Original Research Paper

Geolocation System Module Creation to Validate User Location Coordinates in an Android Application Using Flutter Framework

[1]Elmer Matthew Japara, [2]Samsul Arifin and [3]Edy Irwansyah

[1]*Department of Mathematics, School of Computer Science, Bina Nusantara University, Jakarta, Indonesia*
[2]*Department of Statistics, School of Computer Science, Bina Nusantara University, Jakarta, Indonesia*
[3]*Department of Computer Science, School of Computer Science, Bina Nusantara University, Jakarta, Indonesia*

Article history
Received: 07-05-2023
Revised: 16-06-2023
Accepted: 07-07-2023

Corresponding Author:
Samsul Arifin
Department of Statistics,
School of Computer Science,
Bina Nusantara University,
Jakarta, Indonesia
Email: samsul.arifin@binus.edu

Abstract: The target audience for this scientific paper's writing is anyone who is interested in the statistical information that will be used to create a method for determining the separation between two coordinate points. Given that as technology advances, a growing number of applications will make use of geolocation data for economic purposes, such as processing location data to maximize product sales. The development of satellite constellations, coordinate arrangements and even more sophisticated tasks like the construction of a space station require a high degree of accuracy. These are not just economic problems. However, no one is certain at this time of the precise calculation of the separation between the two coordinate points. The Haversine method and the Vincenty method are two of the several techniques used today to calculate coordinates. Each approach yields, naturally, different outcomes due to its basic differences. The variation in the results is anticipated to serve as a database for future study into the creation of new techniques to measure the distance between two coordinates at the point of the earth and to determine the size of the earth with a more precise value. However, if the application needs to calculate longer distances with higher accuracy, the Vincenty method may be more appropriate.

Keywords: Geolocation, Vincenty, Haversine, Flutter Android

Introduction

As the era progresses, almost every aspect of human life goes hand in hand and is also influenced by technology. This forces various parties, both from a company or individual, to implement technology in business practices or daily needs. As the global economy has entered the industry 4.0 stage, which means that traditional manufacturing and production practices are starting to switch to using smart technology, Machin to Machine (M2M) communication, and the Internet of Things (IoT) on a large-scale integrated for increased automation, increased self-monitoring and communication and production of intelligent machines that can analyze and diagnose problems without the need for human intervention (Dalenogare *et al*., 2018). In recent years, geolocation services (geolocation services) have become important in various fields and applications. Although end users may not be aware of this, many websites visited daily use geolocation information for local advertising, local content (such as local news and weather forecasts), and compliance with local laws. Mathematical concepts are needed to process geolocation data such as calculating distances between coordinates. There are several ways to calculate the distance between two coordinates, in which the author will take a method, namely the Vincenty Inverse, and will compare it with the numbers generated by the Haversine method, which is used by Google Maps (Great Circle Distance) (Hartanto *et al*., 2018; Setyorini *et al*., 2019). To process geolocation data, of course, a specially assigned module is needed. This module will be simulated through an application based on the Flutter Android framework (Tyagi, 2021; Arb and Al-Majdi, 2020).

Each method used to find the distance between two coordinates has its drawbacks or limitations. Vincenty Inverse will fail when calculating the distance between two coordinate points that are antipodal, whereas Haversine uses the assumption that the earth is round. Thaddeus Vincenty uses the assumption that the earth is an ellipsoid and claims that his method has an accuracy of up to 0.5 mm (Azdy and Darnis, 2020; Arifin *et al*., 2021). Until now, a more accurate method has not been found than these two methods. However, the required level of accuracy in calculating the distance between coordinates varies for the user. In its application in the realm of satellite system constellations, of course, the level of accuracy is very much considered to the millimeter. However, in local or close-range use, the level of accuracy is not too important. This research will discuss the differences and analyses between the two methods and their implementation in the creation of geolocation modules. The application is expected to have an accurate Geolocation system that can validate the location coordinates entered by the user. The need for accurate Geolocation data is essential in several applications such as ride-hailing, food delivery, e-commerce, etc. Flutter framework is used for mobile application development due to its flexibility, ease to learn, and efficient coding. Geolocation system module creation is crucial to provide accurate location data in several applications such as ride-hailing, food delivery, and e-commerce. The accuracy of the geolocation system is essential to validate the user location coordinates (Hernández-Lamas *et al*., 2021; Al Maki *et al*., 2023). However, it is challenging to create a geolocation system that can validate user location coordinates accurately. This problem becomes more challenging when the user is in a location with poor GPS signals, such as inside a building. Flutter framework is a popular framework used for mobile application development due to its flexibility, ease to learn, and efficient coding. Therefore, this research aims to develop a geolocation system module to validate user location coordinates in an Android application using the Flutter framework. Based on the background above, the main problem to be solved in this research is how to develop a geolocation system module to validate user location coordinates in an Android application using the Flutter framework. The specific issues to be addressed are how to design a geolocation system module that can validate the user location coordinates accurately, how to integrate the geolocation system module into the Android application, and how to test the accuracy of the geolocation system module (Winoto and Christanto, 2020; Zammetti, 2019).

The Haversine method, commonly used to calculate the distance between two points on the Earth's surface, has important limitations that need to be considered. This method is based on the assumption that the Earth is a perfect sphere, with a uniform shape over the entire surface. However, the Earth is not a perfect sphere, but an oblate spheroid. Therefore, when using the Haversine method to calculate distances between geographic points, the results may experience slight deviations or errors due to discrepancies with the actual shape of the Earth. In cases where greater precision is required or when the calculated distance involves significant elevation differences, an alternative method such as the Vincenty may be a more appropriate choice. Haversine's method is a fairly accurate estimate in most cases, but it does not take into account certain factors that may affect distance calculations. These include atmospheric refraction, which is the bending of light as it passes through Earth's atmosphere. This can cause errors in distance calculations, especially at very large distances, where the effect of atmospheric refraction becomes more significant. If the application or research requires very accurate distance calculations and involves very large distances, the Haversine method may not be the optimal choice. Other methods such as Vincenty or other methods that take into account factors such as atmospheric refraction can provide more accurate and reliable results. The Vincenty method is considered one of the most accurate methods for calculating the distance between two points on the Earth's surface, but it has certain limitations. These include difficulty reaching convergence and producing inaccurate results when two points are very close to each other, as well as difficulty when used to calculate the distance between points located close to the Earth's poles due to the changing shape of the Earth around the area. In these cases, alternative methods may need to be used to ensure more accurate and reliable results (Cagol and Colombi, 2016).

The Flutter framework for Android app development has a larger file size than native Android apps due to the presence of the Flutter engine and other components required to run the application. This can affect the user experience, especially for users with limited storage or slow internet connections. Therefore, careful consideration is needed when deciding to use Flutter as a framework for Android application development, especially when efficient file size is an important factor. Computer-based localization methods have significantly advanced the field of positioning and navigation. Two notable methods, "Enhancement of GPS Position Accuracy Using Machine Vision and Deep Learning Techniques" and "Improving GPS Position Accuracy by Identification of Reflected GPS Signals using Range Data for Modeling of Urban Structures," offer innovative solutions for improving GPS accuracy in different contexts. These methods have the potential to revolutionize the field of positioning and navigation by leveraging advanced technologies such as machine vision, deep learning, and urban modeling. Research and development in these areas will continue to drive advancements in the field and pave the way for more accurate and reliable localization technologies (Kumar *et al*., 2014). The purpose of this research is to develop a geolocation system module to validate user location coordinates in an Android application using the Flutter framework. The specific objectives of this research are (1) To design a geolocation system module that can validate the user location coordinates accurately, (2) To integrate the geolocation system module into the Android application and (3) To test the accuracy of the geolocation system module. The significance of this research is to provide a solution to the problem of inaccurate geolocation data in several applications such as ride-hailing, food delivery, and e-commerce. This research will contribute to the development of a geolocation system module that can validate the user location coordinates accurately in an Android application using the Flutter framework. This research will also contribute to the advancement of mobile application development using the Flutter framework. The results of this research are expected to be useful for researchers and developers in the field of mobile application development and geolocation system module creation (Shavitt and Zilberman, 2011; Jackson and Jackson, 2017).

Materials and Methods

The research stages that will be carried out in this study are using the waterfall model, with a sequential or sequential approach starting from analysis, design, coding, and testing as well as supporting stages. The following figure is a diagram of the research stages (Arifin and Muktyas, 2018). Figure 1 is about the research stages of the waterfall diagram.

First, we will talk about Stage I: Communication. In this first stage, there are two sequential parts, the first is problem identification and the second is literature study. Here, we will study problem identification and literature study (Pamunuwa *et al*., 2022). Problem Identification: This section is the first part of writing this research. In this section, the authors identify the problems encountered and start looking for solutions to these problems and identify some of the variables needed. Based on the formulation of the problem listed, the problem that will be discussed in this study is how the process of making a geolocation module is to be used in an application as well as the application and comparison or analysis of the results of the Haversine and Vincenty method algorithms. The process of making this module covers the analysis of differences or discrepancies in the results of each of the computational algorithms of the Haversine and Vincenty methods in general and under certain conditions. Further, the use of the two mathematical concepts as mentioned in the previous paragraph is due to the difference in complexity in the Haversine method which is not so complex compared to the Vincenty method, which is complex but, according to literature studies, has a higher level of accuracy than the Haversine method. The use of these two methods in the manufacture of this system module aims to determine the difference in results, both processing speed and nominal differences in certain parameters embedded in the two methods. Of course, in making geolocation modules, especially for measuring distances between coordinates, the level of accuracy of an algorithm is very necessary. Therefore, knowing the difference in results between the two methods as well as the correct implementation of the algorithms of the two methods in a special code script for the Flutter framework is something important.

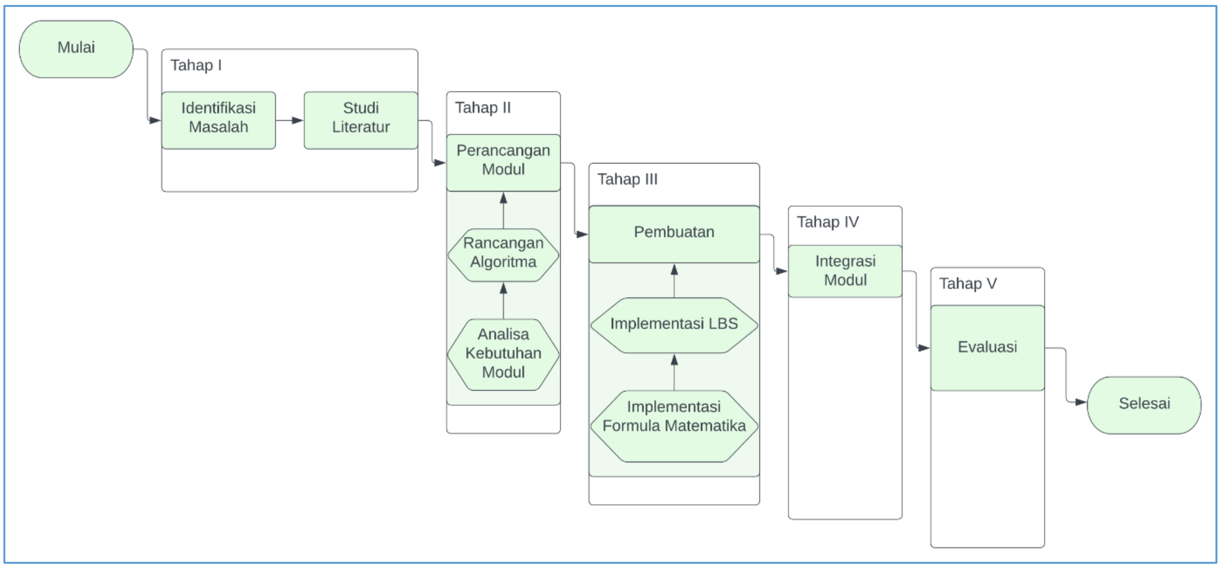


Fig. 1: Research stages waterfall diagram

Literature Study: After the problems on this research topic have been identified, the writer continues the research by starting a literature study. A literature study is needed to deepen the understanding of related theories. Theories related to this research such as trigonometry, Haversine formula, and Vincenty's formula are very important theories to understand more deeply because they are complex theories. These three theories have relationships with each other such as Haversine which is a trigonometric function that is rarely used in the modern era and correlates with other trigonometric functions (such as sine and cosine) and Vincenty's Formula which requires a very deep understanding of trigonometry. In addition to studying the theories mentioned in the previous paragraphs, the author also studies the theories and terms regarding navigation, geolocation, coordinates, or the system in question. Literature study regarding these subtopics has proven to be important because the author knows that there are certain standards for geolocation systems such as the WGS-84 standard which is commonly used by various countries in the world. The author also delves deeper into matters related to the programming side in writing this research such as the Flutter framework, the Dart programming language, asynchronous programming, and Android as well as the rules needed in developing a neat system or application (Kumar *et al*., 2014; Gade, 2010).

Next, in the second part, we will talk about. Stage II: Planning. At this stage, the authors analyze and design system requirements to meet the required criteria. Some of the criteria that need to be met are making the module work quickly so that it can process data in real-time, accessing location data, implementing the Haversine and Vincenty methods, the module operating automatically without manual input from the user, and the variables according to the Haversine and Vincenty methods. Regarding the integration side of the application, the author will design and create a programming algorithm for the module so that it can meet the desired criteria. Here is the flowchart for this module. The following Fig. 2 is about the flowchart diagram for the module we made.

Moreover, in the third part, we will talk about Stage III: *Modelling.* At this stage, the author will implement the logic or algorithms from both the Haversine and Vincenty methods in the form of programming code in the Dart programming language and adapt it to the Flutter framework. Here, we will study the implementation of mathematical formulas and the implementation of LBS (Location-Based Service). Implementation of Mathematical Formulas: In this section, the implementation of the mathematical formulas from the Haversine and Vincenty methods will be explained so that the module can calculate the distance between coordinates. The use of these two methods is because these methods have fundamentals that are suitable for use in calculating coordinates on a spherical plane. As explained on the theoretical basis, Haversine and Vincenty each have different fundamentals in their application to calculating distances between earth coordinates, namely, Haversine assumes that the earth is round, while Vincenty assumes that the earth is slightly ellipsoid. Therefore, there are significant formula differences based on what is learned from the theoretical basis and because of these differences in formulas, the way to implement them is also different. Of course, because this module uses two methods of calculating distances between coordinates, this module will have two classes that have templates in their abstract class with variables that have been initialized according to the WGS84 standard (Nabie *et al*., 2018). Figure 3 is the code of the WGS84 standard.

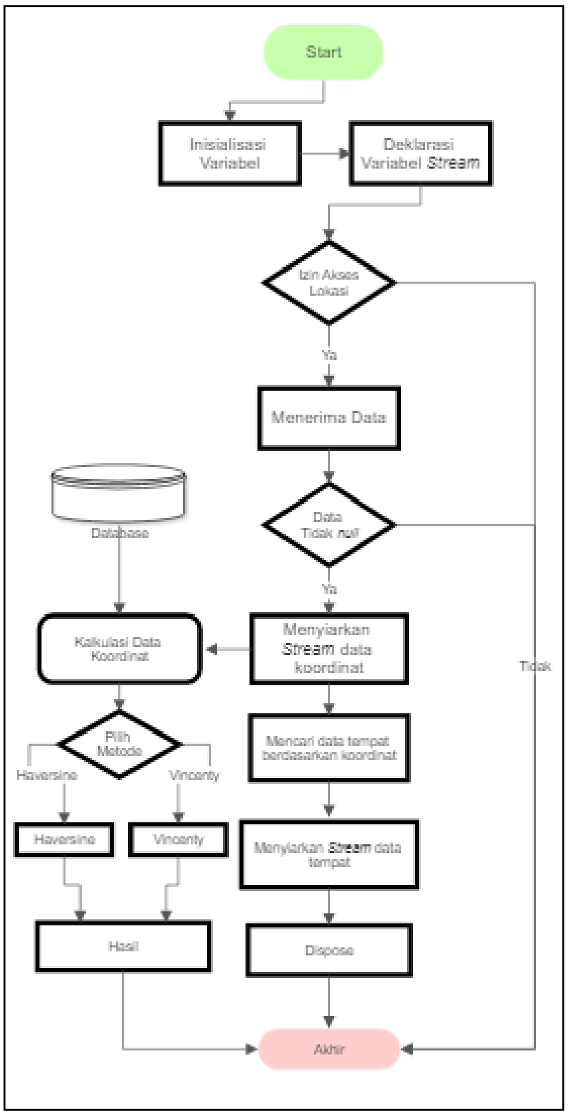


Fig. 2: Flowchart diagram for the module

Haversine Method: The Haversine method in this module is used to calculate distances between coordinates with the assumption that the earth is spherical and uses a trigonometry function that is rarely used now namely the Haversine function. It is known that the Haversine formula on coordinates is as follows:

$$hav(\theta) = hav(\varphi_2 - \varphi_1) + cos(\varphi_1)\cos(\varphi_2)hav(\lambda_2 - \lambda_1)$$

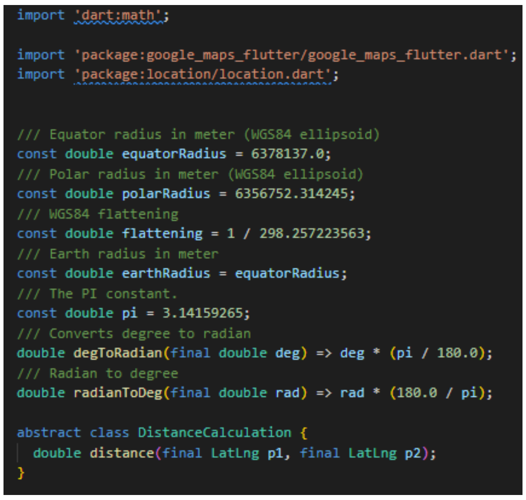
$$hav(\theta) = \frac{1}{2}versin(\theta)$$

$$ver\,sin(\theta) = 1 - cos(\theta) = 2sin^2\left(\frac{\theta}{2}\right)$$

$$hav(\theta) = sin^2\left(\frac{\theta}{2}\right)$$

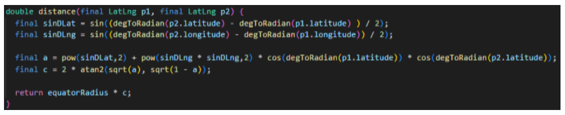
By changing the haversine function to a sine form, the Dart programming language can calculate this method. Here is the code in the Fig. 4.

Vincenty's Method: The implementation of the Vincenty method into a programming algorithm is complicated because of the complexity of the logic or algorithm in the method. The Vincenty method has certain numbers that have been given by its inventor. The numbers in question are; 16, 47, 74, 128, 175, 256, 320, 768, 1024, 4096, and 16384 are constant numbers. In Vincenty's Formula, there are two methods, namely the direct method and the inverse method. This module will use the inverse method from Vincenty to find the distance between two coordinate points. The direct method from Vincenty will not be used because it is not necessary to find the azimuth of two coordinate points (Mahmoud and Akkari, 2016; Pressman, 2005). The Vincenty method has its own mathematical notations, therefore the notations to be declared in the code are as in Fig. 5:



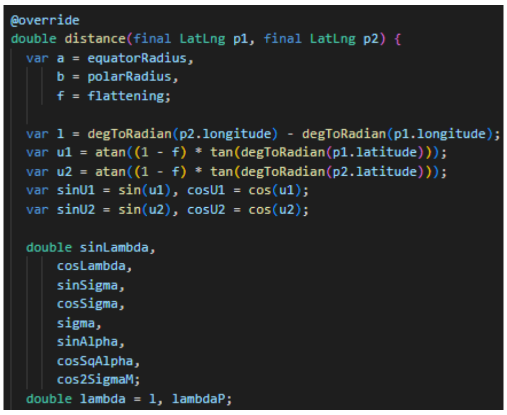
```
import 'dart:math';

import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:location/location.dart';


/// Equator radius in meter (WGS84 ellipsoid)
const double equatorRadius = 6378137.0;
/// Polar radius in meter (WGS84 ellipsoid)
const double polarRadius = 6356752.314245;
/// WGS84 flattening
const double flattening = 1 / 298.257223563;
/// Earth radius in meter
const double earthRadius = equatorRadius;
/// The PI constant.
const double pi = 3.14159265;
/// Converts degree to radian
double degToRadian(final double deg) => deg * (pi / 180.0);
/// Radian to degree
double radianToDeg(final double rad) => rad * (180.0 / pi);

abstract class DistanceCalculation {
  double distance(final LatLng p1, final LatLng p2);
}
```

Fig. 3: Code of WGS84 standard



```
double distance(final LatLng p1, final LatLng p2) {
  final sinDLat = sin((degToRadian(p2.latitude) - degToRadian(p1.latitude) ) / 2);
  final sinDLng = sin((degToRadian(p2.longitude) - degToRadian(p1.longitude)) / 2);

  final a = pow(sinDLat,2) + pow(sinDLng * sinDLng,2) * cos(degToRadian(p1.latitude)) * cos(degToRadian(p2.latitude));
  final c = 2 * atan2(sqrt(a), sqrt(1 - a));

  return equatorRadius * c;
}
```

Fig. 4: Code of distance function



```
@override
double distance(final LatLng p1, final LatLng p2) {
  var a = equatorRadius,
      b = polarRadius,
      f = flattening;

  var l = degToRadian(p2.longitude) - degToRadian(p1.longitude);
  var u1 = atan((1 - f) * tan(degToRadian(p1.latitude)));
  var u2 = atan((1 - f) * tan(degToRadian(p2.latitude)));
  var sinU1 = sin(u1), cosU1 = cos(u1);
  var sinU2 = sin(u2), cosU2 = cos(u2);

  double sinLambda,
      cosLambda,
      sinSigma,
      cosSigma,
      sigma,
      sinAlpha,
      cosSqAlpha,
      cos2SigmaM;
  double lambda = l, lambdaP;
```

Fig. 5: Code of distance function

At the beginning of the operation of the Vincenty formula, the value of λ must be found by finding the value of the sine and cosine of the delta (Nystedt, 2017; Arifin and Garminia, 2019). After the equations have been entered into the algorithm, then do as many repetitions as desired with conditions of accuracy level of 10^(-12) or 0.6 mm (Utami and Mampouw, 2020). Here's the code.

At the beginning of the operation of the Vincenty formula, the value of λ must be found by finding the value of sine and cosine of delta first using the following equation:

$$\sin\sigma = \sqrt{\left(\cos U_2 \sin\lambda\right)^2 + \left(\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos\lambda\right)}$$

$$\cos\sigma = \sin U_1 \sin U_2 + \cos U_2 \cos U_1 \cos\lambda$$

$$\sigma = arctan\,2(\sin\sigma, \cos\sigma)$$

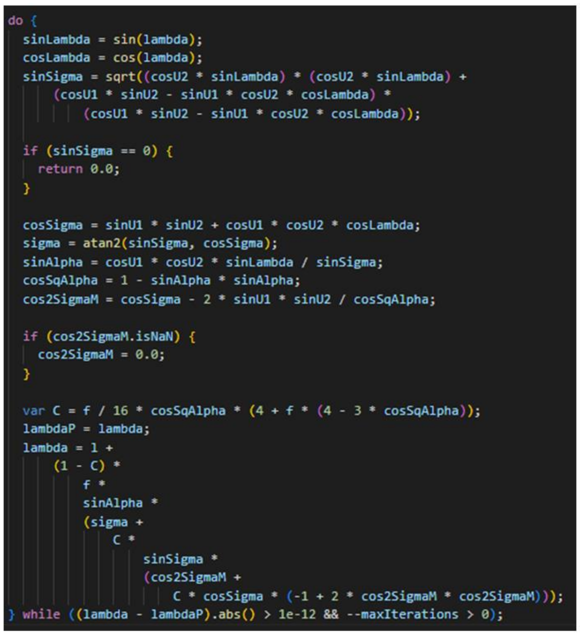
$$\sin\alpha = \frac{\cos U_1 \cos U_2 \sin\lambda}{\sin\sigma}$$

$$\cos\left(2\sigma_m\right) = \cos\sigma - \frac{2\sin U_1 \sin U_2}{\cos^2\sigma}$$

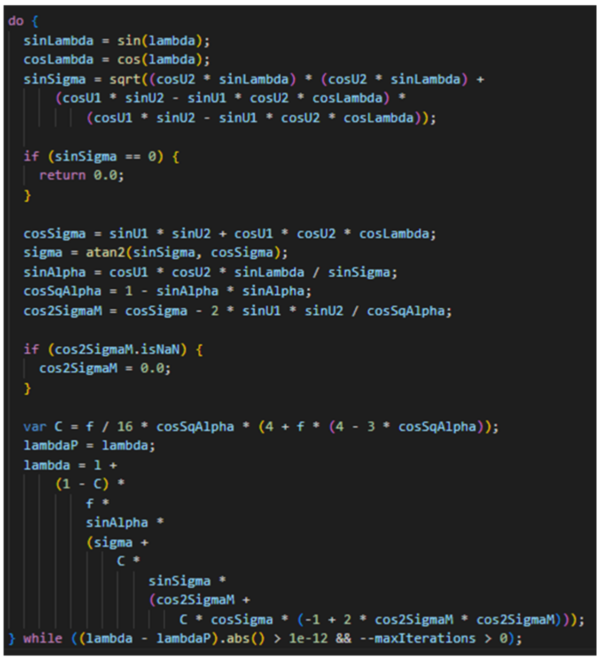
$$C = \frac{f}{16}\cos^2\alpha\left[4 + f(4 - 3\cos^2\alpha)\right]$$

$$\lambda = L + (1 + C) f \sin\alpha \left\{\alpha + C\sin\sigma\left[\cos(2\sigma_m) + C\cos\sigma(-1 + 2\cos^2(2\sigma_m))\right]\right\}$$

After the equations have been entered into the algorithm, then do as many repetitions as desired with conditions of accuracy level of 10^(-12) or 0.6 mm (Utami and Mampouw, 2020). Here's the code in Fig. 6. Furthermore, after finding the lambda value, you can only operate the following equation to find the distance between the two coordinates (Maor, 2019). Fig. 7 is about the code of distance of two coordinates.



```
do {
  sinLambda = sin(lambda);
  cosLambda = cos(lambda);
  sinSigma = sqrt((cosU2 * sinLambda) * (cosU2 * sinLambda) +
      (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda) *
          (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda));

  if (sinSigma == 0) {
    return 0.0;
  }

  cosSigma = sinU1 * sinU2 + cosU1 * cosU2 * cosLambda;
  sigma = atan2(sinSigma, cosSigma);
  sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma;
  cosSqAlpha = 1 - sinAlpha * sinAlpha;
  cos2SigmaM = cosSigma - 2 * sinU1 * sinU2 / cosSqAlpha;

  if (cos2SigmaM.isNaN) {
    cos2SigmaM = 0.0;
  }

  var C = f / 16 * cosSqAlpha * (4 + f * (4 - 3 * cosSqAlpha));
  lambdaP = lambda;
  lambda = l +
      (1 - C) *
          f *
          sinAlpha *
          (sigma +
              C *
                  sinSigma *
                  (cos2SigmaM +
                      C * cosSigma * (-1 + 2 * cos2SigmaM * cos2SigmaM)));
} while ((lambda - lambdaP).abs() > 1e-12 && --maxIterations > 0);
```

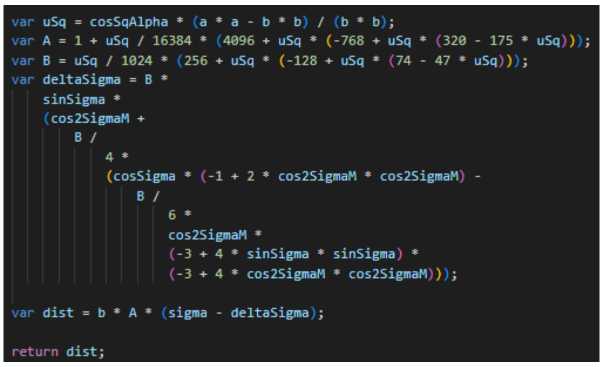


```
do {
  sinLambda = sin(lambda);
  cosLambda = cos(lambda);
  sinSigma = sqrt((cosU2 * sinLambda) * (cosU2 * sinLambda) +
      (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda) *
          (cosU1 * sinU2 - sinU1 * cosU2 * cosLambda));

  if (sinSigma == 0) {
    return 0.0;
  }

  cosSigma = sinU1 * sinU2 + cosU1 * cosU2 * cosLambda;
  sigma = atan2(sinSigma, cosSigma);
  sinAlpha = cosU1 * cosU2 * sinLambda / sinSigma;
  cosSqAlpha = 1 - sinAlpha * sinAlpha;
  cos2SigmaM = cosSigma - 2 * sinU1 * sinU2 / cosSqAlpha;

  if (cos2SigmaM.isNaN) {
    cos2SigmaM = 0.0;
  }

  var C = f / 16 * cosSqAlpha * (4 + f * (4 - 3 * cosSqAlpha));
  lambdaP = lambda;
  lambda = l +
      (1 - C) *
          f *
          sinAlpha *
          (sigma +
              C *
                  sinSigma *
                  (cos2SigmaM +
                      C * cosSigma * (-1 + 2 * cos2SigmaM * cos2SigmaM)));
} while ((lambda - lambdaP).abs() > 1e-12 && --maxIterations > 0);
```

Fig. 6: Code of sinLambda, cosLambda, and sinSigma

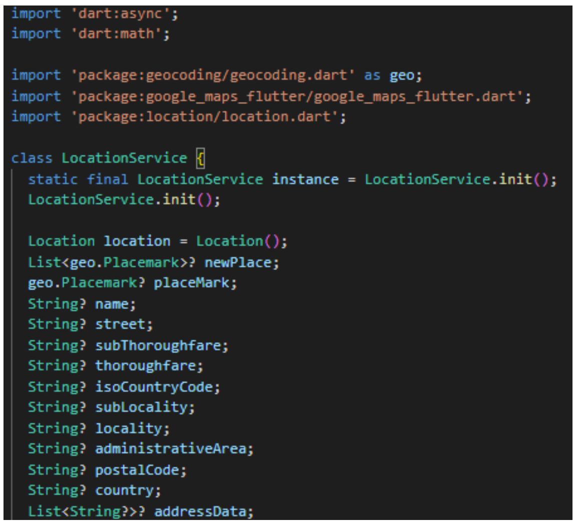
With dist having results in the distance between the two coordinates. Keep in mind that when the two coordinates are in the antipodal position, which means on the opposite side of the earth (180` difference in longitude and 0` difference in both latitudes from point 0`), the Vincenty method will fail to calculate (Maor, 2019). Implementation of LBS (Location-Based Service): In this section, the process that the author will carry out is to design an algorithm to use LBS in the geolocation module. There are two algorithms that will be used in the geolocation module, namely the algorithm for the Haversine method and for the Vincenty method. Each of these algorithms is required to meet the module criteria, namely requiring the algorithm to process dynamically as quickly as possible so that it can process real-time input data or change input data. Initialize the variables related to the module: This initial step is the process of initializing any variables that will be needed by the module to function as intended. This module has a dependency on the Flutter library (Hartanto *et al*., 2018; Jackson and Jackson, 2017). We can see the module function in Fig. 8.

Validate user device location access permission: The next step in LBS implementation, it is very important for applications to be able to access the user's device location. The algorithm needed at this stage is to validate the permission to access the device location. Figure 9 is about the code of LBS implementation.



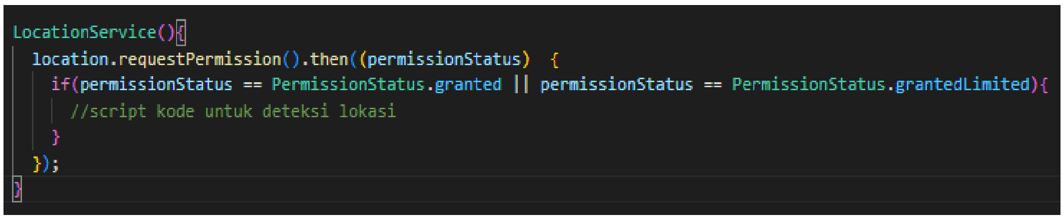
```
var uSq = cosSqAlpha * (a * a - b * b) / (b * b);
var A = 1 + uSq / 16384 * (4096 + uSq * (-768 + uSq * (320 - 175 * uSq)));
var B = uSq / 1024 * (256 + uSq * (-128 + uSq * (74 - 47 * uSq)));
var deltaSigma = B *
    sinSigma *
    (cos2SigmaM +
        B /
            4 *
            (cosSigma * (-1 + 2 * cos2SigmaM * cos2SigmaM) -
                B /
                    6 *
                    cos2SigmaM *
                    (-3 + 4 * sinSigma * sinSigma) *
                    (-3 + 4 * cos2SigmaM * cos2SigmaM)));

var dist = b * A * (sigma - deltaSigma);

return dist;
```

Fig. 7: Code of distance of two coordinates



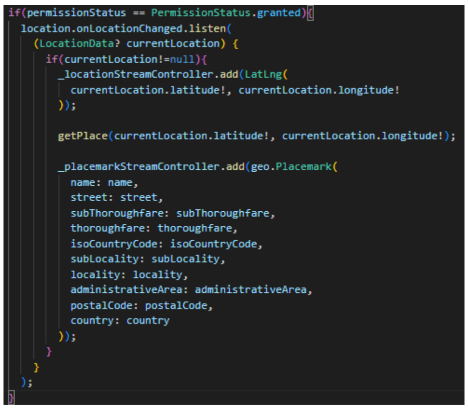
```
import 'dart:async';
import 'dart:math';

import 'package:geocoding/geocoding.dart' as geo;
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:location/location.dart';

class LocationService {
  static final LocationService instance = LocationService.init();
  LocationService.init();

  Location location = Location();
  List<geo.Placemark>? newPlace;
  geo.Placemark? placeMark;
  String? name;
  String? street;
  String? subThoroughfare;
  String? thoroughfare;
  String? isoCountryCode;
  String? subLocality;
  String? locality;
  String? administrativeArea;
  String? postalCode;
  String? country;
  List<String?>? addressData;
```

Fig. 8: The module to function



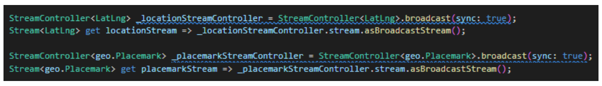
```
LocationService(){
  location.requestPermission().then((permissionStatus)  {
    if(permissionStatus == PermissionStatus.granted || permissionStatus == PermissionStatus.grantedLimited){
      //script kode untuk deteksi lokasi
    }
  });
}
```

Fig. 9: Code of LBS implementation



```
if(permissionStatus == PermissionStatus.granted){
  location.onLocationChanged.listen(
    (LocationData? currentLocation) {
      if(currentLocation!=null){
        _locationStreamController.add(LatLng(
          currentLocation.latitude!, currentLocation.longitude!
        ));

        getPlace(currentLocation.latitude!, currentLocation.longitude!);

        _placemarkStreamController.add(geo.Placemark(
          name: name,
          street: street,
          subThoroughfare: subThoroughfare,
          thoroughfare: thoroughfare,
          isoCountryCode: isoCountryCode,
          subLocality: subLocality,
          locality: locality,
          administrativeArea: administrativeArea,
          postalCode: postalCode,
          country: country
        ));
      }
    }
  );
}
```

Fig. 10: Code of detecting device location data



```
StreamController<LatLng> _locationStreamController = StreamController<LatLng>.broadcast(sync: true);
Stream<LatLng> get locationStream => _locationStreamController.stream.asBroadcastStream();

StreamController<geo.Placemark> _placemarkStreamController = StreamController<geo.Placemark>.broadcast(sync: true);
Stream<geo.Placemark> get placemarkStream => _placemarkStreamController.stream.asBroadcastStream();
```

Fig. 11: Code of display data in real-time

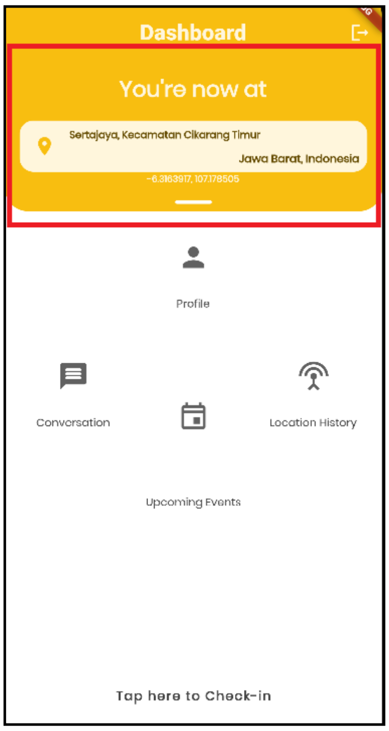
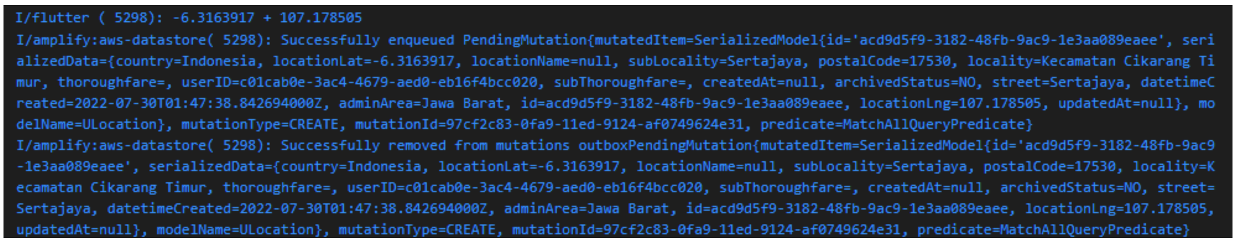


Fig. 12: Display of dashboard apps

Detect device location data: The algorithm will take input data from the user which is a coordinate. Data capture requires device location data access permission. The image below shows the validation of device location data access permissions and data capture. Figure 10 is about the code to detect device location data. Display data in real-time: To display data in real-time, it is necessary to register several variables to broadcast data changes to the stream. The variables whose values you want to display in real-time or create a stream are as in Fig. 11.

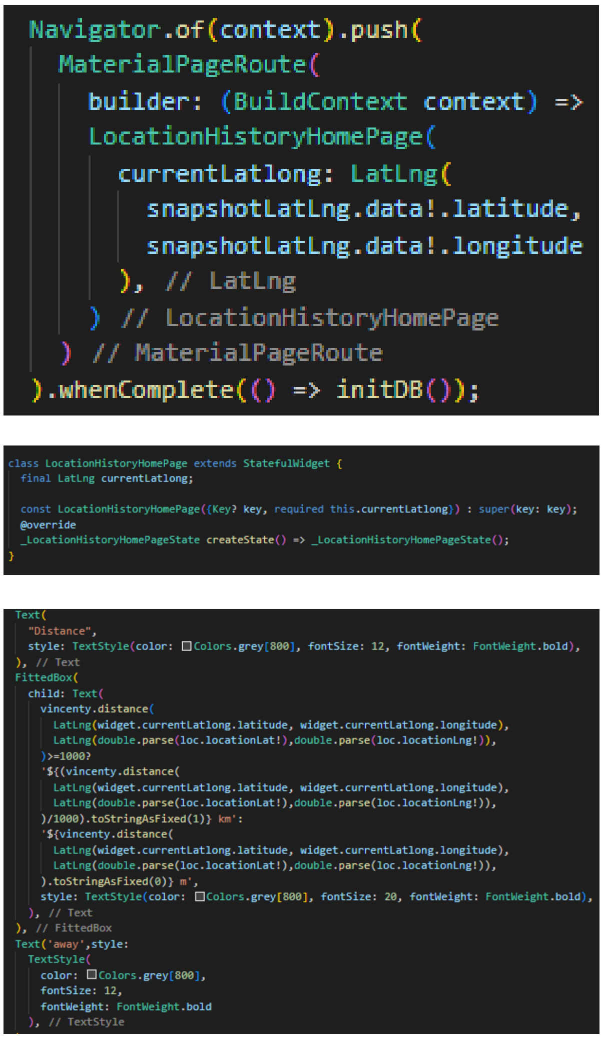
Further, in the fourth part, we will talk about Stage IV: Construction. At this stage, the author will carry out the module integration process with the application aiming to test whether the module meets the criteria and whether the functionality of the module runs fully. To integrate the module into the application, there will be changes in the application code script, especially changes to the class on the application screen. There are two types of State in the main class used in the screen in Flutter, namely the Stateful Widget class and the Stateless Widget class. A stateful Widget indicates that the screen can display dynamic data, Widget shapes change (transition), and animation. Meanwhile, the stateless widget indicates that the screen will not display dynamic data, only static. Therefore, applications are required to use Stateful Widgets on certain screens that this module will integrate. Apart from that, Flutter also prepares builders in the form of Widgets for data types and Stream classes which must be used when you want to use the real-time data processing functions of this module (Arb and Al-Majdi, 2020). Figure 12, we can see the display of Dashboard apps.

The value contained in the red box in the image above is the value obtained from the module that was created. These values are dynamic and are obtained from Stream data. The data changes in real-time for both place-name data and coordinates (below the place-name box). The author also tests whether the displayed data can be stored in the application database by pressing the "Tap here to Check-in" button at the bottom of the screen. The app outputs information like the following in the Debug Console box in the IDE (Restrepo-Calle *et al*., 2020). Figure 13 is about the value obtained from the module.



```
I/flutter ( 5298): -6.3163917 + 107.178505
I/amplify:aws-datastore( 5298): Successfully enqueued PendingMutation{mutatedItem=SerializedModel{id='acd9d5f9-3182-48fb-9ac9-1e3aa089eaee', seri
alizedData={country=Indonesia, locationLat=-6.3163917, locationName=null, subLocality=Sertajaya, postalCode=17530, locality=Kecamatan Cikarang Ti
mur, thoroughfare=, userID=c01cab0e-3ac4-4679-aed0-eb16f4bcc020, subThoroughfare=, createdAt=null, archivedStatus=NO, street=Sertajaya, datetimeC
reated=2022-07-30T01:47:38.842694000Z, adminArea=Jawa Barat, id=acd9d5f9-3182-48fb-9ac9-1e3aa089eaee, locationLng=107.178505, updatedAt=null}, mo
delName=ULocation}, mutationType=CREATE, mutationId=97cf2c83-0fa9-11ed-9124-af0749624e31, predicate=MatchAllQueryPredicate}
I/amplify:aws-datastore( 5298): Successfully removed from mutations outboxPendingMutation{mutatedItem=SerializedModel{id='acd9d5f9-3182-48fb-9ac9
-1e3aa089eaee', serializedData={country=Indonesia, locationLat=-6.3163917, locationName=null, subLocality=Sertajaya, postalCode=17530, locality=K
ecamatan Cikarang Timur, thoroughfare=, userID=c01cab0e-3ac4-4679-aed0-eb16f4bcc020, subThoroughfare=, createdAt=null, archivedStatus=NO, street=
Sertajaya, datetimeCreated=2022-07-30T01:47:38.842694000Z, adminArea=Jawa Barat, id=acd9d5f9-3182-48fb-9ac9-1e3aa089eaee, locationLng=107.178505,
updatedAt=null}, modelName=ULocation}, mutationType=CREATE, mutationId=97cf2c83-0fa9-11ed-9124-af0749624e31, predicate=MatchAllQueryPredicate}
```

Fig. 13: The value obtained from the module



```
Navigator.of(context).push(
  MaterialPageRoute(
    builder: (BuildContext context) =>
    LocationHistoryHomePage(
      currentLatlong: LatLng(
        snapshotLatLng.data!.latitude,
        snapshotLatLng.data!.longitude
      ), // LatLng
    ) // LocationHistoryHomePage
  ) // MaterialPageRoute
).whenComplete(() => initDB());
```

```
class LocationHistoryHomePage extends StatefulWidget {
  final LatLng currentLatlong;

  const LocationHistoryHomePage({Key? key, required this.currentLatlong}) : super(key: key);
  @override
  _LocationHistoryHomePageState createState() => _LocationHistoryHomePageState();
}
```

```
Text(
  "Distance",
  style: TextStyle(color: ■Colors.grey[800], fontSize: 12, fontWeight: FontWeight.bold),
), // Text
FittedBox(
  child: Text(
    vincenty.distance(
      LatLng(widget.currentLatlong.latitude, widget.currentLatlong.longitude),
      LatLng(double.parse(loc.locationLat!),double.parse(loc.locationLng!)),
    )>=1000?
    '${(vincenty.distance(
      LatLng(widget.currentLatlong.latitude, widget.currentLatlong.longitude),
      LatLng(double.parse(loc.locationLat!),double.parse(loc.locationLng!)),
    )/1000).toStringAsFixed(1)} km':
    '${vincenty.distance(
      LatLng(widget.currentLatlong.latitude, widget.currentLatlong.longitude),
      LatLng(double.parse(loc.locationLat!),double.parse(loc.locationLng!)),
    ).toStringAsFixed(0)} m',
    style: TextStyle(color: ■Colors.grey[800], fontSize: 20, fontWeight: FontWeight.bold),
  ), // Text
), // FittedBox
Text('away',style:
  TextStyle(
    color: ■Colors.grey[800],
    fontSize: 12,
    fontWeight: FontWeight.bold
  ), // TextStyle
```

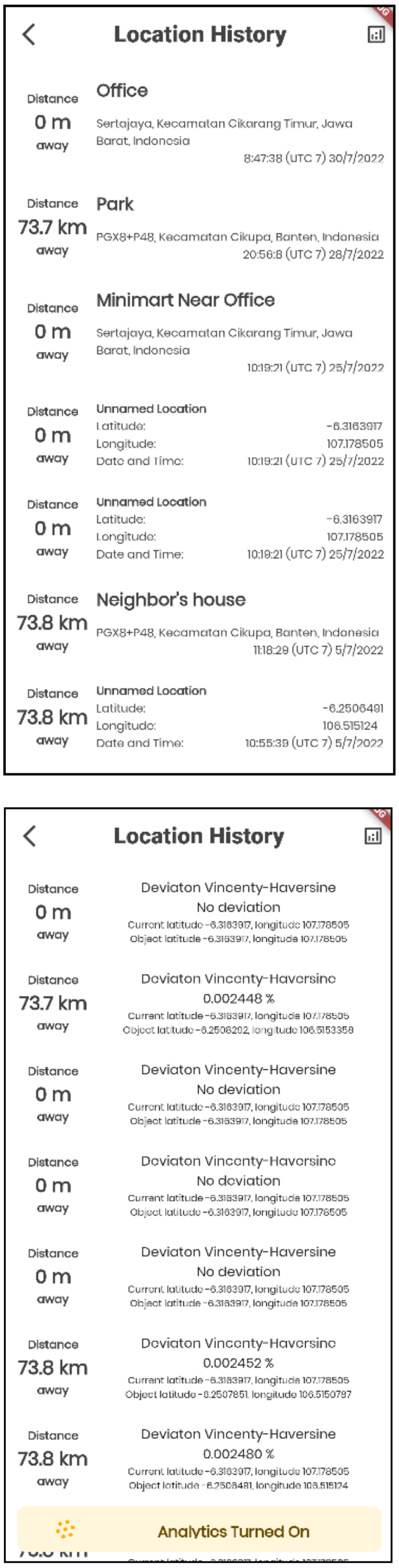
Fig. 14: Code of distance between the current user's coordinates

Seen in the first line in the image above are the coordinates detected by the application and obtained from the module. After detecting the coordinates, the application will query to store data in the database as shown in the next line. The words "Successfully removed from outboxPendingMutation mutations" mean that the back-end process of the application has successfully entered data into the database because the outbox Pending Mutation mutations are no longer needed. In the second line I/amplify: Aws-datastore (5298), there are the coordinates as well as the location. The events in the above figures indicate a successful integration of the module for accessing real-time location data. Furthermore, the author will test if other functions of the application function successful, such as testing the process of calculating the distance between the two coordinates (Wu, 2019). After module integration in the main menu, it will be continued with integration in other sections using the same method. In the location history section (Location History), there will be a calculation of the distance between the current user's coordinates to the available item coordinates using the following code script.

Figure 14 shows the use of the Vincenty method in finding the distance between two coordinates. User coordinates are represented by the widget. currentLatLong where the data from the variable is the data passed from the previous class, namely the main menu. Figures 14-16 show data passed from the previous class (main menu) in Fig. 16 to the next class in Fig. 15. This passed data is static data obtained from the last data in a stream on the main menu. The use of static data is due to the complexity of the Vincenty algorithm (Vincenty, 1975), which requires a long processing time with many different data as shown below.

Because the author uses the same emulator and location preset, there are items that are 0 m away to validate whether the module has errors in the form of bugs. The author also created a new button on the AppBar to display the analytical mode view to see the deviation between the two methods. The image above shows a little analysis of the data for each item. If the distance between the device and the item is 0 m, the application will display "No Deviation" (Azdy and Darnis, 2020). To find the deviation in percent of the two methods, a function has been prepared in this module.

The above functions are contained in the LocationService class, a class that contains LBS (Location Based Service) access functionality as well as functions that require using both methods such as finding the average between the results of the two methods and the deviation in percent. The module developed in the Flutter framework is an extension package that must be listed in the pubspec.yaml file in the application's project folder (Hartanto *et al*., 2018). Figure 17 is a snippet of the pubspec. yaml file so that the module can be used in the application.



Location History

Distance 0 m away
Office
Sertajaya, Kecamatan Cikarang Timur, Jawa Barat, Indonesia
8:47:38 (UTC 7) 30/7/2022

Distance 73.7 km away
Park
PGX8+P48, Kecamatan Cikupa, Banten, Indonesia
20:56:8 (UTC 7) 28/7/2022

Distance 0 m away
Minimart Near Office
Sertajaya, Kecamatan Cikarang Timur, Jawa Barat, Indonesia
10:19:21 (UTC 7) 25/7/2022

Distance 0 m away
Unnamed Location
Latitude: -6.3163917
Longitude: 107.178505
Date and Time: 10:19:21 (UTC 7) 25/7/2022

Distance 0 m away
Unnamed Location
Latitude: -6.3163917
Longitude: 107.178505
Date and Time: 10:19:21 (UTC 7) 25/7/2022

Distance 73.8 km away
Neighbor's house
PGX8+P48, Kecamatan Cikupa, Banten, Indonesia
11:18:29 (UTC 7) 5/7/2022

Distance 73.8 km away
Unnamed Location
Latitude: -6.2506491
Longitude: 106.515124
Date and Time: 10:55:39 (UTC 7) 5/7/2022

Location History

Distance 0 m away
Deviaton Vincenty-Haversine
No deviation
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.3163917, longitude 107.178505

Distance 73.7 km away
Deviaton Vincenty-Haversine
0.002448 %
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.2508292, longitude 106.5153358

Distance 0 m away
Deviaton Vincenty-Haversine
No deviation
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.3163917, longitude 107.178505

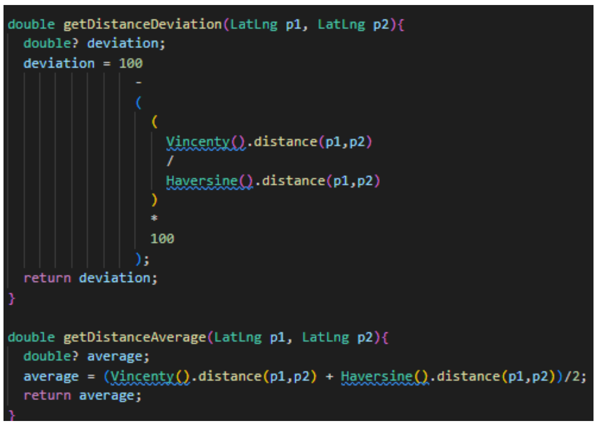
Distance 0 m away
Deviaton Vincenty-Haversine
No deviation
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.3163917, longitude 107.178505

Distance 0 m away
Deviaton Vincenty-Haversine
No deviation
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.3163917, longitude 107.178505

Distance 73.8 km away
Deviaton Vincenty-Haversine
0.002452 %
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.2507851, longitude 106.5150787

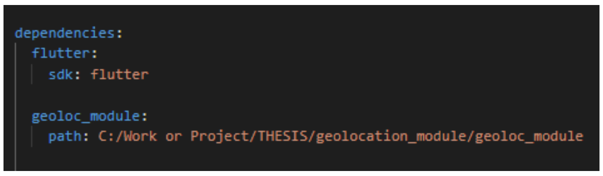
Distance 73.8 km away
Deviaton Vincenty-Haversine
0.002480 %
Current latitude -6.3163917, longitude 107.178505
Object latitude -6.2506481, longitude 106.515124

Fig. 15: Display of location history



```
double getDistanceDeviation(LatLng p1, LatLng p2){
  double? deviation;
  deviation = 100
              -
              (
                (
                  Vincenty().distance(p1,p2)
                  /
                  Haversine().distance(p1,p2)
                )
                *
                100
              );
  return deviation;
}

double getDistanceAverage(LatLng p1, LatLng p2){
  double? average;
  average = (Vincenty().distance(p1,p2) + Haversine().distance(p1,p2))/2;
  return average;
}
```

Fig. 16: Code of deviation in percent



```
dependencies:
  flutter:
    sdk: flutter

  geoloc_module:
    path: C:/Work or Project/THESIS/geolocation_module/geoloc_module
```

Fig. 17: Code of dependencies

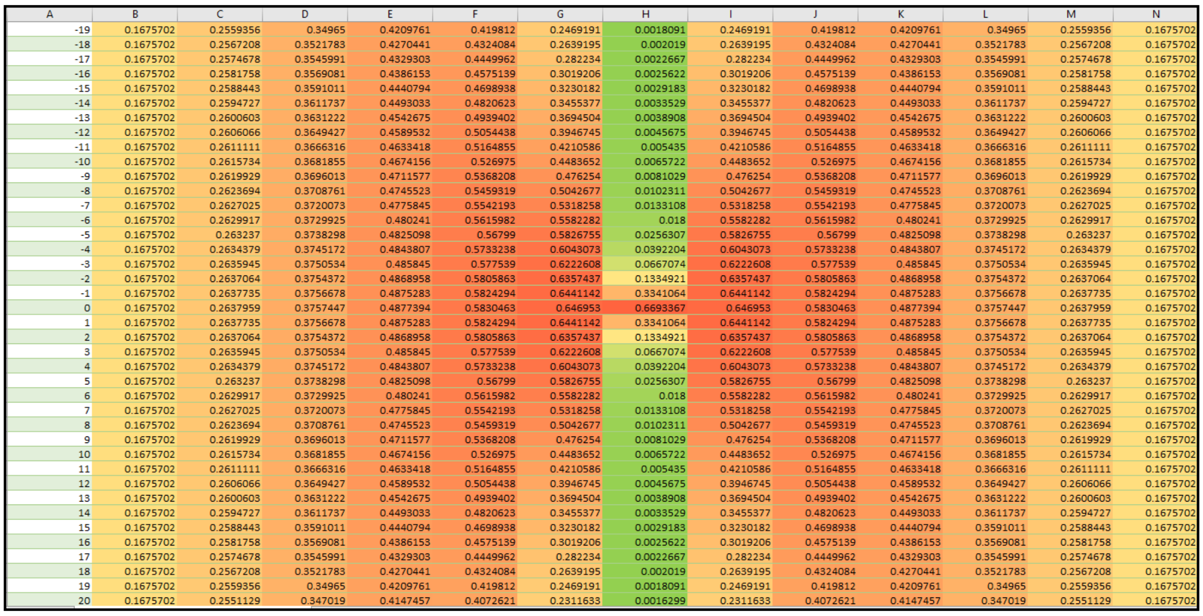
Finally, in the fifth part, we will examine Tahap V: Deployment. This stage is the final stage which includes the launch of a project or module in the real world. At this stage, the author will launch a module that has been integrated into the GeoSoc application and test and evaluate the deficiencies contained in the module so that it meets the specified standards, namely, specifically for screen design (UI), the Eight Golden Rules standard and evaluates UX. using methods such as the User Experience Questionnaire (Kusuma *et al*., 2023; Arifin *et al*., 2022).

Results and Discussion

The article discusses the creation of a geolocation system module to validate user location coordinates in an Android application using the Flutter framework. Despite using advanced algorithms and techniques, there were instances where the obtained coordinates deviated from the actual location. Additionally, there were challenges related to privacy and security concerns, such as data encryption, secure transmission, and storage of user location coordinates. Additionally, computational challenges emerged during the processing of large volumes of location data, such as the efficiency and scalability of the algorithms used in the module. These challenges significantly influenced the development and implementation of the module and it is important to discuss them to provide a comprehensive view of the study's limitations and potential areas for improvement. The challenge of creating a geolocation system module was to ensure that it could handle a large number of user requests without compromising performance or introducing delays in the validation process. Optimization techniques, parallel processing, and memory management strategies were explored to address these challenges. Additionally, practical limitations such as connectivity issues, signal interference from buildings or natural obstacles, and limitations imposed by the hardware or software components of Android devices had to be considered. Additionally, geographical challenges such as variations in satellite coverage, signal availability, or environmental factors in different regions affected the accuracy and reliability of the module. These discussions contribute to the advancement of geolocation technology, guide future research efforts, and aid in the development of more reliable and robust geolocation system modules in Android applications.

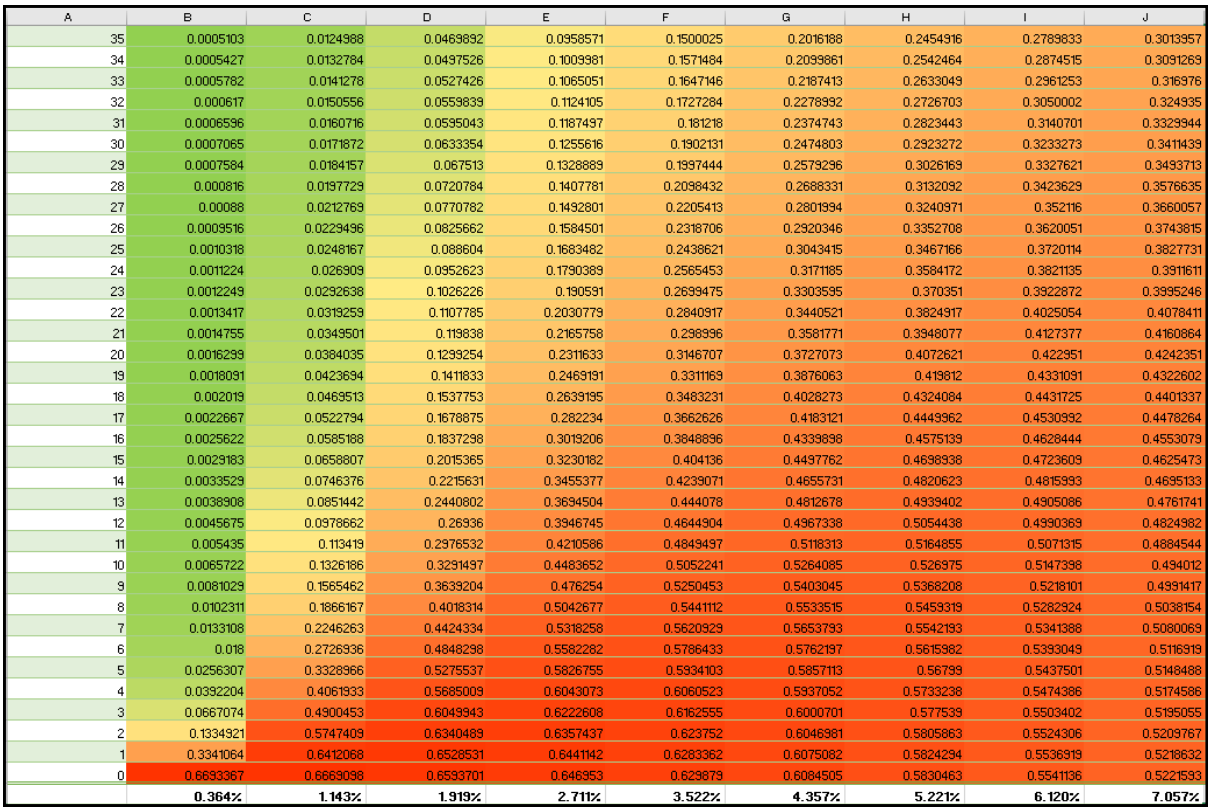
In the user location validation stage, testing is carried out by testing the user's location at several points and comparing the results with the actual coordinates. Tests are performed at different times of the day to ensure accurate results. The test results show that the system successfully retrieves the user's coordinates with high accuracy and sends them correctly to the server. The test results show that the geolocation system that has been created has high accuracy in determining the user's location. In the tests carried out, the coordinates sent by the system to the server have a small difference from the actual coordinates. This shows that the system can retrieve the user's location with high accuracy. In testing application performance, testing is carried out to see how quickly the application responds to changes in the user's location and displays the user's location on the map. The test results show that the application responds quickly and can display the user's location accurately in a short time. This shows that the application can be used properly and can help users to determine their location accurately. This section will describe the mathematical analysis of the methods used in module development. The intended mathematical analysis is the deviation of the numbers resulting from calculating the distance between coordinates. The Fig. 18 is the data and its mathematical analysis.

Information and explanations regarding the picture. The table in the image is the result of calculating the deviation between the numbers from the Vincenty and Haversine methods and the condition when the starting point is at coordinates (0, 0) (latitude 0, longitude 0) or at the equator to the coordinates at the point according to the image. The latitude limit is -90 < latitude < 90. The longitude limit is -180 ≤ longitude < 180. In the picture above, you can see that there is column A represents the longitude value from the destination point, and columns B to N represent the latitude value. For longitude, the data is being simulated from -180° up to 179° in column A. For latitude, the data is being simulated from -90° up to 90° with increments of 15° each column except for column H because it has a value of 1°. The value of 1° is given in column H because the Vincenty method fails to calculate at points close to the equator. It can be seen that the calculation of the deviation in the image above has a symmetrical pattern in which the difference in distance between points (0.0) to points (-15,-1), (-15,1), (15,1), (15, 1) have the same number. Therefore, data retrieval can be simplified into the Fig. 19.



A	B	C	D	E	F	G	H	I	J	K	L	M	N
-19	0.1675702	0.2559356	0.34965	0.4209761	0.419812	0.2469191	0.0018091	0.2469191	0.419812	0.4209761	0.34965	0.2559356	0.1675702
-18	0.1675702	0.2567208	0.3521783	0.4270441	0.4324084	0.2639195	0.002019	0.2639195	0.4324084	0.4270441	0.3521783	0.2567208	0.1675702
-17	0.1675702	0.2574678	0.3545991	0.4329303	0.4449962	0.282234	0.0022667	0.282234	0.4449962	0.4329303	0.3545991	0.2574678	0.1675702
-16	0.1675702	0.2581758	0.3569081	0.4386153	0.4575139	0.3019206	0.0025622	0.3019206	0.4575139	0.4386153	0.3569081	0.2581758	0.1675702
-15	0.1675702	0.2588443	0.3591011	0.4440794	0.4698938	0.3230182	0.0029183	0.3230182	0.4698938	0.4440794	0.3591011	0.2588443	0.1675702
-14	0.1675702	0.2594727	0.3611737	0.4493033	0.4820623	0.3455377	0.0033529	0.3455377	0.4820623	0.4493033	0.3611737	0.2594727	0.1675702
-13	0.1675702	0.2600603	0.3631222	0.4542675	0.4939402	0.3694504	0.0038908	0.3694504	0.4939402	0.4542675	0.3631222	0.2600603	0.1675702
-12	0.1675702	0.2606066	0.3649427	0.4589532	0.5054438	0.3946745	0.0045675	0.3946745	0.5054438	0.4589532	0.3649427	0.2606066	0.1675702
-11	0.1675702	0.2611111	0.3666316	0.4633418	0.5164855	0.4210586	0.005435	0.4210586	0.5164855	0.4633418	0.3666316	0.2611111	0.1675702
-10	0.1675702	0.2615734	0.3681855	0.4674156	0.526975	0.4483652	0.0065722	0.4483652	0.526975	0.4674156	0.3681855	0.2615734	0.1675702
-9	0.1675702	0.2619929	0.3696013	0.4711577	0.5368208	0.476254	0.0081029	0.476254	0.5368208	0.4711577	0.3696013	0.2619929	0.1675702
-8	0.1675702	0.2623694	0.3708761	0.4745523	0.5459319	0.5042677	0.0102311	0.5042677	0.5459319	0.4745523	0.3708761	0.2623694	0.1675702
-7	0.1675702	0.2627025	0.3720073	0.4775845	0.5542193	0.5318258	0.0133108	0.5318258	0.5542193	0.4775845	0.3720073	0.2627025	0.1675702
-6	0.1675702	0.2629917	0.3729925	0.480241	0.5615982	0.5582282	0.018	0.5582282	0.5615982	0.480241	0.3729925	0.2629917	0.1675702
-5	0.1675702	0.263237	0.3738298	0.4825098	0.56799	0.5826755	0.0256307	0.5826755	0.56799	0.4825098	0.3738298	0.263237	0.1675702
-4	0.1675702	0.2634379	0.3745172	0.4843807	0.5733238	0.6043073	0.0392204	0.6043073	0.5733238	0.4843807	0.3745172	0.2634379	0.1675702
-3	0.1675702	0.2635945	0.3750534	0.485845	0.577539	0.6222608	0.0667074	0.6222608	0.577539	0.485845	0.3750534	0.2635945	0.1675702
-2	0.1675702	0.2637064	0.3754372	0.4868958	0.5805863	0.6357437	0.1334921	0.6357437	0.5805863	0.4868958	0.3754372	0.2637064	0.1675702
-1	0.1675702	0.2637735	0.3756678	0.4875283	0.5824294	0.6441142	0.3341064	0.6441142	0.5824294	0.4875283	0.3756678	0.2637735	0.1675702
0	0.1675702	0.2637959	0.3757447	0.4877394	0.5830463	0.646953	0.6693367	0.646953	0.5830463	0.4877394	0.3757447	0.2637959	0.1675702
1	0.1675702	0.2637735	0.3756678	0.4875283	0.5824294	0.6441142	0.3341064	0.6441142	0.5824294	0.4875283	0.3756678	0.2637735	0.1675702
2	0.1675702	0.2637064	0.3754372	0.4868958	0.5805863	0.6357437	0.1334921	0.6357437	0.5805863	0.4868958	0.3754372	0.2637064	0.1675702
3	0.1675702	0.2635945	0.3750534	0.485845	0.577539	0.6222608	0.0667074	0.6222608	0.577539	0.485845	0.3750534	0.2635945	0.1675702
4	0.1675702	0.2634379	0.3745172	0.4843807	0.5733238	0.6043073	0.0392204	0.6043073	0.5733238	0.4843807	0.3745172	0.2634379	0.1675702
5	0.1675702	0.263237	0.3738298	0.4825098	0.56799	0.5826755	0.0256307	0.5826755	0.56799	0.4825098	0.3738298	0.263237	0.1675702
6	0.1675702	0.2629917	0.3729925	0.480241	0.5615982	0.5582282	0.018	0.5582282	0.5615982	0.480241	0.3729925	0.2629917	0.1675702
7	0.1675702	0.2627025	0.3720073	0.4775845	0.5542193	0.5318258	0.0133108	0.5318258	0.5542193	0.4775845	0.3720073	0.2627025	0.1675702
8	0.1675702	0.2623694	0.3708761	0.4745523	0.5459319	0.5042677	0.0102311	0.5042677	0.5459319	0.4745523	0.3708761	0.2623694	0.1675702
9	0.1675702	0.2619929	0.3696013	0.4711577	0.5368208	0.476254	0.0081029	0.476254	0.5368208	0.4711577	0.3696013	0.2619929	0.1675702
10	0.1675702	0.2615734	0.3681855	0.4674156	0.526975	0.4483652	0.0065722	0.4483652	0.526975	0.4674156	0.3681855	0.2615734	0.1675702
11	0.1675702	0.2611111	0.3666316	0.4633418	0.5164855	0.4210586	0.005435	0.4210586	0.5164855	0.4633418	0.3666316	0.2611111	0.1675702
12	0.1675702	0.2606066	0.3649427	0.4589532	0.5054438	0.3946745	0.0045675	0.3946745	0.5054438	0.4589532	0.3649427	0.2606066	0.1675702
13	0.1675702	0.2600603	0.3631222	0.4542675	0.4939402	0.3694504	0.0038908	0.3694504	0.4939402	0.4542675	0.3631222	0.2600603	0.1675702
14	0.1675702	0.2594727	0.3611737	0.4493033	0.4820623	0.3455377	0.0033529	0.3455377	0.4820623	0.4493033	0.3611737	0.2594727	0.1675702
15	0.1675702	0.2588443	0.3591011	0.4440794	0.4698938	0.3230182	0.0029183	0.3230182	0.4698938	0.4440794	0.3591011	0.2588443	0.1675702
16	0.1675702	0.2581758	0.3569081	0.4386153	0.4575139	0.3019206	0.0025622	0.3019206	0.4575139	0.4386153	0.3569081	0.2581758	0.1675702
17	0.1675702	0.2574678	0.3545991	0.4329303	0.4449962	0.282234	0.0022667	0.282234	0.4449962	0.4329303	0.3545991	0.2574678	0.1675702
18	0.1675702	0.2567208	0.3521783	0.4270441	0.4324084	0.2639195	0.002019	0.2639195	0.4324084	0.4270441	0.3521783	0.2567208	0.1675702
19	0.1675702	0.2559356	0.34965	0.4209761	0.419812	0.2469191	0.0018091	0.2469191	0.419812	0.4209761	0.34965	0.2559356	0.1675702
20	0.1675702	0.2551129	0.347019	0.4147457	0.4072621	0.2311633	0.0016299	0.2311633	0.4072621	0.4147457	0.347019	0.2551129	0.1675702

Fig. 18: The data and its mathematical analysis



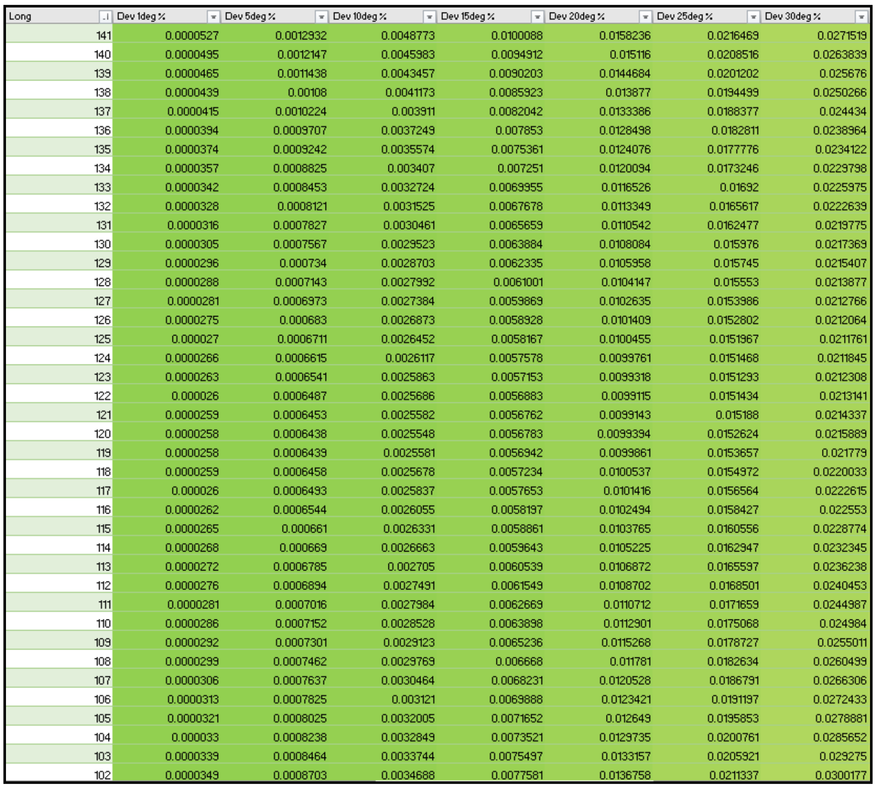
A	B	C	D	E	F	G	H	I	J
35	0.0005103	0.0124988	0.0469892	0.0958571	0.1500025	0.2016188	0.2454916	0.2789833	0.3013957
34	0.0005427	0.0132784	0.0497526	0.1009981	0.1571484	0.2099861	0.2542464	0.2874515	0.3091269
33	0.0005782	0.0141278	0.0527426	0.1065051	0.1647146	0.2187413	0.2633049	0.2961253	0.316976
32	0.000617	0.0150556	0.0559839	0.1124105	0.1727284	0.2278992	0.2726703	0.3050002	0.324935
31	0.0006596	0.0160716	0.0595043	0.1187497	0.181218	0.2374743	0.2823443	0.3140701	0.3329944
30	0.0007065	0.0171872	0.0633354	0.1255616	0.1902131	0.2474803	0.2923272	0.3233273	0.3411439
29	0.0007584	0.0184157	0.067513	0.1328889	0.1997444	0.2579296	0.3026169	0.3327621	0.3493713
28	0.000816	0.0197729	0.0720784	0.1407781	0.2098432	0.2688331	0.3132092	0.3423629	0.3576635
27	0.00088	0.0212769	0.0770782	0.1492801	0.2205413	0.2801994	0.3240971	0.352116	0.3660057
26	0.0009516	0.0229496	0.0825662	0.1584501	0.2318706	0.2920346	0.3352708	0.3620051	0.3743815
25	0.0010318	0.0248167	0.088604	0.1683482	0.2438621	0.3043415	0.3467166	0.3720114	0.3827731
24	0.0011224	0.026909	0.0952623	0.1790389	0.2565453	0.3171185	0.3584172	0.3821135	0.3911611
23	0.0012249	0.0292638	0.1026226	0.190591	0.2699475	0.3303595	0.370351	0.3922872	0.3995246
22	0.0013417	0.0319259	0.1107785	0.2030779	0.2840917	0.3440521	0.3824917	0.4025054	0.4078411
21	0.0014755	0.0349501	0.119838	0.2165758	0.298996	0.3581771	0.3948077	0.4127377	0.4160864
20	0.0016299	0.0384035	0.1299254	0.2311633	0.3146707	0.3727073	0.4072621	0.422951	0.4242351
19	0.0018091	0.0423694	0.1411833	0.2469191	0.3311169	0.3876063	0.419812	0.4331091	0.4322602
18	0.002019	0.0469513	0.1537753	0.2639195	0.3483231	0.4028273	0.4324084	0.4431725	0.4401337
17	0.0022667	0.0522794	0.1678875	0.282234	0.3662626	0.4183121	0.4449962	0.4530992	0.4478264
16	0.0025622	0.0585188	0.1837298	0.3019206	0.3848896	0.4339898	0.4575139	0.4628444	0.4553079
15	0.0029183	0.0658807	0.2015365	0.3230182	0.404136	0.4497762	0.4698938	0.4723609	0.4625473
14	0.0033529	0.0746376	0.2215631	0.3455377	0.4239071	0.4655731	0.4820623	0.4815993	0.4695133
13	0.0038908	0.0851442	0.2440802	0.3694504	0.444078	0.4812678	0.4939402	0.4905086	0.4761741
12	0.0045675	0.0978662	0.26936	0.3946745	0.4644904	0.4967338	0.5054438	0.4990369	0.4824982
11	0.005435	0.113419	0.2976532	0.4210586	0.4849497	0.5118313	0.5164855	0.5071315	0.4884544
10	0.0065722	0.1326186	0.3291497	0.4483652	0.5052241	0.5264085	0.526975	0.5147398	0.494012
9	0.0081029	0.1565462	0.3639204	0.476254	0.5250453	0.5403045	0.5368208	0.5218101	0.4991417
8	0.0102311	0.1866167	0.4018314	0.5042677	0.5441112	0.5533515	0.5459319	0.5282924	0.5038154
7	0.0133108	0.2246263	0.4424334	0.5318258	0.5620929	0.5653793	0.5542193	0.5341388	0.5080069
6	0.018	0.2726936	0.4848298	0.5582282	0.5786433	0.5762197	0.5615982	0.5393049	0.5116919
5	0.0256307	0.3328966	0.5275537	0.5826755	0.5934103	0.5857113	0.56799	0.5437501	0.5148488
4	0.0392204	0.4061933	0.5685009	0.6043073	0.6060523	0.5937052	0.5733238	0.5474386	0.5174586
3	0.0667074	0.4900453	0.6049943	0.6222608	0.6162555	0.6000701	0.577539	0.5503402	0.5195055
2	0.1334921	0.5747409	0.6340489	0.6357437	0.623752	0.6046981	0.5805863	0.5524306	0.5209767
1	0.3341064	0.6412068	0.6528531	0.6441142	0.6283362	0.6075082	0.5824294	0.5536919	0.5218632
0	0.6693367	0.6669098	0.6593701	0.646953	0.629879	0.6084505	0.5830463	0.5541136	0.5221593
	0.364%	**1.143%**	**1.919%**	**2.711%**	**3.522%**	**4.357%**	**5.221%**	**6.120%**	**7.057%**

Fig. 19: The data retrieval

The table shows that the accuracy of the Haversine method is lower as it approaches the equator and the longitude is 0°. The difference in the numbers calculated by the Haversine and Vincenty methods at a distance of (0.0)→(1.0) has a deviation of 0.6%. The difference in these figures is due to the different fundamentals of the two methods. Haversine has a fundamental assumption that the earth is round while Vincenty assumes that the earth is an ellipsoid. Please note that the image above only represents data up to latitude represents data up to 40° latitude. However, as the point is further away from 0° longitude, the deviation becomes smaller. This is because, in the operation of the Vincenty method, the method has constant variables which, according to Thaddeus Vincenty, optimize the level of accuracy in calculating distances. The curvature of the line based on the ratio of flatness of the earth affects this difference in distance. Compared to the Haversine method, this method does not have a constant variable or even assumes that the earth is an ellipsoid, so we can see the difference. Fig. 20 consists of the level of accuracy in calculating distances.

The table in the image shows the area where the calculation of the distance between coordinates (0,0) → (*Latitude*, *Longitude*) using both the Vincenty and Haversine methods has the smallest deviation. The smallest deviation point is located at the distance (0,0)→(1, 119)~(1, 120). The following is a radar chart to support the visualization of deviation data for the table in Fig. 21.

The description is as follows: The red line is the deviation in percent at latitude 1° of the point of destination. The yellow line for latitude is 30° from the point of destination. The green line for latitude 60° of the point of destination. The blue line for latitude 90° of the point of destination. The conclusions that can be drawn from the visualization with the graph above are: The deviation becomes more stable as it approaches the poles. Therefore, the pole can be assumed as a stable point to explore further the deviation of these two methods. As seen in the graph above, when the distance approaches -180° latitude at 1° latitude, which is the point approaching the antipodal, the deviation spikes. This proves Thaddeus Vincenty's statement that the method will fail to find a solution when approaching an antipodal point or even calculating a line at the equator. The analysis will be continued with the next parameter, which is centered on the polar point. What is meant by centered on the polar point is the starting point centered on one of the earth's poles. The following Fig. 22 is an image regarding the graph for the deviation of the polar parameters.



Long	Dev 1deg %	Dev 5deg %	Dev 10deg %	Dev 15deg %	Dev 20deg %	Dev 25deg %	Dev 30deg %
141	0.0000527	0.0012932	0.0048773	0.0100088	0.0158236	0.0216469	0.0271519
140	0.0000495	0.0012147	0.0045983	0.0094912	0.015116	0.0208516	0.0263839
139	0.0000465	0.0011438	0.0043457	0.0090203	0.0144684	0.0201202	0.025676
138	0.0000439	0.00108	0.0041173	0.0085923	0.013877	0.0194499	0.0250266
137	0.0000415	0.0010224	0.003911	0.0082042	0.0133386	0.0188377	0.024434
136	0.0000394	0.0009707	0.0037249	0.007853	0.0128498	0.0182811	0.0238964
135	0.0000374	0.0009242	0.0035574	0.0075361	0.0124076	0.0177776	0.0234122
134	0.0000357	0.0008825	0.003407	0.007251	0.0120094	0.0173246	0.0229798
133	0.0000342	0.0008453	0.0032724	0.0069955	0.0116526	0.01692	0.0225975
132	0.0000328	0.0008121	0.0031525	0.0067678	0.0113349	0.0165617	0.0222639
131	0.0000316	0.0007827	0.0030461	0.0065659	0.0110542	0.0162477	0.0219775
130	0.0000305	0.0007567	0.0029523	0.0063884	0.0108084	0.015976	0.0217369
129	0.0000296	0.000734	0.0028703	0.0062335	0.0105958	0.015745	0.0215407
128	0.0000288	0.0007143	0.0027992	0.0061001	0.0104147	0.015553	0.0213877
127	0.0000281	0.0006973	0.0027384	0.0059869	0.0102635	0.0153986	0.0212766
126	0.0000275	0.000683	0.0026873	0.0058928	0.0101409	0.0152802	0.0212064
125	0.000027	0.0006711	0.0026452	0.0058167	0.0100455	0.0151967	0.0211761
124	0.0000266	0.0006615	0.0026117	0.0057578	0.0099761	0.0151468	0.0211845
123	0.0000263	0.0006541	0.0025863	0.0057153	0.0099318	0.0151293	0.0212308
122	0.000026	0.0006487	0.0025686	0.0056883	0.0099115	0.0151434	0.0213141
121	0.0000259	0.0006453	0.0025582	0.0056762	0.0099143	0.015188	0.0214337
120	0.0000258	0.0006438	0.0025548	0.0056783	0.0099394	0.0152624	0.0215889
119	0.0000258	0.0006439	0.0025581	0.0056942	0.0099861	0.0153657	0.021779
118	0.0000259	0.0006458	0.0025678	0.0057234	0.0100537	0.0154972	0.0220033
117	0.000026	0.0006493	0.0025837	0.0057653	0.0101416	0.0156564	0.0222615
116	0.0000262	0.0006544	0.0026055	0.0058197	0.0102494	0.0158427	0.022553
115	0.0000265	0.000661	0.0026331	0.0058861	0.0103765	0.0160556	0.0228774
114	0.0000268	0.000669	0.0026663	0.0059643	0.0105225	0.0162947	0.0232345
113	0.0000272	0.0006785	0.002705	0.0060539	0.0106872	0.0165597	0.0236238
112	0.0000276	0.0006894	0.0027491	0.0061549	0.0108702	0.0168501	0.0240453
111	0.0000281	0.0007016	0.0027984	0.0062669	0.0110712	0.0171659	0.0244987
110	0.0000286	0.0007152	0.0028528	0.0063898	0.0112901	0.0175068	0.024984
109	0.0000292	0.0007301	0.0029123	0.0065236	0.0115268	0.0178727	0.0255011
108	0.0000299	0.0007462	0.0029769	0.006668	0.011781	0.0182634	0.0260499
107	0.0000306	0.0007637	0.0030464	0.0068231	0.0120528	0.0186791	0.0266306
106	0.0000313	0.0007825	0.003121	0.0069888	0.0123421	0.0191197	0.0272433
105	0.0000321	0.0008025	0.0032005	0.0071652	0.012649	0.0195853	0.0278881
104	0.000033	0.0008238	0.0032849	0.0073521	0.0129735	0.0200761	0.0285652
103	0.0000339	0.0008464	0.0033744	0.0075497	0.0133157	0.0205921	0.029275
102	0.0000349	0.0008703	0.0034688	0.0077581	0.0136758	0.0211337	0.0300177

Fig. 20: The level of accuracy in calculating distances

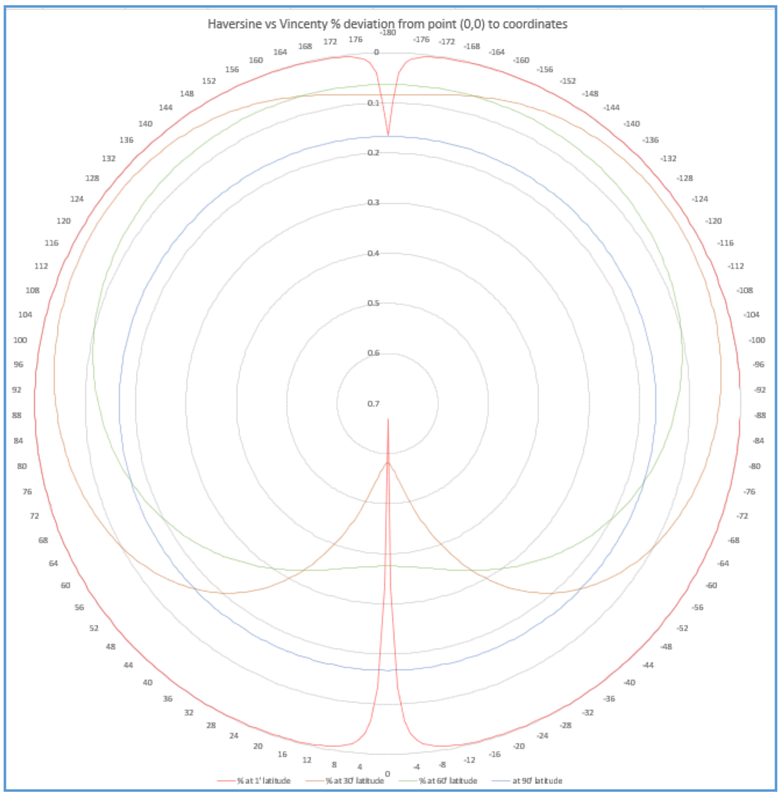


Fig. 21: The radar chart to support the visualization of deviation data

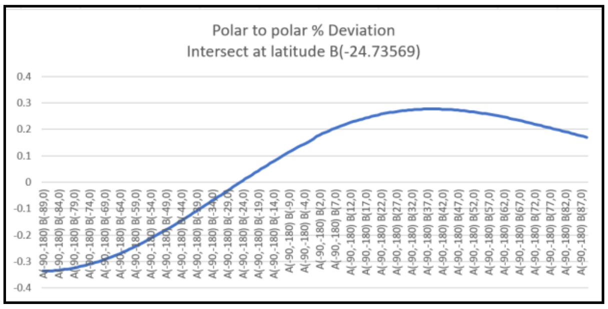


Fig. 22: The graph for the deviation of the polar parameters

The graph in the image above visualizes the deviation between the two numbers from the Vincenty and Haversine methods. As seen in the initial iteration, namely (-90, -180) → (89, 0), both methods have a deviation of around -0.33631% where the distance calculated using the Haversine method is smaller than Vincenty. This deviation will continue to shrink to -24.73569° latitude and expand to -35° latitude and then shrink again until it approaches the antipodal point at the south pole. From this graph, it can be concluded that the deviation will shrink when approaching the antipodal point and when the latitude is closer to the polar. Using the Haversine method is much easier than the Vincenty method. However, based on data from WGS84, the earth is not round but has an ellipsoid shape. The level of the earth's ellipsoid is represented by the flattening variable which is 298.257223563. The use of the Haversine method can be superior when calculating the distance approaching the antipodal point from the initial distance even though the level of accuracy is less. The visualized data can be used as a reference for further research to improve the Vincenty method by using the deviation between the two methods considering that the polar radius and equatorial radius are only less than 100 km apart.

In this research, a geolocation system module has been successfully developed that can be used to validate user location coordinates in Android applications using the Flutter framework. Validation of the coordinates of the user's location is carried out by comparing the coordinates obtained from the geolocation module that has been made with the user's original coordinates obtained from Google Maps. The validation results show that the built geolocation module can produce coordinates of the user's location with a good level of accuracy. Application performance has also proven to be good with an average application response time that is quite fast. Although this research has succeeded in producing a geolocation system module that can be used to validate the coordinates of a user's location on an Android application, there are several things that can serve as suggestions for further research. First, future research can try to integrate the geolocation module that has been built with the existing geolocation system on Android devices, such as using GPS services or network-based locations. This is expected to increase the accuracy of the results of the geolocation module that has been built. Second, future research can try to develop a geolocation module that can be used to validate user location coordinates on iOS applications or other mobile platforms. Third, future research can try to integrate the geolocation module that has been built with the navigation system on the Android application so that users can see the closest route to their destination in real-time.

Conclusion

The conclusions drawn by the authors in this study can be grouped into several aspects which include aspects of mathematics and programming (UI design and User Experience). In the mathematical aspect, the use of the Haversine method in matters requiring high accuracy such as setting the satellite constellation is very dangerous because if it is assumed that the Vincenty method is an accurate method, then the difference in distance of 0.6% as shown in the analysis section is very large from the point of interest. The deviation data between the calculation results of the two methods prepared at the time of writing this research is intended for future researchers to be able to refine the Vincenty method by creating an equation based on the deviation with the results from the Haversine method. This equation is expected to be able to complete what is lacking in the use of the Vincenty method such as failure and inaccuracy of calculations approaching the antipodal point. In the programming aspect, the application of the Vincenty method to modules is made unreliable in conditions where the distance calculation is close to the antipodal point because the programming algorithm will error (not finding the meeting point even though it has gone through the highest iteration).

Aspects of separate programming can still be divided into 2 parts, namely as follows: (a) In the UI design section, applications that are integrated with modules meet the criteria based on the Eight Golden Rules theory. This statement can be proven by survey results which are dominated by “good” and “very good” responses to the 8 questions according to the eight golden rules. (b) In the User Experience (UX) section, applications integrated with modules have varied responses. It can be concluded that this application lacks attractiveness because the value achieved on the UEQ during the evaluation process has a value below the average (1.05). Even though it does not have so high attractiveness, the highest score is obtained on the accuracy scale with a very good value

(1.97) which means that the module is made according to its purpose. Further, it is hoped that the data contained in this study will be used to develop a new method of calculating the distance between the two coordinates on the earth's surface, at least a method that enhances the Vincenty method. For the programming side, it is expected that the algorithms of the two methods obtained in the research can be developed using other mathematical knowledge to make the method a hybrid. If the algorithm is too complex and requires integral operations, it is advisable not to use Flutter/Dart because the programming language and framework do not yet support something complex and it will take a long time to process the data. It is also recommended, to find a method that has results close to real results, further surveys and research are needed including dedicated hardware that can guarantee accurate results such as laser rangefinders and micro-satellite.

The Haversine and Vincenty methods are both used to calculate the distance between two points on the surface of a sphere, such as the Earth. The Haversine method is simpler and faster but assumes that the earth is a perfect sphere. The Vincenty method is more complex and slower but takes into account the fact that the earth is an oblate spheroid, which is more accurate than the Haversine method. In the context of a geolocation system module creation, the choice between the two methods depends on the required level of accuracy and the computational resources available. If the application only needs to calculate short distances and speed is a priority, the Haversine method may be more suitable. However, if the application needs to calculate longer distances with higher accuracy, the Vincenty method may be more appropriate.

Acknowledgment

The authors would like to thank the reviewers for their informative comments, and suggestions ideas, which have helped mould this manuscript into something that is worthy of publication. This study is supported by the research and technology transfer office, Bina Nusantara University.

Funding Information

This study is supported and funded by Bina Nusantara University Research and Technology Transfer Office under the terms of the university's International Research Grant (PIB 2023) under contract number 029/VRRTT/III/2023.

Author’s Contributions

Elmer Matthew Japara and Samsul Arifin: Coding the program, written, and finalizing the manuscript.

Edy Irwansyah: Simulating the data, tidying up the theoretical basis and the methods we use.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that there is no conflict of interest in this study and no ethical issues involved.

References

Al Maki, W. F., Tajrial, R., & Arifin, S. (2023). Automated Classification of Multi-Class Human Protozoan Parasites using Xception as Transfer Learning. *International Journal of Intelligent Systems and Applications in Engineering*, *11*(2), 817-825. https://www.ijisae.org/index.php/IJISAE/article/view/2895

Arb, G. I., & Al-Majdi, K. (2020, May). A freights status management system based on Dart and Flutter programming language. In *Journal of Physics: Conference Series* (Vol. *1530*, No. 1, p. 012020). IOP Publishing. https://doi.org/10.1088/1742-6596/1530/1/012020

Arifin, S., & Garminia, H. (2019). Uniserial dimension of module zm × zn over Z using python. *Int. J. Sci. Technol. Res*, *8*, 194-9. https://www.researchgate.net/publication/334769797_Uniserial_Dimension_Of_Module_ZmxZn_Over_Z_Using_Python

Arifin, S., & Muktyas, I. B. (2018). Membangkitkan suatu matriks unimodular dengan python. *Jurnal Derivat: Jurnal Matematika dan Pendidikan Matematika*, *5*(2), 1-9. https://doi.org/10.31316/j.derivat.v5i2.361

Arifin, S., Muktyas, I. B., & Sukmawati, K. I. (2021, February). Product of two groups integers modulo m, n and their factor groups using python. In *Journal of Physics: Conference Series* (Vol. *1778*, No. 1, p. 012026). IOP Publishing. https://doi.org/10.1088/1742-6596/1778/1/012026

Arifin, S., Muktyas, I. B., Al Maki, W. F., & Aziz, M. M. (2022). Graph coloring program of exam scheduling modeling based on Bitwise coloring algorithm using Python. *Journal of Computer Science*, *18*(1), 26-32. https://doi.org/10.3844/jcssp.2022.26.32.

Azdy, R. A., & Darnis, F. (2020, April). Use of Haversine formula in finding distance between temporary shelter and waste end processing sites. In *Journal of Physics: Conference Series* (Vol. 1500, No. 1, p. 012104). IOP Publishing. https://doi.org/10.1088/1742-6596/1500/1/012104

Cagol, M., & Colombi, A. E. (2016). Pythagoras on the rocks. A proof with bricks. *Form@ re-Open Journal Per La Formazione in Rete*, *16*(1), 268-274. https://doi.org/10.13128/formare-17903

Dalenogare, L. S., Benitez, G. B., Ayala, N. F., & Frank, A. G. (2018). The expected contribution of Industry 4.0 technologies for industrial performance. *International Journal of Production Economics*, *204*, 383-394. https://doi.org/10.1016/j.ijpe.2018.08.019

Gade, K. (2010). A non-singular horizontal position representation. *The Journal of Navigation*, *63*(3), 395-417. https://doi.org/10.1017/S0373463309990415

Hartanto, A. D., Susanto, M. R., Ilham, H. D., Retnaningsih, R., & Nurdiyanto, H. (2018, July). Mobile Technologies of Formulation Haversine Application and Location Based Service. In *ICASI 2018: Joint Workshop KO2PI and The 1st International Conference on Advance & Scientific Innovation* (p. 92). European Alliance for Innovation. ISBN: 1631901621.

Hernández-Lamas, P., Cabau-Anchuelo, B., de Castro-Cuartero, Ó., & Bernabéu-Larena, J. (2021). Mobile Applications, Geolocation and Information Technologies for the Study and Communication of the Heritage Value of Public Works. *Sustainability*, *13*(4), 2083. https://doi.org/10.3390/su13042083

Setyorini, I., & Ramayanti, D. (2019). Finding nearest mosque using Haversine formula on Android platform. *Jurnal Online Informatika*, *4*(1), 57-62. https://doi.org/10.15575/join.v4i1.267

Jackson, W., & Jackson, W. (2017). An Introduction to Android 7.0 Nougat. *Android Apps for Absolute Beginners: Covering Android 7*, 1-15. https://doi.org/10.1007/978-1-4842-2268-3_1

Kumar, A., Sato, Y., Oishi, T., Ono, S., & Ikeuchi, K. (2014). Improving gps position accuracy by identification of reflected gps signals using range data for modeling of urban structures. *Seisan Kenkyu*, *66*(2), 101-107. https://doi.org/10.11188/seisankenkyu.66.101

Kusuma, W. A., Jantan, A. H., bin Abdullah, R., & Admodisastro, N. (2023). Mapping User Experience Information Overload Problems Across Disciplines. *JOIV: International Journal on Informatics Visualization*, *7*(1), 22-29. https://doi.org/10.30630/joiv.7.1.1588

Mahmoud, H., & Akkari, N. (2016, March). Shortest path calculation: A comparative study for location-based recommender system. In *2016 world symposium on computer applications & research (WSCAR)* (pp. 1-5). IEEE. https://doi.org/10.1109/WSCAR.2016.16

Maor, E. (2019). *The Pythagorean theorem: A 4,000-year history*. Princeton University Press. ISBN: 0691196885.

Nabie, M. J., Akayuure, P., Ibrahim-Bariham, U. A., & Sofo, S. (2018). Trigonometric Concepts: PRE-Service Teachers' Perceptions and Knowledge. *Journal on Mathematics Education*, *9*(1), 169-182.

Nystedt, P. (2017). A proof of the law of sines using the law of cosines. *Mathematics Magazine*, *90*(3), 180-181. https://doi.org/10.4169/math.mag.90.3.180

Pamunuwa, V. P. Waidyaratne, P. R. Weerakoon, DS. S. Wisidagama, N. S. Rmdkn, R., Pamunuwa, P. Waidyaratne, P. Weerakoon, S. Wisidagama, N . Rathnayake, D. (2022). “Study on Waterfall Model over PcD. UcT Model,” *Softw. Model. Cl. Work.*, no. May, https://doi.org/10.13140/RG.2.2.33241.60007

Pressman, R. S. (2005). *Software engineering: A practitioner's approach*. Palgrave macmillan. ISBN: 007301933X.

Restrepo-Calle, F., Ramírez-Echeverry, J. J., & González, F. A. (2020). Using an interactive software tool for the formative and summative evaluation in a computer programming course: An experience report. *Global Journal of Engineering Education*, *22*(3), 174-185.

Shavitt, Y., & Zilberman, N. (2011). A geolocation databases study. *IEEE Journal on Selected Areas in Communications*, *29*(10), 2044-2056. https://doi.org/10.1109/JSAC.2011.111214

Tyagi, P. (2021). Introduction to Flutter, in *Pragmatic Flutter*, 1st Ed. CRC Press of Taylor & Francis Group, pp: 8.

Utami, A. N., & Mampouw, H. L. (2020). Pengembangan Media Smart Trigo untuk Pembelajaran Trigonometri. *Jurnal Cendekia: Jurnal Pendidikan Matematika*, *4*(2), 939-949. https://doi.org/10.31004/cendekia.v4i2.227

Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, *23*(176), 88-93. https://doi.org/10.1179/sre.1975.23.176.88

Winoto, D. A., & Christanto, F. W. (2020). Implementasi Google Maps Api Dalam Pengembangan Sistem Informasi Geografis Taman Kota Dan Kampung Tematik Berbasis Android Di Kota Semarang (Studi Kasus: Pejabat Pengelola Informasi Dan Dokumentasi Kota Semarang). *Jurnal Teknologi Informasi dan Ilmu Komputer*, *7*(1), 9-16.

Wu, Z. Y. (2019). An radio-frequency identification security authentication mechanism for Internet of things applications. *International Journal of Distributed Sensor Networks*, *15*(7), https://doi.org/10.1177/1550147719862223

Zammetti, F. W. (2019). *Practical Flutter: Improve Your Mobile Development with Google’s Latest Open-Source SDK*, 1st Editio. New York: Apress.