Biometric Authentication System Documentation

Authors

January 16, 2024

1 Introduction

The Biometric Authentication System is a prove of concept of an IoT solution designed to provide security authentication through biometric verification, including galvanic skin response, heart rate analysis, and facial recognition. This project's primary objective is to establish a user-friendly system for authenticating individuals in secure environments as well as experimenting and learning with IoT devices. The following documentation comprises detailed descriptions of system components,project milestones, a user story depicting the system's functionality, and technical insights into the component integration and operation.

2 General Project Milestones

The development of the Biometric Authentication System was achieved through the accomplishment of several key milestones. These milestones collectively ensured the project's progress was systematically tracked and adhered to the proposed timeline. Note that component specific milestones with more detailed information, priorities and dependencies can be found on component-implementations

1. **Project Initiation:** Defining scope, objectives, and resource allocation.
2. **Component Sourcing and Assembly:** Hardware setup of the development environment.
3. **System Design:** Architecting overall system interactions and data flow.
4. **Component Integration:** Ensuring seamless operation between different system modules.
5. **Software Development:** Writing and testing code for individual components and overall system coordination.
6. **Project Wrap-Up:** Final evaluation, documentation, and project delivery.

3 Component Implementations

This section outlines the individual components that comprise our system, detailing their objectives, functionalities, and integration with the broader system context.

3.1 RaspberryPi

3.1.1 Objective and Functionality

The RaspberryPi is the backbone of the project, it serves as the MQTT broker and merges the collected data from both the Biometric Sensors and the camera it has attached to provide a prediction of who may the user interacting with system be, sending that final prediction to the Monitor System. Its responsibilites include:

- Subscribing to and processing biometric data from the ESP-01 module attached to biometric sensors via MQTT.
- Processing data from the webcam connected to the RaspberryPi via usb and running it through a neural network to get a prediction of who is the user in the picture.
- Run the biometric data obtained through MQTT and the result of the neural network through a decision tree to get a final prediction of who is the user interacting with the system.
- Sending the final result to the 'rpi/prediction' topic so the System Monitor can show the result.

3.1.2 Project Definition and Milestones

The development of the RaspberryPi system involved the following milestones:

1. Setting up an MQTT broker to handle all the IoT devices communications.
2. Establishing MQTT communication to receive biometric data and send predictions.
3. Training a neural network to work with the webcam input and predict which user is in front of it.
4. Training a classification tree to get a final prediction using both biometric data and neural network prediction.

3.1.3 Achieved milestones, execution order, priority, and dependencies

1. **Milestone 1: Setting up an MQTT broker** - *Priority:* High. Fundamental for all the data transmission. - *Dependencies:* Basic WiFi setup. - *Execution Order:* First, as it is crucial for data reception.

2. **Milestone 2: Establishing MQTT communication to receive and send data** - *Priority:* Medium. Important for working with real data. - *Dependencies:* Successful MQTT setup. - *Execution Order:* Second, building upon established communication.

3. **Milestone 3: Training neural network** - *Priority:* High. Essential for effective prediction. - *Dependencies:* Functional hardware setup (webcam). - *Execution Order:* Third, getting a first prediction.

4. **Milestone 4: Training a classification tree** - *Priority:* Medium. Important for a more accurate prediction. - *Dependencies:* Functioning MQTT communication and neural network. - *Execution Order:* Fourth, finalizing the machine learning prediction.

5. **Milestone 5: Sending the results** - *Priority:* High. Essential for system effectiveness. - *Dependencies:* Functioning MQTT communication and classification tree. - *Execution Order:* Fourth, last step towards getting a prediction shown to the user.

3.1.4 Hardware setup

The hardware setup for the RaspberryPi comprises:

- A RaspberryPi 4 with WiFi capabilites.
- A logitech webcam with usb connection.

3.1.5 Software Implementation

The software, written in python has the key functions:

- Recieving biometric data from 'sensor3/galvanic' and 'sensor3/heart' topics.
- Triggering a neural network prediction through the webcam obtained footage.
- Run the result from the neural network and the biometric data through a classification tree.
- Publish the final prediction to the 'rpi/prediction' topic.

The operating system (RaspberriPi OS Lite) is responsible of connecting to WiFi and running MQTT broker. So basically it has to:

- Run a service for MQTT broker that launches on startup.
- Establish WiFi connection.
- Run a service that initializes the prediction pipeline when a user is detected.

NOTE: The data collection for training the classification tree has also been done on the RaspberryPi as well as the training itself, code for replicating this steps can be found on the project repository.

3.1.6 Testing

Testing has been mainly focused on the machine learning aspect, since getting accurate results was non trivial. By generating artificial data (data augmentation) it was possible to finally achieve a more convincing accuracy. Regarding MQTT, setting up the broker was a very straight forward process with no trouble.

3.1.7 Dedication Time

Approximately 30 hours were dedicated to developing the RaspberryPi system, using most of the time on the machine learning aspect.

3.1.8 Challenges and Solutions

- **Hardware Challenges:** Collecting footage from the usb webcam.
- **Software Challenges:** Getting an accurate prediction from the model. Using a foundational model trained for object detection wasn't working very well for telling faces apart so the solution provided is to use a representative item for each member of the group which is what is placed in front of the camera, this way the neural network can tell us apart easily.

3.1.9 Hardware and Software Integration

The only hardware used on this setup was a webcam connected through usb to the RaspberryPi.

3.1.10 MQTT Topics

- **sensor3/heart:** This is the topic used to collect heart rate biometric data.
- **sensor3/galvanic:** Provides galvanic resistance data, and together with the first topic, this two provide the full biometric data needed.
- **rpi/prediction:** Topic used to publish the result of the prediction.

3.1.11 Conclusion

The development on the RaspberryPi system was a very interesting experience, it was the first time working with machine learning for some of us and it was very challenging to get an accurate prediction, but after some research and testing it was possible to get a good result. The MQTT broker setup was very straight forward and it was easy to get it working.

3.2 System Monitor

3.2.1 Objective and Functionality

The System Monitor is a pivotal component in a biometric data-based framework for user access and verification. It serves two main functions: position detection using user location data and user identification. The System Monitor's functionalities include:

- Subscribing to and processing location data from two ESP-01 modules with ultrasonic sensors via MQTT.
- Displaying user location and verification data on an LCD screen.
- Visually representing user proximity using an LED bar.

3.2.2 Project Definition and Milestones

The development of the System Monitor involved the following milestones:

1. Establishing MQTT communication to receive data from ultrasonic sensors.
2. Integrating and configuring the LED bar and LCD with the ESP32.
3. Developing and testing the software for proximity detection and data display.
4. Efficient multitasking and task synchronization within FreeRTOS.

3.2.3 Achieved milestones, execution order, priority, and dependencies

1. **Milestone 1: Establishing MQTT Communication** - *Priority:* High. Fundamental for data transmission. - *Dependencies:* Basic WiFi setup. - *Execution Order:* First, as it is crucial for data reception.
2. **Milestone 2: LED Bar and LCD Integration** - *Priority:* Medium. Important for user interface. - *Dependencies:* Successful MQTT setup. - *Execution Order:* Second, building upon established communication.
3. **Milestone 3: Software Development** - *Priority:* High. Essential for functionality. - *Dependencies:* Functional hardware setup. - *Execution Order:* Third, focusing on processing and display.
4. **Milestone 4: Multitasking and Task Synchronization** - *Priority:* High. Critical for system reliability. - *Dependencies:* Completion of initial software development. - *Execution Order:* Fourth, finalizing the system integration.

3.2.4 Hardware Development

The hardware setup for the System Monitor comprises:

- An ESP32 microcontroller connected to a breadboard.
- An LED bar and an LCD screen interfaced with the ESP32 for display.
- Two ESP-01 modules, each with an ultrasonic sensor, for distance measurement.

3.2.5 Software Implementation

The software, written in C++, leverages FreeRTOS for effective multitasking. Key functions include:

- `WiFiTask` for managing WiFi connectivity.
- `MQTTTask` for handling MQTT communication.
- `LCDDisplayTask` and `checkForPresenceAndDirection` for controlling the display based on sensor data.
- Semaphore (`xSemaphoreLCD`) to ensure safe LCD access.

3.2.6 Interaction with Other Components

The System Monitor interacts with the ESP-01 modules to determine the user's location based on distance data, which is then displayed on the LED bar and LCD screen.

3.2.7 Testing

Testing ensured accurate sensor data transmission via MQTT and responsive displays on the LED bar and LCD. An iterative approach was applied to refine the system's performance.

3.2.8 Dedication Time

Approximately 20 hours were dedicated to developing the System Monitor, focusing on both hardware assembly and software programming.

3.2.9 User Story

As a user approaches, the System Monitor detects their presence, lighting up the LED bar and displaying their approximate location (left, right, or center) on the LCD screen, providing real-time feedback.

3.2.10 Challenges and Solutions

- **Hardware Challenges:** Complex wiring and space constraints on the breadboard, along with troubleshooting LED issues.
- **Software Challenges:** Establishing reliable communication and processing sensor data accurately. Achieved through careful coding and testing.

3.2.11 Hardware and Software Integration

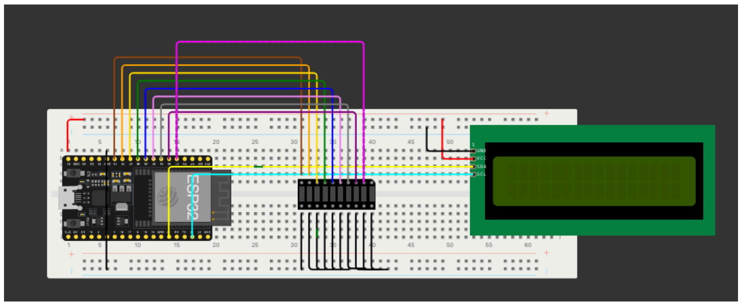

Figure 1: ESP32 with LED bar and LCD display connected on a breadboard.

3.2.12 Microcontroller Interaction Protocol

The System Monitor project involves coordinated communication between multiple microcontrollers, primarily the ESP32 and two ESP-01 modules. The communication is structured as follows:

Communication Protocol The project utilizes the MQTT (Message Queuing Telemetry Transport) protocol, a lightweight and efficient messaging protocol ideal for IoT applications. This protocol is chosen for its low bandwidth usage and its ability to provide reliable communication over WiFi.

WiFi Configuration Each microcontroller in the system is configured to connect to a WiFi network, enabling them to send and receive data over the network. The network credentials (SSID and password) are programmed into the microcontrollers.

Data Flow and Configuration The ESP-01 modules, each equipped with an ultrasonic sensor, collect distance data and publish this information to specific MQTT topics. The ESP32, acting as the central unit, subscribes to these topics to receive the data.

3.2.13 MQTT Tree Structure

The MQTT protocol in the System Monitor project uses a structured approach to manage the data flow. Below is the outline of the MQTT topics and their functions:

MQTT Topics

- **sensor1/distance:** This topic is used by the first ESP-01 module. It publishes the distance data measured by its connected ultrasonic sensor. The ESP32 subscribes to this topic to receive updates on the user's distance from this sensor.
- **sensor2/distance:** Similar to the first, this topic is for the second ESP-01 module. It provides distance data from its sensor, allowing the ESP32 to determine the user's location relative to this second sensor.

Data Organization and Utilization The data sent to these topics include numerical values representing the measured distances. The ESP32, upon receiving this data, processes it to determine the user's proximity and location. Based on these calculations, the ESP32 then controls the LED bar and the LCD display to provide real-time feedback about the user's position.

3.3 Integration of Biometric Sensors with Arduino NANO-BLE33 and ESP-01

3.3.1 Objective and Functionality

The user data collector is the part of the system responsible of collecting the users heart rate and galvanic skin resistance in order to later predict who is that user.

- Reading BPM and galvanic resistance data from the sensors on the Arduino NANO
- Getting the collected data from the Arduino to the ESP01 module through serial.
- Publishing the collected data to the corresponding MQTT topics (sensor3/heart and sensor3/galvanic)

3.3.2 Project Definition and Milestones

The development of the user data collector involved the following milestones:

1. Read data from BPM.
2. Read data from galvanic.
3. Get data from Arduino to ESP01.

4. Establishing connection with MQTT.
5. Publish the results on their corresponding topics.

3.3.3 Achieved milestones, execution order, priority, and dependencies

1. **Milestone 1: Read data from BPM and galvanic** - *Priority:* High. Fundamental for data collection. - *Dependencies:* Working hardware setup. - *Execution Order:* First, as it is the backbone of this system.
2. **Milestone 2: Establish connection with MQTT and publish data** - *Priority:* Medium. Important, the rest of the system needs this data. - *Dependencies:* Working WiFi setup. - *Execution Order:* Second, needed for debugging the data retrieval from Arduino to ESP01.
3. **Milestone 3: Getting data from Arduino to ESP01** - *Priority:* High. Essential for usage of real data. - *Dependencies:* Functional hardware setup and MQTT communication. - *Execution Order:* Third, focusing on data transmission.

3.3.4 Hardware setup

The biometric data collection system comprises the following hardware components:

- **Arduino NANO-BLE33:** Central to the collection and initial processing of biometric data, including heart rate and galvanic skin response (GSR).
- **ESP-01:** Functions as a WiFi module, enabling the Arduino NANO-BLE33 to connect to the internet and transmit data using MQTT.
- **MAX30105 Heart Rate Sensor:** Connected to the Arduino for monitoring the user's heart rate.
- **GSR Sensor:** Attached to the Arduino for measuring galvanic skin response.

3.3.5 Hardware setup diagram

The following diagram illustrates the physical connections between the Arduino NANO-BLE33, ESP-01, and the biometric sensors (Heart-rate and GSR sensors).

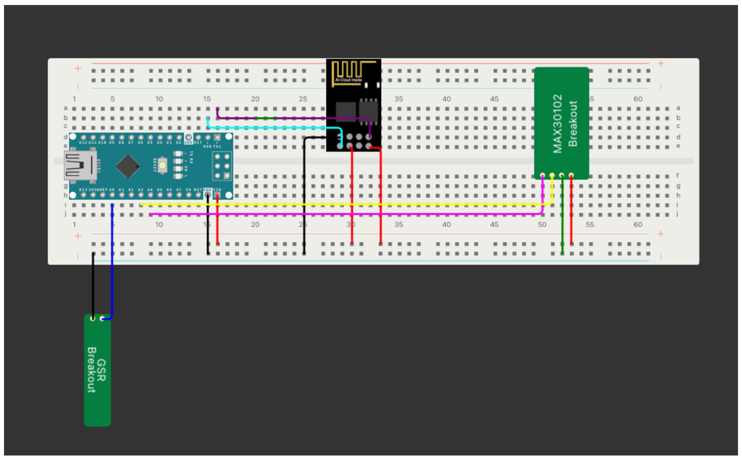

Figure 2: Hardware connection diagram of Arduino NANO-BLE33 with ESP-01 and biometric sensors.

3.3.6 ESP-01

The ESP-01 module is programmed to receive the biometric data from the Arduino NANO-BLE33 and publish it to an MQTT broker. The implementation covers:

- Establishing a WiFi connection.
- Setting up MQTT client and handling reconnections.
- Reading data from the Arduino via Serial communication.
- Publishing the received data to the MQTT broker under the topic "sensor3/data".

3.3.7 Challenges and Solutions

During the integration of the biometric sensors with the Arduino and ESP-01, several challenges were encountered and subsequently addressed:

- **Serial Communication:** The initial challenge was to establish a stable and reliable serial communication between the Arduino NANO-BLE33 and the ESP-01. This was achieved by setting a consistent baud rate and implementing a protocol to ensure complete data packets were sent and received.
- **MQTT Connectivity:** Maintaining a stable MQTT connection, especially handling reconnections and network instability, was critical. This was addressed by implementing a reconnection strategy in the ESP-01's software.

3.3.8 Dedication Time

Approximately 30 hours were dedicated to developing the user data collection system, using most of the time on playing with malfunctioning or hard to use sensors.

3.3.9 MQTT Topics

- **sensor3/heart:** This is the topic used to publish heart rate biometric data.
- **sensor3/galvanic:** Used to publish galvanic resistance data.

3.3.10 Conclusion

The integration of the Arduino NANO-BLE33 with the ESP-01 for biometric data collection and transmission via MQTT represents a significant advancement in the biometric verification system. This setup not only provides real-time data monitoring but also enhances the system's capability to make a prediction on the user's identity.

3.4 Presence and location

3.4.1 Objective and Functionality

The presence and location is a set of two devices each with an ultrasonic sensor that determine user presence and location (left-right). Each of the sensors is responsible of sending collected data to their respective topic (sensor1/distance or sensor2/distance).

3.4.2 Project Definition and Milestones

The development of the presence and location system just involves establishing an MQTT connection with the broker to send the collected data from the ultrasonic sensors.

3.4.3 Achieved milestones, execution order, priority, and dependencies

1. **Milestone 1: Establishing MQTT Communication** - *Priority:* High. Fundamental for data transmission. - *Dependencies:* Basic WiFi setup. - *Execution Order:* First, as it is crucial for data reception.
2. **Milestone 2: Send the collected that from the sensor** - *Priority:* High. Essential for functionallity - *Dependencies:* Successful MQTT setup. - *Execution Order:* Second, building upon established communication.

3.4.4 Hardware setup

The hardware setup for the location and presence comprises:

- Two ESP-01 modules.
- Two ultrasonic sensor, one for each ESP-01 module.

3.4.5 Software Implementation

The software, written in arduino programming language, has the following key functionalities:

- Connect to WiFi (and try to reconnect if needed).
- Establish MQTT connection.
- Measure and calculate distance using ultrasonic sensor pins.
- Send the resulting data to the corresponding topic (sensor1/distance or sensor2/distance)

3.4.6 Dedication Time

Approximately 15 hours were dedicated to developing the System Monitor, mainly spent on finding out sensor malfunctioning given that the software implementation was simple and straightforward.

3.4.7 Challenges and Solutions

- **Hardware Challenges:** Malfunctioning sensors which drove us crazy.
- **Software Challenges:** Processing sensor data accurately, since this cheap sensors can provide really confusing data.

3.4.8 Hardware diagram

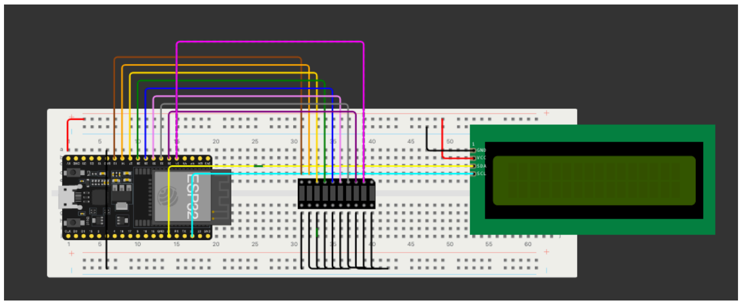

Figure 3: Ultrasonic sensors connected to ESP01 modules.

Data Flow and Configuration

3.4.9 MQTT Tree Structure

The MQTT protocol in the presence and location project uses a structured approach to manage the data flow. Below is the outline of the MQTT topics and their functions:

MQTT Topics

- **sensor1/distance:** This topic is used by the first ESP-01 module. It publishes the distance data measured by its connected ultrasonic sensor.
- **sensor2/distance:** Similar to the first, this topic is for the second ESP-01 module.

Data Organization and Utilization The data sent to these topics include numerical values representing the measured distances.

4 User Story

As the user approaches the authentication system, he is immediately detected by the ultrasonic sensors, which guide him to stand in the optimal position for scanning. As he places his fingers on the biometric sensors, his heart rate and skin resistance data are quietly collected and sent to the Raspberry Pi. At the same moment, the user looks at the camera, and his facial features are analyzed. The Raspberry Pi processes this information using a neural network and merges the results of the facial prediction with the biometrics sensors data by running them into a decision tree which outputs a final prediction, which appears on the screen and will tell which of the authorized users matches with if any. Within seconds, if the system identifies the user, his name is displayed on the monitors.

5 Final Project Conclusions

The development of the Biometric Authentication System has been a path characterized by learning, adaptability, and fighting with malfunctioning sensors. One of the main challenges was synchronizing data from various biometric sensors with the real-time processing on the Raspberry Pi. By applying concurrency principles we managed to overcome this challenges and end up with a significantly accurate system.

Overall, the system has realized robust biometric authentication with an impressive accuracy rate. The modularity of the system design also allows for easy updates and scalability.

For future work, we suggest exploring alternative biometric modalities such as iris recognition or palm-print scanning to further enhance system security.

Additionally, implementing machine learning models with more extensive datasets can offer continued improvement in recognition accuracy and pave the way for personalized user experiences.

6 Technical Documentation

For a in depth technical documentation of the system, detailed instructions on how to setup the system and how to use it, please refer to the following specifications (which can also be found in the project repository):

6.1 Raspberry Pi 4 Configuration

We use the raspberry pi 4 as a: - MQTT Broker - EdgeImpulse node for running the IA for detecting objects.

6.1.1 Configure the raspberry pi

You can configure your raspberry pi liunx image with https://github.com/raspberrypi/rpi-imagerrpi imager, which comes pretty handy since you can:

- Enable wifi with an SSID and password.
- Enable ssh and give it a public key that is allowed to connect to the rpi
- Add a username and password

This saved us a lot of time, and we didn't have to use any ethernet cable or screen for configuring it.

6.1.2 Install mosquitto broker

Install mosquitto with:

[] # We're using the raspbian image for the rpi $sudoapt-getupdate\&\&sudoapt-getupgrade$ sudo apt-get install mosquitto

6.1.3 Run mosquitto broker

Once mosquitto is installed, copy our configuration file and run it with:

[] $mosquitto-cmosquitto.conf-v$

It will serve the mqtt server on port 1883.

6.1.4

6.2 Configure your Linux Computer as an AP of the project

Please watch the figure on assets/network-diagram.png for understanding this article.

6.2.1 Hardware you need:

- A wifi adapter network that supports AP virtual interface (we used an Alfa AWUS036ACH-C, but some integrated wifi cards can handle promiscuous mode and AP virtual interfaces)
- A computer running Linux with systemd.

6.2.2 Software you need:

- https://github.com/lakinduakash/linux-wifi-hotspotLinux Wifi Hotspot

6.2.3 Brief overview of the steps:

- Patch the create_ap program that comes with Linux Wifi Hotspot for enabling an static IP for the raspberry pi.
- Change the Internet interface and the Wifi interface on the ap.conf.
- Execute!

Patch the create_ap program By default, https://github.com/lakinduakash/linux-wifi-hotspotLinux Wifi Hotspot executes: - https://wiki.gentoo.org/wiki/Hostapdhostapd: For creating the wifi access point - https://wiki.archlinux.org/title/Dnsmasqdnsmasq: For adding DHCP server to the network. - https://wiki.archlinux.org/title/Iptablesiptables: For forwarding all the traffic that comes from the AP network (in other words, in your AP network your computer acts as a router, forwarding the packets).

Using the **create_ap** script that comes with the Linux Wifi Hotspot packages become very handy, since we don't have to create the hostapd, dnsmasq and iptables ourselves, buuut has a catch: **it doesn't allow us to put dhcp rules (such as static ip for the raspberry pi) on his config file**.

This is because the **create_ap** script automatically generates the configuration for hostapd, dnsmasq and iptables given the configuration file you gave it (and obviously this config file doesn't have an option for adding dhcp rules...), https://github.com/lakinduakash/linux-wifi-hotspot/blob/d73242ab812284b5ed65275f630b8bc306b725c5/src/

So, we had to patch the **create_ap** script for adding our custom rule! Yay! First of all find where is the **create_ap** script living on your computer:

[] *whereiscreate_ap#/usr/bin/create_apifyou*"0027*reinarchlinux*

Then, patch it with:

[] *patch/usr/bin/create_ap< create_ap_raspberry_pi.patch*

Be aware that this will add a custom rule for the dnsmasq program, which is:

```
dhcp-host=e4:5f:01:f2:b4:df,192.168.33.213
```

If you don't have the same raspberry we're using, the raspberry mac address will change, this means you will have to tweak this patch file. If you're following our network diagram you don't have to change the ip address.

Change the interfaces of the config file Every linux distribution names his network devices differently (and everyone has different devices), so execute the following command:

[] $ipa....17 : wlp0s20f0u2 : < BROADCAST, MULTICAST, UP, LOWER\_UP > mtu2312qdiscmqstateU$

In my case I see that the Alfa Wifi Card is named wlp0s20f0u2, and the ethernet connection to the internet is named enp0s20f0u6, so change the ap.conf acording to this interface names:

```
....
WIFI_IFACE=wlp0s20f0u2
INTERNET_IFACE=enp0s20f0u6
```

Run the access point! Everything is ready now! Execute:

[] $sudocreate\_ap--configap.conf$

If any problem occurs is 99% problem of your wifi card. See the requirements on the hardware section.