Junior Year Championship

Variable Frequency Drive (VFD) Control through Facial Identification

Shazil Ejaz
2020456
FEE(E)
GIKI, Topi, Swabi, Pakistan
u2020456@giki.edu.pk

Syed Usman Shah
2020486
FEE(E)
GIKI, Topi, Swabi, Pakistan
u2020486@giki.edu.pk

Ahsan Idrees
2020056
FEE(E)
GIKI, Topi, Swabi, Pakistan
u2020056@giki.edu.pk

Abstract— This report elaborates about a Variable Frequency Drive (VFD) which is an advanced electronic control device that enables precise control over the speed and torque of an electric motor. By varying the frequency and voltage supplied to the motor, VFDs can adjust the motor's speed to match the specific needs of a given application. This makes them highly versatile and suitable for use in a wide range of industrial applications, such as HVAC systems and more. In addition to their precision control capabilities, VFDs also offer significant energy savings and improved process control, making them a valuable investment for businesses looking to optimize their operations. With ongoing advancements in VFD technology, these devices continue to provide ever-improving levels of efficiency and performance for a variety of applications.

Keywords— VFD, Motor control, Arduino, Face recognition, Machine Learning, Python, Serial Communication, USB protocol, Image processing, Deep Learning

I. INTRODUCTION

The Variable Frequency Drive (VFD) control system, combined with object identification using machine learning (ML) algorithms, offers immense potential for precise motor control and automation in various industries [2]. In this project, we aim to design and implement a VFD control system that integrates object identification capabilities, utilizing an Arduino for PWM generation, MOSFETs (1RF840) for power switching, a diode (10A10) for rectification, and a capacitor (470uF) for smoothing. Additionally, a 20W motor will is for testing and demonstration purposes.

The project builds upon the concept of face identification through ML algorithms, where three faces have been selected as symbols for identification. A dataset of images representing these objects has been generated and labeled, enabling the training of an ML model using Python and OpenCV package. The trained ML model will enable real-time identification of the objects based on input images.

To control the VFD, the facials are recognized through Laptop camera and serially communicated with the Arduino that generates pulse width modulation (PWM) signals. These signals will drive the MOSFETs, facilitating the conversion of DC power to AC power through the use of an IGBT network. The diode (10A10) will ensure proper rectification of the AC voltage, and the capacitor (470uF) will smoothen the output voltage, reducing ripples.

Simulink simulation has been performed to verify the effectiveness and feasibility of the proposed VFD control system. The simulation provides valuable insights into the expected performance and behavior of the system, aiding in the optimization of control parameters and ensuring efficient motor operation.

By utilizing Arduino for PWM generation, MOSFETs for power switching, and incorporating object identification capabilities through ML algorithms, this project aims to showcase the synergy between advanced control techniques and intelligent motor control. The integration of these components provides an opportunity to enhance motor control accuracy, adaptability, and automation, ultimately leading to improved efficiency in industrial applications.

Throughout this project, the focus will be on harnessing the power of ML algorithms, leveraging Arduino's capabilities for precise PWM generation, and validating the performance using Simulink simulation. The successful implementation of this integrated VFD control system has the potential to revolutionize motor control processes, leading to increased efficiency and productivity in various industries.

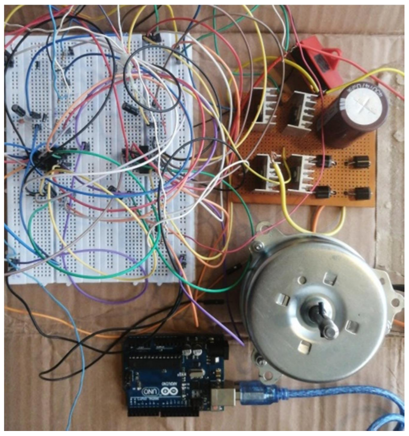
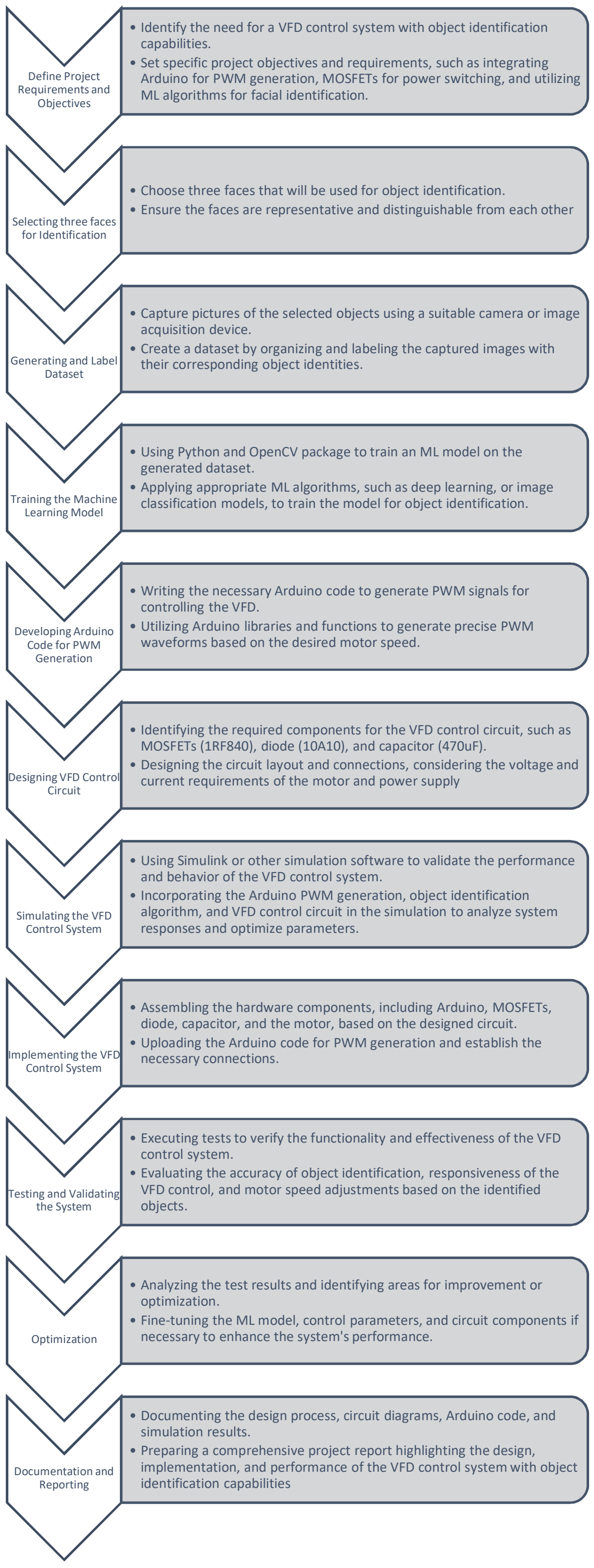


Figure 1 VFD Circuit Implementation

II. DESIGN PROCESS FLOW



Define Project Requirements and Objectives
- Identify the need for a VFD control system with object identification capabilities.
- Set specific project objectives and requirements, such as integrating Arduino for PWM generation, MOSFETs for power switching, and utilizing ML algorithms for facial identification.

Selecting three faces for Identification
- Choose three faces that will be used for object identification.
- Ensure the faces are representative and distinguishable from each other

Generating and Label Dataset
- Capture pictures of the selected objects using a suitable camera or image acquisition device.
- Create a dataset by organizing and labeling the captured images with their corresponding object identities.

Training the Machine Learning Model
- Using Python and OpenCV package to train an ML model on the generated dataset.
- Applying appropriate ML algorithms, such as deep learning, or image classification models, to train the model for object identification.

Developing Arduino Code for PWM Generation
- Writing the necessary Arduino code to generate PWM signals for controlling the VFD.
- Utilizing Arduino libraries and functions to generate precise PWM waveforms based on the desired motor speed.

Designing VFD Control Circuit
- Identifying the required components for the VFD control circuit, such as MOSFETs (1RF840), diode (10A10), and capacitor (470uF).
- Designing the circuit layout and connections, considering the voltage and current requirements of the motor and power supply

Simulating the VFD Control System
- Using Simulink or other simulation software to validate the performance and behavior of the VFD control system.
- Incorporating the Arduino PWM generation, object identification algorithm, and VFD control circuit in the simulation to analyze system responses and optimize parameters.

Implementing the VFD Control System
- Assembling the hardware components, including Arduino, MOSFETs, diode, capacitor, and the motor, based on the designed circuit.
- Uploading the Arduino code for PWM generation and establish the necessary connections.

Testing and Validating the System
- Executing tests to verify the functionality and effectiveness of the VFD control system.
- Evaluating the accuracy of object identification, responsiveness of the VFD control, and motor speed adjustments based on the identified objects.

Optimization
- Analyzing the test results and identifying areas for improvement or optimization.
- Fine-tuning the ML model, control parameters, and circuit components if necessary to enhance the system's performance.

Documentation and Reporting
- Documenting the design process, circuit diagrams, Arduino code, and simulation results.
- Preparing a comprehensive project report highlighting the design, implementation, and performance of the VFD control system with object identification capabilities

Figure 2 Design Flowchart

III. Machine Learning

A. What is machine learning?

Machine learning is a subfield of artificial intelligence that focuses on the development of algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. It involves training a computer system on a large amount of data, allowing it to recognize patterns, make generalizations, and improve its performance over time. By leveraging statistical techniques and computational power, machine learning algorithms can extract meaningful insights, discover hidden patterns, and automate complex tasks across various domains, contributing to advancements in fields such as image recognition, natural language processing, and predictive analytics. [3]

B. Data Set

In the provided code, a dataset of 200 pictures is used for model training. The dataset serves as a collection of images used to train the face recognition system. The pictures in the dataset represent different individuals or faces that the system will be able to recognize.

Before using the dataset for training, it is important to ensure that the data is in a clean format and suitable for the training process. Therefore, data cleansing is performed as a preprocessing step. The specific details of the data cleansing process likely involve removing any noise or inconsistencies in the dataset that could affect the accuracy of the face recognition system. [4]

Additionally, the dataset is processed to focus on the facial region of interest. This involves utilizing algorithms such as the Haar Cascade Classifier, specifically the pre-trained models haarcascade_frontalface_default.xml and haarcascade_eye.xml, to detect and extract the face area and eyes within each image. This step helps in isolating the relevant facial features for accurate face recognition.

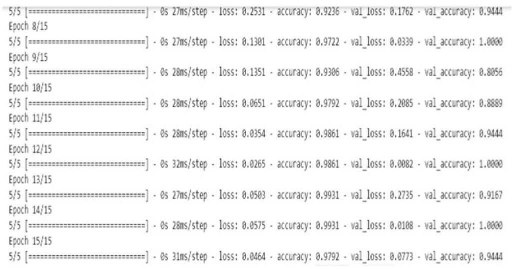
Furthermore, the dataset is divided into three different classes, likely corresponding to the three individuals whose images are provided (Ahsan, Shazil, and Usman). This classification allows for cropping the images to a standardized dimension of 180 by 180 pixels. The cropped images are then used as training data, which contributes to improving the efficiency and accuracy of the face recognition system.

Overall, the dataset plays a crucial role in training the face recognition model, and various preprocessing techniques are applied to ensure its cleanliness and optimize the accuracy of the system.

C. Background

Initially, we collected a dataset of 200 images for training our model. However, we encountered several errors while using Google Colab [6] for training. To address this, we performed data cleansing and preprocessing. The preprocessing step involved utilizing a code to detect the eyes and the surrounding facial area in each image. We utilized a pre-trained model for this purpose.

The initial model resulted in a low accuracy of 34%. To improve the accuracy, we implemented a cascading classifier combined with a Tensorflow deep learning stack. Tensorflow deep learning stack. [14] This approach involved dividing the data into three different classes and cropping the images to a dimension of 180 by 180 pixels. This transformation significantly increased the efficiency of the model, achieving an accuracy of approximately 97% as depicted in Figure 3 and Figure 4.



```
5/5 [==============================] - 0s 27ms/step - loss: 0.2531 - accuracy: 0.9236 - val_loss: 0.1762 - val_accuracy: 0.9444
Epoch 8/15
5/5 [==============================] - 0s 27ms/step - loss: 0.1301 - accuracy: 0.9722 - val_loss: 0.0339 - val_accuracy: 1.0000
Epoch 9/15
5/5 [==============================] - 0s 28ms/step - loss: 0.1351 - accuracy: 0.9306 - val_loss: 0.4558 - val_accuracy: 0.8056
Epoch 10/15
5/5 [==============================] - 0s 28ms/step - loss: 0.0651 - accuracy: 0.9792 - val_loss: 0.2085 - val_accuracy: 0.8889
Epoch 11/15
5/5 [==============================] - 0s 28ms/step - loss: 0.0354 - accuracy: 0.9861 - val_loss: 0.1641 - val_accuracy: 0.9444
Epoch 12/15
5/5 [==============================] - 0s 32ms/step - loss: 0.0265 - accuracy: 0.9861 - val_loss: 0.0082 - val_accuracy: 1.0000
Epoch 13/15
5/5 [==============================] - 0s 27ms/step - loss: 0.0503 - accuracy: 0.9931 - val_loss: 0.2735 - val_accuracy: 0.9167
Epoch 14/15
5/5 [==============================] - 0s 28ms/step - loss: 0.0575 - accuracy: 0.9931 - val_loss: 0.0108 - val_accuracy: 1.0000
Epoch 15/15
5/5 [==============================] - 0s 31ms/step - loss: 0.0464 - accuracy: 0.9792 - val_loss: 0.0773 - val_accuracy: 0.9444
```

Figure 3 Results of accuracy for Cascading Classifier

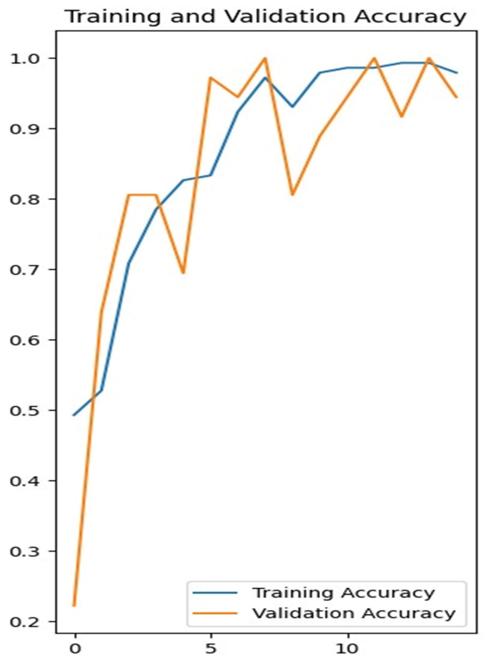


Figure 4 Training and Validation Accuracy for Cascading Classifier

D. Python Script

Appendix A is a Python script that uses various libraries to perform face recognition on a video feed from a webcam. Here's a breakdown of what the code does: It imports the necessary libraries: (OpenCV), dlib, face_recognition [7], and serial (for communication with Arduino).

It initializes a serial connection to an Arduino using the specified port and baud rate. It loads images of three persons and their corresponding face encodings. These images are used as reference for face recognition. It initializes the video capture from the default webcam (index 0). It enters a loop where it continuously captures frames from the video feed. It detects faces in the resized frame using face_recognition.face_locations and computes face encodings using face_recognition.face_encodings. It compares the computed face encodings with the known face encodings to determine if there is a match. If a match is found, it assigns the corresponding name to the face. Otherwise, it assigns the name "Unknown".

It displays the video feed with rectangles around the detected faces and labels showing the names of the recognized persons as shown in figures 5, 6, and 7.

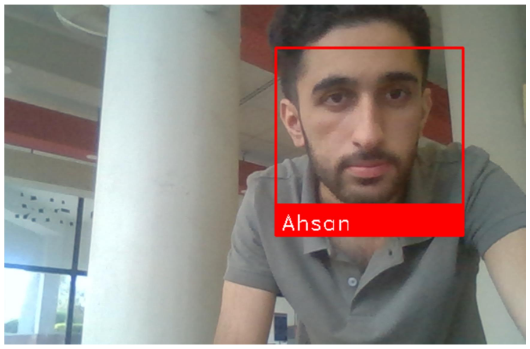


Figure 5 Facial Recognition of "Ahsan"

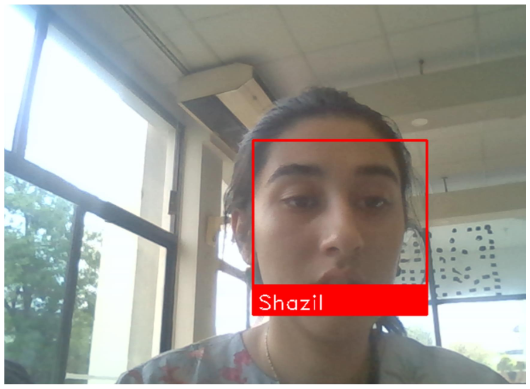


Figure 6 Facial Recognition of "Shazil"

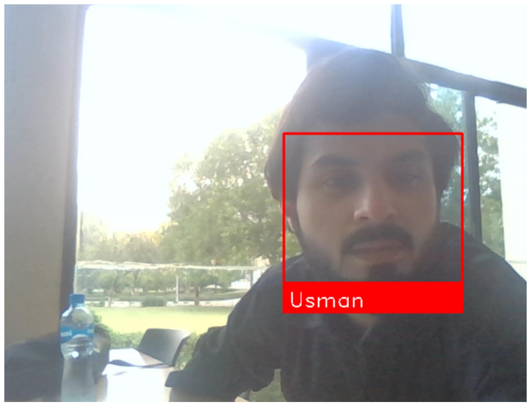


Figure 7 Facial Recognition of "Usman"

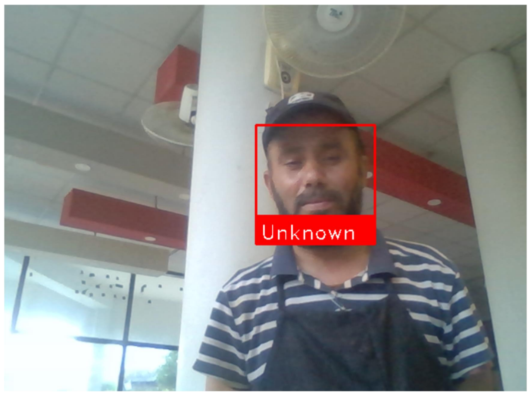


Figure 8 Facial Recognition of "Unknown"

E. Conclusion

We employed a method to address the issues we encountered in the initial approach. In this new method, we utilized a Python script that incorporated various libraries, including OpenCV, Dlib, and Face Recognition. The script involved capturing a video feed from a webcam and performing facial recognition on each frame. Initially, we trained the model using a dataset of 200 images each, but faced challenges in Google Colab. To overcome this, we cleansed and preprocessed the data by implementing facial recognition algorithms and cascade classifiers for eye detection. By dividing the data into three classes and cropping the images to a standardized dimension, we significantly improved the accuracy from 34% to approximately 97%. The script successfully recognized and labeled faces by comparing the computed face encodings with known face encodings. The resulting video feed displayed rectangles around the detected faces and labeled them with the corresponding names. This method proved to be effective in achieving our desired results.

IV. COMMUNICATION

A. Communication Module Used

In the implemented system, a communication module was utilized to establish seamless communication between the Python script and an Arduino board. The "serial" module, a standard library in Python, was employed for this purpose. This module provides comprehensive support for serial communication over different interfaces, including USB (Universal Serial Bus).

To establish the connection, the specific port and baud rate settings were configured in the code. The port is represented by a string, such as '/dev/ttyUSB0' or 'COM5', depending on the operating system and hardware configuration. The baud rate indicates the communication speed in bits per second and must match the corresponding setting on the Arduino board.

Once the connection was successfully established using the serial module, the Python script was able to transmit data to the Arduino. In the provided code snippet in Appendix B, the face names (i.e., the recognized persons' names) were encoded as bytes using the face_names.encode() function. These encoded bytes were then sent through the serial connection using the ser.write() function. This facilitated the transfer of the recognized face names to the Arduino, enabling further processing or display on connected hardware components, such as LED displays or motors.

The integration of the communication module served as a vital component in the system, ensuring seamless data transmission and synchronization between the Python script and the Arduino board. This capability expanded the possibilities for real-time applications and interactions, allowing the face recognition system to interact with and control external hardware components effectively.

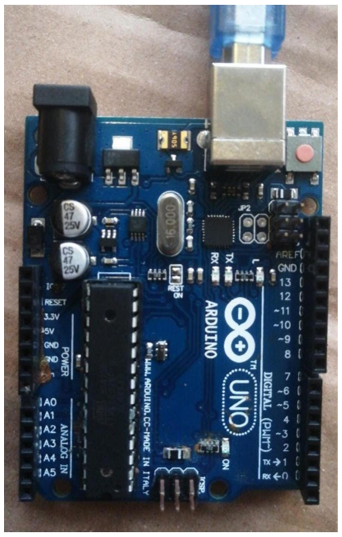


Figure 9 Arduino UNO

B. Protocol being used

The communication module used in the provided code is based on the Universal Serial Bus (USB) protocol. USB is a widely adopted industry-standard protocol that enables reliable and efficient communication between devices and computers. In this code, the serial module in Python is employed to establish communication with an Arduino board via a USB port. The specific USB port is specified in the code, such as '/dev/ttyUSB0' or 'COM5', depending on the operating system. The baud rate, representing the data transfer speed, is also set accordingly. By leveraging the USB protocol, the code facilitates seamless data exchange and control between the Python-based face recognition system and the Arduino board, ensuring smooth and reliable communication between the two components.

Figure 10 USB Protocol for serial communication

V. HARDWARE

A. Invertor

A Variable Frequency Drive (VFD) uses a full wave rectifier bridge to convert AC power to DC power [9]. The rectifier bridge is made up of four 10A10 diodes, arranged in a specific configuration, so that they allow current to flow in only one direction. This converts the AC voltage to a pulsating DC voltage.

The output of the rectifier bridge consists of a series of pulses, which are not suitable for most electronic devices. To smooth out these pulses, a large 470uF 450V electrolytic capacitor was connected across the output of the rectifier bridge. This capacitor charges up during the peaks of the rectified voltage and discharges during the gaps between them, resulting in a smoother DC voltage with less ripple.

Once the rectification and smoothing are complete, the VFD can then use additional circuitry to control the frequency and amplitude of the DC voltage. By varying these parameters, the VFD can produce an AC voltage at the desired frequency and voltage level to drive the motor.

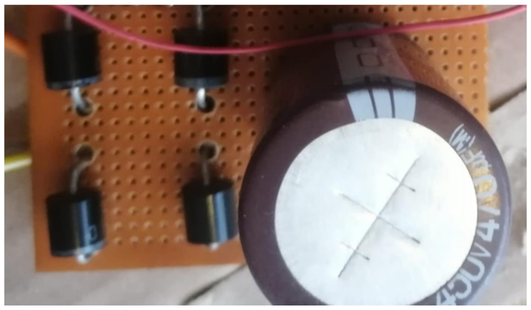


Figure 11 Invertor Circuit Implementation

B. *Motor driver IC*

In a Variable Frequency Drive (VFD), the motor driver IC plays a crucial role in controlling the speed and direction of the motor. The motor driver IC is responsible for receiving signals from the control circuitry and converting them into electrical energy that can be used to power the motor.

The motor driver IC is specifically designed to handle the high voltages and currents required to drive the motor. It acts as an interface between the low-voltage control circuitry and the high-power motor, providing isolation and protection while ensuring optimal performance.

The motor driver IC contains a 4 MOSFETs arranged in a configuration known as a H-bridge. These transistors switch on and off rapidly to generate a pulse-width-modulated (PWM) waveform that controls the amount of power delivered to the motor [10]. The PWM waveform can be adjusted to vary the frequency and amplitude of the voltage supplied to the motor, thereby controlling its speed and torque.

Overall, the motor driver IC is a critical component of a VFD, enabling precise control over the speed and direction of the motor while protecting it from damage due to overvoltage, overcurrent, or overheating. [11]

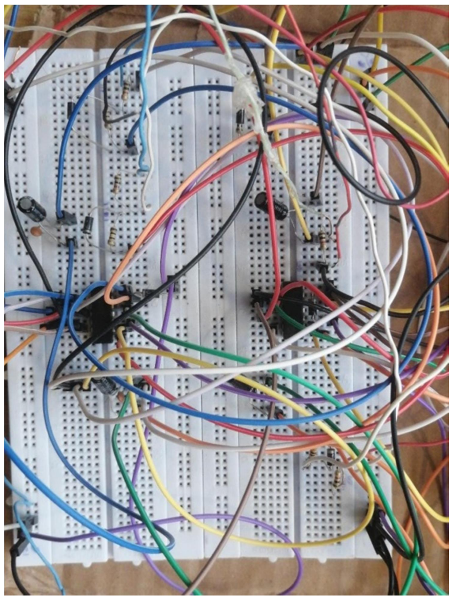


Figure 12 Implemented Motor Driver IC Circuit

C. *H – bridge*

The H bridge is a key component in a Variable Frequency Drive (VFD) because it allows for the control of the motor's speed and direction. The H bridge is essentially a circuit that can switch the direction of current flow in a motor by using four IRF840 8A MOSFETs. By switching the current flow in a specific way, the H bridge can control the voltage and frequency being supplied to the motor, which changes its speed.

In a VFD, the H bridge is used to convert the incoming AC power supply into a DC voltage, which is then used to generate a variable frequency AC output for the motor. By controlling the frequency and voltage of this output, the VFD can control the speed and torque of the motor.[12]

The H bridge also plays a crucial role in protecting the VFD and the motor from damage due to overcurrent or short circuits. When a fault occurs, the H bridge can quickly shut off the power supply to the motor, preventing further damage.

By controlling the timing of these switches, the VFD can create a series of voltage pulses that simulate a sinusoidal wave. The frequency and amplitude of these pulses can be varied to control the speed and torque of the motor. The VFD's microcontroller such as Arduino determines the proper timing of the switches based on inputs like the desired speed or torque, feedback from sensors like encoders and other factors like current limits to protect the motor and drive. [13]

Overall, a VFD provides a highly efficient way to control the speed and torque of an AC induction motor, compared to other methods like mechanical speed controllers or fixed-speed drives. Its ability to adjust the frequency and voltage of the output allows it to precisely match the motor's load requirements and minimize wasted energy.

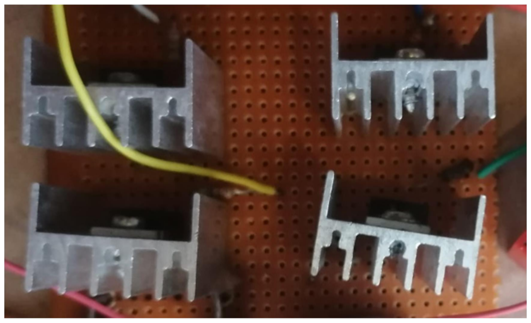


Figure 13 H-Bridge circuit implementation with MOSFETS

VI. SIMULATION RESULTS

The circuit was simulated using a Simulink to verify its performance.

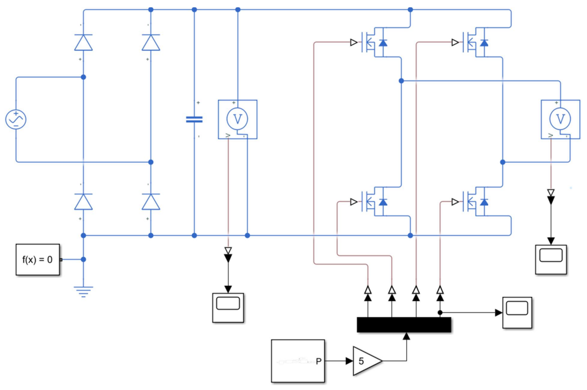


Figure 14 VFD Simulation on Simulink

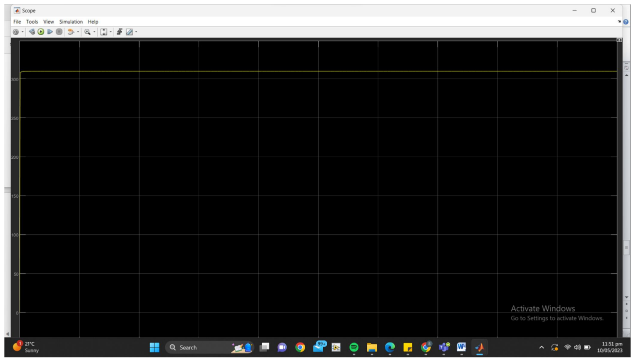


Figure 15 Waveform across Rectifier showing 312V output

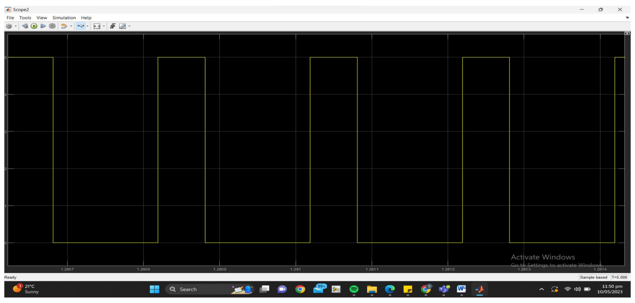


Figure 16 Gate Driver Waveform

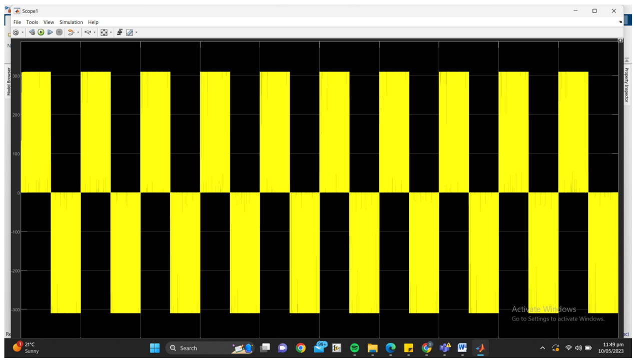
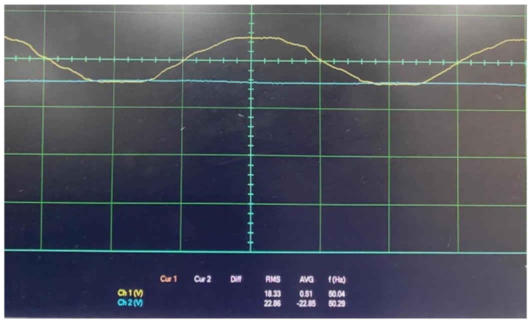


Figure 17 Output Waveform across MOSFETS

VII. TESTING RESULTS

During the testing phase, an input voltage of 312V was applied to the system, and the resulting waveforms were observed using the LVDAC EMS (Low Voltage Differential Analog-to-Digital Converter Electro-Magnetic System). This allowed for a comprehensive analysis of the electrical characteristics and behavior of the system under the specified voltage input. The waveforms obtained were carefully examined to assess the efficiency, stability, and integrity of the variable frequency drive (VFD) and its associated components. By analyzing the waveforms, it was possible to verify the accuracy of the AC to DC conversion, the smoothness of the output voltage, and the successful DC to AC conversion through the MOSFET network. These tests provided valuable insights into the performance of the system, confirming its capability to effectively handle the given voltage input and deliver the desired controlled speed for the induction motor. Figures below show the test waveforms:



```
          Cur 1   Cur 2   Diff    RMS     AVG     f(Hz)
Ch 1 (V)                          18.33   0.51    50.04
Ch 2 (V)                          22.86   -22.85  50.29
```

Figure 18 Resulting Waveform against Rectifier

- The yellow sinusoid shows the AC input. 🟨
- The blue straight line shows the DC output waveform against the capacitor. 🟦

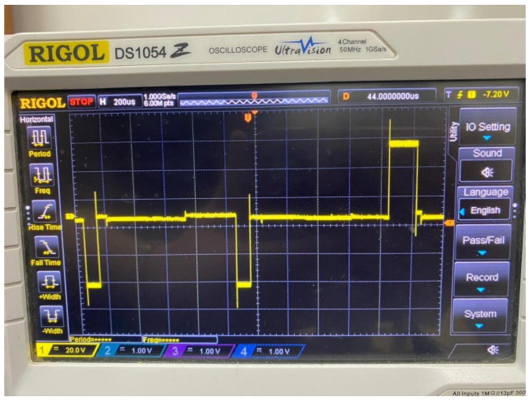


Figure 19 PWM from Arduino

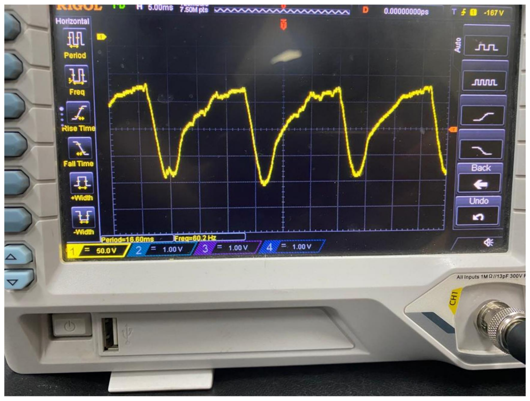


Figure 20 Final VFD Output

VIII. VERIFICATION OF RESULTS

Verification of Variable Frequency Drive (VFD) performance is an essential aspect of ensuring that the drive is operating as intended. Functional testing involves physically testing the drive under normal operating conditions to confirm that it meets its specifications. The comparison between the simulation waveforms and actually hardware waveform clearly intend to be quite similar as shown. This provides the testimonial of the actual VFD working appropriately as expected in accordance to the simulations. Graphs clearly provide valuable insights into the performance of the VFD; the real-world testing verifies the performance of the hardware. Ultimately, verifying the performance of the VFD is important not only for ensuring that the drive is operating efficiently.

IX. CONCLUSION

In summary, this project involved the detection of faces belonging to three group members and the creation of a comprehensive dataset comprising their images. The dataset was labeled and utilized for training a machine learning model using Python and OpenCV. The faces were successfully identified through the implementation of a machine learning-based classification algorithm. Furthermore, a variable frequency drive (VFD) was developed to enable control of the motor's speed based on face recognition. The VFD control system was divided into three segments: Analog Power to DC, DC to AC, and Control Unit. AC to DC conversion was achieved using a full wave rectifier bridge to convert single-phase AC power supply into DC power, with a capacitor employed for voltage smoothing. Subsequently, DC to AC conversion was carried out using an IGBT network to transform the DC supply back into AC supply. By controlling the frequency of the AC supply, the speed of the induction motor could be effectively regulated. The control unit played a vital role in generating appropriate signals for switching the IGBT network. Upon receiving signals from the computer via an established communication channel, the control unit generated corresponding signals for IGBT switching. These signals consisted of four unique values, with one corresponding to motor stoppage and the remaining three generating distinct speeds for the motor.

X. REFERENCES

[1] "Variable Frequency Drives - Energy.gov": https://www.energy.gov/sites/prod/files/2014/05/f15/Basics_of_Energy Efficient Motors and Drives.pdf

[2] Zawad Ali, Mohammad & Shabbir, Md. Nasmus Sakib Khan & Kawsar Zaman, Shafi Md & Liang, Xiaodong. (2020). Single- and Multi-Fault Diagnosis Using Machine Learning for Variable Frequency Drive-Fed Induction Motors. IEEE Transactions on Industry Applications. PP. 1-1. 10.1109/TIA.2020.2974151.

[3] Xiaoling Shu, Yiwan Ye, Knowledge Discovery: Methods from data mining and machine learning, Social Science Research, Volume 110,2023,102817, ISSN 0049-089X, https://doi.org/10.1016/j.ssresearch.2022.102817.

[4] Khalid K. Al-jabery, Tayo Obafemi-Ajayi, Gayla R. Olbricht, Donald C. Wunsch II,2 - Data preprocessing,Editor(s): Khalid K. Al-jabery, Tayo Obafemi-Ajayi, Gayla R. Olbricht, Donald C. Wunsch II,Computational Learning Approaches to Data Analytics in Biomedical Applications,Academic Press,2020,Pages 7-27,ISBN 9780128144824, https://doi.org/10.1016/B978-0-12-814482-4.00002-4.

[5] "Understanding Variable Frequency Drives - AutomationDirect": https://library.e.abb.com/public/7e429b407f8af6f5c1257d5a003a9d8b/BU_generaldoc_Basics_of_Variable_Frequency_Drives_EN_V1_0603.pdf

[6] https://colab.research.google.com/drive/1nX9y2DsYNTW0Q9NTrnGQviP-hB4OfGcy#scrollTo=q_q2yU9YLWuF

[7] https://github.com/ageitgey/face_recognition

[8] https://www.youtube.com/watch?v=tl2eEBFEHqM

[9] Keljik, J. J. (2012). Electricity 4: AC/DC Motors, Controls, and Maintenance. United States: Cengage Learning.

[10] Electrical & Electronics Abstracts. (1997). United Kingdom: Institution of Electrical Engineers.

[11] https://emcsolutions.com/2021/05/20/common-causes-of-vfd-failure/

[12] Petruzella, F. (2009). Electric Motors and Control Systems. Colombia: McGraw-Hill Education.

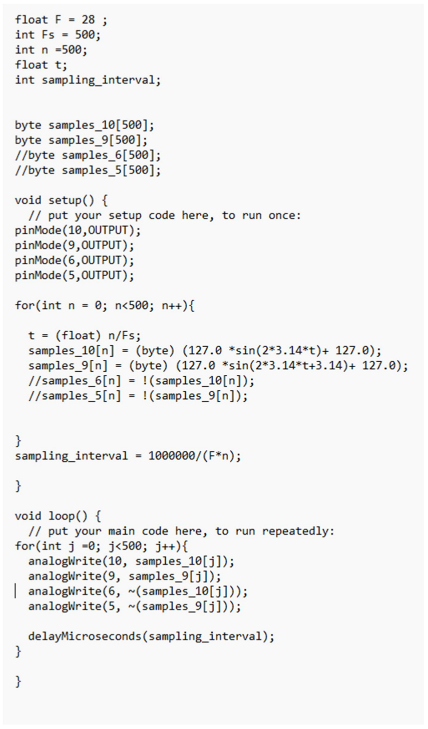
[13] https://vfds.com/blog/what-is-a-vfd/

[14] https://www.tensorflow.org/overview

APENDIX A

```
import os
import cv2 #Computer vision library
import dlib
import face_recognition #open source library

import serial #Serial communication library

port = '/dev/ttyUSB0'  # Update with your Arduino's port might be COM3 or something like that
baud_rate = 9600  # Update with your Arduino's baud rate
ser = serial.Serial(port, baud_rate)


# Load images of the three persons
person1_image = face_recognition.load_image_file("ahsan.jpeg")
person1_face_encoding = face_recognition.face_encodings(person1_image)[0]

person2_image = face_recognition.load_image_file("shazil.jpeg")
person2_face_encoding = face_recognition.face_encodings(person2_image)[0]

person3_image = face_recognition.load_image_file("usman.jpeg")
person3_face_encoding = face_recognition.face_encodings(person3_image)[0]

known_face_encodings = [person1_face_encoding, person2_face_encoding, person3_face_encoding]
known_face_names = ["Ahsan", "Shazil", "Usman"]

# Load an image to test
# Initialize the video capture
video_capture = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    # Resize frame for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (used by OpenCV) to RGB color (used by face_recognition)
    rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

    # Find all the faces and face encodings in the current frame
    face_locations = face_recognition.face_locations(rgb_small_frame)
    face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

    face_names = []
    for face_encoding in face_encodings:
        # Check if the face is a match for any of the known faces
        matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
        name = "Unknown"
        # If a match was found, use
the name of the known person
        if True in matches:
            match_index =
matches.index(True)
            name =
known_face_names[match_index]

        face_names.append(name)


ser.write(face_names.encode())  #
Convert string to bytes and send


    # Display the results
    for (top, right, bottom, left),
name in zip(face_locations,
face_names):
        # Scale back up the face
locations since the frame was resized
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

       # Draw a rectangle around the
face
        cv2.rectangle(frame, (left,
top), (right, bottom), (0, 0, 255),
2)

        # Draw a label with the name
below the face
        cv2.rectangle(frame, (left,
bottom - 35), (right, bottom), (0, 0,
255), cv2.FILLED)
        font =
cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name,
(left + 6, bottom - 6), font, 0.8,
(255, 255, 255), 1)

    # Display the resulting image
    cv2.imshow('Video', frame)


    # Press 'q' to quit
    if cv2.waitKey(1) & 0xFF ==
ord('q'):
        break

# Release the video capture
video_capture.release()
cv2.destroyAllWindows()
```

APENDIX B



```
float F = 28 ;
int Fs = 500;
int n =500;
float t;
int sampling_interval;


byte samples_10[500];
byte samples_9[500];
//byte samples_6[500];
//byte samples_5[500];

void setup() {
  // put your setup code here, to run once:
pinMode(10,OUTPUT);
pinMode(9,OUTPUT);
pinMode(6,OUTPUT);
pinMode(5,OUTPUT);

for(int n = 0; n<500; n++){

  t = (float) n/Fs;
  samples_10[n] = (byte) (127.0 *sin(2*3.14*t)+ 127.0);
  samples_9[n] = (byte) (127.0 *sin(2*3.14*t+3.14)+ 127.0);
  //samples_6[n] = !(samples_10[n]);
  //samples_5[n] = !(samples_9[n]);


}
sampling_interval = 1000000/(F*n);

}

void loop() {
  // put your main code here, to run repeatedly:
for(int j =0; j<500; j++){
  analogWrite(10, samples_10[j]);
  analogWrite(9, samples_9[j]);
  analogWrite(6, ~(samples_10[j]));
  analogWrite(5, ~(samples_9[j]));

  delayMicroseconds(sampling_interval);
}

}
```