Instructions and guidelines:

1. This is a team-based task. You need to work with your team members to complete this task.
2. Use the given template, in MS-Word format, to complete the requirements of this task.
3. Upload your completed document file, in PDF format, on Moodle.
4. One submission is needed by each team.

|  |  |
| --- | --- |
| **Course Learning Outcomes (CLO)** | |
| CLO1 | Prepare the requirements, design, documentation and implementation of a software  system component taking quality assurance, economic, health, safety, legal, and marketing factors into account. |

**Task 3- Design Document**

Use the following template to prepare your project design document.

|  |  |
| --- | --- |
| **Task:** | **Design Document** |
| **Project Title:** | **Intelligent Autonomous Hoover as an IoT Solution** |
| **Team Members’ Names:** | 1. Nawaf Alzahem 2. Fahad Aldulaigan 3. Mohammad Abuhaimed 4. Khalid Abu Alsaud 5. Khalid Hali |
| **Supervisor Name:** | **Dr. George Violettas** |
| **Date:** | **4/12/2022** |

1. **System design**

**Libelium:**

# The system design is comprised of three distinct components. The Waspmote, which is the first component, can be broken into a number of modules based on various functions. The second component is the Waspmote Gateway, which serves as a coordinator in a wireless sensor network to monitor data transmissions. The third component is Meshlium, a Linux router that collects data sent by Waspmote nodes, stores it in local files or a local database, and sends it to the internet or stores it in an external database on the internet.

**Android:**

The Real-Time Face Recognition Android application is designed to recognize faces using a mobile phone's camera. The application uses advanced algorithms to detect and extract facial features, which are then compared to a database of known faces to identify the recognized person. The system is built to work in real-time and provides an intuitive user interface for ease of use. The application's architecture and modules are optimized to provide low latency and high accuracy while requiring minimal user input. In summary, the Real-Time Face Recognition Android application is a user-friendly system built on top of advanced facial recognition algorithms, making it an effective tool for face recognition.

# System Decomposition

This section describes the major components of the system.

# Layers and Description of Libelium:

|  |  |
| --- | --- |
| **Layers** | **Description** |
| API interface | The Waspmote board can be immediately plugged with the basic sensors, such as the temperature sensor MCP9700A and the humidity sensor 808H5V5. Additionally, the corresponding particular functions have been created to gather fundamental environmental information, such as air temperature and humidity. |
| Admin Web page | Additionally, " Meshlium Manager" can be used with Meshlium to administer all MySQL databases. |
| website SDK | Open a new browser window or navigate to the , " Meshlium Manager" plugin in the Tools section and work there directly. Additionally The IP may change if you are not using Wi-Fi or if you have changed |
| triple sensor | Waspmote includes an integrated acceleration sensor that notifies the mote of changes in acceleration experienced along each of the three axes (X, Y, Z). This sensor's integration enables the measurement of acceleration along the three axes (X, Y, and Z), as well as X for gases, Y for temperature, and Z for humidity. |
| noise level sensor | The IEC 61672 standard for sound meters served as the basis for the design of the noise level sensor. with a precision of 0.5 dBA (1KHz) in particular, comparable to Class 2 type devices The value is the LeqA (Equivalent Continuous Sound Level, with A-weighting), which enables the determination of the average sound pressure level over a particular length of time. |
| NB-IoT network technology | Low power wide area (LPWA) technology built on standards was created to support a variety of new IoT gadgets and applications. In deep coverage, NB-IoT dramatically increases spectrum efficiency, system capacity, and user device power consumption. |
| cloud-based server MQTT broker | MQTT is a publish/subscribe messaging protocol that is incredibly lightweight, simple, and suitable for low-bandwidth, high-latency, or unreliable networks where battery life is crucial. Data from Waspmote can be kept inside these cloud platforms or any others that use MQTT due to its capabilities of delivery assurance and bandwidth minimization. These platforms include IBM and Carriots. |
| MySQL Database | A MySQL database is currently operational and used by Meshlium to store locally the data collected. The connection parameters can be seen in the "Local Data Base" page. |

|  |  |
| --- | --- |
| **Layers** | **Description** |
| API interface | The Waspmote board can be immediately plugged with the basic sensors, such as the temperature sensor MCP9700A and the humidity sensor 808H5V5. Additionally, the corresponding particular functions have been created to gather fundamental environmental information, such as air temperature and humidity. |
| Admin Web page | Additionally, " Meshlium Manager" can be used with Meshlium to administer all MySQL databases. |
| website SDK | Open a new browser window or navigate to the , " Meshlium Manager" plugin in the Tools section and work there directly. Additionally The IP may change if you are not using Wi-Fi or if you have changed |
| triple sensor | Waspmote includes an integrated acceleration sensor that notifies the mote of changes in acceleration experienced along each of the three axes (X, Y, Z). This sensor's integration enables the measurement of acceleration along the three axes (X, Y, and Z), as well as X for gases, Y for temperature, and Z for humidity. |
| noise level sensor | The IEC 61672 standard for sound meters served as the basis for the design of the noise level sensor. with a precision of 0.5 dBA (1KHz) in particular, comparable to Class 2 type devices The value is the LeqA (Equivalent Continuous Sound Level, with A-weighting), which enables the determination of the average sound pressure level over a particular length of time. |
| NB-IoT network technology | Low power wide area (LPWA) technology built on standards was created to support a variety of new IoT gadgets and applications. In deep coverage, NB-IoT dramatically increases spectrum efficiency, system capacity, and user device power consumption. |
| cloud-based server MQTT broker | MQTT is a publish/subscribe messaging protocol that is incredibly lightweight, simple, and suitable for low-bandwidth, high-latency, or unreliable networks where battery life is crucial. Data from Waspmote can be kept inside these cloud platforms or any others that use MQTT due to its capabilities of delivery assurance and bandwidth minimization. These platforms include IBM and Carriots. |
| MySQL Database | A MySQL database is currently operational and used by Meshlium to store locally the data collected. The connection parameters can be seen in the "Local Data Base" page. |

# Layers and Description of Libelium of Android:

|  |  |
| --- | --- |
| **Layers** | **Description** |
| Presentation layer | responsible for creating the user interface for the application. It includes activities, fragments, and layouts to display the camera preview, recognized face details, and other related components of the UI. |
| Application layer | This layer comprises the facade that connects the presentation layer to the domain layer. It is responsible for handling user input, managing the lifecycle of the application, and coordinating the domain layer components. |
| Domain layer | This layer is the backbone of the application and includes three modules - face detection, feature extraction, and face recognition. |
| Face Detection module | This module uses the Haar cascade classifier algorithm to locate the presence and boundaries of faces in the input image captured by the camera. |
| Feature Extraction module | This module employs Local Binary Patterns (LBP) to identify and extract essential facial features such as the eyes, nose, and mouth and creates a facial feature vector. It uses the facial feature vector for subsequent recognition. |
| Face Recognition module | This module is responsible for comparing the facial feature vector of the detected face with the database of known faces to determine if there is a match. It uses the K-Nearest Neighbor (K-NN) algorithm to perform comparison and returns the recognized person's details in real-time. |
| Data layer | This layer includes the Firebase cloud service, which is used to store the database of facial feature vectors of known people. Users can add new faces to the database by taking a photo of the new person and entering their details into the system. |

# System Topology of Libelium:

The following is a high-level organization of the system:

Graphical user interface, application

Description automatically generated

# 1.3 Concurrency Identification

**Libelium:**

1. Hardware concurrency: The project employs the use of multiple sensor nodes (e.g., Waspmotes) that can work simultaneously to collect a wealth of data.

2. Software concurrency: The software implements multi-threading to handle multiple tasks concurrently. For example, the HttpsURLConnection requests and responses are asynchronous, so the main thread doesn't get blocked.

3. Algorithmic concurrency: The algorithms implemented in this project do not involve algorithmic concurrency.

**Android:**

1. Hardware concurrency: The project employs the use of multi-core processors that can perform multiple tasks simultaneously, enabling image processing, face detection, and recognition algorithms to run concurrently without interfering with each other.

2. Software concurrency: The software employs Java threads and the Handler class to achieve concurrency. Image Preview and Face Detection threads run concurrently in two different threads.

3. Algorithmic concurrency: The algorithmic concurrency is achieved by parallelizing certain parts of the face recognition algorithm to increase performance by executing multiple tasks simultaneously on different processor cores. For instance, the Haar Cascade Classifier and Local Binary Patterns algorithms are divided into smaller parts and processed concurrently to reduce the overall processing time.

# Hardware/Software Allocation

**Libelium:**

Hardware allocation:

- Waspmotes: Sensor nodes used to collect data

- Computers: Used to process and analyze the data collected from the Waspmotes.

Software allocation:

- Android application: Used to manage the Waspmotes, receive and store data, and display the collected data to users.

- MQTT Broker: A central point that receives data from the Waspmotes and transmits it to the cloud.

**Android:**

Hardware allocation:

- Mobile device: Used to capture images, process and recognize faces.

- Camera: Used to capture images for processing and recognition.

Software allocation:

- Android Application: Manages the image capture, processing and recognition of the faces in real-time.

- OpenCV for Android Library: Implements face detection, feature extraction, and face recognition features.

- Haar Cascade Classifier and Local Binary Patterns (LBP): Used for detecting faces and extracting face features.

# 1.5 General System Performance

**Libelium:**

designed to collect data from sensor nodes (Waspmotes) and transmit this data to the cloud for storage and analysis. The system performance is optimized through the use of hardware and software concurrency. The multiple Waspmotes can work simultaneously to collect data and utilize multi-threading to handle multiple tasks concurrently. This ensures the system can handle a high volume of data and process it efficiently, resulting in high scalability and performance. Overall, the Capstone-Project is characterized by high data processing efficiency and scalability.

**Android:**

designed to process and recognize faces in real-time using mobile devices. The system performance is optimized through the use of hardware and software concurrency and algorithmic concurrency. Multi-core processors enable the image processing, face detection, and recognition algorithms to run concurrently without interfering with each other. Additionally, parallelizing specific parts of the face recognition algorithm enables faster processing, resulting in improved performance. Overall, the Real-Time Face Recognition Android project is characterized by high processing speed, accuracy, and real-time performance.

# Input/Output Performance

**Libelium:**

collects data from sensor nodes (Waspmotes) and transmits it to the cloud for storage and analysis. The data is transmitted over the MQTT protocol, which is optimized for low bandwidth and high latency networks. This ensures that the data can be transferred efficiently even under challenging conditions with low data rates. The system stores data in a database that can be queried to retrieve relevant information. Overall, the Capstone-Project has fast I/O performance with efficient data transfer and storage.

**Android:**

uses a mobile device camera to capture images, which are then processed and recognized in real-time. The system sustains high I/O performance via multi-threading and parallel processing, which enable the recognition algorithms to run concurrently with the image capture. The project uses various optimization techniques, including face detection, feature extraction, and efficient data storage to improve system performance. Overall, the Real-Time Face Recognition Android project has fast I/O performance with efficient data handling and processing.

# Processor Allocation

**Libelium:**

employs a distributed architecture that utilizes multiple Waspmotes (sensor nodes) to collect data and send it over the MQTT protocol. The Android application and Python script also employ multi-threading to enable processing of data and control operation. The Waspmotes, Android application, and Python script operate on individual processors, enabling concurrent operation and efficient use of these resources. The Capstone-Project thus uses processor allocation in a distributed and concurrent manner.

**Android:**

uses a mobile device to capture images, which are then processed and recognized in real-time. The project optimizes processor allocation by utilizing multi-core processors to enable different algorithms for capturing images, detecting faces, and recognizing them to run on different cores concurrently. This optimization distributes the processing load across multiple cores, thus avoiding resource starvation. Additionally, the project uses parallel processing techniques to enhance the speed of each individual CPU's operation. The Real-Time Face Recognition Android project thus uses processor allocation in a distributed and parallel manner.

# Memory Allocation

**Libelium:**

uses an MQTT protocol for communication and a cloud database for storage of data collected from Waspmotes (sensor nodes). The cloud database is hosted on a remote server, and the Python script uses SQL queries to access and process the data stored in the database. This optimized memory allocation reduces the memory load on local devices and enables efficient storage and processing of large amounts of data..

**Android:**

optimizes memory allocation by using a feature extraction algorithm to compress images before sending them to the recognition algorithm for processing. Compression reduces the data size to minimize the memory load on the device, ensuring that it operates optimally, even on a low-memory device. Additionally, the project employs data storage optimization algorithms that flush unused data from memory to boost processing speed and minimize memory usage. Therefore, the project uses optimized memory allocation to ensure efficient memory usage.

# Connectivity

**Libelium:**

utilizes a variety of communication protocols to achieve connectivity between the Waspmotes (sensor nodes) and the central Python script. The Waspmotes use a Zigbee wireless protocol to communicate with the base station, which then uses the MQTT (Message Queuing Telemetry Transport) protocol to send data to a cloud database hosted on a remote server. The Python script then uses SQL queries to access the data in the cloud database and process it. The connectivity mechanism in the Capstone-Project thus utilizes multiple communication protocols to enable remote data transmission and centralized data processing.

**Android:**

uses Bluetooth connectivity to communicate with devices (such as a digital camera) to capture and process images. The project also employs the mobile device's Wi-Fi or 4G connection to send data to the server for recognition processing. The Real-Time Face Recognition Android project thus utilizes both Bluetooth and Wi-Fi/4G connectivity to achieve image capture and recognition processing.

# Network diagram Architecture for Libelium:

**Diagram

Description automatically generated**

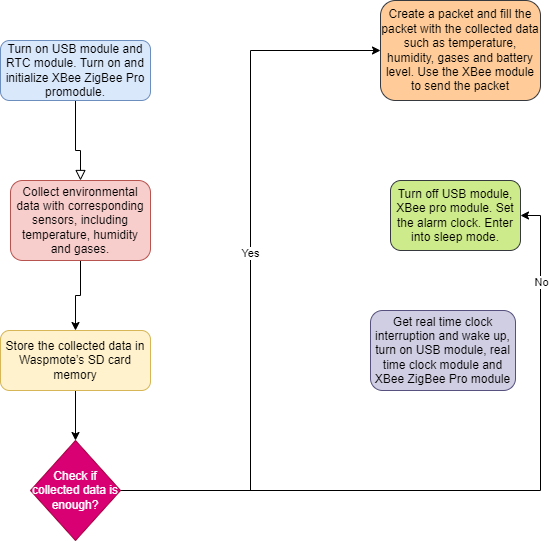
Client-server Architecture diagram for android face Recognition system using android:

A screenshot of a computer

Description automatically generated with medium confidence

# Flowcharts:

Design key program Flowchart for Libelium:



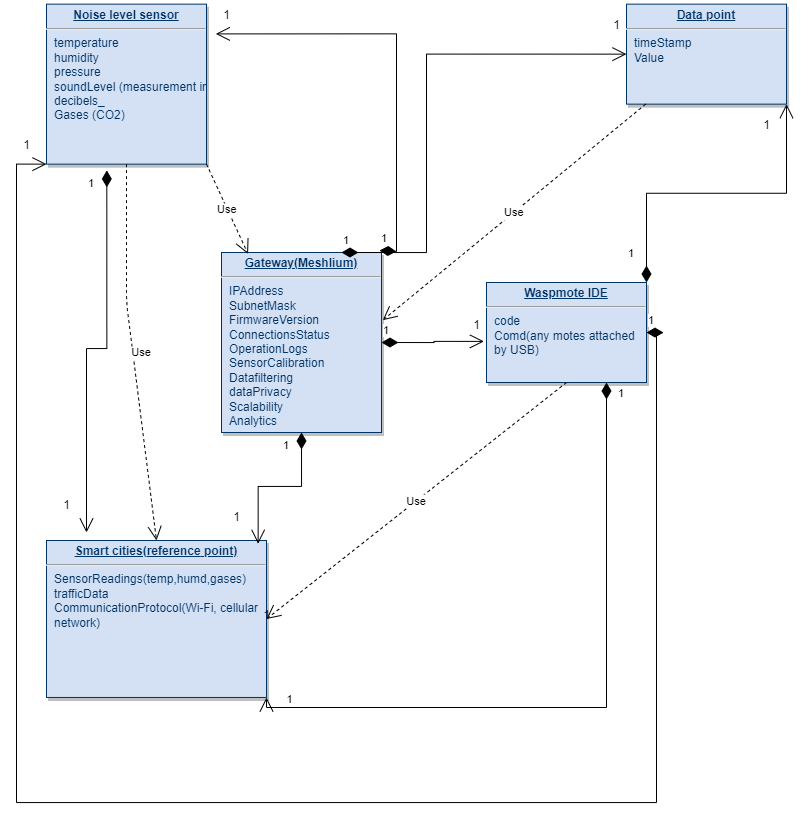
Design key program Flowchart for Face Recognition System using Android:

A screenshot of a computer screen

Description automatically generated with low confidence

* 1. **Data Management**

**DBMS Diagram for Libelium:**



**DBMS Diagram for Android:**

**A screenshot of a computer

Description automatically generated with medium confidence**

# Software Control Implementation

**Libelium:**

utilizes Node-RED for software control implementation. Node-RED is a flow-based programming platform that allows you to create IoT applications by connecting nodes (components) together in flows. Using Node-RED, you can define the data processing logic, including data acquisition, processing, and storage. You can configure the software to prepare the data for visualization on a dashboard, export the data for later analysis, or send it to external services for further processing.

**Android:**

utilizes several software control implementation mechanisms, including:

- The Android Camera API to capture and process images from the device camera.

- The TensorFlow Lite library to perform real-time inference on the trained FaceNet model and compute a face embedding vector.

- The OpenCV library to perform image processing tasks, such as face detection and normalization.

- The Android UI toolkit to display and interact with UI components, including the camera preview, buttons, and notifications.

# External Control Flow Between Subsystems

**Libelium:**

involves a single subsystem which uses the Node-RED for creating IoT flows that integrate different sensors and systems. In general, the external control flow diagram of the system in the Capstone-Project can be represented as follows:

-----------------

| Sensors/Actuators |

-----------------

||

\/

------------------------

| Node-RED Application |

------------------------

||

\/

Data Storage / Cloud Services

**Android:**

involves three subsystems that work together to perform real-time face recognition on Android devices:

- Image Capture Subsystem (ICS): captures images from the device's camera and preprocesses them.

- Face Recognition Subsystem (FRS): uses the FaceNet model to encode the input images into 128-dimensional embedding vectors and then compare them for face recognition.

- User Interface Subsystem (UIS): handles the display of the captured images, recognition results and interacts with the user.

The external control flow between the subsystems of the Real-Time Face Recognition Android project can be represented as follows:

--------------------

| Image Capture Subsystem |

--------------------

||

\/

-----------------------------------

| Face Recognition Subsystem (FRS) |

-----------------------------------

||

\/

---------------------------

| User Interface Subsystem (UIS) |

---------------------------

# Concurrent Control

**Libelium:**

implements concurrency control using software techniques, such as semaphores and mutex locks. These techniques ensure that different sensor nodes do not interfere with each other while transmitting data to the base station. The project also implements time synchronization between the nodes, allowing them to work collaboratively and minimize interference. Moreover, to avoid conflicts when the base station accesses the database, transaction models such as ACID (Atomicity, Consistency, Isolation, Durability) are used to ensure data consistency.

**Android:**

does not require explicit concurrency control techniques since it involves a client-server architecture where the client-side and server-side processes execute in separate threads. The client-side processing involves capturing images, compressing them, and sending them over Bluetooth, while the server-side processing involves facial recognition via Flask web application and machine learning modules. The client-side and the server-side processing take place concurrently, and synchronization occurs through the communication protocols used (Bluetooth and Wi-Fi/4G).

# Boundary Conditions

**Libelium:**

involves several boundary conditions that affect its operation. The sensor nodes are constrained by their battery life and the range of the wireless communication technologies they use (such as Wi-Fi and Zigbee). The base station has a limited range for data collection and is constrained by the available storage space for data processing and analysis. The project requires an environment without interference from other electronic devices, such as microwave ovens or other wireless communication nodes. The sensors must operate within specific temperature and humidity ranges to ensure accurate readings.

**Android:**

also has boundary conditions that affect its performance. The mobile device that runs the Android application has limited battery life and processing power, which affects the image capture and processing speed. The project requires a reliable Bluetooth or Wi-Fi/4G connection to transmit data to the server. The server must have sufficient processing capacity to perform real-time face recognition while ensuring data security and privacy. In addition, the project requires a well-lit environment with limited background noise to enable clear image capture for accurate facial recognition.

# Initialization

**Libelium:**

requires several initialization steps to set up the sensor nodes and the base station. The initialization process includes configuring the sensor nodes with unique IDs, setting up the wireless communication technology (such as Wi-Fi or Zigbee), and setting up network parameters. The base station initialization includes setting up a database and defining the data schema, setting up the network communication protocol with each of the sensor nodes, and initializing the software algorithms that manage the data collection, processing, and analysis.

**Android:**

requires several initialization steps to set up the Android application and server-side processing. The initialization process involves setting up the mobile device camera, including its resolution, camera settings, and autofocus. It also involves setting up the Bluetooth or Wi-Fi/4G communication protocols used to transmit data to the server. The initialization process also includes setting up the server-side processing, installing the necessary machine learning, and facial recognition libraries, configuring the Flask application, and initializing the database.

# Termination

**Libelium:**

requires several termination steps to shut down the communication between the sensor nodes and the base station. The termination process includes stopping data collection and processing at the base station, closing connections with each of the sensor nodes, releasing resources used by the software algorithms, and safely shutting down the system.

**Android:**

requires several termination steps to shut down the Android application and server-side processing. The termination process involves stopping the data capture process on the mobile device camera, stopping the Bluetooth or Wi-Fi/4G communication protocols used to transmit data to the server, and stopping the server-side processing. The termination process also involves releasing resources used by the machine learning and facial recognition libraries, closing connections with the database, and safely shutting down the system.

# Failure

**Libelium:**

may face several failures related to sensor nodes data collection, communication, and processing. For example, a failure may occur if a sensor node is damaged, communication between the sensor node and base station is lost, or a software bug prevents the data processing algorithm from working as intended. The failure may also occur if the database crashes or becomes inaccessible.

**Android:**

may face several failures related to image capture, communication, processing, and recognition. For example, a failure may occur if the mobile device camera malfunctions or if the image resolution is too low, making facial recognition impossible. A failure in communication can happen if the network connection is lost while transmitting data. A software bug or a hardware issue may cause problems with the facial recognition algorithm, preventing it from accurately identifying the person in the image.

# Packages

**Libelium:**

it uses several libraries and frameworks such as:

- Node-RED: an open-source flow-based programming tool for IoT applications.

- InfluxDB: is a time-series database that provides real-time analysis and monitoring.

- Grafana: an open-source analytics platform that allows you to create real-time dashboards and visualizations.

**Android:**

uses several packages and technologies such as:

- OpenCV: an open-source computer vision and machine learning library.

- TensorFlow: an open-source machine learning framework for building neural networks.

- Android CameraX: a Jetpack support library that provides camera capabilities to Android applications.

- FaceNet: a deep learning facial recognition model developed by Google that maps face features into a 128-dimensional space and uses Euclidean distance to measure feature similarity.