PROJECT REPORT

Theft Detection System

GROUP MEMBERS:

MUHAMMAD HAMZA KALEEM (FA23-BESE-0003)

MUHAMMAD HASSAN SHAH (FA23-BESE-0005)

MUHAMMAD SAFWAN (FA23-BESE-0030)

HASNAIN YASEEN (FA23-BESE-0040)

USMAN MATLOOB (FA23-BESE-0049)

TAHA SHABIR (FA23-BESE-0029)

COURSE: OBJECT ORIENTED PROGRAMMING.

Contents

Theft Detection System (Safe Lock) Report

Introduction

Security is a critical concern in both residential and commercial settings. Traditional security systems often rely on human intervention and can be prone to error. To address this issue, we have developed a theft detection system (Safe Lock), which applies IoT-based object-oriented programming using a Raspberry Pi.

System Overview

The theft detection system is designed to monitor a designated area and detect any unauthorized entry or suspicious activity. The system uses a combination of sensors, a camera, and intelligent software to detect potential theft, alert the user in real-time, and send pictures via email.

Key Features

- Real-time monitoring
- Motion detection using IR sensors
- Image capture and processing with OpenCV
- Save images with timestamps
- Alerts via the ESP8266 Wi-Fi module
- Send pictures via email using Wi-Fi

Hardware Components

The main hardware components of the system include:

- Raspberry Pi (3B): The central processing unit of the system.
- Camera Module (5 MP): Captures images and videos of the monitored area.
- IR Sensors: Detects infrared radiation emitted by an object or surface within its field of view.
- ESP8266 Wi-Fi Module: Sends images over the internet.
- Buzzer: Sounds an alert when motion is detected.
- Power Supply: Powers the Raspberry Pi and other components.
- SD Card: Stores the operating system and software.

Software Components

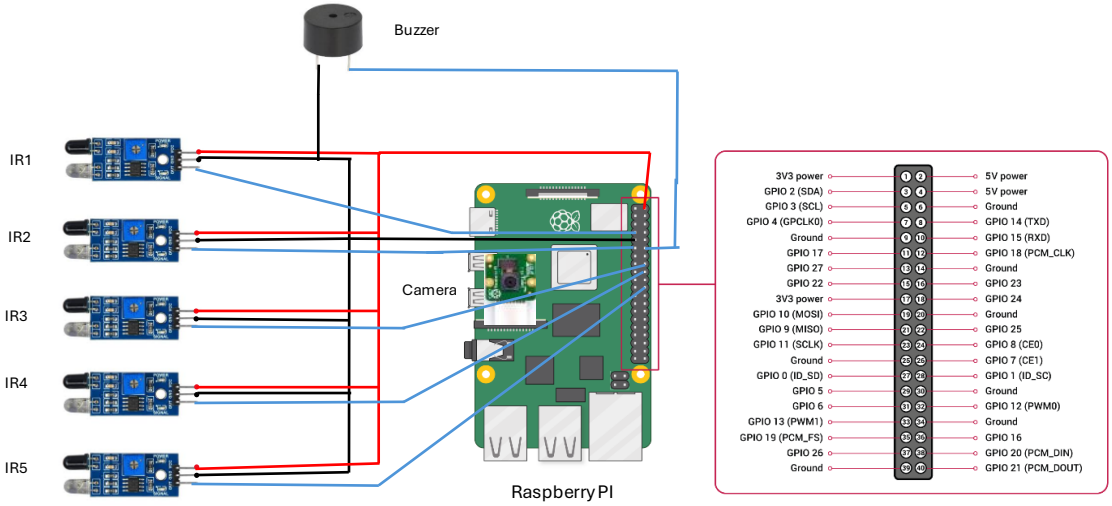
The software components are critical for the functionality of the theft detection system. They include:

- Raspberry Pi OS: The operating system running on the Raspberry Pi.
- IDE: Geany and VS Code IDE for programming.
- Python: The primary programming language used for writing the detection and alert scripts.
- OpenCV: An open-source computer vision library used for image processing.
- SMTPLIB: An open-source library to send pictures via email.
- GPIO Zero: A Python library for controlling the GPIO pins on the Raspberry Pi.
- ESP8266 Library: For communication with the ESP8266 Wi-Fi module.

Data Sheet and Pins Configuration

- Raspberry Pi (3B)
    - GPIO Pins: General Purpose Input/Output pins used for interfacing with sensors and actuators.
- Camera Module (5 MP)
    - Interface: CSI (Camera Serial Interface) port on the Raspberry Pi.
    - Pins:
    - VCC: 5V Power
    - GND: Ground
    - SDA and SCL: I2C communication
- IR Sensors
    - Pins: Connected to GPIO pins (4, 17, 23, 24, 25) for motion detection.
    - VCC: 5V Power
    - GND: Ground
    - OUT: Digital Output
- ESP8266 Wi-Fi Module
    - Pins: Connected to GPIO pins for sending images over the internet.
    - VCC: 3.3V Power
    - GND: Ground
    - TX and RX: UART communication
- Buzzer
    - Pin: Connected to GPIO pin (18) for sounding an alert.
    - VCC: 5V Power
    - GND: Ground
    - SIG: Signal

Circuit Diagram



Operational Methods

- ### Initialization

  Initialize sensors, buzzer, and camera. This setup ensures all components are ready for detection and alert processes.

- ### Detection

  Continuously monitor sensors for motion detection. If any sensor is triggered, the system proceeds to the alert phase.

- ### Alert

  Activate the buzzer and capture an image when motion is detected. The buzzer serves as an immediate deterrent to potential intruders.

- ### Image Processing

  Save the captured image with a timestamp for future reference and evidence.

- ### Notification

  Send the captured image via email using the ESP8266 Wi-Fi module. This ensures the user is notified promptly, regardless of their location.

Limitations

- ### Dependency on Internet

The system relies on a stable internet connection to send email alerts, which might not be reliable in remote areas.

- ### Power Supply
  Continuous power supply is required for the Raspberry Pi and connected components. Power outages can render the system ineffective.
- ### Environmental Factors
  IR sensors can sometimes give false positives due to environmental conditions like heat or movement of small objects, affecting the system's reliability.

Future Enhancements

- ### Cloud Storage Integration
  Integrate cloud storage services (e.g., AWS S3, Google Drive) to store captured images and video feeds securely, allowing for remote access and backup.
- ### Machine Learning for Advanced Detection
  Implement machine learning algorithms to differentiate between humans and animals, reducing false alarms and enhancing accuracy.
- ### Mobile App Integration
  Develop a mobile application to receive real-time alerts, view images or video feeds, and control the system remotely, providing added convenience and security.
- ### Battery Backup
  Add a battery backup system to ensure continuous operation during power outages, enhancing the system's reliability.

Source Code

```
import cv2
from gpiozero import DigitalInputDevice, Buzzer
from datetime import datetime
import time
import os
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage

SENSOR_PINS = [4, 17, 23, 24, 25]
BUZZER_PIN = 18

IMAGE_SAVE_PATH = f'/home/mhamza-rpi/Desktop/oop_project/images'
```

```
class Devices:
    def is_triggered(self):
        pass

class Sensor(Devices):
    def __init__(self, pin):
        self.sensor = DigitalInputDevice(pin)
        print(f"Sensor on pin {pin} initialized")

    def is_triggered(self):
        return not self.sensor.value

class BuzzerDevice(Devices):
    def __init__(self, pin):
        self.buzzer = Buzzer(pin)
        print("Buzzer initialized")

    def activate(self):
        self.buzzer.on()
        print("Buzzer activated")

    def deactivate(self):
        self.buzzer.off()
        print("Buzzer deactivated")

class Camera(Devices):
    def __init__(self):
        self.cap = cv2.VideoCapture(0, cv2.CAP_V4L2)
        if not self.cap.isOpened():
            raise ValueError("Failed to open camera")
        print("Camera initialized")

    def capture_image(self):
        ret, frame = self.cap.read()
        if ret:
            try:
                if not os.path.exists(IMAGE_SAVE_PATH):
                    os.makedirs(IMAGE_SAVE_PATH)
                timestamp = datetime.now().strftime("%d-%m-%y_(%H-%M-
%S)")
                image_path = os.path.join(IMAGE_SAVE_PATH,
f'image_{timestamp}.jpg')
                cv2.imwrite(image_path, frame)
                print(f"Image saved: {image_path}")
                return image_path
            except PermissionError:
                print(f"Permission denied: Unable to save image to
{IMAGE_SAVE_PATH}")
```

```
                return None
        else:
            print("Failed to capture image")
            return None

    def release(self):
        self.cap.release()
        print("Camera released")

def main():
    sensors = [Sensor(pin) for pin in SENSOR_PINS]
    buzzer = BuzzerDevice(BUZZER_PIN)
    camera = Camera()

    try:
        while True:
            if any(sensor.is_triggered() for sensor in sensors):
                buzzer.activate()
                time.sleep(0.3)
                image_path = camera.capture_image()
                if image_path:
                    email = 'fa23bese0003@maju.edu.pk'
                    remail = 'fa23bese0003@maju.edu.pk'
                    app_password = 'oraoxnhjzaiwvsis'

                    subject = f"Theft is detected on system at
{timestamp}"
                    message = f"Theft is detected on system at
{timestamp} \nHere is the attachment:\n"

                    msg = MIMEMultipart()
                    msg['From'] = email
                    msg['To'] = remail
                    msg['Subject'] = subject
                    msg.attach(MIMEText(message, 'plain'))

                    with open(image_path, 'rb') as img_file:
                        img = MIMEImage(img_file.read())
                        img.add_header('Content-Disposition',
'attachment', filename='image.jpg')
                        msg.attach(img)

                    try:
                        server = smtplib.SMTP('smtp.gmail.com', 587)
                        server.starttls()
                        server.login(email, app_password)
                        server.sendmail(email, remail, msg.as_string())
                        print('Email sent successfully!')
```

```
                    except Exception as e:
                        print(f'Failed to send email: {e}')
                    finally:
                        server.quit()

                time.sleep(2)
            else:
                buzzer.deactivate()
            time.sleep(0.5)
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        camera.release()
        for sensor in sensors:
            sensor.sensor.close()
        buzzer.buzzer.close()

if __name__ == "__main__":
    main()
```

Implementation of OOPs Concept in Your Project

Class Definitions:

Devices Class:

Purpose: Serves as a base class for all devices, providing a common interface (is_triggered method) for derived classes.

Sensor Class:

Inherits: Devices
Attributes: Initializes with a specific GPIO pin.
Methods: Implements is_triggered to check if the sensor is triggered.

BuzzerDevice Class:

Inherits: Devices
Attributes: Initializes with a specific GPIO pin.
Methods: activate and deactivate methods to control the buzzer state.

Camera Class:

Inherits: Devices
Attributes: Initializes the camera using OpenCV.

Methods: capture_image to capture and save images, and release to release the camera resource.

Object-Oriented Principles Applied:

Inheritance:

The Sensor, BuzzerDevice, and Camera classes inherit from the Devices base class, demonstrating inheritance. This allows for code reusability and a hierarchical organization of classes.

Polymorphism:

Polymorphism is demonstrated through the is_triggered method in the Devices class. Each derived class (Sensor) can implement this method as per its specific behavior.

Abstraction:

The Devices class provides an abstract interface (is_triggered method) that is implemented by the derived classes. This abstracts the specific implementations of sensor and buzzer functionalities.

Encapsulation:

The properties and methods related to sensors, buzzer, and camera are encapsulated within their respective classes. This hides the internal implementation details and provides a clean interface for interacting with these components.

Results

The system was tested in a controlled environment. When motion was detected by any of the IR sensors, the buzzer activated, and an image was captured and sent via email with timestamps successfully. The system reliably detected motion and captured images, providing clear evidence of any unauthorized entry.

Conclusion

The Raspberry Pi-based theft detection system provides a reliable and cost-effective solution for security monitoring.