BluEyes – Bluetooth Localization and Tracking

Ei Darli Aung, Jonathan Yang, Dae-Ki Cho, Mario Gerla
University of California, Los Angeles

Abstract

With GPS and Wi-Fi triangulation, localization and tracking technology has existed and heavily utilized for the past couple years. However, people do not always carry GPS and Wi-Fi devices on them at all times, making localization not always readily available. With millions of Bluetooth phones being carried around everywhere, Bluetooth has become one of the most accessible wireless technologies carried by a person at any given time. Is it possible then to build a localizing and tracking system using Bluetooth radios?

This paper proposes BluEyes, a system model for localizing and tracking a Bluetooth device by having sensor nodes measure the RSSI of the tracked device and estimating the position of the device by comparing the measured RSSI value to previously learned RSSI values at all positions on the map. We improve the accuracy of the location estimation by calculating the distance of live samples to the previously learned samples and choose the coordinate to which the current samples seem closest. Furthermore, we implement auto-regression on distance calculations to improve accuracy. We present experimental results that demonstrate the ability of BluEyes to track and locate a device with very good accuracy even when the device is mobile.

1 Introduction

Two mainstream technologies are currently being used for device localization and tracking: GPS and Wi-Fi. GPS provides very accurate positioning measurements, but the greatest downside to GPS is that it does not operate in indoor environments. Once indoors, the satellite reception to the device is lost and the device can no longer be located. As a result, GPS has primarily been used only in outdoor environments, and localizing devices indoors has relied on other wireless technology such as Wi-Fi. Though not as precise as GPS, especially when the device is mobile, localization using Wi-Fi has been shown to be quite accurate. However, Wi-Fi devices are still not as widely used in comparison to Bluetooth phones. Wi-Fi laptops are very prevalent but due to their relative large size, laptops tend to stay in static locations and are powered off when mobile. Therefore, Bluetooth phones serve as a perfect candidate to be tracked since they are always on and carried by the user.

A practical application for Bluetooth device tracking is recording customer traces in a retail store. Companies spend considerable time and effort into ensuring that both displays and paths are well designed to attract the attention of customers. Studies have shown that putting displays in high traffic areas in the store affect customer buying patters. In order to evaluate the effectiveness of a display, stores currently hire people to follow customers around the store. The path of the customer, length of time spent in a section of the store, and effect the displays had on the customer are all recorded and evaluated. However, if Bluetooth phones could be located and tracked throughout the store, the need to hire a person to follow the customer around would be eliminated. By putting sensor nodes around the store, a customer's trace could be recorded by tracking the Bluetooth phone carried on the customer. Best of all, customers do not need to interact with the tracking system in any way, making the entire system seamless and nonintrusive.

The remainder of this paper is organized as follows. In Section 2, we survey related work in location determination technologies. Section 3 covers preliminary experiments we performed. Section 4 presents the proposed system model for BluEyes. The results of our system model are shown in Section 5. Finally, we present our conclusion in Section 6.

2 Related Work

There are two related work in the area of user location and tracking that are deemed as works of interest: RADAR and Cricket. Both systems use Wi-Fi RF signals to calculate the location of the device, while Cricket also utilizes ultrasonic pulses to locate the device.

RADAR [1] is a radio frequency based wireless network system for locating and tracking users inside buildings. It uses signal strength information gathered at multiple sensor locations to triangulate the device coordinates. Two models were proposed to determine the location of the node: Empirical Model and Radio Propagation Model.

In the Empirical Model, the system "learns" the signal strengths measured by each sensor node at every coordinate on the map. A global table is created to keep information about the map. The information includes the coordinate, direction of which the device is facing (north, south, east, west), signal strength according to each sensor, and signal to noise ratio according to each sensor. The reason for recording the direction is because the signal strength measured is affected by the direction to which the device is facing, even if the device is at the same location. To track the location of a device, RADAR compares the live data with the learned data stored in the table and the location is determined based on the best match. The downside to the Empirical Model is the significant effort required to construct the learned data set. In addition, the learned data is only applicable to the physical environment in which it is measured. If the system were to change locations, a new set of learned data would be needed.

The Radio Propagation Model was created to avoid the inconvenience created by the Empirical Model. It uses a mathematical model of indoor signal propagation and generates a set of theoretically computed signal strength data similar to the data set in the Empirical Model. While this model greatly reduces the amount of time to generate learned data and can be applied to different physical environments, the learned data is not as accurate as the data generated in the Empirical Model.

Cricket is a decentralized location support system. Unlike most of other location tracking systems or location support systems, cricket uses a combination of radio frequency and ultrasound signals to provide location tracking. It consists of beacons and listeners. A beacon is a small device mountable on a wall or a ceiling, publishing information about the location through RF signal. With each advertisement, the beacon also transmits ultrasound pulse concurrently. A listener is also a small device, which can be attached to any static or mobile device. It listens to messages from beacons and infers its location using those messages. When the listener hears an RF signal, it turns on its ultrasound signal receiver and starts listening to the corresponding ultrasound signal. Upon receiving the ultrasound signal, it calculates the time difference between the receipt of the first bit of RF info and the ultrasound signal to determine the distance to the beacon. While providing accurate location tracking, Cricket requires an add-on infrastructure, albeit at a low cost, to be installed on every tracked device, which makes it unsuitable for certain purposes.

Past research done in Bluetooth localization and tracking has not been encouraging. The inability to have multiple Bluetooth connections caused by the nature of Bluetooth makes triangulation difficult [3]. This is because when triangulating a device, the accuracy depends on if all sensors take a distance measurement at the same time. Because Bluetooth cannot have multiple connections simultaneously, the accuracy of Bluetooth triangulation is compromised due to the nature of Bluetooth. Additionally, past research has stated that the Received Signal Strength Indicator (RSSI) values of Bluetooth are reported to be of little or no use due to the lack resolution and slow update rate. Using RSSI values to measure the location of a device is analogous to RADAR's approach of using signal strength to localize.

The Host Controller Interface (HCI) of every Bluetooth device provides access to three connection status parameters: Link Quality (LQ), RSSI, and Transmit Power Level (TPL). Past research done on using these three parameters for Bluetooth localization [4] found certain parameters useful in correlating the parameter value to distance, while other parameters have been found to be of little use. The following graphs were produced by the research to show their results:

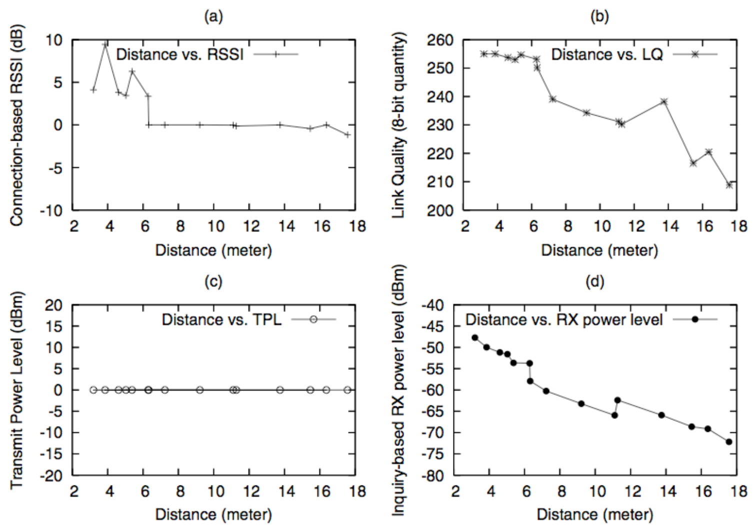

Figure 1: Relationship between various Bluetooth signal parameters & distance.

The results show that both RSSI and TPL correlate poorly to distance while LQ is able to produce a negative linear trend but without much resolution in the front 6 meters. The best parameter that correlates with distance is the RX Power Level, a parameter from which RSSI is calculated. While the RX power level correlates the strongest to distance, it still does not provide a perfectly negative linear relationship, which would cause inaccuracies if used to measure the distance of a device.

There have not been any proposed models of Bluetooth localization and the consensus of past research is that Bluetooth serves poorly for localizing devices because of long latencies and poor accuracy due to the lack of a signal parameter that strongly correlates to distance.

3 Preliminary Experiments

Before proposing a system model, we decided to conduct experiments of measuring RSSI to distance. We begin with a description of our experimental testbed. We then present and discuss the results of our experiments.

3.1 Experimental Testbed

We conduct all our experiments in the long empty corridor on the third floor of Boelter Hall. The Bluetooth devices to serve as sensor/master nodes in our system are Nokia N800 PDAs. The first slave device to be tracked is a LG KG70 phone. The second slave device used is an Apple MacBook Pro. All distances in the experiments are measured in meters.

3.2 RSSI Samples vs. Distance

We decided to conduct our own experiment to study the correlation between RSSI and distance. The PDAs were already equipped with a program that would inquire any Bluetooth devices in its range and measure the RSSI value reported by the device. In separate experiments, we collected RSSI samples of the phone and laptop at every meter for 20 meters. The results are shown in the following graphs:

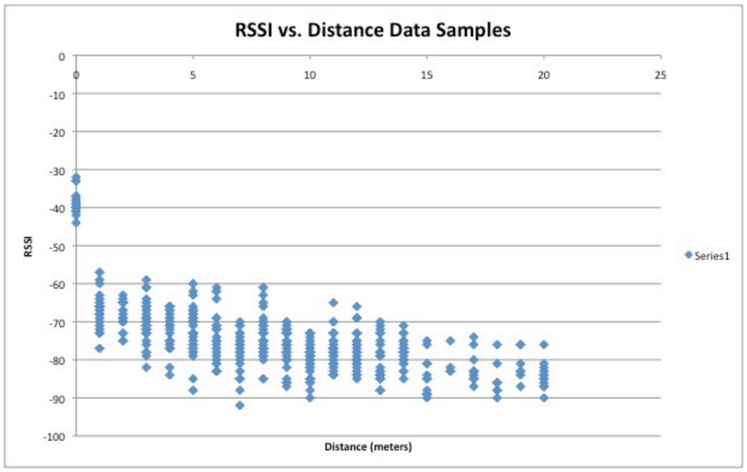

Figure 2: RSSI samples for LG KG70

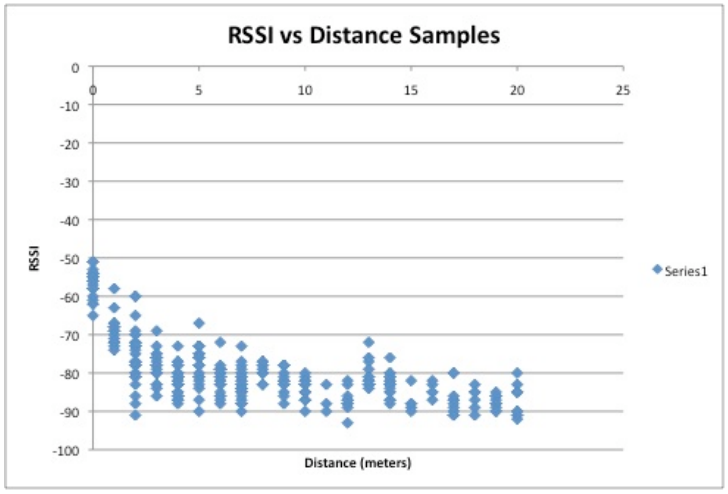

Figure 3: RSSI samples for Apple MacBook Pro

Our results confirm previous research statements that RSSI provides low resolution in the fact that the RSSI values measured at a same location can vary drastically. Therefore, it is not easy to associate a RSSI value to correlate with a specific meter. However, if we average the RSSI values at each meter, the following graphs are produced:

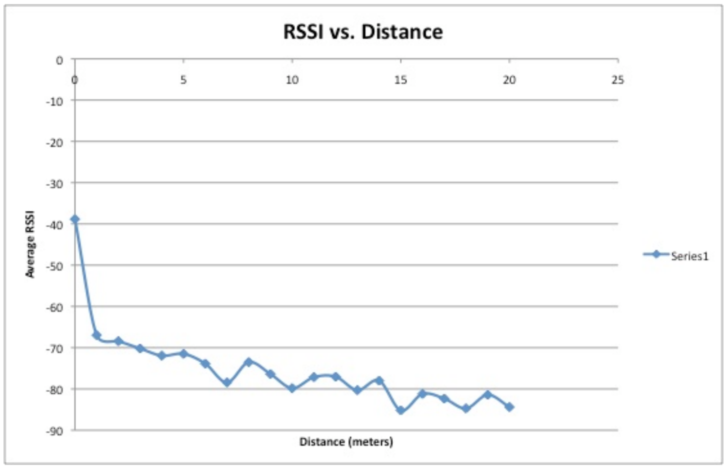

Figure 3: Average RSSI for LG KG70

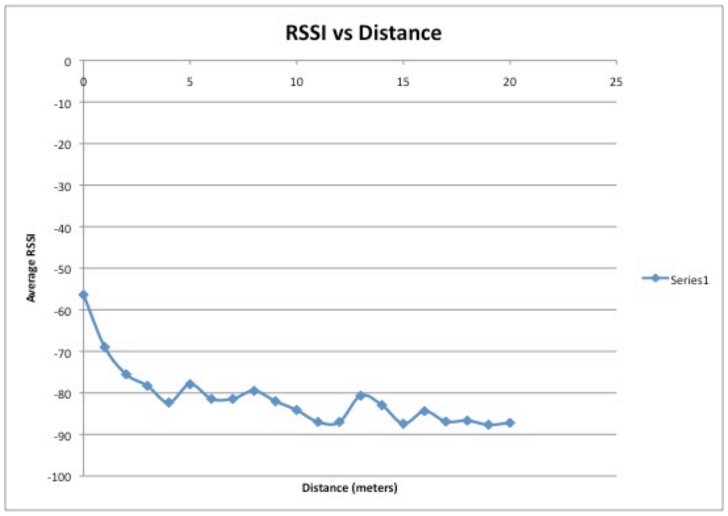

Figure 4: Average RSSI for Apple MacBook Pro

The results clearly show a negative linear trend, although inconsistent at certain meters. While it may be difficult to distinguish a meter with the meter closest to it, it is very likely that a location five or more meters will have a larger difference in the RSSI value.

The graphs also show that even at the same location, the devices show different RSSI values. The phone had most of its RSSI values fall between -70 and -

80 while the laptop had most of its values between -80 and -90. This is because each device has different Bluetooth chips and antennas, affecting the RSSI values measured. Because of the discrepancy between devices, we decided to simplify our model and only focus on tracking the LG cell phone in our later experiments.

To ensure that RSSI measurements do not differ among master nodes, although they are both Nokia N800s, we measured the average RSSI value for each master node measuring the cell phone from 5 meters away. Both master nodes measured -64, which confirms that Bluetooth devices of the same model measure consistent RSSI values.

4 System Model

Assumptions

Our system was built based on the following assumptions:

- We only have 2 sensor nodes and are tracking only 1 slave device
- Slave device travels along a straight line between the two sensor nodes
- Since sensor nodes are placed 10 meters apart, we only store learned RSSI values for up to 10 meters range

Database

Our system includes a database to store learned RSSI values and range values in meters, sensor nodes and a server. We only record learned RSSI values for up to 10 meters, since that is how far the sensor nodes are placed from each other.

The database has 2 tables:

Sensors(ID, Distance, RSSI)
Average(Distance, Avg)

Sensor node

Sensor node scans Bluetooth enabled devices and collects RSSI values along with the MAC address continuously. Then it periodically sends sensor MAC, slave MAC, RSSI values and timestamp, grouped by slave MAC. In our experiment, since we are tracking only one specific device, sensor node filters out RSSI values received from MAC address other than the MAC address of the device we are tracking and sends a packet of 5 RSSI values at a time. The packet format is as follows:

[Sensor MAC Address] [Mode (learn or scan)] [Meter (if in learn mode)]
[Timestamp] [Device MAC Address] [RSSI Value]
[Timestamp] [Device MAC Address] [RSSI Value]
...

Server

The server is running at all times with multithreading. When it receives data from a sensor node, it parses the data and abstracts the RSSI values. Then it calculates distance using the algorithm getDistance( ) described here:

```
getDistance(RSSI)
        Live_avg = average(RSSI)

        /* First find the estimated best matched meter by calculating the
        difference between live_avg and the learned average values for
        each meter */
        Closest_meter = get_Closest_meter_Given_Avg(Live_avg)

        /* if the closest_meter is 5, we will also consider 3, 4, 6 and 7)
        */
        meters_to_consider = closest_meter and  ± 2 meters

        for each m in meters_to_consider:
                dist_avg[m] = calculate_distance_avg(db, RSSI, m)

        /* get_distance_for_min_avg() returns distance that corresponds to
        minimum of averages */
        winner_distance = get_distance_for_min_avg(dist_avg)

        /*
        Distance history is a class that has history table and functions
        to calculate auto-regression.
        Distance_history.record( ) records winner distance along with
        sensor mac address to apply Autoregression.
        Since we are applying order 5 for auto-regression, we only keep 5
        past values of winner distance for each (sensor_mac, device_mac)
        pair*/
        Distance_history.record(winner_distance, sensor_mac, device_mac,
timestamp)

        Distance_history.get_final_distance_with_ar(device_mac)

calculate_distance_avg(db, RSSI, m)
        results = get_leanred_RSSI_for_given_meter(m)
        num_groups = results.recordCount / len(RSSI)
        i = 0
        for each group in num_groups
                distances[i]  = findDistance(group, RSSI)
                 i++
        return average(distance)

findDistance(group, RSSI)
        sum = 0
        for i in range(0, len(live_RSSI)):
                sum = sum + pow(learned_tuple[i][0] - live_RSSI[i], 2)
      return sqrt(sum)

class Distance_history
        get_final_distance_with_ar(device_mac)
```

```
s_i = 0
for each sensor in history that corresponds to device_mac
      i = 0
      for each past distance value
            sum[s_i] += coeff[i] * past_distance[i]
            i++
return sum
```

When we display the calculated distance, we first translate the distances for the device into global coordinate, which is in respect to sensor 1. Then we take average of the two values with the uncertainty being the difference between the average and one of the distances. For example, if the device is 3 meters away from sensor 1 and 7 meters away from sensor 2, then sensor 2's value will be translated into 3 meters away from sensor 1. Hence the final distance for the device is 3 meters away from sensor 1. However, this example is a perfect scenario. An imperfect case would be if the distance from sensor 1 measures 2 meter while the distance from sensor 2 measures 6 meter. Then the distance from sensor 2 will be translated into 4 meters from sensor 1. Hence the final distance will be 3 meters away from sensor 1 with an uncertainty of 1 meter range.

To calculate coefficients for AR model, we use a C program implemented by Paul Bourke[5]. We use 1100 RSSI values as data points to calculate coefficients for order 5. Then we use the coefficients in applying auto-regression in our distance calculation.

5 Experiments and Discussion

The equipment used in the experiments is the same as those listed in the preliminary experiments. The device being tracked is the LG KG70 cell phone.

5.1 Learned Data

We measured our learned data by averaging the RSSI value over 100 samples at each meter. The following is our learned data as well as a graphical representation:

Meter	Average RSSI
0	-51.25
1	-69.75
2	-71.43

3	-73.88
4	-72.43
5	-74.76
6	-75.39
7	-75.84
8	-77.73
9	-78.64
10	-81.69

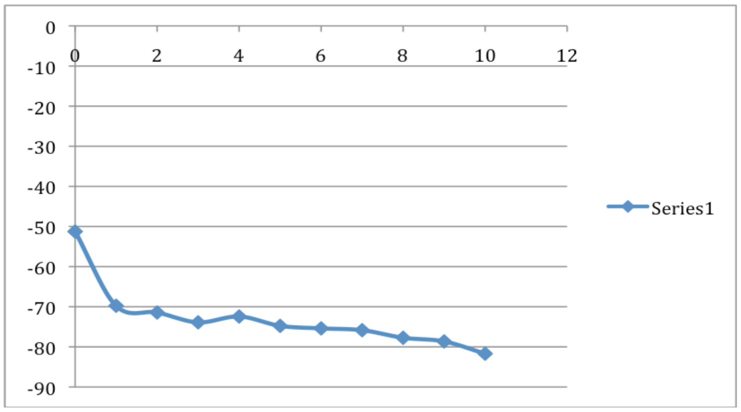

Figure 5: Learned RSSI values at each meter

We found that increasing the number of samples of RSSI values produces a much better linear trend. Our learned results are very close to a perfectly negative linear trend; much better than the graphs produced in the preliminary experiments. We predict that taking more samples would improve the accuracy even more, but did not try it due to the large amount of time required to collect the samples.

In our location calculating algorithm, once we find the closest RSSI match, we consider ±2 meters to compensate for the fact that our learned data is not perfectly linear. For example, if the closest RSSI match was meter 3 at -73.88, then we would want to consider meter 5 as well with -74.76 since meter 4 is has a higher learned RSSI value than meter 3.

A method to improve this heuristic would be to calculate the standard deviation of the learned samples and consider the meters that are within two standard deviations away. This would cover 95% of the possible meters statistically and would ensure all meters that should be considered for distance calculation are indeed.

5.2 AutoRegression

In this experiment, we test the effectiveness of AutoRegression (AR). We recorded a trace of the device starting at 5 meters. We then move to 0 meters, back to 6 meters, stay for a few seconds, then finally move to 10 meters. The graphs show the results without and with AR:

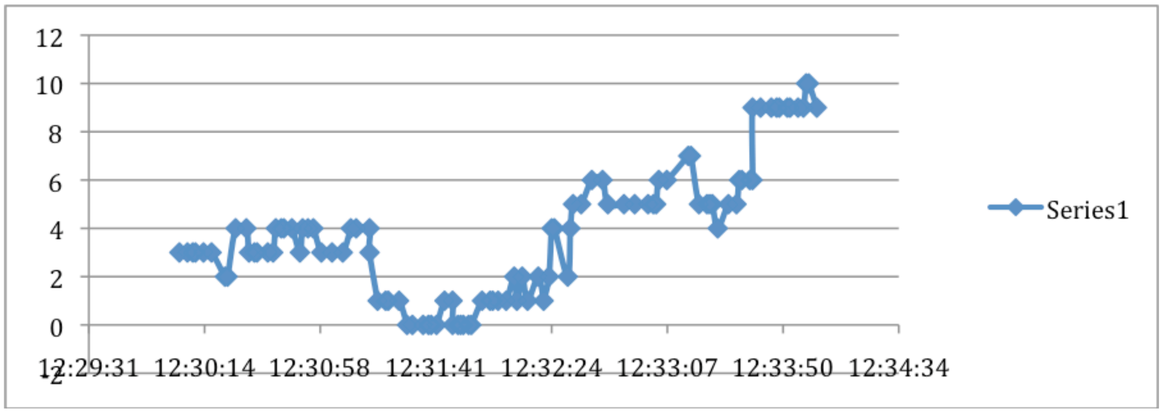

Figure 6: Trace without AR

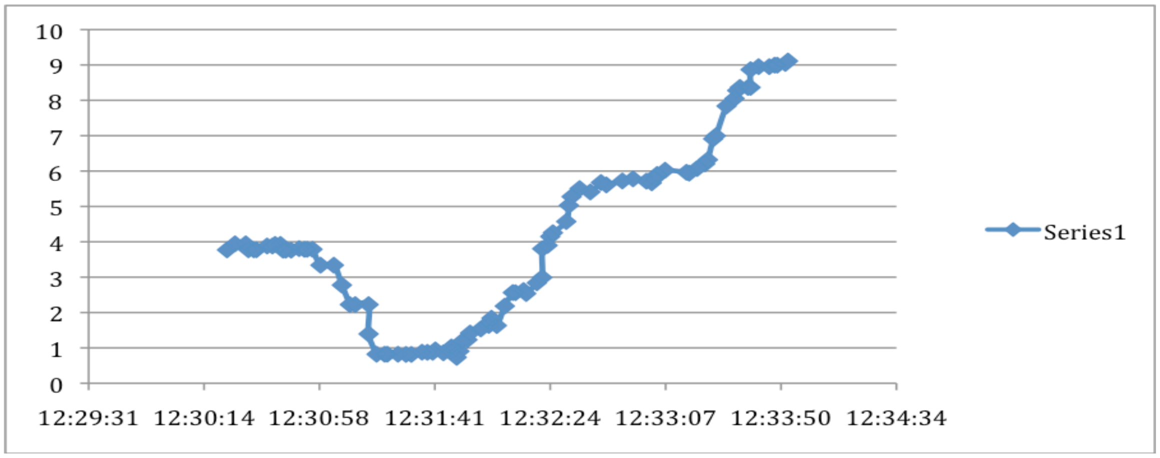

Figure 7: Trace with AR

The results show that AR drastically improves the accuracy of the trace by smoothing the calculated locations measurements. Additionally with AR, the location calculated at any given time was mostly within one meter away from the actual location. This is most apparent at the beginning when the the phone was at meter 5 and the calculated location is flat at 4, when the phone reached meter 0 and the calculated location was flat at 1, when the phone reached meter 6 and the calculated location was at 6, and when the phone reached meter 10 and the calculated location was at 9. Implementing AR clearly improved our tracking results. While we implemented AR on meter calculations, another possibility would be to implement AR on every RSSI sample collected. Instead of filtering the final result, AR on RSSI samples would filter the data being used to calculate the final result. We did not test this and so do not have data on its effectiveness.

5.3 Accuracy of Tracking

Our model has already shown itself to be very effective in locating a device when it is stationary or moving very slowly. The accuracy thus far has usually been within one meter of the actual location. Thus, we experimented with the accuracy of BluEyes when the device is moving at faster speeds. We recorded traces of the device moving from 0 meters to 10 meters. The first trace was recorded with the device moving at about half a meter a second while the second trace was recorded with the device moving at about a meter a second.

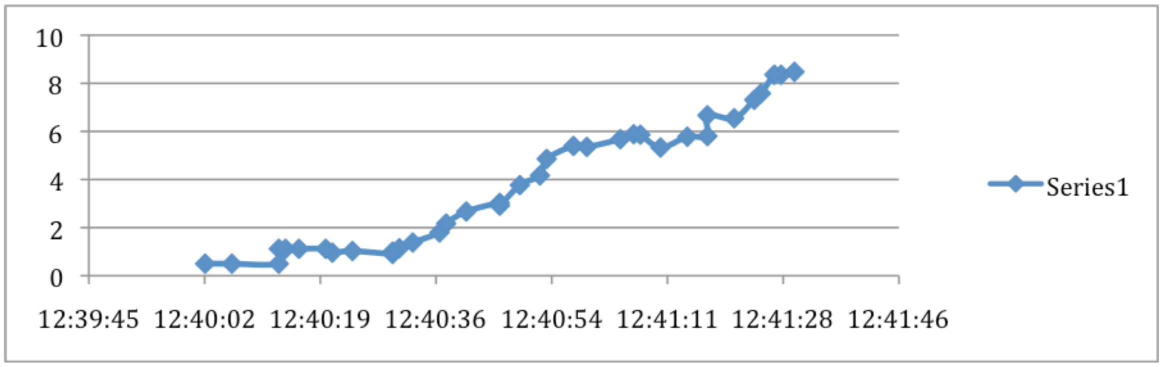

Figure 8: Trace recorded while moving at 0.5m/sec

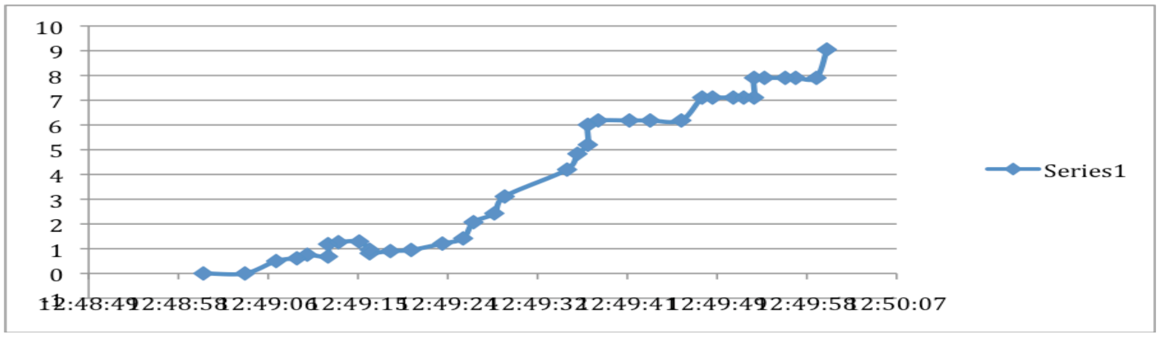

Figure 9: Trace recorded while moving at 1m/sec

The results show a smoother trace with more data points when moving at half a meter a second. The trace recorded when moving at one meter a second is much more jagged and data points are more sparse. This is expected because the faster the device moves, the less time there is to calculate the location of the device.

From our earlier experiments, we noticed that within a second, a master node can receive two to five RSSI readings. However, those readings are divided by any Bluetooth device within the vicinity. For example, if three Bluetooth devices were around a sensor node, the sensor may only have 1 or 2 RSSI readings from a single device in one second. Furthermore, the sensor collects more RSSI readings from devices closest to it since those devices have stronger signal strengths. Since our packets sent to the server include five RSSI readings, the time it takes to form a complete packet takes anywhere between two to six seconds depending on how many Bluetooth devices are around the sensor.

The reason why we chose to collect five RSSI measurements in a packet is because if the number of RSSI values is too small, the average value of the RSSI values may not be accurate enough to represent the actual position. From Figure 2, we are reminded that the RSSI samples can vary greatly at a single location. Therefore, collecting more samples to create a "stronger" average would provide a more accurate estimate of the device location. Initially, we had set the number of samples to be ten per packet. However, the time to gather ten samples was too long and would negatively affect our AR algorithm. The AR algorithm filters the current meter calculated based on the influences of the past calculations. However, if it takes ten seconds to collect ten samples, then our AR algorithm would be filtering the current calculations from device positions measured over ten seconds ago. The performance of the system would be quite poor when assuming the devices are mobile. Therefore, choosing five samples per packet was a tradeoff of update speed to a stronger average RSSI value.

When we moved the device at one meter a second, the update rate from the sensor nodes was not fast enough to coincide with the speed of the device. Therefore, the results of the trace are more jagged than traces recorded at slower speeds. The update rate of the sensors could be improved if all other Bluetooth devices were turned off around the environment, a condition we could not provide because we were in an office environment. Additionally, the wireless network to which the sensor nodes and server were connected was not very stable. There would be packet loss and due to TCP's reliable protocol, the packets will eventually reach the server, although they may be out of order and the propagation delay may be too large. These are conditions our model did not take into account. We would suggest future models to enable wired connections between the sensors and server to eliminate network problems. As for the slow updates due to many Bluetooth devices around the sensors, this is a problem that must be solved if BluEyes is to support tracking of many devices simultaneously.

6 Conclusion

Through our results, we have shown BluEyes to be effective in localizing and tracking Bluetooth devices. When the device is stationary or moving at a slow speed, the calculated location is usually within one meter of the actual location, an impressive feat when considering past research statements stating that Bluetooth serves as poorly for locating devices. Using distance calculations and AutoRegression, we managed to substantially improve the accuracy of the system even with greatly varying RSSI samples. Future improvements include improving the update rate of the sensor nodes while maintaining the accuracy of the system to make the entire system more responsive to faster moving devices.

References

1. P. Bahl and V.N. Padmanabhan, RADAR: An in-building RF-based user location and tracking system, *Proc. IEEE Infocom*, Vol. 2, pp. 775-784, March 2000.
2. N.B. Priyantha, A. Chakraborty and H. Balakrishnan, The Cricket Location-Support System, *ACM MOBICOM*, August 2000.
3. A. Madhavapeddy and A. Tse, A Study of Bluetooth Propagation Using Accurate Indoor Location Mapping, *UbiComp*, 2005.
4. A.K.M.M. Hossain and W. Soh, A Comprehensive Study of Bluetooth Signal Parameters For Localization, *PIMRC*, 2007.

5. AutoRegression Analysis (AR). November 1998. Paul Bourke. 11 Dec 2008 < http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/ar/>.