Precise Indoor Positioning without GPS using a Device Mesh

Gavy Aggarwal
California Institute of Technology
1200 E California Blvd.
Pasadena, California 91126
gavy@caltech.edu

Abirami Kurinchi-Vendhan
California Institute of Technology
1200 E California Blvd.
Pasadena, California 91126
abirami@caltech.edu

Annie Wang
California Institute of Technology
1200 E California Blvd.
Pasadena, California 91126
bwwang@caltech.edu

Kabir Brar
California Institute of Technology
1200 E California Blvd.
Pasadena, California 91126
kbrar@caltech.edu

ABSTRACT

With recent advancements in technology, the need for very precise location determination has arisen. Currently, GPS positioning only performs well outside, and even then, itfis only accurate to about 5 meters. We proposed and tested an application that connects individual phones together, created a device mesh from which an accurate location can be determined. Each phone sends out a bluetooth signals, as well as other sensor data, and receiving phone use that data to gauge its location in reference to other phones in the near vicinity. This has proven successful for distances within 6 meters, with accuracy of up to 10 centimeters. We hope that this can be applied in conjunction with GPS technology to provide navigation in dense, indoor places such as airports or malls.

KEYWORDS

microlocation, BLE, trilateration, sensors, indoor localization, dead reckoning

ACM Reference format:
Gavy Aggarwal, Abirami Kurinchi-Vendhan, Annie Wang, and Kabir Brar. 2016. Precise Indoor Positioning without GPS using a Device Mesh. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 7 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Smartphone GPS technology has come a long way since its birth, yet struggles to meet current demands of accurate location detection. While its accuracy of up to 4.9 meters is enough to navigate cars on roads, itfis not enough to pinpoint user location in buildings. Additionally, the signal is easily obstructed indoors, meaning locations can only be obtained in outdoor, open spaces in many situations. Our app alleviates this problem by using bluetooth technology to create a network of devices that communicate with each other to establish a more accurate location estimate.

2 THE PROBLEM

The inability to get such precise positioning isnfit due to the lack of trying. The advancement of technology introduces many novel ways in which an accurate position can be obtained, but finding the best one can be a challenge. One can place bluetooth beacons indoors to improve location tracking, but that requires a large investment in setup at any location that wants increased location accuracy, as well as introduces a high cost of maintenance. Another alternative is the use of visual markers, which uses a camera to detect objects and determine spatial coordinates from the angle to the object. This reduces the need for additional hardware, but still requires prior mapping of the area, which can be expensive and requires frequent updating as the area changes. Instead of using new hardware, we can use pre-existing smartphone sensors and communication methods gauge distance from one phone to the other, increasing accuracy in smaller areas. This can also be used indoors since it does not rely on satellite signals, merely signals from nearby devices.

3 SOLUTION

3.1 Bluetooth Trilateration

The initial solution was to use Bluetooth low energy (BLE) to gauge distance from other phones in the network and subsequently perform trilateration to calculate the most probable location. Further, utilizing Bluetooth enabled its dual usage as the primary mode of communication between nearby devices. Through BLE advertising and scanning, the devices transmit relevant information about their current location while reading the locations of nearby devices. The data is advertised as a 13-byte message, and includes the device's ID (as a byte) and its xyz coordinates (4 bytes each). A unique app ID is attached to the messages to help distinguish relevant advertisement packets. This data is then parsed and used by the optimization algorithm on other devices to calculate their current locations when mobile. To measure the distance from other devices in the mesh, Bluetooth Received Signal Strength Indicator (RSSI) values of broadcasted messages were measured. The values, in dBm, were found to be well correlated with distance within a range of 5

m. Past this threshold, the RSSI value stayed constant with distance, making it difficult to determine an accurate distance from the value. This uncertainty was accounted for later in the trilateration algorithm. The collected data points and the derived curve of best fit are shown in Figure 1.

Unfortunately, the Bluetooth RSSI values were not always consistent and were susceptible to noise. Interference from other broadcasting phones caused the values to fluctuate. Further, the fiopennessfi of the surroundings influenced readings, presumably due to the variable number of surfaces obstructing and/or redirecting the signal. Additionally, we could not precisely control the transmittance power of the Bluetooth signals, which also led to some variance in the RSSI values. However, this correlation was strong enough to yield accurate base calculations of relative distances even with these limitations. The following equation was used to estimate the distance in meters, d, from the rssi value in decibels, r, of a nearby device:

$$d = 0.000163643e^{0.167732r} + 30.1904$$

This equation had a R^2 value of 0.533 for the entire dataset and a R^2 value of 0.674 when the device was within a 5 meter range.

3.2 Barometer, Compass, and Accelerometer

While Bluetooth is perhaps the most accurate alternative to conventional GPS, it alone cannot make up for the absolute positioning afforded by the latter. In particular, while one can use trilateration with just Bluetooth RSSIs to determine onefis relative position, one cannot easily do so in a highly dynamic scenario that involves, for example: long distances beyond the strength of Bluetooth, highly signal-turbulent locations (i.e. where Bluetooth signals are bouncing around a lot), keeping track of onefis absolute cardinal direction, and in some cases keeping track of onefis change in position relative to certain objects (e.g. to a particular point in a completely unmapped area when all phones involved in trilateration are also moving). Consequently, we rely on several other sensors available on modern Android devices to try to account for these difficulties, and essentially build up a set of varied metrics corresponding to a phonefis movement that ultimately, if in agreement, approximate high accuracy tracking (and alternatively let us know if something is wrong if there is disagreement) that matches and even exceeds modern GPS. In the context of the larger literature about information/position tracking and GPS-alternatives, this combinatorial approach is often referred to as fiSensor Fusion,fi and we took inspiration from some such fusion initiatives that have been pursued in the past on Android devices. In particular, we focused on four other sensor modalities available on the LG G2 (amongst many Android devices): barometer, accelerometer, compass, and microphone.

Since API Level 3 (Android 1.5), the Android platform has provided four types of environment sensors (as opposed to various phone-effect sensors) of which we use the barometer to track changes in altitude. For every phone, we register a listener to detect changes in ambient air pressure at the default delay rate (approximately 200 milliseconds), and capture local air pressure measurements provided by the Android API in hectopascals in a queue. In particular, when the phone is not moving, we reset a locally stored reference pressure and reference height – the former being the average of the pressure measurements collected up to that point (consequently also clearing that queue) and the latter being the last height set by the overall trilateration algorithm. Note that in our typical starting scenario for trilateration, we have set the locations of the phone in advance – all at reference locations that are designed to be zero height/altitude relative to the phonefis future movement. Otherwise, when the phone is moving, we receive a new pressure measurement, update our running average reference pressure (adding to its corresponding queue) as well as a current pressure variable. With this information, we have a rough estimate of a devicefis relative height by computing:

$$H_c = \frac{(P_r - P_c)}{2 * HPM} + H_r$$

where:

$$H_c$$

is the current height estimate,

$$H_r$$

is the reference height set during device initialization,

$$P_r$$

is the reference pressure,

$$P_c$$

is the current pressure, and

$$HPM$$

is the ratio of hectopascals per meter (equivalent to 0.11179333 hPA/m). Here, we are finding the last change in pressure, converting it to meters, dividing by 2 to help account for barometer noise, and then adding to the original reference height to find the current height (always relative to initialization position). When trilateration is performed, this estimated height acts as an extra gradient term (see

$$C_{pressure} * \sqrt{(y - y_{pressure})^2}$$

) that is also modulated by a set constant reflecting the extent to which we wish the barometer to influence trilateration (with a larger magnitude reflecting higher importance).

The primary adjustment to Bluetooth RSSI values comes from local accelerometer tracking. Accelerometer tracking in phones, amongst other small devices containing similar pieces of hardware, is notoriously noisy on it own – though we pose that combining accelerometer data with Bluetooth data gives us a pretty good estimate of a phonefis position to a degree of certainty beyond GPS. In particular, we register listeners for Androidfis gravity sensor (which isolates local measurements of the gravitational force) and Androidfis raw accelerometer (which measures raw phone acceleration values), again at the default delay rate. We keep a constantly-updated queue of the past two accelerometer measurements along with their respective timestamps and, for every update to the queue, calculate the new accelerometer-based position by double integration. Specifically, we isolate linear acceleration values by subtracting the current gravitational values (updated by the gravity sensor as changes are detected) and then convert these values from the phonefis default frame of reference to the absolute frame of reference used in trilateration. This is possible due to

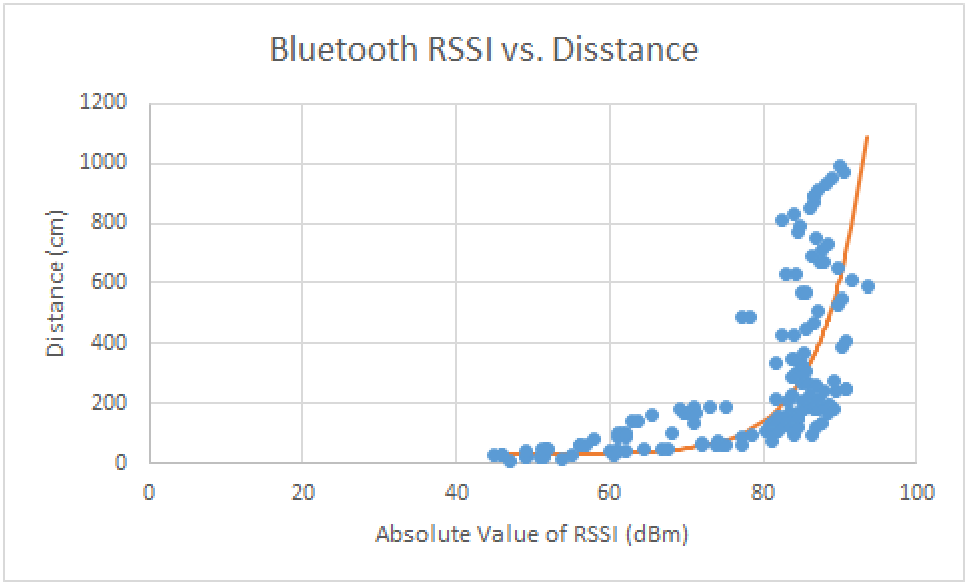


Figure 1: Correlation between distance and BLE signal strength (RSSI).

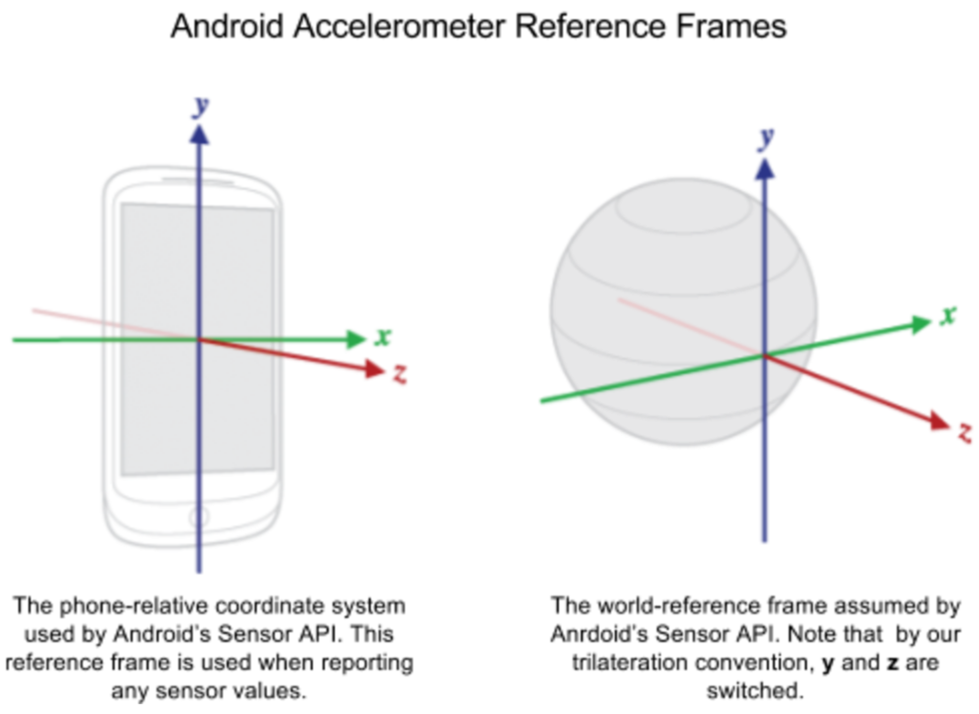

Android Accelerometer Reference Frames
y
x
z
y
x
z
The phone-relative coordinate system used by Android's Sensor API. This reference frame is used when reporting any sensor values.
The world-reference frame assumed by Anrdoid's Sensor API. Note that by our trilateration convention, **y** and **z** are switched.

Figure 2: Axes for reference in accelerometer systems of Android devices.

Androidfis Sensor API also providing a method to extract a rotation matrix that can convert vectors from the phonefis frame of reference to the worldfis frame of reference. Briefly, if

R

is this rotation matrix, then in the case where the phone is aligned with the world coordinate system (i.e.

X

points East,

$$Y$$

points North,

$$Z$$

points to space)

$$R$$

is the identity matrix. Otherwise, if

$$v$$

is some vector in the phonefis reference frame, one can convert it to the global reference frame by computing

$$v\text{fi} = R^{-1}v$$

. After conversion, we use the time difference of the past two accelerometer measurements to double integrate to find first current accelerometer-based velocity and then accelerometer-based position – with both of these measurements having been initialized to zero on the assumption that trilateration begins with all phones not moving. Consequently, we are able to end up with a pretty decent measure of where a phone is generally moving that is computationally distinct from that of Bluetooth RSSI values. However, we also note that two of the larger problems that we experienced (having to do with the noisiness of current phone accelerometers) were drift and lag. Because a phone is constantly under the force of gravity even when it is not moving, a phonefis position is naturally subject to drift over time if we use pure accelerometer values. Even subtracting the forces of gravity using Androidfis gravity sensor values, or a high-pass filter in earlier instances of our app, does not completely account for this drift. Additionally, while new measurements are received fairly regularly, there is still a noticeable lag in terms of tracking changes in the phonefis rotation and acceleration – over time, this lag also adds up to a penalty that exacerbates drift and generally makes it difficult to update a phonefis current accelerometer-based position if the phone is in a turbulent environment (e.g. being jostled around while walking).

The final correction to Bluetooth RSSI comes from the Androidfis magnetometer – one of the more accurate sensors in the Sensor API suite. While this correction is not used in the final build of the app, it was initially experimented with as a fifinal checkfi to make sure that the direction of movement implied by other phone sensors is correct. In particular, we register a listener at the normal delay rate and then, on every update, get the rotation matrix of the phone as above and calculate the cardinal direction of the phone. Androidfis Sensor API provides methods to extract the phonefis orientation (containing pitch, roll, and azimuth in terms of their Sensor API definitions) from a rotation matrix, where azimuth gives us a measurement of where the phone is pointing in radians. We convert the azimuth to degrees and then set standards to delineate general cardinal directions (i.e. on the unit circle, 45 degrees to 135 degrees counts as North, 135 to 225 counts as West, 225 to 315 as South, and so on). We found in general that Android does a pretty consistent job of getting cardinal directions correct, though this measurement is also prone to some error when the phone is quickly moved around due to lag as well as phone rotation (which is ameliorated through other methods Androidfis Sensor API provides for remapping the coordinate system of a rotation matrix in such scenarios).

3.3 Auditory Localization

For any trilateration approach, the distances and locations of several nearby devices is required. While it was possible to share the coordinates of proximal devices using bluetooth advertising, determining the distances to these devices based on the signal intensity was susceptible to several pitfalls. For one, the amount of interference present in an environment caused large fluctuations in the signal strength and thus, the distance estimates. Also, the broadcasting power varied between devices and based on factors such as battery charge level, skewing the estimated distance between devices. As an alternative, an auditory localization technique was explored in which a device emits an ultrasonic ping that will be echoed back by another device. This approach would calculate the distance separation between the devices using the time it takes to hear the echo and would rely on the speed of sound which will not be affected by interference or battery level. Thus, the distance, d, between two devices by

$$d = \frac{(t - l_1 - l_2)}{2} v(T, \phi)$$

where t is the time taken for a device to hear back a ping, l_1 is the duration of the latency from sound processing for the emitting device, l_2 is the duration of the latency from the echoing device, and $v(T, \phi)$ is the velocity of sound in the air at a temperature T and humidity ϕ, which can be obtained by the built in temperature or humidity sensor if available or from a weather forecast for the current location using general GPS coordinates. This technique was implemented using the Java AudioTrack and AudioRecord APIs provided by Android as well as in C++ using the Native OpenSL ES API. The native C++ API performed much better than the Java Android API and produced a precision small enough to be useful in a trilateration application.

To allow multiple devices to communicate, each device produced a unique ultrasonic tone for its audible pings and a sliding discrete fourier transform (SDFT) was used to identify each device. However, in practice, these technique was not used in the final app because the number of points in the SDFT had to be reduced significantly in order to process microphone input in realtime. At a sample rate of 44.1 kHz, an additional buffer needed to be processed every 5 ms whereas it took approximately 20 ms to perform a SDFT on the buffer. This problem was further exacerbated by the microphone and speaker hardware on the LG G2 development devices which have poorer performance in the ultrasonic range as opposed to the audible range. Thus, while auditory localization has been proven to be successful and have a high level of precision, it is currently unsuitable for a real-time trilateration application but has future potential as computation power and speaker and microphone hardware continues to improve.

3.4 Pedestrian Dead Reckoning

To allow devices to determine their location without other nearby devices and with low latency, a pedestrian dead reckoning approach was used to estimate location. Pedestrian dead reckoning assumes that the phone is carried by a person traveling by foot to estimate its location. Since human gait has a very distinct accelerometer signature, it is possible to detect each step that a person has taken. For this application, the step detector software sensor in Android

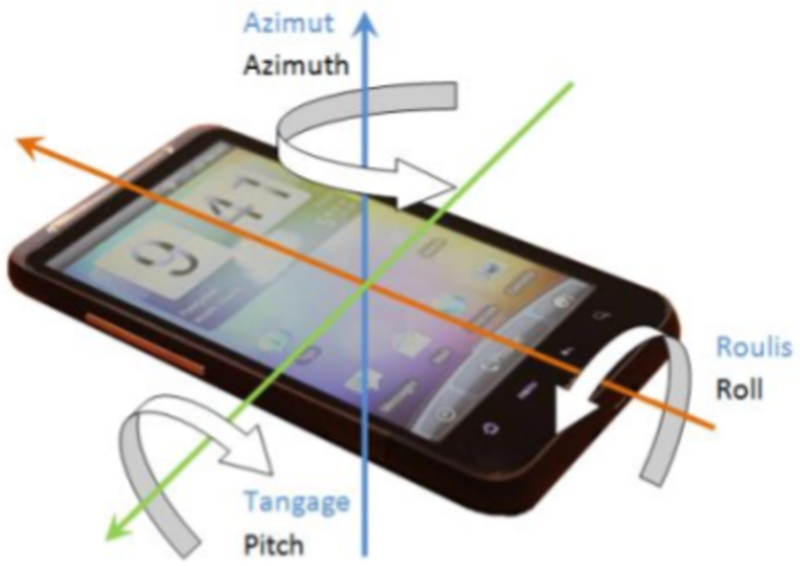
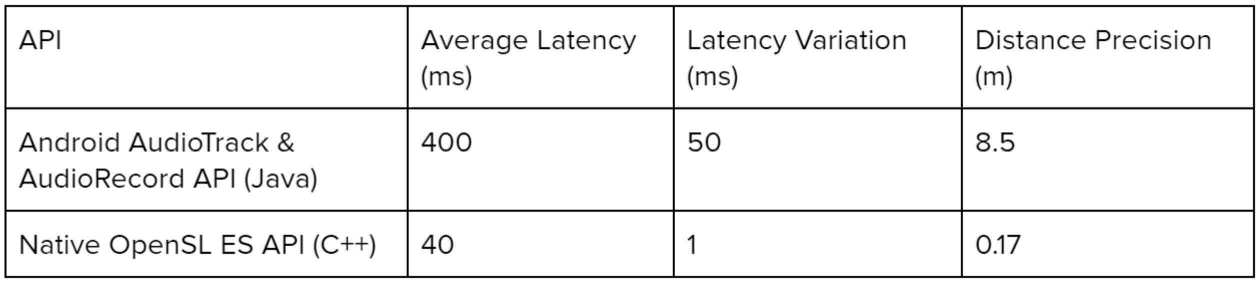


Figure 3: The angle definitions used by Android accelerometers.



API	Average Latency (ms)	Latency Variation (ms)	Distance Precision (m)
Android AudioTrack & AudioRecord API (Java)	400	50	8.5
Native OpenSL ES API (C++)	40	1	0.17

Figure 4: Localization performance for various audio APIs.

was used to determine when each step was taken. In addition, there is a magnetometer sensor that is used to calculate the azimuth of the device relative to the north pole. The coordinate system is defined so that the x-axis is increasing in the northward direction and the z-axis is increasing in the eastward direction. Whenever a step is taken, the x coordinate is increased by

$$scos(m + a)$$

and the z coordinate by

$$ssin(m + a)$$

where s is a scaling constant that represents the stride length in meters, m is the magnetometer reading, and a is the angle that the azimuth is rotated by to factor in the orientation that the user is holding the device. For example, if the device is orientated in the userfis pocket so that it is facing 90 degrees away from the direction the user is facing, a would be 90 degrees so that the location increases in the eastward direction if the subject is walking southward. In practice, s and a were determined by a calibration routine in which the subject must walk north for 10 meters and s and a are calculated such that the initial position is $x = 0, z = 0$, and the final position is $x = 10, z = 0$.

3.5 Optimization Algorithm

To combine the somewhat noisy inputs from various sources and form a precise location estimate, an optimization procedure was used. The optimization procedure predicted the best location at a certain time based on the locations and distances from nearby bluetooth devices, height estimates using the barometer, and lateral displacements calculated from dead reckoning. The objective function used was

$$
\begin{aligned}
S = \quad & \sum_{i=1}^{n} c_i(\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}-d_i)^2+ \\
& c_N\sqrt{(x-x_{old})^2+(y-y_{old})^2+(z-z_{old})^2+1}+ \\
& c_B(y-y_{pres})^2+ \\
& c_D\sqrt{(x+x_{pdr}-x_{old})^2+(z+z_{pdr}-z_{old})^2}
\end{aligned} \tag{1}
$$

where $c_i = (d_i+1)^{-1}$ is the weight of the reading from the i^{th} device used in trilateration, (x_i, y_i, z_i) is the position of the i^{th} device, d_i is the distance from the i^{th} device, $c_N = 1$ is the weight of normalization term, $c_B = 5$ is the weight of the barometer term, y_{pres} is the height estimate based on the barometer reading, $c_D = 2$ is the weight of the dead reckoning term, and (x_{pdr}, z_{pdr}) is the relative change in location produced by dead reckoning. The current location of the device, (x, y, z) is found by minimizing the objective function using stochastic gradient descent over 1000 iterations with a step size of 0.01.

4 RESULTS

In summary, our app is able to track phone movement on a scale beyond the scope of current GPS technologies. We use a sensor fusion approach that centers around Bluetooth RSSI values while utilizing Android accelerometer, barometer, audio, and magnetometer data as well as dead reckoning techniques to determine a phonefis relative position on-the-fly. Quantitatively speaking, our app is able to detect fine changes in position within five meters to great accuracy – something out of the scope of GPS – with GPS only beginning to match our progress around 8 meters (which is well beyond the radius for the applications that our app is primarily targeted at). This makes sense, given that even in open areas, it takes a significant change in position (on the order of around ten meters) for the position in most GPS applications to meaningfully update.

The above results withstanding, microlocation still faces limitations in various scenarios. While our microlocation system is uniquely well suited for indoor environments out of the reach of GPS, our app generally begins to perform poorly on the order of position change at which GPS systems update. This is primarily due to the significant dropoff in Bluetooth signal quality as well as accelerometer data quality on such longer distances. Additionally, the final build of our application still relies on a known starting position for a phone relative to at least three other devices. Finally, Bluetooth, like GPS, also faces certain limitations from interference and signal reflections, though the exact extent to which this is a problem is still unclear.

5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

It has been established in this paper that it is possible to achieve accurate indoor positioning with present technology and existing hardware in smart phones using a combination of sensors and algorithms. This technology can enable advanced indoor mapping solutions that could be used in airports, subway stations, malls, and sports stadiums. For example, an airport maps app that could provide directions as specific as 'take the escalator in 500ft to reach your terminal'. It can also be used in homes, where electronics can turn themselves on or off as users enter and exit a room.

5.2 Future Work

Currently, running the app drains battery at a significant rate, decreasing the practicality of using accurate location technologies for longer periods of time. In order for this to be seamlessly integrated into daily lives, there needs to be an optimization that prevents unnecessary battery usage.

Further, the possibility of using a Kalman filter or particle filter in place or in addition to the optimization algorithm from Section 3.5. Unfortunately, this task unable to be completed successfully by the end of the term. However, it deserves further investigation and experimentation.

Next, this technology can be built in directly into the iOS and Android operating systems so that the network will encompass all mobile phone uses, and thus provide more reliable precise location tracking. Applications built on this technology will be able to use reliable indoor mapping to better target consumers or provide smart-home and smart-city solutions to any person with a phone.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Adam Wierman and Xiaoqi Ren for providing useful suggestions and guidance during the development of indoor locationing solution.

REFERENCES

http://www.gps.gov/systems/gps/performance/accuracy/

Roberto Michel, (2016) Information Management: Wearables come in for a refit, Modern Materials Handling, Retrieved Dec 28, 2016

https://www.lifewire.com/sensors-that-make-iphone-so-cool-2000370

https://developer.android.com/guide/topics/sensors/sensors_overview.html

http://www.mdpi.com/1424-8220/15/9/23168/pdf

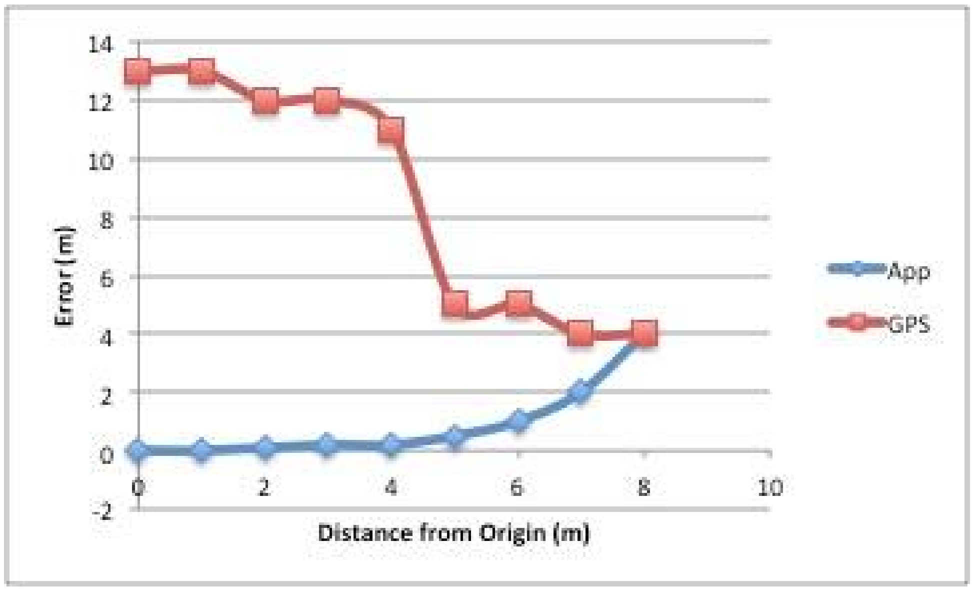


Figure 5: The distance error between GPS and our microlocation app for walking away from a fixed origin.