ACCURATE LOCATION DETECTION SYSTEM FOR AUGMENTED REALITY APPLICATION

Dewagirige Chamila Kanchana Rathnayake

(168262X)

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March 2020

ACCURATE LOCATION DETECTION SYSTEM FOR AUGMENTED REALITY APPLICATION

Dewagirige Chamila Kanchana Rathnayake

(168262X)

M.Sc. in Computer Science

This dissertation submitted in partial fulfillment of the requirements for the Degree of MSc in Computer Science specializing in Software Architecture

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March 2020

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for degree or diploma in any other university or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to university of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: ……………… date: ……………….

Name: Dewagirige Chamila Kanchana Rathnayake.

The supervisor/s should certify the thesis/dissertation with the following declaration.

I certify that the declaration above by the candidate is true to the best of my knowledge and that this report is acceptable for evaluation for the draft MSc Thesis.

Signature of the supervisor: …………………………. Date: ………………..

Name: Dr. Indika perera

ACKNOWLEDGEMENTS

I would like to express gratitude to my supervisor, Dr. Indika Perera, for his invaluable support by providing relevant knowledge, materials, advice, supervision and useful suggestions throughout this research work. His expertise and continuous guidance enabled me to complete my work successfully. Further I would like to thank Amodth Jayawardena for providing valuable resources and advise in the area of this research.

Finally, I would like to thank the academic and non-academic staff of Department of Computer Science and Engineering of the University of Moratuwa.

ABSTRACT

The proposed research has been done as an extended version of the research named "A Framework for Mixed Reality Application Development: A Case Study on Yapahuwa Archaeological Site" which was done by Amodth Jayawardena. The above research implemented a proper framework for developing augmented reality applications. The researcher has planned to implement a working model of the above project in Yapuhuwa Archaeological site. However, still, there is a problem in implementing this project in real-life situations. That is because of non-accurate location data. Although the technology today has improved, GPS technology still doesn't accurately offer the exact location data. Therefore, when a user tries to use the above-mentioned AR application on his/her mobile, the virtual graphics generated for that location may get inaccurate because of the slight GPS location data fluctuations. However, we can improve the accuracy of the inbuilt GPS data of a mobile phone using third party devices, which are expensive. The proposed solution implemented in this is based on giving a solution to find the most accurate geographic locations for augmented reality applications. Since improving location data accuracy using third party devices is not practical in an implementation environment like Yapahuwa this research paper focused on getting the exact location using image processing technology and mobile phone's sensor data (when needed). An image file system is maintained separately for implementing the system. As an example, when the research is implemented in the real world it can identify and generate the graphic image to the relevant location when a user aims his/her mobile phone to an archaeological place. The technology used in the research can be implemented in many other industries for purposes like Google Maps in the future.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
GPS	Global Positioning System
CPU	Central Processing Unit
VT	Vocabulary Tree
DB	Database
SIFT	Scale-invariant feature transform
AR	Augmented Reality
ORB	Oriented FAST and Rotated BRIEF Algorithm
SURF	Speeded up Robust Features Algorithm
FAST	Features from Accelerated and Segments Test

CHAPTER 1

INTRODUCTION

During the past recent years, mobile phones have been developed rapidly with the latest technologies including GPS, smart cameras, high power CPUs, wireless networks and gravity sensing units with fewer costs. Users of these mobile phones can take pictures, record videos and publish them in anywhere they want. As a result, the mobile phone has become a day to day using device among the world population. Due to its popularity more and more mobile applications have been developed for mobile phones since the beginning of the smartphone era. Due to the high computational power of mobile phones complex high CPU cost applications have been developed. Above mentioned "Framework for Mixed Reality Application Development" is a good example.

1.1 Approaching to the Research on Place Detection

The research "A Framework for Mixed Reality Application Development: A Case Study on Yapahuwa Archaeological Site" is an open framework to develop any kind of mixed reality application easily. The case study was carried out based on the research carried out for the Yapahuwa archaeological site in Sri Lanka. However, there was an issue implementing this solution in the real world due to the inaccuracy of location data. In detail, when a user tries to aim his/her smartphone to an archaeological location current software generates the graphic images for that location mainly based on the GPS data. However, the GPS data captured by the mobile phone is unable to measure the accurate GPS location. It gives 12 meters to 7 meters inaccuracy with exact geolocation which is not helpful to run the mixed reality application mentioned above. To minimize the inaccuracy third party device can be used which is very expensive and not practical for a public place like Yapahuwa. Therefore as a solution, getting the most accurate geolocation with a combination of visual data and mobile phone's sensor data has been proposed in this research paper. Visual data is processed using image recognition technology.

1.2 Geolocation Detection

Geolocation detection can be done using GPS technology at present. However, GPS is unable to produce the pinpoint location of a user since the accuracy of GPS data limits to near 4 meters RMS. Augmented reality applications require to pinpoint accuracy of location data. Otherwise, the AR application cannot generate graphic details accordingly.

In solving the above issue, proposing a direct urban and non-urban mobile visual recognition system for a smartphone was suggested. First, an algorithm was selected and developed to the images with unique features. Then, another algorithm is developed to maintain the user location at the events where there are no unique images available.

1.3 Problem

The main research problem tried to address through was to develop a mobile application which is capable of identifying most accurate geolocations based on the visual and mobile sensor data (if needed) for a mixed reality application to improve its accuracy. The solution is capable of run in the background of the main application as well as should not affect the user experience of the mixed virtual reality application. Therefore the development of a cost-effective (less resource consuming), less time consuming and high performance (detailed explanations [16]) software architecture & a proper mechanism was needed to successfully implement the project which needed a good technical and architectural solution.

To come up with a reliable solution it was required to explore many research areas under this research study. The core of this research was based on image detection used for location identification using less amount of computational power and response time. These important research areas were explored and discussed in the literature review section separately.

1.4 Objective

The main objectives of the research are

1. To track the accurate geolocation based on the visual data and inbuilt sensor data in the mobile phone.
2. Solution (1) should be able to successfully implement in a selected area (Ex: Yapahuwa archaeological site).
3. Avoid using external devices (connected to the device which contains the AR application) to measure accurate geo-locations.
4. Provide a framework to run the solution (1) in average or low specification mobile phones (The term 'average and low specification mobile phones' represents the mobile phone available as at 31st March 2020).

CHAPTER 2

LITERATURE REVIEW

2.1 A Brief Introduction to Research Studies Carried Out

There are many types of research done in the field of image recognition in indoor and outdoor [17]. However, there aren't many researches done in the field of image recognition for a large area (scenery detection) based on mobile phones. Most of the researches are done for the robotic industry and ordinary desktop computers. Several types of research carried out in the field of image recognition which I referred are mentioned in below sections under this topic.

Mobile augmented reality became more popular once the leading mobile developer Apple launched a new framework called ARKit. It has motion tracking in 6 degree of freedom which runs IOS operating system (Especially mobile devices like iPhone & IPads). Google has its version of Augmented Reality libraries called monicker ARCore for a limited number of Android devices. After that, the count of applications which use location tracking with 3D rendering appears in the PlayStore increased.

Currently, users can use augmented reality for

- Visualization - e.g. visualizing how a new curtain looks like in your room
- Contextual information - e.g. visualizing instructions on top of something for explanation purposes.
- Immersive experiences - e.g. walking with visualized animals like Dinosaurs etc. in a live environment.
- Natural interfaces - e.g. edit something in the real world at the same time the user is viewing the object through the application.

There are guided tours for museums with augmented reality using Google Tango devices which are available in the front desk of the museums. User can pick up a device and walk around the museum. Although the expectation is a user will carry out these devices at least 20 minutes, but in reality, these devices were difficult to carry and people leave them on the table instead of carrying them.

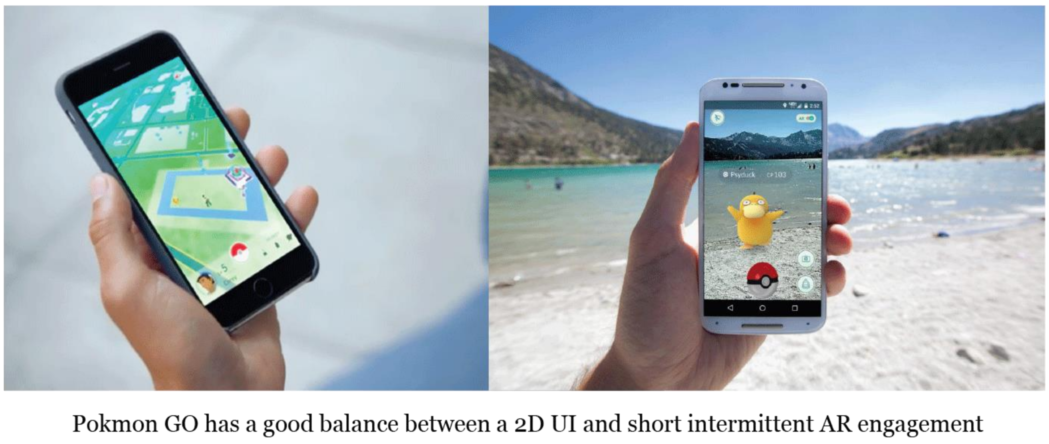

Pokmon GO has a good balance between a 2D UI and short intermittent AR engagement

Figure 1 – Pokmon Go

Figure 2 - Project Tango

The key factor to take into account is when you use a mobile phone to look AR content, you will see the content in limited screen size. As a result, users who use the application will not get their expected satisfaction level. As an example imagine that you are viewing a huge hippopotamus through your phone. It will be very difficult to see a small portion of the content to the users who are using it.

This applies to any user who is holding the phone to his/her eye level. In most cases, users tend to hold their phone with one hand. In such cases generating graphic contents for different angles tend to be very difficult and users will not like it.

.

Augmented reality applications can be divided into below categories depending on what they augment.

- Augmenting nothing,
- Object Augmenting or
- Location Augmenting

Location-based applications are not new. Foursquare and Google Maps [26/27] are all fall under that category. It is true GPS is not sufficient to measure the required measurements in these applications due to lack of accuracy in limited area movements. Wi-Fi and Bluetooth beacons help develop indoor navigation applications. (Example: Pokémon Go). However, those solutions require external device supporting [25/36].

Figure 3 - Pokemon GO Augmented Reality

Figure 4 - Google Street View

There are two types of Augmented Reality Applications available.

1. Introducing Map to the Application
2. Introducing Application to the Map

Introducing map to the application

Location is different for each environment. Applications like above need customized maps for each location to do the tasks. As an example matching household items like furniture in an organization or house will require a customized map of that particular location. Therefore each location like above will require a customized map for each of them. Users could be able to build these maps. Benefits of having these applications (Basically two)–

- We can save locations relevant to physical space.
- Content positions can be aligned for various viewers to create the illusion of VR experiences.

Introducing Application to the Map

The museum tour guide, shopping complex navigation can be taken as examples. In the above situation, the content is rendered clearly for a specific map. Pokémon GO, Yelp, Foursquare etc are also in the same category.

Location detection in here works as a two-part system, first has an external map with referencing and the second has a sensor which is internal which is capable of saving data relevant to the map. GPS can be taken as an example. The external reference here is geostationary satellites and the internal sensor is a GPS receiver that triangulates its position.

In augmented reality, there are 4 location detection systems available.

GPS Alternatives

WiFi Triangulation and beacons are GPS alternatives which we can use for indoor navigation. For this system, the external party is a collection of beacons set up inside an indoor space. The receiving sensor is the client's mobile phone. However, this particular system has some drawbacks. They are additional hardware requirements, installation costs and the accuracy of location data is low and maintaining costs.

As an example image that you walked into your hotel room and turned on your augmented reality application. The application should be able to show you where your towels, plates, tea bags, key holders located etc. Doing such a thing needs to know the exact position and orientation down to the millimetre. Such kind of thing is only possible in today's world by using a technology like SLAM (virtual mapping and localization system).

Figure 5- Bluetooth Beacon

Recognition of outdoor scenes on a metropolitan level has a major relationship with image data retrieval or object recognition problems with big scale. A public schema used is based on the bagof-features model, which will extract features from local images, measuring its descriptors for visual words and applying text search methods for image retrieval.

Improvement of retrieval accuracy of this has been done by several researchers in different ways. Examples

- Introducing Progressive Sampling Consensus [1]
- Presenting efficiency levels to the initial recovery results as queries [2]
- Local features are assigned to the descriptor and spatial domains implementing weak geometry constraints and contextual weighting. [3,4,5]

On the other hand, researchers have used GPS and other sensors to improve the accuracy of visual recognition. In brief "outdoors augmented reality on mobile phone using loxel-based visual feature organization" [6] uses GPS data to get the images coming from adjacent locations. The research paper named "Experiments on visual loop closing using vocabulary trees" [7] uses GPS data to improve performance of the image scenery searching process. They have achieved it by ignoring the borderline of two sections.

2.2 Visual Object Recognition

The main area related to this project is, object recognition in an environment based on the image data regardless of indoor or outdoor. Based on the object data collected in a given environment an algorithm will finally give the expected results. There are several types of research carried out in the field of visual object recognition which I have referred listed below in detail.

Visual mapping and localization system can be used to parse visual imagery and locate position down to the centimetre. One method of doing implementing this is to use custom marker images or specific code signs. Those markers will tell you the position if you map them with exact locations.

Cameras in devices like Google Tango, Hololens enable marker-less mapping and localization system where the devices will save 3D maps of spaces and re-localize itself.

Currently, there are already build frameworks for real-time object detection. Tensorflow, OpenCV algorithms, YOLO are few examples.

The framework named YOLO (You Only Look Once) is the currently the fastest framework available for real-time feature detection. When considering about a Titan X graphics card, it can processes image data 40 to 90 frames per second and has an mAP on VOC 2007 of 78.6% and an mAP of 48.1% on COCO test-dev.

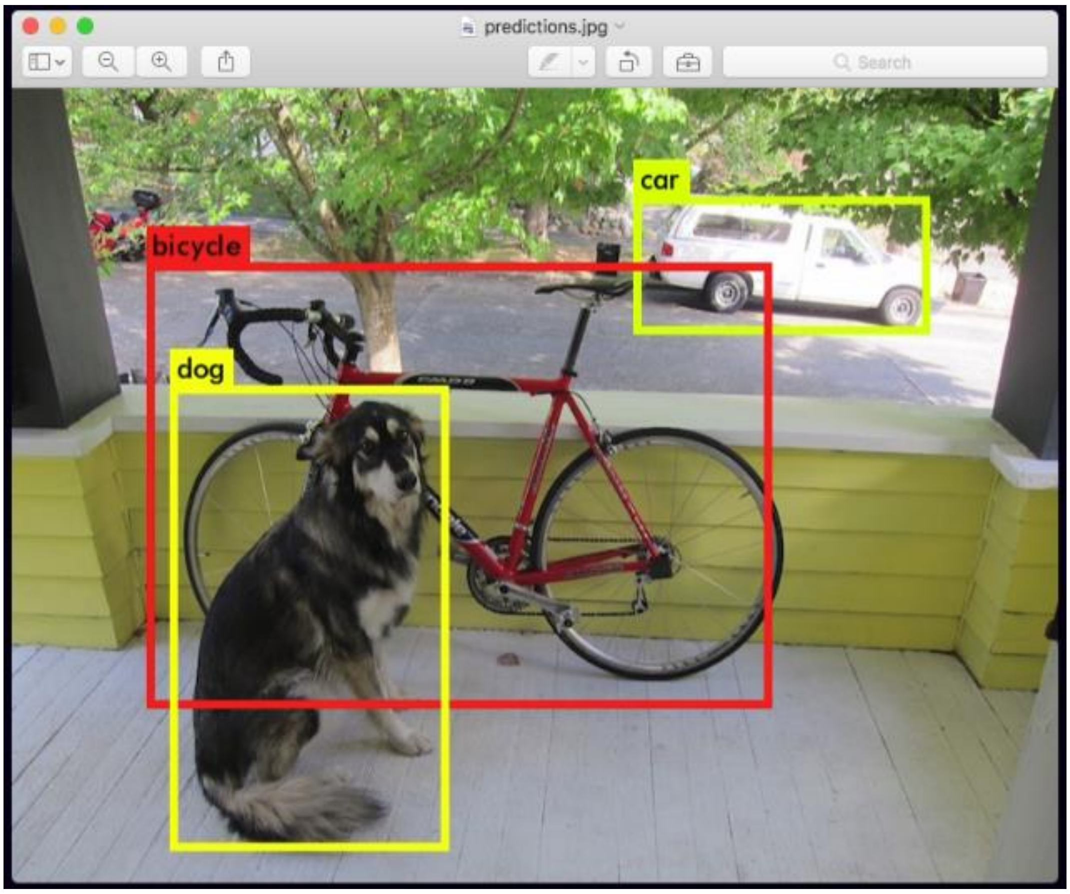


Figure 6 - Identifying objects using YOLO Framework

In previous, classifiers and localizers were repurposed by the detection systems. At multiple image locations and image scales, the models were applied. Areas which contain high scores are treated as detections.

However, in YOLO it follows a different approach. YOLO applies one neural network to the entire image itself. After that, the network will divide the image into sections and calculate probabilities for each section separately. These sections are called bounding boxes and they are weighted by the probabilities predicted.

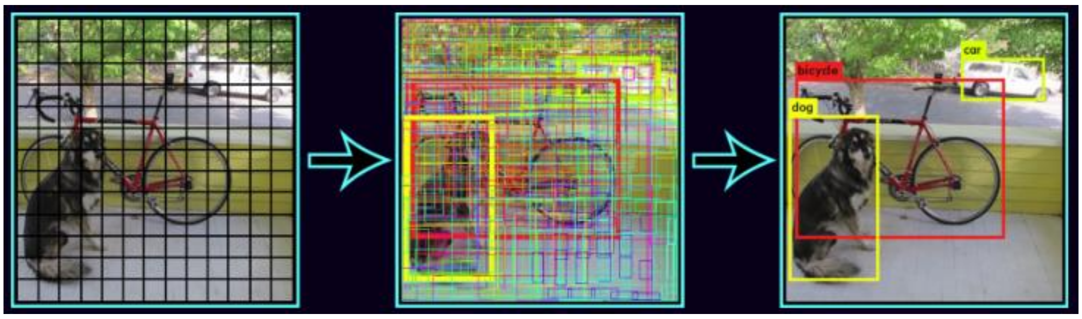


Figure 7 - How YOLO Identifies an Object

There are a few advantages in YOLO when compare to classifier-based systems. YOLO predictions are based on the overall context of the image. Unlike R-CNN which utilizes thousands for one image, it uses a single network evaluation which makes YOLO very fast in performance. It is estimated that YOLO is 1000 times faster than R-CNN.

2.2.1 Visual Object Recognition in the Context of Mobile Vision Services

Due to the improvement of technology in the field of mobile cameras and related equipment access to digital visual information has become popular. Nowadays most sophisticated cameras are being sold with smartphones. Analyzing these images for environment and object awareness, locating, examination, and explanation in general. The paper [8] provides a common technology for the identification of metropolitan (example: objects in cities) objects. When a user directs his/her camera to a location it will receive annotation about location significance and the identity of the building. Metropolitan recognition has been achieved by definite detection of architecture from line-based features proposed by "Consistent line clusters for building recognition in CBIR" [9]. The research paper named "Modelling and interpretation of architecture from several images" [10] suggests a framework for recovering structures that in the area of building recognition. The research [8] proposes a solution which is accurate and reliable with a working service of recognition, presenting more information on performance evaluation, both on smart mobile imaging technology and database. MAP decision making will be grounded by the recognition system where weak object hypotheses responses in the mobile image takings. The paper [8] provides an improved version over the standard SIFT key detector by selecting only informative (i-SIFTs) (more details [20]) keys for descriptor matching. The selection is chosen first to decrease the complication of the model and then to quicken discovery based on a selectable consideration. They have a decision tree which is trained to effectively find the SIFT's later entropy value, holding only the keys for detailed analysis and polling with high information content. The trials were done on low quality mobile phone imaging on metropolitan sites under different conditions. An alternative to SIFT can be found on [20] named ORB. A vision-based localized mobile robot is discussed in paper [29] which is capable of using image features which are scale-invariant as the original state of landmarks in non-modified environments. As a result robot localization and map building will be suitable because of the invariance of image translation, rotation and scaling. With their vision system, the landmarks will be localized and estimated by minimizing of least number of landmark squares. Some experiments indicate the above-mentioned landmarks are strongly matched and as a result, a three-dimensional map can be built from it. They always carry out image analysis since the features of the image are not free of noise. Kalman filters were used to detect the landmarks in a changing environment where database map with landmarks should be available. The paper [30] introduces an 'approach combining visual-based simultaneous localization and mapping (V-SLAM) and global positioning system (GPS) correction for accurate multi-sensor localization of an outdoor mobile robot in geo-referenced maps. The proposed framework combines two extended Kalman filters (EKF); the first one, referred to as the integration filter, is dedicated to the improvement of the GPS localization based on data from an inertial navigation system and wheels' encoders. Secondly, the EKF tools the V-SLAM process. GPS/INS/Encoders integration filter gives linear and angular velocities in V-SLAM EKF filter (dynamic). Changes in the dynamics in the integration filter and therefore the geo-referenced localization is determined by the output of the V-SLAM EKF filter. Therefore whenever there is a GPS connection unavailability, the solution increases the accuracy and the robustness of the positioning. This also allowed SLAM in a less featured environment.

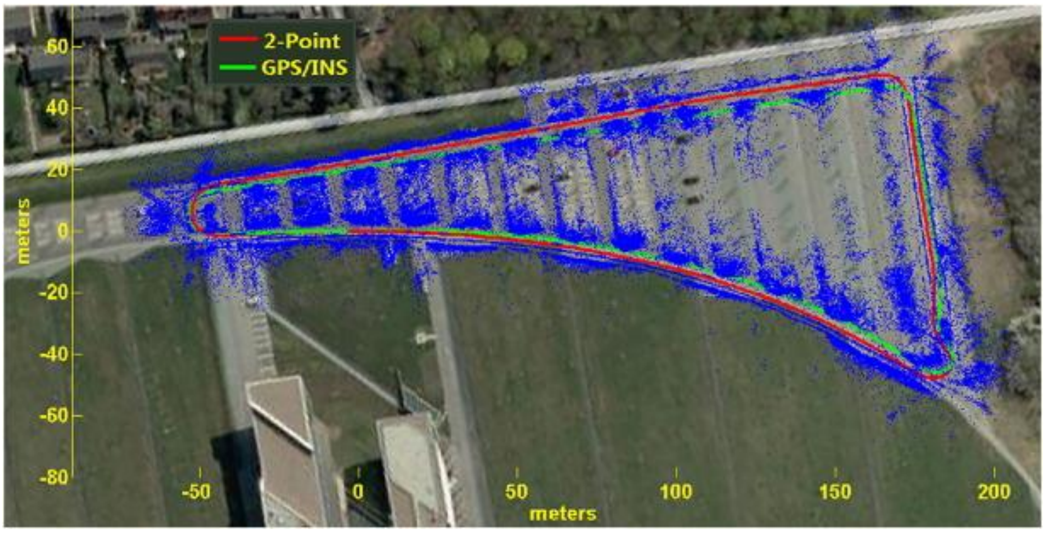


Figure 8 - 3D visual perception for self-driving cars

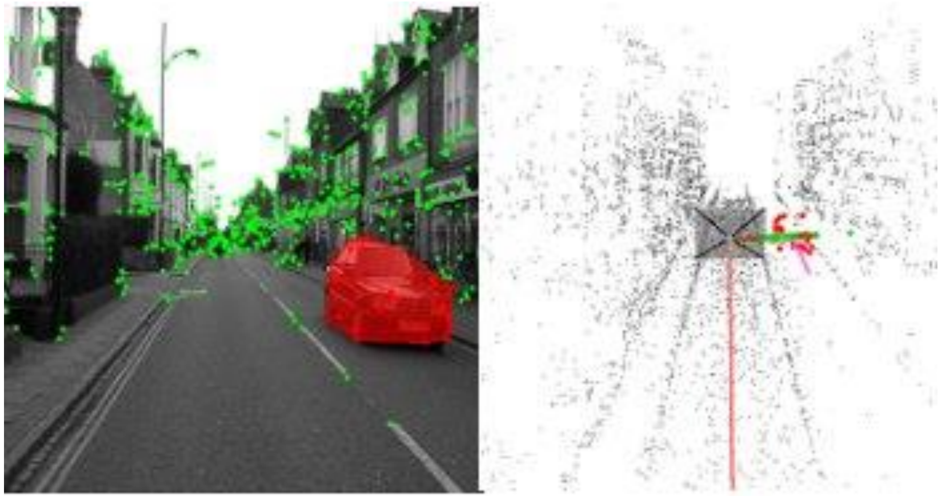


Figure 9 - Real time edge detection

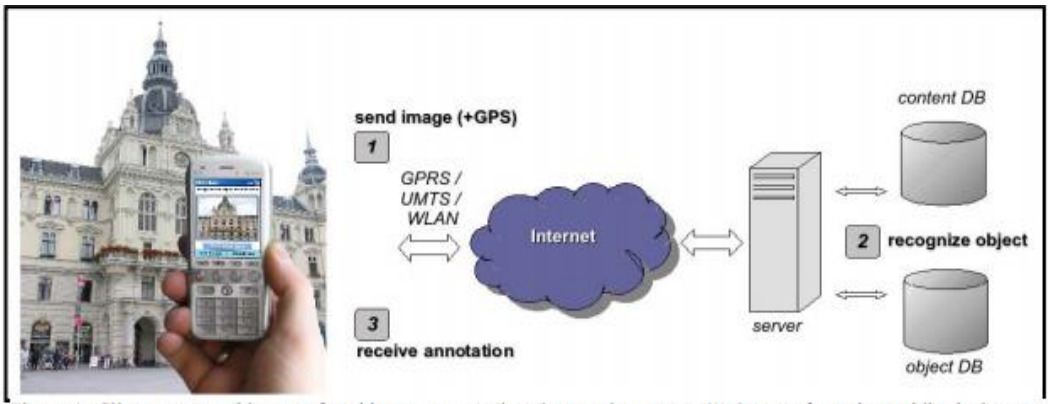


Figure 1: Client-server architecture for object awareness in urban environments. (1) Images from the mobile devices are transferred to the server for (2) recognition and content information. (3) User receives annotation about object and locations.

Figure 10 - Client-server architecture for object awareness.

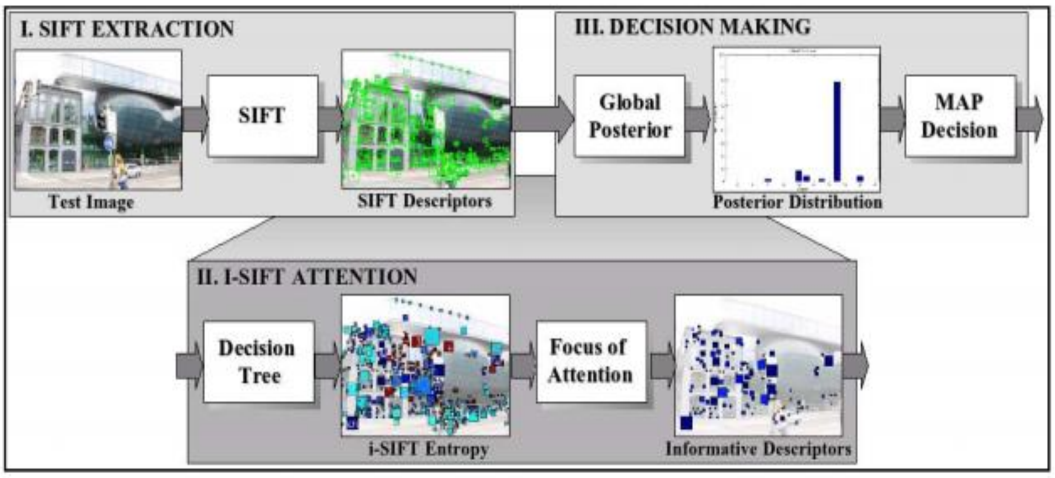


Figure 11 - Concept for recognition from informative local descriptors.

Figure 12 - related to [8]

2.2.2 Object Recognition from Natural Features on a Mobile Phone

The research paper [11] was introduced in 2008 and although it is a little bit of old it introduces an algorithm which allows recognizing hundreds of objects on an image using a mobile phone. They have used FAST corner detection technology by stripping down SIFT descriptors and a vocabulary tree combined with brute-force matching. The solution can process up to ten images per second on an Asus P535. The algorithm is used to implement a basic prototype to evaluate the implications of marker-less image recognition on a mobile phone. The algorithm introduced can be fine-tuned furthermore to improve performance and efficiency. To be specific, brute-force matching can be substituted by more advanced mechanisms. In future, more advanced studies should be done to get a better understanding of the consequences that accompany marker-less object recognition. This kind of study is quite necessary if 3D objects are involved in object recognition rather than 2D objects. More details under this topic can be found in [21].

Figure 13 - User interacting with an interactive Poster.

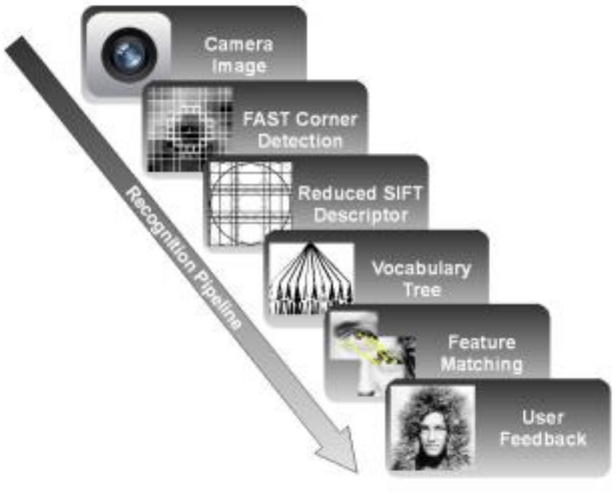


Figure 14 - Overview of the recognition pipeline

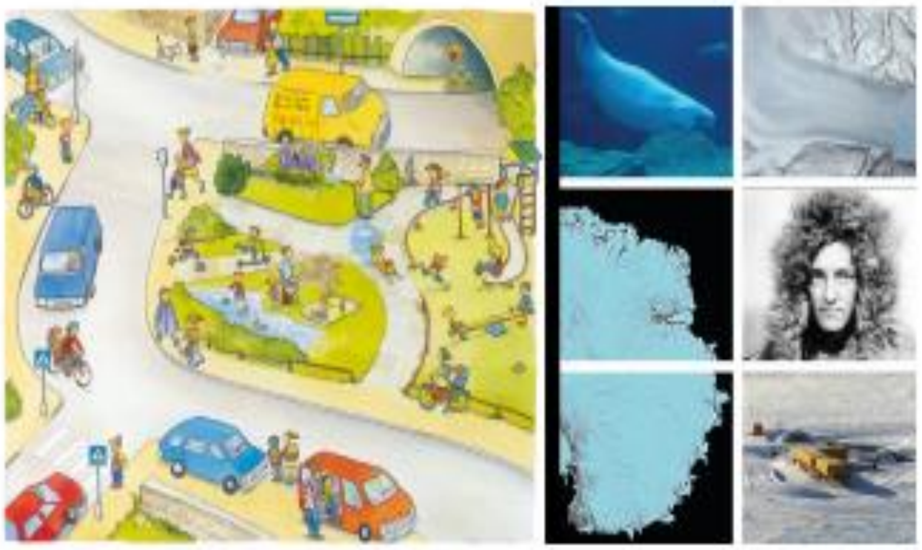


Figure 15 - The poster in first task evaluation (left).

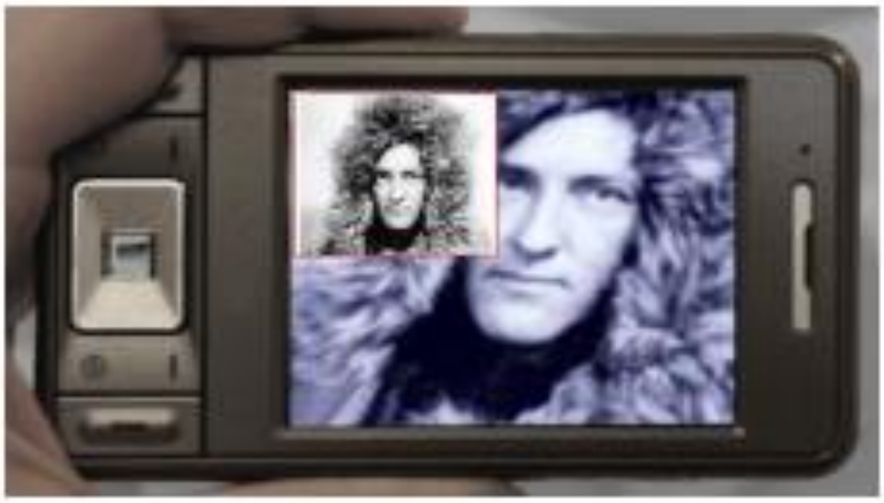
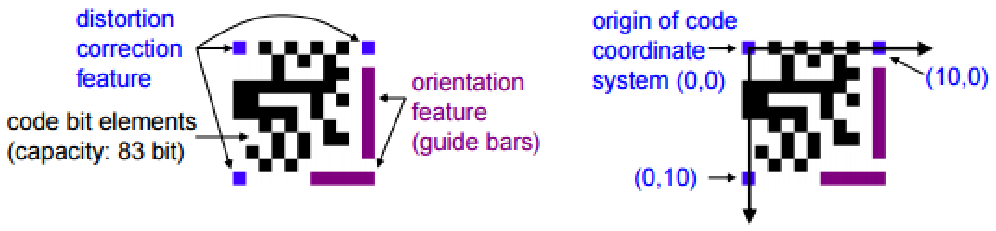


Figure 16 - ASUS smartphone running the used prototype

2.2.3 Using Camera-Equipped Mobile Phones

This research paper was not written in an era where sophisticated smartphones have been made. The suggested method will use mobile phone cameras as sensors which can be used for 2-dimensional visual codes. Codes can be printed on anywhere including electronic screens, papers or attached to physical objects and act as a key to access object-related information and functionality. The solution provided in this paper [12] is designed for the low image quality data which are produced by old mobile cameras. When compared to similar implementations, the recognition algorithm concurrently monitors multiple codes, decides the direction of the codes and computes the coordinates of the target.



distortion correction feature
code bit elements (capacity: 83 bit)
orientation feature (guide bars)
origin of code coordinate system (0,0)
(10,0)
(0,10)

Figure 17 - Components of the visual code and code coordinate system.

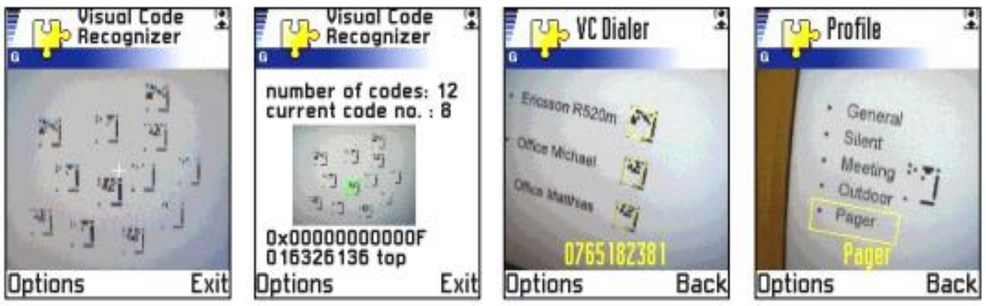


Figure 18 - Application screenshots: Visual Code Recognizer, Dialer, and Profile

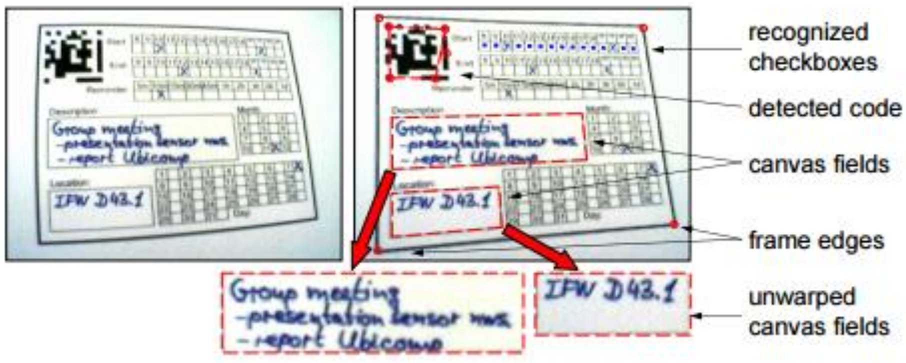


Figure 19 - code, frame, checkboxes, and canvas fields recognized.

2.2.4 Server-Client Object Recognition for AR

This research paper mainly focuses on how to use real-time visual data recognition for Augmented Reality (AR) applications. In this research paper, they have proposed a hybrid model, which delegates the object recognition task to a server and carries out tracking on the client-side. They have stated that similar technology is already implemented in Google Goggles projects however it is not a real-time image processing application.

Advantages of the proposed system are

- The application allows retrieving image from large databases nearby without keeping the bulky databases on the client side.
- The application is capable of allowing user-friendly interactive usage to the client with sending single images on a button click.
- While preserving the interactivity of the application, it also limits the number of communications done with the server.

According to the researchers below are the main contributions done by this paper [13].

- Tracking client-side objects and send them to server-side for recognition (efficient way).
- Includes a server-side database which contains millions of objects.
- Giving memory-efficient way to retrieve large scale objects and methods to facilitate near real-time AR requirements.
- Test client side tracking based on integrating sensor data.
- Deploy a fully functioning system in the Android platform.

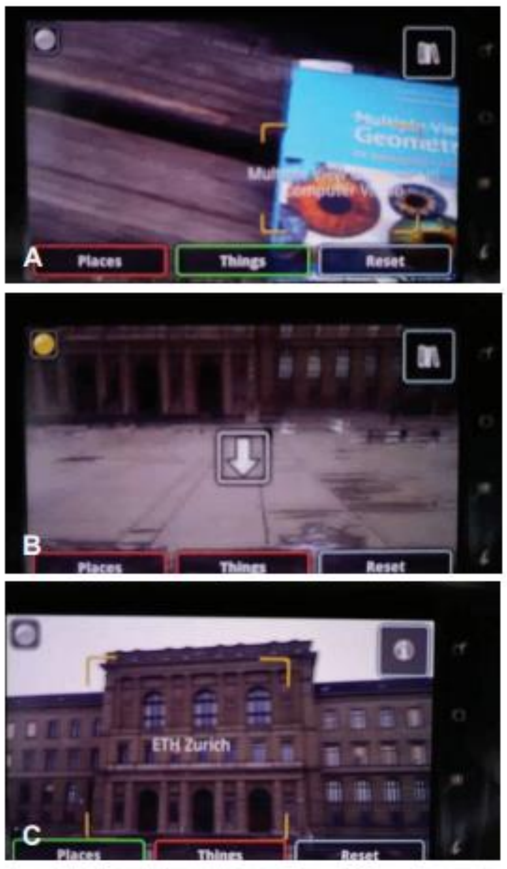


Figure 20 - Screenshot of mobile augmented reality application.

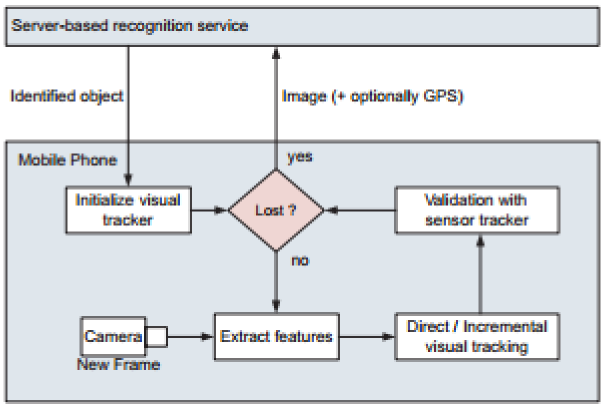

Server-based recognition service
Identified object
Image (+ optionally GPS)
Mobile Phone
yes
Initialize visual tracker
Lost ?
Validation with sensor tracker
no
Camera
New Frame
Extract features
Direct / Incremental visual tracking

Figure 21 - The mobile phone sends recognition requests to an external service.

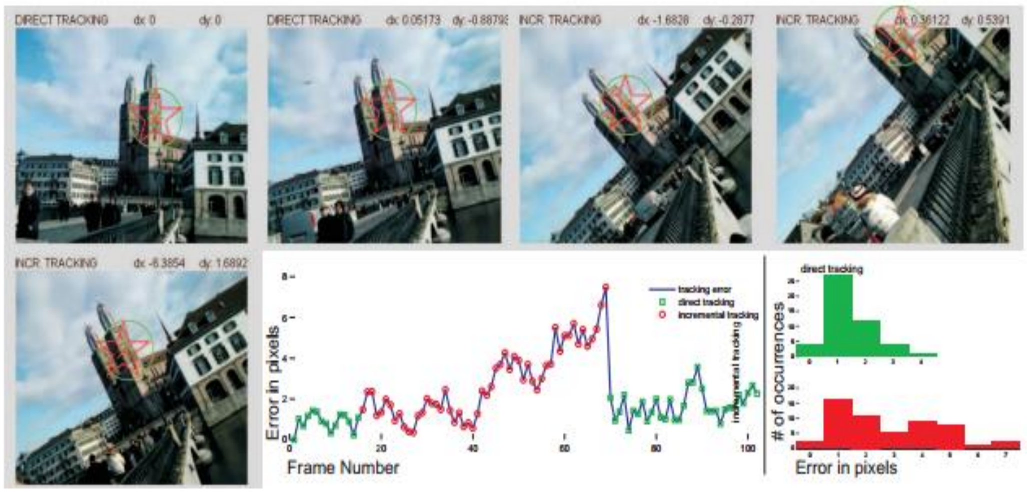


Figure 22 - Top: Manually annotated test sequence.

2.2.5 Various Building Detection Using Lidar Data

Detecting the constructions such as buildings in a difficult task in photogrammetry and computer vision. Because detecting them are very crucial in most of the applications nowadays. Aviation, construction, telecommunication, park management, military applications, flight simulators, drone operations, satellite imagery analysis, urban development planning and many more fields require recognizing and require building objects. As an example telecom industry need to plan their wireless connection lines. Urban planning and government offices need building models to impose taxes and help with their law investigations. Visualization and navigation also can be taken as similar applications as above mentioned.

Currently, there are not any up-to-date and accurate building data available. Therefore a high amount of automation is needed for generating and updating accurate and complete building data.

As input data, we use both digital images and LIDAR data. The proposed methods, through simplifications and slight modifications, could be employed using the only image or LIDAR data. However, researchers believe that these data are partly complementary and should be used in combination. E.g. digital images offer better image quality and spectral information and better edge (often surface discontinuity) definition, while LIDAR points have less and smaller gross errors, can provide a DTM everywhere (including the trees), provide multiple echoes per pulse and thus can also model the vertical structure of objects, e.g. trees. In the proposed methods, both DSMs and DTMs are used. These can come from LIDAR data or DSMs can be generated by image matching and then using DSM filtering, as for the classification of LIDAR points, a DTM can be derived. However, such a DTM will be inferior to a LIDAR DTM in case of trees, especially when covering a large area.

For methods that use a single image for detection, purposes can be good to detect simple shapes. But when it comes to complex shapes the results will get limited. Obstructions and shadows in the sceneries will increase the error rate if we use one image. Therefore in general, those methods will identify very simple shapes like flat rooftops. Therefore the user interaction will be required for accurate results. For the methods that use stereo or multi images, generally, there are several parameters, which have to be tuned for the processing, or some manual interactions are needed to get robust results. The results are relatively accurate, especially for simple buildings. Since surface discontinuities in an image-based DSM are not so sharp with older matching methods, complex-shaped buildings are difficult to extract, so usage of additional LIDAR data can lead to better results. In LIDAR-based methods, the results are mainly depending on the density of points. The cost of LIDAR data is generally lower than image-based DSMs, while the overall processing is faster. The capability of LIDAR data for the detection of surface discontinuities is weaker compared to edges extracted from images [19]. In general, it is difficult to detect accurate building models with low-density LIDAR data. LIDAR and photogrammetry are considered as complementary to each other by [8]. The integration of both technologies is believed to lead to more accurate and complete products [8]. For example, image data have advantages by providing spectral 21, 2(2014), 341-349 341 Various building detection methods with the use of image and LIDAR data N. Demir information, and LIDAR data have strengths to extract features using geometrical information and provide information on the vertical structure of semi-transparent objects like trees. There are several methods which use both LIDAR and image data ([9, 10, 11, 20]), and the research on the combination of both datasets is still a hot topic while existing approaches have some limitations in the full exploitation and combination of these datasets and their partial results.'

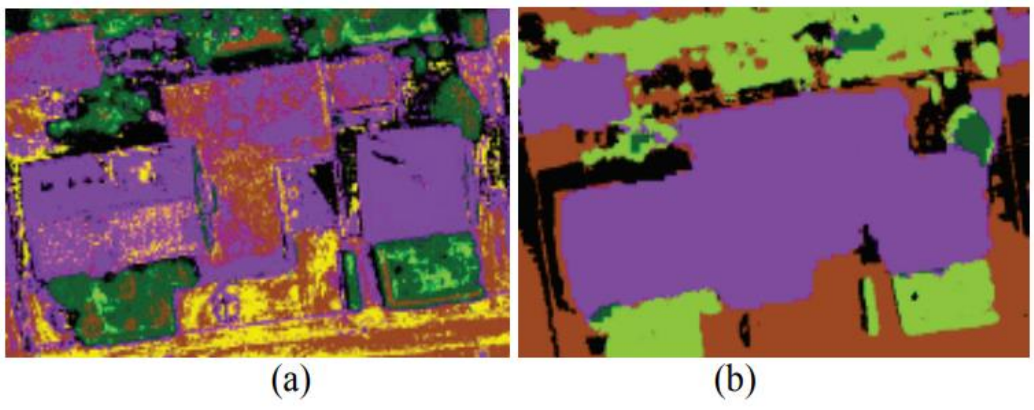


Figure 23 - Refinement of the classification result

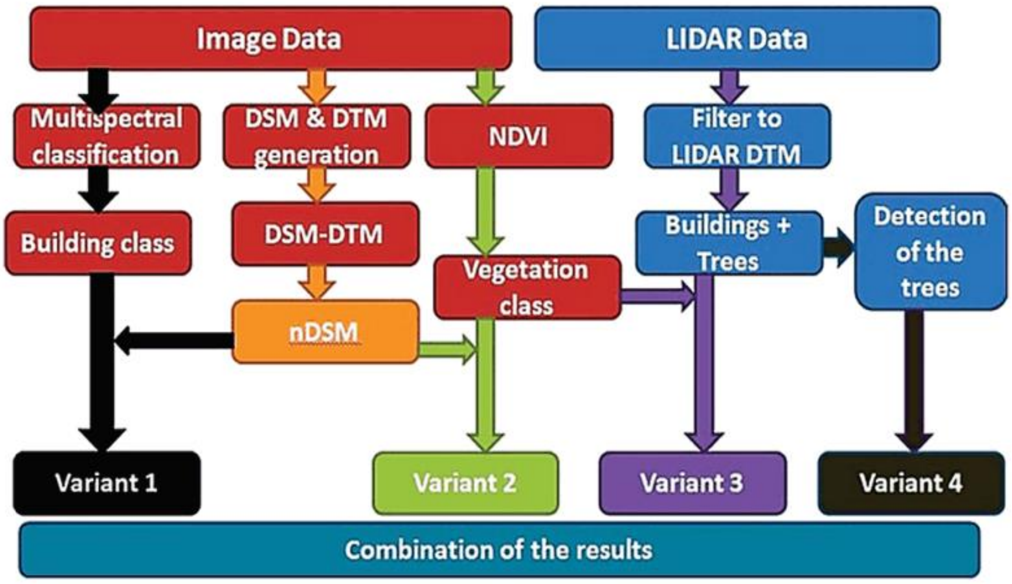


Figure 24 - Workflow of building detection.

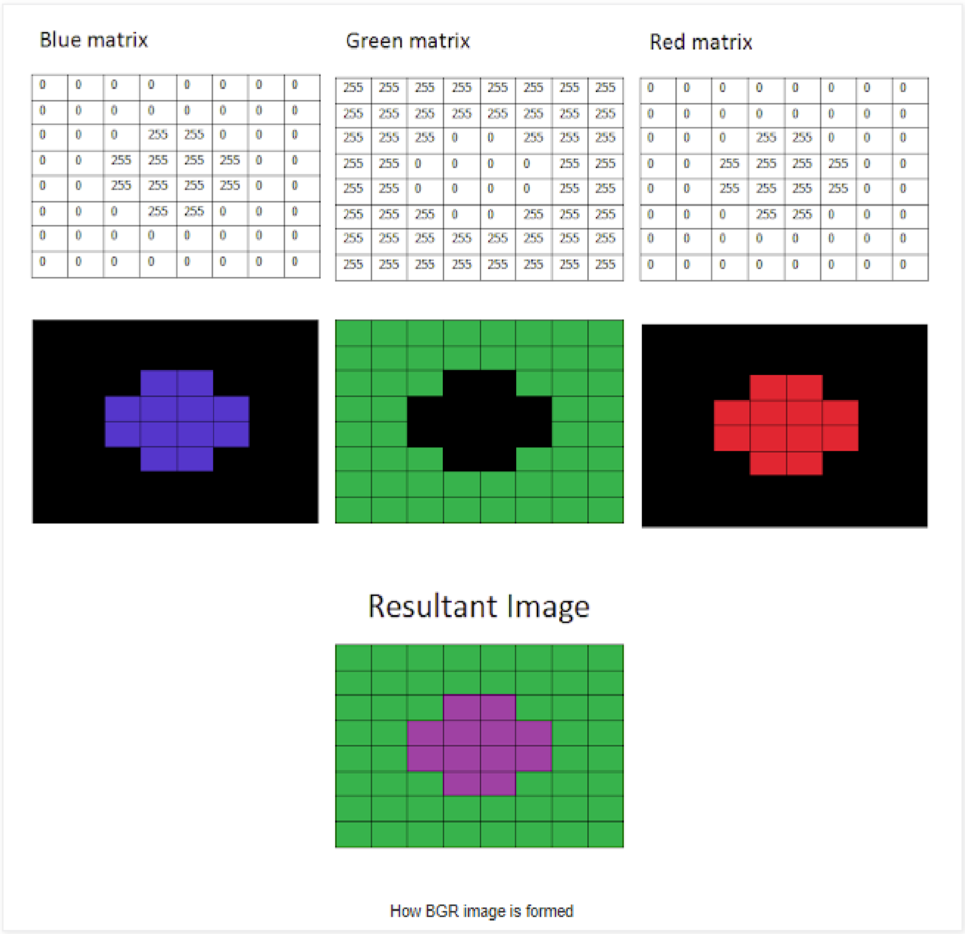
2.2.6 Classification of a Person Picture and Scenery

The strategy for the face area identification is the pattern to be turning out to be assortment, for example, shading, landscape trademark, measurement, layout conformer. One technique is to utilize colour analysis. This is to contrast the colour in the image and the colour guide to be made in the colour space. The performance will be good for the size and the direction of the face as per the demonstration above. Anyway, issues will occur at high light conditions. The other strategy is the technique to base at a territory highlight of the face. This concentrates the fundamental part of the face, for example, eye, nose and mouth. Also, in this way removes a face district from the principle segment's interrelations through the investigation. This can extricate a face locale even though the face comes to cover up halfway or inclines. However, it is troublesome that the interrelation among face components is resolved to the standard and made the calculation.

2.2.7 Detecting Objects Using Colours

Object detection using colours can be possible. However, currently, this is in a very primary stage. Object detection with accurate manner and segmentation is the difficult task in computer vision. It is a fundamental requirement in most of the applications related to image recognition and feature matching. However, it is still an open research area since currently there is no 100% accurate way to achieve these results.

Colour based methods can be used to easily recognize and segment a scenery or object. However, to achieve the best results, there should be a significant colour difference between the object and the background.



Blue matrix

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0
0	0	255	255	255	255	0	0
0	0	255	255	255	255	0	0
0	0	0	255	255	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Green matrix

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	0	0	255	255	255
255	255	0	0	0	0	255	255
255	255	0	0	0	0	255	255
255	255	255	0	0	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Red matrix

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0
0	0	255	255	255	255	0	0
0	0	255	255	255	255	0	0
0	0	0	255	255	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 25 - How BGR is formed

2.2.8 Tensorflow Object Detection API

In computer vision, making machine learning models which can be able to localize and recognize many objects in a single image is a challenge. TensorFlow object detection API is a framework which is open source and inherited from TensorFlow. As a result, this model will be able to effectively develop, train and deploy models related to object detections.

Models related to TensorFlow are available in GitHub which are very useful for some tasks since they are already pre-trained. Since TensorFlow has a very good API, it is capable of easily train object detection models for different kind of applications. TensorFlow will handle any technical condition including live stream sessions with high frames per seconds, desktop models and as a result, it will be easier to train and export a model.

This could be done by below steps

- Gathering a data set
    - Pascal and COCO, exist already, creating and labelling our data set are necessary to train a class which contains custom object detection class.
- Bounding Box Creation
    - There is a need for an image’s height, width, x and y-axis minimum and maximum values to train this object detection model. It is called a bounding box. Each image will require the above information. The bounding box is the key feature that captures the location of the class in the image.
- Install the object detection API
- Convert labels to the TFRecord format
- Choose a model

- Retrain the model with your data
- Implement new model with TensorFlow

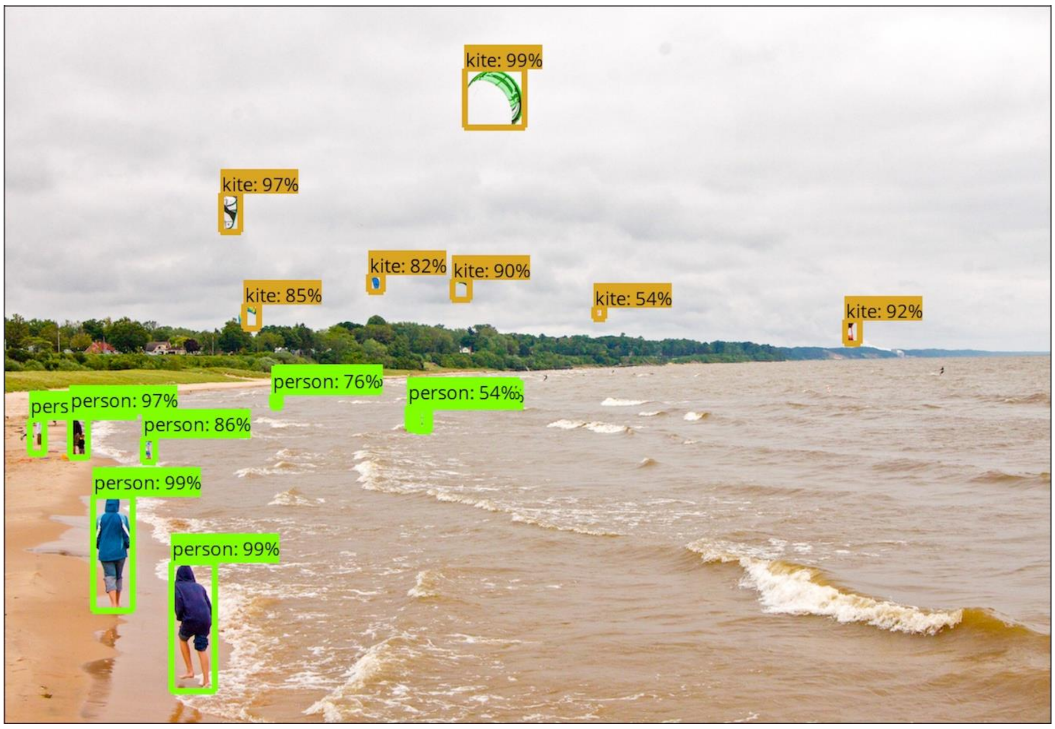


Figure 26 – Tensor-flow Object Detection

CHAPTER 3

METHODOLOGY

This chapter describes how accurate geolocation detection was obtained with the use of image processing technology combined with proper software architectural design. To decide which design architecture is best for the system implementation, a broad software design experience and knowledge are the essential facts. So, this research implemented an accurate geolocation detection system for mixed reality applications.

3.1 Important Considerations When Implementing the Solution

- Tracking of accurate geo-locations based on the visual data and inbuilt sensor data in mobile phone.
- This implementation can only be implemented to a selected location or an area since the area should be equipped with special server/s and network equipment.
- The implemented system doesn’t use any external pluggable devices to the mobile phone to increase the accuracy of location data.
- The implemented solution is acting as a framework to run the solution in average or low specification mobile phones (The term ‘average and low specification mobile phones’ represents the mobile phone available as at 31st March 2020).

3.2 Key Features the Implemented System Contains

Key features considered in implementing the proposed solution were to preserve computational power, minimum resource utilization and performance. To full fill above requirements, an image detection algorithm was used which consumes fewer resources and computational time. The consideration of fewer resources and less computational power is essential because this application runs in the background of an augmented reality application which also consumes a considerable amount of hardware resources. There are situations that this application may run for an extended amount of time when it is used in a large site like Yapahuwa. Therefore minimizing resource utilization, response time and CPU utilization is necessary.

3.3 Implementation of a Geolocation Detection System for Mobile

The research is mainly focused on detecting the locations accurately mainly using below two methods.

1. Image Recognition Technology (3.3.1)
2. Use of mobile phone's inbuilt sensors (3.3.2)

3.3.1 Image Recognition Technology

The image recognition part was implemented using a client-server architecture. The server is responsible to run the image recognition application, which was implemented using OpenCV. The mobile phone acts as the client which is responsible for continuously sending images to the server.

The algorithm used in the implemented system.

The image recognition algorithm which was used for the implementation is the ORB algorithm. ORB stands for "Oriented FAST and Rotated BRIEF" algorithm. It was first developed by OpenCV labs with the help of several engineers named Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski in 2011. This algorithm has become a very good alternative to SIFT and SURF algorithms. While SIFT and SURF algorithms are not free, ORB is free to use. Performance of the ORB algorithm is similar to SIFT and better than SURF. ORB is developed as a combination of two existing algorithms named FAST key-point detector and BRIEF descriptor.

What is FAST algorithm?

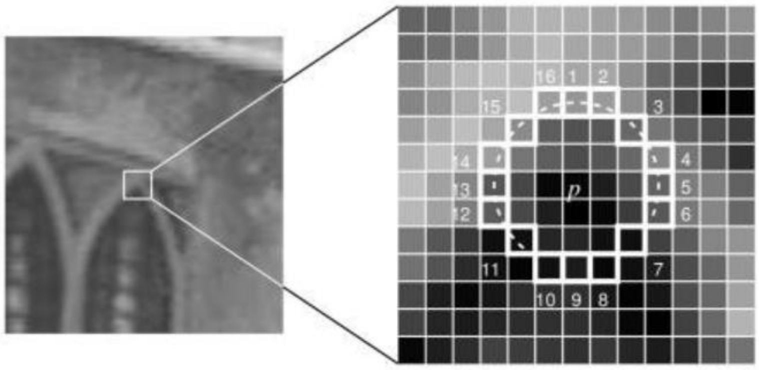


Figure 27 – How FAST algorithm works

As the above figure shows [27], a selected pixel will compare the brightness of its surrounding pixels in a circle (16 pixels). Then based on the brightness of those pixels will be divided into 3 parts. That is lighter than the selected pixel, darker than the selected pixel and similar to the selected pixel. If there are more than 8 pixels found which are lighter or darker than the selected pixel, ‘p’ will be selected as a key point. However, ‘FAST’ cannot identify the same image with different orientations and sizes. Therefore, the ORB algorithm uses orientation components and multi-scale features. ORB uses a pyramid of different scaled images of the same image as shown in figure [28]. It uses a multi-scale and multi-resolution representation of a single image [28]. Once the pyramid is created, ORB detects key points to match images. Once the key points are correctly identified ORB will allocate orientation for each key point based on the intensity (amount of light or the numerical value of a pixel) which will vary around that key point. ORB uses intensity centroid to calculate the intensity.

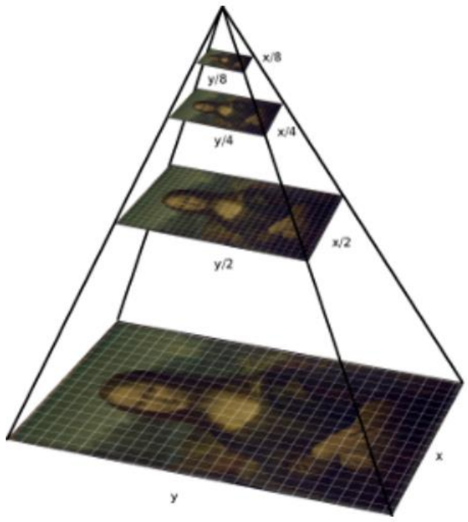


Figure 28 – ORB multi-scale image pyramid.

BRIEF (Binary robust independent elementary feature)

BRIEF gathers all the key points collected in FAST and forms a binary feature vector (binary feature descriptor) which can be represented as an object. The above-mentioned vector-only contains only 1s and 0s [29]. Every key point is represented by a feature vector with 128-512 bits string. BRIEF uses a Gaussian kernel to prevent high-frequency noise. Pixels around the neighbourhood are named as a patch (a square with width and height).

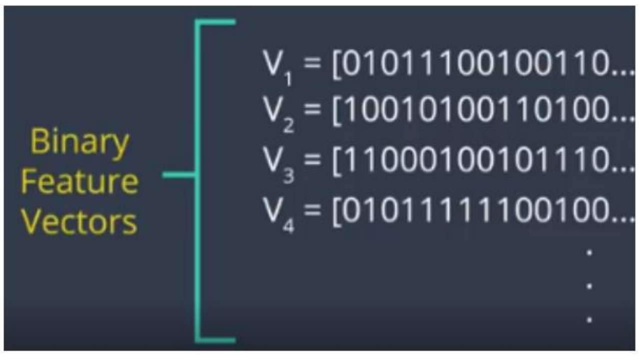

Binary Feature Vectors
V_1 = [01011100100110...
V_2 = [10010100110100...
V_3 = [11000100101110...
V_4 = [01011111100100...
.
.
.

Figure 29 – Binary Feature Vector used in BRIEF.

3.3.2 Use of Mobile Phone's Inbuilt Sensors

For the solution provided, a sensor available in the mobile phone is capable of maintaining the user step count. Once the location is correctly identified from the image recognition technology mentioned above (3.3.1) there was a requirement to maintain the correct location when a user walks in between an area where there are no special images to recognize location correctly. Therefore, an inbuilt sensor in the mobile phone is used to track the user location by counting the steps.

The sensor that is used for tracking user steps is 'Accelerometer'. An accelerometer is a small inbuilt device in a mobile phone which can measure the force of acceleration [30]. Therefore, it can measure the speed of an object. Accelerometer senses movement, gravity and the angle of the device.

Inbuilt step count methods available in mobile phone operating systems cannot be used since they have a small delay in counting steps. As an example, the inbuilt step count function available in the IOS operating system updates the step count in every 4 seconds. That delay is not acceptable for the implemented system. Therefore, to accurately measure the steps, we need to implement a custom way. For that collect the output data from accelerometer when walking is required and the data should be analyzed to filter out the relevant measuring points related to walking activities.

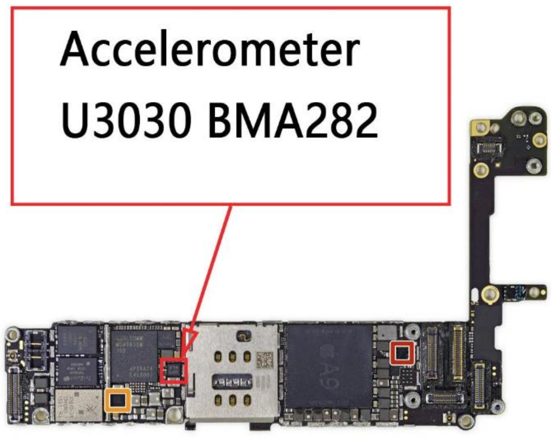


Figure 30 – Accelerometer Sensor (IPhone 6s)

Using the above-mentioned ORB algorithm, the implemented system can identify different locations based on the specific images provided for those locations. Images which are sent to the server can be divided into two main parts.

1. Special signboard images to identify specific locations.
2. Use a specifically identifiable image of an object or scenery as the template image.

There are situations where the implemented solution cannot identify a specific location by its specific features (As an example indoor navigation in plain coloured walls). In such scenarios, to identify the location specifically, we used some specifically identifiable signboard to identify the location (explained in point 1). In situations where an object or an entire scenery can be specifically identified by the ORB algorithm, we can go for the above-mentioned point 2.

3.4 Summary

The implemented solution provides accurate geo locations to mixed reality applications based on the user movements (indoor or outdoor). To achieve it, as the first step, different samples of the site images were prepared with unique identifiable feature/s using different image filters which reduced the complexity of the images. These sample images can be either special recognizable signs (signboards) or scenery images (ex: building, statue). Then they were stored in an image database. The image processing process which uses in the client-side (mobile phone) filters out unwanted data from the images and send them to the web service which is capable of recognizing images. As explained earlier the algorithm used for image recognition is ORB. Once the images are recognized, web service responds to the server the correct geolocation mapped to the image. Since we cannot place especially identifiable images in every nook and corner, there should be a mechanism to update location in between the transitions between two images. For that, the implemented solution uses a custom made step counter which uses mobile phone’s inbuilt accelerometer sensor data to count the user steps.

CHAPTER 4

SOLUTION ARCHITECTURE AND IMPLEMENTATION

The solution was implemented for augmented reality mobile application to work on a particular environment accurately, the architecture that was used is a basic 3-tier architecture which contains client, application server and database server [31].

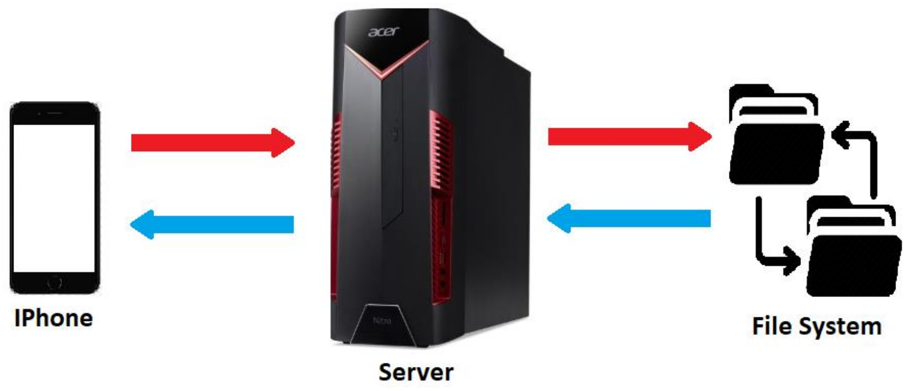


Figure 31 – Three Tier Architecture.

The software architecture of the solution is divided into three parts as shown in Figure 31. That is client, server and database. For the solution provided, an IOS device was used as the client. A general-purpose i7 desktop computer was used as the server. The server contains a web service written in Python language. OpenCV library was used as the image processing library while ORB algorithm was used as the image detection algorithm. Reasons for selecting OpenCV is because the application is a real-time application which requires quick response time. OpenCV is fast in performance and uses fewer resources compared to other popular image processing platforms. ORB is a good substitute to SIFT and SURF in computation cost, performance and the requirement of having patents. SIFT and SURF are patented products which mean users of them are supposed to pay for its use. However, ORB is completely free to use.

The client is capable of running a background process on top of the augmented reality application. The client (mobile application) was developed using IOS latest technologies SwiftUI and Swift. Therefore it is capable of sending visual images to the server every 1 or 2 seconds (time interval could vary depending on settings). Once the server receives the image data, the server runs the object detection algorithm (ORB) with the help of OpenCV to determine the exact location based on the images sent by the client. The client sends the images using a REST API. The server was implemented using a Python web service. Once the server is done with the relevant processing server responds to the client with accurate location data. Addition to that in the client-side, accelerometer is used to track the user movements in between two locations which have no image data to recognize accurate location (as explained in Chapter 3).

If further explained, the web service (REST API) was used as the communication media between the client and the server. The image data captured by the mobile phone is saved in a buffer in mobile phone performing the basic processing before sending to the server. Therefore the accuracy and responsiveness of data can be preserved. When the data arrive in the server-side, it uses another buffer to process the arrived image data.

As explained in chapter 3, the research is mainly focused on detecting the locations based on two approaches. That is image recognition technology and the use of a mobile phone’s inbuilt sensors. In below, above mentioned two approaches were explained in terms of the implementation.

4.1 Image Recognition Technology (Server Implementation)

The server-side in the implemented client-server architecture is responsible for running the image recognition application. The mobile phone acts as a client. It is responsible for continuously sending images to the server.

Implementation of the server-side

As shown in figure 32 separate method was developed to recognize images receiving from the client. The input parameter ‘recImg’ receives the images sent from the client. The input parameter ‘imgName’ is the template image name. Since this method is called iteratively the ‘imgName’ get changed. The calling of the method ‘imageIdentifier’ is available in figure 33. When the image sent by client received to the above method it performs below basic steps to for the image recognition process.

Steps:

I. Convert the image received as base64 string to a stream of binary data.

II. Save the received image in a variable.

III. Save the template image in a variable.

IV. Convert the received image and template image to grayscale.

V. Initializes an ORB class instance

VI. Detect key point by calling the method ‘orb.detectAndCompute’ for the above mentioned two images.

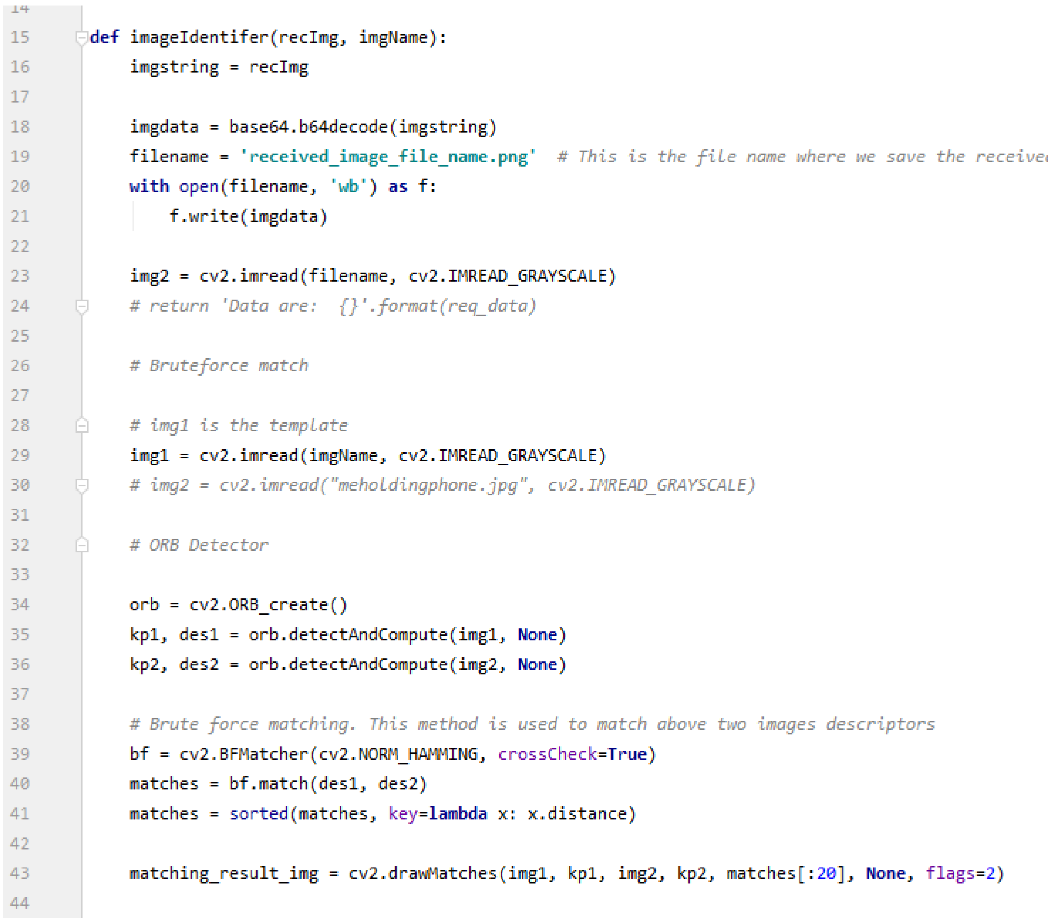
VII. Once the key points are calculated, brute force matching is performed by calling ‘bf.match(des1, des2)’ [32]

VIII. The matches acquired are then sorted into ascending order.

IX. Collect the matches into an array and then get the minimum value of the match result.

X. Since images are stored by the specific location names in the next step we remove the extension name of the image.

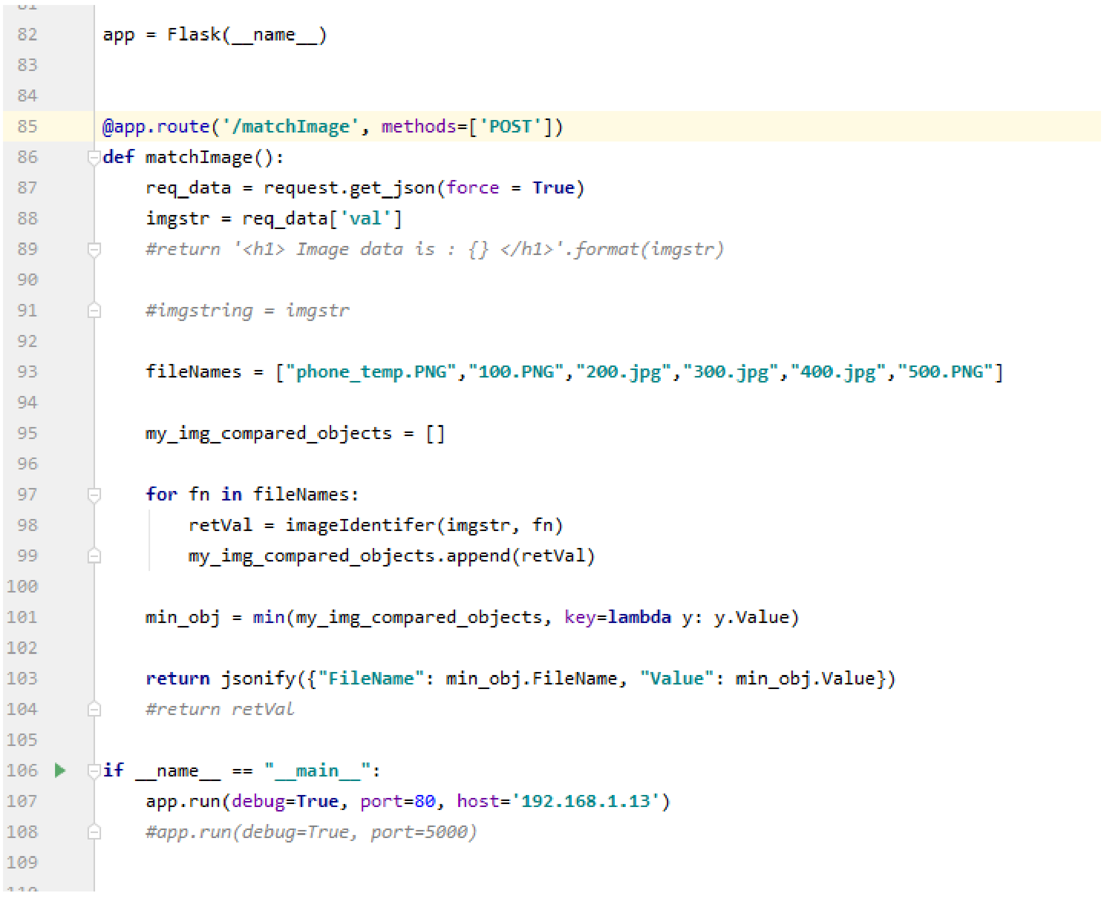
XI. Finally, if the minimum value is below 27, we consider that as a match. The value 27 was fine-tuned by doing a lot of tests. However, this can be changed based on the requirement.



```python
def imageIdentifer(recImg, imgName):
    imgstring = recImg

    imgdata = base64.b64decode(imgstring)
    filename = 'received_image_file_name.png'  # This is the file name where we save the receive
    with open(filename, 'wb') as f:
        f.write(imgdata)

    img2 = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    # return 'Data are:  {}'.format(req_data)

    # Bruteforce match

    # img1 is the template
    img1 = cv2.imread(imgName, cv2.IMREAD_GRAYSCALE)
    # img2 = cv2.imread("meholdingphone.jpg", cv2.IMREAD_GRAYSCALE)

    # ORB Detector

    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

    # Brute force matching. This method is used to match above two images descriptors
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)

    matching_result_img = cv2.drawMatches(img1, kp1, img2, kp2, matches[:20], None, flags=2)
```

Figure 32 – Server side code part 1

The above-mentioned method works as a service. An already written REST API library called “Flask” was used as shown in figure 33 to implement the web service. Since template images are stored from their file names in the file system, an array of file names were created to call the method iteratively for each file name. In the service level, another array is available to collect the minimum values returned from method (‘imageIdentifier’). Once the iterative calls are over the service gets the minimum of all those returned values and responses the server with the correct file name which is the accurate location.



```python
app = Flask(__name__)


@app.route('/matchImage', methods=['POST'])
def matchImage():
    req_data = request.get_json(force = True)
    imgstr = req_data['val']
    #return '<h1> Image data is : {} </h1>'.format(imgstr)

    #imgstring = imgstr

    fileNames = ["phone_temp.PNG","100.PNG","200.jpg","300.jpg","400.jpg","500.PNG"]

    my_img_compared_objects = []

    for fn in fileNames:
        retVal = imageIdentifer(imgstr, fn)
        my_img_compared_objects.append(retVal)

    min_obj = min(my_img_compared_objects, key=lambda y: y.Value)

    return jsonify({"FileName": min_obj.FileName, "Value": min_obj.Value})
    #return retVal

if __name__ == "__main__":
    app.run(debug=True, port=80, host='192.168.1.13')
    #app.run(debug=True, port=5000)
```

Figure 33 – Server side code part 2

4.2 Use of Mobile Phone's Inbuilt Sensors (Client Implementation)

Client-side implementation of this application plays a major role in terms of usability and accuracy. The sensor that was used in this solution is the accelerometer. It is capable of maintaining the user step count. Once the location is correctly identified from the image recognition technology, there was a requirement to maintain the correct location when a user is walking in between an area where there are no special images to recognize location correctly. Therefore an inbuilt sensor (accelerometer) in the mobile phone was used to track the user location by counting the steps.

For the implemented solution an iPhone device was used as the client. The client application was built for the IOS operating system with native language SwiftUI and Swift as explained earlier. SwiftUI is the latest technology for IOS development currently available. Inbuilt step count methods available in IOS operating systems cannot be used since it has a small delay in counting steps as explained earlier. As an example the inbuilt step count function push updates for every 4 to 5 seconds which is not acceptable for this type of real-time system. Therefore to accurately measure the steps in real-time a custom method was developed using accelerometer sensor data outputs. Output data gathered when walking was obtained and analyzed to filter out the relevant measuring points related to walking activities. Below steps explains how the step count method was implemented as shown in figure 34.

Steps:

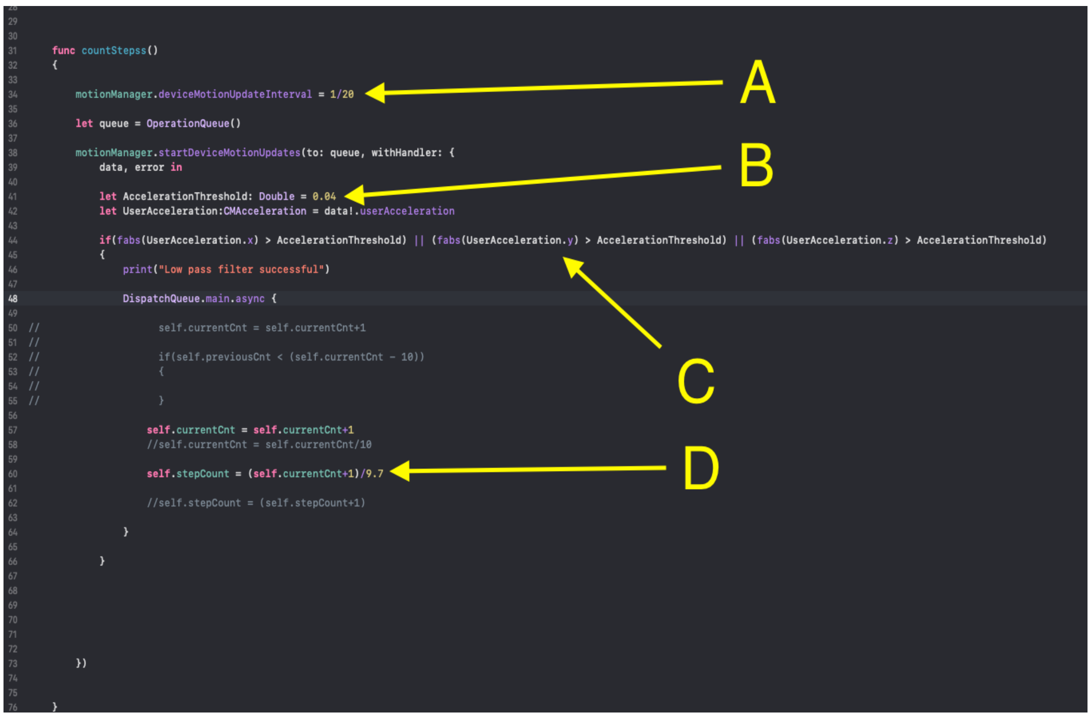
I. At the first stage, recognize of time interval the method should run (A)

II. Define a threshold value for the acceleration data from sensors (B)

III. Obtain the absolute values of the sensor data coming from the accelerometer (C). In here there are three types of data receiving. Movement data of the phone for X-axis, Y-axis and Z-axis. The above-mentioned threshold value was used to filter the data only related to steps.

IV. Dividing the count data by 9.7. This is to omit unwanted data (D).

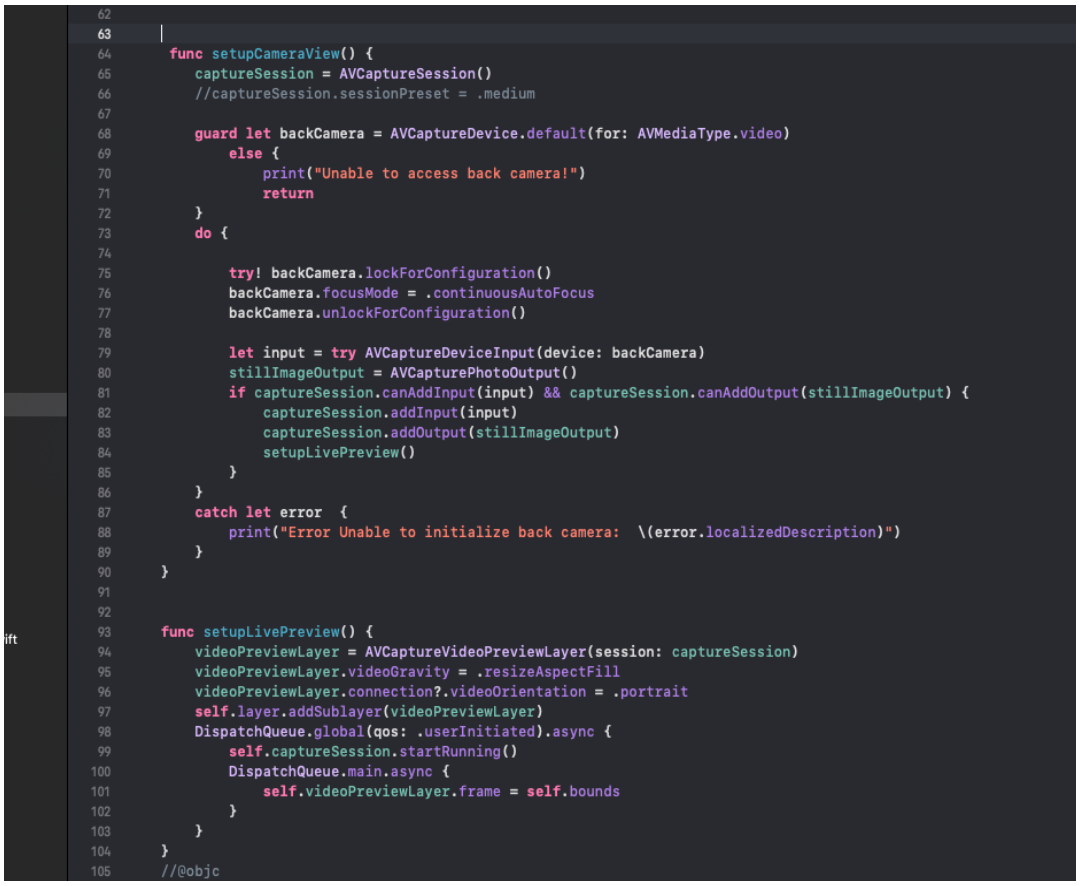
In all the above steps the values used for different variable factors were gained by doing a lot of testing.



```
    func countStepss()
    {

        motionManager.deviceMotionUpdateInterval = 1/20

        let queue = OperationQueue()

        motionManager.startDeviceMotionUpdates(to: queue, withHandler: {
            data, error in

            let AccelerationThreshold: Double = 0.04
            let UserAcceleration:CMAcceleration = data!.userAcceleration

            if(fabs(UserAcceleration.x) > AccelerationThreshold) || (fabs(UserAcceleration.y) > AccelerationThreshold) || (fabs(UserAcceleration.z) > AccelerationThreshold)
            {
                print("Low pass filter successful")

                DispatchQueue.main.async {

//                      self.currentCnt = self.currentCnt+1
//
//                      if(self.previousCnt < (self.currentCnt - 10))
//                      {
//
//                      }

                    self.currentCnt = self.currentCnt+1
                    //self.currentCnt = self.currentCnt/10

                    self.stepCount = (self.currentCnt+1)/9.7

                    //self.stepCount = (self.stepCount+1)

                }

            }





        })


    }
```

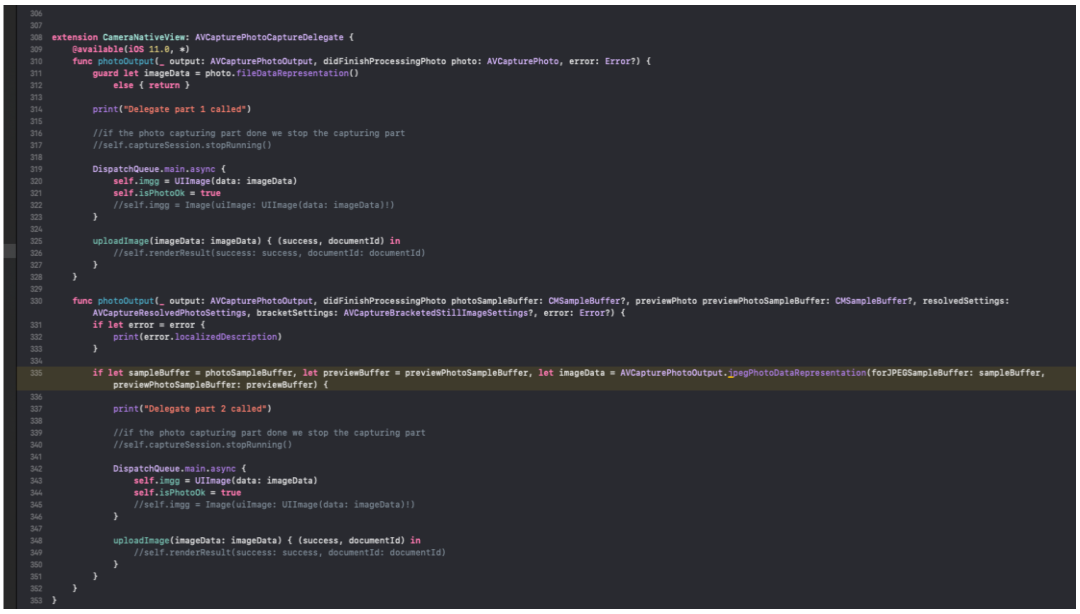
Figure 34 – Accelerometer Implementation of Client

After developing the step count method the next implementation stage contained the development of camera application and the REST API. Camera application contains a mix of two technologies. That is Swift and SwiftUI. A custom-developed camera application was used since the application required more control on hardware features. Therefore the custom made camera application was developed using Swift language and the user interface is connected using a representable view as shown in figure 35, 36 and 37.



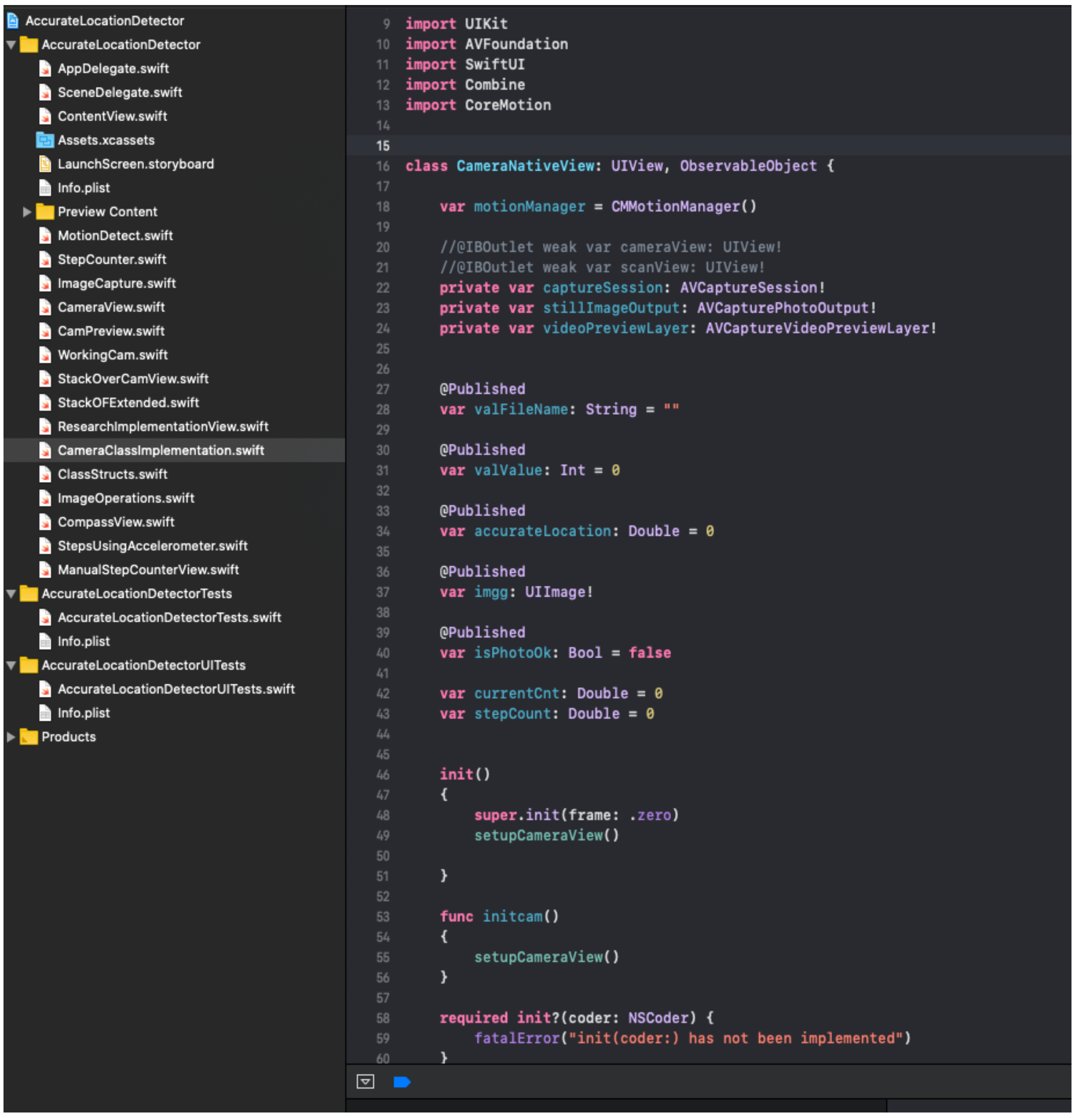
```swift

    func setupCameraView() {
        captureSession = AVCaptureSession()
        //captureSession.sessionPreset = .medium

        guard let backCamera = AVCaptureDevice.default(for: AVMediaType.video)
            else {
                print("Unable to access back camera!")
                return
        }
        do {

            try! backCamera.lockForConfiguration()
            backCamera.focusMode = .continuousAutoFocus
            backCamera.unlockForConfiguration()

            let input = try AVCaptureDeviceInput(device: backCamera)
            stillImageOutput = AVCapturePhotoOutput()
            if captureSession.canAddInput(input) && captureSession.canAddOutput(stillImageOutput) {
                captureSession.addInput(input)
                captureSession.addOutput(stillImageOutput)
                setupLivePreview()
            }
        }
        catch let error  {
            print("Error Unable to initialize back camera:  \(error.localizedDescription)")
        }
    }


    func setupLivePreview() {
        videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
        videoPreviewLayer.videoGravity = .resizeAspectFill
        videoPreviewLayer.connection?.videoOrientation = .portrait
        self.layer.addSublayer(videoPreviewLayer)
        DispatchQueue.global(qos: .userInitiated).async {
            self.captureSession.startRunning()
            DispatchQueue.main.async {
                self.videoPreviewLayer.frame = self.bounds
            }
        }
    }
    //@objc
```

Figure 35 - Camera View Code Section 1



```swift


extension CameraNativeView: AVCapturePhotoCaptureDelegate {
    @available(iOS 11.0, *)
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVCapturePhoto, error: Error?) {
        guard let imageData = photo.fileDataRepresentation()
            else { return }

        print("Delegate part 1 called")

        //if the photo capturing part done we stop the capturing part
        //self.captureSession.stopRunning()

        DispatchQueue.main.async {
            self.imgg = UIImage(data: imageData)
            self.isPhotoOk = true
            //self.imgg = Image(uiImage: UIImage(data: imageData)!)
        }

        uploadImage(imageData: imageData) { (success, documentId) in
            //self.renderResult(success: success, documentId: documentId)
        }
    }

    func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photoSampleBuffer: CMSampleBuffer?, previewPhoto previewPhotoSampleBuffer: CMSampleBuffer?, resolvedSettings:
        AVCaptureResolvedPhotoSettings, bracketSettings: AVCaptureBracketedStillImageSettings?, error: Error?) {
        if let error = error {
            print(error.localizedDescription)
        }

        if let sampleBuffer = photoSampleBuffer, let previewBuffer = previewPhotoSampleBuffer, let imageData = AVCapturePhotoOutput.jpegPhotoDataRepresentation(forJPEGSampleBuffer: sampleBuffer,
            previewPhotoSampleBuffer: previewBuffer) {

            print("Delegate part 2 called")

            //if the photo capturing part done we stop the capturing part
            //self.captureSession.stopRunning()

            DispatchQueue.main.async {
                self.imgg = UIImage(data: imageData)
                self.isPhotoOk = true
                //self.imgg = Image(uiImage: UIImage(data: imageData)!)
            }

            uploadImage(imageData: imageData) { (success, documentId) in
                //self.renderResult(success: success, documentId: documentId)
            }
        }
    }
}
```

Figure 36 - Camera View Code Section 2



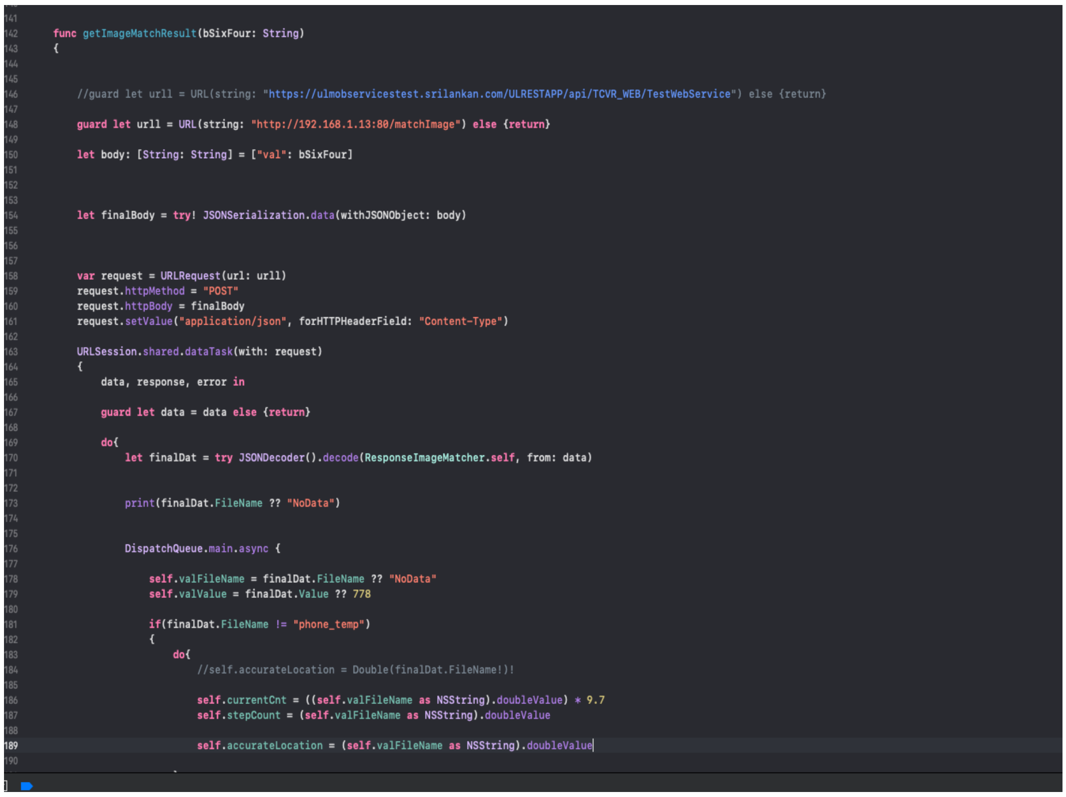
```swift
import UIKit
import AVFoundation
import SwiftUI
import Combine
import CoreMotion


class CameraNativeView: UIView, ObservableObject {

    var motionManager = CMMotionManager()

    //@IBOutlet weak var cameraView: UIView!
    //@IBOutlet weak var scanView: UIView!
    private var captureSession: AVCaptureSession!
    private var stillImageOutput: AVCapturePhotoOutput!
    private var videoPreviewLayer: AVCaptureVideoPreviewLayer!


    @Published
    var valFileName: String = ""

    @Published
    var valValue: Int = 0

    @Published
    var accurateLocation: Double = 0

    @Published
    var imgg: UIImage!

    @Published
    var isPhotoOk: Bool = false

    var currentCnt: Double = 0
    var stepCount: Double = 0


    init()
    {
        super.init(frame: .zero)
        setupCameraView()

    }

    func initcam()
    {
        setupCameraView()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
```

Figure 37 - Camera View Code Section 3

Below explanations are related to the code available in figure 35, 36 and 37.

- Variables which are named as ‘@Published’ [37] are used to inform the changes to the interface of the camera application.
- Autofocus mode is enabled as shown in figure 35.
- The camera application was set up to get still image outputs [35]
- The live preview which can be seen when the application is running was implemented in the method ‘setupLivePreview()’ [35]

REST API Implementation



```
    func getImageMatchResult(bSixFour: String)
    {


        //guard let url1 = URL(string: "https://ulmobservicestest.srilankan.com/ULRESTAPP/api/TCVR_WEB/TestWebService") else {return}

        guard let url1 = URL(string: "http://192.168.1.13:80/matchImage") else {return}

        let body: [String: String] = ["val": bSixFour]



        let finalBody = try! JSONSerialization.data(withJSONObject: body)



        var request = URLRequest(url: url1)
        request.httpMethod = "POST"
        request.httpBody = finalBody
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        URLSession.shared.dataTask(with: request)
        {
            data, response, error in

            guard let data = data else {return}

            do{
                let finalDat = try JSONDecoder().decode(ResponseImageMatcher.self, from: data)


                print(finalDat.FileName ?? "NoData")


                DispatchQueue.main.async {

                    self.valFileName = finalDat.FileName ?? "NoData"
                    self.valValue = finalDat.Value ?? 778

                    if(finalDat.FileName != "phone_temp")
                    {
                        do{
                            //self.accurateLocation = Double(finalDat.FileName!)!

                            self.currentCnt = ((self.valFileName as NSString).doubleValue) * 9.7
                            self.stepCount = (self.valFileName as NSString).doubleValue

                            self.accurateLocation = (self.valFileName as NSString).doubleValue

```

Figure 38 - REST API Implementation (Client)

The REST API in the client application was used to automatically send the images taken to the server. Images are automatically taken based on a time interval that can be adjustable. An image taken in a time interval is first converted into a base64 string and the converted base64 string is attached to the web service as an input parameter and send to the server [38]. JSON requests are used to communicate with server and client. Once the server completes the processing it responses the client another JSON indicating the accurate location.

CHAPTER 5

EVALUATION

6.1 Evaluation

Evaluation is the only way to show the success of an implementation. There are two types of evaluations available, which are interface evaluation and functionality evaluation.

6.2 Hardware Specifications

- Server Specifications:
    - Intel Core i7 3770k
    - MSI GTX 1070 Graphics Card
    - 8 GB of RAM
- Operating System: Windows 10 64 bit
- Mobile Phone used for testing :
    - iPhone 6S Plus
- Number of Sign Images tested: 20
- Number of Scenes tested: 30

6.3 Evaluation process Steps and Results

Several questions were asked from 8 randomly selected users initially and below are the results of the questionnaires. Since there is no interface for the system the interface evaluation has not been done.

Functionality Evaluation

The main idea of evaluating the functionality is to get the usefulness and the performance of the system. To calculate them the following questions were asked from all invited users. Following questions were asked to get the overall idea of the system.

Evaluation 1 - Questionnaire 1 results (Appendix A) [Test 1]

Does the system lag? 7 out of 8 users said no.

Is the process running background? 8 out of 8 users said yes

Can the server-side system save new objects? 8 out of 8 users said yes.

Can the system identify objects correctly? 6 out of 8 users said yes

How is the battery consumption of the system? 6 out of 8 users said average.

Can the system correctly identify differences of geographical locations? 6 out of 8 users said yes.

Does the system generate graphic images related to the correct location in the virtual reality application? 6 out of 8 users said yes.

What do you think about overall system performance? 7 out of 8 users said well.

Two system evaluations were conducted and the results are listed below.

Evaluation 2 results (Appendix B) [Test 2]

After completing the questionnaire 1 the server part of the application was tested by feeding 20 images into the image recognition algorithm and below are the results.

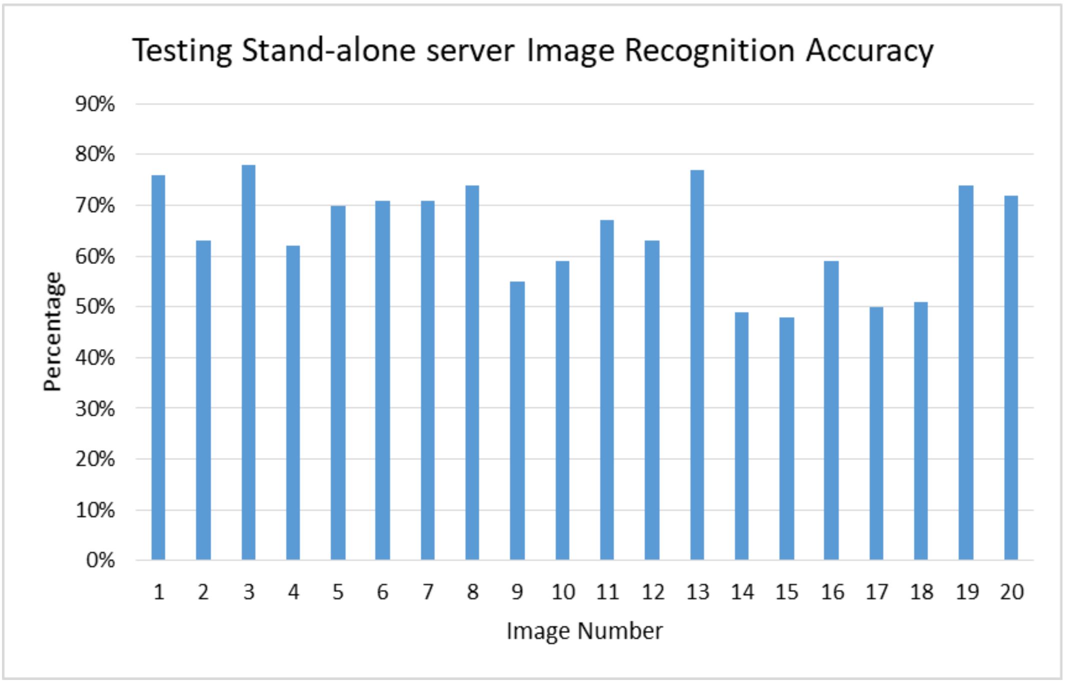


Figure 39 – Server Image Identification Accuracy

Image No:	Accuracy
1	76%
2	63%
3	78%
4	62%
5	70%
6	71%
7	71%
8	74%
9	55%
10	59%
11	67%
12	63%
13	77%
14	49%
15	48%
16	59%
17	50%

18	51%
19	74%
20	72%

In Figure 39 except image 14, 15 and 17, all other 17 scenarios show considerably good averages above 50% accuracy reading which indicates the stand-alone server application is capable of identifying images using feature extraction technology.

- The overall test indicates 85% of success in the above gives 20 test scenarios.
- The outdoor image recognition test shows a 76% success rate. Refer Appendix B for more information.

Evaluation 3 results [Test 3]

After completing the evaluation 1, evaluation 2 was conducted using both server application and the IOS application (Using Mobile Phone). In here the application was tested by running it in 30 different sceneries. The test results are shown below.

- The overall test indicates 80% of success in the above gives 30 test scenarios.
- The outdoor image recognition test shows a 72% success rate. Refer Appendix C for more information.

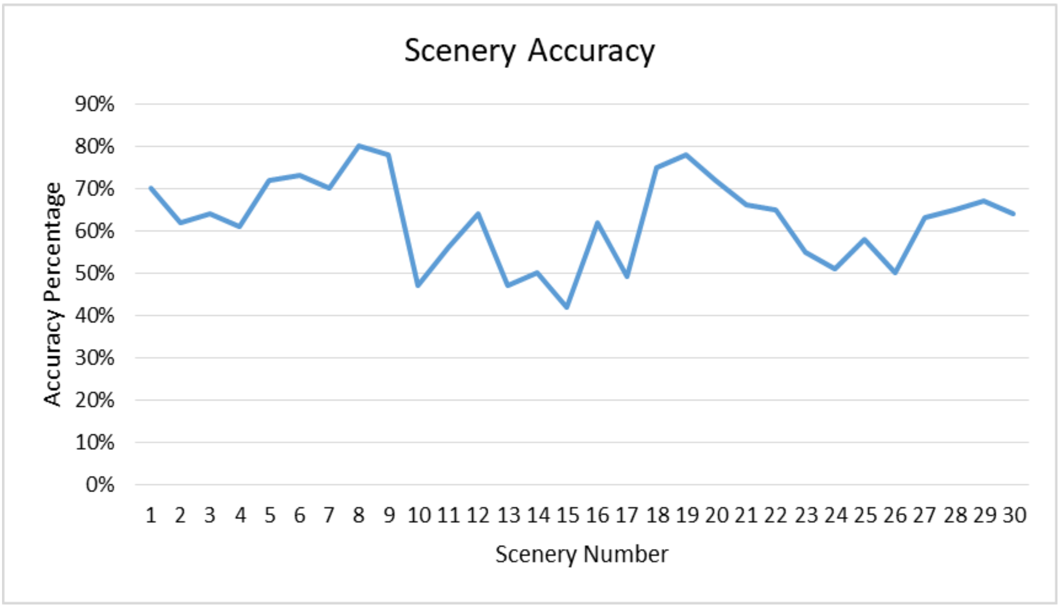


Figure 40 – Scenery Accuracy

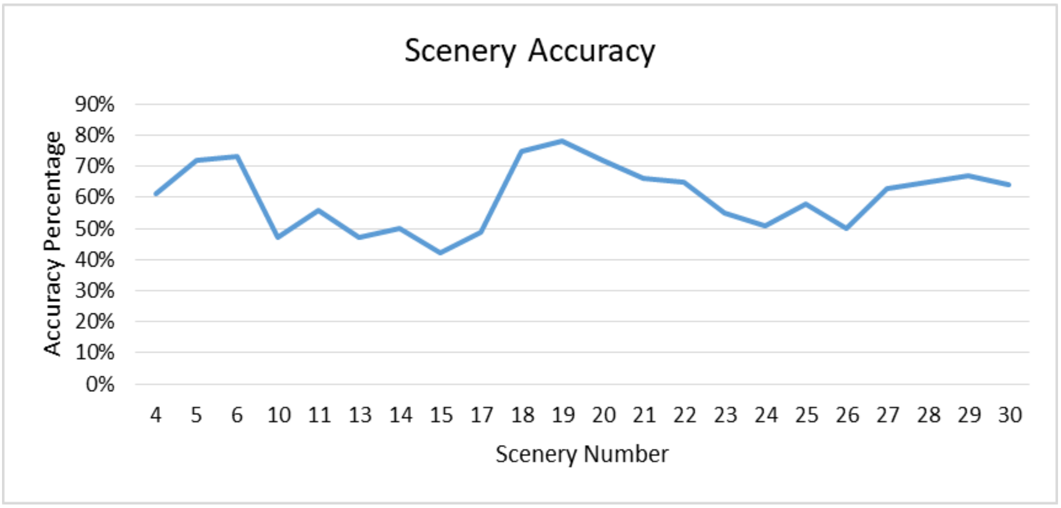


Figure 41 - Scenery Accuracy (Outdoor Only)

In Figure 40, except sceneries 10, 13, 14, 15, 17 and 26, all other 24 scenarios show considerably good accuracy readings above 50%. That indicates the overall system is relatively better in the aspect of accuracy.

The overall system evaluation has depicted that the results indicate all the evaluations were satisfied with the current mixed reality application. Hence, the project objectives have successfully achieved within a limited boundary.

6.4 Sample images of the developed system

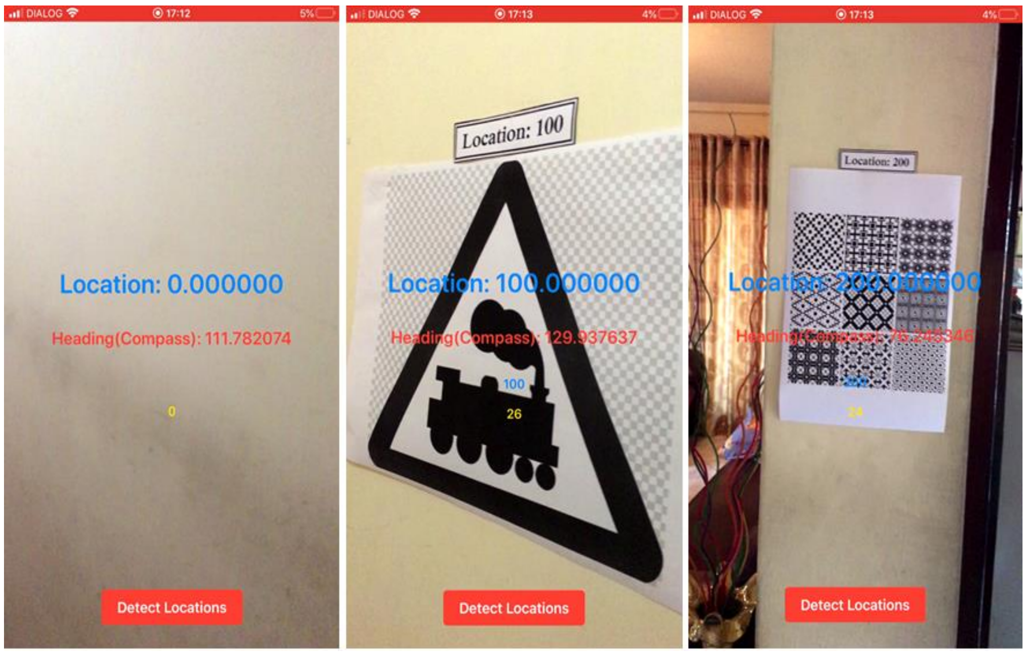


Figure 42 - Sample images of the implemented system

CHAPTER 6

CONCLUSTION

7.1 Conclusion

The implemented solution was mainly developed to accurately identifying pinpoint geolocation in a given site. The solution was developed for mixed reality applications. Implementation of this research was done to a limited area of a site to prove that the same methodology can be applied for a larger site. The main objective of the research, in brief, was to develop a solution which is capable of identifying most accurate geolocations based on the visual and mobile phone's sensor data which can be used for mixed reality applications. The solution runs in the background of the main application while not affecting the user experience of the virtual reality application.

There were many types of research done in the field of image recognition in indoor and outdoor. However, there aren't many researches done in the field of indoor navigation using image recognition & mobile phone's sensor data. Most of the researches were done for the robotic industry and ordinary desktop computers.

The implemented solution is followed by a three-tier architecture. In the first stage, unique images related to the sceneries were collected and selected to identify the relevant sceneries accurately. After the collection was done those images were used to identify the sceneries accurately using an image recognition algorithm.

The evaluation was conducted via eight independent evaluators. The Evaluation was not divided into two main sections called Interface evaluation and functionality evaluation because the implementation doesn't contain an interface. The evaluation had good evaluation results such as, for the overall system evaluation. Therefore as an overall, the implemented solution is successful.

7.2 Limitations

There are limitations in the implementation. They are both hardware and software limitations. Below are the limitations. The research was implemented for a limited area of Yapahuwa Archeological site since it requires a lot of image training.

Hardware Limitations:

Since the solution provided needs to run in parallel with an augmented reality application it consumes a considerable amount of energy. Average mobile phones cannot handle high processing tasks (mobile phones available as at 31st March 2020). To make the solution available for the majority of mobile devices available in the market the three-tier architecture was introduced. Three-tier architecture requires additional hardware resources. Example: application server, database server.

At some points due to packet transfer delays in network devices the image recognition results from the server get delayed. At such points, the application cannot generate accurate location details to its consumer.

Software Limitations:

As the methodology chapter explains, identification of signboards or scenery images can be done only if they are uniquely identifiable. If there are no uniquely identifiable features in an image the application outputs invalid results. Therefore before we set up the system for a particular site, lot of effort need to be put on selecting uniquely identifiable images. Addition to that, sceneries with no recognizable objects cannot be identified with the use of this application.

The system is capable of recognizing sceneries and detect locations only in good lighting conditions. If the lighting conditions are bad the system won't be able to identify features of the images. Hence, incorrect results are outputted.

No user interface to train scenery and signboard images in a user-friendly way. The application has to be configured for a selected site manually by adding uniquely identifiable images.

7.3 Future Work

Future work can be done in the area of improving the performance and accuracy of the system. Below areas can be further modified as future work.

Currently, the system uses 3 tier architecture to run the application. This can be further improved to run the application only in one mobile device without any help of a 3rd party. To achieve this, less resource consuming & efficient image recognition algorithm can be used.

The application can be further developed to identify any given scenery specifically regardless of uniquely identifiable objects. Therefore it should be capable of identifying sceneries with no objects. This can be achieved by doing research related to scenery detection using a combination of edge detection as well as colour variation detection.

The application can be developed in a way where it will use less network traffic which will overcome the problems of network delays. This will improve the response time of the entire application. The network structure design can be improved to achieve this.

The system can be further developed to identify the sceneries both day and night time. This can be achieved by either using high-end lenses or night vision technology. However, a specially designed image recognition algorithm should also be needed.

A user-friendly interface can be implemented for training sceneries. Implementing this feature will enable application owners to configure their environment without any technical knowledge. Features such as uniquely identifiable image detection, rejecting non-appropriate images can be implemented.

Use pattern recognition technology to improve step count feature in the implemented solution. Using pattern recognition, user activities such as walking can detect with high accuracy. To implement that a lot of training has to be done by collecting user activity data (Ex: walking, standing etc).

Use GPS data to improve accuracy and performance in large sites. Currently, the implemented solution does not use GPS technology to measure the location. However, GPS technology can be helpful to identify areas in large sites. Because GPS will help to keep the location accuracy up to some specific area. Since the implemented solution needs a lot of images at the site as samples, implementing the solution in a large site requires a huge number of unique images. As a result, the performance of the application may get decreased during the search time. As a solution, we can use GPS technology to narrow down search time by allowing the searching algorithm to search only for particular images related to a specific area. That can be done by mapping GPS coordinate details with image names by using a separate database. Therefore the time taken to search the images will decrease. In this way, the application can be developed to suit any large site.

7.4 REFERENCES

[1] O. Chum and J. Matas, “Matching with PROSAC-Progressive sample consensus,” *in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR ’05), vol. 1, pp. 220–226, June 2005.

[2] X. Wang, M. Yang, and K. Yu, “Efficient re-ranking in vocabulary tree based image retrieval,” *in Proceedings of the 45th Asilomar Conference on Signals, Systems and Computers,* pp. 855–859, 2011.

[3] S. S. Tsai, D. Chen, G. Takacs et al., “Fast geometric re-ranking for image-based retrieval,” *in Proceedings of the 17th IEEE International Conference on Image Processing* (ICIP ’10), pp. 1029–1032, Hong Kong, September 2010.

[4] H. Jegou, M. Douze, and C. Schmid, “Hamming embedding ′ and weak geometric consistency for large scale image search,” *Journal of Sensors*

[5] X. Wang, M. Yang, T. Cour, S. Zhu, K. Yu, and T. X. Han, “Contextual weighting for vocabulary tree based image retrieval,” *in Proceedings of the IEEE International Conference on Computer Vision* (ICCV ’11), pp. 209–216, Barcelona, Spain, November 2011.

[6] G. Takacs, V. Chandrasekhar, N. Gelfand et al., “Outdoors augmented reality on mobile phone using loxel-based visual feature organization,” *in Proceedings of the 1st International ACM Conference on Multimedia Information Retrieval* (MIR ’08), pp. 427–434, New York, NY, USA, August 2008.

[7] A. Kumar, J.-P. Tardif, R. Anati, and K. Daniilidis, “Experiments on visual loop closing using vocabulary trees,” *in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (CVPR ’08), pp. 1–8, Anchorage, Alaska, USA, June 2008.

[8] Lucas Paletta. “Visual Object Recognition in the context of mobile vision services”, *JOANNEUM RESEARCH Forschungsgesellschaft mbH, Institute of Digital Image Processing, Computational Perception Group (CAPE),* Wastiangasse 6, 8010 Graz.0

[9] Y. Li and L. Shapiro. “Consistent line clusters for building recognition in CBIR”. *In Proc. International Conference on Pattern Recognition,* ICPR 2002, volume 3, pages 952–957, 2002.

[10] A.R. Dick, P.H.S. Torr, and R. Cipolla. “Modelling and interpretation of architecture from several images”. *International Journal of Computer Vision,* 60(2):111–134, 2004.

[11] Niels Henze, Torben Schinke, Susanne Boll, “Object Recognition from Natural Features on a Mobile Phone”.

[12] Michael Rohs∗ , Beat Gfeller, “Using camera-equipped mobile phones for interacting with real-world objects”. *in 2004.*

[13] “Server-side object recognition and client-side object tracking for mobile augmented reality - IEEE Xplore Document,” *Server-side object recognition and client-side object tracking for mobile augmented reality - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/5543248/. [Accessed: 02-Jan-2017].

[14] “Real-time object tracking on mobile phones - IEEE Xplore Document,” *Real-time object tracking on mobile phones - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/6166663/. [Accessed: 20-Dec-2016].

[15] “Single-view recaptured image detection based on physics-based features - IEEE Xplore Document,” *Single-view recaptured image detection based on physics-based features - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/5583280/. [Accessed: 27-Nov-2016].

[16] “Development of a real time image based object recognition method for mobile AR-devices,” *Development of a real time image based object recognition method for mobile AR-devices*. [Online]. Available: http://dl.acm.org/citation.cfm?id=602355. [Accessed: 15-Dec-2016].

[17] “Outdoors augmented reality on mobile phone using loxel-based visual feature organization,” *Outdoors augmented reality on mobile phone using loxel-based visual feature organization*. [Online]. Available: http://dl.acm.org/citation.cfm?id=1460165. [Accessed: 19-Dec-2016].

[18] “SURFTrac: Efficient tracking and continuous object recognition using local feature descriptors - IEEE Xplore Document,” *SURFTrac: Efficient tracking and continuous object recognition using local feature descriptors - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/5206831/. [Accessed: 12-Dec-2016].

[19] “Server-side object recognition and client-side object tracking for mobile augmented reality - IEEE Xplore Document,” *Server-side object recognition and client-side object tracking for mobile augmented reality - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/5543248/. [Accessed: 04-Jan-2017].

[20] “ORB: An efficient alternative to SIFT or SURF - IEEE Xplore Document,” *ORB: An efficient alternative to SIFT or SURF - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/6126544/. [Accessed: 05-Jan-2017].

[21] “Streaming mobile augmented reality on mobile phones - IEEE Xplore Document,” *Streaming mobile augmented reality on mobile phones - IEEE Xplore Document*. [Online]. Available: http://ieeexplore.ieee.org/document/5336472/. [Accessed: 07-Jan-2017].

[22] S. H. Neven, H. Neven, and G. Inc., “Patent US7565139 – Image-based search engine for mobile phones with camera,” *Google Books*. [Online]. Available: https://www.google.com/patents/US7565139. [Accessed: 04-Jan-2017].

[23] “City-scale landmark identification on mobile devices - IEEE Xplore Document,” *City-scale landmark identification on mobile devices - IEEE Xplore*

Document. [Online]. Available: http://ieeexplore.ieee.org/document/5995610/. [Accessed: 28-Dec-2016].

[24] C. Stauffer and W. Grimson. Adaptive background mixture models for realtime tracking. In Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, page 246252, 1999.

[25] Ashley Colley1* , Jacob Thebault-Spieker2* , Allen Yilun Lin3* , Donald Degraen4 , Benjamin Fischman2 , Jonna Häkkilä1 , Kate Kuehl2 , Valentina Nisi5 , Nuno Jardim Nunes5 , Nina Wenig6 , Dirk Wenig6 , Brent Hecht3**, Johannes Schöning6 "The Geography of Pokémon GO: Beneficial and Problematic Effects on Places and Movement." *University of Lapland (Finland), University of Minnesota (USA), Northwestern University (USA), Saarland University (Germany), Madeira-ITI (Portugal), University of Bremen (Germany)*

[26] "Updating Google Maps with Deep Learning and Street View," Research on *Google Blogs*. [Online]. Available: https://research.googleblog.com/2017/05/updating-google-maps-with-deep-learning.html. [Accessed: 04-June-2017].

[27] "Google Street View: Capturing the World at Street Level" Available: http://www.vincent-net.com/luc/papers/10ieeecomputer_theworldatstreetlevel.pdf [Accessed: 04-June-2017].

[28] "What does it mean to be a location-based augmented reality app" Available: https://placenote.com/blog/2018/01/15/location-based-augmented-reality/[Accessed: 04-June-2017].

[29] Stephen Se and David Lowe, Jim Little "Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks. *First Published on August 1, 2002.*

[30] Sid Ahmed Berrabah1,2, Hichem Sahli2 and Yvan Baudoin1 "Visual-based simultaneous localization and mapping and global positioning system correction for geo-localization of a mobile robot" *Published 15 November 2011 • 2011 IOP Publishing Ltd.*

*[31] Nusret Demir “*VARIOUS BUILDING DETECTION METHODS WITH THE USE OF IMAGE AND LIDAR DATA*” ISSN 1330-3651 (Print), ISSN 1848-6339*

*[32] Myoung-Bum Chung and Il-Ju Ko “*Classification of a Person Picture and Scenery Picture Using Structured Simplicity*” Department of Media, Soongsil University, SangDo-Dong Dongjak-Gu Seoul Korea*

[33] “Color Detection & Object Tracking” Available: https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html [Accessed: 2 July 2017]

[34] “Object detection with TensorFlow” Available: https://www.oreilly.com/ideas/object-detection-with-tensorflow [Accessed: 3 July 2017]

[35] F. Zheng, D. Schmalstieg, and G. Welch, “Pixel-wise closed-loop registration in video-based augmented reality,” *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014.

[36] D. Stricker and T. Kettenbach, “Real-time and markerless vision-based tracking for outdoor augmented reality applications,” *Proceedings IEEE and ACM International Symposium on Augmented Reality*.

[37] A. Henrysson, M. Ollila, and M. Billinghurst, “Mobile Phone Based Augmented Reality,” *Emerging Technologies of Augmented Reality*, pp. 90–109.

[38] J.-Y. Didier, D. Roussel, and M. Mallem, “A Texture Based Time Delay Compensation Method for Augmented Reality,” *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*.

[39] T. Engelke, S. Webel, and N. Gavish, “Generating vision based Lego augmented reality training and evaluation systems,” *2010 IEEE International Symposium on Mixed and Augmented Reality*, 2010.

[40] S. Gammeter, A. Gassmann, L. Bossard, T. Quack, and L. V. Gool, “Server-side object recognition and client-side object tracking for mobile augmented reality,” *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops,* 2010.

[41] H. Wechsler, “Object Recognition,” *Computational Vision,* pp. 302–365, 1990.

[42] S. S. Kumar, M. Sun, and S. Savarese, “Mobile object detection through client-server based vote transfer,” *2012 IEEE Conference on Computer Vision and Pattern Recognition,* 2012.

7.5 Appendix A - Before System Evaluation Questionnaire

Does the system lag?	
1) Yes	
2) No	
Is the process running background?	
1) Yes	
2) No	
Can the system save new objects?	
1) Yes	
2) No	
How is the battery consumption of the system?	
1) Good	
2) Average	
3) Bad	
Can the system identify objects correctly?	
1) Yes	
2) No	
Can the system correctly identify differences of geographical locations?	
1) Yes	
2) No	
Is the system generate graphic images related to the correct location in the virtual reality application?	
1) Yes	
2) No	
What do you think about overall system performance?	
1)Good	
2)Average	
3)Bad	

7.6 Appendix B – Functionality Evaluation

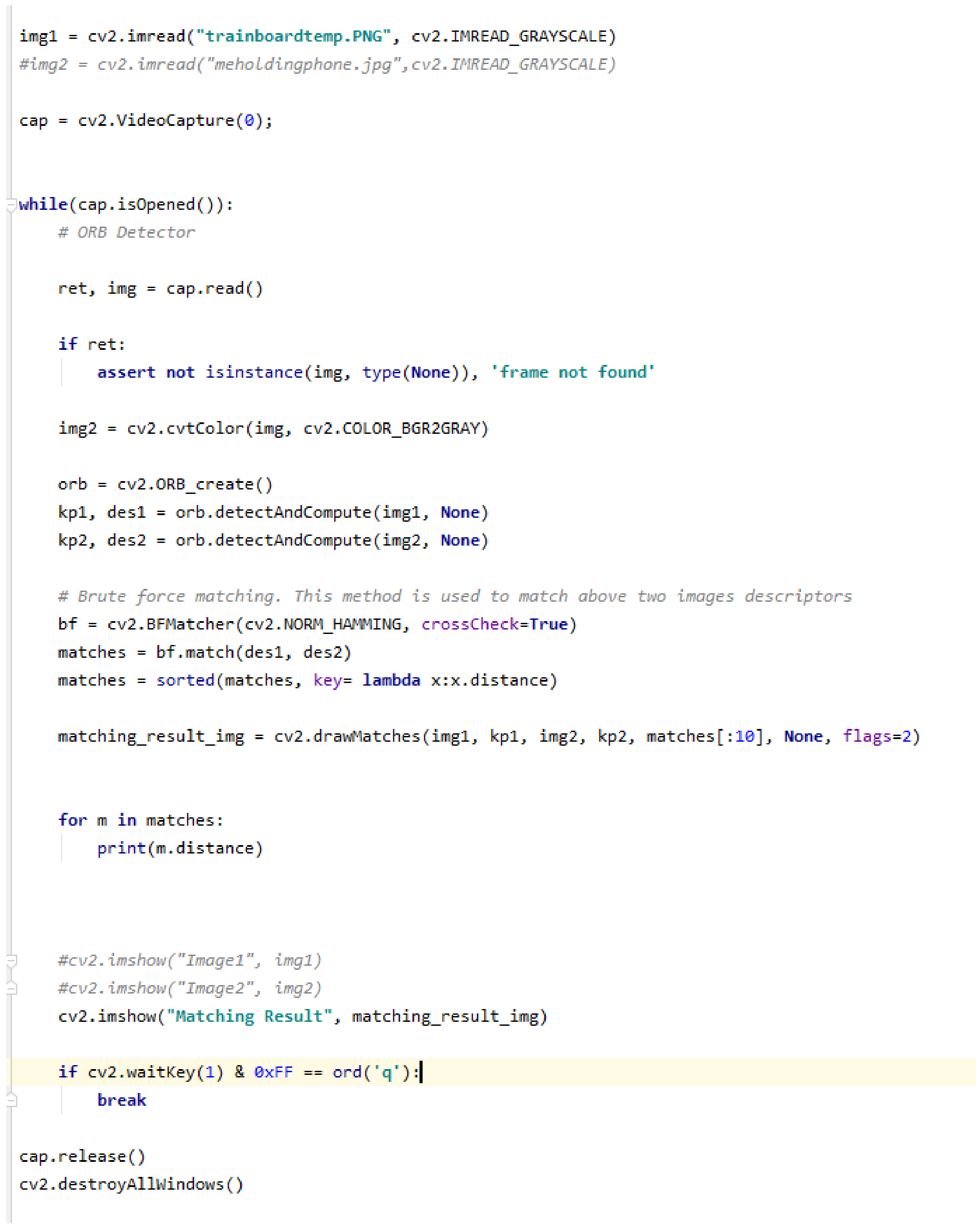
Outdoor+Indoor

Scenery No:	Accuracy	Indoor/OutDoor
1	76%	INDOOR
2	63%	INDOOR
3	78%	INDOOR
4	62%	OUTDOOR
5	70%	OUTDOOR
6	71%	OUTDOOR
7	71%	INDOOR
8	74%	OUTDOOR
9	55%	OUTDOOR
10	59%	OUTDOOR
11	67%	INDOOR
12	63%	INDOOR
13	77%	OUTDOOR
14	49%	OUTDOOR
15	48%	OUTDOOR
16	59%	INDOOR
17	50%	OUTDOOR
18	51%	OUTDOOR
19	74%	OUTDOOR
20	72%	OUTDOOR

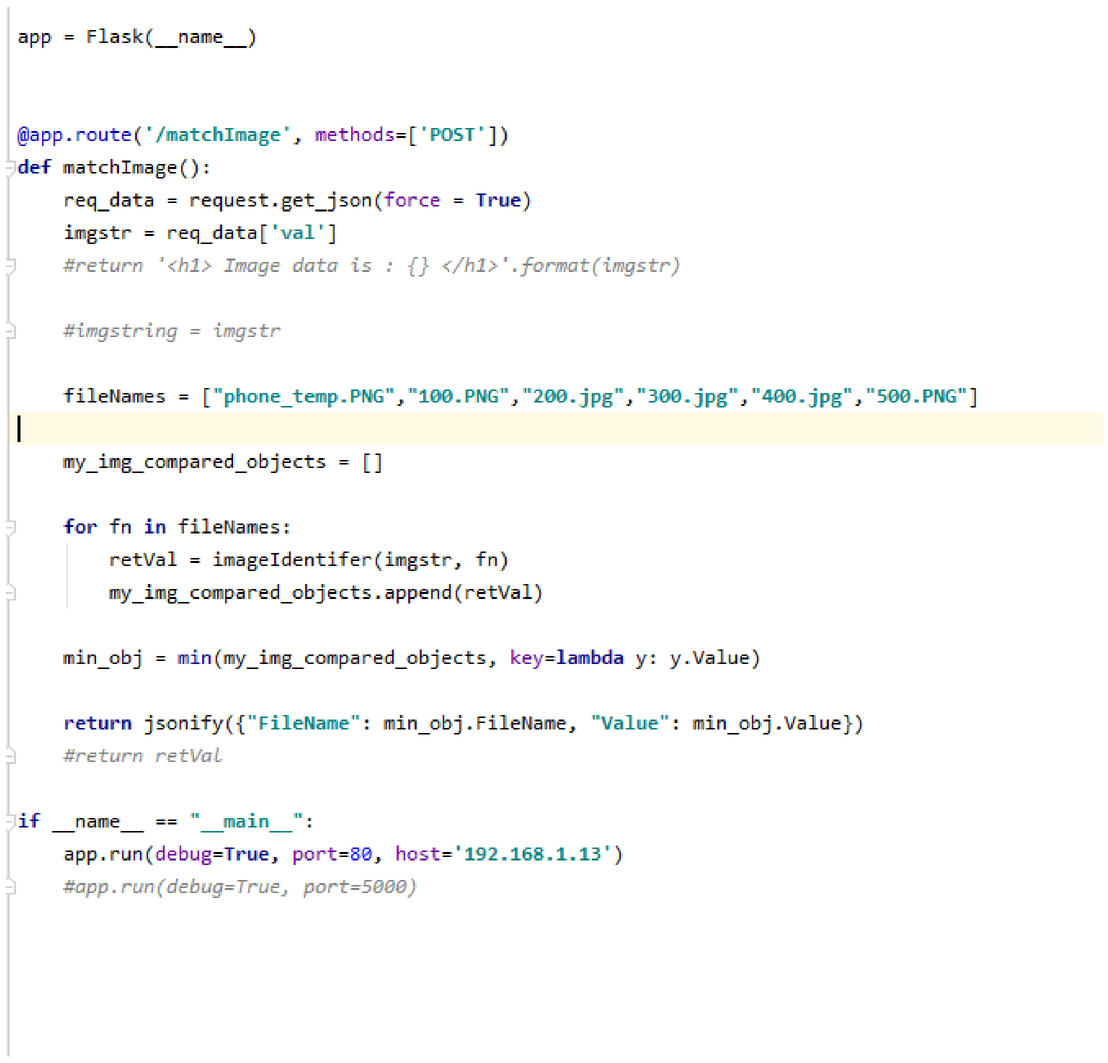
Outdoor Only

Scenery No:	**Accuracy**	**Indoor/OutDoor**
4	62%	OUTDOOR
5	70%	OUTDOOR
6	71%	OUTDOOR
8	74%	OUTDOOR
9	55%	OUTDOOR
10	59%	OUTDOOR
13	77%	OUTDOOR
14	49%	OUTDOOR
15	48%	OUTDOOR
17	50%	OUTDOOR
18	51%	OUTDOOR
19	74%	OUTDOOR
20	72%	OUTDOOR

7.7 Appendix C – Sample Server Side Implementation

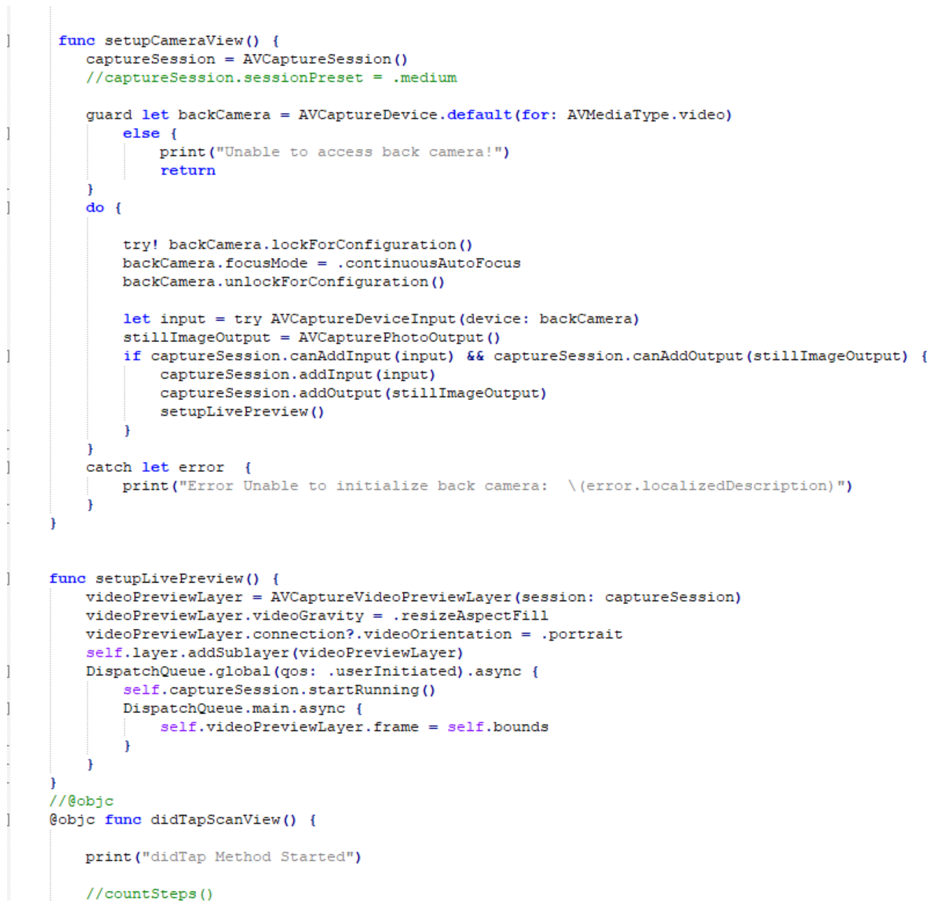


```python
img1 = cv2.imread("trainboardtemp.PNG", cv2.IMREAD_GRAYSCALE)
#img2 = cv2.imread("meholdingphone.jpg",cv2.IMREAD_GRAYSCALE)

cap = cv2.VideoCapture(0);


while(cap.isOpened()):
    # ORB Detector

    ret, img = cap.read()

    if ret:
        assert not isinstance(img, type(None)), 'frame not found'

    img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

    # Brute force matching. This method is used to match above two images descriptors
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key= lambda x:x.distance)

    matching_result_img = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)


    for m in matches:
        print(m.distance)



    #cv2.imshow("Image1", img1)
    #cv2.imshow("Image2", img2)
    cv2.imshow("Matching Result", matching_result_img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

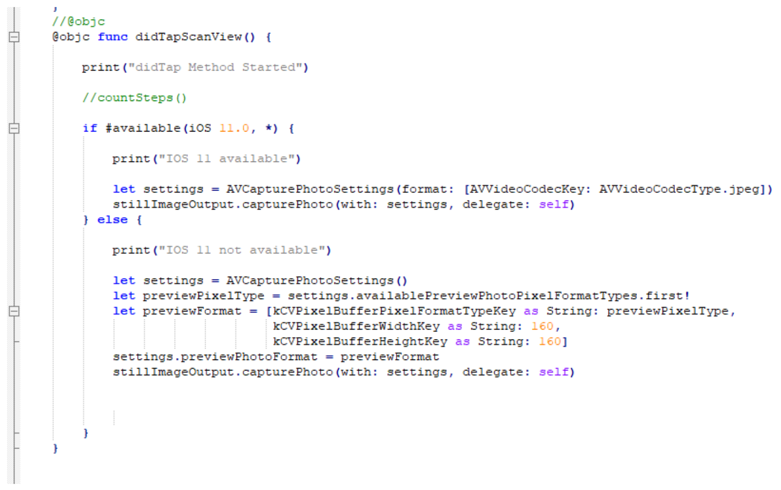


```python
app = Flask(__name__)


@app.route('/matchImage', methods=['POST'])
def matchImage():
    req_data = request.get_json(force = True)
    imgstr = req_data['val']
    #return '<h1> Image data is : {} </h1>'.format(imgstr)

    #imgstring = imgstr

    fileNames = ["phone_temp.PNG","100.PNG","200.jpg","300.jpg","400.jpg","500.PNG"]

    my_img_compared_objects = []

    for fn in fileNames:
        retVal = imageIdentifer(imgstr, fn)
        my_img_compared_objects.append(retVal)

    min_obj = min(my_img_compared_objects, key=lambda y: y.Value)

    return jsonify({"FileName": min_obj.FileName, "Value": min_obj.Value})
    #return retVal

if __name__ == "__main__":
    app.run(debug=True, port=80, host='192.168.1.13')
    #app.run(debug=True, port=5000)
```

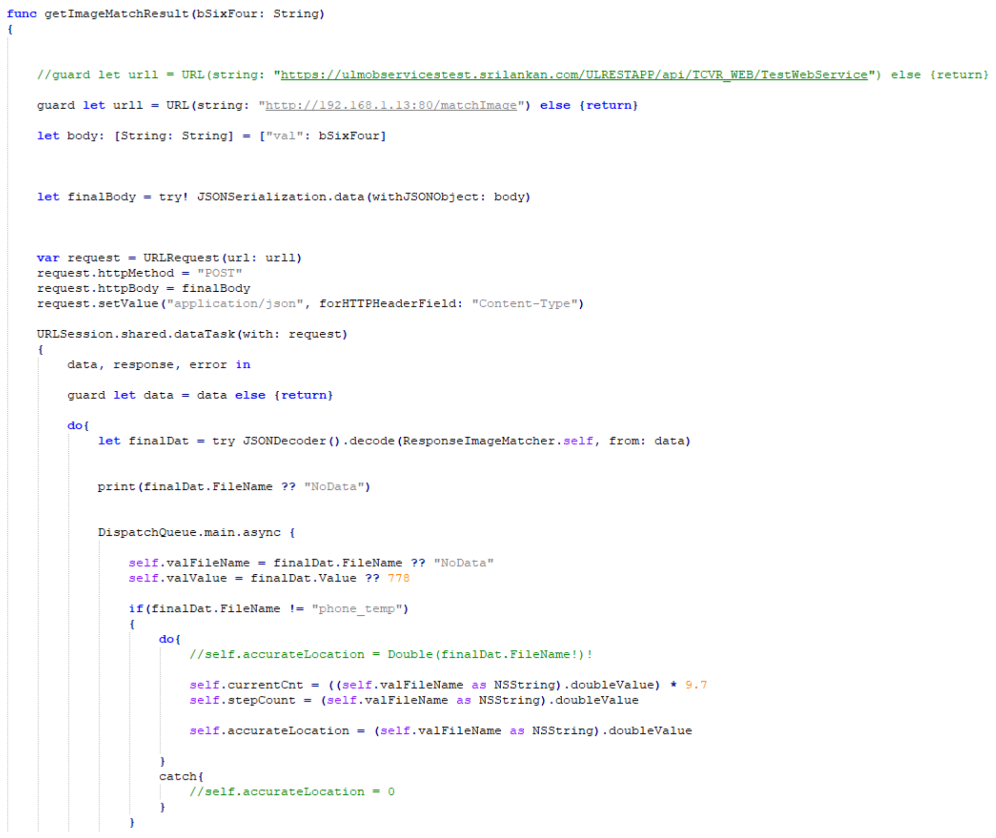
7.8 Appendix D – Sample Client Side Implementation (IOS SwiftUI)



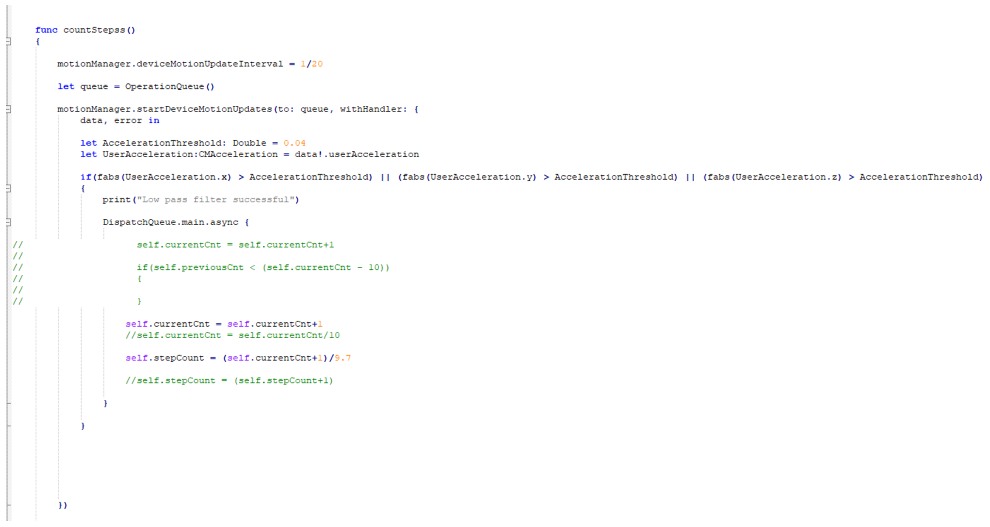
```swift
 func setupCameraView() {
    captureSession = AVCaptureSession()
    //captureSession.sessionPreset = .medium

    guard let backCamera = AVCaptureDevice.default(for: AVMediaType.video)
        else {
            print("Unable to access back camera!")
            return
    }
    do {

        try! backCamera.lockForConfiguration()
        backCamera.focusMode = .continuousAutoFocus
        backCamera.unlockForConfiguration()

        let input = try AVCaptureDeviceInput(device: backCamera)
        stillImageOutput = AVCapturePhotoOutput()
        if captureSession.canAddInput(input) && captureSession.canAddOutput(stillImageOutput) {
            captureSession.addInput(input)
            captureSession.addOutput(stillImageOutput)
            setupLivePreview()
        }
    }
    catch let error  {
        print("Error Unable to initialize back camera:  \(error.localizedDescription)")
    }
}

func setupLivePreview() {
    videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    videoPreviewLayer.videoGravity = .resizeAspectFill
    videoPreviewLayer.connection?.videoOrientation = .portrait
    self.layer.addSublayer(videoPreviewLayer)
    DispatchQueue.global(qos: .userInitiated).async {
        self.captureSession.startRunning()
        DispatchQueue.main.async {
            self.videoPreviewLayer.frame = self.bounds
        }
    }
}
//@objc
@objc func didTapScanView() {

    print("didTap Method Started")

    //countSteps()
```



```swift
,
//@objc
@objc func didTapScanView() {

    print("didTap Method Started")

    //countSteps()

    if #available(iOS 11.0, *) {

        print("IOS 11 available")

        let settings = AVCapturePhotoSettings(format: [AVVideoCodecKey: AVVideoCodecType.jpeg])
        stillImageOutput.capturePhoto(with: settings, delegate: self)
    } else {

        print("IOS 11 not available")

        let settings = AVCapturePhotoSettings()
        let previewPixelType = settings.availablePreviewPhotoPixelFormatTypes.first!
        let previewFormat = [kCVPixelBufferPixelFormatTypeKey as String: previewPixelType,
                             kCVPixelBufferWidthKey as String: 160,
                             kCVPixelBufferHeightKey as String: 160]
        settings.previewPhotoFormat = previewFormat
        stillImageOutput.capturePhoto(with: settings, delegate: self)



    }
}
```



```swift
func getImageMatchResult(bSixFour: String)
{

    //guard let url1 = URL(string: "https://ulmobservicestest.srilankan.com/ULRESTAPP/api/TCVR_WEB/TestWebService") else {return}

    guard let url1 = URL(string: "http://192.168.1.13:80/matchImage") else {return}

    let body: [String: String] = ["val": bSixFour]


    let finalBody = try! JSONSerialization.data(withJSONObject: body)


    var request = URLRequest(url: url1)
    request.httpMethod = "POST"
    request.httpBody = finalBody
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")

    URLSession.shared.dataTask(with: request)
    {
        data, response, error in

        guard let data = data else {return}

        do{
            let finalDat = try JSONDecoder().decode(ResponseImageMatcher.self, from: data)


            print(finalDat.FileName ?? "NoData")


            DispatchQueue.main.async {

                self.valFileName = finalDat.FileName ?? "NoData"
                self.valValue = finalDat.Value ?? 778

                if(finalDat.FileName != "phone_temp")
                {
                    do{
                        //self.accurateLocation = Double(finalDat.FileName!)!

                        self.currentCnt = ((self.valFileName as NSString).doubleValue) * 9.7
                        self.stepCount = (self.valFileName as NSString).doubleValue

                        self.accurateLocation = (self.valFileName as NSString).doubleValue


                    }
                    catch{
                        //self.accurateLocation = 0
                    }
                }
```



```swift
    func countStepss()
    {

        motionManager.deviceMotionUpdateInterval = 1/20

        let queue = OperationQueue()

        motionManager.startDeviceMotionUpdates(to: queue, withHandler: {
            data, error in

            let AccelerationThreshold: Double = 0.04
            let UserAcceleration:CMAcceleration = data!.userAcceleration

            if(fabs(UserAcceleration.x) > AccelerationThreshold) || (fabs(UserAcceleration.y) > AccelerationThreshold) || (fabs(UserAcceleration.z) > AccelerationThreshold)
            {
                print("Low pass filter successful")

                DispatchQueue.main.async {

//                        self.currentCnt = self.currentCnt+1
//
//                        if(self.previousCnt < (self.currentCnt - 10))
//                        {
//
//                        }

                     self.currentCnt = self.currentCnt+1
                     //self.currentCnt = self.currentCnt/10

                     self.stepCount = (self.currentCnt+1)/9.7

                     //self.stepCount = (self.stepCount+1)

                }

            }




        })
```