Journal of Network and Computer Applications 165 (2020) 102714

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Using perceived direction information for anchorless relative indoor localization

Steven M. Hernandez, Eyuphan Bulut *

Department of Computer Science, Virginia Commonwealth University, 401 West Main St. Richmond, VA, 23284, USA

A R T I C L E I N F O

Keywords:
Indoor localization
Monte Carlo algorithm

A B S T R A C T

Identifying the positions of mobile devices within indoor environments allows for the development of advanced context-based applications and general environmental awareness. Classic localization methods require GPS; an expensive, high power consuming and inaccurate solution for indoor scenarios. Relative positioning instead allows nodes to recognize location in relation to neighboring nodes without the requirement of GPS. To triangulate their own position however, indoor localization methods either use Received Signal Strength Indication (RSSI) retrieved from neighboring devices to determine distance or simple binary contact information denoting whether two nodes are in communication range of one another. RSSI however is plagued by many sources of noise, thus decreasing distance prediction accuracy as well as being unreliable for networks of heterogeneous devices. Further, using only binary contacts provides a limited information for localization. In our work, we first demonstrate the unreliable nature of RSSI in heterogeneous networks. We then demonstrate our intermediate solution between unreliable RSSI and oversimplified binary classifications by introducing Perceived Direction Information (PDI) composed of three states: approaching, retreating and invisible. Through real world experiments, we demonstrate that PDI can be predicted using a Dense Neural Network with more than 95% accuracy even on devices not used during training. We then describe an anchorless Monte Carlo Localization (MCL) algorithm which uses PDI to achieve higher accuracy and a reduction of communication over the state-of-the-art MCL based methods.

1. Introduction

Indoor localization aims to identify people (Wang et al., 2012), robots (Santos et al., 2013) or assets (Pease et al., 2017) as they move through indoor settings where ubiquitous localization solutions such as GPS are unable to provide accurate results. Primarily, indoor localization is achieved by obtaining GPS-like coordinates or by recognizing the closeness of static targets to specific areas such as at key entrances or exits to a building which may be identified by unique signatures such as through radio spectrum fingerprinting (Wang et al., 2012; Alzantot and Youssef, 2012; Luo et al., 2016). These systems not only require high setup costs in both materials and human involvement, but also suffer from diminishing returns as models produce less accurate results as environments change over time. Relative positioning on the other hand aims to recognize similar features in dynamic environments. Instead of judging the closeness of a node to some static landmarks, the relative closeness to the neighboring mobile nodes is considered. Through this method no additional infrastructure will need to be deployed within the environment, thus decreasing the costs associated with setup within the environment. To further reduce the costs of this initial setup, relative positioning models are typically generalized to work in any new environment without requiring time consuming manual setup.

In this work,[1] we study relative positioning of the nodes in mobile ad hoc and opportunistic networks. We aim to locate the positions of nodes relative to each other while minimizing the required data communication between any pair of neighboring nodes. Many indoor localization and relative positioning solutions (Rajan et al., 2019; Abouzar et al., 2016; Chen et al., 2017) use Received Signal Strength Indication (RSSI) levels to judge the physical distance of devices to one another, however this can result in poor estimations due to the high variance in RSSI values. We evaluate the effect of these sources of interference on RSSI collection to show that noise between devices from similar and different manufacturers precludes our ability to directly translate RSSI to distance. Additionally, because the rate of change of RSSI is so rapid,

* Corresponding author.
E-mail address: ebulut@vcu.edu (E. Bulut).

[1] The preliminary version of this study appeared in (Hernandez and Bulut, 2019).

https://doi.org/10.1016/j.jnca.2020.102714
Received 26 November 2019; Received in revised form 19 April 2020; Accepted 13 May 2020
Available online 19 May 2020

sharing this dynamic RSSI value to neighbors beyond one communication hop would cause high message overhead and overwhelm the network bandwidth.

Other works (Abu znaid et al., 2017; MacLean and Datta, 2013; Radak et al., 2017; Chen et al., 2013) simplify the problem to using binary contact information where a device can either communicate (*visible*) or cannot communicate (*invisible*) to another node. However, by reducing the problem to only two states, these algorithms remove all possible information available from RSSI. Instead, we look to find a middle ground between oversimplified contact-based methods and highly volatile RSSI-based methods by introducing *Perceived Direction Information* (PDI) which is made up of three states: *approaching*, *retreating* and *invisible*. We demonstrate through physical experiments that PDI can be computed[2] effectively with more than 95% accuracy using a Dense Neural Network even with previously unseen heterogeneous devices. We develop a Monte Carlo Localization (MCL) algorithm which considers PDI when performing predictions. We then show how PDI can reduce the sampling areas compared to contact-based localization algorithms by upwards of 99%. We evaluate the capabilities of our proposed solution through simulations to show that TrinaryMC can produce higher accuracy than existing MCL methods. Furthermore, because existing MCL algorithms require GPS enabled anchor nodes, our anchorless algorithm removes the need for high energy consuming GPS hardware while also decreasing the amount of communication overhead required.

The rest of the paper is formatted as follows. We begin by describing existing related localization works in Section 2. Then, we demonstrate through experiments the unreliability of RSSI for heterogeneous networks in Section 3. We then describe our method for extracting PDI from unreliable RSSI through machine learning in Section 4. Recognizing our ability to reliably collect PDI, we describe a novel MCL algorithm using PDI in Section 5 and evaluate our algorithm in Section 6. Finally, we conclude our work in Section 7.

2. Related works

Early work in indoor localization employed Radio Frequency Identification (RFID), successfully applying the technology to fields such as manufacturing and healthcare (Sanpechuda and Kovavisaruch, 2008; Zhou and Shi, 2008). However RFID localization requires the use of static readers placed throughout a building to supply adequate coverage in addition to requiring specialized equipment to tag items for localization. Bluetooth beaconing takes a similar approach by replacing RFID with Bluetooth Low Energy (BLE) (Chen et al., 2017; Cheraghi et al., 1802; Yucel and Bulut, 2018). The primary advantage of such a system is the ubiquity of BLE enabled smart devices specifically in applications requiring the tracking of human subjects. Still, static hardware must be placed throughout a building to provide adequate coverage. Other works take hybrid approaches such as in (Yuanfeng et al., 2016) where the authors develop a method for combining Pedestrian Dead Reckoning with existing WiFi based localization methods for cases when WiFi infrastructure is unavailable for localization in areas within a given building.

Relative positioning recently gathered interest for locating devices in indoor settings. Studies such as DiscoveryTree (Chabloz et al., 2017) create a multi-hop mesh tree network which allows devices to simply recognize whether a given node is within the network and if so, which other nodes are in range. Bellrock (Zidek et al., 2018) utilizes the mobile devices as Bluetooth beacons when they are stationary and stops beaconing when the devices move to prevent tracking of individuals' movements for privacy preservation. Many other works (Rajan et al., 2019; Abouzar et al., 2016) take the approach of determining the distance of pairs of nodes through RSSI readings and then employing Multi-Dimensional Scaling (MDS). However, sharing constantly changing RSSI information between neighbors could very easily overwhelm the bandwidth of the network. Furthermore, as argued in (Heurtefeux and Valois, 2012; Konings et al., 2017), without extensive and time-consuming testing, raw RSSI values are suggested to be unacceptable for use in mapping to distances between devices. The authors do suggest that by testing devices, parameters can be set to achieve slightly higher results, however as shown in (Chen et al., 2017; Lui et al., 2011), radio hardware between different manufacturers and even those produced in the same product line can produce very different RSSI values. Thus, because testing cannot be completed between each pair of devices in any large network, RSSI cannot be an adequate solution. Some works (Kumar et al., 2016) consider not the distance but instead the relative velocity between nodes while others (Wang et al., 2008) remove the use of unreliable RSSI by only considering binary contacts between nodes.

More recent works have begun looking at Channel State Information (CSI) from a device's radio to produce richer insight into the physical layer aiming to produce better localization than RSSI alone (Wang et al., 2017; Yang et al., 2013). CSI data, however, is not openly available on most consumer products (e.g., smartphones).

One method popularized for use in decentralized localization is Monte Carlo Localization (Znaid et al., 2017a). These techniques internally simulate multiple possible situations a given mobile node may be in subject to some constraints such as location or transmission range of a neighbor node. These simulations are then used to determine the most probable position for the node. Existing MCL methods expect the presence of anchor nodes; nodes with access to their exact location through GPS or some other method. For example, solutions in (Chen et al., 2013; Alaybeyoglu, 2015) rely exclusively on the presence of anchor nodes to guide predictions. Other works such as (Abu znaid et al., 2017; MacLean and Datta, 2013; Khedr, 2015) relax this constraint by using non-anchor nodes in addition to anchor nodes when making predictions. However, none of the existing works specifically focus on the cases when no anchors are present. Our aim is to provide a method which does not rely on anchor nodes in order to have low hardware cost per node and less reliance on a small group of nodes within the network.

3. Motivation

RSSI localization, described as a range-based localization method, has been a popular form of device localization in many situations. Models for RSSI localization often ignore the full complexity of environmental noise by simply using the common log-distance path loss model (Cho et al., 2015; Subedi and Kwon, 2016) or by attempting to minimize the negative effects of specific sources of noise explicitly in their models (Golestanian et al., 2018; Yoon et al., 2019; Trogh et al., 2016). Pinpointing sources of noise often requires extensive site survey (Ramani and Tank, 1405) which however can only produce models that are not generalized enough for use in new environments without repeating this same site survey process per environment. Needless to say, the high noise and interference inherent to indoor environments make it much harder to translate RSSI directly into physical distances for localization.

3.1. Sources of noise and interference

For indoor situations, interference of radio signals appears from a number of sources, causing both unexpected drops and rises for RSSI. As Nguyen et al. explored in high rise building structures (Nguyen et al., 2017), with more energy efficient building materials used in construction, more radio *penetration-loss* occurs. Because of penetration-loss, devices which are physically nearby to one another but separated by a wall will be recognized as being much further based on RSSI. Works on

[2] We use Bluetooth based RSSI measurements between the devices to extract PDI information. However, similar results can be obtained with WiFi based RSSI data based on our experiments that are omitted in the paper.

penetration-loss consider simulating the amount of loss occurring based on signal propagation through multiple mediums such as floors and walls (Solahuddin and Mardeni, 2011). Another common source of variance for indoor environments is *fading* caused by *multipath loss* or *shadowing*. In multipath scenarios typical to indoor environments, a signal sent from a transceiver may bounce off from objects or walls. Therefore, the signal strength at the receiver is defined by both the line-of-sight (LOS) and non-LOS (NLOS) multipaths, with unexpected variations in amplitude. Fading typically occurs when an object in the environment prevents the signal from travelling through the direct LOS, thus only multipaths travelling the NLOS arrive successfully to the receiver. These fading issues have been shown to have less effect when using Channel State Information (CSI) (Yang et al., 2013), however most WiFi enabled devices do not give access to CSI and thus application level software is not able to gain the benefits of CSI.

Multi-radio coexistence is another source of interference to consider in dense radio networks, causing interference not only in-band (commonly handled by the likes of CSMA) but also from out-of-band caused by heterogeneous radio systems (Zhu et al., 2007). Specifically considering the existence of both WiFi and Bluetooth (common in a number of consumer grade products) *intermodulation* and external noise become more of an issue because both use 2.4 GHz frequencies. In networks composed of heterogeneous devices, differences in hardware at the transceiver level from different manufacturers result in unexpected RSSI ranges for similar positions. We explore this issue in this work by conducting experiments with different models of smartphones. Further, when considering networks where not all devices are controlled, it is not always possible to recognize the transmission power (TX-Power) of given devices to sufficiently assess relative distances. As a result, a device with high TX-power would appear closer than another device with lower TX-power even at the same physical distance.

3.2. Empirical RSSI analysis

To evaluate these issues inherent from the use of RSSI, we develop a system for collecting RSSI from Android and iOS smartphone devices along with the true distance between devices. The overall system consists of three separate elements as seen in Fig. 1. To collect Bluetooth RSSI values, we need two smartphones which independently listen and transmit beacon messages over Bluetooth Low Energy (BLE). Next, to determine the baseline truth for each experiment, we develop a prototype distance measurement device seen in Fig. 2 (right). One phone is attached to the body of the prototype while the other device is fastened to a retractable rope to reliably measure the distance in real time.

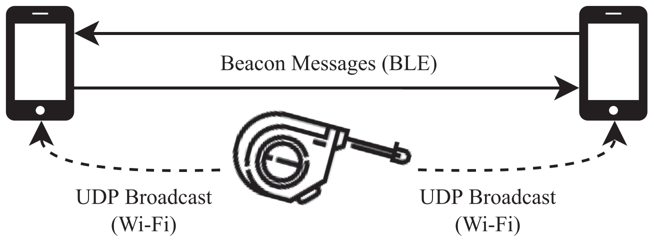

Fig. 1. Diagram representing the communication for our prototype. Smartphones communicate by Bluetooth Low Energy beaconing to allow access to RSSI data while the prototype distance measuring device transmits real-time distance measurements through UDP broadcasts to both devices over WiFi.

3.2.1. Distance measurement device

Our prototype distance measurement device uses an ESP8266 WiFi enabled microcontroller attached to a rotary encoder as shown in Fig. 2 (right). The rotary encoder is then connected to an auto-retracting wheel mechanism. As the rope is pulled out, the rotary encoder is rotated in one direction. An auto-retracting mechanism ensures the rope remains taut when devices approach one another thus allowing the rotary encoder to rotate back in the opposite direction. Rotations are defined by encoder *clicks* which can then be translated to physical distance, to collect the baseline for each experiment. As the distance changes between smartphones, the ESP8266 immediately broadcasts the current distance through a UDP broadcast to the data collecting app running on each smartphone over WiFi.

3.2.2. Data collecting smartphone apps

For our experiments, we are interested in observing the effects of heterogeneous transceiver hardware on RSSI. Thus, we develop custom data collection apps for both iOS and Android smartphones. The apps themselves collect, record and display the true distance in meters, number of clicks from the rotary encoder along with the current RSSI value as shown in Fig. 2 (left). Each experiment is performed on a pair of smartphones and each device transmits BLE beacon packets for RSSI collection. We used four different models of smartphones for our experiments: two using the iOS operating system and two using the Android operating system. Details about each phone model used can be found in Table 1.

3.2.3. Results

For our first experiment, we make the two devices come together and apart at a normal human walking speed eight distinct times. The

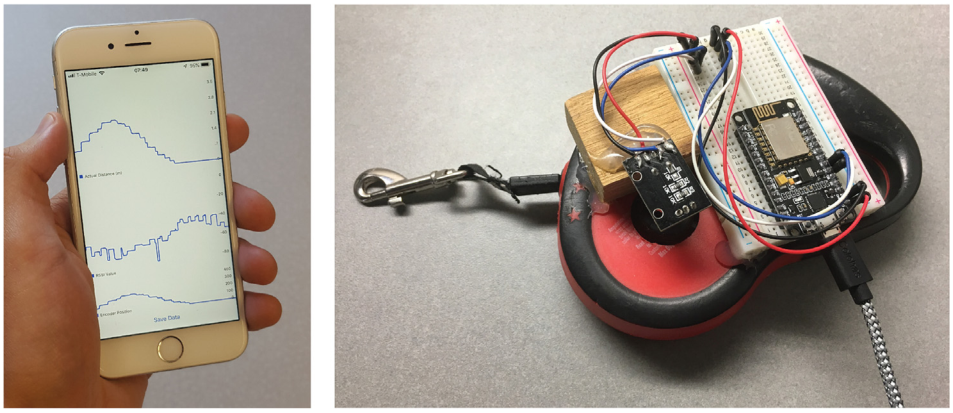

Fig. 2. (left) Data collection app records BLE beacon RSSI along with the current physical distance. (right) Prototype developed with a ESP8266 microcontroller to collect real-time physical distance measurements.

Table 1
Smartphones used in experiments.

Operating System	Model	Model Number	Abbreviation
iOS	iPhone 6s	MKQY2LL/A	iOS6s
iOS	iPhone 8	MQ6V2LL/A	iOS8
Android	Galaxy S6	SM-G920V	AndrS6
Android	Galaxy Note5	SM-N920C	AndrN5

Table 2
Average RSSI difference between pairs of devices.

Smartphone A	Smartphone B	AVG. RSSI Difference
iOS8	iOS6s	2.0297
iOS8	AndrS6	5.2911
iOS8	AndrN5	7.9106
iOS6s	AndrS6	5.9074
iOS6s	AndrN5	8.6392
AndrS6	AndrN5	3.5790

resulting raw RSSI values for this experiment are shown in Fig. 3a. The eight distinct movement events are not clearly visible when looking directly at these raw RSSI values. Thus, we filter the raw RSSI with a rolling average ($RSSI_{avg}^{(t)}$) defined as:

$$RSSI_{avg}^{(t)} = \frac{1}{w}\sum_{i=0}^{w} RSSI^{(t-i)} \tag{1}$$

where w is the number of time steps to be averaged (window size), $RSSI^{(t)}$ is the raw RSSI value at the current time t and $RSSI^{(t-i)}$ is the RSSI value from i time instances ago. Using rolling average with $w = 20$ allows us to more easily recognize these eight events in Fig. 3b. From this, we see that raw RSSI itself is too noisy to clearly recognize useful information. Looking at the true baseline values returned from our encoder in Fig. 3c, we recognize an additional issue with our collected RSSI values. We see that while the encoder value approaches the same maximum and minimum encoder values (meaning the same distance was moved each time), Fig. 3b does not show the same maximum or minimum RSSI values across each event. This shows that moving similar distances at similar times in the exact same environment using the same hardware does not result in the same RSSI output every time.

A further concern we need to identify is the effect of hardware from different manufacturers on the RSSI. To this end, we performed a set of experiments by pairing devices from different manufacturers. Here, we also considered varying speeds, from a slow walk to a faster walking speed, while moving the devices together and apart in order to see how the results vary for different pairs of radio hardware. In Fig. 4a, we first show the RSSI resulting from an experiment between an *Apple iPhone 8* and an *Apple iPhone 6s*. We can see immediately that the RSSI values for these two Apple devices produce almost identical results which we can attribute to the transceiver hardware across the two different models being very similar. Out of all experimental pairings, this pair of devices resulted in the smallest average absolute RSSI difference between devices at only 2.0298. On the opposite end, Fig. 4b shows the resulting RSSI values for an experiment on devices from two different manufacturers: the *Apple iPhone 6s* and the *Samsung Galaxy Note 5*. In this case, we recognize that the difference between the observed RSSI for the two devices is much greater than it is in the previous example. This device pair showcases the highest average absolute RSSI difference for our experiments at 8.6392. The most important observation from this figure is that while RSSI values are different in value, RSSI still exhibits similar changes in slope. This implies that we should still be able to recognize similar PDI values for approaching and retreating across these hardware manufacturers. Table 2 shows the average absolute RSSI difference for each of our experiments. It can plainly be seen that the experiments with the smallest average difference were the two experiments employing smartphones from the same manufacturer while the highest absolute differences are between the devices from different manufactures.

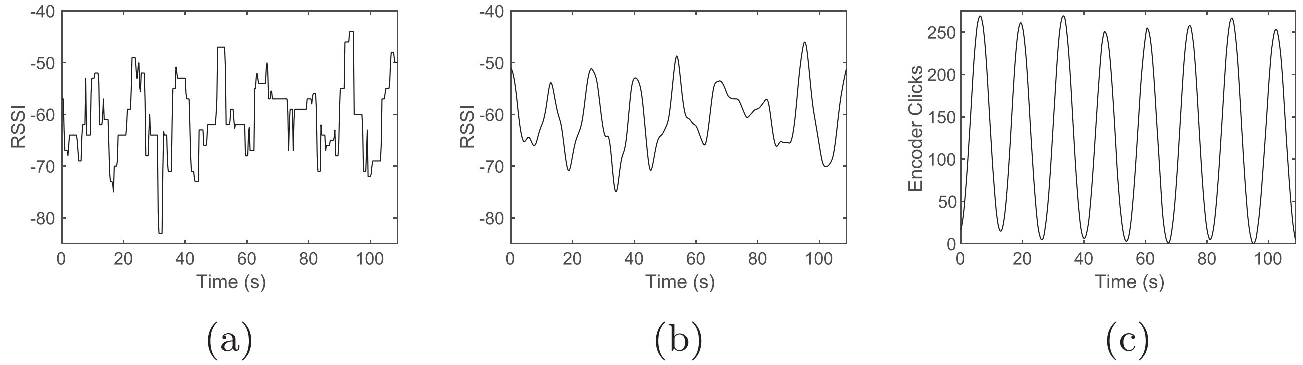

Fig. 3. (a) Raw RSSI from an example experiment where devices move apart then back together eight times. Because of the effect of noise, it is hard to recognize these eight distinct events. (b) RSSI after applying a rolling average on the time-series data. The eight movement events appear more clearly in this representation. (c) Actual encoder values show that each of the eight movement events resulted in very similar maximum distance per event.

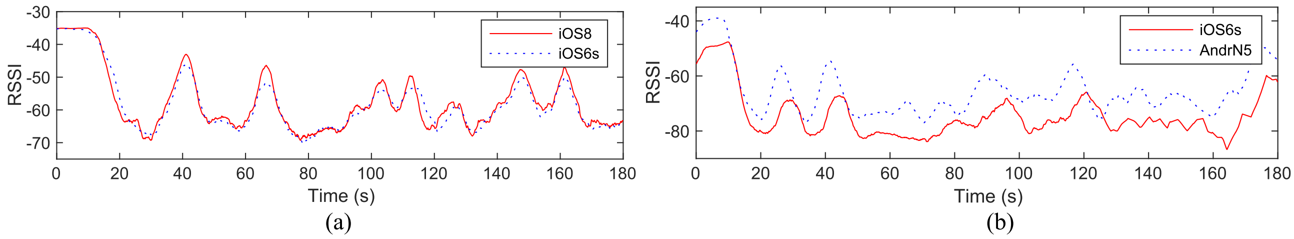

Fig. 4. Comparison of RSSI for two experiments from the perspective of each of the two devices participating in the experiment. (a) Experiment with the two Apple devices showing very similar RSSI across the experiment. (b) Experiment with devices iOS6s and AndrN5 showing high variance in the value of RSSI.

4. Experiments on PDI prediction

Our initial analysis validates our expectations: (i) RSSI is noisy, (ii) RSSI is unreliable in representing physical distances, and (iii) RSSI varies between devices from different hardware manufactures. However, we recognize that while hardware differences affect the amplitude for RSSI, the slope of change between devices matches fairly close. Thus, in this section, we introduce a method for distinguishing the PDI states approaching and retreating using machine learning for heterogeneous networks even for pairs of devices not used in the training of our model.

4.1. Raw RSSI

We develop each of our machine learning models by using the Keras deep neural network library which creates neural network architectures over a TensorFlow backend. For our first model, we create a dense neural network (DNN) with inputs of m raw RSSI readings indicating the most recent RSSI reading and $m - 1$ previous RSSI readings. The expected output is either of the states *approaching* and *retreating* as determined by the encoder values. Training is completed with $m \in \{1, 2, 3\}$. For our input, we filter out a window of samples around transition points from approaching to retreating (or the reverse) because we are only interested in recognizing instances which can be fully defined as one distinct action. We use a grid search to determine the best hyperparameters for each neural network and then most importantly train on the dataset for only one of the experiments (namely the experiment between the iPhone 8 and the iPhone 6s). We select this specific experiment arbitrarily to show that we can recognize patterns from any single pair of devices and then apply our knowledge to other pairs thus demonstrating the generalizability of our model for networks of unseen devices. From the raw data, we do not accomplish particularly high accuracy as can be seen in Fig. 5a with different values of m. We see that the high noise from the raw RSSI translates over to high noise in the validation accuracy for the model. For all values of m, accuracy ranges from as low as 50% up to 90% showing that using raw RSSI for our model will not produce generalizable results.

4.2. RSSI with rolling average

Reconsidering the illustrated example in Fig. 3a and b, we recognize that raw RSSI may not be a useful input for our model. We instead apply (1) to create a rolling average. We train our second DNN with $m \in \{1, 2, 3\}$ and using (1) with averaging window size $w = 20$. As it is shown in Fig. 5b, accuracy now reaches higher consistent levels compared to the accuracy in raw RSSI case with $m = 2$. With $m = 1$, we begin to see more consistently worse results than in the raw RSSI model. This may be accounted for the fact that rolling average can cause a slight drift in measurements which could throw off the model's predictions. Considering $m = 2$, we see that all experiments achieve more than 90% accuracy except for one experiment which still achieves close to 85% accuracy. Surprisingly, we can see that when $m = 3$, the results are worse than $m = 2$. With this additional point of information, the model does not immediately recognize when to ignore the additional time instance so as to achieve similar results with $m = 2$. This implies that we only need to care about the change in RSSI over the past two time instances.

4.3. RSSI change

Next, we recognize a final transformation we can take on our RSSI to create a simplified model by calculating the difference in RSSI from one time instance to the next:

$$RSSI_{diff}^{(t)} = RSSI_{avg}^{(t)} - RSSI_{avg}^{(t-1)} \tag{2}$$

In Fig. 5c we show the accuracy for our machine learning models again using $m \in \{1, 2, 3\}$ and $w = 20$. Our models now achieve consistent results for each value of m. As suggested from the results of the previous model, with $m = 1$, we achieve the best results with all experiments achieving an accuracy more than 95% even though we only train on a single experiment (*ios8* $\rightarrow$ *ios6s*), thus implying that we only need to worry about the change in signal strength between the current time instance and the previous time instance.[3]

Our final selected DNN model is composed of three layers. The first layer is the input layer, taking the current instance of $RSSI_{diff}^{(t)}$ as input. The second layer is a hidden dense layer with 16 hidden nodes. The last layer is an output layer with two nodes representing our classes with each output nodes using a sigmoid activation function. Training loss is calculated using mean squared error. The model was trained using the Adam optimizer with learning rate of 0.001. Batch size for training was 16 samples.

The results show that our model can distinguish approaching and retreating for PDI from incoming noisy RSSI even in cases where the devices used are not known when training the model. Furthermore, because each of our models are implemented in Keras, we are able to translate them into code which can be run directly from an app installed on a smartphone. We develop a simple proof-of-concept application showing that we can in fact use our Keras models on an iPhone using the *coremltools*[4] library. Similar steps can be completed to do the same on an Android smartphone. Now that we can determine in a one-dimensional plane whether two devices are approaching versus retreating with PDI prediction, we next describe how PDI from more than two devices can be used to predict location in a 2D plane through our TrinaryMC algorithm.

5. TrinaryMC localization algorithm

Our TrinaryMC algorithm improves upon existing state-of-the-art Monte Carlo Localization methods by entirely removing reliance on GPS enabled anchor nodes to produce relative positioning. Our method uses the novel PDI concept to describe the state of neighboring nodes when creating predictions. PDI consists of three possible values describing the state of a pair of nodes within the network: *approaching* (1), *retreating* (2) and *invisible* (0), where each state value is identified in our system as the integer shown in parenthesis. Each node transmits beacon packets at given intervals to announce their presence. Neighboring nodes recognize this packet and record the presence of this neighbor along with the perceived directional state: *approaching* or *retreating*. Each node propagates the list of its one-hop neighbors to its neighbors to allow all nodes to recognize the network structure up to some number k-hops away. To collect the current PDI between all neighbors, we create a local PDI matrix, M_t, and denote the current PDI state between two nodes i and j at time t by $M_t^{(i,j)}$.

5.1. Standard Monte Carlo Localization (St-MCL)

We begin by describing the general structure of existing Monte Carlo Localization algorithms (Alaybeyoglu, 2015). The main idea for MCL algorithms is to collect a group of samples in each time instance which can then be used in aggregate to determine the most probable location for the given node under some constraints. All MCL algorithms; including our own, execute the following steps:

- *Initialization Step* - occurs only at startup or when all samples are filtered out. Creates an initial set of random samples subject to some conditions.

[3] Our evaluation with different movement speeds, while being trained with only normal walking speed, consistently show that $m = 1$ provides the best results and all $m \in \{1, 2, 3\}$ values provide more than or around 85% accuracy.

[4] https://apple.github.io/coremltools/.

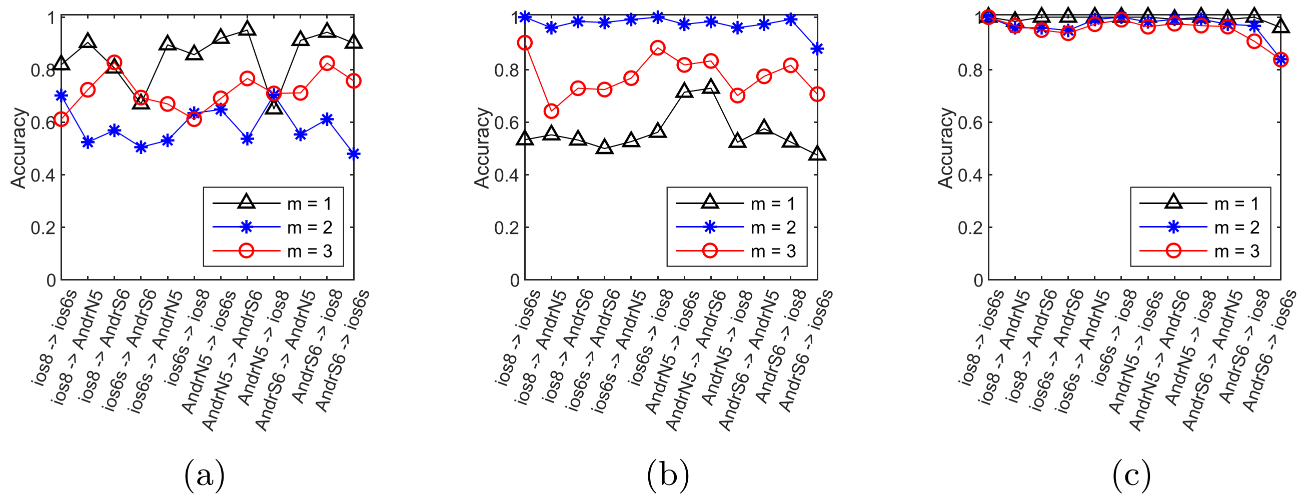

Fig. 5. (a) Accuracy of Machine Learning Model with raw RSSI input. (b) Accuracy of Machine Learning Model with $RSSI_{avg}$ and $w = 20$. (c) Accuracy of Machine Learning Model using $RSSI^{(t)}_{diff}$.

- *Sampling Step (Move Samples)* - using samples from the previous time instance, new samples are created by moving samples randomly within a radius of $v_{\max}$ (i.e., maximum velocity) around the previous sample.
- *Filtering Step (Resample)* - after predicting new locations, any sample not adhering to a given set of constraints are filtered out as invalid samples. If the number of samples left after filtering is above some threshold, then random samples are filtered out to prevent sample sets from exploding in quantity over multiple rounds.

Algorithm 1 TrinaryMC Algorithm

```
1: $\mathbf{S} \leftarrow \mathbf{S}'$
2: if $\|\mathbf{S}\| = 0$ then
3:     $\mathbf{S} \leftarrow Initialization\_Step()$
4: $\mathbf{S}' \leftarrow Sampling\_Step(\mathbf{S})$
5: if $\|\mathbf{S}'\| > \tau_{retained}$ then
6:     $\mathbf{S}' \leftarrow Shuffle(\mathbf{S}')$
7:     $\mathbf{S}' \leftarrow \mathbf{S}'[1 \dots \tau_{retained}]$
8: $\mathbf{S}' \leftarrow Filtering\_Step(\mathbf{S}')$
```

Algorithm 1 demonstrates this general structure for each of the MCL algorithms. The Monte Carlo process is completed every time instance to produce a final predicted set of samples $\mathbf{S}'$. The algorithm begins by setting $\mathbf{S} \leftarrow \mathbf{S}'$. The first time the Monte Carlo algorithm is performed, $\mathbf{S}' = \{\}$ because no samples have been predicted and thus $\|\mathbf{S}\| = 0$ allowing the initialization step to be performed. After the initialization step, the sampling step takes $\mathbf{S}$ either from the previous time instance or from the initialization step to produce a new set of samples $\mathbf{S}'$. After the sampling step, a threshold of a maximum number of samples ($\tau_{retained}$) are retained for the filtering step to prevent the number of samples from growing to an unbounded size. This allows the memory requirements as well as the computation time of the algorithm to be constrained as needed. Finally, after $\tau_{retained}$ samples are selected, any invalid samples are filtered out and $\mathbf{S}'$ is retained for the next call to the procedure in the next time instance.

5.2. Differences of TrinaryMC

Through the use of PDI and our anchorless approach, we introduce the following changes to MCL steps for our own TrinaryMC approach.

5.2.1. Initialization step

In existing MCL algorithms, each sample (p) contains only one (x, y) coordinate pair prediction for a given source node s. However, because of our anchorless approach, a single sample must include an (x, y) coordinate pair $p^{(n)}$ for each neighbor (n) of s. Pseudocode for performing this initialization step is shown in Algorithm 2. The goal of this procedure is to create some number τ_{init} samples in $\mathbf{S}$. To begin, the predicted coordinate pair for s is set to $p^{(s)} = (0, 0)$. Then, for each k-hop neighbor $n \in K$, we determine a random $p^{(n)}$ satisfying our PDI state matrix $M^{(s)}_t$. One important factor to recognize while collecting initial samples is that even though our algorithm uses PDI; in the initialization step, the only states that matter are *invisible* and *visible* because no information exists from previous time instances to distinguish *approaching* from *retreating*.

Algorithm 2 Initialization_Step

```
1: $\mathbf{S} \leftarrow \{\}$
2: for all $i \in \{1 \dots \tau_{init}\}$ do
3:     $p \leftarrow \{\}$
4:     $p^{(s)} \leftarrow (0, 0)$
5:     valid $\leftarrow$ TRUE
6:     for all $n \in K$ do
7:         $p^{(n)} \leftarrow Generate\_Random\_Sample(p, n)$
8:         if $p^{(n)} = \emptyset$ then
9:             valid $\leftarrow$ FALSE
10:    if valid then
11:        $\mathbf{S} \leftarrow \mathbf{S} \bigcup p$
12: return $\mathbf{S}$
```

Generating random samples is completed by using the boxed Monte Carlo approach described in (Baggio and Langendoen, 2008). A naïve approach to generating random samples for Monte Carlo Localization is to select an x and y coordinate pair anywhere within the environment. However, such unconstrained placement would create invalid samples with a higher likelihood. Instead, in the boxed approach, the randomly selected x and y coordinate pair for node n is constrained by the location of previously generated samples. The algorithm for this process is shown in Algorithm 3. The goal of this process is to first iterate through all previously placed nodes (n'). If n' is a neighbor of n, then we know that n must be located within the communication range (r) of n'. To describe this constraint, we update a set of constraints ($\min_x, \min_y, \max_x, \max_y$). These four constraints form a rectangular or boxed area in Euclidean space where samples should be generated from to minimize the likelihood of invalid sample selection. While the constraints $\min_x$ and $\max_x$ define the range across the x-axis, constraints $\min_y$ and $\max_y$ describe the range on the y-axis. Once all previously placed nodes are considered, we use these constraints to generate the final selected coordinate pair ($\widehat{p}$) through the use of a uniform random function $\mathbb{U}(\cdot)$. It is possible that $\min_x > \max_x$ or $\min_y > \max_y$, in which case, the constraints suggest that there is no valid position for node n based on the previously placed nodes. In this case, the sample p is marked invalid and tossed out.

Algorithm 3 Generate_Random_Sample (Boxed Method).

```
1: $\min_x, \min_y = -\infty, -\infty$
2: $\max_x, \max_y = \infty, \infty$
3: for all $n' \in \{1 \dots n-1\}$ s.t. $n \le \|K\|$ do
4:   if $M_t^{(n,n')} \neq 0$ then
5:     $min_x \leftarrow max(min_x, p_x^{(n')} - r)$
6:     $max_x \leftarrow min(max_x, p_x^{(n')} + r)$
7:     $min_y \leftarrow max(min_y, p_y^{(n')} - r)$
8:     $max_y \leftarrow min(max_y, p_y^{(n')} + r)$
9: if $\min_x < \max_x$ and $\min_y < \max_y$ then
10:   $\widehat{p} \leftarrow \{\mathbb{U}(min_x, max_x), \mathbb{U}(min_y, max_y)\}$
11: else
12:   $\widehat{p} \leftarrow \emptyset$
13: return $\widehat{p}$
```

The initialization step attempts to generate τ_{init} samples, however as some of the attempts may fail, by the end of the initialization step, $\|\mathbf{S}\| \le \tau_{init}$. After the initialization step is completed, we repeat the sampling and filtering steps for each future time instance.

5.2.2. Sampling step (move samples)

Algorithm 4 Sampling_Step

```
1: $\mathbf{S}' \leftarrow \{\}$
2: for all $p \in \mathbf{S}$ do
3:   for all $i \in \{1 \dots \tau_{children}\}$ do
4:     $p' \leftarrow \{\}$
5:     valid $\leftarrow$ TRUE
6:     for all $n \in p$ do
7:       $\Delta \leftarrow \mathbb{U}(1, v_{max})$
8:       $\theta \leftarrow \mathbb{U}(0, 2\pi)$
9:       $p'^{(n)}_{prev} \leftarrow p^{(n)}$
10:      $p'^{(n)} \leftarrow p^{(n)} + \{\Delta\cos\theta, \Delta\sin\theta\}$
11:    for all $n \in K$ do
12:      if $n \notin p'_{prev}$ then
13:        $p'^{(n)} \leftarrow Generate\_Random\_Sample(p', n)$
14:        if $p'^{(n)} = \emptyset$ then
15:          valid $\leftarrow$ FALSE
16:    if valid then
17:      $\mathbf{S}' \leftarrow \mathbf{S} \bigcup p'$
18: return $\mathbf{S}'$
```

In the sampling step, the goal is to take the set of samples $\mathbf{S}$ from $t - 1$ and move each neighbor n so that M_t continues to be satisfied at time t. Creating new samples based on samples from previous time instances is common to other MCL methods, however, because our goal is unique by using PDI, the method which we accomplish this is novel to MCL. Similar to existing MCL algorithms, in this step, each neighbor is moved at most $v_{\max}$ distance from its previous location. The algorithm for applying this movement to all subsamples is shown in Algorithm 4. We can see that for each preexisting parent sample in $\mathbf{S}$, the algorithm attempts to generate some number ($\tau_{children}$) of child samples. The randomly predicted movement of the node (Δ) is constrained by $v_{\max}$ while θ denotes a randomly predicted angle of movement for the node. Each sample also keeps a memory of the previous position ($p'^{(n)}_{prev}$) for validation in the filtering step. If a node $n \in K$ did not exist in the previous time instance (e.g. $n \notin p'_{prev}$), then $p'^{(n)}$ is generated as in the initialization step.

5.2.3. Filtering step

Algorithm 5 Filtering_Step

```
1: $\widehat{\mathbf{S}} \leftarrow \{\}$
2: for all $p \in \mathbf{S}'$ do
3:   valid $\leftarrow$ TRUE
4:   for all $n \in K$ do
5:     for all $n' \in K$ do
6:       $d \leftarrow \left\|p^{(n)} - p^{(n')}\right\|$
7:       $\overline{d} \leftarrow \left\|p^{(n)}_{prev} - p^{(n')}_{prev}\right\|$
8:       if $M_t^{(n,n')} = 0$ and $d < r$ then
9:       valid $\leftarrow$ FALSE
10:        if $M_t^{(n,n')} = 1$ and $d > \overline{d}$ then
11:        valid $\leftarrow$ FALSE
12:        if $M_t^{(n,n')} = 2$ and $d < \overline{d}$ then
13:          valid $\leftarrow$ FALSE
14:    if valid then
15:      $\widehat{\mathbf{S}} \leftarrow \widehat{\mathbf{S}} \bigcup p$
16: return $\widehat{\mathbf{S}}$
```

After moving samples in the sampling step, we check the validity of each of the newly generated child samples. Four cases exist: first, the case when a neighbor keeps the same state from $t-1$ to t. The second case is when a neighbor is visible (retreating) at $t - 1$ and then is invisible at t. In this case, we must ensure that our sample point exits the range of the given paired node. The third case is when a neighbor is invisible at $t-1$, but visible at t (approaching). In this case, we do not have any predictions of where the node was at $t-1$, so we do not consider this previous information. Instead, we simply randomly select a point which satisfies all conditions in M_t. Finally, we handle the simple case of transitioning from *approaching* to *retreating* or *retreating* to *approaching*. Algorithm 5 shows the mechanism used to validate each sample set $p \in \mathbf{S}'$. For each pair of neighbors (n, n') in K, the current prospective distance (d) and the previous distance ($\overline{d}$) are used to validate the conditions from $M_t^{(n,n')}$. If any single condition fails, the sample is invalid and ignored. We note that if n or n' are not present in p_{prev}, then we set $\overline{d} = \infty$ because the predicted distance between the two pairs was unknown in the previous time instance.

5.2.4. Final predictions

The final predictions for each node relative to its neighbors are determined based on each computed sample in $\mathbf{S}'$. To this end, for each pair of nodes $(n, n') \in K$ where K is a list of neighbors and $i \neq j$, we simply take the average of distances between the sample points of these nodes in all samples:

$$d_{pred}^{(n,n')} = \frac{1}{\|\mathbf{S}'\|} \sum_{s \in \mathbf{S}'} \left\|p^{(n)} - p^{(n')}\right\| \tag{3}$$

where $\left\|p^{(n)} - p^{(n')}\right\|$ is the Euclidean distance between sample points $p^{(n)}$ and $p^{(n')}$ (i.e., the locations of nodes n and n', respectively in sample p).

5.2.5. Complexity analysis

With each process in the TrinaryMC algorithm discussed, we now consider the complexity of the proposed algorithm.

In the first iteration of the Monte Carlo algorithm, the initialization step will produce at most τ_{init} samples in $\mathbf{S}$. As such, after the sampling step is performed, $\|\mathbf{S}'\| \le \tau_{init} \cdot \tau_{children}$ because each of the τ_{init}

parent samples can generate at most $\tau_{children}$ children candidate samples. In subsequent calls to the Monte Carlo algorithm, the maximum number of samples retained is $\|\mathbf{S}'\| \le \tau_{retained}$ based on the threshold noted in Algorithm 1. In this case, by the end of the sampling step, $\|\mathbf{S}'\| \le \tau_{retained} \cdot \tau_{children}$. After the sampling step, both the $\tau_{retained}$ thresholding step and the filtering step only decrease the number of samples stored, thus never increase memory requirements. Knowing this, we see that the memory requirements of the algorithm are constrained by these designated thresholds: $\tau_{retained}, \tau_{init}, \tau_{children}$. Specifically, given a node with $\|K\|$ neighbors, the memory requirement for storing all samples for a single call to the TrinaryMC algorithm is then $O\left(\|K\| \cdot \|\mathbf{S}'\| \cdot \tau_{children}\right)$, where $\|\mathbf{S}'\| = max(\tau_{init}, \tau_{retained})$. Each subsequent call to TrinaryMC only stores samples for the current time instance and the previous time instance, so performing the algorithm over many time instances does not itself increase the memory requirements.

Next, we consider the time complexity of each step in the proposed algorithm. The initialization step generates τ_{init} number of samples (p) of size $\|K\|$. Each subsequently generated sample $p^{(n)} \in p$ considers all previously generated samples in p. As such, generating a single p takes $O\left(\|K\|^2\right)$ and the initialization step in total takes $O\left(\tau_{init} \cdot \|K\|^2\right)$. As noted, the sampling step begins with at most $O\left(\|\mathbf{S}'\|\right) = O\left(max(\tau_{init}, \tau_{retained})\right)$ samples. Thus, the time complexity of the sampling step is the same as the memory requirements: $O\left(\tau_{children} \cdot \|K\| \cdot \|\mathbf{S}'\|\right)$. After sampling, a random subset of at most $\tau_{retained}$ samples from $\mathbf{S}'$ are selected for the filtering step. The filtering step checks the validity of all $\|K\|^2$ node pairs per sample, thus the filtering step takes $O\left(\tau_{retained} \cdot \|K\|^2\right)$. The final prediction step creates a distance prediction for all $\|K\|^2$ pairs of neighboring nodes using all $\|\mathbf{S}'\|$ generated samples. As such, this prediction step takes the same time as the filtering step: $O\left(\tau_{retained} \cdot \|K\|^2\right)$. Thus, the total time complexity for the algorithm is:

$$O\left(\underbrace{\tau_{init} \cdot \|K\|^2}_{\text{Initialization Step}} + \underbrace{\tau_{children} \cdot \|K\| \cdot \|\mathbf{S}'\|}_{\text{Sampling Step}} + \underbrace{\tau_{retained} \cdot \|K\|^2}_{\text{Filtering and Prediction}}\right).$$

With $\|\mathbf{S}'\| = max(\tau_{init}, \tau_{retained})$, the initialization step and the filtering and prediction steps can be combined into a single component, $O\left(\|\mathbf{S}'\| \cdot \|K\|^2\right)$, and as such, the total time complexity for the algorithm can be reduced to $O\left(\|\mathbf{S}'\| \cdot \|K\|^2 + \tau_{children} \cdot \|K\| \cdot \|\mathbf{S}'\|\right)$ which can be further reduced to a final time complexity of $O\left(\|\mathbf{S}'\| \cdot \|K\| \cdot (\|K\| + \tau_{children})\right)$.

5.3. Generating samples

As the general steps taken by the entire algorithm have been discussed, we now provide an illustration of our method for generating samples. Fig. 6 provides an example case which we will use to illustrate the first iteration of our TrinaryMC algorithm.

We begin by assuming the role of a node A with a neighborhood (K) of size 3, including itself. In the figure, we see the values of M_t on the left. With M_t, we begin the process of creating a sample by randomly selecting positions for each neighbor in K. In the first step, we place A by default at $(0, 0)$, the local center. In the second step, we place B randomly. Because $M_t^{(A,B)} = 1$, the placement of B must be within the communication radius of A. In the illustration, this is successful, but in the event of a failure (at this step or any further steps in this process), the sample is marked invalid. Now that B has been placed, we move to the third step by placing C following the conditions specified in M_t. We can see, because $M_t^{(A,C)} = 0$, C is not a 1-hop neighbor of A. However, because $M_t^{(A,B)} \neq 0$ and $M_t^{(B,C)} \neq 0$ we can recognize that A and C are instead 2-hop neighbors. Thus, we can successfully complete this step by placing C in range of only B. After all conditions in M_t are considered, we have created a single sample for $\mathbf{S}$.

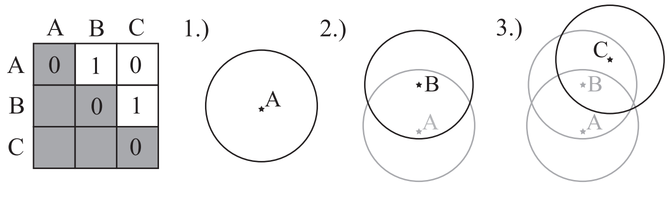

Fig. 6. Process taken to generate a single sample given a state matrix $M_t^{(A)}$ shown on the left containing $3k$-hop (where $k = 2$) neighbors A, B and C. The process takes 3 steps, one for each neighbor. In the first step, the current source node A is placed at position $(0, 0)$. The second step places neighbor B within the communication radius of A because $M_t^{(A,B)} = 1$ indicates A can see (i.e., is in range of) B. The third step places C based on the fact that B sees C, but A does not see C.

In subsequent steps of the algorithm, we take $\mathbf{S}$ from time t to further create $\mathbf{S}'$ for time $t+1$ adhering to the constraints of M_{t+1}. To accomplish this, we simply loop through each neighbor (n) in K_{t+1}. If the neighbor was also in K_t, then we use the coordinate value (x, y) from $\mathbf{S}$ to produce n in $\mathbf{S}'$. The new coordinate must be within a radius of $v_{\max}$ of (x, y) in addition to adhering to the PDI conditions of the pair between t and $t + 1$. We can see this case in Fig. 7. Assume we want to find a new valid position for C given A and B at time t. The shaded area around C in the illustration represents the maximum distance C could have traveled from time $t-1$ to t. While M_{t-1} shows that at time $t-1$, C must be seen by only B, at time t, C must be seen by both A and B. In standard MCL methods, the new position for C would be anywhere in the intersection of A and B. However, in our TrinaryMC approach, C must also be *approaching* B; thus, the region in which valid samples can be taken is dictated not only by the intersection of A and B, but also by the distance between B and C at both time steps. As shown in the illustration, this reduces the valid sampling area for C greatly.

Existing localization algorithms (Znaid et al., 2017b; Bai, 2017; Wang et al., 2010) consider only binary contact information when determining the position of a given node, however each of these algorithms can be augmented to accept PDI. As we showed PDI is recognizable with a high accuracy from noisy RSSI, our next question is recognizing what improvements PDI can offer on standard contact-based approaches? To answer this question, we begin by developing a simulation with four nodes $\{A, B, C, D\}$ placed at the arbitrary (x, y)-coordinates: $(0, 0), (0.70, 0), (0.25, 0.70), (0.20, 0.35)$. These points represent the known positions for each node at time t (these can be computed in a previous iteration of the localization algorithm). Assuming we want to recognize the possible positions of D at time $t+1$, we must recognize the relationship between D and each other node in the network. With communication radius $r = 1.0$, at time $t = 1$, D is visible to all other nodes. We denote the *contact relationship* $z_{contact}$ between two nodes X and Y in the form $X(Y) = z_{contact}$ where $z_{contact}$ can be either *visible* (1) or *invisible* (0). For a *PDI relationship* z_{PDI}, we denote this relationship similarly in the form of $X(Y) = z_{PDI}$ where z_{PDI} can be any of *invisible (0)*, *approaching (1)* or *retreating (2)*.

Given this arrangement, we ask, if D could move to any possible position at time $t+1$, what are the possible states the network could reach? In Fig. 8 (left), we illustrate the answer to this question when considering only binary contacts. If we know at time $t+1$ that A$(D) = 1$, B$(D) = 1$, C$(D) = 1$ then D must remain in the center gray area labeled "1, 1, 1". Alternatively, supposing we knew that only B can see D at time $t+1$, then the current relationship state would be $A(D), B(D), C(D) = 0, 1, 0$ and thus D must reside in the bottom-right yellow area labeled 0, 1, 0. We note that 7 unique contact-based areas exist in this structure, however these areas are rather vast, meaning that there is a large area where D could be for any of these regions. This is where our PDI approach brings its benefits.

If we consider the single central area denoted 1, 1, 1 in Fig. 8 (left), our PDI approach can split this single parent-area into 7 unique sub-areas as illustrated in Fig. 8 (right) thus reducing the possible area where D can be positioned at time $t + 1$. In fact, the largest sub-area (2, 1, 2) takes up only 36.75% the area of the full contact-based parent-

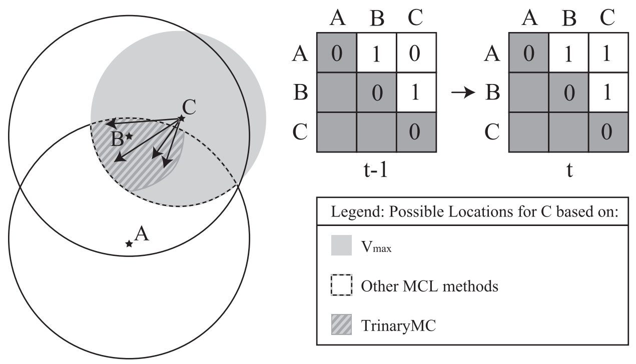

Fig. 7. Possible movement for neighbor C based on $v_{\max}$ is shown in gray, but not all of these locations are possible given the conditions in M_{t+1}. Any point in the area with the dashed perimeter (intersection of circles A, B and the $v_{\max}$ area) is valid for C's next location if only the binary contact matrix is considered as in the case of other MCL methods. Our TrinaryMC method further reduces the valid areas where C can move thanks to the consideration of three PDI states; thus, it produces more accurate samples.

Table 3
Percentage of area reduction when using PDI (versus binary contact).

A(D)	B(D)	C(D)	Percentage Reduction
1	1	1	Not possible
1	1	2	87.0%
1	2	1	99.9%
1	2	2	78.7%
2	1	1	96.4%
2	1	2	63.2%
2	2	1	79.1%
2	2	2	95.6%

area while $2, 1, 1$ takes up an even smaller 3.58%. This means if we can recognize D is *retreating* from A ($A(D) = 2$), D is *approaching* B ($B(D) = 1$) and D is *approaching* C ($C(D) = 1$) the valid sampling area for D when using PDI is reduced by more than 96% compared to the binary contact method. Through our simulation, we also find an area $1, 2, 1$ which is less than 0.1% the area of the contact-based parent-area, resulting in a reduction greater than 99%. Table 3 describes the percentage of area reduction for each sub-area when using PDI. We note $1, 1, 1$ is marked as *not possible* simply because given the initial position of D, D cannot simultaneously approach A, B and C by moving in any direction from its initial position at time t.

One sub-area $(2, 2, 2)$, is split into three separate segments in this illustration. Because these areas are disjoint, issues may arise for existing localization algorithms because each area is so far apart from one another. Selecting the average position for a node in this case would result in an invalid final prediction. One way to address this could be found in an existing contact-based localization algorithm (MacLean and Datta, 2013) where the algorithm opts to filter any segments whose area is less than a threshold (in their case, when the area of the segment is less than $1/3$ the area of the largest segment).

In the worst case we can recognize that the valid area for the contact-based method ($A_{contact}$) is equal to the area through our PDI method (A_{PDI}). This case can occur when two nodes are placed directly on the edge of the communication range of each node. In this case, the only movement option which results in visibility for the nodes would be to approach each other. This is because if they retreat, they would no longer be visible to one another. Thus, because $A_{contact} = A_{PDI}^{(approaching)} + A_{PDI}^{(retreating)}$, if $A_{contact} = A_{PDI}^{(approaching)}$ then $A_{PDI}^{(retreating)} = 0$. This means that if the nodes move closer to each other by a very small amount, then in the next time instance $A_{PDI}^{(retreating)}$ becomes only slightly larger than 0. As a result, we have $0 \leq A_{PDI} \leq A_{contact}$. This then concludes that PDI improves sampling for contact-based localization algorithms by reducing the valid sampling area for the algorithm and the area provided by PDI is never greater than the original contact-based area.

6. Simulations

In this section, we evaluate our PDI enabled TrinaryMC algorithm through simulations. Our simulations use a 500 m $\times$ 500 m area where 50 mobile nodes move using a random waypoint mobility model with a velocity randomly assigned from [1–10] m/s. We use $k = 2$ as a default value in TrinaryMC algorithm, however, we also explore the impact of

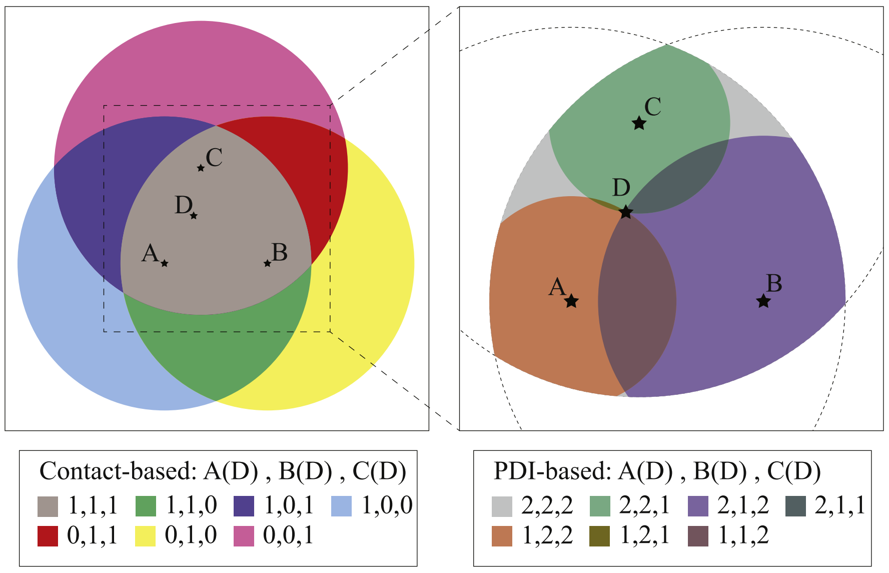

Fig. 8. Given four points A, B, C, D, we derive the following plots if D moves at time $t + 1$. (left) Plot showing the possible locations for D depending on the recognized contact from $A(D), B(D), C(D)$. (right) Showing how contact $A(D), B(D), C(D) = 111$ can be further reduced through the use of PDI (view in color). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

Table 4
Simulation parameters.

Parameter	Value
N	50 nodes
r	50 m
$v_{\max}$	10 m/s
$\tau_{retained}$	100
τ_{init}	100
$\tau_{children}$	10

different values of k. The values of other parameters used in simulations are given in Table 4. Simulation experiments were run on a laptop with an Intel Core i7 2.2 GHz Quad-Core Processor and 16 GB of RAM. The codebase used to perform Monte Carlo Localization simulations is made publicly available.[5]

6.1. Algorithms for comparison

We compare our method to three state-of-the-art MCL techniques namely the Standard Monte Carlo Localization algorithm (St-MCL) (Alaybeyoglu, 2015) (described in section 5.1), Orbit (Orbit-MCL) (MacLean and Datta, 2013) and finally Low Communication Cost MCL (LCC-MCL) (Abu znaid et al., 2017). In addition to comparing our trinary method to existing MCL methods, we also compare it to its variants (i.e., binary-no-memory and the binary methods) that will be described in Section 6.1.3. We describe these unique features of each of these methods below.

6.1.1. Orbit (Orbit-MCL)

The most common improvements made to MCL algorithms is to reduce the possible sampling area with the goal of both decreasing computation time by reducing possible locations and also reducing error by removing invalid samples. One work employing this method is (MacLean and Datta, 2013) where the authors consider graph theory to develop a new MCL method called Orbit. One of the key features they recognize is that when considering negative information (node A cannot see B), there are cases when multiple disjoint regions may occur in the sampling stage. In this work, they determine that ignoring regions with fewer than one-third the number of samples as the largest region reduces sampling area and produces much higher accuracy.

6.1.2. Low Communication Cost Monte Carlo (LCC-MCL)

Previous methods rely exclusively on anchor nodes to inform their predictions, however the LCC-MCL method (Abu znaid et al., 2017) considers not only anchor nodes but also normal, non-anchor nodes. Non-anchor nodes share their own predicted location with neighboring nodes which then use it to derive their own location. The issue with this is that sharing predicted locations of each and every node requires high numbers of packets to update the state between neighbors. The authors mitigate this issue by only sharing locations with neighbors which are believed to be close by. Determining whether nodes are close by one another is a matter of finding the intersection of the set of neighbors for each node. If the number of intersecting neighbors is above a threshold, the nodes are considered close by, in which case, communication occurs.

6.1.3. TrinaryMC derivatives

Our *trinary* algorithm can be simplified in a couple of ways. We use these simplified algorithms to further evaluate our trinary approach later on. First, we consider a simple case where we assume no memory exists between $t-1$ and t for each node. In this case, we have no reason

[5] https://github.com/StevenMHernandez/TrinaryMC.

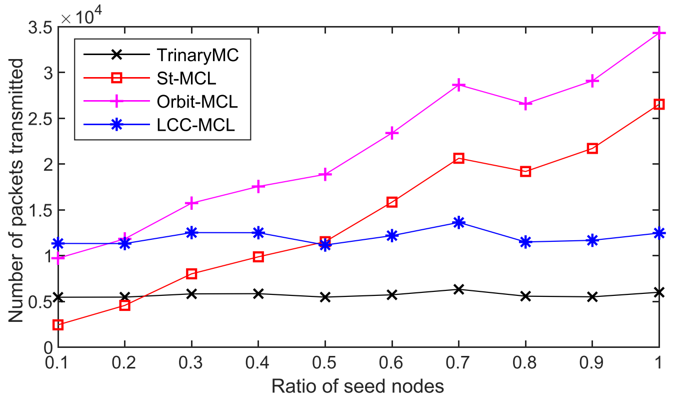

Fig. 9. As the ratio of anchor seed nodes increases, the number of packets communicated through the network also increases except for our TrinaryMC method and the LCC-MCL method.

to move samples or resample. Thus, this first simplified algorithm only runs the initialization step at each time instance no matter if $|\mathbf{S}| > 0$ or not in $t-1$. We call this the *binary-no-memory* method, binary because without memory of $t-1$, neither *approaching* nor *retreating* can be recognized. A middle ground algorithm between this *binary-no-memory* algorithm and our *trinary* algorithm is running the *trinary* algorithm using only the binary states *visible* and *invisible*. For this, we continue to remember the movement of samples over time, thus we retain the sampling and filtering steps. We call this second simplified algorithm as the *binary* method.

6.2. Results

First, we consider the communication required for each method. The primary gain in this aspect for our TrinaryMC method compared to existing methods is that our method needs to share message with first and second hop neighbors only when a trinary state change occurs between a pair of devices. In St-MCL, updates are shared between each k-hop neighbor whenever an anchor node receives a new GPS position. Because we simulate time discretely, all anchor nodes move every single time instance. The Orbit-MCL and LCC-MCL methods not only require communication for each anchor position change but also require updates from non-anchor nodes to provide additional accuracy in their methods. Only LCC-MCL considers this additional communication and provides a method to lower this cost. Instead of sharing GPS position updates from all anchor and non-anchor nodes to all neighboring nodes, a closeness metric is obtained to determine whether a given neighbor is close enough to gain insight from the new GPS position. As the ratio of anchor seed nodes increases, the amount of communication required also increases for all methods except for our TrinaryMC and LCC-MCL as can be seen in Fig. 9. For TrinaryMC, this is because the model does not consider anchors whatsoever. For LCC-MCL this occurs as a result of the use of the closeness metric when determining which nodes to communicate with instead of simply whether the node is an anchor or not. This closeness metric does not change for LCC-MCL as more nodes become anchor seeds.

Next we consider the effects of communication radius (r). As r increases, more neighbors on average are seen within k-hop range of each node, as shown in Fig. 10. Note that, when r is smaller than 30 m, the number of one-hop neighbors on average is smaller than 1. In these cases, as we cannot make predictions without any data from neighbors, we ignore them in our analysis. Our expectation is that increasing the number of neighboring nodes will reduce the valid sampling area which thus means a decrease in overall error. For evaluating the error produced by each method in our simulations, we take the difference of the predicted distance and the actual distance for all node pairs.

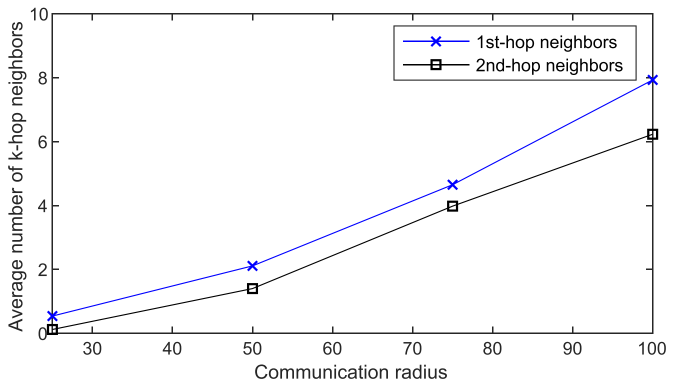

Fig. 10. Average number of *k*-hop neighbors seen given some communication radius (*r*).

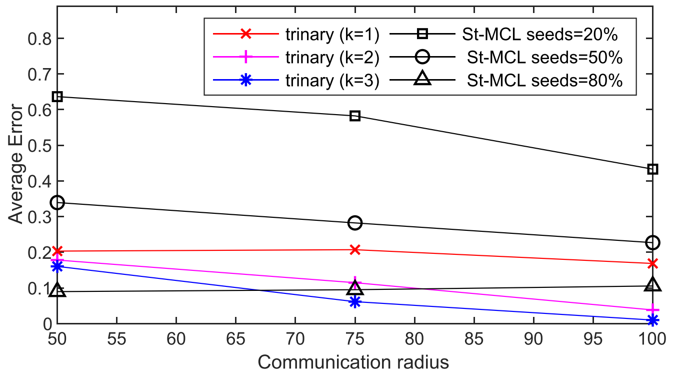

Fig. 12. TrinaryMC method comparing to the Standard Monte Carlo Localization (St-MCL) methods for different number of anchor seeds in the network.

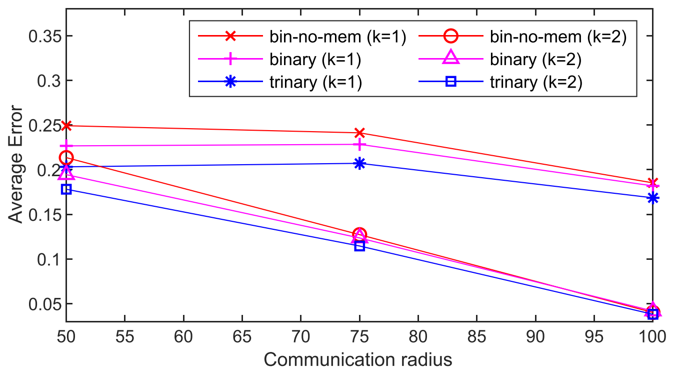

Fig. 11. Comparison of our methods as communication radius changes (thus more neighbors are seen per node) and as *k* increases (also causing more neighbors to be seen).

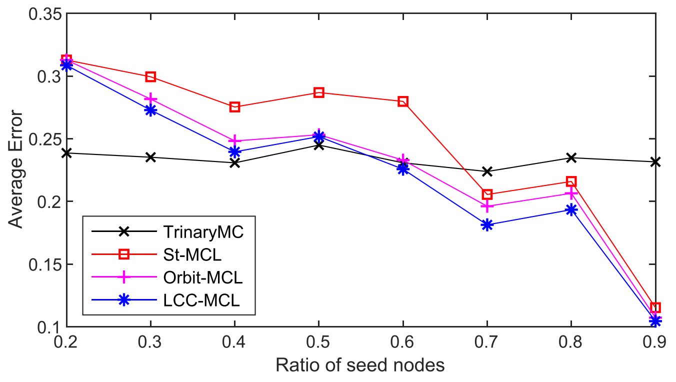

Fig. 13. Comparison of TrinaryMC to state-of-the-art MCL methods when different percentage of the nodes in the network are marked as anchor nodes and communication radius, $r = 50$, and $k = 2$.

Fig. 11 shows the error obtained in our three methods, *binary no-memory*, *binary* and *trinary*. We take the calculated average error per node after 100 time steps divided by the communication radius (r) to produce a percent error relative to r. We can see that as radius increases, error decreases, which can be attributed to the fact that more neighbors are seen with higher r. Further, we can see when $k = 2$, we achieve better accuracy than we do with $k = 1$. We can attribute this increase in accuracy again to more neighbors, which helps create fewer possible locations when generating samples. We also observe that our trinary method provides lower error than the binary method, which itself produces lower error than the binary no-memory method.

Existing MCL algorithms require anchor nodes or *seed* nodes with knowledge of their exact location at any given time through a method such as GPS. As more seeds are added, these anchor-based methods are expected to become more accurate, however with an increase in hardware cost and energy usage. Because our method is anchorless, TrinaryMC is not affected by an increase of seeds. We explore how changing the percentage of seed nodes within the network affects St-MCL and how TrinaryMC compares to St-MCL as a result in Fig. 12. We can see when only 20% of nodes are labeled as seeds, accuracy decreases significantly, ranging from 40% to 70% of r. However, with 50% of nodes as seeds, we begin to reach the same average error as in our trinary case. Another observation we can make from this figure shows that as communication radius for nodes increases, average error decreases. This is explained by the fact that a larger communication radius reveals more neighbors which thus allows for a more restricted area when predicting samples.

Next, with $r = 50$ and $k = 2$, Fig. 13 compares our algorithm with additional state-of-the-art methods. We again see the trend where a lower ratio of seeds (below 50%) produces worse accuracy results than TrinaryMC for each of the existing state-of-the-art algorithms. We also notice that both Orbit-MCL and LCC-MCL perform better than St-MCL when the ratio is between 0.3 and 0.7, but perform similarly to one another. Of course, having a high ratio of seed nodes in the network is not reasonable because of considerations such as hardware cost as well as the issue of battery consumption from modules such as GPS. Thus, even though the solution may provide better results with greater number of seeds, the solution is highly impractical and costly.

We also compare the TrinaryMC algorithm to other algorithms based on cumulative distribution function (CDF) of error with two different ratios of seed nodes. As it is shown in Fig. 14a, with a seed ratio of 0.2, each of the other MCL algorithms performs similarly worse than TrinaryMC. On the other hand, as shown in Fig. 14b, with a ratio of 0.8, all algorithms perform similarly initially then TrinaryMC diverges with worse CDF error results. Similarly, St-MCL diverges from both Orbit-MCL and LCC-MCL between the 0.5–0.9 CDF mark. Both Orbit-MCL and LCC-MCL perform similarly when a high ratio of seed nodes are present.

Next, we compare our method to RSSI based distance estimation. Many existing works use the Log-distance path model (Shue and Conrad, 2017) to explain how RSSI is affected by both distance (d) and noise (n). In works employing this model (Shue and Conrad, 2017; Karvonen et al., 2017), authors set the noise parameter (n) for their model as some static value between 2 and 4. Because the model does not change for a given environment, selecting an incorrect value for n can result in poor distance predictions. To simulate the effect of different values for n on distance predictions, we randomly set a true environmental noise value (n_{true}) along with another randomly assigned predicted noise value (n_{pred}) denoting the value assigned by the model-designer. This means if the model-designer chooses a bad value for n_{pred}, then

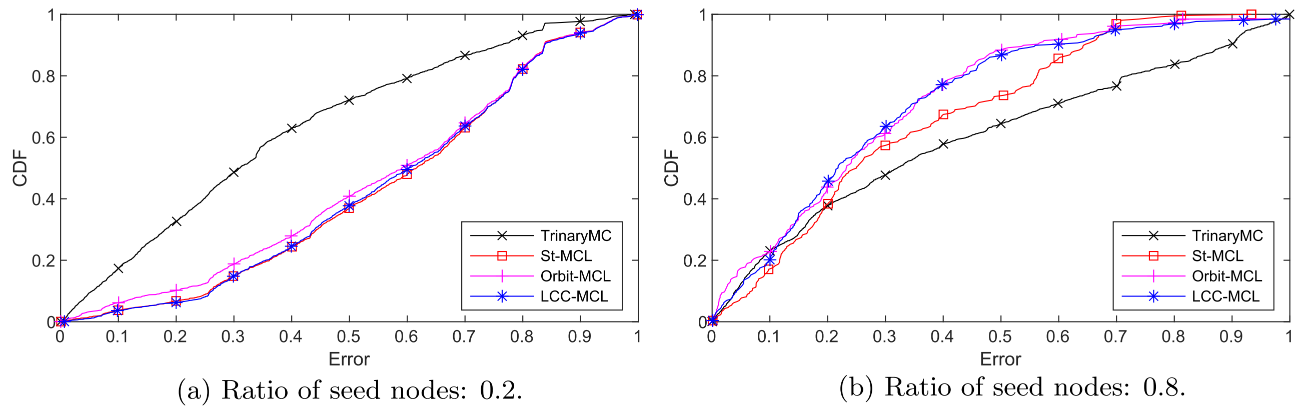

Fig. 14. CDF Error with different ratios of seed nodes.

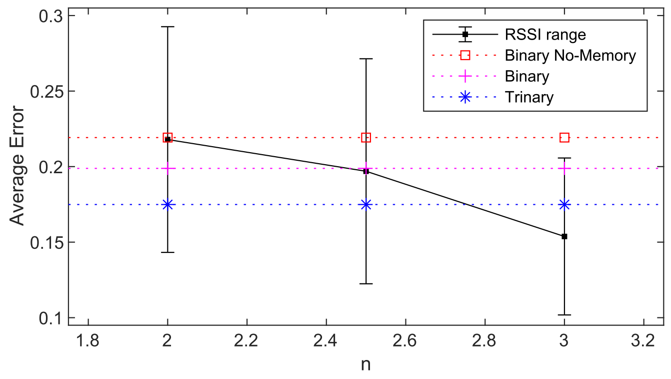

Fig. 15. Results of simulation of RSSI values with different path-loss exponent and variance, and their comparison with our methods.

$|n_{true} - n_{pred}| \gg 0$ and thus distance predictions will subsequently be poor. For this simulation, we calculate the RSSI value based on n_{true}:

$$RSSI = 10 \times n_{true} \times \log(d) + A \tag{4}$$

where A is the baseline RSSI for a pair of devices at a distance of 1 m and d is the actual distance of the devices for some time instance. Using this RSSI value and n_{pred}, we predict distance using the Log-distance path model:

$$d_{pred} = 10^{\frac{A-RSSI}{10 \times n_{pred}}} \tag{5}$$

For both equations, we set $A = -30$ as this was a common value we found through experiments and existing literature. In Fig. 15, we show how different values for n affect the average error. For each value of n we show how well the Log-distance path model performs when n_{true} and n_{pred} are within ± 0.5 of each other (lower error) and when n_{true} and n_{pred} are within ± 1.0 of one another (higher error). We can see the average error with RSSI based distance estimation is equal when $n = 2$ and better on average as n increases than the binary-no-mem method. However, we can see that our trinary method performs better on the average until n reaches 2.8. Still, given a poor choice for n_{pred}, it is possible that RSSI method will still perform worse than both the trinary and binary methods up to $n = 3$. This suggests that our method can perform as good as the RSSI method or better in very uncertain environments such as highly congested areas with more noise or indoor environments with high signal reflections.

While in this study, our focus is the positioning of the nodes relative to each other, it is possible to derive the approximate true locations of all nodes in the network from these relative positions. Thus, to demonstrate the results of TrinaryMC from a full network perspective, we present in Fig. 16a the actual positions and the predicted positions of 15 nodes, each with communication range of 50 units moving within a 100 × 100 unit area. Predicted positions are obtained by performing Multi-Dimensional Scaling (MDS) as described in (Rajan et al., 2019; Abouzar et al., 2016) with our TrinaryMC predicted per-node distances. In the figure, we see very high similarity between the true positions and the predicted positions for this specific example. In order to quantify the expected error of predicted locations in network of further network sizes, we simulate networks with number of nodes $|N| \in$ (Alzantot and Youssef, 2012; Wang et al., 2008). For each value of $|N|$, we calculate the average error for all nodes over 50 simulations with a mobility area of 100 × 100 units and a communication range of 50 units. We see in Fig. 16b that the total MDS error increases as the number of nodes increases. This is expected, because adding more nodes results in more node pairings where additional error may occur. However, if we divide the total MDS error by the number of nodes in each network we can see the average MDS error per node-pair. In this case, we begin to identify a negative trend where an increase of nodes results in a decrease in error per node-pair.

As more neighbors are added, each algorithm performs additional computation when collecting sample sets. Because our TrinaryMC algorithm is designed to perform relative positioning, it is unique when compared to existing non-relative MCL algorithms. In our algorithm, each node must compute sub-samples for each neighbor, while anchor-based MCL algorithms only produce one sub-sample per node regardless of the number of neighbors. We see in Fig. 17a that increasing radius does affect TrinaryMC greater than other methods as a result of this sub-sampling. However, normalizing this computation time by dividing it with the number of computed sub-samples in Fig. 17b reveals that TrinaryMC performs similarly to both St-MCL and LCC-MCL and more efficiently per sub-sample than Orbit. We see in this normalized case that average per node computation increases similarly for each method as r increases. This demonstrates that the increase in computation time for TrinaryMC is dependant solely on the fact that, because TrinaryMC is designed as a relative positioning algorithm, it must compute more information (i.e., position predictions for each neighbor) per sample than anchor-based MCL algorithms which consider only the device's own coordinate per sample. Specifically, this shows that the increase in time to perform one iteration of the TrinaryMC algorithm compared to other algorithms is proportional to the number of neighbors for a given node.

7. Conclusion

In this work, we present an anchorless relative positioning method which utilizes multi-hop neighborhood Perceived Direction Information (PDI) composed of states: *approaching*, *retreating* and *invisible*. We

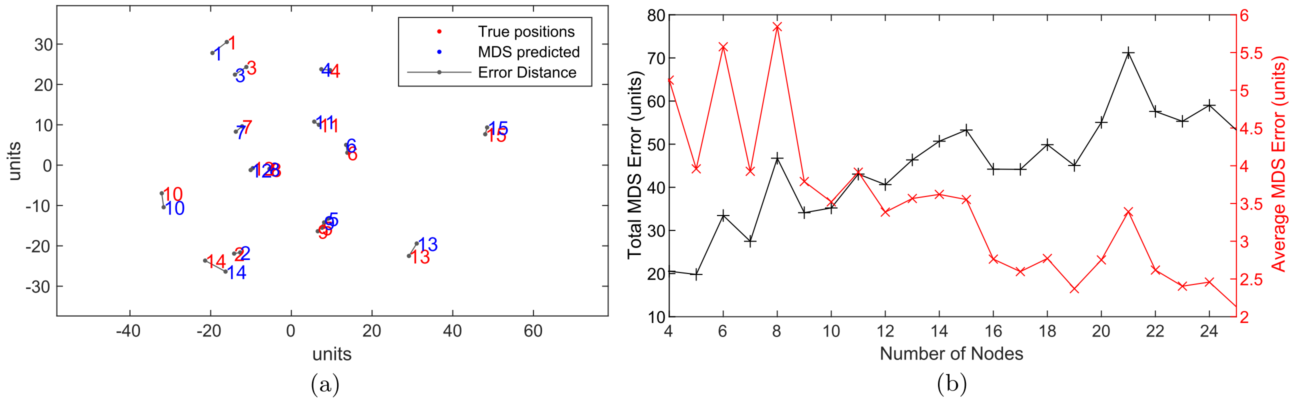

Fig. 16. (a) Example result from the use of MDS on the predictions from TrinaryMC, (b) Average error using TrinaryMC with MDS over 50 simulations with different numbers of nodes.

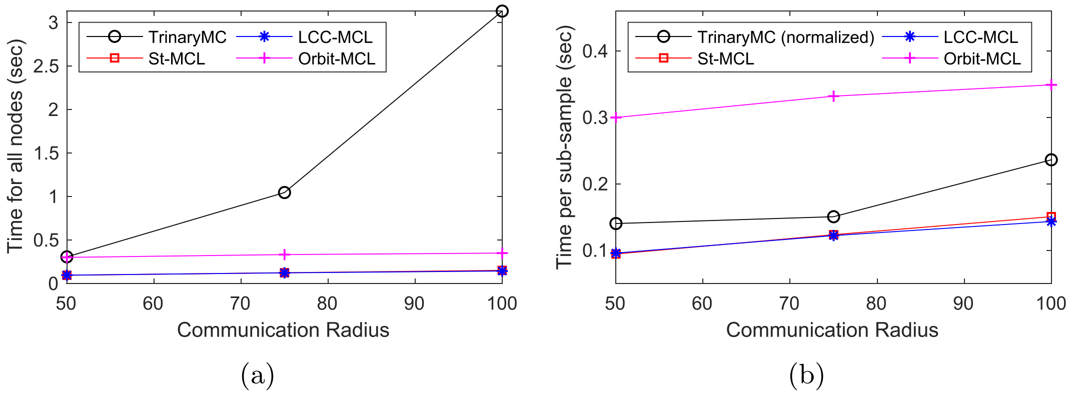

Fig. 17. Time taken to compute each MCL algorithm. (a) Because TrinaryMC is designed for relative positioning rather than localization, TrinaryMC must compute one sub-sample per neighbor resulting in more data and more computation. (b) Normalizing by the number of sub-samples required for each method, TrinaryMC performs more closely with St-MCL and LCC-MCL.

demonstrate issues inherent to the use of RSSI in indoor localization such as amplitude differences across heterogeneous radio hardware. Through our development of a Dense Neural Network for predicting PDI, we find that we can reliably achieve more than 95% accuracy even in networks composed of hardware from unseen manufacturers. We then show how PDI can be used to reduce the sampling area compared to contact-based localization algorithms by up to 99%. Finally we develop a Monte Carlo Localization algorithm using PDI to produce relative positioning with lower error and lower communication overhead than existing Monte Carlo Localization methods which rely on anchor nodes present within the network. Because of our improvement of removing all reliance on anchor nodes, our method can remove energy consumption from the use of GPS radio hardware, decrease hardware cost by removing the need for GPS modules on all devices, and decrease the reliance on the limited individual anchor nodes in a system. Our method benefits as more devices join the system as each not only contributes information about themselves, but also their own *k*-hop neighbors, thus giving each node a further look into the network allowing for more accurate sampling.

CRediT authorship contribution statement

Steven M. Hernandez: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Eyuphan Bulut:** Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1744624. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Abouzar, P., Michelson, D.G., Hamdi, M., 2016. Rssi-based distributed self-localization for wireless sensor networks used in precision agriculture. IEEE Trans. Wireless Commun. 15 (10), 6638–6650, https://doi.org/10.1109/TWC.2016.2586844.
Abu znaid, A.M.A., Idris, M.Y.I., Wahab, A.W.A., Qabajeh, L.K., Mahdi, O.A., 2017. Low communication cost (lcc) scheme for localizing mobile wireless sensor networks. Wireless Network 23 (3), 737–747, https://doi.org/10.1007/s11276-015-1187-6.
Alaybeyoglu, A., 2015. An efficient Monte Carlo-based localization algorithm for mobile wireless sensor networks. Arabian J. Sci. Eng. 40 (5), 1375–1384.
Alzantot, M., Youssef, M., 2012. Crowdinside: automatic construction of indoor floorplans. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems. ACM, pp. 99–108.
Baggio, Aline, Langendoen, Koen, 2008. Monte Carlo localization for mobile wireless sensor networks. Ad Hoc Netw. 6 (5), 718–733.
Bai, E.-W., 2017. Source localization by a binary sensor network in the presence of imperfection, noise, and outliers. IEEE Trans. Automat. Contr. 63 (2), 347–359.

Chabloz, O., ndrade, D.D.S., Upegui, A., Satizbal, H.F., Perez-Uribe, A., 2017. Discoverytree: relative localization based on multi-hop ble beacons. In: 2017 Global Internet of Things Summit (GIoTS), pp. 1–6, https://doi.org/10.1109/GIOTS.2017.8016264.

Chen, H., Gao, F., Martins, M., Huang, P., Liang, J., 2013. Accurate and efficient node localization for mobile sensor networks. Mobile Network. Appl. 18 (1), 141–147.

Chen, D., Shin, K.G., Jiang, Y., Kim, K.-H., 2017. Locating and tracking ble beacons with smartphones. In: Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT 17. ACM, New York, NY, USA, pp. 263–275, https://doi.org/10.1145/3143361.3143385.

S. A. Cheraghi, V. Namboodiri, K. Sinha, Ibeaconmap: Automated Indoor Space Representation for Beacon-Based Wayfinding, arXiv preprint arXiv:1802.05735.

Cho, H., Ji, J., Chen, Z., Park, H., Lee, W., 2015. Me asuring a distance between things with improved accuracy. Proc. Comput. Sci. 52, 1083–1088, https://doi.org/10.1016/j.procs.2015.05.119. the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015) http://www.sciencedirect.com/science/article/pii/S18770 50915009199.

Golestanian, M., Lu, H., Poellabauer, C., Kenney, J., 2018. Rssi-based ranging for pedestrian localization. In: 2018 IEEE 88th Vehicular Technology Conference. VTC-Fall, pp. 1–5, https://doi.org/10.1109/VTCFall.2018.8690714.

Hernandez, S.M., Bulut, E., 2019. TrinaryMC: Monte Carlo based anchorless relative positioning for indoor positioning. In: 2020 17th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, pp. 1–6.

Heurtefeux, K., Valois, F., 2012. Is rssi a good choice for localization in wireless sensor network? In: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications, pp. 732–739, https://doi.org/10.1109/AINA.2012.19.

Karvonen, H., Mikhaylov, K., Hmlinen, M., Iinatti, J., Pomalaza-Rez, C., 2017. Interference of wireless technologies on ble based wbans in hospital scenarios. In: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications. PIMRC, pp. 1–6, https://doi.org/10.1109/PIMRC.2017.8292333.

Khedr, A.M., 2015. New localization technique for mobile wireless sensor networks using sectorized antenna. Int. J. Commun. Netw. Syst. Sci. 8 (9), 329.

Konings, D., Faulkner, N., Alam, F., Noble, F., Lai, E., 2017. Do rssi values reliably map to rss in a localization system? In: 2017 2nd Workshop on Recent Trends in Telecommunications Research. RTTR, pp. 1–5, https://doi.org/10.1109/RTTR.2017.7887867.

Kumar, S., Kumar, R., Rajawat, K., 2016. Cooperative localization of mobile networks via velocity-assisted multidimensional scaling. IEEE Trans. Signal Process. 64 (7), 1744–1758.

Lui, G., Gallagher, T., Li, B., Dempster, A.G., Rizos, C., 2011. Differences in rssi readings made by different wi-fi chipsets: a limitation of wlan localization. In: 2011 International Conference on Localization and GNSS. ICL-GNSS, pp. 53–57, https://doi.org/10.1109/ICL-GNSS.2011.5955283.

Luo, C., Hong, H., Cheng, L., Chan, M.C., Li, J., Ming, Z., 2016. Accuracy-aware wireless indoor localization: feasibility and applications. J. Netw. Comput. Appl. 62, 128–136.

MacLean, S., Datta, S., 2013. Reducing the positional error of connectivity-based positioning algorithms through cooperation between neighbors. IEEE Trans. Mobile Comput. 13 (8), 1868–1882.

Nguyen, H.C., Wigard, J., Kovacs, I.Z., Tavares, F., Mogensen, P., 2017. Evaluation of indoor radio deployment options in high-rise building. In: 2017 IEEE 85th Vehicular Technology Conference. VTC Spring, pp. 1–5, https://doi.org/10.1109/VTCSpring.2017.8108202.

Pease, S.G., Conway, P.P., West, A.A., 2017. Hybrid tof and rssi real-time semantic tracking with an adaptive industrial internet of things architecture. J. Netw. Comput. Appl. 99, 98–109.

Radak, J., Baulig, L., Bijak, D., Schowalter, C., Frey, H., 2017. Moving towards wireless sensors using rssi measurements and particle filtering. In: Proceedings of the 14th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks. ACM, pp. 33–40.

Rajan, R.T., Leus, G., van der Veen, A.-J., 2019. Re lative kinematics of an anchorless network. Signal Process. 157, 266–279, https://doi.org/10.1016/j.sigpro.2018.11.005.

S. V. Ramani, Y. N. Tank, Indoor navigation on google maps and indoor localization using RSS fingerprinting, CoRR abs/1405.5669. arXiv:1405.5669. URL http://arxiv.org/abs/1405.5669.

Sanpechuda, T., Kovavisaruch, L., 2008. A review of rfid localization: applications and techniques. In: 2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, vol. 2, pp. 769–772, https://doi.org/10.1109/ECTICON.2008.4600544.

Santos, J.M., Portugal, D., Rocha, R.P., 2013. An evaluation of 2d slam techniques available in robot operating system. In: 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE, pp. 1–6.

Shue, S., Conrad, J.M., 2017. Procedurally generated environments for simulating rssi-localization applications. In: Proceedings of the 20th Communications & Networking Symposium, CNS 17, Society for Computer Simulation International, San Diego, CA, USA 7:17:11 http://dl.acm.org/citation.cfm?id3107979.3107986.

Solahuddin, Y.F., Mardeni, R., 2011. Indoor empirical path loss prediction model for 2.4 ghz 802.11n network. In: 2011 IEEE International Conference on Control System, Computing and Engineering, pp. 12–17, https://doi.org/10.1109/ICCSCE.2011.6190487.

Subedi, S., Kwon, G., 2016. Seokjoo Shin, Suk-seung Hwang, Jae-Young Pyun, Beacon based indoor positioning system using weighted centroid localization approach. In: 2016 Eighth International Conference on Ubiquitous and Future Networks. ICUFN, pp. 1016–1019, https://doi.org/10.1109/ICUFN.2016.7536951.

Trogh, J., Plets, D., Thielens, A., Martens, L., Joseph, W., 2016. Enhanced indoor location tracking through body shadowing compensation. IEEE Sensor. J. 16 (7), 2105–2114, https://doi.org/10.1109/JSEN.2015.2508002.

Wang, Z., Bulut, E., Szymanski, B.K., 2008. Distributed target tracking with imperfect binary sensor networks. In: IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference, pp. 1–5, https://doi.org/10.1109/GLOCOM.2008.ECP.62.

Wang, Z., Bulut, E., Szymanski, B.K., 2010. Distributed energy-efficient target tracking with binary sensor networks. ACM Trans. Sens. Netw. 6 (4), 32.

Wang, H., Sen, S., Elgohary, A., Farid, M., Youssef, M., Choudhury, R.R., 2012. No need to war-drive: unsupervised indoor localization. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys 12, ACM, New York, NY, USA, pp. 197–210, https://doi.org/10.1145/2307636.2307655.

Wang, X., Gao, L., Mao, S., Pandey, S., 2017. Csi-based fingerprinting for indoor localization: a deep learning approach. IEEE Trans. Veh. Technol. 66 (1), 763–776, https://doi.org/10.1109/TVT.2016.2545523.

Yang, Z., Zhou, Z., Liu, Y., 2013. From rssi to csi: indoor localization via channel response. ACM Comput. Surv. 46 (2), https://doi.org/10.1145/2543581.2543592 25:125:32.

Yoon, J.H., Chung, I., Lee, Y.H., 2019. Location estimation technique in bluetooth beacon based indoor positioning systems. In: Hwang, S.O., Tan, S.Y., Bien, F. (Eds.), Proceedings of the Sixth International Conference on Green and Human Information Technology. Springer Singapore, Singapore, pp. 41–44.

Yuanfeng, D., Dongkai, Y., Huilin, Y., Chundi, X., 2016. Flexible indoor localization and tracking system based on mobile phone. J. Netw. Comput. Appl. 69, 107–116.

Yucel, F., Bulut, E., 2018. Clustered crowd gps for privacy valuing active localization. IEEE Access 6, 23213–23221.

Zhou, J., Shi, J., 2008. Rfid localization algorithms and applicationsa review. J. Intell. Manuf. 20 (6), 695, https://doi.org/10.1007/s10845-008-0158-5.

Zhu, J., Waltho, A., Yang, X., Guo, X., 2007. Multi-radio coexistence: challenges and opportunities. In: 2007 16th International Conference on Computer Communications and Networks, pp. 358–364, https://doi.org/10.1109/ICCCN.2007.4317845.

Zidek, A., Tailor, S., Harle, R., 2018. Bellrock: anonymous proximity beacons from personal devices. In: 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom). IEEE, pp. 1–10.

Znaid, A.M.A.A., Idris, M.Y.I., Wahab, A.W.A., Qabajeh, L.K., Mahdi, O.A., 2017. Sequential Monte Carlo localization methods in mobile wireless sensor networks: a review. J. Sens. 2017, 1–19, https://doi.org/10.1155/2017/1430145.

Znaid, A., Ammar, M., Idris, M., Idna, Y., Abdul Wahab, A.W., Khamis Qabajeh, L., Adil Mahdi, O., 2017. Sequential Monte Carlo localization methods in mobile wireless sensor networks: a review. J. Sens..

Steven M. Hernandez received a B.S. degree in Computer Science from Virginia Commonwealth University in 2018. He is now completing his Ph.D. in the Computer Science Department of Virginia Commonwealth University with funding through the National Science Foundation Graduate Research Fellowship under the supervision of Dr. Eyuphan Bulut. His research interests include Device-to-Device (D2D) communications, machine learning for mobile networks and Wi-Fi sensing. He is a member of IEEE.

Eyuphan Bulut (SM′20) received the Ph.D. degree in the Computer Science department of Rensselaer Polytechnic Institute (RPI), Troy, NY, in 2011. He then worked as a senior engineer in Mobile Internet Technology Group (MITG) group of Cisco Systems in Richardson, TX for 4.5 years. He is now an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University (VCU), Richmond, VA. His research interests include mobile and wireless computing, network security and privacy, mobile social networks and crowd-sensing. Dr. Bulut is an Associate Editor in IEEE Access and has been serving in the organizing committee of several conferences. He is a senior member of IEEE and member of ACM.