TMPL6AyNbFNWa"
#define BLYNK_TEMPLATE_NAME "Oxylator"
#define BLYNK_AUTH_TOKEN "NeGGidT2OMcUEl3EHg1bOoKDFKBxzwG6"
#define BLYNK_PRINT Serial



char ssid[] = "Yousef";
char pass[] = "12345678";

const int beatsToUpdate = 70;

const int pinReady = D7;

#include <Wire.h>
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <SoftwareSerial.h>
#include "MAX30100_PulseOximeter.h"
#define REPORTING_PERIOD_MS     1000

PulseOximeter pox;

BLYNK_WRITE(V0)
{
  // SPO2
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to V0");

}

BLYNK_WRITE(V1)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to BPM");

int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('B');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
```

```
}


BLYNK_WRITE(V2)
{
  // Air Volume

    Serial.println("Blynk.Cloud is writing something to Air Volume");

  int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('V');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
}

BLYNK_WRITE(V3)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to Emergency switch");

  int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('E');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
  //Serial1.println(value);
}


BLYNK_WRITE(V4)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to Manual");

  int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('M');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
  //Serial1.println(value);
}
```

```
int currentBeats = 0;

void onBeatDetected()
{
    currentBeats++;
}

void setup() {
 Serial.begin(9600); /* begin serial for debug */
 //Wire.begin(); /* join i2c bus with SDA=D1 and SCL=D2 of NodeMCU */
 pinMode(pinReady, OUTPUT);
 digitalWrite(pinReady,HIGH);
 Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
 while(!Blynk.connected());
 digitalWrite(pinReady, LOW);
 Blynk.syncVirtual(V1);
 Blynk.syncVirtual(V2);
 Blynk.syncVirtual(V3);
 Blynk.syncVirtual(V4);
    pox.setOnBeatDetectedCallback(onBeatDetected);
 if (!pox.begin()) {
    Serial.println("FAILED");
     for(;;);
  } else {
      Serial.println("SUCCESS");
  }         /* start serial for debug */
}



void loop() {    /* stop transmitting */
 Blynk.run();
 pox.update();
 int SpO2 = pox.getSpO2();
 Blynk.virtualWrite(V0, SpO2);
 Wire.beginTransmission(20);
 Wire.write('S');
 Wire.write(SpO2);
 Wire.endTransmission();
}
```

# C I2C code from slave(Arduino MEGA)

```
void receiveEvent(int howMany) {
 if(Wire.available() > 0){
    char c = Wire.read();
    Serial.println(c);
    if(Wire.available()){

      int value = Wire.read();
      Serial.println(String(value));   /* receive byte as a character */
      if(c == 'B'){
        //Bpm switch
        if(manual){
          bpm = value;
          bpmChange = true;
        }
      }else if(c == 'V'){
        // Air volume switch ( we edit number of steps)
        if(manual){
          air = value;
          airChange = true;
        }
      }else if(c == 'E'){
        // Emergency Switch
        emergencySwitch = value;
        esChange = true;
      }else if(c == 'S'){
        SpO2 = value;
        spChange = true;
        if(manual == 0 && SpO2 != 0){
          // calculate bpm and air volume from spo2
          if(SpO2 >= 95){
            // No intervention needed, stay at the previous state
          }else if(SpO2 >= 91 && SpO2 <= 95){
            bpm = calculateAutoBPM(15,25,91,95,SpO2);
            bpmChange = true;
          }else if(SpO2 >= 86 && SpO2 <= 91){
            bpm = calculateAutoBPM(20,30,86,91,SpO2);
            bpmChange = true;
          }else if(SpO2 <= 86){
            bpm = calculateAutoBPM(25,30,80,86,SpO2);
            bpmChange = true;
          }
        }
```

```
      }
      else if(c == 'M'){
        manual = value;
        mChange = true;
      }
    }           /* print the character */
  }            /* to newline */
}
```

# D I2C Code from master and Blynk(ESP8266)

```
BLYNK_WRITE(V0)
{
  // SPO2
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to V0");

}

BLYNK_WRITE(V1)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to BPM");

int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('B');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();

}


BLYNK_WRITE(V2)
{
  // Air Volume

    Serial.println("Blynk.Cloud is writing something to Air Volume");

  int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('V');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
}

BLYNK_WRITE(V3)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to Emergency switch");
```

```
  int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('E');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
  //Serial1.println(value);
}


BLYNK_WRITE(V4)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.println("Blynk.Cloud is writing something to Manual");

  int value = param.asInt();

  Wire.beginTransmission(20);/* begin with device address 8 */
  Wire.write('M');
  Wire.write(value);  /* sends hello string */
  Wire.endTransmission();
  //Serial1.println(value);
}
```

# E Screen's Code

```
#include "Nextion.h"

NexSlider bpmslider = NexSlider(1, 8, "BPMSlider");
NexText bpmValue = NexText(1, 3, "BPMValue");
NexSlider airslider = NexSlider(1,9,"AIRSlider");
NexText airValue = NexText(1,7,"AIRValue");
NexText spValue = NexText(1,5,"SPOValue");
NexText emergencyState = NexText(1, 10, "t0");
NexText manualState = NexText(1, 11, "t1");

NexTouch *nex_listen_list[] = {
  &bpmslider,
  &airslider,
  NULL
};


void exitLoadingScreen(){
  nexSerial.write("page 1");
  nexSerial.write(0xFF);
  nexSerial.write(0xFF);
  nexSerial.write(0xFF);
}

void bpmsliderPopCallback(void *ptr) {
  uint32_t number = 0;
  char temp[10] = {0};
  // change text with the current slider value
  bpmslider.getValue(&number);
  Serial.println(String(number));
  utoa(number, temp, 10);
  bpm = number;
  bpmValue.setText(temp);
}

void airsliderPopCallback(void *ptr) {
  uint32_t number = 0;
  char temp[10] = {0};
  // change text with the current slider value
  airslider.getValue(&number);
  Serial.println(String(number));
  utoa(number, temp, 10);
```

```
  air = number;
  airValue.setText(temp);
}
```

# F Stepper's Code

```
const int dirPin  = 3;
const int stepPin = 4;
const int enPin   = 5;
const int esp32Ready = 10;

int stepsNum = 600;
int speedDelay = 600;
int holdTime = 100;
int bpm = 10;
int air = 100;
int SpO2 = 50;
int wait;

void moveSteps(int steps, int dir){
  digitalWrite(dirPin,dir);
  wait = 60000/bpm - (steps*2*speedDelay*2)/1000 - 300;
  for(int x = 0; x < steps; x++) {
      digitalWrite(stepPin,HIGH);
      delayMicroseconds(speedDelay);
      digitalWrite(stepPin,LOW);
      delayMicroseconds(speedDelay);
  }
}

void stepperMakeBreath(){
  int steps = 300 + (600-300)*air/100;
  moveSteps(steps,0);
  moveSteps(steps,1);
  delay(wait);
}
```

# G Pulse Oximeter's code

```
#define REPORTING_PERIOD_MS     1000
#define REPORTING_PERIOD_MS     1000
#include "MAX30100_PulseOximeter.h"

PulseOximeter pox;

void main(){
 pox.update();
 int SpO2 = pox.getSpO2();
 Blynk.virtualWrite(V0, SpO2);
 Wire.beginTransmission(20);
 Wire.write('S');
 Wire.write(SpO2);
 Wire.endTransmission(); pox.update();
 int SpO2 = pox.getSpO2();
 Blynk.virtualWrite(V0, SpO2);
 Wire.beginTransmission(20);
 Wire.write('S');
 Wire.write(SpO2);
 Wire.endTransmission();
}
```

# References


[1] Arduino. *Arduino*. Accessed 2023. URL: https://www.arduino.cc/.

[2] Kelly Pneumatics. *A Brief History of Mechanical Ventilation Systems*. Accessed 2023. URL: https://kellypneumatics.com/a-brief-history-of-mechanical-ventilation-systems/.

[3] Make It. *NodeMCU Details & Specifications*. Accessed 2023. URL: https://www.make-it.ca/nodemcu-details-specifications/.

[4] Palestinian Central Bureau of Statistics. *Palestinian Central Bureau of Statistics (PCBS) Announces Results of Impact of COVID-19 Pandemic (Coronavirus) on the Socio-economic Conditions of Palestinian Households Survey (March-May), 2020*. Accessed 2023. URL: https://www.pcbs.gov.ps/postar.aspx?lang=ar&ItemID=3824.

[5] Physiopedia. *Pulse Oximeter*. Accessed 2023. URL: https://www.physiopedia.com/Pulse_Oximeter.

[6] Thingiverse. *3D Printed Ventilator Valve*. Accessed 2023. URL: https://www.thingiverse.com/thing:4292941?fbclid=IwAR04X0O8ZfdktAqv3YJXW9A0p4kVsjCIsC