M30226 UP2060907 - Smart Student Attendance System

Jason Nghiem

Computer Science, University of Portsmouth
Portsmouth, United Kingdom
[1]up2060907@myport.ac.uk

***Abstract*— This report showcases the design and implementation of a smart student attendance system with the use of RFID readers and facial recognition software. An ESP32 module and a RC522 module is used to take a unique identifier and communicate with a face recognition script using a MQTT broker for bidirectional communication. This solution solves the problem of students' lack of appearance in lectures despite recorded attendance with a cheap and scalable solution for use in multiple locations. Limitations include the need of multiple images of every student to ensure the accuracy of verification from the face recognition system and the lack of development message assurance systems that ensure a message is received by clients.**

***Keywords*— Face recognition, RFID, IoT, ESP32**

I. Introduction

This report aims to document the design and discussion of the implementation of a Smart Student Attendance system for a more efficient and accurate method of monitoring student attendance. Currently at the University of Portsmouth, students are required to scan their given student IDs to confirm attendance. While this method is seemingly accurate for monitoring student attendance, there are some limitations with this solution: there is no current way to confirm that the individual who scanned the ID is the correct person, students may scan their ID and leave the venue without attending the lecture. As a result of the ease of bypassing this current solution, inaccurate data is gathered about lecture attendance and students who already have poor attendance are able to rectify this in deceitful ways. My proposed solution is to incorporate an Internet of Things (IoT) system using RFID and facial recognition to track the students' attendance and confirming their physical attendance. This solution can be broken down into two distinct parts: student scanning their ID to retrieve credentials for facial recognition, facial recognition of the student and determining if this is the correct student for scanned ID. With this solution, universities are able to ensure that students attend lectures and more importantly monitor students who have poor attendance. A more sophisticated system, such as the one proposed, would also act as a deterrent against truancy due to its multiple tracking methods. In addition to this, more accurate and useful data concerning student attendance will be provided to the university allowing for more in depth analysis of potential problem areas. For example, if there is a particular lecture that has shown to have poor attendance then this system allows for the necessary data to prove that there is an underlying issue present which requires further investigation.

II. Design

This section will contain a comprehensive breakdown of the system design. Hardware and software components will be described and its use justified based on cost and suitability for this project. Also the explanation of how each module contributes to solving the identified problems. Also pseudo code and logic flow will be presented to demonstrate the solution clearly.

A. High level system overview

Figure 1 illustrates a high level overview of the system. It demonstrates a process map of what inputs are taken and abstracted processes that take place.

Using the RC522 module to detect an RFID chip ID, the students credentials are fetched from a database (this will not be implemented due to scope and time constraints). Use of an RFID scanner simplifies the process of identification of the student and enables efficient searching of a student database.

The student will be required to scan their face with a camera. A decision process occurs ensuring

returning a boolean value. If the process fails then a visual indicator is given to the student and resets the system for another scan. If the process succeeds, a visual indicator occurs. Facial recognition is a core component of this system to ensure that the scanned ID and student match ensuring the student is present.

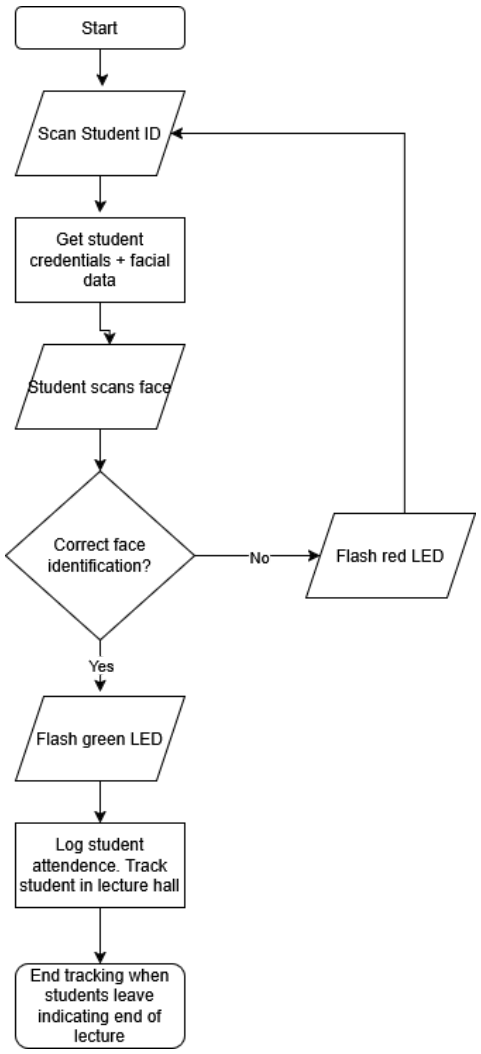

Fig. 1 System flowchart

B. Hardware

A. ESP32 WROOM 32

This module will be used to take RFID inputs from the RC522 and use MQTT to publish the unique identifiers to a subscriber who contains the Python script for facial recognition. ESP32 is a suitable module for use in this project due to its built-in WiFi capabilities allowing for the connection to the network and transmission of data.

This module is low cost with unit pricing between £4-£7 making it ideal for use in large sites such as a University campus where multiple rooms will require these systems.

In addition, this module is highly compact (18 x 25.5x 3.10 mm)allowing for ease of installation. Little to no alterations to existing infrastructure is needed to install these small devices.

ESP32 also is computationally more powerful than an Arduino Uno with more onboard memory and additional GPIO pins available for use. This allows for additional scalability when using these modules for more sensors or processing of more complex algorithms. Unlike the ESP32, the Arduino Uno does not have WiFi connectivity therefore an additional module is needed for network connections. Also the Uno is marginally more expensive than the ESP32 whilst delivering less performance and capabilities (Agarwal, 2024).

ESP32 is capable of running facial recognition onboard therefore the need of additional servers is not necessary. However, a camera module is required. With this project's budget constraints, we have opted to instead run facial recognition scripts on a separate personal computer with the use of a USB camera. Running facial recognition on the ESP32 can improve ease of integration due to the lack of need to implement infrastructure to handle facial recognition off site but will increase costs by having to purchase additional camera modules that are compatible with the ESP32.

B. RC522

This module is used to scan student cards and get the necessary credentials for facial recognition. Modules can be found for under £10 making it extremely affordable and suitable for large scale deployment without significant budget increases.

It is also compact, 39mm x 59.5mm (Handson Technology Data Specs RC522 RFID Development Kit, n.d.), for ease of installation into any space.

Supports the ISO/IEC 14443 A standard which is compatible with common RFID cards/tags making it ideal for use in a University setting.

Compatibility and extensive documentation of this module in addition to its form factor and widely supported standard makes it the ideal candidate for this project.

C. *Python facial recognition libraries*

OpenCV is an open source and free Python library for real time computer vision applications. In conjunction with face_recognition, which is a Dlib wrapper package that simplifies the use of dlib's facial recognition functionality, facial recognition can be achieved.

These libraries are open source therefore costs remain low for this implementation. Also documentation for OpenCV, Dlib + face_recognition is widely available for ease of implementation in case of errors and it makes maintenance easier.

Although this cannot be used solely on an ESP32 due its lack of an onboard camera, we are able to implement this with a standard USB camera that is connected to a network that will send responses to the ESP32 whether a student has passed or failed facial recognition. This is a limitation of the project as additional infrastructure is needed to host various cameras in multiple locations which send real time camera data to a network. Performance is lost but assurance of registration is possible.

For this system to operate, an extensive library of student pictures is needed for reference. For this project, a set of pictures will be used to identify the user.

D. *MQTT*

MQTT is used to establish bidirectional communication between the facial recognition component with the ESP32. It allows for real-time communication between the ESP32 and other devices within the network.

It is highly efficient with low overheads allowing for responsive delivery unlike alternatives. There is built-in support for persistent connections that ensure reliable transmission of data.

Open source brokers, such as mosquitto, exist for ease of implementation on both microcontrollers and servers with extensive documentation. This ensures that solutions can be implemented quickly and efficiently and debugging easier.

MQTT uses a Publish-Subscribe communication model for decoupled communication between publishers and subscribers. This improves the scalability of the system with simple additions of either publishers and subscribers to the system unlike other communication models which may require additional infrastructure and processing.

This low cost but high performance in real-time applications is suitable to handle this project's communication with the ability to scale based on the number of lecture halls allowing for growth.

III. Implementation

This section aims to document and explain the implementation of this system.

A. *ESP32 + RC522 + LED*

A. *Wiring*

The hardware connections between the ESP32 and RC522 are shown in table 1.

TABLE 1
ESP32 and RC522 Pin connections

ESP32 and RC522 connections	
RC522	**ESP32**
SDA	GPIO5
SCK	GPIO18
MOSI	GPIO33
MISO	GPIO19
GND	GND
RST	GPIO21
3.3v	3.3v ONLY

LED wiring is connected to ground and any general purpose GPIO pin. For this project,

GPIO26 and 27 were used. See fig. 2 for the whole system.

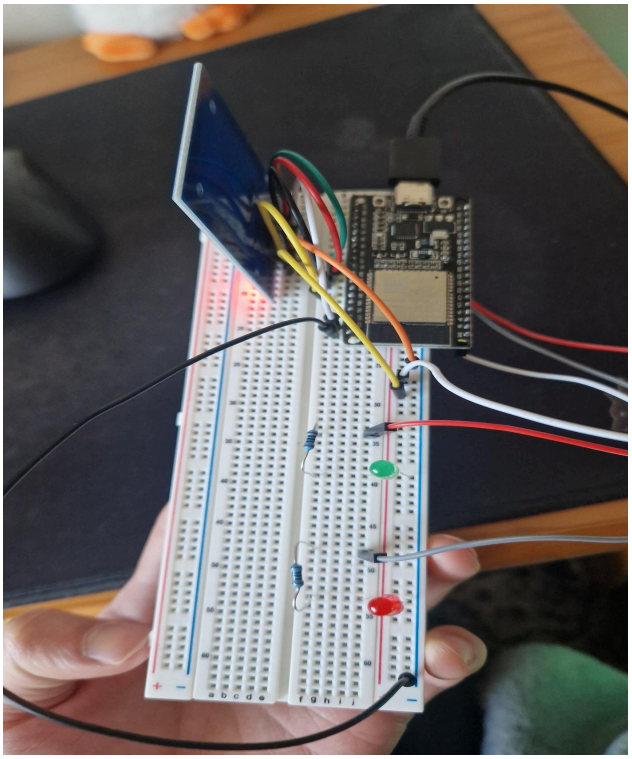

Fig. 2 ESP32 + RC522 + LED wiring

The use of 100Ω resistors are used for the LEDs are used to prevent burning out the LEDs in extended use.

It is important to note that the RC522 has a working voltage of 3.3v and use of higher voltage can damage the module.

B. Code

Code was implemented in the Arduino IDE with the following packages for use of MQTT and RC522 RFID reader

```
#include <MFRC522v2.h>
#include <MFRC522DriverSPI.h>
#include
<MFRC522DriverPinSimple.h>
#include <WiFi.h>
#include <PubSubClient.h>
```

Within this sketch, core functionality is split into three separate functions: connectWiFi(), connectMQTT() and response().

connectWiFi is used to connect the ESP32 to an access point for network access. The use of a mobile hotspot is used to ensure consistent connectivity. It first checks whether the WiFi status and if it's not connected then it will attempt connection with the given network name and password. The number of connections is counted and if it exceeds a threshold then the ESP32 is restarted to ensure that the module is working correctly. When connected, it will then attempt to connect to the MQTT broker.

connectMQTT() is a function that attempts to connect to the MQTT broker and once connected will subscribe to a response topic that will receive messages from the server whether verification has succeeded or not. If it cannot connect to an MQTT broker, status codes for failures are printed onto the serial monitor for debugging purposes.

```
void connectMQTT() {
  client.setServer(mqtt_server,
mqtt_port);

  while (!client.connected()) {
    Serial.print("Connecting to
MQTT ");
    if
(client.connect("ESP32Client")) {
    Serial.println("connected.");

client.subscribe(mqtt_topic_sub);
    } else {
    Serial.print("failed. State=");
    Serial.print(client.state());
    delay(2000);
    }
  }
}
```

response() is a function that contains message handling logic for the response topic. In this function, it will receive a payload and read the contents. If the message is “success” then verification has succeeded and the green LED turns on to inform the user that verification has

passed. If failed, the red LED will show the failed verification.

```
if (message == "success") {
    Serial.println("Face
verification succeeded");
    for (int i = 0; i < 10; i++) {
    digitalWrite (greenLedPin,
HIGH);
    delay(500);
    digitalWrite (greenLedPin,
LOW);
    }
  } else if (message == "fail") {
    Serial.println("Face
verification failed");
    for (int i = 0; i < 10; i++) {
    digitalWrite (redLedPin, HIGH);
    delay(500);
    digitalWrite (redLedPin, LOW);
    }
  }
```

In setup(), the necessary pins are initialised and connectWiFi and connectMQTT are called with the client callback response set to response() to handle receiving messages.

Within the main loop, it once again checks if the client is connected to MQTT and if not it will reattempt connection in case the module loses connection. client.loop() is used to prevent network timeouts by sending empty requests to the broker to ensure a persistent connection. Logic for handling card reading is found here where it checks whether any card is scanned and returns if not. When a card is scanned, it will read the card's UID and publish this to a request topic for the server to take and perform facial recognition on.

B. Facial Recognition Scripts

Two scripts are used within this system, extract_embeddings.py and recognise_face_cam.py. Before the use of these scripts, a labeled dataset of faces must be created for the model to learn from. For each person, they must have a folder named with their name and within this folder various images of their face must be included. For optimal effectiveness, multiple images are used with varied environments. For example, changes in lighting in different images help the model pick on details with lower light levels. Also, images with the subject making different faces such as smiling were included to improve accuracy when students scan their face whilst talking or making expressions.

Fig. 3 Pictures used for tuning model

A. extract_embeddings.py

This script takes a given labelled dataset of people and uses 'face_recognition' library to create an encoding file containing each person's facial encoding used to recognise their face. Face is located with this library with `face_recognition.face_locations(rgb,model='hog')` and their labelled encodings are stored in a dictionary. Face encodings are numerical values that distinguish facial features from the dataset.

```
# Face encoding
    encodings =
face_recognition.face_encodings(rg
b, boxes)
    for encoding in encodings:

knownEncodings.append(encoding)
          knownNames.append(name)

data = {"encodings":
knownEncodings, "names":
knownNames}
```

This dictionary is dumped into a given file for use in recognise_face_cam.py for facial

recognition. This removes the need for real time encoding of a given face and allows for the reuse of this file which improves performance. As this system may be used in multiple locations at the same time, performance is crucial to prevent wait times and frustrations from staff and students that are attempting to attend their lecture.

B. recognise_face_cam.py

This script connects to the same MQTT broker as the ESP32 module to receive a UID and publishes the result of verification for the ESP32 to indicate to the user the result of verification. This is also where facial recognition logic is held and called.

Facial recognition logic uses OpenCV and face_recognition to carry out the verification. For this prototype, UID's and their corresponding student are stored in a dictionary. In practice, this can be stored in an existing relational database. The face encodings file that was generated with the previous script is loaded into this script and stored in addition to frontal face detection libraries (haarcascade_frontalface_alt2). A timer is started which halts after a certain amount of time. As the target face is not found within this time frame, we can deduce that either a face cannot be detected or that the given face does not match the target therefore verification fails. Using the encoding file, a check is carried out with face_recognition.compare_faces. Within the frame, a count is made to tally the occurrences of a named match that occurred and from this it takes the named match with the highest count as the final match result. A final check is carried out between the match name and the target name and returns true if it matches.

Once a decision has been made, a message is published to a topic. message() Function handles the logic of receiving a UID and publishing the result.

```
def message(client, userdata,
msg):
    uid = msg.payload.decode()
    print(f"UID: {uid}")
    result = face(uid)
    if result:
        print("Match found")
        msg = "success"
        client.publish(topic_pub,
msg)
        return
    else:
        print("No match found")
        msg = "fail"
        client.publish(topic_pub,
msg)
        return
```

C. MQTT

The use of an open-source message broker such as Mosquitto is used to implement the use of MQTT within this system. Installation is done on a given server. This project is developed on Windows therefore the Windows installation wizard is used. A basic config file is created to store the parameters of the broker to allow anonymous users and to use a specific port for ease of development and testing. Using the verbose terminal output, connections to the broker from the ESP32 and Python script are verified to be working.

Two topics are used with this broker. Firstly, attendance/request topic is used to send scanned UIDs to the Python script. Finally, attendance/response is used to send the result/response to the ESP32 used to determine whether the green or red LED will be lit.

D. Summary - detailed system overview

In summary, when the ESP32 boots, it will attempt to connect to a specific wireless network and a MQTT broker and will attempt this until successful connection. The Python script recognise_face_cam.py will also do the same. Once connected, both systems will continuously check for messages which ensures that connection persists and that incoming messages are received.

Once the student scans their card, a UID is taken from the RC522 scanner and a message of this UID is published to attendance/request. The Python script receives this message and then uses this as a key to get the target's name. Then facial recognition begins with the loading of a face encoding file that is compared with the face present in the camera. If a match is not found within 10 seconds then the system halts and a failed verification message is published to attendance/response. On successful verification, the ESP32 module receives the message and then lights up the LED.

TABLE 2
ALL FUNCTIONS AND DESCRIPTION

Function descriptions	
Function	**Description**
face(uid)	Python function that performs facial verification. Input: UID Output: Boolean
connect(client, userdata, flags, rc)	Python function that connects to MQTT and subscribes to the attendance/request Input: client, userdata, flags, rc Output: None
message(client, userdata, msg)	Python function for message handling. Input: client, userdata, msg Output: None
publish(client, userdata, mid)	Python function that handles publishing logic Input: client, userdata, mid Output: Confirmation message
connectWifi()	C++ ESP32 function that attempts connection to WiFi network Inputs: SSID, Password Output: None
connectMQTT()	C++ ESP32 function that attempts connection to MQTT broker Input: Broker IP, Broker Port Output: None
response(char* topic, byte* payload, unsigned int length)	C++ ESP32 function for message receival. Input: Topic, Payload Output: None
setup()	C++ ESP32 function carrying ESP32 setup code. connectWifi(), connectMQTT() is called here. Client callback is set to response() here. RC522 module is initialised here. LED pins are initialised here. Input: None Output: None
loop()	C++ ESP32 function carrying out looped code. connectMQTT() is called here again if not connected. Check if an RFID card is present. Get a UID or RFID card when scanned. Published UID to topic attendance/request. Input: None Output: None

IV. DISCUSSION

The implemented system successfully demonstrates a new smart student attendance system. Using the integration of RFID identification, facial recognition with MQTT based communication results in a solution that is both scalable and performant. Real time verification of student attendance and physical presence solves the problem where the student may scan their ID and not attend the booked session.

The key strength of this solution is the scalability of the solution. With the decoupled real time communication with MQTT, it allows additional clients to be connected to the broker. This simplifies the deployment process of installing the ESP32 and RC522 modules on site with connections automated for efficiency. It is possible to also alter individual units separately without affecting other modules in the network making maintenance easier.

For this project, facial recognition logic is handled on a separate server to the ESP32 units. This works for quick development of the system in a controlled environment, it will require infrastructure to be built to accommodate this. For example, there must be an additional system that allows new students to be entered into the system which will need to include various pictures of the student in a variety of different lightings and angles. This may be difficult for those students who may not have the ability to complete this step due to the lack of a camera device. Additional services that will complete this step for students will need to be provided.

Another limitation, this project does not implement systems that handle dropped messages

in cases of network dropouts. Therefore, additional systems will need to be developed that ensures that the message is published and received by subscribers should be done to combat network unreliability.

Future enhancements include the introduction of real time lecture tracking. This ensures that the student is found within the lecture hall and attends for the duration of the entire lecture. Also, scripts and services that add new student data for facial recognition to streamline onboarding processes.

References

Agarwal, T. (2024, January 16). ESP32 vs Arduino: Definition & the Main Differences. ElProCus - Electronic Projects for Engineering Students. https://www.elprocus.com/difference-between-esp32-vs-arduino/

Handson Technology Data Specs RC522 RFID Development Kit. (n.d.). https://www.handsontec.com/dataspecs/RC522.pdf

ISO 14443 contactless card standard—Thales. (n.d.). Retrieved 18 April 2025, from https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/iso14443