RESEARCH ARTICLE OPEN ACCESS

Survey of Geofencing Algorithms

Pratik Deshmukh[1], Anuja Bhajibhakre[2], Shubham Gambhire[3], Aman Channe[4] , Dr. Neeta Deshpande[5]

1(Department of Computer Engineering, Pune/D.Y.Patil College Of Engineering Akurdi)
2(Department of Computer Engineering, Pune/D.Y.Patil College Of Engineering Akurdi)
3(Department of Computer Engineering, Pune/D.Y.Patil College Of Engineering Akurdi)
4(Department of Computer Engineering, Pune/D.Y.Patil College Of Engineering Akurdi)
5(Department of Computer Engineering, Pune/HOD Computer Department at D.Y. Patil College Of Engineering Akurdi)

ABSTRACT— An large number of today's mobile applications benefit of location-based services (LBS) in order to notify mobile users about location dependent content or to execute location-dependent actions once the user enters or leaves a dedicated zone. Geofencing is a feature in a software application that uses the global positioning system (GPS) or radio frequency identification (RFID) to define geographical boundaries.A geo-fence is a virtual barrier.. It is a small geographic area that is defined to generate a location event when a user enters or leaves this geofence . In this paper, we survey different algorithms to determine whether the point is in the given geofence range. This article thus, provides an introduction to geofencing and working of different geofencing algorithms that help identify whether a user is within geofence range or not.

Keywords— ***Geofencing , LBS, Ray Casting, Winding Number, Haversine Formula, GeofencingApi's.***

1. INTRODUCTION

GeoFencing is a boundary or region of interest in the geographical region.Geo-fencing is used for many applications and it provides many benefits to users. One of the major applications for Geo-fencing is security, when anyone enters or leaves a particular area, an alert or text messages has been sent to the user [1]. In military applications this can be used, when the enemy vehicles enter into our boundary it gives an alarm sound to alert the officers and takes necessary actions.

Geofence is also a solution to the commonly occurring problems. For example- It happens that the person is unaware of the blood bank in his area where he can find the required blood at that time, thus he may have to travel far and waste his time. So to overcome this problem geofence technology can be used, where he would specify his range of travel(geofencing radius) and will be able to see all the blood banks within his geofence radius just on his mobile phone so that his time is not wasted.

Geofencing combines awareness of current user's location with awareness of the user's proximity Geofencing combines awareness of current user's location with awareness of the user's proximity to locations which can be specified using its latitude and longitude. Radius is added to adjust the proximity for the location.Hence to define a geofence,i.e. creating a circular area, or fence, around the location of interest , the radius, longitude, latitude are required[3].One can have multiple geofences active at a time,with a limit of 100 per device[3].

Geofence on Android:

On Android, there are different ways to deal with geofences. One can use Google's GeofencingApi. This API is component of Google's Location Apis which has GeofencingRequest, GeofenceApi, Geofence, and GeofencingEvents.. The GeofencingApi class is the entry point for all interactions with Google's geofencing API. The GeofencingApi can be used to add geofence by calling addGeofence() method and remove the geofences by calling removeGeofences() method. Now to check whether a person is within a geofence range we can make use of different algorithms such as Ray-casting, Winding Number ,TWC (Triangle Weight Characterization) and Circular Geofencing using Haversine Formula. Section II discusses the above algorithms in detail.

2. GEOFENCING LGORITHMS

Point In Geofence Framework and Algorithm:

The following Fig. 1 shows basic procedure of accessing whether the point of interest is within the geofence range or not.

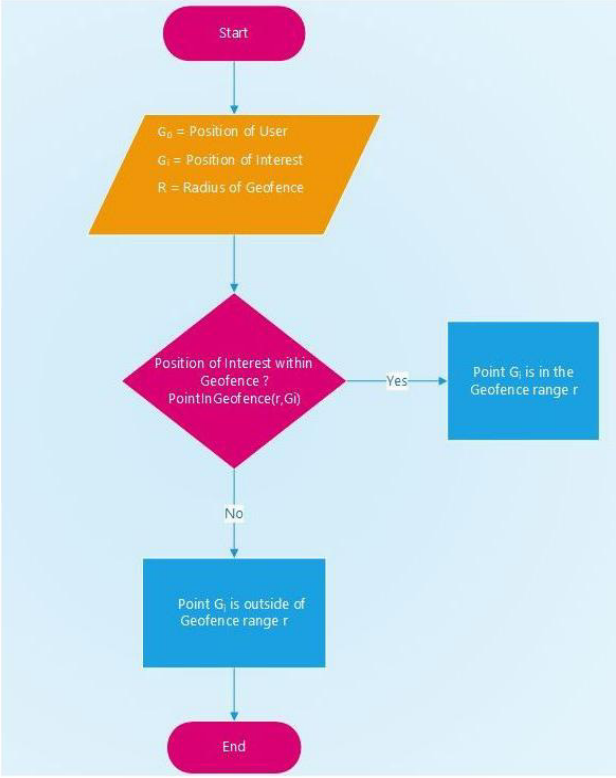

Fig. 1 Point in Geofence flowchart.

The algorithm based on the flowchart is as follows:

Input: **r** is the radius of the geofence. **g** = [g_o , g_i]
g_i is the position of the interest, go is the position of user.

Output: true if **r** does not violate **g**, otherwise false

1: if pointInGeofence(g_i , r)then
2: return true
3: end if
4: for all $g_{o(i)}$ in g_o do
5: if pointInGeofence(r, $g_{o(i)}$) then
6: return false
7: end if
8: end for
9: return true

The above algorithm consists of the input parameters r and g. r = (x, y) is the current position to check for geofence violation.
The geofence is specified by g = [g_i , g_o] where g_i is the keep-in geofence boundary polygon and G_o = {g_{o1} , ..., g_{0n} } is the set of keep-out boundaries. g_{oj} is the jth of n keep-out geofence boundary polygons.

The PointInGeofence() function can be implemented by any of the algorithm like Ray Casting , Winding Number , TWC and Circular Geofencing using Haversine formula.

1. **Ray Casting** :

[4] The Ray Casting algorithm determines whether or not the position of interest, G_i, is inside a given polygon p, by projecting an infinite ray from G_i. If the infinite ray intersects an odd number of polygon edges, then r is contained in p, otherwise, r is outside of p. As the Ray Casting algorithm iterates over all edges of p and does not have an initialization step, if the geofence boundaries change from one time step to the next, code execution and results of the Ray Casting algorithm are not impacted.

Algorithm:

Input: p is a simple polygon
G_i is the position of interest
buf is a buffer distance.

Output: true if p contains **G_i**, otherwise false

1: count = 0
2: s is an infinite ray in the +y direction, originating at **G_i**
3: for all edges e in p do
4: if **G_{i_x}** is within buf of ex then
5: $e_{x,buf}$ = ex -2 * buf
6: else
7: e_{buf} = e
8: end if
9: if **G_i** is within buf of e or e_{buf} then
10: return false

Lines 9 -11 of above Algorithm state that if the position of interest, **G_i**, is within the buffer distance, buf , of the edge currently being considered, then **G_i** is considered outside polygon p.

2. **Winding Number** :

The winding number accurately determines if a point is inside a non simple closed polygon. It does this by computing how many times the polygon winds around the point. [5]

The point is outside only when this "winding number" wn = 0; otherwise, the point is inside.

Algorithm for Winding Number:

Input :
G_i = a point of Interest,

V[] = vertex points of a polygon V[n+1]
with V[n] = V[0]
n = number of vertices

Output :

winding_number
(when the winding_number =0 ,P is outside And winding_number is non-zero if P is inside)

```
PIP_windingNumber ( Point Gi, Point V[], int n )
                    {
        wn = 0;     1:  int                 // the
                    winding number counter

 2: for each edge E[j]:V[j] V[j+1] of the Polygon
do
 3:      if (E[j] crosses upward )
  4:            if (P is  strictly left of E[j])
  5:                  ++winding_number;
  6:            end if
 7:       else if (E[j] crosses downward )
  8:            if (P is  strictly right of E[j])
  9:                  --winding_number;
 10:           end if
11:      end if
12: end for
13: return winding_number;
}
```

3. **Triangle Weight Characterization(TWC):**

Triangle Weight Characterization, consists of an initialization step and a run-time step as shown in Algorithm of TWC[4] .The initialization step must be executed for all keep-in and keepout geofences when the system first activates. If there are any changes to any of the geofence boundaries after the original initialization, each keep-in or keep-out geofence that is changed must be initialized again[4].

The initialization step subdivides each of the original geofences from simple polygons to y-monotone polygons and then to triangles[4] .The run-time step checks whether the position of interest is within each triangle. If the position of interest is inside any of the triangles, then it is within that polygon. Otherwise, it is outside the polygon[4].

Algorithm for TWC :

Input:
p is a simple polygon
$\mathbf{G_i}$ is the position of
interest Output:
true if p contains $\mathbf{G_i}$, otherwise false

Initialization:
1: Divide p to m y-monotone polygons
2: for all y-monotone polygons M in p do
3: Divide polygon M to n triangles
4: end for

Run-Time:
5: for all N triangles in p do
6: if N contains $\mathbf{G_i}$ then
7: return true
8: end if
9: end for
10: return false

Circular Geofencing Using Haversine Formula[7]:

In the below algorithm,

geofence of radius 'r' is created around the point G_o

The distance between G_o and G_i is calculated using Haversine formula.

The Haversine **formula** determines the great-circle distance between two points on a sphere given their longitudes and latitudes [6].

Algorithm:

Input:
G_i the position of the interest. Gi = [lat_i, $long_i$]
G_o is the current position of the user. G_o [lat_o, $long_o$] r is the radius of the geofence.

Output:
True if the G_i is within Geofence range of G_o

```
    checkWithInGeofenceRange ()
    {
1:   Distance = 0
2:   Distance = getDistanceFrom Location ( Go , Gi ) ;
3:        If ( d < r )
4:                Return true
5:        Else
6:                 Return false
7:        End if
    }
  Get DistanceFromLoaction ( Go , Gi)
  {
8:   radius =6371;        //radius of earth
9:   dlat = deg2rad ( lati , -lato );
10:  dlong = deg2rad ( longi, -longo );
11:  a = Math.sin ( dlat / 2 ) * Math. sin( dlat/2 )
       + Math. cos ( deg2rad( lato) )
        * Math. cos ( deg2rad( lati) )
        * Math. sin ( dlong/2 ) *Math. sin ( dlong/2 );

12:  c= 2 * Math.atan2 ( math.sqrt ( a ) ,
          Math.sqrt ( 1-a ) );
13:  D= R* c;
14:  Return d;
  }
```

So, after calculating the distance, if the distance is greater than radius then the point of interest Gi will be discarded and if the distance is less than radius then the point of interest G_i is highlighted with marker within geofence.

The Fig 2. shows the geofence generated around G_o (shown by red marker) and G_i which is inside Geofence of G_o is shown with Blue marker.

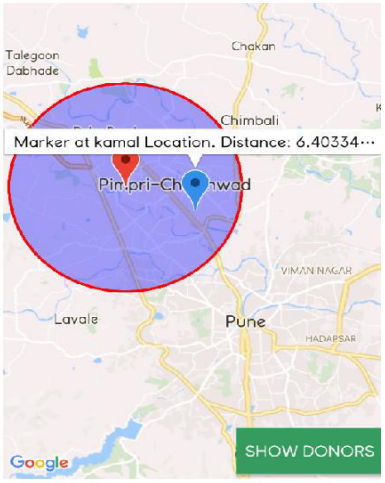

Fig 2. Geofence generated around G_i

I. DISCUSSION

1. Geofences can be of any shape circular or polygon.There are some limitations to what shape csn be used. For example ,When a geofence is to be added around a single beacon,circle is the optimal shape.

2. Generally calculation that is required for detecting polygonal geofences is much heavier, so it will not be as fast or reliable as with circular shapes[8].

3. It is also observed that circular fencing is easy compared to polygonal fencing in terms of implementation[7].

4. When Winding number algorithm is compared to cross number algorithm like ray casting, winding number algorithm has the same efficiency as the analogous ray casting algorithm. Since winding number algorithm is more accurate in general, the winding number algorithm is preferred method to determine inclusion of a point in an arbitrary polygon[5].

5. [4] has studied Ray Casting, Fast Ray Casting and TWC algorithms and were applied to 50000 randomly generated nearby positions of interest. The time (in seconds) required to run all 50000 points for each of the methods is observed. Based on the results they conclude that either Fast Ray Casting or TWC were good choices for the geofence violation.Fast Ray Casting and TWC execute inapproximately the same amount of time, but only TWC requires an initialization step and thus Fast Ray Casting algorithm would be the best choice since it does not require an initialization step, unlike TWC[4].

The difference between Ray Casting and Fast Ray Casting is that Ray Casting includes a check

of the position of interest's proximity to the geofence boundaries. The edge proximity check is a useful feature because it enables to usage of a buffer distance to allow for state estimation Imprecision [4].

II. CONCLUSION

This paper has discussed about Geofencing in brief and four different algorithms-Ray casting, TWC, Winding number and Circular Geofencing using Haversine formula to check whether the point in within the geofence .The shape of the geofence can either be circular or a polygon.Ray Casting, TWC and winding number algorithms are based on polygon shaped geofences whereas Circular geofencing using haversine is based on circular geofence.Thus, it is inferred that between both types of geofences , circular geofences are more easier to implement than polygon shaped geofences.The survey thus studied concept of Geofencing ,four point in geofence algorithms, and their comparisons.

III. REFERENCES

[1]. Sandro Rodriguez Garzon,Dmytro Arbuzin and Axel Kupper,"Geofence Index: A Performance Estimator for the Reliability of Proactive Location-based Services",2017 IEEE 18th International Conference on Mobile Data Management.

[2]Sachin W. Rahate, Dr. M.Z. Shaikh," Geo-fencing Infrastructure: Location Based Service",International Research Journal of Engineering and Technology (IRJET),Nov 2016.

[3]Creating and Monitoring Geofences, https://developer.android.com/training/location/geofencing.html

[4]Mia N. Stevens Hossein Rastgoftar and Ella M. Atkins, "Specification and Evaluation of Geofence Boundary Violation Detection Algorithms", 2017 International Conference on Unmanned Aircraft Systems (ICUAS) June 13-16, 2017, Miami, FL, USA.

[5]Dan Sunday,Inclusion of point in polygon , http://geomalgorithms.com/a03-_inclusion.html

[6] https://en.wikipedia.org/wiki/Haversine_formula

[7] P.L.Pratyusha , V.P.S.Naidu " Geo-Fencing for Unmanned Aerial Vehicle", National Conference "Electronics, Signals, Communication and Optimization" (NCESCO 2015).

[8]Kalle,"What is Geofence ? –A complete guide to geofencing" , https://proximi.io/geofence-complete-guide-geofencing/ , January 31 ,2018