Indoor Way-finding Using Bluetooth Low Energy Beacons

Group 7
Arman Pouraghily, Fubao Wu

Abstract—With the new advancement in technologies and with pervasive use of smart phones, we are observing a new paradigm of way finding and navigation. Most of the new smart phones has the ability to receive the GPS signals and use them for localization. GPS based localization and navigation is simply replacing the old fashion map based navigation which was used for long time. The only problem with this new approach is that it can not be used inside the buildings. In this work, we implemented a localization and wayfinding application in in Java for android based smart phones and tablets which is used for navigation inside the buildings. The localization is based on the data collected from some bluetooth low energy tags which are well placed in the first floor of Marcus building. The system catches the signals coming from all the beacons in its range and estimates the distance to each of them. Using the distance from the three nearest tags we could find our location related to the location of those tags and by knowing the absolute location of those tags we could find our location. After finding our location, we could use a simple navigation algorithm for navigate through the building to any desirable location.

Index Terms—Bluetooth Low Energy, Indoor Way-finding, Longitude, Latitude, Circle Intersection

1 Introduction

As said before, most of the new smart phones have the ability to receive the GPS signals and use them for localization. GSM based localization and navigation is useful when you are outdoor and you can see the sky over your head but not inside the buildings. Navigation inside the buildings maybe seems unnecessary at the first sight because inside the buildings, you could simply find your way by using appropriate signs. But you can do so, if you have an acceptable vision but what about the people with visual impairment. In this work we are focusing on those people. We have developed an application which make use of bluetooth low energy tags to find our location and try to navigate through the building to any desirable location using voice commands.

In order to implement any kind of localization algorithm, we need to know the absolute location of some points as the basis of localization process, in this project we have known the exact location of the deployed beacons in terms of latitude and longitude. So in order to find our exact location, first we need to find our location relative to the location of those tags and then using a simple procedure, map our relate relative location to the absolute location.

2 Phase I

In this section, we describe the application and algorithm used for indoor localization which has been done in phase one.

2.1 Implementation details

At the first step, we should filter the student tags out by the major ID. We have stored the details of the 17 tags used in Marcus building. When we received a signal, we just check for its major ID to see if it matches with one of those 17 values. If yes, we go to the processing step and if not, we simply ignore it. Since the signal values are not stable enough, we use averaging to make them more stable. We have implemented 17 queues for these 17 tags (one queue per each tag) with depth of 4 (the number of samples used for averaging). Once we receive a new signal value from a tag, we put that value in the appropriate queue and if we receive no signal from a tag, it means that we are not in the range of that tag, so we put -100 in that tag's queue. At each round, we calculate the average of the samples stored in the queues and sort them to find the closest tags. After sorting the tags based on their average signal values, we should pick the three strongest ones. In the next step, we need to calculate the distance to those tags. The distance to each tag can be calculated using the following formula:

$$Distance\ (feet) = -0.00903 * RSSI^2 - 2.171 * RSSI - 94$$

After finding three closest ones, we try to find our location relative to their location. For this purpose, we pick the two stronger ones and draw two circles with centers on those two tags and radius equal to the corresponding distances to them. By intersecting those circles, we could have one or two points (one point is very unlikely but possible) or even no intersection. If there is no intersection, we draw an imaginary line between those two tags and we will find a point between those two points with the distance proportional two the distance from those two points. In case of 1 intersection point, that point would definitely be our answer and in case of two intersecting points, each of those two points could

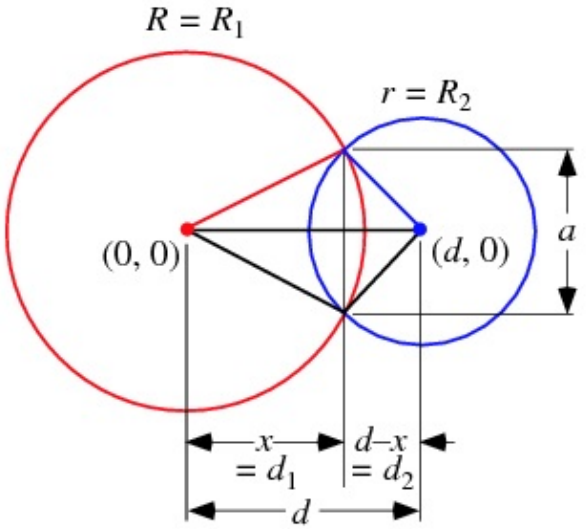

Fig. 1: Two circles with centers on x axis

$$x = \frac{d^2 - r^2 + R^2}{2d}.$$

$$y^2 = R^2 - x^2 = R^2 - \left(\frac{d^2 - r^2 + R^2}{2d}\right)^2 = \frac{4d^2R^2 - \left(d^2 - r^2 + R^2\right)^2}{4d^2}.$$

Fig. 2: Formulas to find the intersection of two circles

be our probable location. In this case, we use the distance to the third nearest tag. We draw another circle with center on the third tag's position and the radius equal to the distance to that tag. If this new circle intersects with those two circles, we would have a common region between all those circles and our location would be estimated by the center of that common region. But if the third circle doesn't have intersection with both of those circles, we would select one of those two points which is closer to the third tag.

In order to draw those circles and intersect them, first we need to know the location of those tags. In order to do so, we have implemented a look up table with the location of each tags but in geographical coordinate system. Each location could be obtained by the minor code of each tag. First of all, we look up the location of each tag by looking up its minor code. The next step is to build a new coordinate plate for the selected tags and also its needed to find their distance in meters (since the distance obtained from BLE signals is in feet and converted to meter). Using a simple transformation, we could find the x and y displacement of the points relatively to each other. We also use the left one as the origin of this new plate. After that, we have the x and y of each point and our distance to each of them, we could draw those mentioned circles and intersect them.

If we have two circles with center of them on the (0,0) and the center of the other one also on (d,0), we could find their intersection as follows:

As it can be seen in Fig 1, we drew two circles with centers of (0,0) and (d,0) and the radius of R1 and R2 respectively. The goal is to find x and y of those two intersection points. According to [1], we could find the location of those points using these the formula shown in Fig 2. But we should note that this formula will work only when both the center points of those circles are on the x axis.

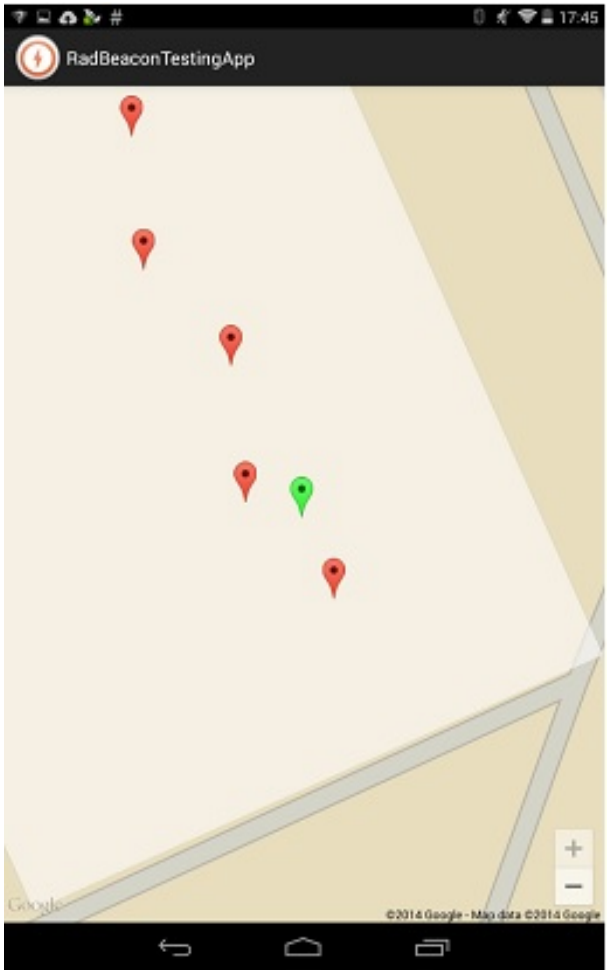

Fig. 3: Localization result between tag #1 and tag #2

In order to use that formula, we should use a transfer function and also a rotation function. After applying one transfer and one rotation, we could have the x-y of those points. After calculating the x-y of the intersection points, we should bring them back to the plate they actually were which means we should use a rotation first and a transfer after that.

2.2 Application Details

In the tagSeachingActitiy, it will enter the search of tags screen, at least three tags should be detected for navigation.

In ShowGoogleMapActivity file,
1: Search all the tags that can search in a subthread.
2.Transfer the signal to distance use the signal to distance model formula. 3. Store the hashmap of the key and distance value of corresponding searched tags, the distance is stored used a queue.
4. when there is 12 sample in the queue, we calculate the average distance of sample to 12 to the average distance. When there are more than 12 samples, we will remove the head the queue ,enter the new sample into the queue.
5. then we setup a hashmap for the key and average distance.
6. if the tags numbers are equal or bigger than 3, then we use the searched tags to sort the hashmap above.
7.then we use localization method in the UI thread!5: After localization, we continue the step 1 again.
The localization method shown in the report!

Localization uses three circles/two circles intersections.

First, if the first nearest and second nearest tags has no intersection, We will use the center of the edge of two circles along the line on the center of circles!

Second, If the first and two intersection has intersection, and the third circle has no intersection, we will use two intersection points p1 and p2 of the two nearest circles. The third circle center point is C3, We will calculate the distance P1-C3, between p1 and C3, and the distance P2-C3 between P2 and C3. If P1-C3 is smaller than P2-C3, we will use p1, Otherwise, we will use p2.

Third, if first three circles have intersection, we will use three circles intersection, Get the three points of their common intersection. I will make up a triangle, Then use the center point of the triangle to use the location!

2.3 Testing Scenarios

We tested whether we obtained the three nearest points correctly and observed location point in different scenarios:

1) When we hold the tablet in one place still, the location is almost accurate and the error distance is about 1-2 meters. As can be seen in Fig. 1, we are standing between tags 1 and 2 and the location is estimated pretty accurately.

2) When we move directly from the tag 1 to tag 15 location or the opposite direction, it approximately show the location of phones, but the error is becoming bigger. It also depends on the speed of phones movement. the much slower we move, more accurate the location estimated. But there are some jitters and instability, it also depends on the receiving signal strengthen. Sometimes, we find the remote tags signal is strong, and thus could affect the accuracy and cause the error value of localization. We write down the geographic location of every times calculation into the SD Card text file and find the altitude and longitude moving with the right direction.

3 PHASE II

In the second phase, we added the navigation capability to the localization application we developed in the first phase. In this section we will describe the details of this application and its implementation.

3.1 Implementation Details

In order to add the navigation capability to our application, we divided the area into three sub areas: Main corridor, Doorway #1, and Doorway #2. In each of these sub areas, there are two different possible moving directions. In the doorways, the directions are *to west* and *to east* and in the main corridor, we have *to north* and *to south*. When we start the navigation, we first check the current position and also the destination position with the sub areas boundaries to see in which sub area they are located. If they are both located in the main corridor, we just compare the latitudes. If the destination latitude is bigger than the source one, the direction is *to north* otherwise it should be *to south*. The same comparison holds true but with respect to the longitude in the current location and the destination are both in one of those doorways. If the current location and the destination are not in the same sub area, we will do a multi-step navigation. We defined two junction points at the junction of each doorway and the main corridor. When we want to change the sub area, we should first go to one these junction point (depends on which of the sub areas we are) and then change the sub area. For example if we are in the main corridor and the destination is in the lower doorway, we should first go to the junction of the lower doorway and the main corridor and then go in the doorway until we get to the destination.

Another aspect of our navigation system is to find the right direction. We calculated the angle between the north direction and the corridor and also the angle between north direction and the doorways off-line and stored the values in a table. During navigation, using the internal gyroscope of the tablet/cellphone, we roughly calculate the orientation of the device and using some voice commands, try to compensate the orientation and bring the user in the orientation of the route.

3.2 Application Details

In order to do the navigation, the user should tap and hold on the desirable destination and after that, press the *Navigate* button on the top left corner of the screen. After pressing that button, the navigation procedure start

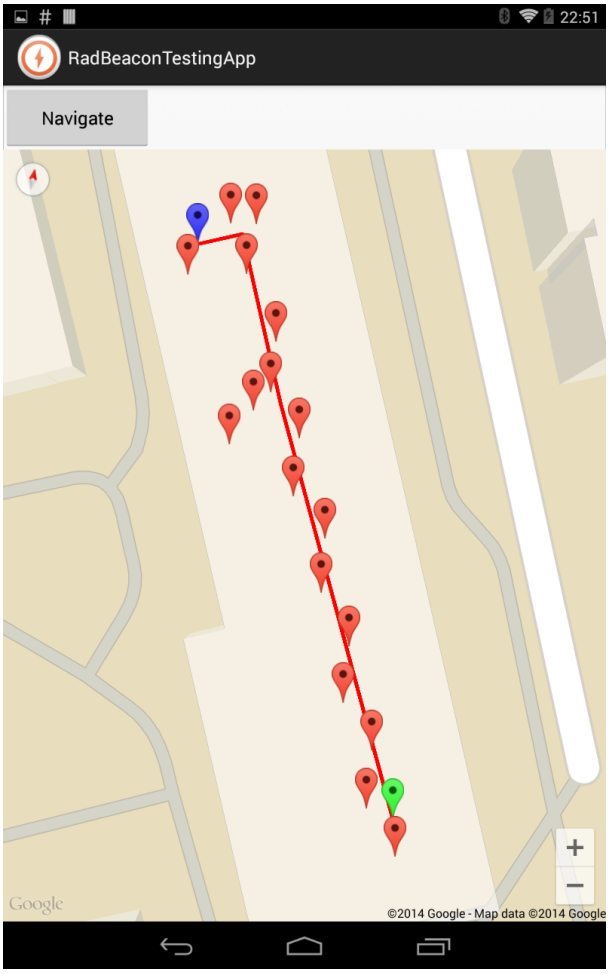

Fig. 4: Route from the main corridor to the upper doorway

Fig. 5: Route from the lower doorway to the main corridor

which details are as follows: At the first step, we form a simple graph in *ShortestPath.java*. This simple graph, initially consists of two nodes which are located at the junctions of the doorways and the main corridor. After that, we should add the destination and our location to this basic graph before starting the navigation procedure. After adding these two nodes, we ready to find the best route. The best route is calculated by use of Dijkstra algorithm [2]. After finding the best route, it's time for navigation. For navigation purpose, we firstly, need to find the correct direction of the device. In order to find the direction of the device. To find the direction, we used the internal gyroscope of the device. Using the function *onSensorChanged(SensorEvent evnt)*, we find out the angle between the device direction and north. If there is any deviation from the angle calculated and the right angle to the next hop, we use a voice command to correct the direction and if there is no considerable deviation (i.e. the deviation is less than 45 degrees) we use the voice command of *go straight for x meter* where x is the distance to the next hop. Upon arriving at the next hop or updating the location for 5 times, we again calculate the angle deviation and say it. The reason behind that we don't say the correction angle is that, we don't want the device to speak continuously and we assume that the user will walk in a straight fashion.

3.3 Testing Scenarios

In order to test the navigation system, we came up with different scenarios. As said before, we divide the whole area into three sub areas. In these scenarios, we made sure that the routing is working between any pairs of source and destination within different sub areas. Some routing examples can be found in figure 4 through figure 6. In Fig. 4 we could see the route from the main corridor to the upper doorway. In Fig. 5, the current position is in the lower doorway while the destination is in the main corridor and finally, in Fig. 6 we could see the route from lower doorway to the upper doorway. As said befor, in this case we should first come to the junction of lower doorway and the main corridor, after that go to the junction of the main corridor and the upper doorway and finally find the destination within the upper doorway.

Fig. 6: Route from the lower doorway to the upper doorway

4 POSSIBLE IMPROVEMENTS AND EXTENSIONS

In this section we briefly talk about some weaknesses in our application which could be addressed in future to improve the functionality of the application or improve its accuracy.

1) The data read from the gyroscope doesn't seem to be that much accurate. Maybe we could find some calibration or compensation method to make the data more reliable.

2) The interface used in the application is not suitable for the people with visual impairment but since, we used an modular architecture in the design of our application, a new interface can be simply plugged to the application and be used.

3) In most cases, we have good signal strength from at least three different tags which is sufficient for our localization algorithm but in very rare case in which we have only two tags in rage, the localization seems to be a little inaccurate which could be improved by employing a new localization algorithm in that special cases.

4) In this very first version of our application, we didn't limit the destination to be within the boundaries of our area which may cause our application to misbehave in some cases where the destination is set outside the area. It doesn't seem to need very much works and could be done easily.

5 APPENDIX I

We have discussed the phase1 together and how to design generally.

Fubao Wu's main contribution:

Fubao Wu Designed and implemented the phase 1 code and did debugging

Designed and implemented all the codes of phase2 and did debugging and some tests!

Added the detail of application to the final report.

Fubao Wu wrote the seperate program design file and the setup program file.

Arman Pouraghily's main contribution:

Arman Pouraghily did some test of phase1 and write the report of phase1

Arman Pouraghily did some tests of phase2 and write the report of phase2 generally

REFERENCES

[1] MathWorld. Circleintersection. [Online]. Available: http://mathworld.wolfram.com/Circle-CircleIntersection.html

[2] Wikipedia.org. Dijkstra. [Online]. Available: http://en.wikipedia.org/wiki/Dijkstra's_algorithm