
UNIVERSITY OF PISA

DEPARTMENT OF COMPUTER SCIENCE

Master Degree in Computer Science

# Exploiting Neural Networks and BLE in IOT Devices for Accessibility

Thesis Advisor:

**Prof. Stefano Chessa**

**Dr. Michele Girolami**

Candidate:

**Francesco Fattori**

ACADEMIC YEAR 2021/2022

# Contents

# List of Figures

# Chapter 1

# Introduction

In the recent years, the evolution of technology brought us IoT devices with always more powerful processors, lower power consumption, arrays of sensors and communication methods. They can be powered by small batteries, they can also be worn as bracelet or necklace given also the low weight and small size and then they can transmit data via Wi-Fi, Bluetooth or also cellular. We are living in a world where the majority of the information are given in a visual way: lights, colors or screens. These IoT devices could play a huge role in the accessibility field to help visually impaired people to navigate in their daily life outside their houses. The technology most utilized in this mobile field is the Bluetooth, especially the Bluetooth Low Energy (BLE). With this protocol small devices with a chip that supports it can advertise data at regular intervals, being powered by a single coin cell battery. Each one of these advertisements is called beacon, and usually it has an unique identifier (UUID) and some custom data. The goal of this thesis is to investigate the best combination of Hardware and Software to analyze Bluetooth Low Energy (BLE) beacons emitted by some BLE Tags and determine if they are in proximity. Proximity is a relative term and it depends on the context in with we are operating. We have seen in the recent years, during the COVID-19 outbreaks, that contact tracing apps, like Immuni[8], or other evaluation papers[34] defined the proximity as two people less than 2 meters apart for few minutes. There have been also studies of proximity in the sense of how people tends to interact with each other and how they perceive the proximity .The science that studies these phenomena is known as Proxemics. One of the greatest exponents was Edward T.Hall that in 1966 studied and defined the interpersonal distances of humans in four distinct zones[21] (Figure 1.1): the intimate spaced, the personal space, the social space and the public space.

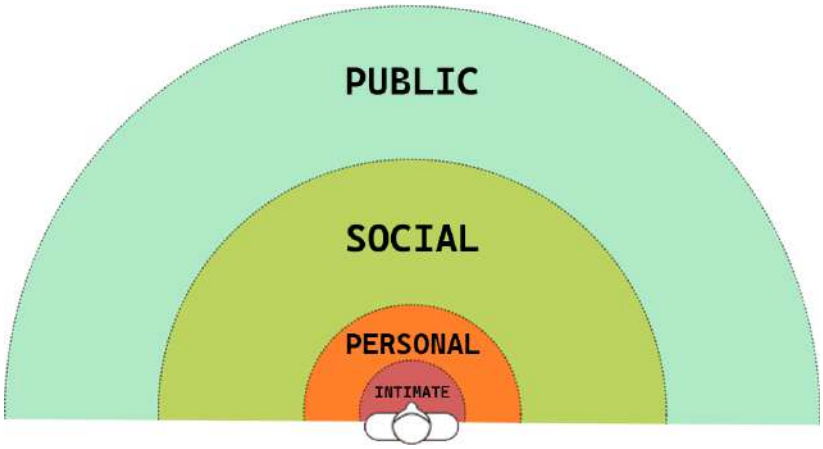


Figure 1.1: Edward T.Hall Interpersonal Distances of Humans

In this work we are evaluating the performance, the cost and the effectiveness of collecting BLE beacons, processing them and running Neural Networks and on an embedded device to estimate proximity between people or between a person and a point of interest in the Social, Personal and Intimate Space (between 0 and 2.5m). Furthermore we want to use the findings to develop a smartphone app capable of helping blind or visually impaired people navigate their way outside their houses. The application must be able to notify the user with a multi-channel feedback its proximity with some point of interests, represented by BLE tags.

## 1.1 Objectives

The main objectives for this work are:

- Evaluate the adoption of micro-controller to collect and analyze BLE beacons on board.
  There are different micro-controllers available on the market, each one with their specifications and strengths. One of the goals for our work is to create a portable IoT device, so we need to find the best device that has a low power consumption and a low beacon loss rate.

- Designing a Neural Network that can detect the proximity of BLE Tags on a micro-controller.
  We want to approach the proximity problem with something more powerful than a simple threshold to have the possibility in the future to study signal strength patterns and develop new accessibility features with datasets that could include also the cardinal position of a specific Tag.

- Collecting a representative dataset based on interactions with various orientation

  The training of a Neural Network requires an accurately acquired dataset that must includes all the possible scenario in which the interaction could happen. We want to acquire beacons with all the possible pattern of orientation between two actors (one with the tag and one with the micro-controller), in this way we could have better results also in case of signal attenuation from not having a direct line of sight between the emitter and the microcontroller.

- Evaluating the Neural Network experimentally

  We aim at testing different types of Neural Network and trying to tweak every single parameter experimentally to get the best performance for each type of network we choose.

- Creating a case study with the created Neural Network, namely the BLEind App

  Finally, we want to use all the finding in this work to create a real use case scenario that could help visually impaired people navigating outdoor and breaking down the barriers that texts, lights and colors create for them.

## 1.2 Methodology

In this section, we detail the adopted methodology to achieve the previously described objectives.

We start by reviewing the state-of-the-art in order to identify what technologies has already been explored for proximity and IoT devices. It appear that no similar work was made. All the existing works doesn't use neural networks or exploit the tags for indoor localization. Then we explore the available micro-controllers on the market that are easily obtainable by consumers and have a Bluetooth module that supports BLE. We quickly discover that the Bluetooth library of Arduino, one of the micro-controllers picked, doesn't provide an API to extract the payload data of a BLE beacon, so we work on the implementation of this functionality in the library. To this purpose, we review some GitHub repositories and we acquire technical documentation to understand the available code, modify the source code of the library and push the changes on our GitHub repository [9]. Then we run tests on the different micro-controllers to analyze the capability of capturing beacons, the beacon-loss rate and their power consumption.In this way we can pick the micro-

controller that has the perfect combination of reliability and power consumption. Then we start developing the neural network from scratch. To do this we need:

- A dataset to train the network
- The actual neural network

We then start the data collection campaign with the creation of the dataset that is a 2 hours session of beacons sampling that included UUID, RSSI, timestamp and label. This is a log of a real world Bluetooth traffic scenario, that later on we could process and aggregate to create training sets or simply pass the individual values to the training process. This dataset, once processed, is split in Training Set, Validation Set and Test Set. In this way we can train our neural network and test the trained network with values that it has never seen in training. After the creation of the neural network it follows an empirical tweaking phase to discover which training and structure parameters were having the best results. After this, to be sure that our idea would actually work in the real world, we run every test also on the actual devices.

## 1.3 Obtained Results

After a study on what embedded device would fit the best our use case, it came out that the most reliable device was the EspressIf ESP32, with lower power consumption, a way more stable BLE beacon reception rate and a low beacon loss rate. We then created a dataset using two actor and placed them in 9 positions for fixed amount of time to be sure that our dataset included the case in which the signal is attenuated by the body of a person. In this dataset, analyzed in Chapter 4.4, we have 23987 rows of individual beacons acquired with UUID, RSSI, Timestamp, Label. We explored two different types of Neural Networks. The first network was a fit forward fully connected neural network based on TensorFlow's Dense layers, while the second one was based on LSTM layers. The network that showed better results was the second one, that achieved an average precision and recall of 0.84. During the application development of the use case we explored also which protocol had better connection stability and had a lower power consumption. We run different tests and it came out that the best pick was Bluetooth Serial.

# Chapter 2

# Background and Related Work

This chapter aims to give a over all presentation of all the technologies explored and their state of the art. It details the Bluetooth and BLE protocols, the different microcontroller with a BLE network interface that we analyzed, the TinyML framework we used for machine learning on embedded devices and the framework we used for building the use-case application, BLEind.

## 2.1 The Bluetooth Protocol

Bluetooth is a short distance wireless communication standard that use the 2.4GHz spectrum. It use key authentication and all messages are encrypted. Once two devices are paired they can exchange file or services with up to 2 Mbps bitrate, depending on the protocol version and of the signal strength.
BLE (Bluetooth Low Energy) [3] is an extension of the Bluetooth protocol introduced in 2010 to lower the chips cost and the power consumption. Nowadays it is the standard for portable IoT devices like smartwatches or fitness bands. It use the same 2.4GHz frequency of the Bluetooth classic while preserving battery life by lowering transmission range, speed and exploiting the*sleep state* of Bluetooth when no data transfer are happening. An advertising BLE packet has a 2 bytes header and a payload with a variable size between 6 to 37 bytes. The length of the payload is store in a 6 bit field in the header.
There are three main BLE advertising packet format that could be transmitted in the payload: iBeacon[2], AltBeacon[1] and EddyStone[6].

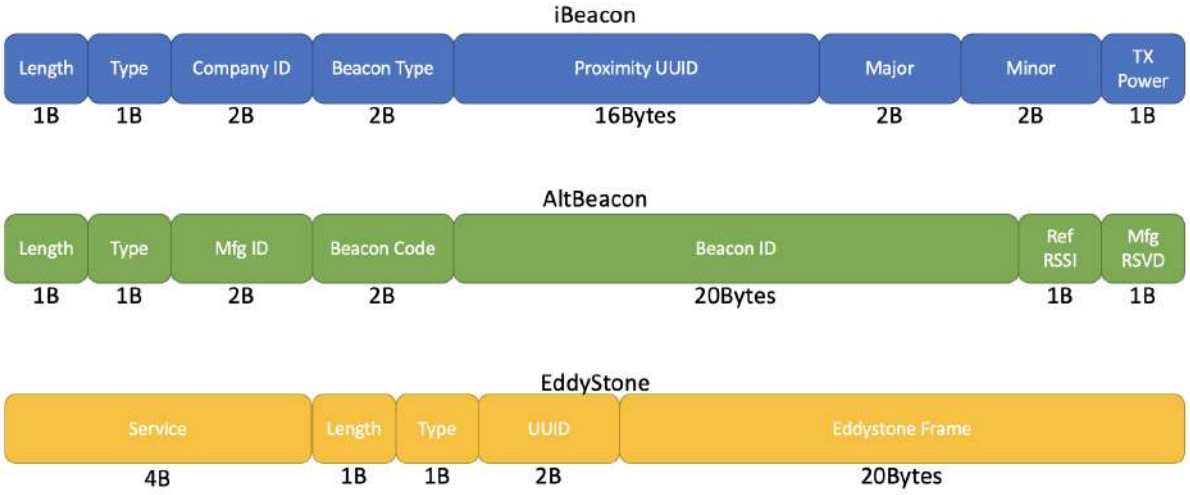


Figure 2.1: Schematic of the different BLE advertising packets

### iBeacon

iBeacon was created by Apple and revealed to the public in 2013 at the Developer Conference (WWDC). It is the first BLE protocol and originally iBeacon was design only with Apple devices (iPhone, iPod and iPad) in mind and to use this technology your had to enroll via a specific iBeacon Certification Program. Nowadays is supported by the majority of BLE SDKs for all devices, including Android.
As we can see in Figure 2.1, an iBeacon payload has the following structure:

- Length (1 Byte): Length of the Advertising packet
- Type, Company ID and Beacon Type (5 Bytes Total): this represents the iBeacon protocol identifier.
- UUID (16 Bytes): this represents an alphanumeric 32 characters identifier for the emitter
- Major (2 Bytes): personalized Integer, often used as Group ID for beacons
- Minor (2 Bytes): personalized Integer, often used to identify a beacon in a Beacons Group (Major)
- TX Power (1 Bytes): calibrated TX Power at 1m of the device

### AltBeacon

AltBeacon is an open-source alternative developed by Radius Network. It has a similar structure to the iBeacon frame structure. The 2 Bytes Major and the 2

Bytes Minor fields are embedded alongside the 16 bytes UUID in the Beacon ID field. We also have a field similar to the TX Power field of iBeacon called Reference RSSI and a 1 Byte field that is manufacturer based and could transmit the battery level or room temperature for example, depending on the implementation.

#### EddyStone

EddyStone is an open-source protocol developed by Google and it's cross-platform by design.
The generic structure of an EddyStone payload is:

- Service (4 Byte):list of Advertised services.
- Length (1 Byte): length of the Advertising packet.
- Type (1 Byte): type of EddyStone Frame data.
- UUID (2 Bytes): 16-bit fixed EddyStone UUID.
- EddyStone Frame (20 Bytes): variable payload based on the type.

An EddyStone emitter could exploit the 20 Bytes EddyStone Frame depending of the Type. There are four type in this protocol: UID, EID, URL and TLM.
UID is a iBeacon like type and it includes a 16 Bytes identifier (UUID) and a TX Power at 1 meter field. It is often used to trigger notification on mobile devices or trigger in-app actions.
EID is like UID but it broadcasts a rotational ephemeral identifier.
URL is a package that emits a 17 Bytes encoded HTTPS url to nearby devices.
TML is a telemetry package that can be broadcasted alongside other types and it includes Battery Voltage, Beacon Temperature, Number of Sent Packets and Beacon Uptime.

## 2.2 Micro-controller supporting the BLE network interface

We analyzed the two most popular and inexpensive micro-controller that have a BLE network interface. These two devices are the Arduino 33 Sense [3] and the ExpressIf ESP32.

**Arduino 33 Sense**

Figure 2.2: Arduino 33 Sense from the top

The Arduino 33 Sense is the first embedded device from Arduino to be AI enabled out of the box. It is an evolution of the standard Arduino Nano, but with a newer 32-bit ARM Cortex M4 CPU that runs at 64MHz, 256KB of RAM and 1MB of program memory that are capable of running heavier and bigger firmwares. It also has a built-in modem that supports both WiFi, Bluetooth and NFC. To complete the feature set it also includes an inertial sensor (LSM9DS1), a humidity and temperature sensor (HTS221), a barometric sensor (LPS22HB), a microphone (MP34DT05) and a sensor that can estimate proximity, light color and light intensity (APDS9960).

**ExpressIf ESP32**

Figure 2.3: ESP32-WROOM-32 from the top

The ESP32 is a low-cost system on a chip developed by the company ExpressIf. It has a 32-bit Xtensa LX6 processor with two cores that run at 240MHz, a built-in Wifi and Bluetooth module and antennas. The device we are going to use is the NodeMCU development kit in the version ESP32-WROOM-32 that allows us to connect to the Serial port via a simple USB-microB connector. This device is superior to the Arduino in all his specifications while having a very small price (around 4€ for the NodeMCU ESP32 and 30€ for the Arduino 33 Sense).

## 2.3 TinyML learning framework

Tensorflow [9] is an open-source library for machine learning that lets you create and train neural network in a high-level language. Tensorflow has different APIs to

integrate support for multiple languages, like Python, C++ and Node.js. TensorFlow Lite instead is a mobile library to make inference on compressed and mobile-optimized TensorFlow neural networks on less powerful device such as smartphones and tablets. Both the microcontrollers taken in consideration early are supported officially by Tensorflow Micro[13]. Tensorflow Micro is a library for microcontrollers that enables the load of a TensorFlow Lite model and the execution of inferences on it.

## 2.4 Flutter

Flutter is an open-source framework for Dart from Google relased in 2017. From the version 2.0 it allows the development of multi platform applications, with support of Windows, MacOS, Linux, Android, iOS and Web. Flutter is composed by mainly two parts: an SDK and a Framework. In SDK we can find all the tools needed for the development (ex. the compiler). At the core of each exported application there is the Flutter Core, written in C++, that allows a low level graphic rendering and other handy feature during development such as the possibility of live updating previews that doesn't require recompilation ("Hot-Reload"). And then the Framework, that include reusable components such as Buttons, Sliders and Inputs. Each component in Flutter is called "Widget".

## 2.5 Related Work

In this section we will explore some of the existing exploitation of BLE and mobile devices.
Mackey et al. [31] highlighted how RSSI can be influenced by the environment and they proposed a statistical approach to solve the problem applying Bayesian filters. This is an interesting approach but we wanted to explore some more dynamic system that could also take in consideration RSSI patterns, not just the computed distance given by the signal strength.
Croce D. et al. [24] presented a system aimed to help visually impaired people navigating outdoor, but they used BLE beacons just to help their pose estimation algorithm based on computer vision. This approach has two main drawbacks: low smartphone battery life due to the constant use of camera and CPU for the neural network running onboard and relying on custom road markings that would need constant maintenance to prevent fading.

Kim J. et al. [28] presented a turn-by-turn navigation using BLE tags and the smartphone compass. They used a web server to compute the navigation and played an audio direction message whenever a specific beacon get in reach. The main problem with this is that they don't take in consideration the real proximity of the emitter, and in the real-world environment, things like wind, can sometime boost a radio signal, resulting in a wrong information.
Deepesh P. C. et al. [25] wanted to explore indoor localization with iBeacon BLE Tags via fingerprinting. They got good results in a test room, but they arrived to the conclusions that fingerprinting BLE beacons is not the solution to the indoor localization problem due to the unreliability of the RSSI signal of iBeacons, the resources needed to fingerprint a large area and the requirement that the area must not change in the future, otherwise the fingerprinting process has to be done from scratch. They think that iBeacon is more functional for proximity estimation.
Anna Fay E N. et al [32] developed a faculty attendance tracker system that exploited BLE Tags and the professors' smartphones. Each BLE Tag represented a class and when it came in the range of signal of the smartphone running their application, it would be communicated to a central server that manages all the presences crossing with the shifts in the school schedule.
Alexandre Alapetite et al. [18] implemented a system for disabled people that exploit the proximity of the user to a BLE Tag as a switch for home devices. Through the payload they built different type of switches, from relays for an LED lamp to a virtual play/pause button for a music player.

All these approaches use the proximity of a BLE beacon in a static way, computing the distance from it with a formula. Our approach is more dynamic and put the fundamentals for more future features, like RSSI pattern recognition. Also we don't run any neural network on the user's smartphone to prevent battery drain and we don't use any remote web server, so an internet connection is not required.
There are other implementations that are not using Bluetooth, but are still interesting to analyze.
R. Santhana Krishnan et al. [30] described in their work a way to alert the blind person via vibrations, buzzer and Text-To-Speech of the presence of an object in the way or a puddle. They created an extension for the blind's white cane bottom part with an ultrasonic sensor, a fire detector and a moisture sensor. These sensors are in charge of detecting object, fires, or puddles. Another very interesting feature implemented by them is the SOS Button. The blind person that lost his/her way can send automatically an SOS SMS with the GPS location to the Emergency Con-

tact saved via a GSM module installed in the cane and a relay server that receive the request and send the SMS. Also Zeeshan Saquib et al. [34] described a similar approach with the addition of few more sensors, like a MQ2 gas sensor for detecting smoke.

Teja Chava et al. [22] approached the problem in the same way via ultrasonic sensors, but they conceptualized a pair of smart shoes and glasses that can communicate with each other, enabling a wider field of (ultrasonic)view. In addition to this connected IoT devices approach they wanted to use a brush-less motor on the shoe to make the ultrasonic sensor spin and have a 360 degree field of view. In Table 2.1 we reported a summary of the state of the art regarding the fields of Bluetooth Low Energy and Accessibility for blind and visually impaired people.

| Authors | BLE | Sensors | Goal |
|---|---|---|---|
| Mackey et al. [31] | ✓ | - | filtering RSSI |
| Croce D. et al. [24] | ✓ | camera | outdoor navigation |
| Kim J. et al. [28] | ✓ | compass | outdoor navigation |
| Deepesh P. C. et al. [25] | ✓ | - | indoor localization |
| Anna Fay E N. et al [32] | ✓ | - | attendance tracker |
| Alexandre Alapetite et al. [18] | ✓ | - | proximity switches |
| R. Santhana Krishnan et al. [30] | ✗ | ultrasonic, fire and wet | object avoidance |
| Zeeshan Saquib et al. [34] | ✗ | ultrasonic, fire, smoke and wet | object avoidance |
| Teja Chava et al. [22] | ✗ | ultrasonic | object avoidance |

Table 2.1: Summary of related works and state of the art

# Chapter 3

# Detecting BLE Beacons with Micro-Controllers

In this Chapter we provide an overview of the BLE Tag we used, evaluate the libraries available for each micro-controller and the micro-controllers them self, to make sure that every library provided RSSI and UUID values for each beacon received and that each device could operate in a mobile environment reliably.

## 3.1 BLE Tag

A BLE Tag is a small chip, usually battery powered, with a BLE antenna that advertise data at regular interval. Each one of these advertisements is called beacon, and usually it has an unique identifier (UUID) and an advertisement message (payload). Our Tag has been configured to use the iBeacon protocol. Our tag has a firmware that allows to configure this advertisements with a companion Android app called "SKYBEACON"[12] provided by GlobalTag.

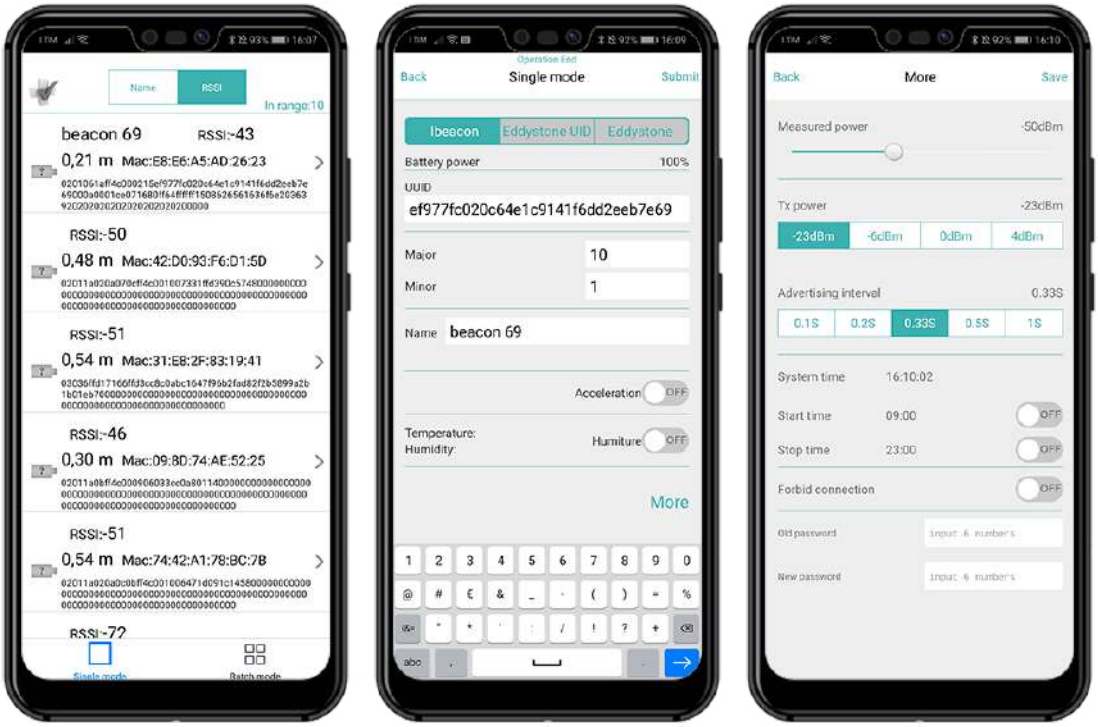


Figure 3.1: SKYBEACON app interface

The mobile app allows customizing a number of parameters, as reported in the following:

- UUID: 32 characters beacon identifier
- Major: broadcasted integer value
- Minor: broadcasted integer value
- Name: friendly Bluetooth name of the device
- Tx Power: power use to advertise messages (in dBm)
- Advertising interval: time in seconds of each advertisement

## 3.2 BLE Library for Arduino

This section reports the approach we followed to enable an Arduino board to detect and collect BLE Beacons. In order to access the Bluetooth functionality we used the standard Arduino library, that is ArduinoBLE[11]. This library is provided by Arduino and it exposes APIs to interact with Bluetooth Classic and BLE devices.

With this library it is possible to scan for the the available devices, pair with them, connect to them, scan for BLE advertisements or exchange data via GATT. GATT is the data-exchange responsible in the BLE protocol with the connection oriented model. BLE has three main communication models (Figure 3.2):

- Point-to-Point: a connection based model that requires scan, connection and paring
- Data Broadcast: a connection-less based model that has an emitter and multiple observer
- Mesh: a large scale connection based model for the interconnection of BLE devices

We used the Data Broadcast connection-less model and we exploited the continuous scan of BLE advertisements API provided by the ArduinoBLE library.


Figure 3.2: Bluetooth communication models

At the time of writing the present work, the ArduinoBLE does not expose the Manufacturer data of the received BLE beacon or any of its derived data. This required a modification of the internal library code, that has been pushed to our remote GitHub repository [9].
The library is composed by mainly 6 classes:

- BLECharacteristic: is a container that manages a value. A BLE Server can expose this characteristic and a BLE Client can read or write in it.
- BLEDescriptor: is an attribute that contain additional information about a BLE Characteristic.
- BLEService: is a pool of Characteristics that a BLE Device has.
- BLEDevice: represents a BLE Device and with this class we can for example retrieve characteristics and services.

- BLEStringCharacteristic: represents a BLE Characteristic with a String value.
- BLETypedCharacteristics: represents a BLE Characteristic with non-String value.

The changes are done inside the class BLEDevice and are mainly two additional functions: manufacturerData() that returns a String containing them and hasManufacturerData() to check if they are available for the given beacon.

```
const char* BLEManufacturerData::manufacturerDataToString(
    const uint8_t* data, uint8_t length)
{
    static char manufacturerData[32 * 2 + 1];
    char* c = manufacturerData;

    for (int i = 0; i < length - 1; i++) {
        uint8_t b = data[i];

        utoa(b >> 4, c++, 16);
        utoa(b & 0x0f, c++, 16);
    }

    *c = '\0';

    return manufacturerData;
}

String BLEDevice::manufacturerData() const
 {
   String manufacturerData = "";

   for (int i = 0; i < _eirDataLength;) {
     int eirLength = _eirData[i++];
     int eirType = _eirData[i++];

     if (eirType == 0xFF) {
       manufacturerData = BLEManufacturerData::
    manufacturerDataToString(&_eirData[i], eirLength);
       break;
     }

     i += (eirLength - 1);
   }
```

```
33
34    return manufacturerData;
35  }
36
37  bool BLEDevice::hasManufacturerData() const
38  {
39    return (manufacturerData().length() > 0);
40  }
```

## 3.3 BLE Library for ESP32

While on Arduino we had to do some changes to the internals of the main Bluetooth library, the standard bluetooth library for ESP32 has already the features we need. The libraries we need are in the package called "ESP32 BLE Arduino"

We used a fork of this library called "NimBLE-Arduino"[10]. This fork was made with memory optimizations as the main focus.

## 3.4 Performance Evaluation of the BLE Libraries

In this Section we analyzed the main requirements that a mobile device must have. Our device must be able to:

- acquire beacons with a frequency of at least 2HZ
- store messages to being able to cover multiple interactions in a time span of at least 2 minutes
- receive and process beacons in real-time
- maintain an average low power consumption

### 3.4.1 Analysis of BLE Beacon rate

We started evaluating the reception of beacons for our two micro-controllers, plotting on three charts the reception of beacons of three distinct tags.

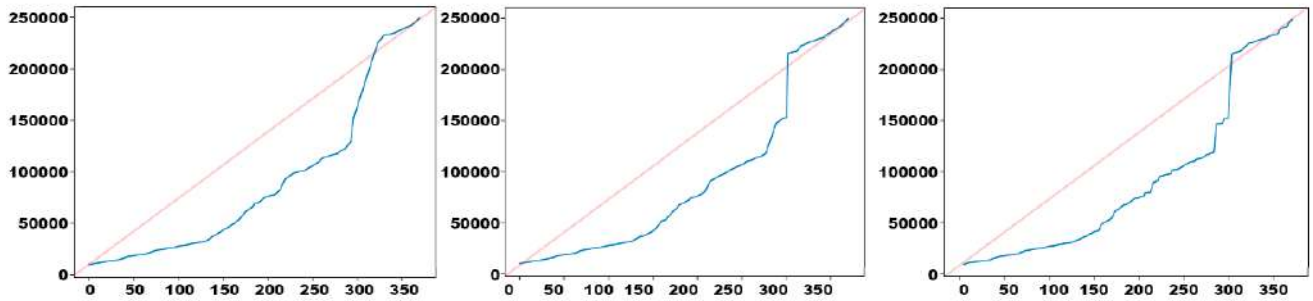


Figure 3.3: Number of BLE Beacons received along the time by the Arduino

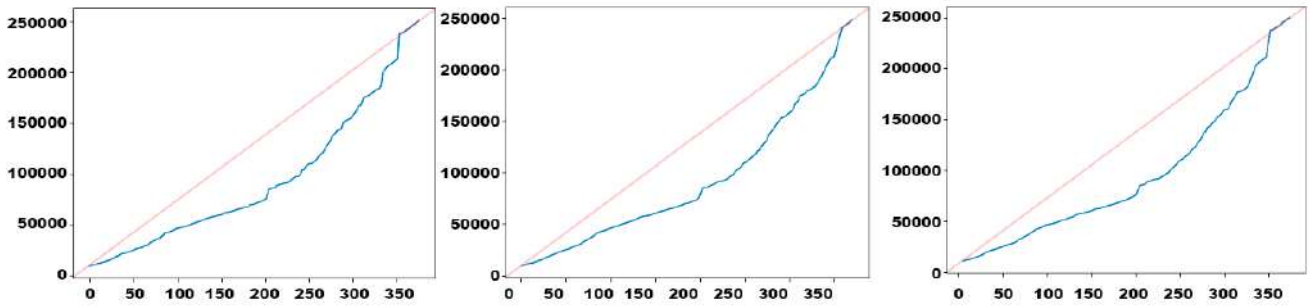


Figure 3.4: Number of BLE Beacons received along the time by the ESP32

In the Figure 3.3 and 3.4 we can see the results of three scans for each device. Figure 3.3 shows the number of BLE beacons received by an Arduino 33 Sense in 350 seconds, while Figure 3.4 shows the same scan made by an ESP32. The ideal chart would be a straight 45 degree line (represented in red in the figures) , meaning that the number of received beacons grows linearly over the time, without beacon losses or reception delay. We can see that the Arduino 33 Sense was receiving almost all the BLE beacons, but it had recurrent reception delay, breaking our requirement of real-time reception. The ESP32 gets close to the ideal linear reception, showing an average consistency during the scan.

### 3.4.2 Analysis of Power Consumption

Another important metric related to the portable nature of these devices is power consumption. The Power Consumption of an IoT device is very important because a higher value means that the battery that will power it will need to be charged more often and this could determine the usability of our work. We decided to log the Ampere that the Arduino and the ESP32 use running in loop BLE scans using a current clamp. The full log is visible in the Figure 3.5.

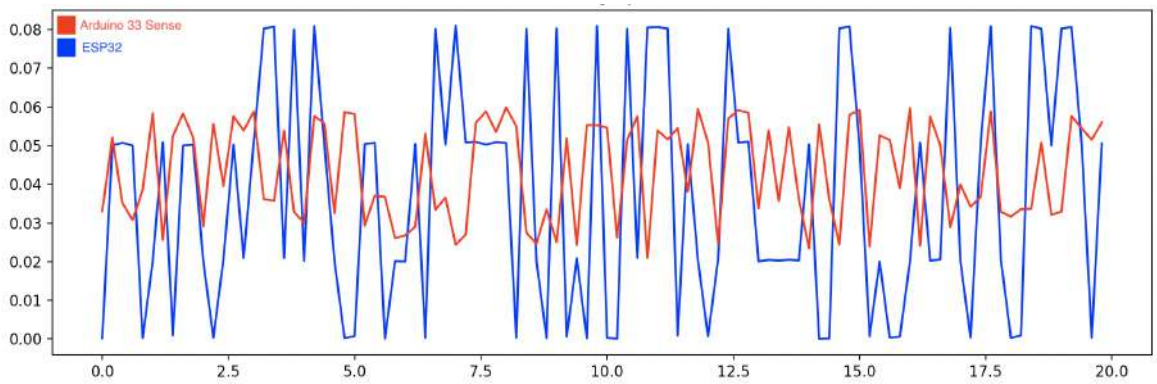


Figure 3.5: ESP32 and Arduino 33 Sense Power Consumption

| | Max A | Min A | Avg A |
|---|---|---|---|
| ESP32 | 0.080A | 0.003A | 0.031A |
| Arduino | 0.059A | 0.020A | 0.045A |

Table 3.1: Power Consumption Analysis

### 3.4.3 Results

Given our initial requirements and the results of the tests explained in this section we decided to base our work on the ESP32 board for its reliability and power consumption.

Expecially the Arduino problem in the delayed reception of beacons is very concerning for our domain. We need to be sure that a beacon nearby get processed as soon as possible, not several seconds after. Even if we can see in Figure 3.4 that the ESP32 has a higher current draw these spikes are balanced by the 0A that often it consumes. The Arduino has a lower maximum current draw but its average is higher than the ESP32, resulting in a potential shorter battery life.

# Chapter 4

# Detecting Proximity with TinyML Micro-Controllers

In this Chapter we develop the second objective of this work: designing and implementing a Neural Network that could run on the ESP32 micro-controller.

## 4.1 Design of the Neural Network for Proximity Detection

### 4.1.1 Proximity Detection

In our work with the term proximity, we refer to a measure of distance between two targets, ie. people, objects or people with point of interest, of about 1.5m for the majority of the time in a time frame of 15 seconds. To estimate this proximity we exploit the RSSI values that our receiver antenna compute. RSSI stands for "Received Signal Strength Indicator" and it is a measurement that indicates the power level that our device is receiving from a wireless emitter. It is regulated by the 802.11 standard but each vendor can adjust the range (from 0 to RSSI_max), making comparison between different receiver chipset useless. This value is related in the first place with the transmitted power of the emitter (TX Power), to the sensitivity of the receiving antenna, but also to the environment in which the two devices are located, including physical objects that could absorb the signal or other radio signals on the same frequency that could create interference.

Figure 4.1: RSSI fluctuations during an optimal proximity test.

In the Figure 4.1 we can see the RSSI fluctuations during a proximity test. The red line reports the proximity ground-truth and the blue dots reports the RSSI value of the BLE beacons received. This is the optimal case scenario in which a beacon is really close to the receiver antenna or at 5 meters.


Figure 4.2: RSSI fluctuations during a real case proximity test.

In the figure 4.2 instead we can observe how noisy the RSSI value can be in a regular use case where the emitter is not at centimeters of distance in the proximity case. In fact, working with RSSI lead to some problems to take in considerations:

- Multi-path fading

  This happen to all the wireless communication and is a phenomenon that occurs when the emitted signal takes different paths, for example reflecting or refracting on different surfaces, and can create duplicates or interference.

- Beacon loss-rate
  This could happen in different scenario where between the emitter and the receiver there is some kind of barrier that attenuates the signal enough to cause the loss of some beacons.

- Device Calibration
  All beacon protocols expose a TX Power value in the advertisement package that needs to be calibrated with the RSSI value received at 1m of distance. The main problems are two. The first is that this value measured changes with the usage of the tag's battery. The second one is that this value also is influenced by the receiver antenna.

## 4.2 The Design of a Fully Connected Neural Network

The first neural network we build was based on Tensorflow's Dense layers to create a feed forward fully connected network. We decided to analyze capturing windows of 15 seconds and we applied some statistical functions to create the inputs for our network.

We decided to calculate the following features based on the RSSI values of the window:

- AVG: average RSSI value (Equation 4.1)
- STD: standard deviation of the RSSI values (Equation 4.2)
- MAX: maximum RSSI value
- MIN: minimum RSSI value
- SKW: skewness of the RSSI values (Equation 4.3)
- KURT: kurtosis of the RSSI values (Equation 4.4)

$$Average = \frac{1}{n}\sum_{i=i}^{n} x_i \tag{4.1}$$

$$Std = \sqrt{\frac{1}{N}\sum_{i=i}^{n} (x_i - \bar{x})^2} \tag{4.2}$$

$$Skew = \frac{1}{N}\sum_{i=1}^{N}\frac{(x_i - \overline{x})^3}{\sigma^3} \tag{4.3}$$

$$Kurt = \frac{1}{N}\sum_{i=1}^{N}\frac{(x_i - \overline{x})^4}{\sigma^4} \tag{4.4}$$

Standard Deviation measures the dispersion of our RSSIs values, Skewness measures the symmetry of our distribution of them, while Kurtosis measures if our values are heavy-tailed or light-tailed in respect of the normal distribution. These three metrics are important for feeding our network a representation of the pattern of RSSIs in the current 15 seconds window. All the values in dBm has been normalized between 0 and 1 with a scale from 0 to -120dBm. Values of RSSI >=-120dbm can be considered too far, such value also represents the lower bound for the ESP32 chip of signal strength. The normalization is very important to allow the neural network an easier convergence without triggering large gradient updates that sparse and double digit number can cause.

Our neural network has three Dense layers separated by two Dropout layers, to avoid overfitting, and an L2 regularizer applied with a coefficient of 0.001. The first two Dense layers have $ReLU$ as activation function and have respectively 128 and 64 neurons. The last Dense layer has sigmoid as activation function with a single neuron, in order to have a boolean classification.

The neural network has been compiled with adam optimizer and the loss was calculated based on the Mean Square Error (mse), while monitoring also the accuracy as a metric.

The high-level representation can be seen in Figure 4.2, while the coding implementation can be seen in Figure 4.3.

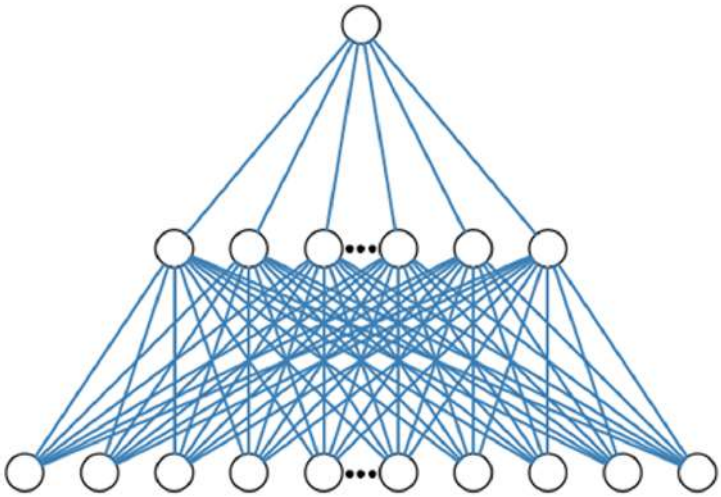

Figure 4.3: Fully Connected Neural Network architecture

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(64, activation="relu"))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))
model.compile(optimizer="adam", loss="mse", metrics=["acc"])
```

Figure 4.4: TensorFlow implementation with Dense layers

## 4.3 The Design of the LSTM Neural Network

The second neural network we built was based on LSTM and its TensorFlow's layer. LSTM stands for Long Short-Term Memory and it's in the family of Recurrent Neural Networks (RNNs), so nodes can be connected with them self, creating a cycle, so that their outputs can affect the next input received. The peculiarity of this type of RNN is that it carries information across multiple timesteps. This was created to solve the problem of vanishing gradients that a simple RNN has, forgetting the time dependency across a long time sequence adding complexity to the training and poorer results. To solve this problem it implements the Gates, that manage the recurrent signal and give the possibility to change the focus between present and past dynamically by changing the weight on them.
We decided to analyze capturing windows of 15 seconds and give these vectors as

input to the LSTM layer.

Our neural network has one LSTM layer formed by 32 neurons followed by two Dense layers. The first Dense layer has 128 neurons and has *ReLU* as activation function, while the second Dense layer has a single neuron with *sigmoid* as activation function to produce a boolean classification.

The neural network has been compiled with adam optimizer and the loss was calculated based on the Mean Square Error (mse), while monitoring also the accuracy as a metric.

The high-level representation can be seen in Figure 4.4, while the coding implementation can be seen in Figure 4.5.

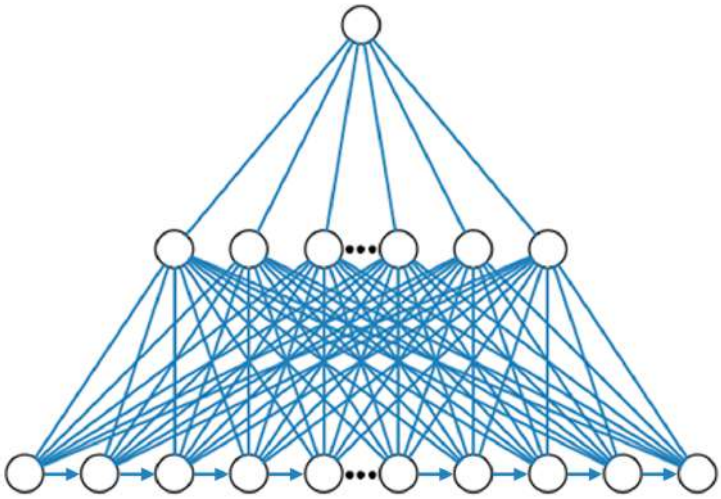

Figure 4.5: LSTM Neural Network architecture

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(32, input_shape=(15,
    inputs_train.shape[-1])))
model.add(tf.keras.layers.Dense(128, activation="relu"))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))
model.compile(optimizer="adam", loss="mse", metrics=["acc"])
```

Figure 4.6: LSTM TensorFlow implementation

## 4.4 The Experimental Dataset

To train our model we needed a dataset containing proximity data that included RSSI values, timestamps and the grand-truth, to let the model learn how to distinguish a BLE Tag in the proximity against one that is not. For this purpose we created a Dataset with 23987 samples. The capturing of those samples has been done with a fully charge BLE tag, an ESP32 connected to a powerbank and two actors, one had the BLE tag in the hand, while the other had the ESP32 in the hand. We covered all the scenarios shown in Figure 4.6 and for each interaction in the figure we replicated that position for the class Proximity and Non-proximity to have a balanced dataset. The sampling of each position was around 2 minutes, 1 minute for class. We wanted to have samples from all these positions because of the signal attenuation that a body can make standing between the emitter and the receiver.

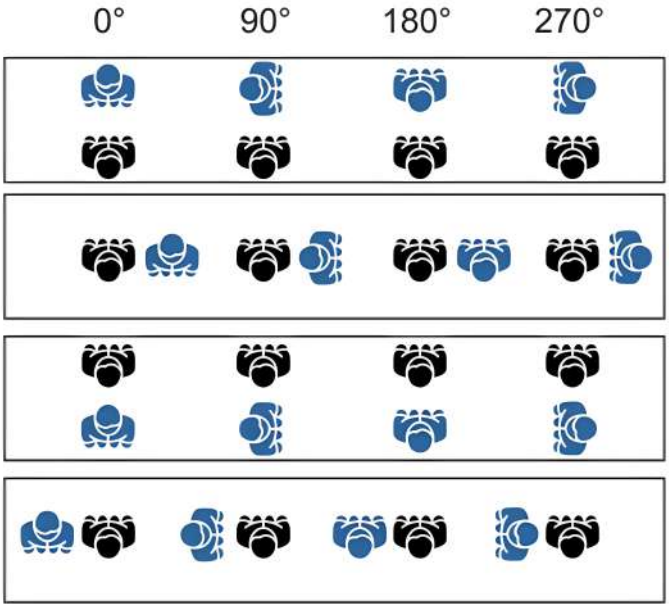


Figure 4.7: Position of BLE beacons capture.
Black microcontroller, Blue emitter

The firmware used in the ESP32 was a sketch that used the NimBLE library with a BLEAdvertisedDeviceCallbacks class that in the function onResult would append to a local file UUID, RSSI, milliseconds from boot and grand truth of that advertisement. The grand truth variable value was influenced by a ESP32 GPIO, if a jumper was connected the grand truth would be True (1), otherwise False (0).

The dataset has the following features recorded:

- UUID: tag identifier
- rssi: power of the tag signal in dB
- millis: time from ESP32 boot in ms
- label: grand truth of the proximity (0 if we were far, 1 if we were nearby)

Table 4.1 reports an example of the collected dataset. For each sampling the UUID column reports the beacon identifier, the RSSI column reports the strength of the signal in dBm, the millis column reports the micro-controller up-time in milliseconds and the label column reports a boolean value with the grand truth, 1 when the beacon is in proximity, 0 when it is not.

| UUID | rssi | millis | label |
|---|---|---|---|
| ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63 | -87 | 611 | 0 |
| ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63 | -88 | 1125 | 0 |
| ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63 | -81 | 2130 | 0 |
| ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63 | -86 | 3137 | 0 |
| ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63 | -86 | 4152 | 0 |

Table 4.1: Example of collected Dataset

Table 4.2 reports a summary of the values in the Dataset and the acquiring conditions.

| | |
|---|---|
| Number of Samples | 23987 |
| Samples with label 0 | 11902 |
| Samples with label 1 | 12085 |
| Orientations | 9 |
| BLE Tags | 2 |
| Micro-controllers | 1 |

Table 4.2: Summary of the Dataset

We wanted to make sure that our dataset values were correct and without errors, so we looked for outliers and checked the correlation between the attributes. To

detect outliers we examined the numeric attributes using the Interquartile range method. For each of these attributes, we defined a range with lower bound and upper bound, and we considered as outliers all the attributes that are outside this range. Then we plotted the results exploiting a Barplot chart.

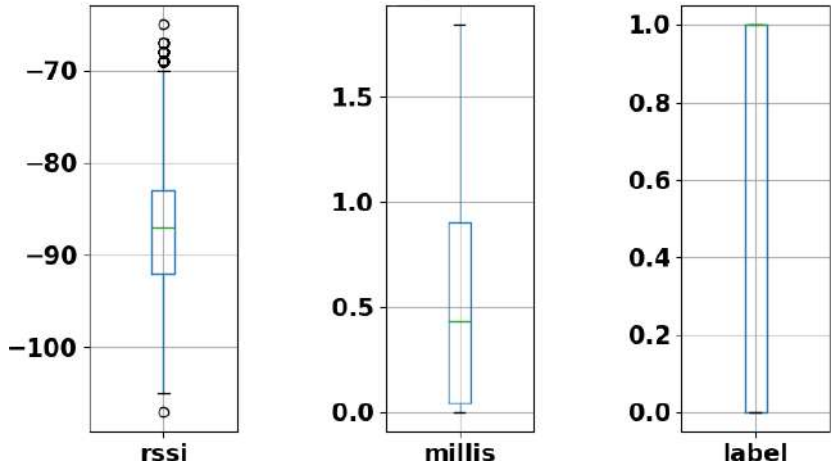


Figure 4.8: Outlier Boxplots

From Figure 4.7 we can see that the only values that are marked as outliers are some RSSI value that are around -70 dBm. We decided to keep these values because they are valid measurement inside the operation interval of our antenna, just not so common in the dataset.

Then we proceeded to calculate the Pairwise correlation between the attributes to make sure that just the RSSI and Label columns are strongly correlated.

| | rssi | millis | label |
|---|---|---|---|
| rssi | 1.000000 | 0.257230 | 0.576709 |
| millis | 0.257230 | 1.000000 | 0.395373 |
| label | 0.576709 | 0.395373 | 1.000000 |

Table 4.3: Pairwise Correlation matrix

From these results we can see that between RSSI and Label there is a strong positive correlation, while the other fields have a lower value, so there is no strong statistical relationship between them, as we expected.

## 4.5 Experimental Results

### 4.5.1 Fully Connected

To train our model we shuffled and split the dataset in three parts: Train Set, Validation Set and Test Set. The first two sets were given in input to the train command, while the last one will be used for verifying our results. Having three distinct and shuffled sets is very important for the training of a neural network, because in this way we can avoid that it learns the pattern we used to create the dataset and we can do a final test to evaluate the neural network performances with values that has not been processed during the training phase. During this last testing step we will evaluate the Precision and the Recall for each of our labels. The Precision is the number of true correct results in a given class divided by the number of all the observations of that class, while the recall is the number of correct results of each class divided by the actual number of object in that class. With precision we can make sure that what we identify as proximity is actually a proximity, while with recall we can make sure to not miss out other positive observations.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{4.5}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{4.6}$$

We preceded training the model with 500 epochs and we can see the training history of the Loss value in Figure 4.7 and the Accuracy value in Figure 4.8 both for the Training Set and the Validation Set.

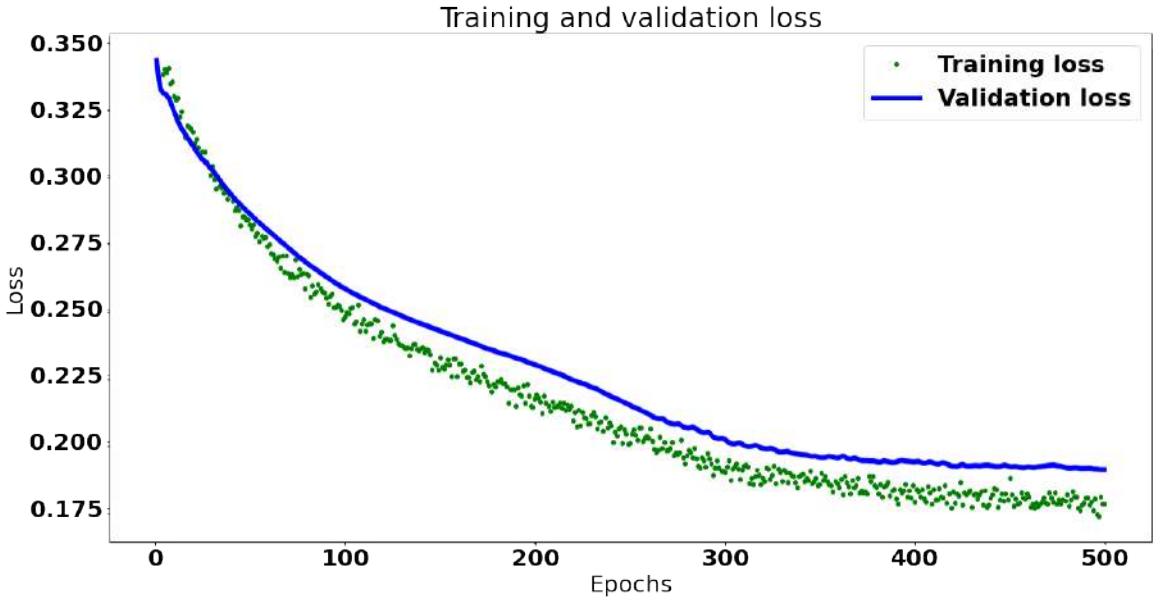


Figure 4.9: Training and Validation Loss during Dense Training

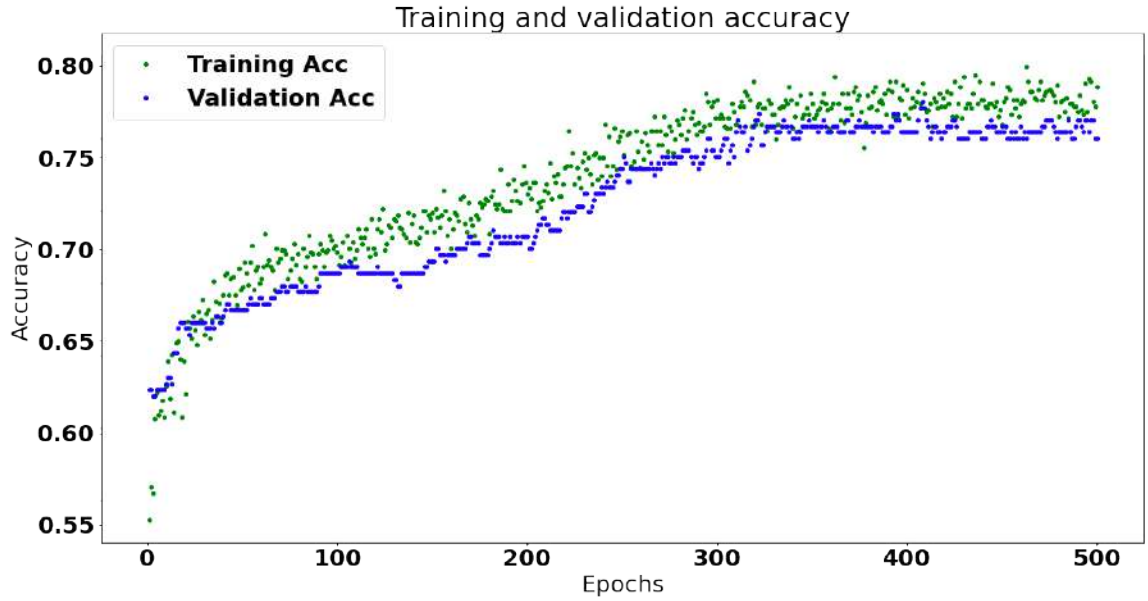


Figure 4.10: Training and Validation Accuracy during Dense Training

We also tried to inference the values of our Test Set and calculated some metrics with the classification report method given by Scikit-learn[33], shown in Table 4.4. We obtained good results both for precision and recall.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Non-Proximity | 0.80 | 0.72 | 0.77 | 103 |
| Proximity | 0.80 | 0.89 | 0.84 | 131 |
| accuracy | | | 0.81 | 234 |
| macro avg | 0.82 | 0.80 | 0.81 | 234 |
| weighted avg | 0.81 | 0.81 | 0.81 | 234 |

Table 4.4: Classification Report of the Fully Connected Neural Network

### 4.5.2 LSTM

While before the training of the Fully Connected neural network we shuffled the data, this time we didn't do it, in this way the temporal correlation between the samples is kept and so also the patterns and events represented by the series of values in our dataset. We split the dataset in three parts: Train Set, Validation Set and Test Set. The first two sets were given in input to the train command, while the last one will be used for verifying our results. We trained the model with 400 epochs.

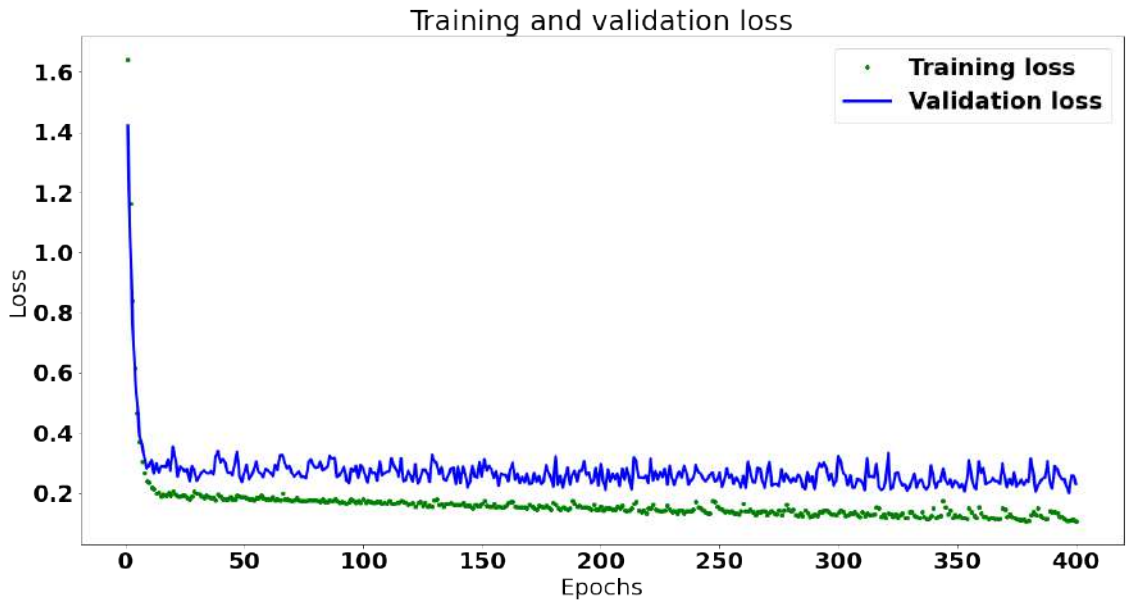


Figure 4.11: Training and Validation Loss during LSTM Training

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0.0 | 0.85 | 0.82 | 0.83 | 582 |
| 1.0 | 0.84 | 0.86 | 0.85 | 627 |
| accuracy | | | 0.84 | 1209 |
| macro avg | 0.84 | 0.84 | 0.84 | 1209 |
| weighted avg | 0.84 | 0.84 | 0.84 | 1209 |

Table 4.5: Classification Report of the LSTM based Neural Network

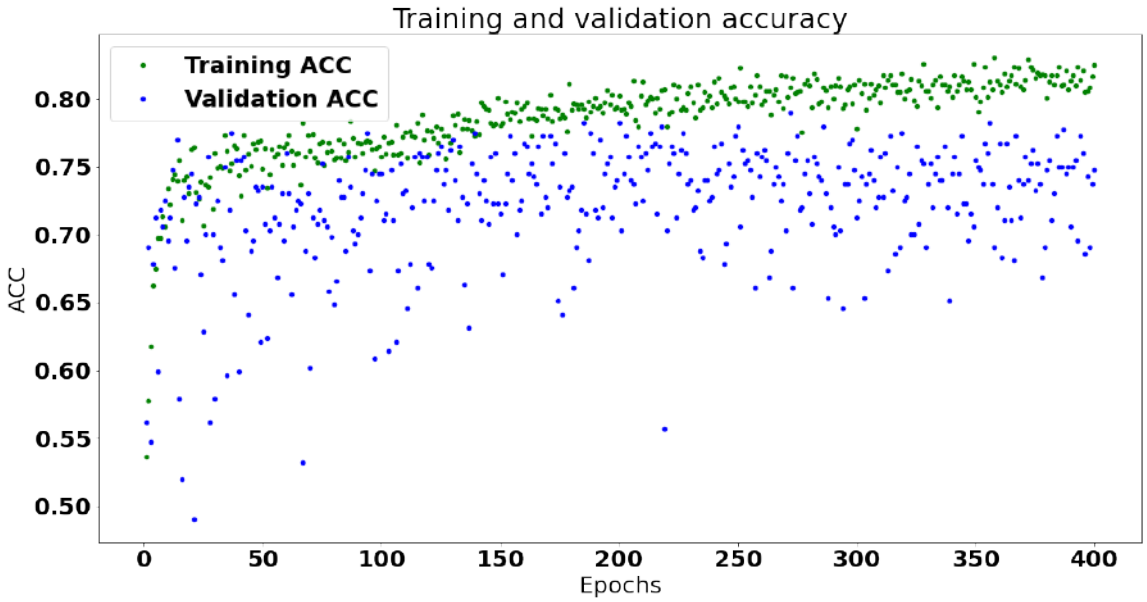


Figure 4.12: Training and Validation Accuracy during LSTM Training

We also tried to inference the values of our Test Set and calculated some metrics with the classification report method given by Scikit-learn, shown in Table 4.5. The results have a good precision and recall, higher then the results of the Fully Connected neural network.

### 4.5.3 Discussion

LSTM networks are structurally more complex, but this complexity doesn't penalize the inference time, that also on on the ESP32 board takes 1 millisecond or less (Figure 4.13 and Figure 4.14). On the other hand the Fully Connected network would require more computation before starting the inference, because it have to compute

all the aggregation functions described in Section 4.2 (i.e. average, standard deviation, kurtosis, ...), while the LSTM requires just the raw list of RSSIs acquired. The training phase is faster on the Fully Connected network(13 seconds vs 6 minutes), but it's a one time procedure and it's a waiting that will not affect the regular user. We ended up picking the LSTM network for its higher overall accuracy and balance of precision and recall for both proximity classes. This confirmed that Recurrent Neural Networks, especially LSTM, can learn patterns of values over time. We focused more on this in the Future Works section, where we spoke about improvements and new feature that could be added.

```
Ready to compute!
Processing beacons...
Map Size: 3
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63, predicted: 0.585322 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e65, predicted: 0.280134 (Inference time: 0 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e69, predicted: 0.981029 (Inference time: 1 ms)
Processing beacons...
Map Size: 3
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63, predicted: 0.221472 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e65, predicted: 0.248018 (Inference time: 0 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e69, predicted: 0.38141 (Inference time: 1 ms)
Processing beacons...
Map Size: 3
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63, predicted: 0.699971 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e65, predicted: 0.379878 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e69, predicted: 0.876222 (Inference time: 0 ms)
```

Figure 4.13: Inference time for Fully Connected network on ESP32

```
Ready to compute!
Processing beacons...
Map Size: 3
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63, predicted: 0.236376 (Inference time: 0 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e65, predicted: 0.770004 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e69, predicted: 0.961041 (Inference time: 1 ms)
Processing beacons...
Map Size: 3
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63, predicted: 0.910533 (Inference time: 0 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e65, predicted: 0.679916 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e69, predicted: 0.602362 (Inference time: 0 ms)
Processing beacons...
Map Size: 3
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e63, predicted: 0.276061 (Inference time: 0 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e65, predicted: 0.300012 (Inference time: 1 ms)
UUID: ef977fc0-20c6-4e1c-9141-f6dd2eeb7e69, predicted: 0.770468 (Inference time: 1 ms)
```

Figure 4.14: Inference time for LSTM based network on ESP32

# Chapter 5

# Detecting Proximity with the BLEind application: A case Study

In this Chapter we describe how we used all the finding and development made in Chapter 3 and 4 to create a real use case scenario that could help visually impaired people navigating outdoor. Ordinary actions that we do everyday outside, like crossing a street or entering the right train coach, can be very tough for them, so we wanted to help them breaking down the barriers that texts, lights and colors create for them.

## 5.1 Use-Case Description

The exploring and development done in Chapter 1,2,3 and 4 highlighted a clear use-case that has its focus on accessibility. We wanted to develop a mobile application for smartphones that could help blind or visually impaired people in their daily life exploiting the Neural Network developed in Chapter 4 running on a commercially available device evaluated in Chapter 3, namely the ESP32.
The main requirements for this application were: Screen Reader support, multi-channel feedbacks and the use of commercial tags.

#### Screen Reader

Screen reader is a feature that is present both on iOS and Android. On Android we can find it under "TalkBack", while on iOS under "VoiceOver". It allows a blind person to use a touch screen device reading what is present on the display and the possible ways of interaction available.

#### Multi-channel Feedback

An alert generated by the application could be a critical information that the user should get. To avoid missing information we want that the app is capable of reading alert via Text-to-Speech both with the smartphone speaker and with headsets, but also via vibration patterns.

#### Commercial Tags

We wanted our application to be as flexible as possible in terms of integration with the real word. Sp we wanted to use standard commercial BLE Tags that can run with a single coin-cell battery for long time-frames, don't require wiring or solar panels and are easily available to a consumer. This last part is critical for the widespread distribution in private building, such as offices or supermarkets.

## 5.2 The BLEInd application

BLEind is a mobile application developed in Flutter, compatible with both Android and iOS. BLEind, once paired with the microcontroller running the neural network developed in Chapter 4, can detect the beacons emitted by different Point of Interests and send alerts via headphones, speaker or vibration. BLEind notifies you when a person is in proximity of a smart POI and can also retrieve their state. For example state could be the light color for a Traffic Light or the number of train coach we are approaching. This would solve for example the problem of visually impaired people walking in the city center at night, when sometimes traffic light's speakers are disabled to avoid loud noises, because with this app they could know the state of the lights without any loud speakers.

### 5.2.1 UI and Accessibility

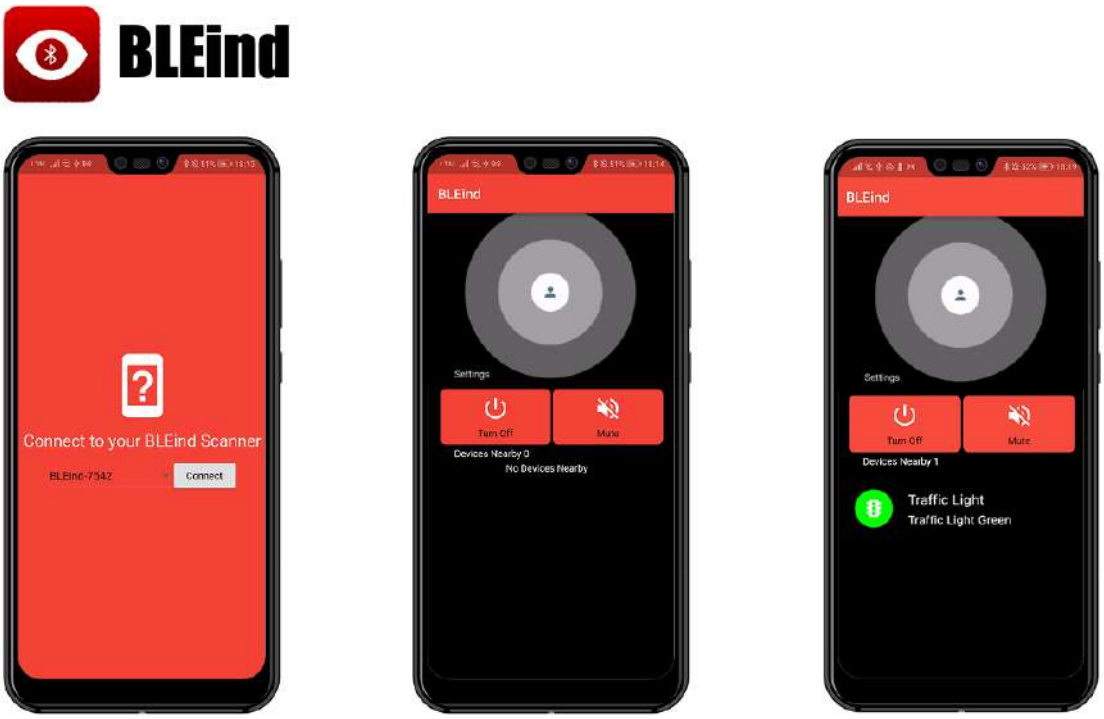


Figure 5.1: BLEind user interfaces

The User Interface has been developed with high-contrast color, big icons and full support for Screen Readers. In this way blind people can navigate the app using their smartphone built-in screen-reader, while visually impaired people can recognize the different buttons for each action thanks to the high contrast and big icons.

All the alerts are emitted both via audio (speaker or headphone) and vibrations. This second method is really important for busy situation in which an audio message could be missed.

The app also has a fast repeat alert function that can be activated by simply shaking the phone.

### 5.2.2 Communication Protocol

The main protocols explored for the communication between our IoT device and a Smartphone are:

- WiFi with WebSockets
- Bluetooth Classic with Bluetooth Serial

Both these two ways are fully supported with libraries to integrate them easily in Flutter. To decide which is the best option we are going to analyze mainly the power consumption.

#### WiFi with WebSocket

This communication protocol require that the smartphone is connected to a WiFi Hotspot created by the ESP32 microcontroller, then all the communication are taken in place via the WebSocket protocol.

WebSocket is a protocol based on TCP that creates a full-duplex tunnel of communications between a client and a server. Once a connection is established each party can send and receive messages in real-time without any polling, that would be necessary with other TCP-based protocols, such as HTTP.

#### Bluetooth Classic with Bluetooth Serial

This communication protocol require just the pairing of the smartphone with the ESP32 microcontroller.

This protocol moves the standard USB Serial Console with inputs/outputs of the Microcontroller SDK over Bluetooth. The package that we are going to use to integrate it in Flutter is called *flutter_bluetooth_serial*.

### 5.2.3 Power Consumption

The main concern we have for these communications was the power consumption that would have the user's smartphone while connected to our microcontroller. We decided to develop two version of the application and of the microcontroller firmware: one with support to WiFi WebSockets and one with support to Bluetooth Serial. We tested one hour of application usage with the same smartphone conditions:

- same geographical position
- full cellular signal strength
- connected to WiFi
- Bluetooth turned on
- 50% of display brightness
- constant proximity to a tag.

During the testing we logged the battery percent of the smartphone with an Android app available on the Google Play Store called "Battery Log" that could export a CSV file.

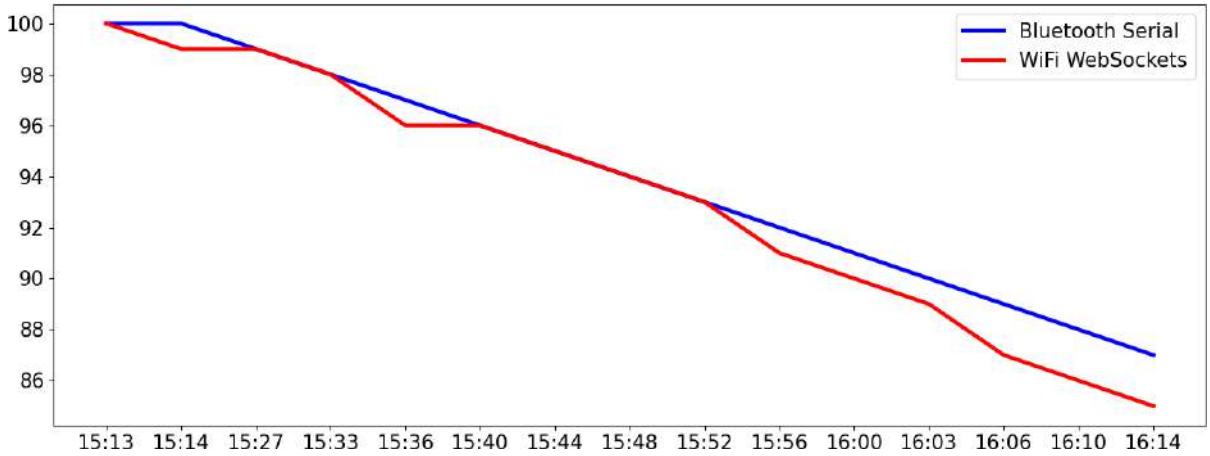


Figure 5.2: Power Consumption of WiFi and Bluetooth

As we can see from the chart, the WiFi solution tended to use more battery then the Bluetooth one. Our final choice is the Bluetooth Serial library.

## 5.3 Existing Commercial Alternatives

### 5.3.1 LetiSmart Voce

A similar commercial grade solution is already available from SCEN and it's called LetiSmart Voce. This product use a LoRa wireless module to detect nearby LetiSmart Tags and it's built as an extension of a standard white cane. On the top has an USB-C port to recharge its battery and an 3,5mm Audio Jack to connect an headset and get the warnings.

BLEind could achieve this while being more flexible with Over-The-Air updates and more accessibility feature such as vibration. Also all the LetiSmart Tags require a 12v power source, while BLEind Tags just need a Coin cell battery and they can last for an entire year.

# Chapter 6

# Conclusions and Future Works

In this thesis work we approached the usage of Neural Networks on embedded devices and an integration between them and a smartphone Companion App. The goal was to further explore this research field and get a first working BLE proximity detector exploiting iBeacon protocol. After a study on what embedded device would fit the best our use case, it came out that the most reliable device is the EspressIf ESP32, with lower power consumption and a way more stable BLE beacon reception rate. In order to train our future neural network we created a dataset with over 23000 BLE beacon samples classified as Nearby or Far. To build this we used two actor and placed them in 9 positions (Figure 4.6) for fixed amount of time to be sure that our dataset included the case in which the signal is attenuated by the body of a person. On the ESP32 we explored two different types of Neural Networks running with TensorFlow Micro library. The first network was a feed forward fully connected neural network based on TensorFlow's Dense layers, while the second one was based on LSTM layers. The network that showed better results was the second one, that achieved an average precision and recall of 0.84. After having a working IoT device with the ability to detect BLE tags in the proximity we started developing a companion app for smartphones to interact with it and built an accessibility tool around this explorations. The application was developed in Dart with the framework Flutter and for the communications between the IoT device and the Smartphone we picked Bluetooth Serial. This communication method allowed us to have a good stability of the connection, good battery life, both of the IoT receiver and of the smartphone, and an easy implementation with Flutter.

Although our proximity neural network works well it would be a nice addition to add new accessibility features to the application, such as People Awareness. With this feature the application would be able to help the blind user to locate the posi-

tions of people in room analyzing the trends that the RSSI signal has. This could be done with different methods.
This first is exploiting more deeply our recurrent neural network, acquiring another ad-hoc dataset in which are present different degrees of classification, for example we can start by classifying Nord, Sud, East, West as the BLE position. This would allow us to maintain the same hardware and just put a new trained neural network on the microcontroller.
The second option is integrating new sensors on the microcontroller and feed the neural network with additional information such as inertia, stereo microphones or a compass.
The last option is to upgrade to Bluetooth 5.0 to exploit a new feature introduced called Direction Finding[4]. This feature would allow the microcontroller to locate BLE devices with a precision of few centimeters. This is made possible by an array of antennas on the chipset that can estimate the Angle of Arrial (AoA) of the signal by analyzing the phase difference of the incoming signal.

To further help blind people there are more interesting features that could be added to the application, are an SOS Emergency function and a live remote tracking. These function could help a person that lost their way get help, both that he noticed it (SOS) or not (Live tracking).
The SOS Emergency function could be implemented in the BLEind app through a combination of physical buttons, a drawing on the screen or a hardware button placed on the wearable that sends the command via Bluetooth to the application, that sends an SMS with the coordinates of the person.
The Live Tracking enable a tutor or a nursing home to monitor remotly a person to be sure that he is not losing their way and take preventive actions. This feature would require an internet connection and a server that is able to store the history of locations recieved and show them to the allowed operators. It could be implemented both via a WebSocket tunnel or multiple temporized HTTP requests.
Implementing these features in the app we can avoid the additional GSM and GPS module for the ESP32 and a subscription for the additional cellular connectivity, because we can simply use the mobile plan and GPS that the person has on their smartphone to send sms, connect to the internet or retrieve their location.

# Bibliography


[1] Altbeacon. https://altbeacon.org.

[2] Apple ibeacon. https://developer.apple.com/ibeacon/.

[3] Arduino 33 specifications. https://store-usa.arduino.cc/products/arduino-nano-33-ble-sense.

[4] Bluetooth 5.0 direction finding. https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/direction-finding/.

[5] The bluetooth® low energy primer. https://www.bluetooth.com/wp-content/uploads/2022/05/Bluetooth_LE_Primer_Paper.pdf.

[6] Eddystone. https://github.com/google/eddystone.

[7] flutter_bluetooth_serial plugin. https://pub.dev/packages/flutter_bluetooth_serial.

[8] Immuni documentation. https://github.com/immuni-app/immuni-documentation.

[9] Modified ble library for arduino. https://github.com/wnlab-isti/ble-arduino/tree/main/ble-library.

[10] Nimble-arduino by h2zero. https://github.com/h2zero/NimBLE-Arduino.

[11] Official arduinoble library. https://www.arduino.cc/reference/en/libraries/arduinoble/.

[12] Skybeacon app. https://github.com/wnlab-isti/ble-arduino/tree/main/SKYBEACON.apk.

[13] Tensorflow lite for microcontrollers. https://www.tensorflow.org/lite/microcontrollers.

[14] Understanding the different types of ble beacons. `https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/`.

[15] What is bluetooth low energy (ble)? how does ble work? `https://litum.com/blog/what-is-ble-how-does-ble-work/`.

[16] Work with websockets. `https://docs.flutter.dev/cookbook/networking/web-sockets`.

[17] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[18] Alexandre Alapetite, John Paulin Hansen, John H. L. Hansen, and John Paulin Hansen. Dynamic bluetooth beacons for people with disabilities. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016.

[19] Paolo Baronti, Michele Girolami, Fabio Mavilia, Filippo Palumbo, and Giancarlo Luisetto. On the analysis of human posture for detecting social interactions with wearable devices. *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*, 2020.

[20] Paolo Barsocchi, Paolo Barsocchi, Michele Girolami, and Davide La Rosa. Detecting proximity with bluetooth low energy beacons for cultural heritage. *Sensors*, 2021.

[21] N. Brown. Edward t. hall, proxemic theory, 1966, 2001.

[22] Teja Chava, A. Tarak Srinivas, A. Lohith Sai, and Venubabu Rachapudi. Iot based smart shoe for the blind. *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, 2021.

[23] François Chollet. *Deep Learning with Python.* Manning Pubns Co, 1st edition, 2018.

[24] Daniele Croce, Laura Giarre, Federica Pascucci, Ilenia Tinnirello, Giovanni Galioto, Domenico Garlisi, and Alice Lo Valvo. An indoor and outdoor navigation system for visually impaired people. *IEEE Access*, 2019.

[25] P. C. Deepesh, Rashmita Rath, Akshay Tiwary, Vikram Nagaraja Rao, and N. Kanakalata. Experiences with using ibeacons for indoor positioning. *ISEC*, 2016.

[26] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. `http://www.rfc-editor.org/rfc/rfc6455.txt`.

[27] Michele Girolami, Fabio Mavilia, and Franca Delmastro. Sensing social interactions through ble beacons and commercial mobile devices. *Pervasive and Mobile Computing*, 2020.

[28] Jee-Eun Kim, Masahiro Bessho, Shinsuke Kobayashi, Noboru Koshizuka, and Ken Sakamura. Navigating visually impaired travelers in a large train station using smartphone and bluetooth low energy. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, page 604–611, New York, NY, USA, 2016. Association for Computing Machinery.

[29] Jee-Eun Kim, Masahiro Bessho, Noboru Koshizuka, and Ken Sakamura. Mobile applications for assisting mobility for the visually impaired using iot infrastructure. In *Proceedings of 2014 TRON Symposium (TRONSHOW)*, 2014.

[30] R. Santhana Krishnan, K. Lakshmi Narayanan, S. Mathumitha Murali, A. Sangeetha, C. Ramasamy Sankar Ram, C. Ramasamy Sankar Ram, and Y. Harold Robinson. Iot based blind people monitoring system for visually impaired care homes. *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, 2021.

[31] Andrew Mackey, Petros Spachos, Liang Song, and Konstantinos N. Plataniotis. Improving ble beacon proximity estimation accuracy through bayesian filtering. *IEEE Internet of Things Journal*, 7(4):3160–3169, 2020.

[32] Anna Fay E Naive, Anna Fay E Naive lowast, Paul Joseph M Estrera, and Archie O Pachica. Design and implementation of faculty class attendance monitoring system using ble beacons. 2020.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Zeeshan Saquib, Vishakha Murari, and Suhas N Bhargav. Blindar: An invisible eye for the blind people making life easy for the blind with internet of things (iot). *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information  Communication Technology (RTEICT)*, 2017.

[35] Seyed, Craig Greenberg, and Douglas Olson. Nist pilot too close for too long (tc4tl) challenge evaluation plan, 2020-06-18 2020.