A TinyML-Approach to Detect the Proximity of People Based on Bluetooth Low Energy Beacons

Michele Girolami
Italian National Council of Research ISTI-CNR
Pisa, Italy
michele.girolami@isti.cnr.it

Francesco Fattori
Department of Computer Science, Pisa University of Pisa
Italy, Pisa
f.fattori4@studenti.unipi.it

Stefano Chessa
Department of Computer Science, Pisa University of Pisa and ISTI-CNR
Italy, Pisa
stefano.chessa@unipi.it

***Abstract*—Proximity detection is the process of estimating the *closeness* between a target and a point of interest, and it can be estimated with different technologies and techniques. In this paper we focus on how detecting proximity between people with a TinyML-based approach. We analyze RSS values (Received Signal Strength) estimated by a micro-controller and propagated by Bluetooth's tags. To this purpose, we collect a dataset of Bluetooth RSS signals by considering different postures of the involved people. The dataset is adopted to train and test two neural networks: a fully-connected and an LSTM model that we compress to be executed directly on-board of the micro-controller. Experimental results conducted over the dataset show an average precision and recall metrics of 0.8 with both of the models, and with an inference time less than 1 ms.**

***Index Terms*—Proximity, TinyML, Deep Learning, Arduino**

I. Introduction

Indoor localization techniques have been dramatically increasing their accuracy and effectiveness during the last ten years [1]. The advances of IoT devices combined with learning-based algorithms have been enabling the possibility of estimating the position of a target indoor with high precision. Under this respect, we focus on a specific problem of indoor localization, namely proximity detection. With the term proximity we refer to the possibility of detecting when a target, ie. a person or an object, is *close* to a point of interest (POI). Many technologies and can be adopted to detect proximity [2], ranging from video-based to radio frequency (RF) technologies. In particular, the family of RF technologies represents one of the most promising alternative, as many commercial devices are already equipped with wireless interfaces exploitable for proximity detection. This is the case of micro-controllers, such as Arduino boards which offer a system-on-chip provisioned with a Wi-Fi and Bluetooth interfaces, and an AI support for machine learning.

Although proximity detection is with Bluetooth signals is widely studied [3], [4], most of the existing results are based on high-performing receiving devices, such as smartphone (commonly adopted for contact tracing applications) or general-purpose devices, such as Raspberry PI boards. Furthermore, only few works address the problem of detecting proximity between people by varying the layout of the interacting subjects. Indeed, often face-to-face interactions are studied by simplifying the testing scenario. Under this respect, we mention the Too Close for Too Long challenge, organized to test real-world solutions for humans proximity based on ML models. Such challenge provides a collection of Bluetooth datasets useful to test the solutions at different conditions [1].

In this work we propose a solution that exploits a learning-based approach and based on Bluetooth RSS analysis and neural networks. In the past, we already tackled the problem of proximity detection but with a very different approaches: smartphone-based [5] and Raspberry PI-based [6]. Differently from the past, we adopt a TinyML approach (Tiny Machine Learning). Indeed, we argue that detection proximity among two people instead that with POI is further complicated by the fact that both people may take different postures and orientations. More specifically, we design and test two neural network models able to classify proximity from non-proximity events. Models are based on the analysis of features extracted from RSS values of the collected beacons. All the computation is executed on board of the micro-controllers. The implemented models are: a fully-connected multi-layer neural network and a LSTM (Long short-term model) model. In both of the cases, we train the models off-line and then we convert them as TensorFlow Lite models. To enable this approach, we produce a realistic and annotated dataset that reproduces proximity and non-proximity events between two subjects. We reproduce proximity by varying the orientation of the subjects: 4 orientations (North, East, West and South) and for each orientation we test 4 rotations shifted of 90°each. The dataset also comprises the ground truth annotation, providing the actual start and end time of each of the events between the subjects. From our experiments, we observe that the LSTM model outperforms the fully-connected model with an average precision and recall metric of 0.84. We also test the models on two commercially available micro-controllers, namely the ExpressIf ESP 32 and Arduino BLE 33 Sense, both supporting the execution of ML models.

[1]Datasets are available at this link: https://www.nist.gov/document/nist-pilot-tc4tl-challenge-evaluation-plan

II. Learning Models for Proximity Detection

To the purpose of detecting proximity between people, we exploit the Bluetooth's RSS values estimated by a receiving device and emitted by Bluetooth tags. RSS indicates the power level that the receiving device estimates for a wireless message. It is regulated by the IEEE 802.11 standard, but each vendor can adjust the range, making comparison between different receiver's chipsets a complex task. RSS varies according to the power transmission of the emitter, the sensitivity of the receiving antenna, and also with the environment in which the emitter and receiver are located. A Bluetooth tag propagates beacon messages, while the receiver collects and analyzes the corresponding RSS values. At static conditions, the closer an emitter to a receiver, the higher the RSS estimated at the receiver side. Such relationship has been widely investigated with several propagation models. The path loss model represents a reference model for wireless propagation[7], according to which the relationship between RSS and distance is given by: $RSS = RSS_0 - 10nlog_{10}(d/d_0), \quad d > d_0$, where d_0 is the reference distance, such that the emitter and the receiver are always in line of sight (typically 1 m), RSS_0 is the RSS at a reference distance d_0, and n is the path loss exponent that regulates how severe is the attenuation in a given environment. Figure 1 shows RSSI (Received Signal Strength Indicator) fluctuations during a proximity event. The red line reports the proximity's ground truth, while the blue dots denote the RSS values estimated by the receiving device. The figure shows a realistic scenario in which RSS values vary in a considerable range (e.g., $-100dBm$ to $-70dBm$).

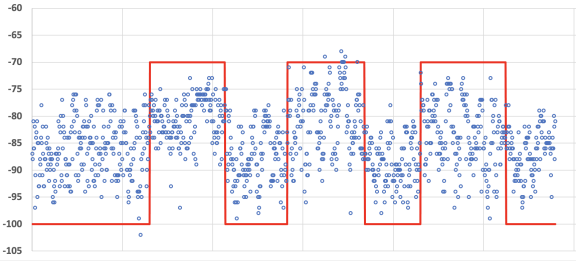


Fig. 1: RSSI fluctuations during a real-world proximity event.

A. The Neural Network Models

The **first** neural network we design is based on Tensorflow's Dense layers. We create a feed-forward fully connected network. We decide to analyze time windows T_k of k seconds, and we extract some RSS's features as input for our network, as reported in the following:

- AVG: average RSSI value of n samples: $\frac{1}{n}\sum_{i=i}^{n} x_i$
- STD: standard deviation of the n values: $\sqrt{\frac{1}{n}\sum_{i=i}^{n}(x_i - \bar{x})^2}$
- MAX: maximum observed RSSI value;
- MIN: minimum observed RSSI value;
- SKW: Skewness of the RSSI values: $\frac{1}{n}\sum_{i=1}^{n}\frac{(x_i-\bar{x})^3}{\sigma^3}$;
- KURT: Kurtosis of the RSSI values: $\frac{1}{n}\sum_{i=1}^{n}\frac{(x_i-\bar{x})^4}{\sigma^4}$;

Standard Deviation measures the dispersion of RSS values, Skewness measures the symmetry of our distribution of them, while Kurtosis measures if values are heavy-tailed or light-tailed in respect of a normal distribution. These three metrics are important for feeding to our network a representation of the RSSI pattern during a time window T_k. All the values are in dBm unit normalized in the range $[0, 1]$. Observed RSS values lower than $120dBm$ can be considered outliers samples. The normalization process is very important to allow the neural network to quickly terminate, and to avoid triggering large gradient updates that sparse and double digit numbers. The design of the neural network consists of 3 Dense layers connected with 2 Dropout layers, to avoid overfitting, and an L2 regularizer applied with a coefficient of 0.001. The first 2 Dense layers have `ReLU` as activation function, with 128 and 64 neurons, respectively. The last Dense layer implements a `sigmoid` activation function, with a single neuron in order to classify the two relevant events: proximity and non-proximity. The network has been compiled with `adam` optimizer and the loss function is calculated based on the Mean Square Error (mse), while monitoring also the accuracy metric.

The **second** neural network is based on the LSTM model. LSTM stands for Long Short-Term Memory and it belongs to the family of Recurrent Neural Networks (RNNs). In this case, nodes of a layer can be connected with them self by 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 timestamps. This model is created to solve the problem of vanishing gradients, a common problem of simple RNNs models. The problem consists of forgetting the time dependency across a long time sequence adding complexity to the training, giving rise to low performance results. To solve this problem, LSTM models implement Gates, managing the recurrent signal and giving the possibility to change the focus between present and past dynamically, by changing the weight on them. We decide to analyze time windows T_k of k seconds and to feed them the corresponding values as input to the LSTM. Our neural network has 1 LSTM layer made of by 32 neurons followed by 2 Dense layers. The first Dense layer consists of 128 neurons with a `ReLU` activation function, while the second Dense layer implements a single neuron with the `sigmoid` activation function, to output a binary classification: proximity and non-proximity. The neural network has been compiled with `adam` optimizer and the loss function is calculated based on the Mean Square Error while monitoring also the accuracy metric.

III. Experimental Settings

We adopt two of the most popular and inexpensive microcontrollers supporting the BLE network interface and ML

models: Arduino 33 Sense and the ExpressIf ESP32. 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, 256 KB of RAM and 1MB of program memory that are capable of running heavier and bigger firmware. It also has a built-in modem that supports both Wi-Fi, Bluetooth and NFC. The board also includes various environment sensors. The ESP32 is a low-cost system on a chip developed by the company ExpressIf, equipped with a 32-bit Xtensa LX6 processor with two cores that run at 240MHz, a built-in Wi-Fi and a Bluetooth module and antennas. Concerning the Bluetooth tags, we adopt the GlobalTag Bluetooth tags. They have a reduce dimension and reduced power consumption. Tags can be configured to emit beacons with different advertisement rates and with different power of emissions, as shown in Figure 2.

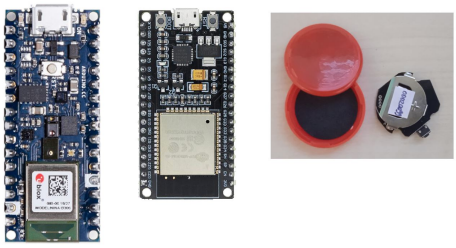


Fig. 2: The adopted hardware: *left*: Arduino BLE Sense, *center*: ExpressIf ESP32, *right*: Bluetooth tags.

A. Data Collection Process

To test the models described in Section II, we organize a data collection campaign. The dataset is collected with the ESP32 micro-controllers, with following information: the tag's identifier (Bluetooth MAC address), the RSS value, the timestamp and the ground truth label: `proximity (0)` and `non-proximity (1)`. To this purpose, we recruit two subjects reproducing a number of proximity and non-proximity events. We vary the orientation of subjects (North, East, West and South) and their relative rotations (0°, 90°, 270°and 360°). In particular, for each of the 4 orientations we test 4 rotations for a total of 16 layouts as reported in Figure 3. Each layout has been repeated 4 times, so that to have a balanced number of proximity and non-proximity events. Subjects start with a non-proximity event at $5m$ distance for 2 minutes, then they move in proximity at $1.5m$ distance for 2 minutes.

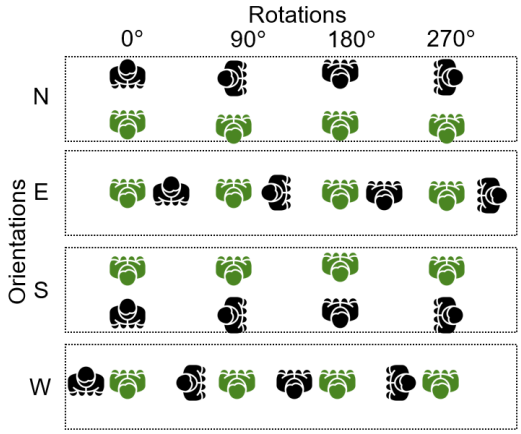
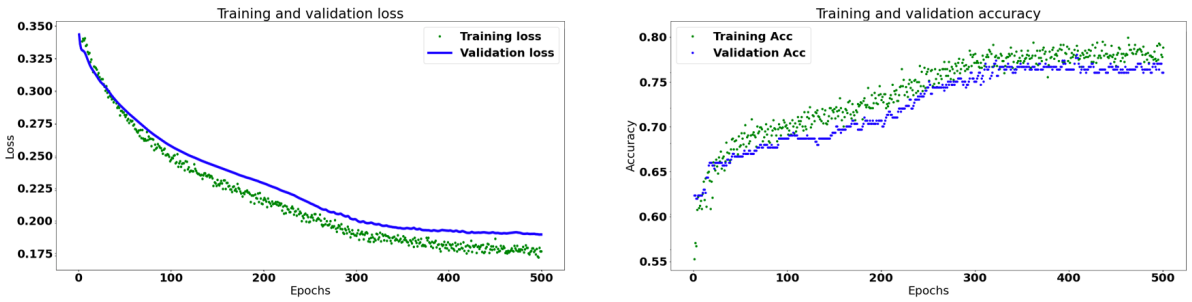


Fig. 3: The testing layouts used for the data collection campaign.

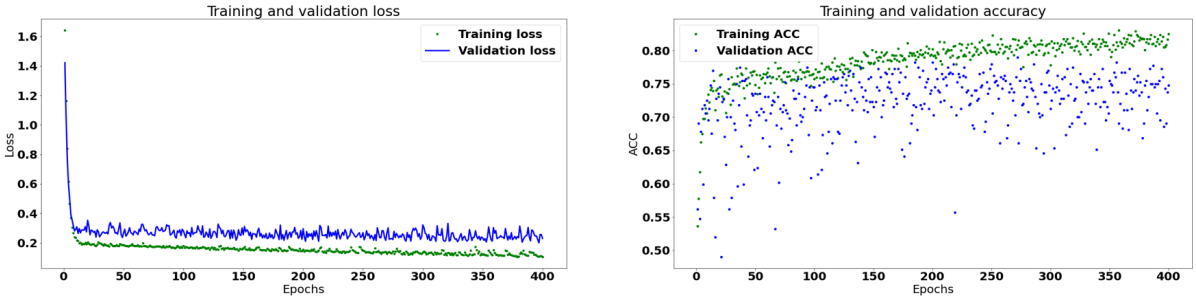
B. Performance Evaluation

In order to test our models, we compute precision and recall metrics to measure the classification task. The precision corresponds to the number of true correct answers in a given class, divided by the number observations of that class. The recall metric measures the number of correct answers of each class divided by the actual number of object in that class. With the precision, we can make sure that what we identify as proximity event, is actually a proximity event., Differently, with the recall metric we can make sure to not miss out other positive observations. Concerning the fully-connected neural network (see Section II), we shuffle and split the dataset in three parts: Train Set, Validation Set and Test Set. The first two sets are given as input to the train command, while the last dataset is used for measuring the performance of the models. Having three distinct and shuffled sets is very important for the training of a neural network, as we can avoid that the network learns the pattern we used to create the dataset and we can perform a final test to evaluate the neural network performances with never seen-before samples. We proceed to the training the model with 500 epochs, and we report in Figure 4a the the Loss and the Accuracy values, both for the training and validation sets. As expected, by increasing the number of epochs, the loss function decreases, while the accuracy score increases. We execute the model with the test set, the results are reported in Table I. We measure an average Precision of 0.8 and Recall metric of 0.8. Concerning the LSTM model, we do not shuffle the training set to preserve the temporal correlation of samples. Similarly to the fully-connected model, we split the dataset in three parts: Train Set, Validation Set and Test Set. We train the model with 400 epochs and the results of the Loss and Accuracy values are reported in Figure 4b, while results with the test set are reported in Table I, with an average Precision and Recall metric of 0.84.

The LSTM model is structurally more complex with respect to the fully-connected model, but this complexity does not penalize the inference time, that also with the



(a) Loss and accuracy values of the fully-connected model.



(b) Loss and accuracy values of the LSTM model.

Fig. 4: Performance results of the two tested models

TABLE I: Performance evaluation of the fully-connected and LSTM models executed with the ESP32 micro-controller.

		Precision	Recall	F1
Fully-connected	proximity	0.8	0.89	0.77
	non-proximity	0.8	0.72	0.77
LSTM	proximity	0.84	0.86	0.85
	non-proximity	0.85	0.82	0.83

ESP32 micro-controllers requires less than 1 ms. Moreover, as reported in Table I, the overall performance of the LSTM network are slightly higher than that of the fully-connected model. Another consideration refers to the complexity required to feed the models. More specifically, the fully-connected model requires more computation before starting the inference, as it is required to extract the list of features previously described (average, standard deviation, Kurtosis, etc). Differently, the LSTM model only requires to provide as input the raw list of RSSI values.

IV. Conclusions

The increasing computational resources available with micro-controllers allows moving part of the computation directly on board. This is the case of micro-controllers supporting the execution of ML models based on TensorFlow Lite specification. In this work, we exploit a TinyML devices to design and implement two neural networks models. They are designed to detect when people are in proximity. We also collect a representative datases which reproduces proximity and non-proximity events. The dataset is obtainbed by testing 16 different layouts (4 orientations and 4 rotations). Experimental results for both of the tested models, report a Precision and Recall metric of at least 0.8. This work represents a first step towards the potentialities of micro-controllers suitable for intelligent environments. We plan to refine the neural network models so that to estimate not only the proximity, but also the relative orientation of the subjects both at static and moving conditions.

References

[1] F. Potortì, A. Crivello, F. Palumbo, M. Girolami, and P. Barsocchi, "Trends in smartphone-based indoor localisation," in *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2021, pp. 1–7.

[2] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019.

[3] C. Martella, A. Miraglia, J. Frost, M. Cattani, and M. van Steen, "Visualizing, clustering, and predicting the behavior of museum visitors," *Pervasive and Mobile Computing*, vol. 38, pp. 430–443, 2017.

[4] N. Allurwar, B. Nawale, and S. Patel, "Beacon for proximity target marketing," *Int. J. Eng. Comput. Sci*, vol. 15, no. 5, pp. 16 359–16 364, 2016.

[5] M. Girolami, F. Mavilia, and F. Delmastro, "Sensing social interactions through ble beacons and commercial mobile devices," *Pervasive and Mobile Computing*, vol. 67, p. 101198, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119220300675

[6] P. Baronti, P. Barsocchi, S. Chessa, A. Crivello, M. Girolami, F. Mavilia, and F. Palumbo, "Remote detection of social interactions in indoor environments through bluetooth low energy beacons," *Journal of Ambient Intelligence and Smart Environments*, vol. 12, pp. 203–217, 2020, 3. [Online]. Available: https://doi.org/10.3233/AIS-200560

[7] P. Barsocchi, S. Lenzi, S. Chessa, and G. Giunta, "Virtual calibration for rssi-based indoor localization with ieee 802.15.4," in *2009 IEEE International Conference on Communications*, 2009, pp. 1–5.