*Program Design and Implementation*

# Phase 1: Localization Technique

1. ***Application Details***

We have developed a solution to tell the application users their approximate location within the basement of Marcus Building. Firstly, by searching tags in the basement, we get several tags with different values of RSSI. All tags we find will be sorted by the strength of RSSI of each, in the algorithm of Bubble Sort. We select the three tags with strongest RSSI (tag A, tag B, tag C) and put them into formulas to determine the user’s location (point p). And also, when there are only two beacons searched by the application, it can also use two beacons to get the approximate location.

## *(2) Implementation Details:*

We have determined a solution to tell the application users their approximate location within the basement of Marcus Building, which is listed in steps below.

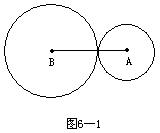
1. By referring to the attenuation model for smartphones, we use the formula Distance = 0.008459 \*RSSI2 + 0. 6711\*RSSI + 15.32 to compute the distances between p and the three tags, respectively. Considering the three distances we get are displayed in feet, we convert the distances in feet into distances in meters (dist1, dist2, dist3 for tag A, tag B, tag C) by using the formula 1m=1foot\*0.3048.
2. We draw a circle (circle A) centered at tag A with a radius of the distance between point p and tag A. Then we draw a circle (circle B) centered at tag B with a radius of the distance between point p and tag B. The data we get from iBeacon consists of 4 elements, UUID, Latitude, Longitude, and RSSI. Before solving the equations of circles, we need to convert the latitude and longitude of tags(lat1, long1 for tag A, lat2 and long2 for tag B, lat3 and long3 for tag C) into values(x, y) in rectangular coordinate system:

x = Latitude\*[(π\*R)/180]

(R is the approximate value of radius of the Earth, and R=6370856 meter);

y = Longitude\*[(π\*R)/180].

1. If circle A is tangent with circle B, then the only one point of tangency is point p.



p

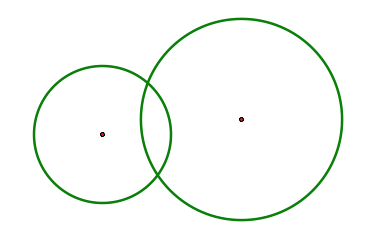
Figure 1: Two tangent circles

So we solve the binary quadratic equations (1) and (2) below and compute the latitude (lat) and longitude (long) of point p:

[(lat1-lat) \*(π\*R)/180]2 +[(long1-long) \*(π\*R)/180]2 =(dist1)2  (1);

[(lat2-lat) \*(π\*R)/180]2 +[(long2-long) \*(π\*R)/180]2 =(dist2)2  (2).

1. If circle A is intersecting with circle B (equation (1) and (2) still meet the circumstance), one of the two points (point p1, p2) of intersection will be point p, i.e., user’s location.



P1

P2

A

B

Figure 2: Two intersecting circles

To determine this, we test the p1 and p2 respectively to see which one of the two satisfies the equation:

[(lat3-lat) \*(π\*R)/180]2 + [(long3-long) \*(π\*R)/180]2 = (dist3)2  (3).

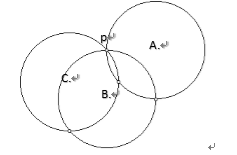


Figure 3: All three intersecting circles intersect in one common point

1. We get the user’s location: p (latitude, longitude). After following step 2, latitude and longitude of user’s position is determined. Also, a blue flag representing the location of the user will show up in the map. Hence, the localization is accomplished.

# Phase 2: Client-Server and Way-finding Technique

1. ***Application Details***

We have designed a technique to find ways for the responder to go to the hazard sites and rescue the victims.

**Client-side (Victims):**

1. Fill in a report about basic hazard information and his/her own information, and the information they submitted is ;
2. Submit his/her information with a hazard and deliver to the responder;
3. Check whether he/she is picked up or not.

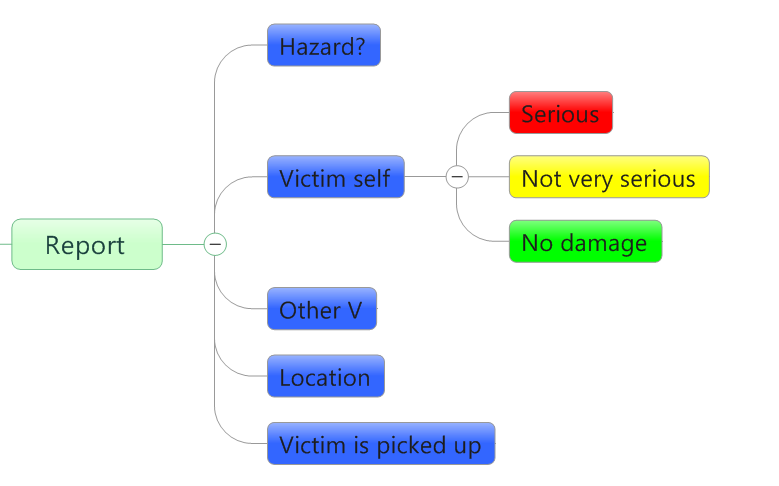


Figure 4 User report accident

**Server-side (Responder):**

1. Add his/her location on any place;
2. Receive information from victims;
3. Add victims’ information;
4. Find a way to get the victims;
5. Edit or delete a victim.
6. ***Implementation Details***

**Client-Server Architecture:** In phase 2, we are doing with main framework with two phones communicating with each other. To create communication between client and server, first get server’s IP address. By knowing server IP address, the client connects to the server. Then the server allocates a port to one client, which is used for report sending, receiving and other communication.

Allocate a port to client

Communicate through port

Get IP address from server

Figure 5 Server-Client Communication

In the first part of way-finding technique, we use a Google Maps API to get directions. It sets lots of nodes on the map for routing. Then when the responder reports the destination, the routing between the location of the responder and the destination (i.e. the basement of Marcus building) will be computed and displayed as polylines on the map. Here is an example of getting the direction from downtown, Amherst (where the responder was located), to the basement of Marcus Building.

**Way-finding technique:** in this second part, we developed a solution to decide the route. First, a case is created when a certain hazard spot is reported. Then we implement the whole technique for this case. The location sent includes specific values of latitude and longitude. Also, we set 30 nodes in the basement, each of which is next to one of the iBeacon tags or located at a specific spot. According to the location of the hazard site, one nearest node to it is calculated and set as the destination. For the responder’s location and the destination and the nodes between, Dijkstra’s Algorithm is used to compute the shortest path.

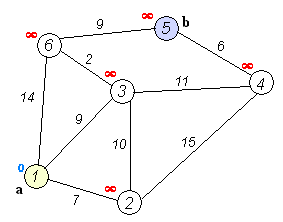


Figure 6 Dijkstra’s Algorithm

For the given source vertex (responder’s location) in the graph, as the graph shown below, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other nodes. Then the responder follows the path to pick up victims. If the responder sees other victims along the way, it can pick up the victims incidentally. The rescue for a case is accomplished when the responder arrives at the destination.

Here is how Dijkstra’s Algorithm works. Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes unvisited. Set the initial node as current. Create a set of the unvisited nodes called the unvisited set consisting of all the nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Select the unvisited node that is marked with the smallest tentative distance, and set it as the new "current node" then go back to step 3.