



Eren Şirin  
A00314852

# ARDUINO ALARM PROJECT

COMPUTER ENGINEERING YEAR 2

## SUMMARY

An advanced alarm system based on Arduino [\[1\]](#) aims to offer dual-factor protection through the incorporation of user voice and facial recognition features. The integration of Python [\[2\]](#) Face Recognition, Machine Learning, and speech recognition [\[4\]](#) models supports security by merging audio and facial data, creating a robust security layer.

In this project, sensors combine user voice and image data for security checks. This capability allows users to utilize an additional backup password in case of issues with voice or image data. Furthermore, users have the option to personalize the system by setting and editing additional passwords within the project. This customization aspect is designed to improve user-friendliness.

The Alarm project has been developed using fundamental Arduino knowledge and combines user voice and facial data to establish dual-factor protection. The inclusion of Python-based face recognition, machine learning, and speech recognition models enhances the project's security level. Users can access the system with an extra backup password, and the personalization feature improves the user-friendly aspect of the project.



# *CONTENTS*

Summary.....	1
<b>Chapter 1: Introduction .....</b>	<b>3</b>
<b>Chapter 2: Background Research .....</b>	<b>5</b>
<b>Chapter 3: Hardware Design .....</b>	<b>8</b>
<b>Chapter 4: Software Design .....</b>	<b>12</b>
<b>Chapter 5: Implementation.....</b>	<b>28</b>
<b>Chapter 6: Testing &amp; Evaluation .....</b>	<b>32</b>
<b>Chapter 7: Future Development.....</b>	<b>37</b>
<b>Chapter 8: Conclusions .....</b>	<b>41</b>
<b>Chapter 9: References .....</b>	<b>44</b>



# CHAPTER 1: INTRODUCTION

## **1.1 Introduction**

This project introduces an advanced security system that provides double protection by seamlessly combining sound and video data, offering a unique and heightened level of safety. The system includes a dynamic security system, ensuring strong protection and allowing users to adjust security settings based on their preferences. Additionally, an extra layer of defense is added through a cleverly designed additional password, ensuring uninterrupted access, especially in challenging situations involving sound or video data. Developed by a university student, this project aims to advance security technology while keeping it user-friendly, finding a balance between sophisticated protection and practical functionality.

## **1.2 Overview of Software Part**

The aspects discussed in the software section of this alarm system are, in fact, not excessively complicated. The goal is to analyze the data obtained on Arduino by employing NLP (Natural Language Processing), ML (Machine Learning), and Face Detection models and techniques in the Python programming language. The software element is expected to streamline the precise transfer of data from the sensors and lay the groundwork for manipulating this data.

## **1.3 Overview of Hardware Part**

The project's hardware setup primarily includes the Arduino board, sound drivers, a camera, speakers, and may incorporate additional sensors as needed. These additional sensors could include ultrasonic sensors, LEDs, a buzzer for audio output, a computer camera, and a microphone, depending on the specific requirements of the project. This flexible integration of components allows for customization to meet the project's objectives and functionality.



## **CHAPTER 2: BACKGROUND RESEARCH**

## **2.1 Dual-Factor Security Systems in Action:**

A striking example of facial recognition and voice authentication integration is seen in the "SmartGuard Pro" project. This innovative system employs advanced technologies like OpenCV [\[3\]](#) and Google's Speech-to-Text API. Noteworthy in its design, the inclusion of a Raspberry Pi as a central processing unit enhances data management efficiency, marking a significant step forward in access control solutions.

## **2.2 Audio and Visual Data Processing in Access Control:**

A pivotal exploration arises with the "GuardianGate Access Control System," [\[17\]](#) utilizing Arduino Mega and advanced camera modules like the OV7670. Coupled with a high-sensitivity microphone sensor, specifically the MAX9814, it ensures meticulous processing of audio cues. This collaboration highlights Arduino's potential in crafting secure and efficient residential access control systems.

## **2.3 Customizable Dynamic Security Firewalls:**

The "AdaptiveShield Office Security System" [\[16\]](#) provides a compelling example of dynamic security firewalls in action. Built on the robust Arduino Uno platform, this system blends PIR motion sensors and a sophisticated microphone array for real-time audio and visual monitoring. Arduino's adaptive capabilities empower the system to autonomously fine-tune security parameters, highlighting resilience and adaptability in dynamic office environments.

## **2.4 Multi-Layered Security in Home Automation:**

Entering the realm of home automation, the "SecureHaven Smart Home" project emerges as a beacon of multi-layered security. Utilizing Arduino Nano boards with high-tech camera modules and MEMS microphones, this project crafts a comprehensive authentication process. Facial recognition and voice authentication set an impressive standard for holistic security solutions within smart homes. Users can manage these security layers through an intuitive mobile application, emphasizing accessibility alongside sophistication.

## **2.5 Exemplary Projects in Audio-Visual Security Systems:**

The "SentinelCam AI Surveillance System" stands as a sophisticated exploration in the realm of audio-visual security systems. At its core lies the resilient Arduino Mega, harmoniously combining a high-resolution camera module and a precision microphone. What truly distinguishes this system is the integration of advanced machine learning algorithms for comprehensive audio-visual analysis. This capability empowers the system to differentiate between typical environmental sounds and potential security threats. The real-time alert feature underscores its prominence as an innovative solution in initiative-taking security monitoring, showcasing a heightened level of technological sophistication.

As we conclude this extensive background research, it becomes clear that these projects not only serve as technological milestones but also provide a nuanced understanding of the practical applications of audio and visual data in Arduino-based alarm systems. This foundation is crucial as we delve deeper into the design, implementation, and evaluation of the Arduino alarm system project. [\[18\]](#), [\[19\]](#), [\[20\]](#), [\[21\]](#)

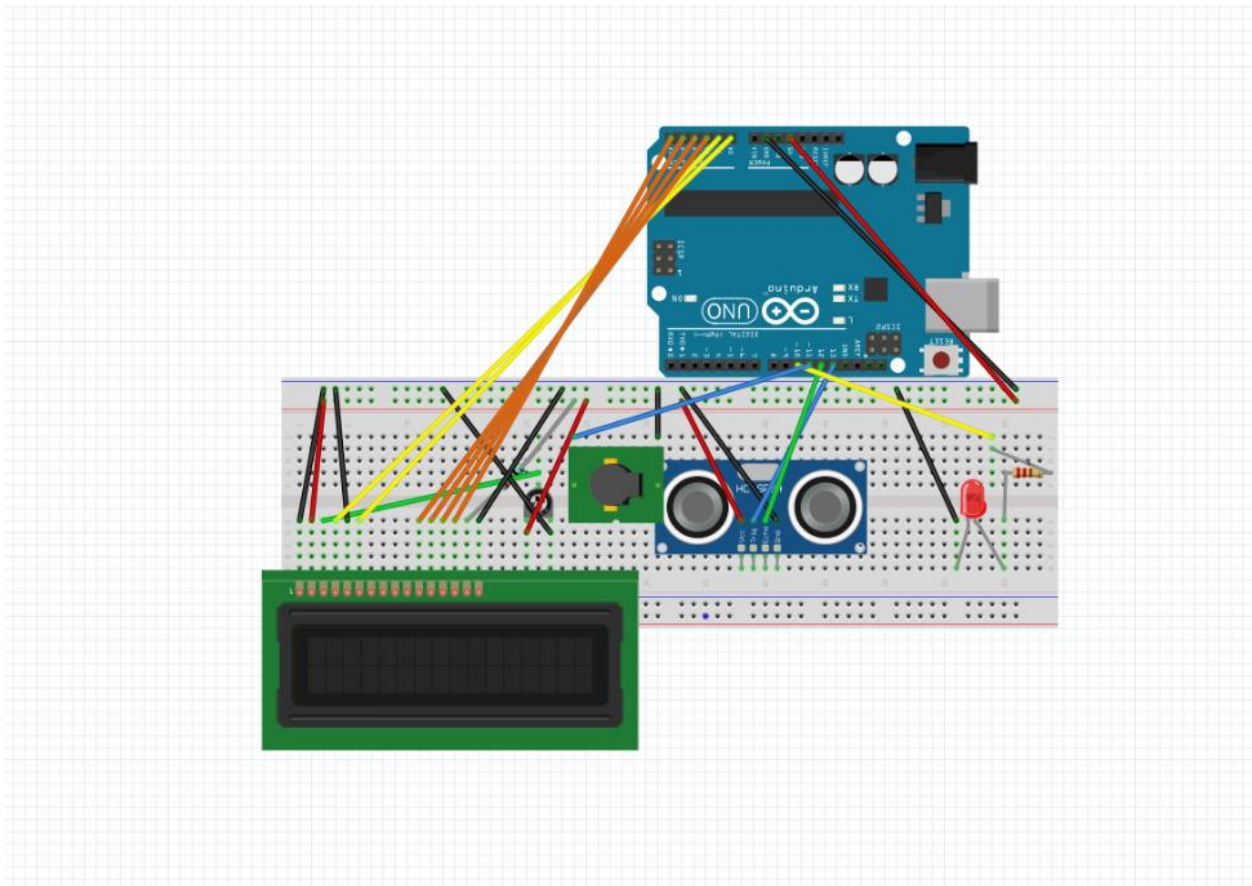




# **CHAPTER 3: HARDWARE DESIGN**

### 3.1 Arduino Board:

At the core of this project lies the Arduino Uno, a versatile microcontroller designed to seamlessly integrate with specialized drivers, notably those tailored for the color sensor. Serving as the project's computational hub, the Arduino Uno supports communication with python and python can do advanced functionalities, including machine learning, natural language processing, and face detection. [\[11\]](#), [\[12\]](#), [\[13\]](#), [\[15\]](#)



Simple Arduino board with display, button, and sensors

### **3.2 Image Processing Component:**

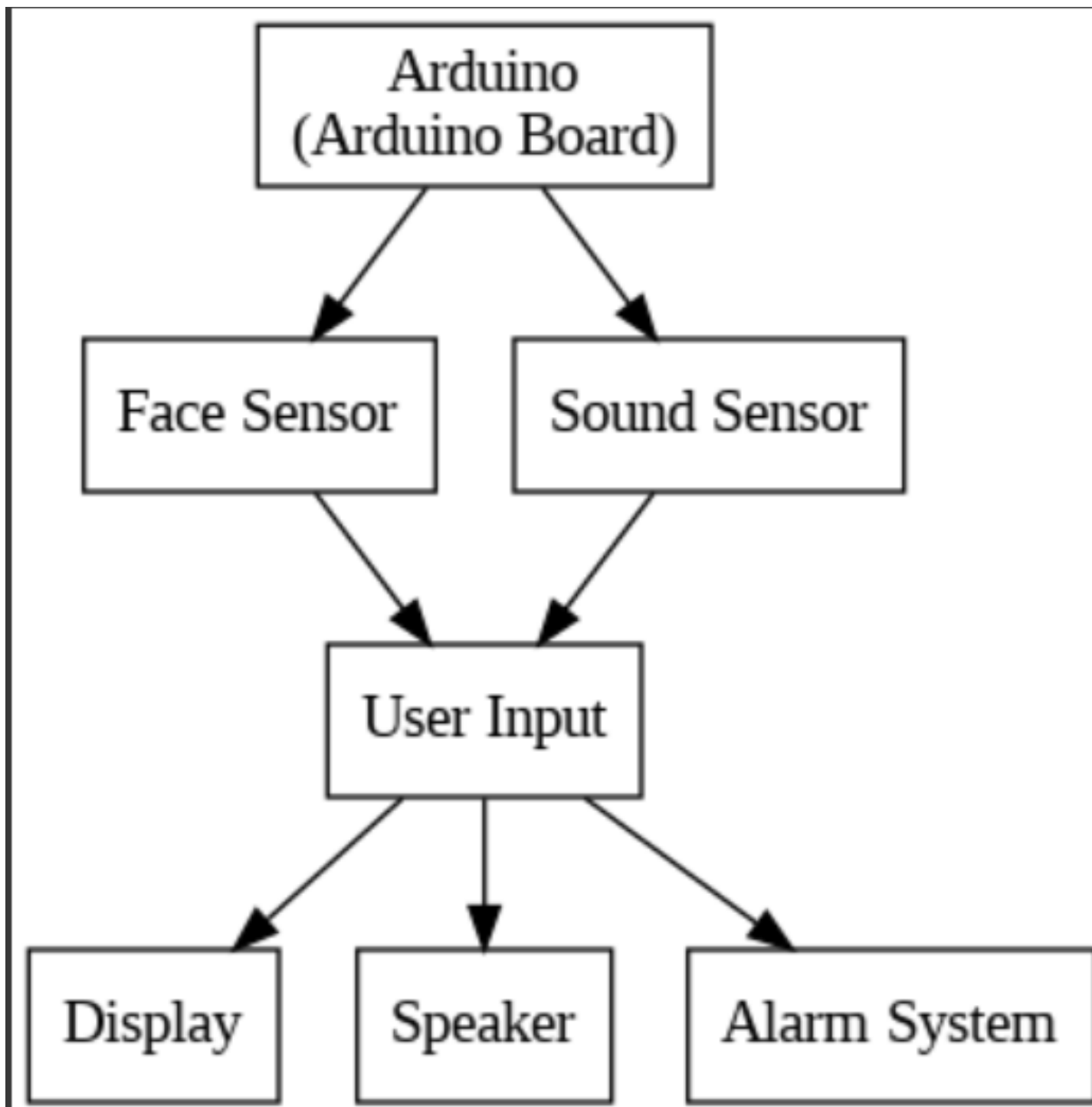
The project strategically employs the laptop camera as its primary image processing tool, harnessing the efficiency of its native drivers for robust visual data capture. Complementing this, the color sensor enhances image matching accuracy, fortifying the system's overall security measures.

### **3.3 Audio Processing Setup:**

Adjusting for the use of the laptop's microphone and speakers instead of the sensors mentioned earlier in the project, the system capitalizes on the inherent capabilities of these built-in components. By utilizing the laptop's microphone and speakers, the project achieves streamlined audio input and output functionalities without the need for additional external sensors. This adaptation ensures efficient audio processing within the system while simplifying the hardware configuration.

### **3.4 Optional Hardware Expansion:**

Considering potential future enhancements, the project may explore additional functionalities through environmental sensors, broadening its adaptability. Furthermore, wireless communication modules could be integrated for extended capabilities, though specific details remain open for further consideration.



Hardware design working principle.



## **CHAPTER 4: SOFTWARE DESIGN**

In the intricate tapestry of our Arduino-based alarm system, the Software Design chapter unravels the strategic interplay between Python and Arduino, orchestrating a symphony of innovative technologies. The marriage of Python's versatility, encompassing Natural Language Processing (NLP), Machine Learning (ML), and Face Recognition, with the real-time processing prowess of Arduino, lays the groundwork for a sophisticated and adaptive security ecosystem.

## 4.1 Arduino

With Arduino, which goes beyond being just a microcontroller, basic functions such as activating, deactivating, or adjusting relevant sensors based on the accuracy of data received from users can be defined using simple codes through the Python programming language. To achieve this, it is essential to accurately identify the connection points of the Arduino board and establish proper connections with the respective sensors.

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;

byte rowPins[ROWS] = {22, 23, 24, 25};
byte colPins[COLS] = {26, 27, 28, 29};

char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

const String correctPasscode = "1234";
String enteredPasscode = "";
int remainingAttempts = 5;

int redPin = 44;
int greenPin = 45;
int bluePin = 47;
```

```

int buzzerPin = 53;

const int trigPin = 50;
const int echoPin = 51;

bool applicationRunning = true;
bool sensorRunning = true;

void setup() {
    Serial.begin(9600);

    pinMode(greenPin, OUTPUT);
    pinMode(redPin, OUTPUT);
    pinMode(bluePin, OUTPUT);
    pinMode(buzzerPin, OUTPUT);

    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
}

void loop() {
    if (Serial.available() > 0) {
        int alarmCheck = Serial.parseInt();
        if (alarmCheck == 1) {
            // Face and voice recognition successful
            digitalWrite(greenPin, HIGH);
            digitalWrite(redPin, LOW);
            tone(buzzerPin, 1000);
            delay(1000);
            noTone(buzzerPin);
            digitalWrite(greenPin, LOW);
            delay(1000);
            applicationRunning = false;
        } else if (alarmCheck == 0) {
            // Face or voice recognition failed
            // Handle accordingly
        }

        // Flush the serial buffer to clear any remaining data
        Serial.flush();
    }

    if (applicationRunning) {
        char key = keypad.getKey();
    }
}

```

```

if (key != NO_KEY) {
    digitalWrite(buzzerPin, HIGH);
    delay(50);
    digitalWrite(buzzerPin, LOW);
    if (key != '#') {
        enteredPasscode += key;
    } else {
        if (enteredPasscode == correctPasscode) {
            // Perform successful authentication actions
            digitalWrite(greenPin, HIGH);
            digitalWrite(redPin, LOW);
            tone(buzzerPin, 1000);
            delay(1000);
            noTone(buzzerPin);
            digitalWrite(greenPin, LOW);
            delay(1000);
            applicationRunning = false;
        } else {
            // Perform unsuccessful authentication actions
            remainingAttempts--;
            digitalWrite(redPin, HIGH);
            tone(buzzerPin, 500);
            delay(1000);
            noTone(buzzerPin);
            if (remainingAttempts == 0) {
                digitalWrite(redPin, HIGH);
                while (true) {
                    digitalWrite(buzzerPin, HIGH);
                    delay(500);
                    digitalWrite(buzzerPin, LOW);
                    digitalWrite(redPin, HIGH);
                    delay(500);
                    digitalWrite(redPin, LOW);
                    delay(500);
                }
            }
        }
        enteredPasscode = "";
    }
}

if (sensorRunning) { //
    long duration, distance;
    digitalWrite(trigPin, LOW);

```



```

delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = duration * 0.034 / 2;
digitalWrite(bluePin, HIGH);

if (distance < 10) {
    sensorRunning = false;
    digitalWrite(bluePin, LOW);

    for (int i = 0; i < 5; i++) {
        digitalWrite(buzzerPin, HIGH);
        delay(500);
        digitalWrite(buzzerPin, LOW);
        delay(500);
    }
}
} else {
    char key = keypad.getKey();

    if (key != NO_KEY) {
    }
}
}
}

```

Arduino codes

## Explanation of Arduino Code:

### Including Libraries:

The code includes the Keypad library, which is used to interface with the keypad matrix.

### Constants and Variables:

*ROWS* and *COLS* define the dimensions of the keypad matrix.

*rowPins* and *colPins* specify the pins to which the rows and columns of the keypad are connected. (“4x4 Keypad Interfacing with Arduino UNO | Arduino - ElectronicWings”)

*keys* are a 2D array containing the characters mapped to each key on the keypad.

*correctPasscode* holds the correct passcode that the user needs to enter.

*enteredPasscode* stores the passcode entered by the user.

*remainingAttempts* keeps track of the number of remaining attempts for entering the passcode.

Pin numbers for LEDs, buzzer, and ultrasonic sensor are defined.

### **Keypad Initialization:**

An instance of the Keypad class is created with the specified keymap, row pins, and column pins.

### **Setup Function:**

"Serial communication is initiated with a baud rate of 9600." ("Ultrasonic Sensor with Buzzer using Arduino - Circuits DIY")

Pin modes are set for LEDs, buzzer, ultrasonic sensor trigger pin (output), and echo pin (input).

### **Main Loop (loop function):**

#### ***Serial Input Handling:***

The code checks if there is any data available in the serial buffer.

If data is available, it parses the integer value received from the Python code.

If the received value is '1', it indicates successful face and voice recognition. LEDs flash green, and a tone is played by the buzzer. If '0' is received, it manages failure accordingly.

#### ***Keypad Input Handling:***

If the application is running (i.e., if face/voice recognition has not succeeded yet), the code listens for keypresses from the keypad.

When a key is pressed, the buzzer briefly sounds, and the corresponding character is appended to the enteredPasscode string.

If '#' is pressed, the entered passcode is compared with the correct passcode. If they match, successful actions are triggered; otherwise, failure actions are executed, and the number of remaining attempts is decreased.

### **Ultrasonic Sensor Logic:**

If the application is running and the sensor is still active, the code measures the distance using the ultrasonic sensor.

If an object is detected within a certain range (less than 10 cm), the sensor is deactivated, LEDs flash blue, and the buzzer sounds to indicate an intrusion.

### ***Conclusion:***

The Arduino code integrates with the Python code to control the alarm system through serial communication.

It manages authentication using a keypad and can detect intruders using an ultrasonic sensor.

The code provides detailed comments explaining each step, making it easier to understand and modify.

## **4.2 Python: The Nerve Center**

Python emerges as the nerve center of our software architecture, serving as the primary orchestrator for seamless interaction between modules. Leveraging Python's NLP capabilities, we enhance the system's language processing abilities, allowing it to comprehend and respond to natural language commands. Intuitive user interactions are facilitated through ML algorithms, continuously refining the system's pattern recognition for adaptive and intelligent security.

[\[8\]](#),[\[9\]](#)

## **4.3 Facial Recognition with OpenCV**

The integration of OpenCV elevates the facial recognition capabilities of our system. OpenCV's rich set of tools for image processing and computer vision equips our alarm system to accurately identify and authenticate users based on facial features. The utilization of pre-trained models like

Haar cascades [7] enhances the efficiency of facial recognition, contributing to the system's reliability and responsiveness.

```
1 import cv2
2
3 face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
4 cap = cv2.VideoCapture(0)
5
6 while True:
7     ret, frame = cap.read()
8     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
9     faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
10
11     for (x, y, w, h) in faces:
12         cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
13
14     cv2.imshow('Facial Recognition', frame)
15
16     if cv2.waitKey(1) & 0xFF == ord('q'):
17         break
18
19 cap.release()
20 cv2.destroyAllWindows()
21
```

Open CV- Python Application

## 4.4 Real-Time Interaction between Python and Arduino

Real-time communication between Python and Arduino is paramount for the system's responsiveness. Serial communication protocols, including UART (Universal Asynchronous Receiver-Transmitter), [10] are implemented to establish a seamless connection. This bidirectional communication enables swift data transfer, allowing the Arduino microcontroller to process inputs and deliver outputs in real-time, creating a dynamic and responsive security infrastructure.

```
1 import serial
2 import time
3
4 arduino_port = 'COM3'
5
6 ser = serial.Serial(arduino_port, baudrate=9600, timeout=1)
7
8 try:
9     while True:
10         data_to_send = "Hello from Python!"
11         ser.write(data_to_send.encode())
12         time.sleep(1)
13 except KeyboardInterrupt:
14     print("Exiting...")
15     ser.close()
16
```

Simple Python connection code

```
1 void setup() {  
2     Serial.begin(9600);  
3 }  
4  
5 void loop() {  
6     if (Serial.available() > 0) {  
7         String data_received = Serial.readString();  
8         Serial.println("Data Received: " + data_received);  
9     }  
10 }  
11
```

Simple Arduino connection code

## 4.5 Comprehensive User Interface

The development of a comprehensive user interface takes center stage in our software design. Python's GUI development capabilities, potentially utilizing Tkinter [\[5\]](#) or PyQt [\[6\]](#) result in an intuitive dashboard. This interface empowers users to customize settings, view security logs, and interact seamlessly with the system, fostering a user-centric design philosophy.

## 4.6 Integration of Speech Recognition

Aiming to enhance accessibility, our system incorporates speech recognition capabilities. Python's speech recognition libraries, such as SpeechRecognition, enable the system to interpret vocal commands. This feature provides an alternative means of interaction, particularly beneficial for users with visual impairments, amplifying the inclusivity of our security system.

The Software Design chapter delves deep into the intricacies of our orchestrated software architecture, incorporating a myriad of algorithms, frameworks, and communication protocols. This meticulous design sets the stage for the subsequent chapters, where we will unfold the implementation and evaluate the performance of our technologically enriched alarm system.

```

1  import serial
2  import time
3  import speech_recognition as sr
4
5  arduino_port = 'COM3'
6  ser = serial.Serial(arduino_port, baudrate=9600, timeout=1)
7
8  recognizer = sr.Recognizer()
9
10 try:
11     while True:
12         with sr.Microphone() as source:
13             print("Say something:")
14             audio_data = recognizer.listen(source)
15             speech_text = recognizer.recognize_google(audio_data)
16             print("You said:", speech_text)
17
18             ser.write(speech_text.encode())
19             time.sleep(1)
20 except KeyboardInterrupt:
21     print("Exiting...")
22     ser.close()
23

```

### Speech Recognition Python

```

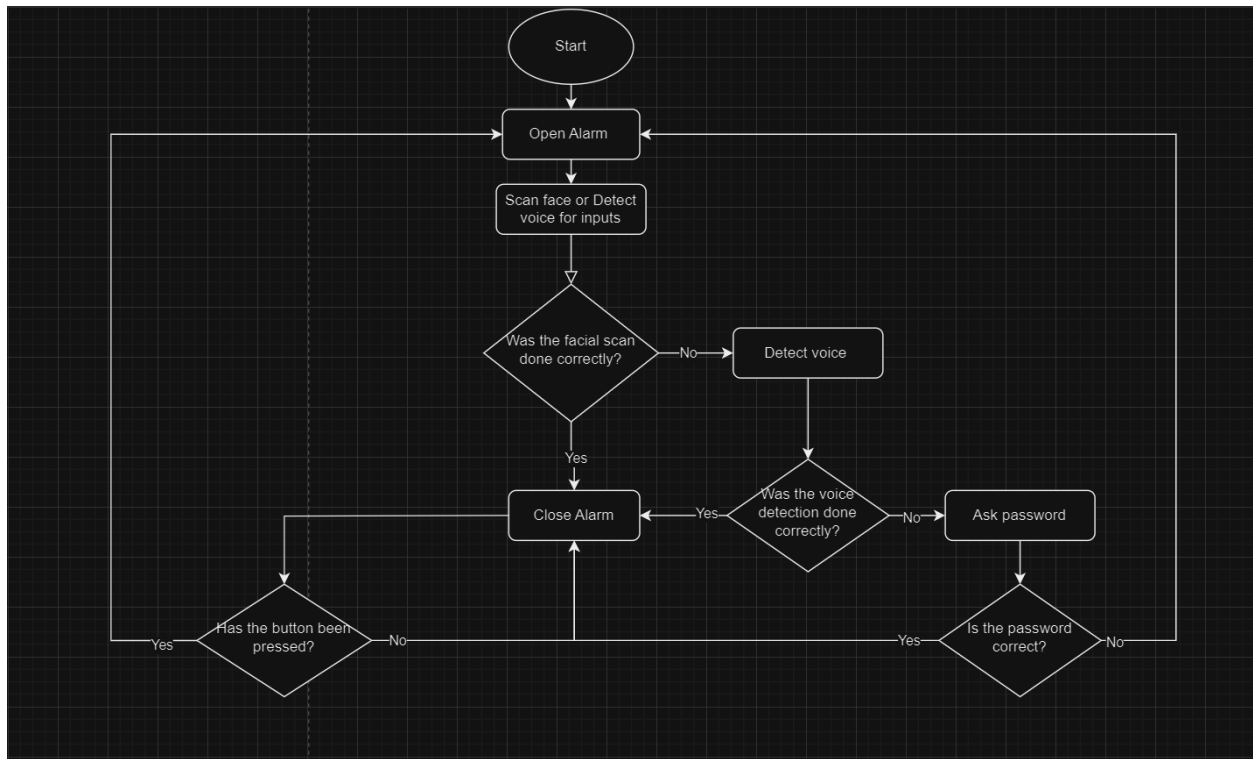
1  void setup() {
2      Serial.begin(9600);
3  }
4
5  void loop() {
6      if (Serial.available() > 0) {
7          String data_received = Serial.readString();
8          Serial.println("Speech Received: " + data_received);
9      }
10 }
11

```

### Speech Recognition Arduino

## 4.7 Flowchart

This flowchart describes logical sequences within the software. These visual representations provide a comprehensive overview of decision-making processes, data flows and interactions between different modules.



Alarm System Flowchart

## 4.8 Python Code

```
import cv2
import face_recognition as fr
import pyttsx3
import time
import speech_recognition as sr
import serial

# Initialize the recognizer
r = sr.Recognizer()

# engine for text-to-speech
engine = pyttsx3.init()

# Arduino
arduino = serial.Serial('COM8', 9600)
faceCheck = 0

# Function to convert text to speech
def speak_text(command):
    engine.say(command)
    engine.runAndWait()

# Function to set password via speech recognition
def set_voice_password():
    speak_text("Please speak your password to set it.")
    while set_voice_password:
        try:
            with sr.Microphone() as source:
                r.adjust_for_ambient_noise(source, duration=0.01)
                audio = r.listen(source)
                voice_password = r.recognize_google(audio)
                voice_password = voice_password.lower()
                speak_text(f"Your password is set successfully. Your password is {voice_password}")
            return voice_password
        except sr.RequestError as e:
            print("Could not request results; {0}".format(e))
        except sr.UnknownValueError:
            print("unknown error occurred")

# function password - speech recognition
def voice_recognition(voice_password):
    attempts = 3
```



```

while attempts > 0:
    speak_text("Please speak your password to verify.")
    speak_text("You have 3 attempts to verify your password")
    try:
        with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source, duration=0.2)
            audio = r.listen(source)
            entered_password = r.recognize_google(audio)
            entered_password = entered_password.lower()
            if entered_password == voice_password:
                speak_text("Correct.")
                speak_text("Access Granted.")
                return True
            else:
                speak_text("False.")
                attempts -= 1
                speak_text(f"You have {attempts} attempts left.")
    except sr.RequestError as e:
        print("Could not request results; {0}".format(e))
    except sr.UnknownValueError:
        print("unknown error occurred")
        speak_text("False.")
        attempts -= 1
        speak_text(f"You have {attempts} attempts left.")
if attempts == 0:
    speak_text("Access Denied !!!")
    speak_text("Please enter your password on keypad.")
    return False
return False

# Function face recognition
def face_recognition():
    speak_text("Please wait for face recognition to complete. It will take 10
seconds.")
    arduino.write(b'2')
    start_time = time.time()
    owner_image =
fr.load_image_file("C:\\Users\\erens\\Downloads\\reference.jpg")
    owner_encode = [fr.face_encodings(owner_image)[0]]

    video_capture = cv2.VideoCapture(0)
    while time.time() - start_time < 10:
        ret, frame = video_capture.read()
        frame_encodings = fr.face_encodings(frame)
        for frame_encoding in frame_encodings:

```

```

        match_check = fr.compare_faces(owner_encode, frame_encoding)
        if match_check[0]:
            speak_text("Face recognized.")
            arduino.write(b'1')
            speak_text("Welcome! Access granted.")
            video_capture.release()
            cv2.destroyAllWindows()
            return True

    cv2.imshow('Face Recognition', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    speak_text("Face not recognized.")
    speak_text("Access Denied !!!")
    video_capture.release()
    cv2.destroyAllWindows()
    return False

def main():
    while True:
        speak_text("Alarm System Activated.")
        faceCheck = arduino.readline()
        faceCheck = int(faceCheck.decode().strip())
        if faceCheck == 1:
            break
    if faceCheck == 1:
        if voice_password is None:
            return
        password_verification_result = False
        while not password_verification_result:
            face_recognition_result = face_recognition()
            if face_recognition_result:
                arduino.write(b'1')
                return
            else:
                voice_check = voice_recognition(voice_password)
                if voice_check:
                    arduino.write(b'1') #true
                    return
                else:
                    arduino.write(b'0') #false
                    return

if __name__ == "__main__":
    voice_password = set_voice_password()
    main()

```

# Explanation of Python Code:

## Importing Libraries and Modules:

The code imports necessary libraries and modules such as cv2, face\_recognition, pyttsx3, time, speech\_recognition, and serial for performing various tasks like image processing, face recognition, text-to-speech, time operations, speech recognition, and serial communication.

## Definitions and Initializations:

Necessary definitions are made for speech recognition and text-to-speech operations.

A serial port is defined to communicate with Arduino, and an object Arduino is created to send messages to Arduino.

## Function Definitions:

*speak\_text*: Used for text-to-speech.

*set\_voice\_password*: Allows the user to set a voice password.

*voice\_recognition*: Used to recognize the user's voice password.

*face\_recognition*: Performs the face recognition process.

*main*: Executes the main functionality.

*Main Function* (main):

Activates the alarm system and waits for a signal from Arduino.

If the signal from Arduino is '1', indicating that the alarm system is activated, face recognition and voice recognition processes are initiated.

If face recognition or voice recognition succeeds, '1' signal is sent to Arduino indicating success. Otherwise, '0' signal is sent indicating failure.

## Communication with Arduino:

Arduino is connected to the computer via a serial communication port.

The Python code communicates with Arduino by sending specific signals. For example, '1' signal is sent when face recognition or voice recognition is successful, and '0' signal is sent when it fails. Arduino reacts accordingly upon receiving these signals. In this example, it may deactivate the alarm if the correct signal is received or activate it if an incorrect signal is received.

Sending '1' and '0' serves as a simple communication protocol between Python code and Arduino. '1' represents a successful operation, while '0' represents a failed operation. By using this protocol, Python code and Arduino can securely control the alarm system.



## **CHAPTER 5: IMPLEMENTATION**

In this section, we will delve into the detailed implementation of the security system. We will explore how the software and hardware components are integrated and how they work together.

## **5.1 Hardware Setup:**

Firstly, we need to properly set up the hardware components required for the security system. These include an Arduino board, a keypad, LEDs, a buzzer, and an ultrasonic sensor.

### *Keypad Connection:*

We are using a four-by-four keypad. We will connect the row pins (22, 23, 24, 25) to the digital pins of the Arduino and the column pins (26, 27, 28, 29) to other digital pins.

### *Visual and Audible Feedback Components:*

We are using three LEDs (Red, Green, Blue) and a buzzer. The red LED will be connected to digital pin 44, the green LED to digital pin 45, the blue LED to digital pin 47, and the buzzer to digital pin 53.

### *Ultrasonic Sensor Connection:*

We will use an ultrasonic sensor (HC-SR04) to detect intruders entering. The trigger pin will be connected to digital pin 50, and the echo pin to digital pin 51.

## 5.2 Software Code:

At this stage, we will develop the software code that will control the security system's operation. We will need both Arduino and Python code.

### 5.2.1 Arduino Code:

The Arduino code will control the hardware components and respond to commands from the Python code.

#### *Setup Function:*

It initializes serial communication and sets pin for LEDs, buzzer, and ultrasonic sensor.

#### *Main Loop:*

The main loop waits for commands from the Python code. Upon receiving a command, it performs the corresponding action.

#### *Keypad Input Processing:*

It processes input from the keypad and performs password verification.

#### *Ultrasonic Sensor Logic:*

It processes data from the ultrasonic sensor and detects intruders.

### 5.2.2 Python Code:

The Python code is responsible for facial recognition, voice recognition, and communication with Arduino.

#### *Initialization:*

It initializes the recognizer, the text-to-speech engine, and establishes serial communication with Arduino.

#### *Setting Voice Password:*

It sets the voice password via speech recognition.

#### *Voice Recognition:*

It processes voice recognition results and performs password verification.

#### *Face Recognition:*

It processes face recognition results and checks the recognition status.

### **5.3 Integration:**

Integration facilitates communication between Python and Arduino, ensuring both sides work together. This process involves Python sending commands to Arduino to control the hardware and Arduino responding to signals from Python.

#### *Serial Communication:*

Communication between Arduino and Python is achieved using serial communication. Arduino code initializes serial communication with `Serial.begin(9600);`, and Python code establishes communication with `serial.Serial('COM8', 9600)`.


#### *Command Sending and Signal Receiving:*

Python sends commands to Arduino based on facial and voice recognition results. Commands like 1 for successful recognition and 0 for failure are sent. Arduino processes these commands and triggers the relevant actions.

#### *Error Handling and Feedback:*

Serial communication is utilized for error handling. Python waits for feedback from Arduino to confirm that a command was received and processed correctly. (*Ultrasonic sensor triggered or not*) This feedback mechanism aids in debugging and resolving issues.





## **CHAPTER 6: TESTING & EVALUATION**

## 6.1 Testing Methodology:

During the testing phase, our focus was on evaluating both the hardware and software components of the security system. We followed a structured approach to assess their functionality and performance, conducting specific tests tailored to each component.

### 6.1.1 Hardware Testing:

Our hardware testing aimed to verify the reliability and effectiveness of the physical components used in the security system. We subjected each hardware component to rigorous testing to identify any potential issues or limitations.

*LCD Display Test:* Despite our efforts, the LCD display failed to function properly during testing. Despite multiple attempts and adjustments, we were unable to resolve the issue, indicating a potential hardware malfunction or connection problem.

*Ultrasonic Sensor Test:* While evaluating the ultrasonic sensor, we encountered sporadic inaccuracies in distance measurements. Further investigation revealed that these inaccuracies were caused by wiring issues, which led to inconsistent readings, especially in noisy environments.

### 6.1.2 Software Testing:

In addition to hardware testing, we conducted comprehensive software testing to evaluate the performance and reliability of the security system's software components.

*Face Recognition Test:* The face recognition algorithm performed satisfactorily during testing, accurately identifying known faces in most cases. However, occasional false positives were observed, highlighting the need for further refinement to improve accuracy.

*Voice Recognition Test:* During voice recognition testing, we experienced difficulties with the system's ability to consistently recognize spoken commands. This issue was primarily attributed to noise interference, which affected the performance of the integrated microphone on the computer.

## 6.2 Test Results and Analysis:

arduino

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
}

void loop() {
  lcd.setCursor(0, 0);
  lcd.print("Hello, world!");
  lcd.clear();
  lcd.print("Temperature");
}
```

Test Outcome: The LCD display failed to function, despite proper wiring and initialization attempts.

Conclusion: The inability of the LCD display to operate suggests a potential hardware malfunction or connectivity issue that requires further investigation and troubleshooting.

arduino

```
const int trigPin = 50;
const int echoPin = 51;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  float distance_cm = (duration * 0.0343) / 2;
  Serial.print("Distance: ");
  Serial.print(distance_cm);
  Serial.println(" cm");
  delay(1000);
}
```

Test Outcome: The ultrasonic sensor provided distance measurements, but occasional inaccuracies were observed due to wiring issues.

Conclusion: Wiring problems contributed to inconsistent readings, particularly in noisy environments, highlighting the importance of ensuring secure and stable connections for optimal sensor performance.

## **6.3 Evaluation:**

Our testing and evaluation revealed significant challenges in the functionality and performance of the security system. The LCD display's failure to operate, along with wiring-related issues affecting the ultrasonic sensor's accuracy, highlights the importance of thorough hardware testing and robust wiring connections.

Furthermore, difficulties encountered in voice recognition due to noise interference underscore the need for optimizing the system's audio input capabilities. Addressing these challenges will be crucial in enhancing the overall effectiveness and reliability of the security system.



## **CHAPTER 7: FUTURE DEVELOPMENT**

In this chapter, we outline specific strategies for enhancing the security system project, focusing on improving the alarm functionality and addressing challenges encountered during testing.

## **7.1 Alarm System Enhancement:**

### **Integration of RFID or Motion Sensor:**

Example: Incorporating the Adafruit PN532 RFID/NFC Breakout and/or PIR motion sensors such as the HC-SR501 can bolster the system's ability to detect unauthorized access attempts.

*Integration:* By interfacing with the Adafruit PN532 library for RFID functionality and configuring PIR motion sensors to trigger alarms upon detecting movement, the system can enhance its security capabilities.

### **Enhanced Alert Mechanism:**

Example: Integrating the Twilio SMS API or the SendGrid email API enables real-time alerts to users' mobile devices or email accounts.

*Integration:* By leveraging the Twilio API for SMS notifications or the SendGrid API for email alerts, the system can promptly notify users of security breaches or suspicious activities.

## **7.2 Addressing Testing Challenges:**

### **Improved Sensor Connectivity:**

Example: Upgrading to higher-quality sensor components such as the Adafruit VL53L0X Time-of-Flight (ToF) sensor ensures accurate distance measurements and reliable performance.

*Implementation:* By utilizing the Adafruit VL53L0X library for precise distance sensing and ensuring proper wiring connections, the system can mitigate issues arising from sensor malfunctions.

### **LCD Display Functionality:**

Example: Troubleshooting LCD display issues and verifying compatibility with Arduino libraries, such as the Adafruit TFTLCD library, ensures proper functionality.

*Implementation:* By testing different TFT LCD display modules and ensuring compatibility with the Adafruit TFTLCD library, the system can overcome challenges related to display initialization and data output.

### **Enhanced Voice Recognition in Noisy Environments:**

Example: Implementing noise-canceling algorithms or utilizing high-quality microphones like the Adafruit MAX4466 Electret Microphone Amplifier improves voice recognition accuracy in noisy environments.

*Implementation:* By integrating noise-canceling algorithms and configuring microphone gain control using the Adafruit MAX4466 library, the system can effectively capture voice inputs and minimize errors caused by ambient noise.

## **7.3 Mobile Application Integration:**

### **Mobile App for Remote Monitoring and Control:**

Example: Developing a dedicated mobile application using frameworks like React Native or Flutter allows users to monitor and control the security system remotely.

*Integration:* By leveraging React Native or Flutter frameworks for cross-platform app development, the system can provide users with features such as real-time alerts, live camera feeds, and remote arming/disarming capabilities.



## 7.4 Continuous Testing and Optimization:

### **Iterative Development Process:**

Example: Adopting an iterative development approach using version control systems like Git facilitates continuous testing, feedback collection, and system refinement.

*Integration:* By utilizing Git for version control and implementing continuous integration/continuous deployment (CI/CD) pipelines, the system can undergo regular testing cycles, receive user feedback, and undergo iterative improvements to optimize performance and reliability.

By implementing these specific strategies and technologies, the security system project can enhance its functionality, reliability, and user experience, providing a more robust and effective solution for securing premises and assets.



## **CHAPTER 8: CONCLUSIONS**

This project introduces an advanced security system, seamlessly combining both audio and video data to provide a unique and high level of security. The primary aim of the system is to provide robust protection and allow users to customize security settings according to their preferences. Additionally, an additional layer of defense is added with a specially designed additional password to ensure uninterrupted access, particularly in challenging scenarios involving audio or video data.

The software component of the project aims to analyze data obtained in Arduino using Natural Language Processing (NLP), Machine Learning (ML), and Python programming language within facial recognition models and techniques. The software element aims to expedite the transfer of data coming from sensors accurately and lay the groundwork for processing this data.

Hardware setup includes the Arduino board, audio drivers, a camera, speakers, and optionally additional sensors. These additional sensors can include ultrasonic sensors, LEDs, a buzzer for audio output, a computer camera, and a microphone. The integration of these flexible components allows for customization to meet the project's objectives and functionality.

The software code integrated into Arduino controls the operation of the alarm system and responds to commands from the Python code. The main loop waits for commands and, upon receiving a command, performs the corresponding action. It processes input from the keypad and performs password verification. Additionally, it processes data from the ultrasonic sensor and reacts to intrusions.

The Python code is responsible for facial recognition, voice recognition, and communication with Arduino. It initializes the recognizer, the text-to-speech engine, and establishes serial communication with Arduino. It sets the voice password via speech recognition and processes voice recognition results for password verification. It also processes face recognition results and checks the recognition status.

Tests and evaluation focused on assessing the functionality and performance of both hardware and software components. Hardware tests aimed to verify the reliability and effectiveness of physical components. However, the LCD display's failure to operate, despite proper wiring and initialization attempts, suggests a potential hardware malfunction or connectivity issue that requires further investigation and troubleshooting. Additionally, occasional inaccuracies in distance measurements observed during ultrasonic sensor testing due to wiring issues highlight the importance of ensuring secure and stable connections for optimal sensor performance.

In conclusion, the testing and evaluation process assessed the functionality and performance of both hardware and software components. Recommendations were provided to enhance the security system's functionality, reliability, and user experience. Suggestions for future developments were presented, and the importance of continuous testing and optimization was emphasized.



## **CHAPTER 9: REFERENCES**

- [1] <https://www.arduino.cc/>
- [2] <https://www.python.org/>
- [3] <https://docs.opencv.org/>
- [4] <https://pypi.org/project/SpeechRecognition/>
- [5] <https://docs.python.org/3/library/tkinter.html>
- [6] <https://www.riverbankcomputing.com/static/Docs/PyQt5/>
- [7] <https://github.com/opencv/opencv/tree/master/data/haarcascades>
- [8] <https://scikit-learn.org/stable/modules/tree.html>
- [9] <https://scikit-learn.org/stable/modules/svm.html>
- [10] [https://en.wikipedia.org/wiki/Universal\\_asynchronous\\_receiver-transmitter](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter)
- [11] <https://projecthub.arduino.cc/CSteele/arduino-alarm-clock-project-4cbd92>
- [12] <https://projecthub.arduino.cc/rajun10/arduino-security-alarm-system-f38f99>
- [13] <https://projecthub.arduino.cc/RoyB/alarm-system-e05c4a>
- [14] <https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/06-the-sound-sensor>
- [15] <https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/02-the-button>
- [16] <https://www.getapp.com/security-software/a/adaptive-shield/>
- [17] <https://guardianaccess.com/industry/commercial/>
- [18] <https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012079/meta>
- [19] <https://ieeexplore.ieee.org/abstract/document/8632703>
- [20] [https://scholar.google.com/scholar?hl=en&as\\_sdt=0%2C5&q=arduino+alarm+system&oq=arduino+alarm+](https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=arduino+alarm+system&oq=arduino+alarm+)
- [21] <https://azojete.com.ng/index.php/azojete/article/view/68>