PREDICTING FLOOR-LEVEL FOR 911 CALLS WITH RECURRENT NEURAL NETWORKS AND SMARTPHONE SENSOR DATA

Anonymous Authors
Paper under double-blind review

ABSTRACT

In cities with tall buildings, emergency responders need accurate floor-level location to find 911 callers quickly. We introduce a system to estimate a victim's floor-level via their mobile device's sensor data in a two-step process. First, we train a Recurrent Neural Network (RNN) using Long Short-Term Memory (LSTM) hidden units to determine when a smartphone enters or exits a building. Second, we use a barometer equipped smartphone to measure the change in barometric pressure from the entrance of the building to the victim's indoor location. Unlike impractical previous approaches, our system is the first that does not require the use of beacons, previous knowledge of the building infrastructure or knowledge of user behavior. We demonstrate real-world feasibility through 63 experiments across five different tall buildings throughout New York City. Our system predicted the correct floor-level within ± 1 floor with 91% accuracy.

1 INTRODUCTION

Indoor caller floor-level location plays a critical role during 911 emergency calls. In one use case, it can help pinpoint heart attack victims or a child calling on behalf of an incapacitated adult. In another use case, it can help locate firefighters and other emergency personnel within a tall or burning building. In cities with tall buildings, traditional methods that rely on GPS or Wi-Fi fail to provide reliable accuracy for these situations. In these emergency situations knowing the floor location of a victim can speed up the victim search by a factor proportional to the number of floors in that building. In recent years methods that rely on smartphone sensors and Wi-Fi positioning (Xue et al., 2017) have been used to formulate solutions to this problem. In this paper we introduce a system capable of delivering an estimated vertical floor location within four meters by combining deep learning with barometric pressure data obtained from a Bosch bmp280 sensor designed for "floor-level accuracy" and available in most smartphones today (Bosch, 2016) (comScore, 2017).

Our method focuses on providing the most accurate floor location estimate without relying on external sensors placed inside the building, prior knowledge of the building or user behavior. It relies simply on sensors aboard a smartphone and assumes our system could be used by an arbitrary user at an arbitrary time and place. As such, we trade-off accuracy for simplicity and viability in the real-world and develop a system that can be deployed on barometer and GPS equipped smartphones. Furthermore, we show that in our trials, our system can achieve 91% floor-level accuracy after learning an estimated floor-ceiling height via clustering repeated visits to the building.

In this paper, we make two contributions. The first is an RNN with LSTM (Hochreiter & Schmidhuber, 1997) hidden-units which we abbreviate as *IOLSTM* trained to classify a mobile phone as either indoors or outdoors that improves on a previous classifier developed by (Avinash et al., 2010). Because neural networks have not been tried in this type of application, we compare performance against strong baseline classifiers, feed-forward neural network, logistic regression, SVM models and Random Forest models. The second is an algorithm that uses a two-step process to predict the floor level for a user using a smart-phone with a GPS sensor and barometer. In step 1, we train an IOLSTM to classify a smartphone as indoors or outdoors using the smartphone's sensor data. In step 2, we measure the change in barometric pressure of the smartphone from the building entrance to the

smartphone's current location within the building and estimate its floor-level location by clustering height measurements through repeated building visits.

Our 63 experiments across six different buildings in New York City show that the system can estimate the vertical displacement inside a building with 91% accuracy and an absolute floor-level with a one floor margin of error.

2 RELATED WORK

Current approaches used to identify floor-level location fall into two broad categories. The first approach classifies user activity, i.e., walking, stairs, elevator, and generates a prediction based on movement offset (Avinash et al., 2010) (Song et al., 2014). The second category uses a barometer to calculate altitude or relative changes between multiple barometers (Parviainen et al., 2008) (Li et al., 2013) (Xia et al., 2015). We note that elevation can also be estimated using GPS. Although GPS works well for horizontal coordinates (latitude and longitude), GPS is not accurate in urban settings with tall buildings and provides inaccurate vertical location (Lammel et al., 2009).

Song et al. (2014) describe three modules which model the mode of ascent as either elevator, stairs or escalator. Although these methods show early signs of success, they require infrastructure support and tailored tuning for each building. For example, the iOS app (Wonsang, 2014) used in this experiment requires that the user state the floor height and lobby height to generate predictions.

Avinash et al. (2010) use a classifier to detect whether the user is indoors or outdoors. Another classifier detects whether a user is walking, climbing stairs or standing still. For the elevator problem, they build another classifier and attempt to measure the displacement via thresholding. While this method shows promise, it needs to be calibrated to the user's step size to achieve high accuracy and the floor estimates rely on observing how long it takes a user to transition between floors. This method also relies on pre-training on a specific user.

Li et al. (2013) conduct a study of the feasibility of using barometric pressure to generate a good prediction for floor-level. Their first method measures the pressure difference between a reference barometer and a "roving" barometer. The second method uses a single barometer as both the reference and rover barometer, and sets the initial pressure reading by detecting Wi-Fi points near the entrance. This method also relies on knowing beforehand the names of the Wi-Fi access points near the entrance.

Xia et al. (2015) equip a building with reference barometers in each floor. Their method thus allows them to identify the floor without knowing the floor height. This method also requires equipping the building with pressure sensors beforehand.

3 DATA DESCRIPTION

To our knowledge there does not exist a dataset for predicting floor heights. As such, we built an iOS app named Sensory[1] to aggregate data from the smartphone's sensors. We installed Sensory on an iPhone 6s and set to stream data at a rate of 1 sample per second. Each datum consisted of the following: indoors, created at, session id, floor, RSSI strength, GPS latitude, GPS longitude, GPS vertical accuracy, GPS horizontal accuracy, GPS course, GPS speed, barometric relative altitude, barometric pressure, environment context, environment mean bldg floors, environment activity, city name, country name, magnet x, magnet y, magnet z, magnet total.

Each trial consisted of a continuous period of time of Sensory streaming. We started and ended each trial by pressing the start button and stop button on the Sensory screen. We separated data collection by two different motives; the first to train the classifier, the second to make floor predictions. The same sensory setup was used for both with two minor adjustments: 1) Locations used to train the classifier differed from locations used to make building predictions. 2) The indoor feature was only used to measure the classifier accuracy in the real world.

[1] https://github.com/williamFalcon/sensory

3.1 DATA COLLECTION FOR INDOOR-OUTDOOR CLASSIFIER

Our system operates on a time-series of sensor data collected by an iPhone 6s. Although the iPhone has many sensors and measurements available, we only use six features as determined by forests of trees feature reduction (Kawakubo & Yoshida, 2012). Specifically, we monitor the smartphone's barometric pressure P, GPS vertical accuracy GV, GPS horizontal accuracy GH, GPS Speed GS, device RSSI level $rssi$ and magnetometer total reading M. All these signals can be gathered from the GPS transmitter, magnetometer and radio sensors embedded on the smartphone. Appendix table 5 shows an example of data points collected by our system. We calculate the total magnetic field strength from the three axis x, y, z provided by the magnetometer by using equation 1. The data collection procedure is detailed in appendix B.5

$$M = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

3.1.1 DATA COLLECTION FOR FLOOR PREDICTION

The data used to predict the floor-level was collected separately from the IOLSTM classifier data. We treat this dataset as the validation and testing set and only use it to measure system performance. We collected 63 trials among five different buildings throughout New York City to explore the generality of our system. Our building sampling strategy attempts to diversify the locations of buildings, building sizes and building material. The buildings tested explore a wide-range of possible results because of range of building heights found in New York City (Appendix 4). As such, our experiments are a good representation of expected performance in a real-world deployment.

The procedure described in appendix B.6 generates data used to predict a floor change from the entrance floor to the end floor. We count floors by setting the floor we enter to 1. Positive floor changes means ascend, negative changes means descend. This trial can also be performed by starting indoors and walking outdoors. In this case our system would predict the person to be outdoors. If a building has multiple entrances, our system may not give the correct numerical floor value as one would see in an elevator. Our system will also be off in buildings that skip the 13th floor or have odd floor numbering. The GPS lag tended to be less significant when going from inside to outside which made outside predictions trivial for our system. As such, we focus our trials on the much harder outside to inside prediction problem.

3.1.2 DATA COLLECTION FOR FLOOR CLUSTERING

To explore the feasibility and accuracy of our proposed clustering system we conducted 41 separate trials in the Uris Hall building using the same device across two different days. We picked the floors to visit through a random number generator. The only data we collected was the raw sensor data and we did not tag the floor level we traveled to. We wanted to estimate the floor level via completely unsupervised data to simulate a real-world deployment of the clustering mechanism. We used both the elevator and stairs arbitrarily to move from the ground floor to the destination floor.

4 METHODS

In the following section, we present the overall system for estimating the floor-level location of a mobile device inside a building using only inputs from the sensors on-board the mobile device. We will describe the neural network architecture we designed to classify a device as either indoors or outdoors. Next we'll present an algorithm to identify indoor/outdoor transitions (IO), an algorithm to measure the relative vertical displacement based on the device's barometric pressure, and a clustering based method to estimate the absolute floor-level.

4.1 RECURRENT NEURAL NETWORK ARCHITECTURE

The first key component of our system, the IOLSTM, classifies each device reading as either indoors or outdoors (IO). This critical step allows us to focus on a subset of the user's activity, namely the part when they are indoors. We conduct our floor predictions in this subspace only. When a user is outside, we assume they are at ground level. This implies that our system does not detect floor-level

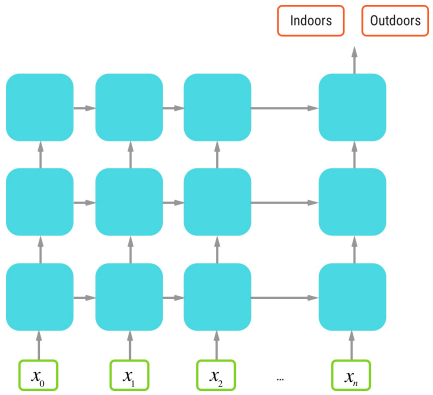

Figure 1: IOLSTM network architecture. A 3-layer stacked RNN with LSTM hidden units. Inputs are sensor readings for n consecutive time-steps.

in open spaces and may show a user who is on the roof as being at ground level. We treat these scenarios as edge-cases and focus our attention on floor-level location when indoors.

Our IOLSTM is a 3-layer RNN with LSTM hidden units. The RNN is fed n consecutive device signal readings as the input X and attempts to label the output y as 1 if the device is indoors or 0 if the device is outdoors. For any set P_i of n points we try to predict the label y_i for the middle point p_i. Through this setup, we allow the network to make an IO prediction for any given p_i by using the points $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ when $k = 3$. The idea is that the network learns the relationship for the given point by looking into the past and future around that point. This design means our system has a lag of $n/2 - 1$ second before the network can give a valid prediction. We chose $n = 11$ as the number of points in X by random-search (Bergstra & Bengio, 2012) over the point range $[1, 30]$.

The IOLSTM network is given the set P_i as the input X and the label for point p_i as label y. The network minimizes the binary cross-entropy between the true indoor state y of example i and the IOLSTM predicted indoor state IOLSTM$(X) = \hat{y}$ of example i. This objective cost function C can be formulated as:

$$C(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{2}$$

Figure 4.1 shows the overall architecture. The final output of the IOLSTM is a time-series $T = \{t_1, t_2, ..., t_i, t_n\}$ where each $t_i = 0, t_i = 1$ if the point is outside or inside respectively.

The IO classifier is the most critical part to our system. The accuracy of the floor predictions is dependent on the IO transition prediction accuracy. The classifier exploits the GPS signal, which does not cut out immediately when walking into a building. We call this the "lag effect." The lag effect renders our system ineffective in 1-2 story houses or glass buildings.

A large lag means the user could have already moved between floors by the time the classifier catches up. This will throw off results by 1-2 floors depending on how fast the device has moved between floors. The same effect is problematic when the classifier prematurely transitions indoors as the device could have been walking up or down a sloped surface before entering the building. We correct for most of these errors by looking at a window of size w around the exact classified transition point. We use the minimum device barometric pressure in this window as our p_0. We set $w = 20$ based on our observations of lag between the real transition and our predicted transition during experiments. This location fix delay was also observed by Nawarathne et al Nawarathne et al. to be between 0 and 60 seconds with most fixes happening between 0 and 20 seconds.

4.2 Indoor-Outdoor transition detector

Given our IOLSTM IO predictions, we now need to identify when a transition into or out of a building occurred. This part of the system classifies a sub-sequence $s_i = T_{i:i+|V_1|}$ of IOLSTM

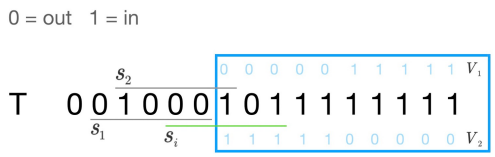

Figure 2: To find the indoor/outdoor transitions, we convolve filters V_1, V_2 across timeseries of Indoor/Outdoor predictions T and pick each subset s_i with a Jaccard distance ≥ 0.4. The transition t_i is the middle index in set s_i.

predictions as either an IO transition or not. Our classifier consists of two binary vector masks V_1, V_2

$$V_1 = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0] \tag{3}$$

$$V_2 = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1] \tag{4}$$

that are convolved across T to identify each subset $s_i \in S$ at which an IO transition occurred. Each subset s_i is a binary vector of in/out predictions. We use the Jaccard similarity (Choi et al., 2010) as an appropriate measure of distance between V_1, V_2 and any s_i.

As we traverse each subset s_i we add the beginning index b_i of s_i to B when the Jaccard distances $J_1 \geq 0.4$ or $J_2 \geq 0.4$ as given by Equation 5.

We define J_j, $j = \{1, 2\}$ by

$$J_j = J(s_i, V_j) = \frac{|s_i \cap V_j|}{|s_i| + |V_j| - |s_i \cap V_j|} \tag{5}$$

Once we have each beginning index b_i of the range s_i, we merge nearby b_is and use the middle of each set b as an index of T describing an indoor/outdoor transition. At the end of this step, B contains all the IO transitions b into and out of the building. The overall process is illustrated by Figure 2 and described in Algorithm 1.

4.3 Estimating the device's vertical height

This part of the system determines the vertical offset measured in meters between the device's in-building location, and the device's last known IO transition. In previous work (Song et al., 2014) suggested the use of a reference barometer or beacons as a way to determine the entrances to a building. Our second key contribution is to use the IOLSTM IO predictions to help our system identify these indoor transitions into the building. The IOLSTM provides a self-contained estimator of a building's entrance without relying on external sensor information on a user's body or beacons placed inside a building's lobby.

This algorithm starts by identifying the last known transition into a building. This is relatively straightforward given the set of IO transitions B produced by the previous step in the system, Section 4.2. We can simply grab the last observation $b_n \in B$ and set the reference pressure p_0 to the lowest device pressure value within a 15 second window around b_n. A 15 second window accounts for the observed 15 second lag that the GPS sensor needs to release the location lock from serving satellites.

The second datapoint we use in our estimate is the device's current pressure reading p_1. To generate the relative change in height m_Δ we can use the international pressure equation (6)

$$m_\Delta = f_{floor}(p_0, p_1) = 44330(1 - (\frac{p_1}{p_0})^{\frac{1}{5.255}}) \tag{6}$$

As a final output of this step in our system we have a scalar value m_Δ which represents the relative height displacement measured in meters, between the entrance of the building and the device's current location.

4.4 RESOLVING AN ABSOLUTE FLOOR LEVEL

This final step converts the m_Δ measurement from the previous step into an absolute floor level. This specific problem is ill-defined because of the variability in building numbering systems. Certain buildings may start counting floors at 1 while others at 0. Some buildings also avoid labeling the 13th floor or a maintenance floor. Heights between floors such as lobbies or food areas may be significantly larger than the majority of the floors in the building. It is therefore very difficult to derive an absolute floor number consistently without prior knowledge of the building infrastructure. Instead, we predict a floor-level indexed by the cluster number discovered by our system.

We expand on an idea explored by Xia et al. (2015) to generate a very accurate representation of floor heights between building floors through repeated building visits. The authors used clusters of barometric pressure measurements to account for drift between sensors. We generalize this concept to accurately estimate the floor-level of a device. First, we define the distance between two floors within a building $d_{i,j}$ as the tape-measure distance from "carpet to carpet" between floor i and floor j. Our first solution aggregates m_Δ estimates across various users and their visits to the building. As the number M of m_Δ's increases, we approximate the true latent distribution of floor heights which we can estimate via the observed m_Δ measurement clusters K. We generate each cluster $k_i \in K$ by sorting all observed m_Δ's and grouping points that are within 1.5 meters of each other. We pick 1.5 because it is a value which was too low to be an actual $d_{i,j}$ distance as observed from an 1107 building $d_{i,j}$ dataset of New York City buildings from the Council on tall buildings and urban habitat (sky, 2017). During prediction time, we find the closest cluster k to the device's m_Δ value and use the index of that cluster as the floor level. Although this actual number may not match the labeled number in the building, it provides the true architectural floor level which may be off by one depending on the counting system used in that building. Our results are surprisingly accurate and are described in-depth in section 5.

When data from other users is unavailable, we simply divide the m_Δ value by an estimator $\hat{m}$ from the sky (2017) dataset. Across the 1107 buildings, we found a bi-modal distribution corresponding to office and residential buildings. For residential buildings we let $\hat{m}_r = 3.24$ and $\hat{m}_o = 4.02$ for office buildings, Figure 6 shows the dataset distribution by building type. If we don't know the type of building, we use $\hat{m} = 3.63$ which is the mean of both estimates. We give a summary of the overall algorithm in appendix (2).

5 EXPERIMENTS AND RESULTS

We separate our evaluation into two different tasks: The Indoor-Outdoor classification task and the floor-level prediction task. In the Indoor-Outdoor detection task we compare the RNN model against four other baseline models. In the floor-level prediction task we evaluate the full system.

5.1 INDOOR-OUTDOOR CLASSIFICATION RESULTS

Table 1: Model performance on validation and training set.

Model	Validation Accuracy	Test Accuracy
LSTM	**0.949**	**0.911**
Feed-Forward	0.952	0.899
SVM	0.938	0.879
Random Forest	0.954	0.82
Logistic Regression	0.91	0.687

In this first task our goal is to predict whether a device is indoors or outdoors using data from sensors on-board a mobile device.

All indoor-outdoor classifiers are trained and validated on data from 35 different trials for a total of 5082 datapoints. The data collection process is described in section 3.1. We used 80% training, 20%

validation split. We don't test on this data, but instead test from separately collected data obtained from the procedure in section 3.1.1.

We train the IOLSTM for 24 epochs with a batch size of 128 and optimize using Adam (Kingma & Ba, 2014) with a learning rate of 0.004. We chose our hyper-parameters for the network and number of points in X through a random search (Bergstra & Bengio, 2012) on the hyper-parameter space. We designed the IOLSTM network architecture through experimentation and picked the best architecture based on validation performance.

RNN architecture: Layer 1 has LSTM cell with 50 neurons followed by a dropout layer set to 0.2. Layer 2 is the same as layer 1 but with 18 neurons. Layer 3 is an LSTM with 2 units fed directly into a 1-neuron feed-forward model with a sigmoid activation function.

We evaluate the IOLSTM against a number of baseline methods, a feed-forward model, SVM, logistic regression and Random Forest. Table 1 gives the performance for each model. The LSTM model outperforms all other models by a large margin and generalizes better than the fully-connected network on the held out test set.

5.2 Floor-Level Prediction Results

Table 2: Floor level prediction error. A floor error of 1 means our system's predicted floor level was 1 higher than the actual floor number as listed in the elevator.

Trials	Floor Error
39	0
14	1
4	2
6	-4+

We measure our performance in terms of number of floors traveled. For each trial, the error between the target floor f and the predicted floor $\hat{f}$ is their absolute difference. Our system does not report the absolute floor number as it may be different depending on where the user entered the building or how the floors are numbered within a building. We ran two tests with different m values. In the first experiment we used $m = m_r = 4.02$ across all buildings. This heuristic predicted the correct floor level within 1 floor with 73% accuracy. In the second experiment we used a specific m value for each individual building.

This second experiment predicted the correct floor with 91% accuracy and a 1-floor margin of error. These results show that a proper m value can increase the accuracy dramatically. Table 2 describes our results. In each trial we either walked up or down the stairs or took the elevator to the destination floor, according to the procedure outlined in section 3.1.1. The system had no prior information about the buildings in these trials and made predictions solely from the classifier and barometric pressure difference.

5.3 Floor Level Through clustering

Figure 5 shows the resulting k clusters from the experiment. In this figure, it is clear to see that this indeed does provide a very accurate estimator that accounts for arbitrary differences between floors. In this particular building, the first floor is 5 meters away from the ground, while the next two subsequent floors have a distance of 3.65 meters and remainder floors a distance of 3.5. To verify our estimates, we used a tape-measure in the stairwell to measure the distance between floors from "carpet to carpet." Table 3 compares our estimates against the true measurements.

6 Conclusion

In this paper we presented a system that can predict a device's floor level within ± 1 floors with 91% accuracy. Unlike previous systems explored by Avinash et al. (2010), Song et al. (2014), Parviainen

Table 3: Estimated distances $d_{i,j}$ between floor i and floor j in the Uris Hall building.

Floor range	**Estimated** $d_{i,j}$	**Actual** $d_{i,j}$
1-2	5.17	5.46
2-3	3.5	3.66
3-4	3.4	3.66
4-5	3.45	3.5
5-6	3.38	3.5
6-7	3.5	3.5
7-8	3.47	3.5

et al. (2008), Li et al. (2013), Xia et al. (2015), our system is completely self-contained and can generalize to various types of tall buildings which can exceed 19 or more stories. This makes our system realistic for real-world deployment with little to no infrastructure support required.

We also introduced the IOLSTM, a Recurrent Neural Network using Long Short-Term Memory hidden units that solves the indoor-outdoor classification problem with 91.1% accuracy. The IOLSTM outperforms our baseline feed-forward network, SVM, random forest, logistic regression and outperforms a previous systems designed by Radu et al. (2014) and Zhou et al. (2012).

Finally, we showed that we can learn the distribution of floor heights within a building by aggregating m_Δ measurements across different visits to the building. This method allows us to generate precise floor level estimates via unsupervised methods. Our overall system marries these different elements to make our system a feasible approach to speed up real-world emergency rescues in cities with tall buildings.

REFERENCES

Council on tall buildings and urban habitat. http://www.skyscrapercenter.com, 2017.

P Avinash, L Ken, V Pradeep, K Aalaya, D Karthik, P Sameera, and SS Gaurav. Coarse in-building localization with smartphones., 2010.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Bosch. Bosch bpm280 datasheet. https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-18.pdf, Nov 2016.

Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

comScore. Comscore reports january 2017 u.s. smartphone subscriber market share, Mar 2017. URL http://www.prnewswire.com/news-releases/comscore-reports-january-2017-us-smartphone-subscriber-market-share-300420345.html.

Sinem Coleri Ergen, Huseyin Serhat Tetikol, Mehmet Kontik, Raffi Sevlian, Ram Rajagopal, and Pravin Varaiya. Rssi-fingerprinting-based mobile phone localization with route constraints. *IEEE Transactions on Vehicular Technology*, 63(1):423–428, 2014.

Adam Clark Estes. How apple's m7 chip makes the iphone 5s the ultimate tracking device, Sep 2013. URL https://gizmodo.com/how-apples-m7-chip-makes-the-iphone-5s-the-ultimate-tr-1286594287.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Hideko Kawakubo and Hiroaki Yoshida. Rapid feature selection based on random forests for high-dimensional data. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

G Lammel, J Gutmann, L Marti, and M Dobler. Indoor navigation with mems sensors. *Procedia Chemistry*, 1(1):532–535, 2009.

Binghao Li, Bruce Harvey, and Thomas Gallagher. Using barometers to determine the height for indoor positioning. In *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*, pp. 1–7. IEEE, 2013.

Kalan Nawarathne, Rudi Ball, Francisco Câmara Pereira, Fang Zhao, Atika Diana Rahardjo, Christopher Zegras, and Moshe Ben-Akiva. Trade-off between smartphone battery life and stop detection accuracy.

Jussi Parviainen, Jussi Kantola, and J Collin. Differential barometry in personal navigation. In *Position, Location and Navigation Symposium, 2008 IEEE/ION*, pp. 148–152. IEEE, 2008.

Valentin Radu, Panagiota Katsikouli, Rik Sarkar, and Mahesh K Marina. Poster: Am i indoor or outdoor? In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, pp. 401–404. ACM, 2014.

Dennis Roberson. Declaration of dennis roberson, Jul 2014. URL `https://ecfsapi.fcc.gov/file/7521384934.pdf`.

Wonsang Song, Jae Woo Lee, Byung Suk Lee, and Henning Schulzrinne. Finding 9-1-1 callers in tall buildings. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, pp. 1–9. IEEE, 2014.

Ashraf Tahat, Georges Kaddoum, Siamak Yousefi, Shahrokh Valaee, and Francois Gagnon. A look at the recent wireless positioning techniques with a focus on algorithms for moving receivers. *IEEE Access*, 4:6652–6680, 2016.

Song Wonsang. Indoor-location. `https://github.com/ws2131/indoor-location`, 2014.

Hao Xia, Xiaogang Wang, Yanyou Qiao, Jun Jian, and Yuanfei Chang. Using multiple barometers to detect the floor location of smart phones with built-in barometric sensors for indoor positioning. *Sensors*, 15(4):7857–7877, 2015.

Weixing Xue, Weining Qiu, Xianghong Hua, and Kegen Yu. Improved wi-fi rssi measurement for indoor localization. *IEEE Sensors Journal*, 17(7):2224–2230, 2017.

Kaiqing Zhang, Hong Hu, Wenhan Dai, Yuan Shen, and Moe Z Win. Indoor localization algorithm for smartphones. *arXiv preprint arXiv:1503.07628*, 2015.

Pengfei Zhou, Yuanqing Zheng, Zhenjiang Li, Mo Li, and Guobin Shen. Iodetector: A generic service for indoor outdoor detection. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pp. 113–126. ACM, 2012.

A APPENDIX

B REAL-WORLD CONSIDERATIONS

In this section we explore potential pitfalls of our system in a real-world scenario and offer potential solutions.

B.1 EXTERNAL PRESSURE CHANGES AND PRESSURE DRIFT

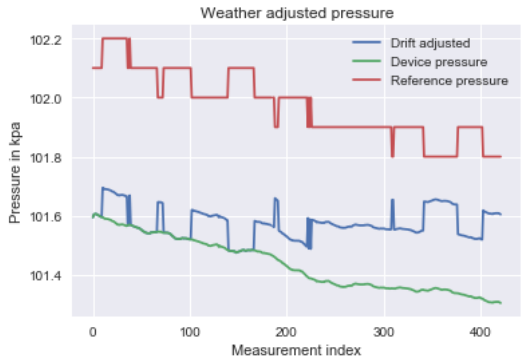

Figure 3: Adjusting device pressure from readings from a nearby station. The readings were mostly adjusted but the lack of resolution from the reference station made the estimate noisy throughout the experiment.

One of the main criticisms for barometric pressure based systems is the unpredictability of barometric pressure as a sensor measurement due to external factors and changing weather conditions. Critics have cited the discrepancy between pressure-sealed buildings and their environments, weather pattern changes, and changes in pressure due to fires (Roberson, 2014).

Li et al. (2013) used a reference weather station at a nearby airport to correct weather induced pressure drift. They showed the ability to correct weather drift changes with a maximum error of 2.8 meters. Xia et al. (2015) also used a similar approach but instead adjust their estimates by reading temperature measurements obtained from a local weather station. We experimented with the method described by (Li et al., 2013) and conducted a single trial as a proof of concept. We measured the pressure reading p from an iPhone device on a table over 7 hours while simultaneously querying weather data w every minute. By applying the offset equation 7 we attempt to normalize the p_i reading to the first p_0 reading generated by the device

$$p_0 \approx |w_i - w_0| + p_i \tag{7}$$

we were able to stay close to the initial p_0 estimate over the experiment period. We did find that the resolution from the local weather station needed to be fine-grained enough to keep the error from drifting excessively. Figure 3 shows the result of our experiment.

B.2 TIME SENSITIVITY

Our method works best when an offset m_Δ is predicted within a short window of making a transition within the building. (Li et al., 2013) explored the stability of pressure in the short term and found the pressure changed less than 0.1 hPa every 10 minutes (Li et al., 2013) on average. The primary challenge arises in the case when a user does not leave their home for an extended period of hours. In this situation, we can use the previously discussed weather offset method from section B.1, or via indoor navigation technology. We can use the device to list Wi-Fi access points within the building and tag each cluster location using RSSI fingerprinting techniques as described by Zhang et al. (2015) Ergen et al. (2014) and Tahat et al. (2016). With these tags in place, we can use the average

floor level tags of the nearest n Wi-Fi access points once the delay between the building entrance and the last user location is too large. We could not test this theory directly because of the limitations Apple places on their API to show nearby Wi-Fi access points to non-approved developers.

B.3 Differences between barometers

Another potential source of error is the difference between barometric pressure device models. Xia et al. (2015) conducted a thorough comparison between seven different barometer models. They concluded that although there was significance variation between barometer accuracies, the errors were consistent and highly correlated between each device. They also specifically mentioned that the Bosch BMP180 barometer, the older generation model to the one used in our research, provided the most accurate measurements from the other barometers tested. In addition, Li et al. (2013) also conducted a thorough analysis using four different barometers. Their results are in line with Xia et al. (2015), and show a high correlation and a constant offset between models. They also noted that within the same model (Bosch BMP180) there was also a measurement variation but it was constant (Li et al., 2013).

B.4 Battery Impact

Our system relies on continuous GPS and motion data collected on the mobile device. Continuously running the GPS and motion sensor on the background can have an adverse effect on battery life. Zhou et al. (2012) showed that GPS drained the battery roughly double as fast across three different devices. Although GPS and battery technology has improved dramatically since 2012, GPS still has a large adverse effect on battery life. This effect can vary across devices and software implementation. For instance, on iOS the system has a dedicated chip that continuously reads device sensor data (Estes, 2013). This approach allows the system to stream motion events continuously without rapidly draining battery life. GPS data, however, does not have the same hardware optimization and is known to drain battery life rapidly. Nawarathne et al. conducted a thorough study of the impact of running GPS continuously on a mobile device. They propose a method based on adjusted sampling rates to decrease the negative impact of GPS on battery life. For real-world deployment this approach would still yield fairly fine-grained resolution and would have to be tuned by a device manufacturer for their specific device model.

B.5 DATA COLLECTION PROCEDURE FOR INDOOR-OUTDOOR CLASSIFIER DATA

1) Start outside a building. 2) Turn Sensory on, set indoors to 0. 3) Start recording. 4) Walk into and out of buildings over the next n seconds. 5) As soon as we enter the building (cross the outermost door) set indoors to 1. 6) As soon as we exit, set indoors to 0. 7) Stop recording. 8) Save data as CSV for analysis. This procedure can start either outside or inside a building without loss of generality.

B.6 DATA COLLECTION PROCEDURE FOR INDOOR-OUTDOOR CLASSIFIER DATA

1) Start outside a building. 2) Turn Sensory on, set indoors to 0. 3) Start recording. 4) Walk into and out of buildings over the next n seconds. 5) As soon as we enter the building (cross the outermost door) set indoors to 1. 6) Finally, enter a building and ascend/descend to any story. 7) Ascend through any method desired, stairs, elevator, escalator, etc. 8) Once at the floor, stop recording. 9) Save data as CSV for analysis.

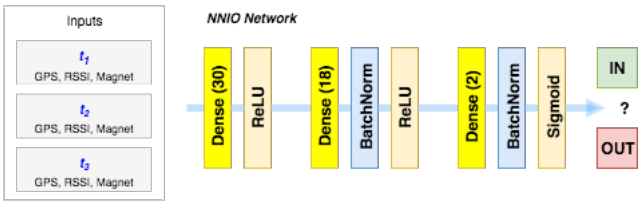

Figure 4: Feed-forward network architecture. A simple, three fully connected layer network. FC30 - FC18 - FC2. Inputs are sensor readings at times t_1, t_2, t_3.

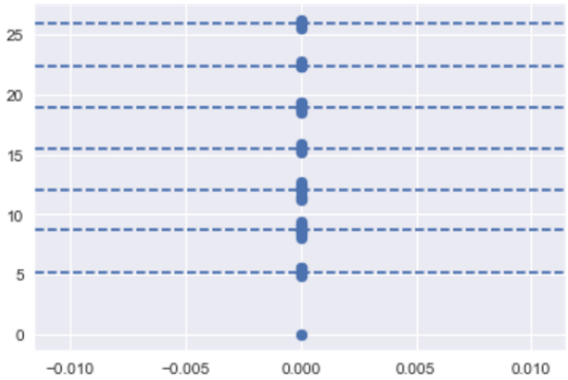

Figure 5: Distribution of m_Δ measurements across 41 trials in the Uris Hall building in New York City. A clear $d_{i,j}$ size difference is specially noticeable at the lobby. Each dotted line corresponds to an actual floor in the building learned from clustered data-points.

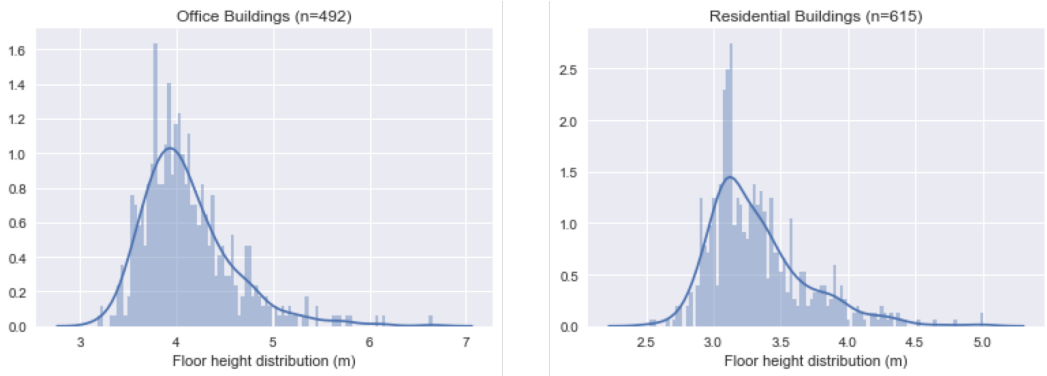

Figure 6: Distribution of "floor to floor" heights $d_{i,j}$ across 615 office (left) and 492 residential buildings (right) in New York City.

Table 4: Description of buildings used in our experiments. Material is the perceived material on the outside of the building.

Bldg Name	Stories	Material	Trials
10 Rockefeller Plz	17	Concrete	10
Uris Hall (CU)	12	Concrete	14
Mudd (CU)	19	Concrete	10
Noco (CU)	14	Glass	10
Social Work (CU)	13	Brick/Glass	20

Table 5: Example data points collected for the indoor-outdoor classifier. Feature vector X is constructed from 3 readings at consecutive time intervals t_i. y is the middle point's IO label (1* in this case).

t_i	rssi	GV	GH	GS	M	P	IO (y)
1	-82	76.05	1414	-1	1015.3	100.7	0
2	-82	48	30	0.36	1019.4	100.3	1*
3	-82	48	30	0.36	1019.4	100.2	1

C ALGORITHM REFERENCES

Algorithm 1 Find Indoor Outdoor transitions

```
procedure FINDIOINDEXES(T)
    V_1 = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
    V_2 = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
    S = []
                                                    ▷ Find Jacc ≥ 0.4
    for i ∈ {1, ..., |T| − |V_1|} do
        s_i = {t_i, ..., t_{i+|V_1|}}
        if J(s_i, V_1) >= 0.4 or J(s_i, V_2) >= 0.4 then
            S ← i
                                                    ▷ Merge s_i ranges
    merged = []
    for each b_i ∈ S do
        if b_i + 2 <= b_{i−1} then
            merged[i] ← Merge(b_i, b_{i−1})

    transitions = []
    for each m_i ∈ merged do
        transitions ← Middle(m_i)
return transitions
```

Algorithm 2 Predict Floor

```
procedure PREDICTFLOOR
    T = CaptureData(GPS, baro, RSSI, magnet)
                                                    ▷ Classify each reading as IO
    for t ∈ T do
        t_IO = NNIO(t)
                                                    ▷ Locate last IO transition
    B = FindIOTransitionIndexes(T)
    b_n = B[n]
    if b_{n,IO} = 0 then return "outdoors"
                                                    ▷ Compute m_Δ
    p_1 = devicePressure(T_n)
    p_0 = devicePressure(b_n)
    m_Δ = f_floor(p_0, p_1)
                                                    ▷ Find the nearest k cluster
    for k_i ∈ K do
        f̂ = |k_i − m_Δ| ≤ 1.5
                                                    ▷ Estimate floor level
    if exists f̂ then
        f̂ = i
    else
        f̂ = m_Δ / bestM()
    return f̂
```