An Accurate Tagging Algorithm in Augmented Reality for Mobile Device Screens

DOĞA ERIŞIK, AHMET KARAMAN, GÜLFEM IŞIKLAR ALPTEKİN,
ÖZLEM DURMAZ İNCEL
Computer Engineering
Galatasaray University
Çırağan Cad. No:36 Ortaköy İstanbul
TURKEY
Doga.erisik@avea.com.tr; Ahmet.Karaman@aktifbank.com.tr; gisiklar@gsu.edu.tr;
odincel@gsu.edu.tr

Abstract: - Augmented reality (AR) is a type of virtual reality aiming to duplicate real world's environment on a computer's video feed. The mobile application, which is built for this project (called SARAS), enables annotating real world point of interests (POIs) that are located near mobile user. In this paper, we aim at introducing a robust and simple algorithm for placing labels in an augmented reality system. The system places labels of the POIs on the mobile device screen whose GPS coordinates are given. The proposed algorithm is compared to an existing one in terms of energy consumption and accuracy. The results show that the proposed algorithm gives better results in energy consumption and accuracy while standing still, and acceptably accurate results when driving. The technique provides benefits to AR browsers with its open access algorithm. Going forward, the algorithm will be improved to more rapidly react to position changes while driving.

Key-Words: - Labeling POI; localization; accurate tagging algorithm; augmented reality; location-based AR, mobile augmented reality application.

1 Introduction

An augmented reality (AR) application allows us to see the real world overlaid with digital information. Overlaying labels or virtual objects provides with richer experiences for individuals. AR systems typically analyze the video stream provided by mobile device camera in real time. They make use of various sensors including GPS, digital compass (magnetometer), accelerometer or gyroscope in order to determine the position and the orientation of the user.

In this research, a sensor-based AR application (called SARAS) is built. The application works as follows: the user gets the viewing angle (or framing) within the camera to be launched in SARAS by looking at a direction in shopping centers, on the street or on the road (highway, city). If there are points of interests (POIs) related to the bank in the viewing angle (also inside framing), this information appears as a list on the screen. If a point is selected from the list, detailed information (such as details of the campaign) is displayed. If the user is in a multi-storey shopping center, floor distinction is made by 3D effect. The main objective of SARAS is to set a tag for each POI that is visible from the mobile screen. The POIs are places in the world represented by a latitude and a longitude. The labels belonging to these points (GPS coordinates) is superimposed in the view of the camera. Particularly, the SARAS application works with the POI's of a private bank in Turkey, Yapı Kredi Bank, namely its merchants, branches and ATMs. Each type of POI is tagged with a different color. This application is considered as an alternative channel for a bank to inform its customers/potential customers about their merchants and campaigns. In case where there is noGPS connection (such as at shopping malls or subway stations), the application reads the QR code that is stuck on the shop window. This approach is called a 'virtual window' in this project.

The focus of this paper is on the tagging algorithm that is proposed for this application. There are different techniques for creating labels: immediate tagging, fast tagging, map tagging and accurate tagging [1]. In accurate tagging, users focus on the POI from several positions, saving the position, Azimuth and roll values. Then, lines are traced from these positions that intersect in the same point. The proposed algorithm does not use these types of lines, but it uses the position and the Azimuth value. Besides, it determines the real location of the POI. These are the reasons why it is considered in the accurate tagging class.

The rest of the paper is organized as follows. Section 2 discusses the related work in the literature and their differences from this one. Section 3 gives brief explanation of SARAS application. In Section 4, the algorithm is given in detailed steps. The results and the performance analysis are discussed in Section 5. Finally, conclusions and future work are given in Section 6.

2 Problem Formulation

An AR system, called LibreGeoSecial that works both outdoors and indoors is introduced in [1]. It allow browsing tags associated with objects of the real world and linking media to objects. It uses GPS and compass to recognize location and orientation similar to our technique; but the difference is that it makes use of these sensor values together with image processing to improve accuracy.

In other research [2], the authors described the important issues arising when developing an AR system in a localization application. Then, they introduced a system, called WorldPlus that presents solution to these problems.

A survey on AR, in which 3D virtual objects are integrated into a 3D real environment in real time is presented in [3]. It is a valuable research where the tradeoffs between optical and video blending approaches are discussed. In another survey paper [4], the authors described the field of AR and several recent AR applications with their limitations. Various challenges in providing location-based AR have been presented in [5], and then the engineering process for developing the core framework has been introduced.

3 Sensor-Based Augmented Reality Application Software (SARAS)

SARAS is built to be used in all kind of Android-based mobile devices. The use case diagram in Figure 1 explains the functioning of the application:

1. User wants to launch the application.
2. If the camera is working, if the device has Internet connection, and if the battery level is sufficient, the application is initialized.
3. Identity number information screen is shown. User can skip this screen or identify himself/herself to the system. If she/he identifies her/himself, she/he access to the program as a bank customer; otherwise as a general user, i.e., a potential customer.
4. User can filter POIs by their types or the distance.
5. POIs are tagged on the screen.
6. If the Internet connection does not exist or it is limited, user can choose to use the QR code of the POI for accessing campaign information.
7. User exits from the program.

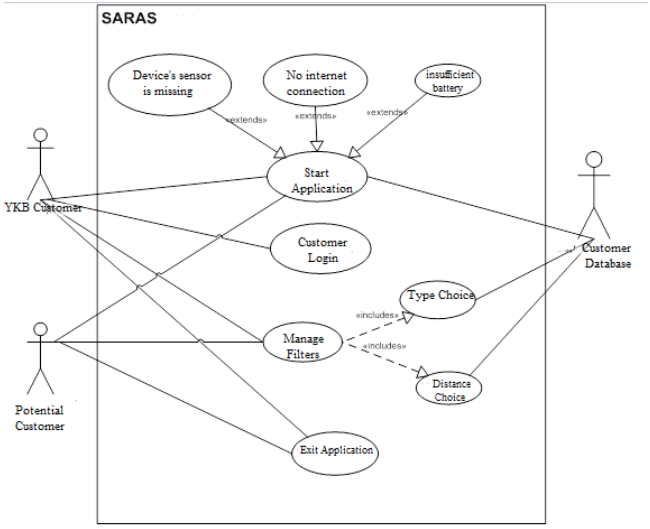

Fig. 1. Use case diagram of SARAS

4 Proposed Tagging Algorithm

The aim of the introduced responsive tagging algorithm is to position POIs on the right point on mobile device screen. The GPS coordinates of the POI is assumed to be known in advance.

4.1 Finding POI's Cardinal Direction

The algorithm considers all possible cardinal (North, South, East, West) and intercardinal (Northeast, Northwest, Southeast, Southwest) directions. It is assumed to exist 360 points that the device may head to. As there are eight different directions, the device is said to have a vision range of 45 points. The algorithm first tries to figure out in which interval of 45 points that the given POI is. In other words, the algorithm first finds the direction of the POI. Then, the tag for this POI needs to be positioned on the right place on the screen. The steps of the algorithm can be summarized as follows:

1. All tags are created to appear on the screen for each POI in respect to determined type and distance filter.
2. All POI tags are made invisible. (All of them are created, because it is faster than recreating them at each device movement.)
3. The difference between the POI's and device's latitude is calculated as *diffLatitude* and the difference between the POI's and device's longitude is calculated as *diffLongitude*.

4. These differences are compared. If (*diffLatitude* > *diffLongitude*), then the POI is defined in North or South. If (*diffLatitude* <= *diffLongitude*), then the POI is defined in East or West.

i.If (*diffLatitude* > *diffLongitude*): If (*POI's latitude* > *device's latitude*), then the POI is in North, because it is assumed to be in the North hemisphere and the latitude increases when going North; otherwise the POI is in South.

ii. If (*diffLatitude* <= *diffLongitude*): If (*POI's longitude* > *device's longitude*), then the POI is in East, because the longitude increase when going East; otherwise the POI is in West.

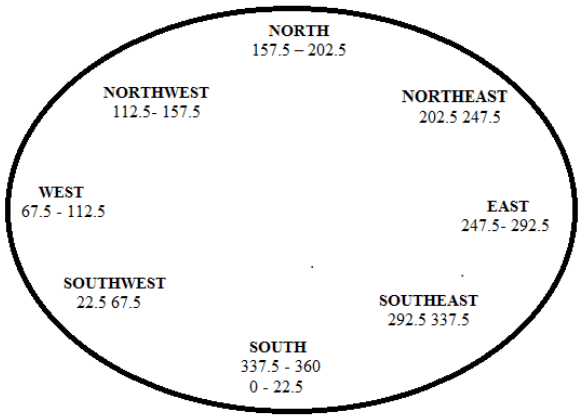

Fig. 2. Representation of directions

The mentioned 360 points and related directions are illustrated in Figure 2. Using the Azimuth range for directions, these 360 points are grouped as:

North: Interval is 157.5 – 202.5.
Northeast: Interval is 202.5 247.5.
Northwest: Interval is 112.5- 157.5.
South: Intervals are 337.5 - 360 and 0 - 22.5.
Southeast: Interval is 292.5 337.5.
Southwest: Interval is 22.5 67.5.
East: Interval is 247.5- 292.5.
West: Interval is 67.5 - 112.5.

5. If the POI is in North (Interval 157.5-202.5), then:

i. The middle point of North is 180 ((157.5+202.5)/2). Looking to the North direction, if the POI is in East (POI's *longitude* > *device's longitude*), then the POI is on the right side of the screen. So, this point should be between 180 and 202.5. Otherwise, if it is in West, it should be between 157.5 and 180.

ii. 22.5 is divided to the *diffLatitude* and it is multiplied with *diffLongitude*.

iii. If the POI is in East, 180 is added to the result of *ii*. Or if the POI is in West, 180 is subtracted from the result of *ii*.

6. If the POI is in **South (337.5-360 and 22.5-0)**, then:

i. The middle point of South is 0 or 360, since it has two intervals. Looking to the South direction, if the POI is in East (*POI's longitude* > *device's longitude*), then the POI is on the left side of the screen. So, this point should be between 337.5 and 360. Otherwise, it should be between 22.5 and 0.

ii. 22.5 is divided to the *diffLatitude* and it is multiplied with *diffLongitude*.

iii. If the POI is in East, the result of *ii* is subtracted from 360. Or if the POI is in West, the result of *ii* is added to 0.

7. If the POI is in **East (Interval 247.5-292.5)**, then:

i. The middle point of East is 270. Looking to the East direction, if the POI is in North (*POI's latitude* > *device's latitude*), then the POI is on the left side of the screen. So, this point should be between 247.5 and 270. Otherwise, it should be between 247.5 and 270.

ii. 22.5 is divided to the *diffLongitude* and it is multiplied with *diffLatitude*.

iii. If the POI is in North, the result of *ii* is subtracted from 270. Or if the POI is in South, the result of *ii* is added to 270.

8. If the POI is in **West (Interval 67.5-112.5)**, then:

i. The middle point of West is 90. Looking to the West direction, if the POI is in North (*POI's latitude* > *device's latitude*), then the POI is on the right side of the screen. So, this point should be between 90 and 112.5. Otherwise, it should be between 67.5 and 90.

ii. 22.5 is divided to the *diffLongitude* and the result is multiplied with *diffLatitude*.

iii. If the POI is in North, the result of *ii* is added to 90. Or if the POI is in South, the result of *ii* is subtracted from 90.

4.2 Finding POI's Cardinal Direction

9. If the *diffLatitude* is two times bigger than *diffLongitude* or vice versa, the POI is assumed to be in North, West, East or South. Otherwise, it is assumed to be in Northeast, Northwest, Southeast or Southwest.

4.3 Placing POI's Tag on the Screen

After determining the direction of the POI, the second step involves placing the POI's tag on the right point on the mobile device's screen. Therefore, the algorithm continues with the remaining steps:

10. Screen range is calculated using the mobile device's Azimuth value. This value is assumed to be the middle point of the screen. 22.5 is subtracted from this value for finding the starting point of the screen. Then, 22.5 is added to this value for calculating the endpoint of the screen. Therefore, the screen has the range of [Azimuth - 22.5 – Azimuth + 22.5].

11. If the device is pointed to the POI's direction, its interval calculated at step 10 is compared to the point calculated in one of the steps 5, 6, 7, 8 or 9. If the POI's point is inside this interval, then it becomes visible.

4.4 Calculating the Point's Position on X-Axis

11. Device's Azimuth value is interpreted as the middle of the screen (same as the step 10).

12. The Azimuth value of the first point of the screen is calculated as: device's calculated Azimuth value - 22.5, since each interval has 45 points.

13. The value calculated in step 12 is subtracted from the POI's point, that is calculated in one of the steps 5, 6, 7, 8 or 9.

14. The width of the screen is determined using Android's *getWidth*() function. Then, it is divided to 45, which is the screen range. Therefore, the portion per point is calculated. As the screen is considered having 45 points, each point should have a range of this calculated portion.

Portion * 45 = Device's screen width

15. The value that is calculated on the step 13 is multiplied with the one that is calculated on step 14.

16. The tag's size/2 is subtracted from the value that is calculated at the step 15.

An example:
Let us assume that the device's Azimuth value is 120 and *x* value of the POI is 135 and the size of the POI's button (*sizePOI*) is 20. Then:
Step 11: 120 is interpreted as the middle point of the screen.
Step 12: 120 – 22.5 = 97.5. This is the starting point that is visible on the screen. 120 + 22.5 = 142.5. This endpoint is visible on the screen.
Let us calculate the location of the tag of POI: The POI's location is assumed to be at 135, which is calculated in one of the steps 5, 6, 7, 8 or 9.
Step 11: 135 is between 97.5 and 142.5. Therefore, it is in the range of the screen. Its tag becomes visible.
Step 13: The POI's point is subtracted from the starting point of the screen: 135 – 97.5 = 37.5.
Step 14: getWidth() is 480. (480/45 = 10.6)
Step 15: 10.6 * 37.5 = 400.
Step 16: 20/2 = 10. 400 – 10 = 390.
Thus, the POI is determined to be at point 390 on X axis. (Note that the range of the screen is 0 - 480.)

4.5 Calculating the Point's Position on Y-Axis

The generation of the POI's position on the Y-axis is done using the pitch value. The tagging algorithm continues following the remaining steps:

17. If the device is directed up or down to the floor, nothing is listed on the screen.

18. If the device is not directed up or down to the floor, the screen is considered having 90 points from up to down. The screen's height is divided to 90 and the value of a portion per point is obtained.

19. The screen's height is divided in 2 for obtaining the value of the middle point of the screen.

20. The pitch value is multiplied with the value calculated on step 18.

21. The value obtained in step 19 is subtracted from the value calculated in step 20.

22. The half of the size of the tag is subtracted from the value calculated in step 21.

Finally, at the end of these 22 steps of the algorithm, the accurate coordinate of the POI's location on the screen is determined.

4.6 Handling Intersections on the Screen

In case multiple POIs are placed at the same point of the screen, they should be repositioned in order to remove these visual intersections. The related algorithm to find the POI's location on X-axis has the following steps:

1. All the available intervals are collected into a hash map.

An example:
It is assumed that we have [*XStartInterval* and *XEndInterval*] and the POI is at *xPOI* with the size of *sizePOI*.
- If (*XStartInterval* < *xPOI*) and (*xPOI* + *sizePOI* < *XEndInterval*), then the new interval becomes: [*XStartInterval* – *xPOI*] – [*xPOI* + sizePOI – *XendInterval*]
- If (*XStartInterval* > *xPOI*) and (*xPOI* + *sizePOI* > *XStartInterval*), then the new interval becomes: [*xPOI* + *sizePOI*] – [*XendInterval*]
- If (*xPOI* < *XEndInterval*) and (*xPOI* + *sizePOI* > *XEndInterval*), then the new interval becomes: [*xStartInterval* - *sizePOI*]

The similar rules are applied to find the location on the Y-axis:
4. This interval (calculated on step 1) is removed from the hash map. New intervals that are calculated on step 3 are added to the hash map.
5. If the POI's calculated position is occupied, the nearest interval is chosen.

Therefore, the POIs that are on the same point can be tagged on the screen as in Figure 3.

2. A suitable interval is determined for the given POI.

3. After placing a POI, this interval is changed by omitting this POI's occupied place. If it is in the middle of an interval, then two intervals are obtained.

5 Performance Analysis

5.1 Comparison with Other Similar Mobile Applications

The chosen applications are the most frequently used AR applications developed by professional software companies. The primary aim is to keep up with their performance level, and then go beyond it. Several differences and advantages of the introduced tagging algorithm are as follows:

- *GPS Libraries and Modes*: None of these commercial mobile applications mentioned offering user to choose a library and mode for fetching GPS information. They are important for energy consumption.
- *Map Support*: In general, this kind of AR applications offer either map support or screen tagging/placement support, but SARAS has both of them.
- *Sensor Calibration*: Sensors can produce faulty data from time to time, so a calibration feature is inserted to the application.
- *Compass*: A compass is available on the main screen.
- *Language Support*: None of these applications offers user the Turkish language selection.

As its source code was open and its features are similar, the proposed algorithm is compared to the one in [2]. This application implemented on Pro Android Augmented Reality [6], that is an AR world browser showing data from Wikipedia and Twitter. It has the following similar features:

- It has a live camera preview.
- Twitter posts and topics of Wikipedia that are located nearby are displayed over this preview.
- A small visible radar allows user to see whether any other overlays are available outside their field of view.
- Overlays are moved in and out of the view as the user moves and rotates.
- The user can set the radius of data collection from 0 m to 100 km.

The algorithm in [2] is implemented in SARAS and then the performance tests are generated.

5.2 Performance Analysis in terms of Resource Utilization

First, the energy efficiency of the algorithms are considered. The results show that the proposed tagging algorithm spends less energy than the given algorithm [2]. Table 1 summarizes the test results with different modes of GPS and different libraries. Best results are obtained in normal mode with Google Play Services library for GPS calculation and using WiFi for Internet connection.

TABLE I: COMPARISON RESULTS IN TERMS OF RESOURCE UTILIZATION DURING CONSUMPTION OF 20 MINUTES OF UNINTERRUPTED RUNNING

SARAS	Pro Android Augmented Reality [2]	
Energy	978 mW	978 mW
Battery	%13	%14
LCD(*)	896 J	892 J
CPU	267 J	265 J

(*) This depends on the number of merchant displayed on screen.

The results reveal that the proposed tagging algorithm in SARAS is more energy efficient than the given algorithm [2]. It consumes %1 less battery in 20 minutes. They both consumes similar amount of CPU power.

5.3 Performance Analysis in terms of Accuracy

In order to test the consistency and the accuracy of the algorithm, two test environments for three modes (when standing still, when walking and when driving) are used. The maximum distance for the map and for the distance filter is set to 10 km.

Fig. 4. Map representation of the test environment with 11 POIs

Accuracy while standing still

The mobile device was in hand when the tester was standing still. Figure 4 shows 11 POIs that are tagged on the screen. They are depicted in East and Northeast, when the mobile device is thought in the middle.

As seen in Figure 5, all of the five POIs in East are placed correctly on the screen; however the benchmarking algorithm [2] listed all the POIs in East, even though several of them were in Northeast.

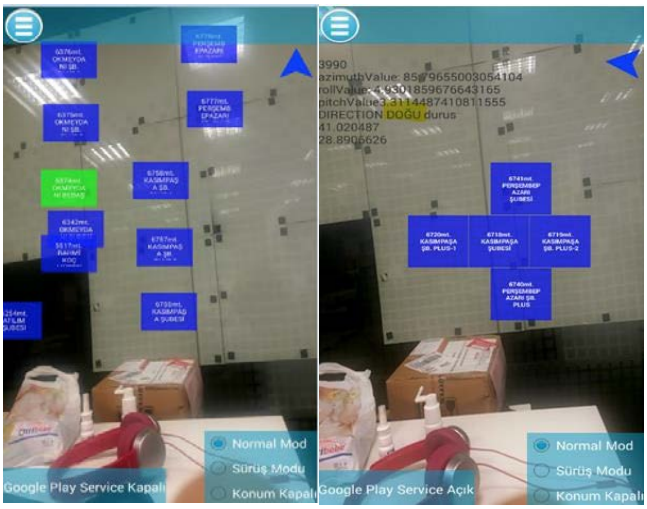

Fig. 5. Screenshot comparison of two algorithms - East

As a different test, the mobile device is turned from right (East) to left (Northeast). Using the proposed algorithm, all of the POIs are tagged on the screen, because it is like an intersection for all of them (Figure 6). Moreover, the POIs which are in East stays in the right side of the screen, while the POIs which stays in Northeast are placed on the left side of the screen. The benchmarking algorithm showed the same tags as in East (Figure 5). It tagged all of the POIs.

With the intention of testing the intercardinal direction handling, the mobile device is turned through Northeast. The resulting screenshots are given in Figure 7. Using the proposed algorithm, all the six POIs in Northeast are tagged properly. The benchmarking algorithm [2] tagged only one of the POIs which is on Northeast. It missed four others. Finally, it is concluded that the proposed algorithm produces more accurate and reliable tags compared to the benchmarking algorithm.

Accuracy while walking

The same tests are generated for the walking case. The outputs were reliable and fast. The application was reacting quickly and the outputs were as same as the test in standing still.

Accuracy when driving

The POI was the Çırağan Adalet Sarayı in İstanbul. The output screens of two algorithms are given in Figure 8 and Figure 9. Comparing these outputs, it can be concluded that both of these algorithms have similar outputs. Both of them placed Çırağan Adalet Sarayı at the correct position.

6 Conclusions

The aim of this research is to develop an algorithm for tagging POIs on mobile device's screen. An AR mobile application (called SARAS) is built. SARAS is compatible with Android OS. One of the biggest banks in Turkey (Yapı Kredi Bank-YKB) and Ministry of Industry and Sciences have supported this research. The POIs are chosen as the merchants, offices and the ATMs of the bank. Using this application, users can find the nearest bank and ATMs, along with their way towards them. Besides, bank may use this application as a new channel to inform their potential users on related campaigns.

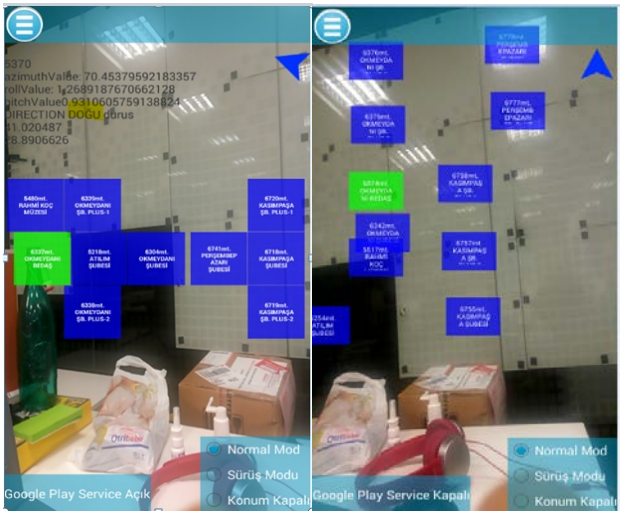

Fig. 6. Screenshot comparison of two algorithms – turning from East to Northeast

Since only the benchmarking algorithm [2] has an open source algorithm, the performance comparisons are done using it. Both algorithms spend almost the same amount of energy. The biggest source of energy consumption of AR applications are the GPS and camera usage. LCD usage depends on the number of POIs that are displayed on the screen, therefore it can be concluded that both algorithms spend almost the same amount. The CPU usage of these two algorithms are at similar ranges, but the proposed algorithm consumes less battery.

The proposed algorithm places accurate tags while standing still and walking. When driving, it

places the tags at accurate locations on the screen in each time, however its response time to position changes need to be improved.

The most significant contribution of this algorithm is its simplicity. It is developed for the countries in North hemisphere, but it is easy to convert it for the South hemisphere. Going further, the performance of the algorithm will be improved to more rapidly react to position changes while driving.

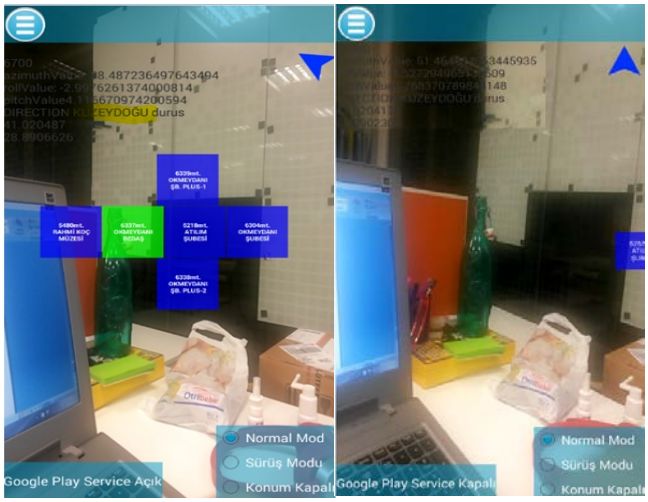

Fig. 7. Screenshot comparison of two algorithms – Northeast

Acknowledgment

This research has been financially supported by Galatasaray University Research Fund, with the project number 15.401.006.

Fig. 8. Screenshot comparison of two algorithms – Driving – 1

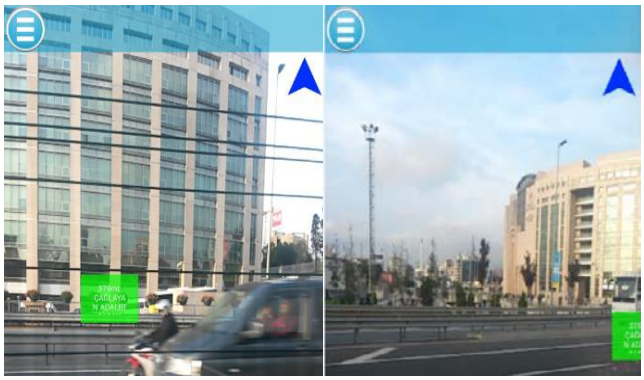

Fig. 9. Screenshot comparison of two algorithms –Driving - 2.

References:

[1] R. Calvo-Palomino, "Mobile Augmented Reality browsers should allow labeling objects". Proc. Mobile AR Summit (MWC 2010), 2010.

[2] S. Graça, J. F. Oliveira, and V. Realinho, "WorldPlus: An Augmented Reality Application with Georeferenced Content for Smartphones - the Android Example," Proc. 20th WSCG International Confrence on Computer Graphics and Computer Vision, 2012.

[3] R.T. Azuma, "A Survey of Augmented Reality," Presence Teleoperators and Virtual Environments vol. 6 (4), 1997, MIT Press, pp. 355 - 385.

[4] D.W.F. van Krevelen, and R. Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations," International Journal of Virtual Reality, vol. 9 (2), 2010, pp.1

[5] P. Geiger, R. Pryss, M. Schickler, and M. Reichert, "Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices," Technical Report, University of Ulm, 2013.

[6] Raghav Sood, Pro Android Augmented Reality, 2012, APress.