UNIVERSITY OF MORATUWA

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION

EN2560 – INTERNET OF THINGS DESIGN AND COMPETITION

PROJECT REPORT

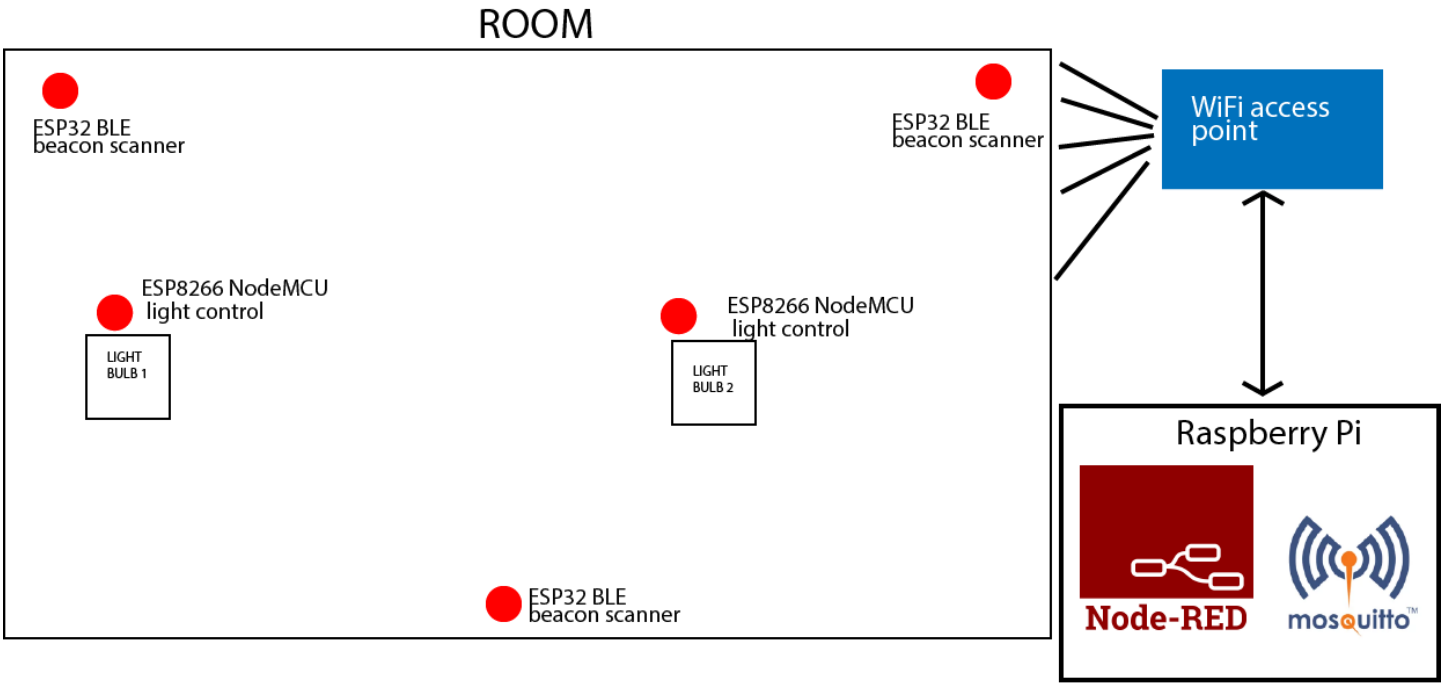
Indoor Positioning with BLE beacons using triangulation

KUMARA EDA	180333X
PETHANGODA RM	180472V
WIJITHARATHNA KMR	180717E

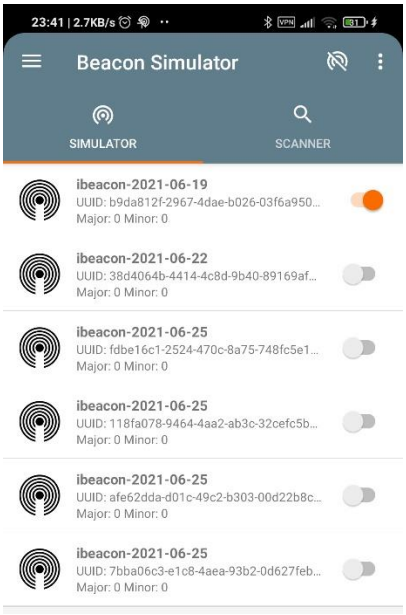
Problem intended to be solved

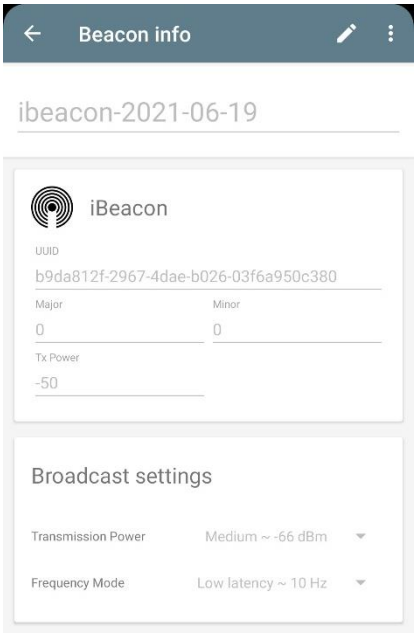
In presence-based automation systems, there is little intelligence of the actual position of the user by the system. The proximity based BLE beacons fails to accommodate scenarios where the user is not very close to the system. We intend to solve this problem by using 3 BLE beacon scanners. With the proposed system it is desired to achieve positioning even at areas where it is impossible to plant a BLE beacon or a scanner.

System overview



The designed system consists of 3 ESP32 modules working in BT + WiFi dual mode. The user must have a beaconing device when he enters the room. This is achieved by using a generic app in Playstore. We have used iBeacons in the implemented system.





The UUIDs of the beacons are used to identify a known user. When a known beacon is received by any of the beacon scanners, they publish the UUID and the Received Signal Strength Indication value along with the scanner ID to a specific topic on MQTT broker running on the RaspberryPi. The published data from all three beacon scanners is processed by a Node Red flow which is also hosted on the RaspberryPi. The relevant decisions are taken according to the data and the two lights are turned on accordingly.

Furthermore, the system integrates a weather API to support the decision making. One of the lights will only turn on after a previously set time of the day. The Node-Red dashboard is used for this purpose and other important data are also displayed on the Node Red dashboard.

The designed system is intended to identify three positions inside the room distinctively.

System operation

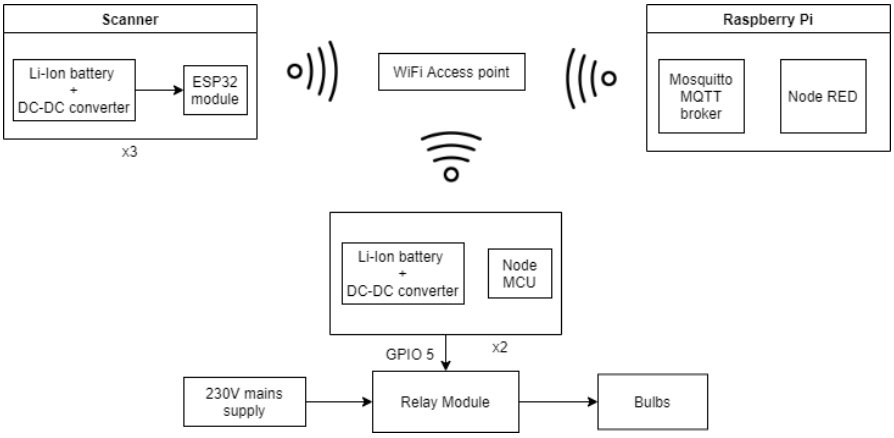
User with a beacon enabled smart device enters the room. The RSSI values corresponding to the UUID of the device is sent to the Node Red flow. It identifies whether the user is inside the room or not using the RSSI values of the received beacons and operates the lights accordingly.

The RSSI values depends heavily on the device being used. Hence initially a calibration process is carried out. User must use the “Calibration” tab of the MQTT dashboard and select his device and the position he intends to calibrate, and then start the calibration by clicking the button. He and the device should be stationary during this process. The Node-Red flow stores the corresponding RSSI values in **text files** on RaspberryPi local storage. Hence this calibration is only needed to be carried out once, the data is preserved even with system restarts.

After calibration, the flow reads in the data from the files. The current RSSI values from a specific device is continuously checked with the previously calibrated data to identify the current position of the user. The detected position is displayed in the Node Red dashboard as well.

If the user is inside the room & the set time has passed or user is in a specified position the lights will turn on. This is achieved by using two ESP8266 Node MCUs which are also subscribed to the same MQTT broker on different topics. A relay module is used to handle high currents.

Implementation



ESP32 modules: (Kolban, 2018)

Three ESP32 modules are used in dual mode. They all support Bluetooth 4.2 BLE beacon scans. They are connected to a WiFi access point, MQTT protocol is used to transfer data to the Node Red flow. Since MQTT broker, Node Red are all hosted on a local RaspberryPi server no internet connectivity is required. (Spiess, BLE Human Presence Detector using an ESP32 (Tutorial, Arduino IDE), 2017)

Power consumption considerations

Despite the fact that the beacon scanners are unable to operate on batteries due to their high-power consumption, efforts were put to minimize the amount. Without these optimizations an ESP32 consumed around 150mA when idle, ~300mA current spikes when booting, scanning and publishing. Since the computations are minimal, we have reduced the core clock speed down to 80MHz, furthermore we tried ESP32 deep sleep mode in order to turn off the modules when no beacons are in range. However, they periodically boot up and scan for any beacons before going back to deep sleep again. After the optimizations the power consumption went down to ~50mA when idle, 140mA when scanning. During deep sleep ~11mA was used, this is mainly due to the quiescent current

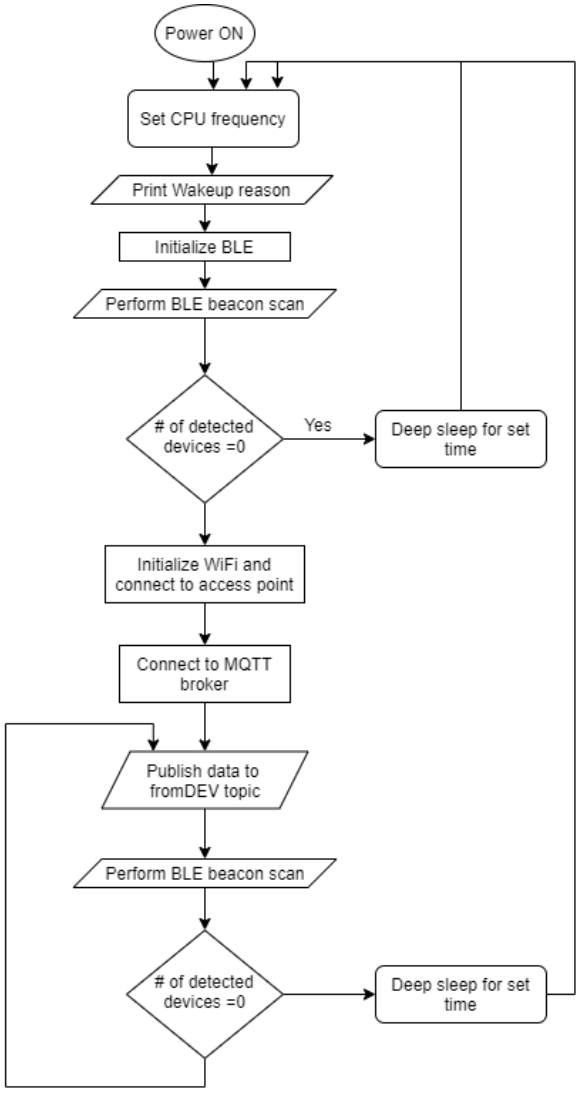
of the onboard regulator (AMS 1117 3.3V) and the LED. Deep sleep was achieved by calling the esp_deep_sleep_start(); function. (Spiess, ESP32 Deep Sleep, RTC Memory, "Secret" LoLin Pins, 2017)

```
_FACTOR);
Serial.flush();                                    :ing(TIME_TO_SLEEP) +
esp_deep_sleep_start();
```

During the deep sleep all the variables will be lost except the data stored in the RTC memory (Real Time Clock), which is 8kB, plenty for our usage. The variables could be stored in RTC memory by simply adding "RTC_DATA_ATTR" macro in front of the variable declaration.

```
RTC_DATA_ATTR int TIME_TO_SLEEP = 15;
RTC_DATA_ATTR int bootCount = 0;
RTC_DATA_ATTR BLEScan* pBLEScan;
RTC_DATA_ATTR const char* ssid     = "AndroidAPE6C9";
RTC_DATA_ATTR const char* password = "123456789";
RTC_DATA_ATTR int rssi = 0;
```

Program flow ESP32



The ESP32 publishes to the "fromDEV{scanner ID}" topic. For example, the scanner 1 publishes to the "fromDEV1" topic. The PubSubClient library only supports QoS levels 0 and 1. However our attempts to use QoS 1 was not successful, furthermore it does not support persistent messages. For security purposes we set up our MQTT broker to allow user logins. (O'Leary, 2020)

```
if (MQTTclient.connect(clientId.c_str(),"dev1","group_6$")) {
  Serial.println("MQTT broker connected");

MQTTclient.subscribe(tsleep,1);  //subscribe at QoS level 1
```

Since we are using deep sleep power mode, a different approach must be taken in the program. Only the void setup() is used, each time the ESP boots up from the deep sleep the setup() function is executed, hence the void loop() is left empty.

Active scanning is used in BLE, the scanning period is set to 5 secs, scan interval is set to 300ms and scan window is set to 100% (300ms) of scan interval for maximum detection.

To reduce the effect of noise RSSI values from 8 packets from a device is averaged and published to the MQTT broker.

Each detected beacon packet (Advertisement) will call the callback function defined in pBLEScan-> setAdvertisedDeviceCallbacks(). Two arrays are updated inside the BLE callback function with the beacon data.

```
////////////////////////////////////BLE SCAN INITIALIZATION///////////////,
BLEScan *pBLEScan;
BLEDevice::init("DEV_3");            //Initialize BLE
pBLEScan = BLEDevice::getScan(); //create new scan
pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
pBLEScan->setActiveScan(true); //active scan uses more power, but get resu
pBLEScan->setInterval(300);
pBLEScan->setWindow(300);  // less or equal setInterval value
```

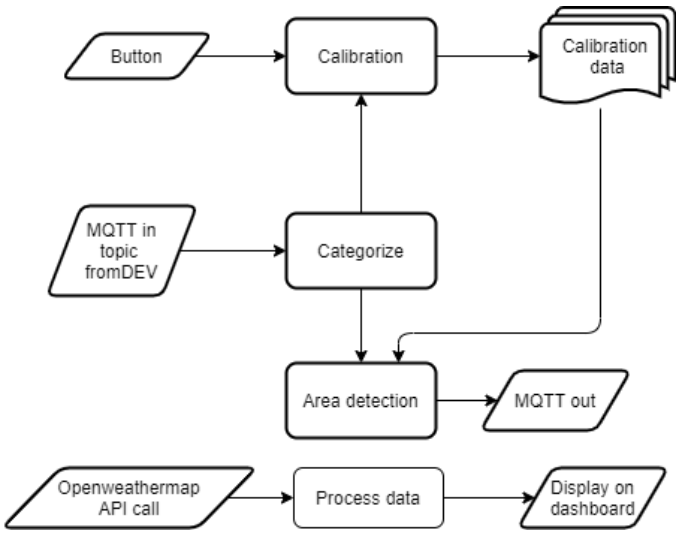
ESP8266 Node MCU relay modules and light

Two Node MCU modules are used in the implementation. They both subscribe to two different topics on the MQTT broker. If the command to turn on the light is received, they will turn on the respective relay by using GPIO 5 pin.

Furthermore, to override the system's operation, user can connect to the NodeMCU's hosted web server using its IP address and use the links.

Node Red Flow

The MQTT messages are first categorized based on the User device and scanner IDs, creating global variables containing each user device's RSSI signal strength from three scanners. The developed system can support up to three user devices. UUIDs are used in this categorization. After this phase, the RSSI values for each device are continuously compared with the calibrated values which were obtained from the files on the local storage of RaspberryPi. (Cope, 2018)

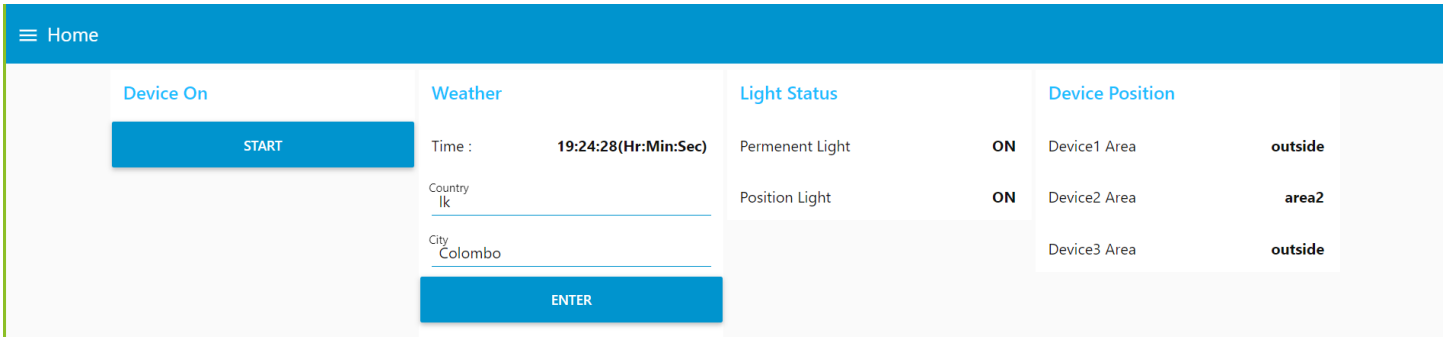


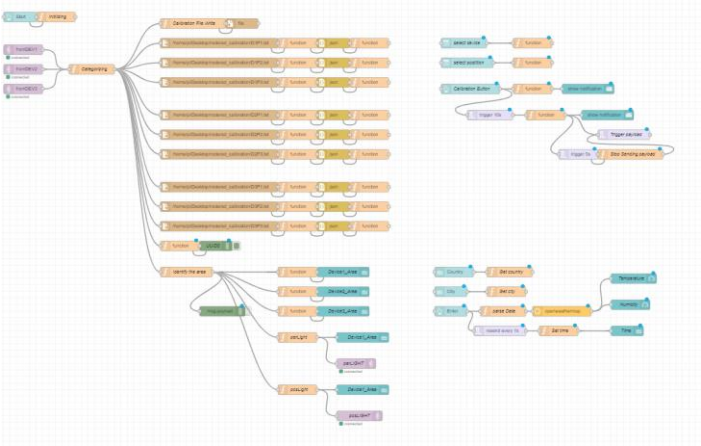
The categorize function further reduces the effect of noise by using a median filter. Each time a message arrives from any of the three scanners the function stores the received value on a stack and when the stack reaches the length of 5, it is sorted, and its middle value used for the rest of the program.

System functionality achieved

The intended functionality of the system was achieved. When the user is inside the room first bulb was turned on, when the user was moved to a pre calibrated position the second light was turned on.

However, the functionality was not robust, there were some occurrences of false detections. The calibration process had to done repeatedly to achieve good performance. Furthermore, some devices showed very good signal transmission strengths leading to being unable to detect the device is out of the room. We suspect this is mainly due to the implemented indoor area was being very spatial.





References

Cope, S. (2018, May 21). *Read and Write Data To a File In Node-Red*. Retrieved from Youtube: https://www.youtube.com/watch?v=bI0bQ7pO1kA&t

Kolban, N. (2018). *Kolban's Book on ESP32.* Texas US.

Node-RED. (n.d.). *Node Red Documentation*. Retrieved from https://nodered.org/docs/

O'Leary, N. (2020, 5 20). *Arduino Client for MQTT Documentation*. Retrieved from Arduino Client for MQTT: https://pubsubclient.knolleary.net/api

Spiess, A. (2017, December 31). *BLE Human Presence Detector using an ESP32 (Tutorial, Arduino IDE)*. Retrieved from Youtube: https://www.youtube.com/watch?v=KNoFdKgvskU

Spiess, A. (2017, July 30). *ESP32 Deep Sleep, RTC Memory, "Secret" LoLin Pins*. Retrieved from Youtube: https://www.youtube.com/watch?v=r75MrWIVIw4