UNIVERSIDAD DE PUERTO RICO

RECINTO UNIVERSITARIO DE MAYAGUEZ

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA

ELECTRICA Y COMPUTADORAS

Final Project – Turn THAT light on!

Jason R. Gutiérrez

Víctor A. Fernández Vicente

César A. Delgado Aponte

INEL4206 Sección 030

Profesor Luis Roa

3 de mayo del 2023

Content

General configuration (System Overview)

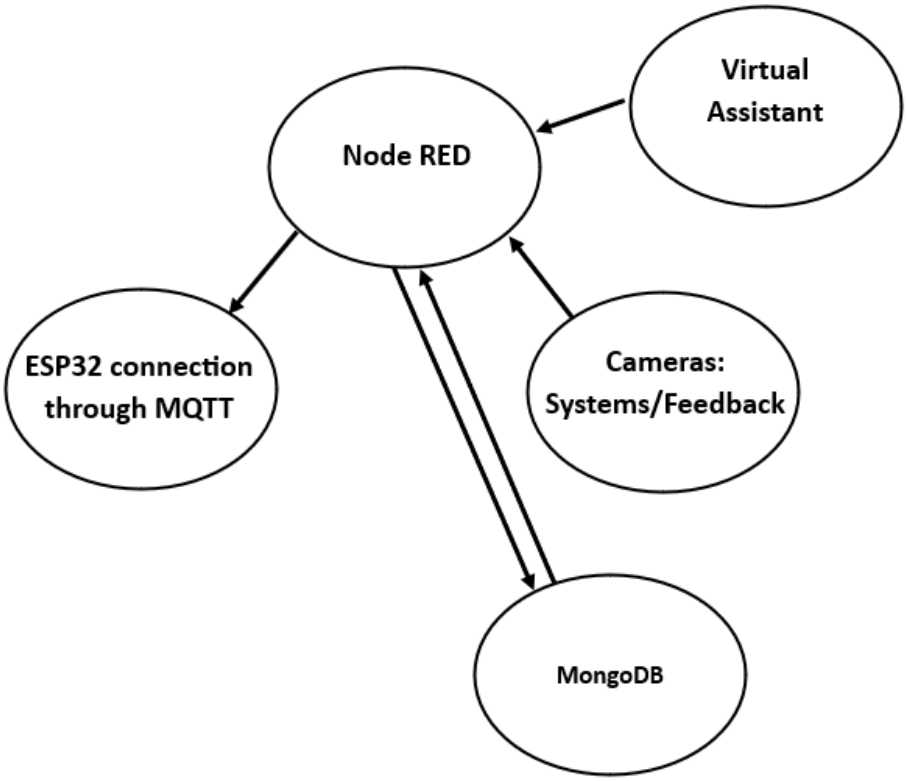
System configuration begins with three primary flows in node red (running on the team's AWS LightSail Server in port 1880). First, key point data relative to the user's shoulders, wrists and hips is received from a live demo running on the server. The primary function node calculates the distance from wrist to hip, changing the value of variable "pointing" to either 100 if the user is pointing left, 200 if pointing right and 300 if no pose is determined. The distance threshold the software uses to determine which direction the user is pointing to is configurable but set to a default of 120. This value is a standard determined after testing was done with all three team members. Afterwards, the pointing value is sent to a helper function node which encases the previous payload into a JSON format in addition to storing the messages timestamp (for further error proofing). This primary flow ends with the user's pointing direction and the exact time it was detected sent to the "Pose" collection inside servers MongoDB database.

The two other primary flows process the users voice command, which is received through a get function executed by Siri. Two domain paths were created to handle the "Turn On" and "Turn Off" command (status1 & status2). For both flows, functionality remains relatively the same: once the user executes a voice command a query is made in the server database for the most recent pose detection and its timestamp. As previously mentioned, the timestamp is checked for relevance, if the current timestamp (of voice execution) differs from the stored timestamp by more than one second, no pose is sent through "pose" topic in MQTT publisher node. And an error string is sent as a response to alert the user of possible bad or outdated data. Otherwise, the most recent pose is published and received in the ESP-32 for control functions.

In terms of the ESP32 circuit configuration, the ESP32 itself is used on a breadboard, with 2 green LEDs and one RGB LED connected to it using cables and resistances. The green LEDs are connected to the ESP through 330-ohm resistances and access the ESP at the GPIO 26 and 27 pins. The RGB LED is connected to the ESP32 using three 1k resistances, each one corresponding to one of the three colors available (red, green, and blue). For the green color, we connect to the GPIO 32 pin, for the blue color we use the GPIO 33 pin, and for the red color we use the GPIO 25 pin. All the LEDs, as well as the ESP32 itself, are connected to common ground.

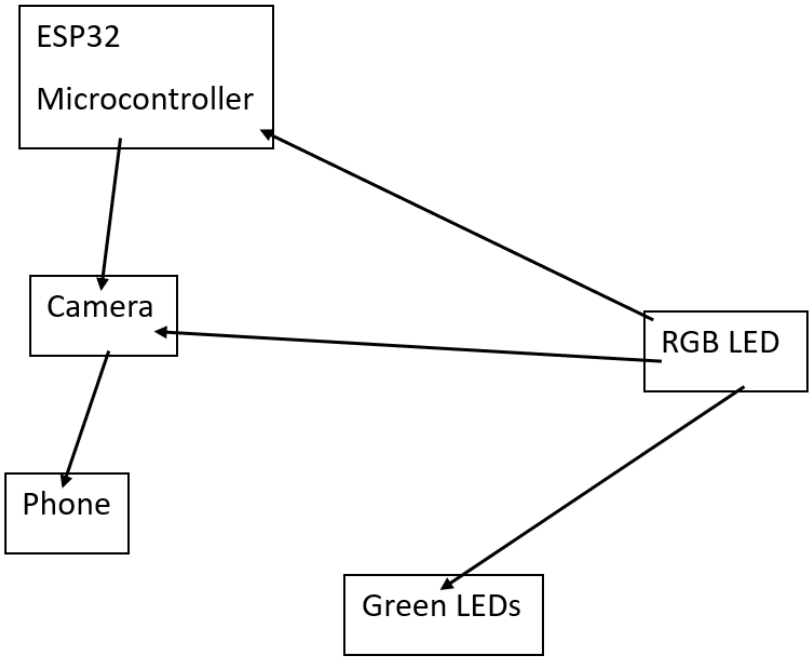
Finally, the ESP32 control code bases its whole functionality around three main functions. First, it connects to the Wi-Fi and the MQTT broker in the setup function. In the loop function it constantly checks that the MQTT broker is connected, and a message is received; if can't establish a connection to broker it calls the reconnect function. This function will try to connect to MQTT every 5 seconds. Secondly, the callback function receives the payload associated to MQTT topic along with its length, it converts each byte of the payload into a character and appends it to the message variable. Lastly, the determine light function receives the message generated in the callback function and then determines which GPIO driver must be turned on, so the LED is also turned on.

Logical Architecture



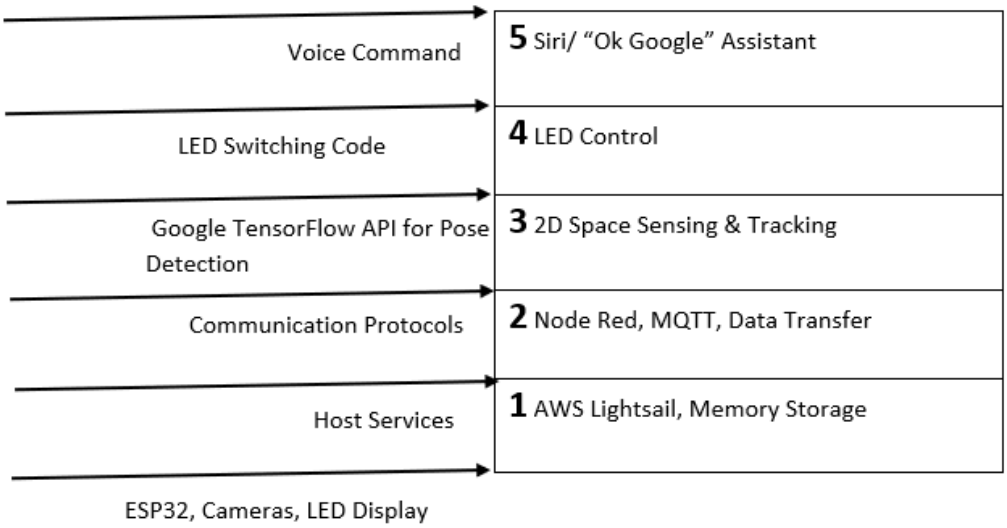
- **Virtual Assistant**: Siri shall send commands to app.
- **ESP32 Connection Through MQTT**: ESP-32 shall be subscribed to "pose" topic on Node Red's MQTT Broker.
- **Cameras System/Feedback**: System to detect and interpret user poses that will communicate with Node Red.
- **MongoDB**: Pose and timestamp data is stored in PoseDB located in server.
- **Node Red**: Houses the majority of critical code concerning pose determination and data management.

Physical Architecture



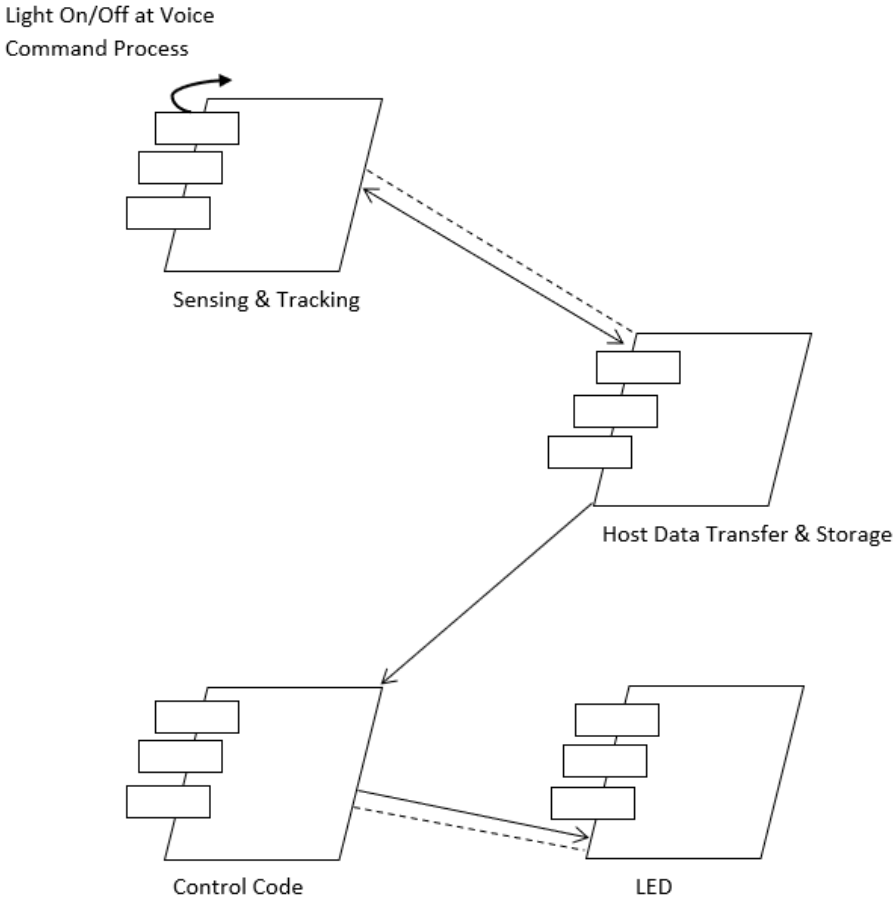
- ESP32: Microcontroller, allows for required real-world scenarios when communication with other components is established.
- Camera: One or multiple camera system that handles pose information when called for by phone component.
- Phone: Cellular device that begins general operation when it receives voice command.
- Green LEDs: Device that turns on when receiving corresponding pose information, depending on the nature of the aforementioned information.
- Red LEDs: Device that turns on or off depending on the efficiency and functionality of the various components in the system, working as a signal representing troubleshooting necessities or correct operation. Used to check Wi-Fi and MQTT/Node-Red connections.

Development Architecture



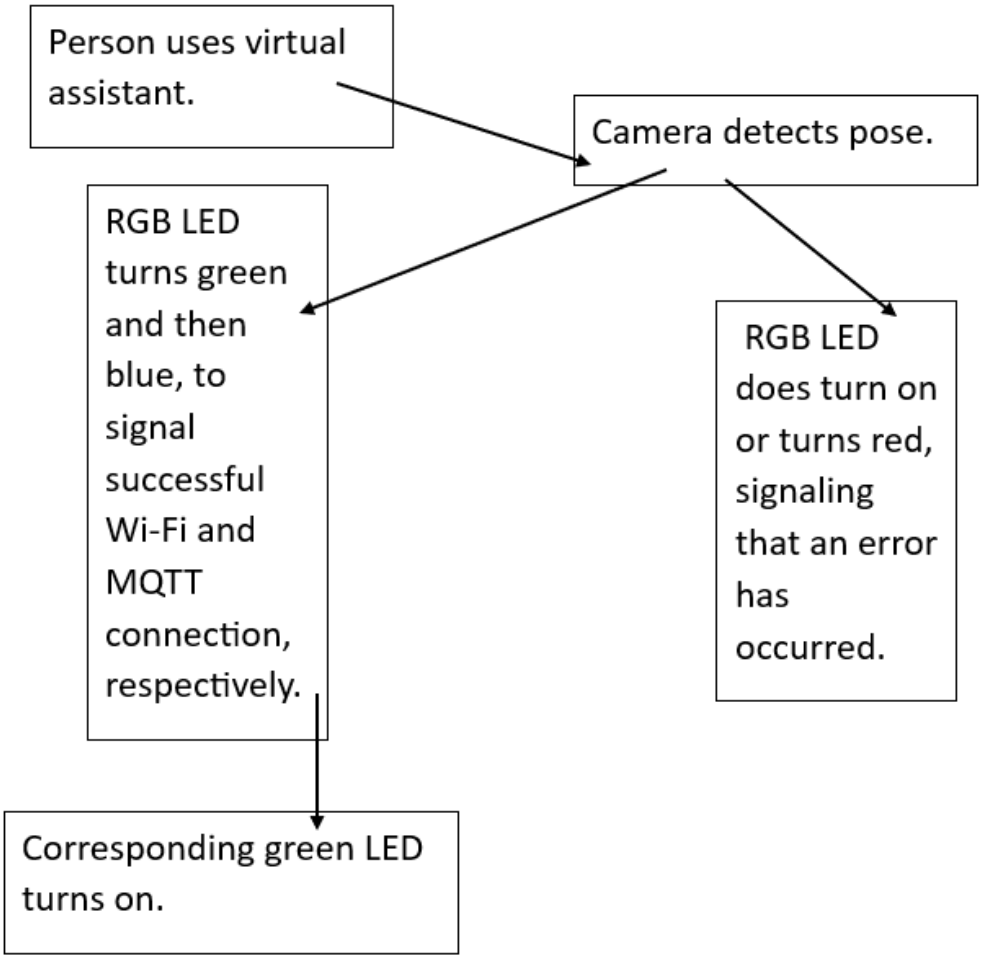
Layers 1 and 2 contain the general requirements for this project: a host server for data storage and transfer as well as general access to the cloud for ESP32. We can start to see the specific subsystems required for this project. At this level communication between the ESP32 and host is handled via MQTT protocol, while backend development is built with Node Red. Layers 3 & 4 are the essence of the project. By utilizing two-dimensional body tracking provided by Google's TensorFlow-Based Pose Detection API, ESP32 can interpret data through implemented code and determine which light to turn on. Finally, Layer 5 contains the higher level of subsystem abstraction which is the triggering voice command issued via Siri or Ok Google assistants.

Process Architecture



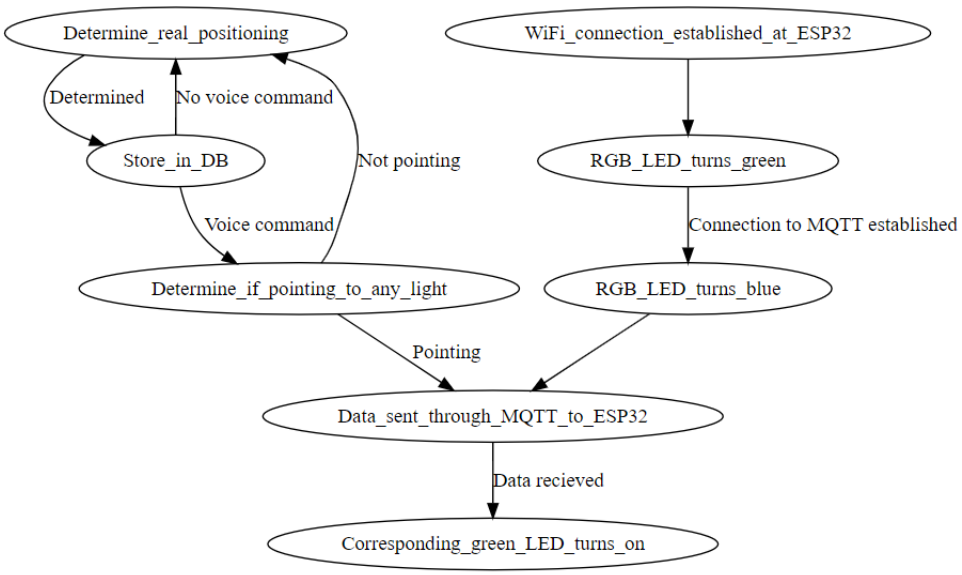
Body tracing is performed once given voice command. At this event, positional data is fed to the host server where it is stored and sent to the next process. After receiving all the necessary information, the control code in the ESP32 determines which LED to turn on or off, which resets the Light On/Off at Voice Command Process.

Scenario


Person uses virtual assistant.
Camera detects pose.
RGB LED turns green and then blue, to signal successful Wi-Fi and MQTT connection, respectively.
RGB LED does turn on or turns red, signaling that an error has occurred.
Corresponding green LED turns on.

Finite State Machine (FSM) Documentation

Diagram



Description:

States:

- Determine_real_positioning
- Store_in_DB
- Determine_if_pointing_to_any_light
- Data_sent_through_MQTT_to_ESP32
- Wi-Fi_connection established at ESP32
- RGB_LED_turns_green
- RGB_LED_turns_blue
- Corresponding _green_LED_turns_on

Events:

- Determined
- No voice command
- Not pointing
- Voice command

- Pointing
- Data received
- Connection to MQTT established

Transitions:

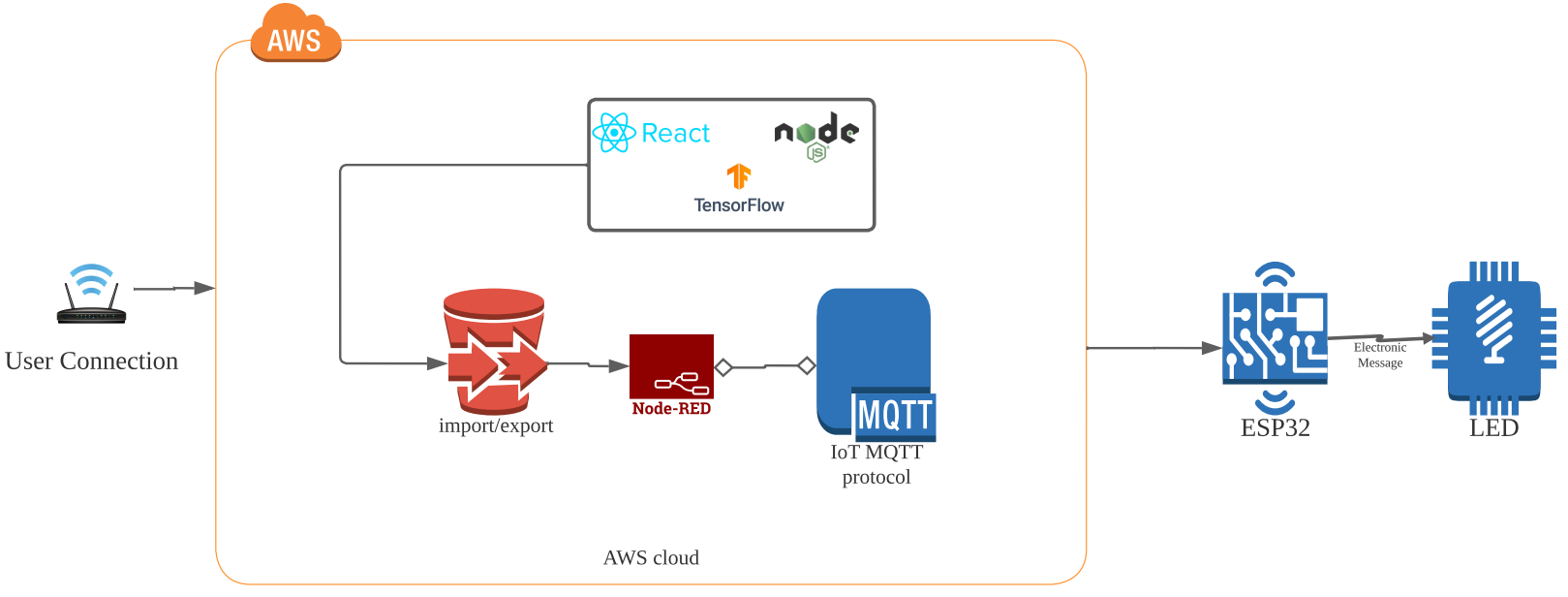
- Determine_real_positioning -> Store_in_DB
- Store_in_DB -> Determine_real_positioning
- Store_in_DB -> Determine_if_pointing_to_any_light
- Determine_if_pointing_to_any_light -> Data_sent_through_MQTT_to_ESP32
- Data_sent_through_MQTT_to_ESP32 -> >Corresponding_green_LED_turns_on
- RGB_LED_turns_blue -> Data_sent_through_MQTT_to_ESP32
- Wi-Fi_connection_established_at_ESP32 -> RGB_LED_turns_green
- RGB_LED_turns_green -> RGB_LED_turns_blue

The Finite State Machine works as follows:

- The Google API, utilizing TensorFlow models, will be continuously registering the coordinates of the various key points.
- When a voice command is issued, one of the node functions will determine the real coordinates of the wrist and elbow key points. The other function will calculate the distance between the key points and determine if the result corresponds to a desired pointing position.
- If position data is relevant, the MQTT node within Node Red will publish a message to the ESP32 in order for it to turn on the corresponding green LED (during a successful boot up of the system, the RGB will turn green to signal a connection to Wi-Fi and will then turn blue to signal a connection to the MQTT broker).

PROVISIONING AND CONFIGURATION

Jason Gutierrez, Cesar Delgado, Victor Fernandez



Representation of two-dimesnional Rigid Body Diagram