Facial Recognition-Based Smart Lock System for Urban and Rural Smart Cities

By:Sharanya ,Aditya,Siddhant

April 26, 2024

Abstract

This report outlines the development and implementation of a facial recognition-based smart lock system designed for urban and rural smart cities. The system utilizes advanced computer vision techniques, including dlib, facial recognition, and OpenCV (cv2), to provide secure and convenient access control.

1 Introduction

The project aims to address the growing need for efficient and secure access control systems in smart cities. With the rapid urbanization and technological advancements, traditional lock systems are becoming obsolete. Hence, integrating facial recognition technology into smart locks offers a sophisticated and reliable solution for enhancing security and accessibility.

2 Internal and External Architecture

2.1 Internal Architecture

In alignment with the principles of a smart city, the internal architecture of the facial recognition-based smart lock system embodies efficiency, reliability, and adaptability. The integration of components such as dlib, facial recognition, cv2, and serial communication reflects the technological sophistication required to address the evolving needs of urban and rural environments within a smart city framework.

- **dlib and Facial Recognition:** These components form the core of the system's intelligence, enabling robust face detection and recognition capabilities.
- **cv2 (OpenCV):** OpenCV serves as the backbone of the system's image processing pipeline, facilitating real-time video analysis and manipulation.
- **Serial Communication:** The inclusion of serial communication functionality enables seamless interaction with the physical lock mechanism, enhancing the system's interoperability and versatility.

2.2 External Architecture

The external architecture of the smart lock system extends beyond the confines of its digital infrastructure to encompass the physical environment of a smart city. By seamlessly integrating with existing urban and rural infrastructure, the system enhances security, accessibility, and efficiency in diverse settings.

- **Physical Lock Mechanism:** At the forefront of the system's external architecture is the physical lock mechanism, which serves as the interface between the digital and tangible realms.
- **Video Capture Device (e.g., Webcam):** The deployment of video capture devices augments the system's situational awareness and surveillance capabilities.

3 Use Cases in Smart Cities

3.1 Urban Areas

In urban smart cities, the smart lock system can be deployed in residential buildings, offices, and public facilities to enhance security and streamline access control.

3.2 Rural Areas

In rural smart cities, the system can be employed in community centers, agricultural facilities, and remote infrastructure sites to secure valuable assets and control access to restricted areas.

4 Hardware Required

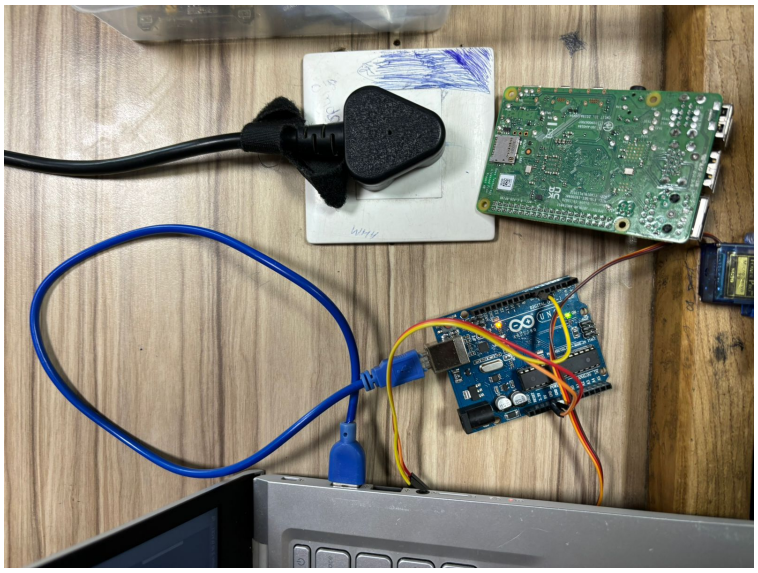

Figure 1:

5 Python Code: Facial Recognition System

```
import face_recognition
import os
import sys
import cv2
import numpy as np
import math
import time
import serial

def face_confidence(face_distance, face_match_threshold=0.6):
    range_val = (1.0 - face_match_threshold)
    linear_val = (1.0 - face_distance) / (range_val * 2.0)

    if face_distance > face_match_threshold:
        return round(linear_val * 100, 2)
    else:
```

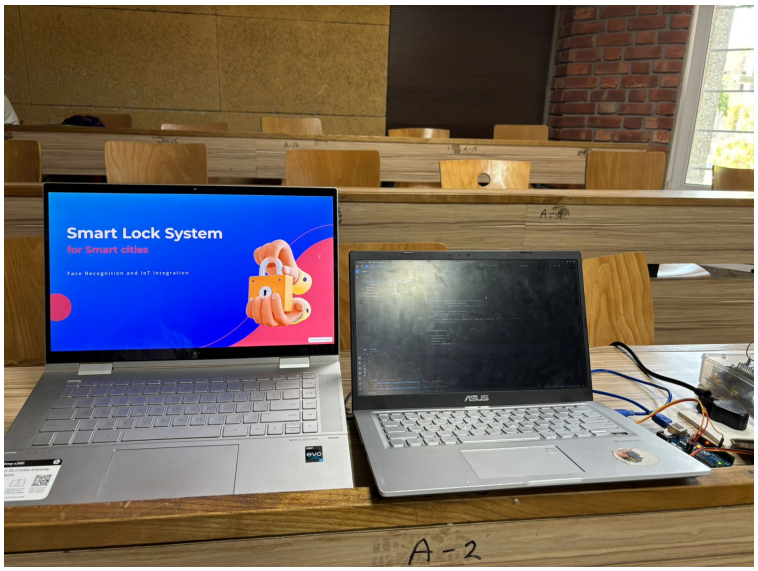

Figure 2: Facial recognition system results

```
gray17          value = (linear_val + ((1.0 - linear_val) * math.pow((linear_val - 0.5) *
                    2, 0.2))) * 100
gray18          return round(value, 2)
gray19
gray20  class FaceRecognition:
gray21
gray22      face_locations = []
gray23      face_encodings = []
gray24      face_names = []
gray25      known_face_encodings = []
gray26      known_face_names = []
gray27      process_current_frame = Trueimport face_recognition
gray28  import os
gray29  import sys
gray30  import cv2
gray31  import numpy as np
gray32  import math
gray33  import time
gray34  import serial
gray35
gray36  def face_confidence(face_distance, face_match_threshold=0.6):
gray37      range_val = (1.0 - face_match_threshold)
gray38      linear_val = (1.0 - face_distance) / (range_val * 2.0)
gray39
gray40      if face_distance > face_match_threshold:
gray41          return round(linear_val * 100, 2)
gray42      else:
gray43          value = (linear_val + ((1.0 - linear_val) * math.pow((linear_val - 0.5) *
                    2, 0.2))) * 100
gray44          return round(value, 2)
gray45
gray46  class FaceRecognition:
gray47
gray48      face_locations = []
gray49      face_encodings = []
gray50      face_names = []
```

```
    known_face_encodings = []
    known_face_names = []
    process_current_frame = True

    def __init__(self):
        self.encode_faces()

    def encode_faces(self):
        for image in os.listdir('FACES'):
            face_image = face_recognition.load_image_file(f'FACES/{image}')
            face_encoding = face_recognition.face_encodings(face_image)[0]

            self.known_face_encodings.append(face_encoding)
            self.known_face_names.append(image)

        print(self.known_face_names)

    def run_recognition(self):
        video_capture = cv2.VideoCapture(0)

        if not video_capture.isOpened():
            sys.exit('Video source not found...')

        time.sleep(2)

        face_detected_time = None
        unknown_face_detected_time = None

        #Open serial port
        ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
        time.sleep(2)  # Allow some time for Arduino to initialize

        while True:
            ret, frame = video_capture.read()

            if self.process_current_frame:
                small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
                rgb_small_frame = small_frame[:, :, ::-1]

                self.face_locations = face_recognition.face_locations(
                    rgb_small_frame)
                self.face_encodings = face_recognition.face_encodings(
                    rgb_small_frame, self.face_locations)

                self.face_names = []
                for face_encoding in self.face_encodings:

                    matches = face_recognition.compare_faces(self.
                        known_face_encodings, face_encoding)
                    name = 'Unknown'
                    confidence = 'Unknown'

                    face_distances = face_recognition.face_distance(self.
                        known_face_encodings, face_encoding)
                    best_match_index = np.argmin(face_distances)

                    if matches[best_match_index]:
                        confidence = face_confidence(face_distances[
                            best_match_index])
```

```
                        name = self.known_face_names[best_match_index]
                        if confidence > 80:
                            if face_detected_time is None:
                                face_detected_time = time.time()
                                ser.write(b'1')  # Send '1' to Arduino
                            elif time.time() - face_detected_time >= 5:  # Wait at
                                 least 5 seconds
                                print(1)
                                ser.write(b'0')  # Send '0' to reset Arduino
                                face_detected_time = None  # Reset the timer
                        else:
                            print(0)
                    else:  # Unknown face detected
                        if unknown_face_detected_time is None:
                            unknown_face_detected_time = time.time()
                        elif time.time() - unknown_face_detected_time >= 10:  #
                            Wait at least 10 seconds
                            print(0)
                            unknown_face_detected_time = None  # Reset the timer

                    self.face_names.append(f'{name} ({confidence}%)')

            self.process_current_frame = not self.process_current_frame

            for (top, right, bottom, left), name in zip(self.face_locations, self.
                face_names):
                top *= 4
                right *= 4
                bottom *= 4
                left *= 4

                cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
                cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0,
                    255), -1)
                cv2.putText(frame, name, (left + 6, bottom - 6), cv2.
                    FONT_HERSHEY_DUPLEX, 0.8, (255, 255, 255), 1)

            cv2.imshow('Face Recognition', frame)

            if cv2.waitKey(1) == ord('q'):
                break

        # Close serial port
        ser.close()
        video_capture.release()
        cv2.destroyAllWindows()


if __name__ == '__main__':
    fr = FaceRecognition()
    fr.run_recognition()
```

Listing 1: Facial Recognition System Code

6 Conclusion

The facial recognition-based smart lock system represents a significant advancement in access control technology, particularly in the context of smart cities. By leveraging cutting-edge computer vision techniques

and seamless integration with existing infrastructure, the system offers a secure, efficient, and user-friendly solution for enhancing security and accessibility in urban and rural environments.

7 Results

Figure 3: Facial recognition system results