Tinder Privacy Revised

Project Report

Joey Dreijer, Eric van den Haak

June 1, 2014
Abstract

This research focuses on privacy aspects regarding Tinder. Research has been done to find out the exact location of Tinder users and match them against their Facebook profile. With clever use of the Tinder API combined with Trilateration, it is possible to find a user’s whereabouts within an approximate 1 mile radius. Using a flaw in the Tinder API allows to identify a user’s exact whereabouts. With manual labor it is possible to find out the real Facebook profile of a Tinder user, resulting in having a profile along with exact location. Concluded is that Tinder is on the good way to improve on their clients’ privacy but still has some flaws.
Contents

1. Introduction ................................. 2
   1.1. Research Question ...................... 2
2. Related Work .................................. 3
3. Requirements .............................. 4
   3.1. Ethical considerations ................... 4
   3.2. Technical requirements ................... 4
4. Research .................................... 5
   4.1. Tinder initials ........................... 5
   4.2. Tinder API ............................... 6
      4.2.1. Request new batch of users ........... 6
      4.2.2. Change current location ............... 8
      4.2.3. Change settings and preferences .......... 9
   4.3. Locating users ........................... 11
      4.3.1. Triangulation ....................... 11
      4.3.2. Trilateration ....................... 12
   4.4. Facebook API ............................. 15
   4.5. Proof of concept and attack scenario ....... 16
5. Conclusion .................................. 19
6. Future work ................................ 20
A. Trilateration Class ......................... 22
1. Introduction

Tinder is a popular dating app that creates around 3.5 million possible matches each day. Tinder uses an existing social platform (Facebook) for authentication and matching parameters. Tinder knows where you are and what you like and hopes to create ‘smart’ dating matches by using this data. However, sharing your location and Facebook data with Tinder can also be dangerous. Personal details can be shared with third parties, or even worse, be abused by people with less ethical intentions. Tinder tries to make the data as anonymous as possible so that malicious users are not able to find your location or Facebook profile. But what if there is a flaw in this anonymization process?

This paper will provide the reader with an extended review how we make use of public data provided by Tinder and reverse-match Tinder accounts back to their original Facebook accounts and track their current location. The paper will explain how we abuse Tinder-provided data and demonstrate a proof-of-concept ‘Tagging/Tracking’ application.

1.1. Research Question

Our main goal is to link Facebook profiles (and thus personal information) to a user’s exact location. Therefore it is necessary to first exploit Tinder to find out user locations. Another requirement is the possibility to link Tinder profiles to real Facebook profiles. When combined, we would like to be able to see if for example it is possible to find out the exact location of a person by just knowing his or her cellphone number. For this we have come up with the following research questions;

Is it possible to find exact whereabouts of Tinder users based upon minimal information?

- Is it possible to exploit the Tinder API in a way that the exact whereabouts of users can be found?
- Can the user profiles of a "stripped" Tinder profile be matched against a real Facebook profile?
2. Related Work

The Tinder API has been reviewed by SNE (System and Network Engineering) students [6] at the University of Amsterdam. This research shows the information that is sent by Tinder from user to user which can aid in this research. The research focussed on the security measures implemented by Tinder.

Another research was performed by a New-York security company to locate Tinder users[8], indicating the possibility to find out exact whereabouts by triangulation. In a response to this research, Tinder’s official statement showed that they do not take this flaw seriously: “Include Security identified a technical exploit that theoretically could have led to the calculation of a user’s last known location” [5] [7]. Tinder tries to make the problem look less serious by claiming that this exploit can only theoretically be abused and that there are no practical implementations found yet.
3. Requirements

3.1. Ethical considerations

By locating specific users and reverse-tracking original Facebook profiles, the ethical line may be broken by intruding other user’s privacy. During the research, several accounts were created for testing purposes. Public data belonging to other (non-test) users may have been received, but were not stored in plain. Only a snipped of the data has been used. The real name and Facebook likes of a user were not stored. Only if a Tinder users gave explicit permission for his/her data to be collected for the purpose of this research, more relevant data, such as names and likes, may have been used for reference and testing purposes. The content of this research will be publicly available. The reader is free to do as he/she likes with the given conclusions and proof-of-concept. This research will hopefully stimulate Tinder to take the necessary steps in order to guarantee the user’s privacy.

3.2. Technical requirements

The Tinder application is available on both iOS (Apple) and Android (Google). When installing Tinder, the user should already have an existing Facebook account. Facebook is used for authentication and matching. For this purpose, we created an ‘active’ Facebook account by the name of ‘Alice Weaver’. Alice has 16 friends on Facebook and is following (liked) different type of pages ranging from sports to musicals. Tinder shows common interest for each match by comparing shared Facebook-likes.

Limited tools and applications were used during the research. Below is a list of all the tools and applications that have been used:

- **Burp Suite**: Burp was used to intersect the Tinder requests. Tinder requests and responses are sent via JSON over HTTP(S).

- **Pyton**: A Python application was created to perform the Tinder localization requests. The application is able to spoof GPS data and request all Tinder users within a given range.

- **MySQL**: The data collected by the Python application was stored in a MySQL database for future reference.

- **PHP/GoogleAPI**: The web application collects all relevant data from the MySQL database for parsing. Using PHP and the Google Maps API, a map is created that displays the location of all Tinder users collected by the Python application.
4. Research

4.1. Tinder initials

From a user-perspective, Tinder is a very easy to use dating application. The user gets around 40 possible matches based on his/her preferences. The user will only see a profile photo and common likes for each match. Whenever a Tinder user accepts a match, he/she will be able to open a chat conversation when the other user mutually accepts the user. After all 40 matches have either been accepted or denied, a new batch is requested. This loop will continue to occur until no new matches are found. The matches received depend on the user’s preferences. A user is able to specify the maximum distance to/from the matching Tinder user and if the match should be male or female.

In normal operation, the either accepted or denied match should no longer appear in future requests. However, this limitation can be bypassed by re-registering (ie. removing the account and registering again). The image below shows a shortened version of the Tinder process-flow.

![Generic flow of tinder](image)

Figure 1: Generic flow of tinder
4.2. Tinder API

We already know what kind of data is being sent to/from the Tinder back-end based on the previous OS3 research[6]. However, Tinder released several new updates which seem to alter the results received.

- Previously, Tinder allowed a user to fetch his/her matching history. Sending the ‘history’ request to the Tinder server will now result in an empty / blank JSON response.

- Tinder no longer allows to receive a variable amount of matches. In the past, a user could request more than the default amount (40).

- The ‘distance to match’ changed from a float to int datatype. This means (in normal operation), you will no longer see the exact distance to user. This specific topic is further discussed in the next subsection.

The default functionality still works and the data we require is still being processed by Tinder. Three different type of requests are relevant for our research. In the following subsection, the different relevant requests and responses are displayed, exploiting the data will be described later. The requests important for our research are:

1. Request new batch of users (fixed amount)
2. Change current location
3. Change settings and preferences

4.2.1. Request new batch of users

The request most commonly used involves fetching new possible matches. The response of this request contains relevant information to pin-point the user’s location. Receiving a new batch of possible matches is sent in the following format:

```
POST /user/recs HTTP/1.1
app_version: 633
platform: android
...
X-Auth-Token: 41294818-ef313-489-fj182-192938136ab
os_version: 19
Host: api.gotinder.com

{"limit":40}
```
As explained in **Tinder Initials** section, the amount of fetches being received is static. The limit is constantly set to ‘40’ (see example above). Changing this value does not alter the amount of matches received. The below JSON response is partially altered to not include real user data. However, the format is identical to the original.

```
"results": [  
  {  
    "distance_mi": 1,  
    "common_like_count": 1,  
    "common_friend_count": 0,  
    "common_likes": [ 45818581 ],  
    "common_friends": [ ],  
    "_id": "5297a3d844c6efa144000076",  
    "bio": "Generic Bio located here\n",  
    "birth_date": "1990-13-37T00:00:00.000Z",  
    "gender": 0,  
    "name": "Joey",  
    "ping_time": "2014-05-12T14:30:06.890Z",  
    "photos": [  
      {  
        "url": "http://images.gotinder.com/imagelocation.jpg",  
        "processedFiles": [  
          {  
            "url": "http://images.gotinder.com/imagelocation.jpg",  
            "height": 640,  
            "width": 640  
          }  
        ],  
        .... Repeat image list/array for each unique image and size  
      },  
      "extension": "jpg",  
      "fileName": "imagenamerandomized.jpg",  
      "xdistance_percent": 0.75,  
      "yoffset_percent": 0,  
      "xoffset_percent": 0.16805554926395416,  
      "id": "412443-e1244-5431-4feg-41255118b0e2",  
      "ydistance_percent": 1  
    },  
    "birth_date_info": "fuzzy birthdate active, not displaying real birth_date"  
  }  
]
```

For our research, only some of this data is relevant. This contains:

- **uid**: Tinder ID (not the original, but hashed)
- **common likes**: Common Facebook likes
• common friends: Common Facebook friends
• Distance mi: Distance in miles to the user/match

Only the tinder ID and distance mi will be sufficient for localization purposes. The remainder of the data is used for Facebook profile reversal.

During previous tests being performed on Tinder, the distance in miles was returned as a Float datatype specifying sub-comma distances. This means the returned distance mi could be ‘2.139491943’ instead of simply ‘2’. This security ‘feature’ means that no near-exact localization can be performed; whenever a user is located inside a 1 mile radius, Tinder will simply say that the user is located within ‘1’ mile distance.

However, we found a bug in the Tinder preference system. When using the app, the user can not select a lower matching distance than 1 mile or Kilometre. Using a custom request can set the matching distance to 0.01 (example) mile. Supplying this setting will only return the tinder matches located within the current 0.01 mile radius. The response will say that the match is 0.01 mile away from the current location. A possible attack scenario using this vulnerability will be displayed in the "Scenarios" subsection.

4.2.2. Change current location

Tinder is frequently sending out location updates. Tinder will store the new location whenever the app is opened (and during use). Since the location of the user is a key-variable for matching, this is not a very strange thing to do from Tinder’s point of view. The request looks like follows:

```
1 POST /user/ping HTTP/1.1
2 app_version: 633
3 platform: android
4 ...
5 X-Auth-Token: 41294818-ef313-489-fj182-192938136ab
6 os_version: 19
7 Host: api.gotinder.com
8
9 {"lon":4.9556716,"lat":52.3543506}
```

The latitude and longitude are received from the mobile-phone’s GPS. Tinder will send a response whether or not the GPS coordinates are successfully received. For a still unknown reason, changing the location via our custom request will not receive a response from Tinder at first try (the location is still stored however). Performing the request a second time will result in a warning message from Tinder:
This message is not relevant, since Tinder still has the new GPS coordinates stored inside the database. This can be confirmed by changing the user’s settings and verifying the response. With each setting the user changes, Tinder sends all the known data back to the user. This also includes the currently set longitude and latitude. We were able to change our location from city to city without getting a warning or time-out. We moved from Utrecht to Amsterdam and from Amsterdam to Beverwijk in less than 2 seconds. This is an important flaw which makes efficient localization possible, since three different locations on the map will be required to measure distances (More about this subject in the "Locating users" section).

4.2.3. Change settings and preferences

Changing the Tinder profile settings can specify how far away each match is allowed to be. This is important to find people within a specific range. Each setting change will result in a POST being sent to the Tinder server:

```
POST /profile HTTP/1.1
app_version: 633
platform: android
...
X-Auth-Token: 41294818-ef313-489-fj182-192938136ab
os_version: 19
Host: api.gotinder.com

{"age_filter_min":23, "gender":0, "gender_filter":0, "distance_filter":11}
```

In the example above, we changed our search radius to 11 miles (as can be seen in the distance filter variable). In the previous subsection, we mentioned that each location change can be verified in Tinder’s response to a profile change. Tinder will send back the following data whenever a profile change is requested:

```
{"promoted_out_of_date":true,"_id":"5382f5099c34d48079d7dcd","age_filter_max":1000,"age_filter_min":23,"banned":[],"bio":"Research project tbv. User Tracking. Vieler spass <3","birth_date":"1985-04-01T00:00:00.000Z","create_date":"2014-05-26T08:02:16.507Z","facebook_id":"100008235517556","gender":1,"gender_filter":0,"location":{"id":"11177152182360","name":"Amsterdam, Netherlands"},"name":"Alice","ping_time":"2014-06-01T20:33:02.201Z","position":{"at":1401654785317,"lon":4.95434915933578,"lat":52.328604505813},"photos":[]},"interested_in":[]},"photos":[]},"interested_in":[]},"photos":[]},"interested_in":[]},"photos":[]}
```

The distance filter variable has successfully been changed to 11 and our current (or last updated) lat/long is also located inside the same response. Changing our
search radius is an important factor for our research. We will demonstrate how this variable is used in the "Proof of concept and attack scenario" section.
4.3. Locating users

Within this section the used mathematics are described. Tri-ang-ulation is based on angles as tri-lat-eration is based on degrees.

4.3.1. Triangulation

Our first approach was to use triangulation in order to find the whereabouts of a Tinder user. As the formula itself worked, the result was not correct because we used the latitude(lat) and longitude(lng) as respectively x and y coordinates. As the lat and lng are degrees this did not work.

The Formula

The triangulation formula we used was provided by AmbrSoft[1].

First, you will have to calculate the distance between the two circle centers;

$$ D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$  \hspace{1cm} (1)

After this is done, calculate the area of the triangle between the circles\(^1\)

$$ A = \frac{1}{4} \sqrt{4r_0^2r_1^2 - (r_0^2 - r_1^2D^2)^2}$$  \hspace{1cm} (2)

Now you will be able to calculate the x and y coordinates of the intersections;

Calculating x

$$ x_{1, 2} = \frac{x_1 + x_2}{2} + \frac{(x_2 - x_1)(r_0^2 - r_1^2)}{2D} \pm 2 \frac{y_1 - y_2}{D^2} A$$  \hspace{1cm} (3)

Calculating y

$$ y_{1, 2} = \frac{y_1 + y_2}{2} + \frac{(y_2 - y_1)(r_0^2 - r_1^2)}{2D} \pm 2 \frac{x_1 - x_2}{D^2} A$$  \hspace{1cm} (4)

\(^1\)Using Heron’s formula[https://en.wikipedia.org/wiki/Heron’s_formula](https://en.wikipedia.org/wiki/Heron’s_formula)
4.3.2. Trilateration

Our next approach is trilateration. Trilateration works by having 3 points in a 3 dimensional world with a radius. For each trilateration it is thus necessary to have exactly 3 x,y,z coordinates and a distance. As coordinates are provided in latitude an longitude, conversion is mandatory. Because the lat and long are degrees based upon the earth, the earth radius is needed to calculate these. In Amsterdam, the earth\( (earthR) \) radius is approximately 6378.36km. Equations have been found using Wikipedia\([4, 3]\) and Stackexchange\([2]\). Our own implementation can be found in appendix A.

Visualization

![Diagram of trilateration](https://en.wikipedia.org/wiki/File/3spheres.svg)

Figure 2: Idea is to find the intersection\(^2\)

---

\(^2\)Source: https://en.wikipedia.org/wiki/File\protect\kern+.2222em\relax3spheres.svg
### Constants

<table>
<thead>
<tr>
<th>Position</th>
<th>lat</th>
<th>lng</th>
<th>radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lat₁</td>
<td>lng₁</td>
<td>r₁</td>
</tr>
<tr>
<td>2</td>
<td>lat₂</td>
<td>lng₂</td>
<td>r₂</td>
</tr>
<tr>
<td>3</td>
<td>lat₃</td>
<td>lng₃</td>
<td>r₃</td>
</tr>
</tbody>
</table>

### Cartesian coordinates

In order to work with the triangulation formula, these coordinates have to be translated to a Cartesian Coordinate System\(^3\).

First convert degrees to radians;

\[
radians = \frac{degrees \times \pi}{180}
\]  

(5)

Then we can calculate the x, y and z coordinates for each circle and put it into a vector;

\[
x = earthR \times \cos(lat) \times \cos(lng)
\]  

(6)

\[
y = earthR \times \cos(lat) \times \sin(lng)
\]  

(7)

\[
z = earthR \times \sin(lat)
\]  

(8)

\[
P = \{x, y, z\}
\]  

(9)

---

\(^3\)https://en.wikipedia.org/wiki/Cartesian_coordinate_system
Transformation

The next step is to transform the coordinates in a way that circle 1 is the origin and circle 2 is on the X-axis of circle 1. Then is to calculate the $d$, $i$, and $j$ in the picture.

\[
ex = \frac{(P_2 - P_1)}{\|P_2 - P_1\|} \quad (10)
\]
\[
i = \text{dot}(ex, P_3 - P_1) \quad (11)
\]
\[
ey = \frac{(P_3 - P_1 - i \times ex)}{\|P_3 - P_1 - i \times ex\|} \quad (12)
\]
\[
ez = \text{cross}(ex, ey) \quad (13)
\]
\[
d = \|P_2 - P_1\| \quad (14)
\]
\[
j = \text{dot}(ey, P_3 - P_1) \quad (15)
\]

Intersection

Now the intersection point relative to the transformation can be calculated.

\[
x = \frac{r_1^2 - r_2^2 + d^2}{2d} \quad (16)
\]
\[
y = \frac{r_1^2 - r_2^2 + i^2 + j^2}{2j} - \frac{i}{j}x \quad (17)
\]
\[
z = \pm \sqrt{r_1^2 - x^2 - y^2} \quad (18)
\]

Re-transformation

Calculated are the $x, y, z$ coordinates of the intersection relative to the transformation. These now have to be re-transformed in order to get the right coordinates ($iV$ = intersection vector).

\[
iV = P_1 + x \times ex + y \times ey + z \times ez \quad (19)
\]
\[
\text{latRad} = \text{asin}(iV[2]/\text{earthR}) \quad (20)
\]
\[
\text{lngRad} = \text{atan2}(iV[1], iV[0]) \quad (21)
\]

The last step is to convert the found lat/lng to degrees again;

\[
degrees = \text{radians} \times \frac{180}{\pi} \quad (22)
\]
4.4. Facebook API

Facebook introduced a new search function called "Graph Search". Using Graph Search, it’s possible to find new Facebook friends with common interests. The search is very extensive and supports many options, (nearly) every variable on a Facebook profile can be included in a search. I could try to look for "People named ‘Alice’ that like Disturbed and like Bruno Mars and live in Amsterdam, the Netherlands". However, due to unfortunate Facebook decisions, it is no longer possible to do graph search via Facebook’s new API system. Facebook disabled Graph Searches via their API after introducing their new v2.0 API kit. Therefore this process could not be automated. However, it is still possible to match users to their real Facebook profile by hand as the ‘normal’ Facebook interface still has Graph Searches enabled. We tried to use the Facebook GUI to automate the process, but due to click/location trackers this is a very complicated thing to do. We tried to focus on a real-working example of Tinder tracking instead.

When you want to find a Tinder user’s Facebook profile, you need to make sure your custom Facebook account (the one you are using for Tinder tracking) likes most of the popular pages you can find. To improve your search results, it is advisable to like/follow some specialized pages as well; think of lesser known death metal-band. This will greatly improve the accuracy and speed of your search. It is important to note that Graph Search will only display results of users that do not have their Facebook profile blocked for searched or have strict privacy settings.

Let’s give a small demo: Meet Alice Weaver (on the right). She is the custom Facebook account we are using for this research project. Alice likes many different bands; ranging from Justin Bieber to Metallica. She doesn’t have many friends (or a stable relationship for that matter), but by using Tinder she hopes to find the love of her life. Alice is using Tinder and happens to see many people having the same interests. Tinder sends back the Facebook Page ID’s of the facebook pages both Alice and the other Tinder user have in common. This Facebook Page ID can be reverted to it’s normal page name via fbider.com. After the ID has been reversed, you can manually graph search for users with all the data you now have. This data contains:
• First name of Tinder user (sent by Tinder app)
• Common likes (sent by Tinder app, name reversed via 3rd party)
• Person’s location (received by our Tinder tracker application)
• Person’s birth-date (also sent by Tinder app)

All of this information combined can be put inside a single graph search. Result:

![Graph Search result](image)

Figure 4: Graph Search result (generic 1 mile)

### 4.5. Proof of concept and attack scenario

In order to visualize the trilateration process, a proof of concept application is developed. When using the application it is possible to choose 3 points on the world and do live tracking. The application will find all of the users located within a given range and apply trilateration to find their location (1 mile accurate). When a user of interest is found, the user can narrow down his search by selecting a sub-decimal (float) search distance and apply another three points on the map. The user can repeat the process to get more accurate localization results. To further demonstrate how we are able to track users by using our research results and proof-of-concept application, the scenario demonstrated below should provide a more concrete example:
Example A

An attacker (malicious user) called Trudy wants to locate girls with common Facebook interests. Trudy doesn’t want to manually go through all of the Tinder matches manually since this is time consuming. Next to that, he prefers to find girls living in a rural/city environment. So to accomplish his goal, Trudy has to build some sort of application to automate the process. Trudy will also have to create a fake Facebook account with common interests (likes) to authenticate with Tinder.

In the end, trudy developed a web-application called ‘Torchwood’ to localize all Tinder users in a variable region. Trudy sets three (blue) points on the map (as seen in the image below) to find all of the girls nearby.

![Map of Torchwood application](https://example.com/torchwood-map.png)

Figure 5: Finding people (generic 1 mile)

After the query is done, red markers will be displayed on the map to indicate a Tinder user’s approximate location. However, Trudy isn’t satisfied with the results. He wants find people and know their specific location. Trudy sets 3 points on a specific street and narrows down his search distance to 0.1 mile.
Example B

Finding out the location of nearby users can also be used in a different (less obvious and requires less Tinder requests) scenario. Let’s say Trudy already knows where his ‘target’ works and/or lives. Trudy will simply mark the locations on the map and requests all nearby Tinder users within a 0.2 mile radius. Whenever the target arrives at work, Trudy will get a notification that the user is within the given area. If Trudy wants to perform a burglary at the target’s house, the perfect moment will be when Trudy finds out that his/her target is at office. This example requires much less requests, since this does not require trilateration and does not necessarily stay inside a loop to find ALL the Tinder users (only until he finds his specific target).
5. Conclusion

Is it possible to exploit the Tinder API in a way that the exact whereabouts of users can be found?

This research shows that it is harder but still possible to find the exact whereabouts of a Tinder user. This can be done by trilateration combined with a flaw in the API. Trilateration can be used to find a user’s whereabouts with an approximate 1mi radius. There is a flaw in the API that allows to use a float (ex. 0.391) to narrow down the search radius, what thus can be smaller than 1mi. Tinder will still round the value of the distance to 1mi, but in fact it is known that the person is not further away than the given radius. An exact location can be found by combining the smaller search radius and many more different (spoofed) GPS locations.

Can the userprofiles of a “stripped” Tinder profile be matched against a real Facebook profile?

Reversing Tinder profiles is possible with manual labor. Automating this is no longer possible because Facebook closed their graphsearch API. With information provided by the Tinder API (Pictures, Name, Distance, Common likes and Common friends) it is possible to find these users with Facebook’s GUI.

Is it possible to find exact whereabouts of Tinder users based upon minimal information?

Concluded is that it is still possible to find exact whereabouts of Tinder users. It has however become more complicated than before as Tinder rounds the distance to a certain user. Our advise to Tinder is to enforce the use of integers to set the search radius. Another advise is to put a “speedlimit” on users as it is now possible to travel at light speed. An additional option might be to limit the amount of request per user per minute. This will circumvent trilateration.
6. Future work

Automating Facebook matching

As it turned out, Facebook has closed its graph search API. Due to a lack of time, it was not possible to research an automation process via the Facebook GUI. Future research may automate this.

Picture matching

Tinder allows only to use pictures that are also on Facebook. To be sure that the Tinder user is matched with the right Facebook profile, Facebook profile pictures can be downloaded and matched against the Tinder pictures provided by the API.
References


Appendices

A. Trilateration Class

```php
<?php
/**
 * Trilateration class
 * @author Eric van den Haak
 * @author Joey Dreijer
 * @link http://gis.stackexchange.com/questions/66/trilateration-using-3-latitude-and-longitude-points-and-3-distances inspiration
 */
class TT_Trilateration extends TT_Trilateration_Abstract {

    /**
     * Intersect.
     *
     * @return \TT_Coordinate
     * @todo Circle getX and getY actually should be getLat and getLng.
     * @todo Fixed earth.
     */
    public function getIntersection() {
        $circles = $this->getCircles();
        
        if (count($circles) < 3)
            throw new TT_Exception('Not enought circles.');

        $circles = array_slice($circles, 0, 3);

        $c1 = current($circles);
        $c2 = next($circles);
        $c3 = next($circles);

        // Earth, fixed for now..
        $earthR = 6378.36;

        // CI
        $latA = $c1->getCoordinate()->getX();
        $lonA = $c1->getCoordinate()->getY();
        $distA = $c1->getR();

        // C2
        $latB = $c2->getCoordinate()->getX();
        $lonB = $c2->getCoordinate()->getY();
        $distB = $c2->getR();
    }
}
```
// C3
$latC = $c3->getCoordinate()->getX();
$lonC = $c3->getCoordinate()->getY();
$distC = $c3->getR();

// C1 coords
$xA = $earthR * (cos(deg2rad($latA)) * cos(deg2rad($lonA)));
$yA = $earthR * (cos(deg2rad($latA)) * sin(deg2rad($lonA)));
$zA = $earthR * (sin(deg2rad($latA)));

// C2 coords
$xB = $earthR * (cos(deg2rad($latB)) * cos(deg2rad($lonB)));
$yB = $earthR * (cos(deg2rad($latB)) * sin(deg2rad($lonB)));
$zB = $earthR * (sin(deg2rad($latB)));

// C3 coords
$xC = $earthR * (cos(deg2rad($latC)) * cos(deg2rad($lonC)));
$yC = $earthR * (cos(deg2rad($latC)) * sin(deg2rad($lonC)));
$zC = $earthR * (sin(deg2rad($latC)));

// Circle coordinates
$p1 = array($xA, $yA, $zA);
$p2 = array($xB, $yB, $zB);
$p3 = array($xC, $yC, $zC);

// Transform
$ex = $this->arr->divide($this->arr->minus($p2, $p1), $this->arr->norm($this->arr->minus($p2, $p1)));
$i = $this->arr->dot($ex, $this->arr->minus($p3, $p1));
$ey = $this->arr->divide($this->arr->minus($this->arr->minus($p3, $p1), $this->arr->multiply($i, $ex)), $this->arr->norm($this->arr->minus($this->arr->minus($p3, $p1), $this->arr->multiply($i, $ex))));
$ez = $this->arr->cross($ex, $ey);
$d = $this->arr->norm($this->arr->minus($p2, $p1));
$j = $this->arr->dot($ey, $this->arr->minus($p3, $p1));

// Relative coords
$x = (pow($distA, 2) - pow($distB, 2) + pow($d, 2)) / (2 * $d);
$y = ((pow($distA, 2) - pow($distC, 2) + pow($i, 2) + pow($j, 2)) / (2 + $j)) - (($i / $j) * $x);
$z = sqrt(abs(pow($distA, 2) - pow($x, 2) - pow($y, 2)));

// Intersection vector
$IV = $this->arr->plus($p1, $this->arr->plus($this->arr->multiply(array($x, $x, $x), $ex), $this->arr->plus($this->arr->multiply(array($y, $y, $y), $ey), $this->arr->multiply(array($z, $z, $z), $ez))));

// Convert back to lat lng
$lat = rad2deg(asin($IV[2] / $IV));
$lon = rad2deg(atan2($IV[1], $IV[0]));
return new TTCoordinate($lat, $lon);

/**
 * Normalise array
 * @param type $array
 * @return float
 */
protected function arr_norm($array) {
    if (count($array) < 2)
        return 0;
    $tot = 0;
    foreach ($array as $val) {
        $tot += pow($val, 2);
    }
    return sqrt($tot);
}

/**
 * Minus array
 * @param array $arr1
 * @param array $arr2
 * @return array
 * @throws TTException
 */
protected function arr_minus($arr1, $arr2) {
    if (!is_array($arr2)) {
        $arr2 = array();
        foreach ($arr1 as $arrval)
            $arr2[] = $arrval;
    } else if (!is_array($arr1)) {
        $arr1 = array();
        foreach ($arr2 as $arrval)
            $arr1[] = $arrval;
    }
    if (count($arr1) != count($arr2))
        throw new TTException("Nr of members do not match.");
    $arr = array();
    foreach ($arr1 as $key => $val)
        $arr[$key] = $arr1[$key] - $arr2[$key];
    return $arr;
}
/**
 * Plus array
 *
 * @param array $arr1
 * @param array $arr2
 * @return array
 * @throws TT_Exception
 */
protected function arr_plus($arr1, $arr2) {
    if (!is_array($arr2)) {
        $div = $arr2;
        $arr2 = array();
        foreach ($arr1 as $arrval)
            $arr2[] = $div;
    } else if (!is_array($arr1)) {
        $div = $arr1;
        $arr1 = array();
        foreach ($arr2 as $arrval)
            $arr1[] = $div;
    }

    if (count($arr1) != count($arr2))
        throw new TT_Exception("Nr of members do not match.");

    $arr = array();
    foreach ($arr1 as $key => $val)
        $arr[$key] = $arr1[$key] + $arr2[$key];
    return $arr;
}

/**
 * Multiply array
 *
 * @param array $arr1
 * @param array $arr2
 * @return array
 * @throws TT_Exception
 */
protected function arr_multiply($arr1, $arr2) {
    if (!is_array($arr2)) {
        $div = $arr2;
        $arr2 = array();
        foreach ($arr1 as $arrval)
            $arr2[] = $div;
    } else if (!is_array($arr1)) {
        $div = $arr1;
        $arr1 = array();
        foreach ($arr2 as $arrval)
            $arr1[] = $div;
    }
}
if (count($arr1) != count($arr2))
    throw new TT_Exception("Nr of members do not match.");

$arr = array();
foreach ($arr1 as $key => $val)
    $arr[$key] = $arr1[$key] * $arr2[$key];

return $arr;
}

/**
 * Divide array
 *
 * @param array $arr1
 * @param array $arr2
 * @return array
 * @throws TT_Exception */
protected function arr_divide($arr1, $arr2) {
    if (!is_array($arr2)) {
        $div = $arr2;
        $arr2 = array();
        foreach ($arr1 as $arrval)
            $arr2[] = $div;
    } else if (!is_array($arr1)) {
        $div = $arr1;
        $arr1 = array();
        foreach ($arr2 as $arrval)
            $arr1[] = $div;
    }

    if (count($arr1) != count($arr2))
        throw new TT_Exception("Nr of members do not match.");

    $arr = array();
    foreach ($arr1 as $key => $val)
        $arr[$key] = $arr1[$key] / $arr2[$key];
    return $arr;
}

/**
 * Dot array
 *
 * @param array $arr1
 * @param array $arr2
 * @return array
 * @throws TT_Exception */
protected function arr_dot($arr1, $arr2) {
    if (count($arr1) != count($arr2))
return false;

$res = 0;
foreach ($arr1 as $key => $val)
    $res += $arr1[$key] * $arr2[$key];
return $res;
}

/**
 * Cross product array
 * @param array $arr1
 * @param array $arr2
 * @return array
 * @throws TT_Exception
 */
protected function arr_cross($arr1, $arr2) {
    if ((sc = count($arr1)) !== 3)
        throw new TT_Exception("Nr of members not exactly 3.");
    else if (count($arr2) != sc)
        throw new TT_Exception("Nr of members do not match.");
    $arr = array();
    return $arr;
}