T.C.
MARMARA UNIVERSITY
FACULTY of ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CSE4097 - Engineering Project I
Analysis and Design Document

SCANNING REAL WORLD USING ARKIT AND APPLICATION OF PATHFINDING

Group Members
Furkan Akan - 150113012
Mevlana Ayas - 150113004
Muhammed Musa Özey – 150113079

Supervisor
Assist. Prof. Ali Haydar Özer

27.02.2018

1. Introduction

Although, we can go to any location by using navigation at the outside, there is no way to find our destination without asking people or examining plans in indoor environment. We repeat this action many times until we find our destination. So, it causes loss of time and even getting lost. In this project, we will develop indoor navigation application that people can choose their destination and can see the shortest path to it.

1.1. Problem Description and Motivation

We observe that there is not variety of works for indoor navigation. However, it is a great way to find destination looking for in indoor environments. It has also great potential to solve different problems. For example, people have difficulty in looking for specific shops in shopping center. But this difficulty can be solved by using indoor navigation which shows the path to them. In addition, instead of examining historical artifacts for a short time and skip them in museum, indoor navigation can inform people about them while guiding. Most importantly, indoor navigation can lead the way to visually impaired people with audio instructions. Therefore, we decided to make an application that can solve these problems by using indoor navigation. Figure 1 shows indoor navigation concept.

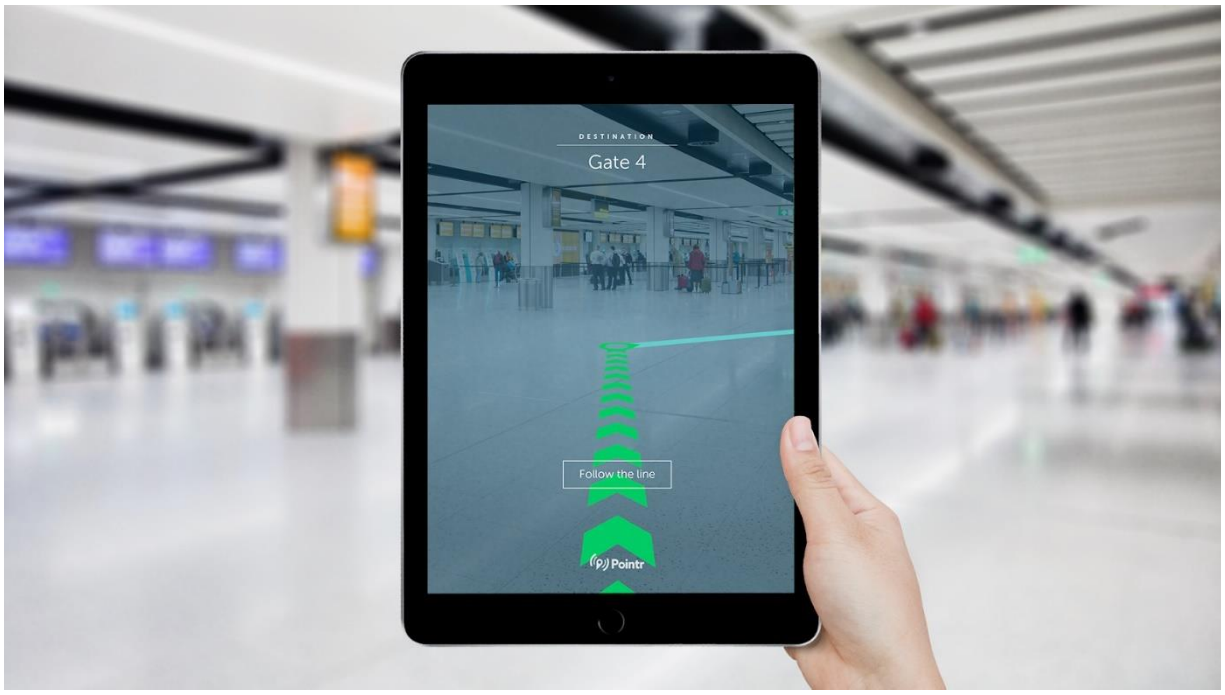


Figure 1. Indoor Navigation Concept [1]

Although, GPS allow us to navigate outside environment, it is not enough to navigate indoor environment as it doesn't provide sensitive location. But one-centimeter difference between real location and GPS provided location is important for indoor navigation.

To do this project, we will scan the indoor environment with ARKit which Augmented Reality library is provided by Apple and store data in server. Then, we will get data from the server in the application which is developed by us using Unity3D Game Engine and find our position and direction in scanned environment. After that, there will be some different specific destinations in the application and calculate the shortest path by using A* after select one of destinations. Lastly, this shortest path will be shown as Augmented Reality in the application.

1.2. Scope of the Project

Our project's first scope is capturing 3D model of environment with mobile devices. We send captured data to web server. This step requires an internet connection to send data. Analyzing data and calculating roadmap will be happened on server side. There should be no extra usage on mobile device except using camera and some sensors. We only try to analyze surfaces and environment that is static. Also, we are assuming that surfaces are flat. Pathfinding is going to work on only two dimensions, X and Y axis. It shall not make searches at Z axis. Also, lighting is an important constraint for ARKit because ARKit is modeling surfaces using camera. There will be no efficiency on model that is created without a good lighting. So, we assume that environment has enough light.

Constraints

Lighting is important for scanning, because lighting is important for camera.

Mobile devices with ARKit support. ARkit was firstly released for iPhone 6s. Supported device list: iPhone 6s and 6s Plus, iPhone 7 and 7 Plus, iPhone SE, iPhone 8 and 8 Plus, iPhone X, iPad Pro (the 10.5-inch model, 12.9-inch model and the older 9.7-inch model), iPad (2017), or newer produced phones that has IOS 11.

Static environment. This is the most difficult decision we have made. Another choice is dynamic environment. We want to add dynamic environment support to application but it requires more cpu power and time because dynamic environment can change while application is processing. When the environment changes we need to scan it and calculate new route. Scanning environment and uploading data to storage take so much time and use device resources more than usual. It is not good idea to scan environment for every changes and process it again. We don't think about the dynamic environment because trade between energy usage-time and dynamic environment support is not reasonable. We decided to develop algorithm for environment that is static and use the remaining work power to develop speed, optimality and resource usage and user friendly application.

Assumptions

Internet connection and internet-reachable area: While two or more users try to find each other, we will send location to server and retrieve location and path detail. So, we need to connect to internet. In this case we also take one of the users as a static object. One user tracks, one user waits.

Charged mobile devices: Capturing surfaces will be done with mobile devices, so in this case, the energy is a very important constraint. To save energy we are going to use mobile device for only capturing surface.

Summary

We keep an outdoor out of scope because it is so difficult to capture whole places and analyze captured data. Also, we are not going to be interested in to calculate moving objects. We are going to keep scope on static environment and indoor places. One of our scope is to collect data with device and create a data structure that shall be used by A* pathfinding algorithm. Also, related to this scope, we have another scope that is to create data structure that compatible with separate pathfinding algorithm such as Breadth-first search, Uniform-cost search, Depth-first search, Depth-limited search and Iterative deepening search.

We did not think about the using Google's ARCore or Project Tango is one of our scopes but in case of failure of ARKit, we need to add it in our scope.

1.3. Definitions, acronyms, and abbreviations

AR: Augmented Reality

ARKit: ARKit is an AR library that developed by Apple for iOS devices. It uses Visual Inertial Odometry. By using this method, it can identify and track environment successfully.

ARCore: ARCore is an AR library that is similar to ARKit. They both uses same techniques for tracking. ARCore is developed by Google.

Project Tango: It is another augmented reality library developed by Google, but it is more advanced library than ARKit and ARCore. It uses also depth sensors in addition to camera and inertial sensor. This project shows the future of AR technologies.

Odometry: It is use of data to estimate position changes over time.

Inertial sensors: gyroscope, compass, altimeter and accelerometer.

Visual inertial odometry: Visual Inertial odometry is usage of camera and inertial sensors for odometry. Using camera or inertial sensors alone can have an increasing error rate. But Visual inertial odometry uses both of them and they correct each other's error.

QR Code (Quick Response Code): is the trademark for a type of matrix barcode. [2]

Unity3D Engine: Unity3D is a cross platform game engine developed by Unity Technologies.

Unit Test: Unit Test is a test method which is written in the form of functions that will determine whether a returned value equals the value you were expecting when you wrote the function. [3]

2. Literature Survey

There are some academic works for indoor environment scanning. They mostly use Microsoft Xbox Kinect as scanning resource.

In their work Oliver, Kang, Wünsche and MacDonald [4] used Xbox Kinect to scan environment. Then they used the captured 3D point cloud data to create 3D model and they used this model for simultaneous localization and mapping. Also, they projected this 3D data to a 2D plane and made a comparison between 2D and 3D simultaneous localization and mapping.

Also, Aguilar and Morales [5] worked on a similar paper. They also used Xbox Kinect V2 to capture indoor environment. But instead of navigation, they applied their work to a robot. They used Rapidly-exploring Random Trees (RRT) to find collision-free optimal paths for robot. RRT is an algorithm that can be used for path planning. RRTs are specifically designed to handle nonholonomic constraints (including dynamics) and high degrees of freedom. [6]

There is an old work from 1999 for indoor localization prepared by Mitsubishi researches Golding and Lesh. They used accelerometers, magnetometers, temperature and light sensors for localization, but the error rate was too high. So, they worked on some machine-learning techniques to reduce error rate. [7]

Also, there are commercial works that is similar to our project. These are applications that use camera and phone sensors for indoor navigation. They also can show 3D objects, navigation arrows and some information in environment.

Firstly, Insider Navigation developed for IOS without using any Visual Inertial Odometry library. [8]

In addition, Cologne Intelligence use building's plans to create maps and use sensors placed at different points of the building to locate users in created maps [9].

Lastly, Google developed Google's Project Tango for Android devices. [10]

Different from the above projects, our project:

- Works on iOS.
- Uses ARKit which uses a Visual Inertial Odometry.
- Allow us to scan the indoor environments to create their maps and use QR code to locate users.

3. Project Requirements

3.1. Functional Requirements

3.1.1. Connection

The system consists of two subsystems which are connected each other with internet:

- Front-end system
- Back-end system

3.1.2. Transaction Selection

The front-end system provides two options to users:

- Start navigation
- Create a new map

3.1.3. Navigation

Users who select "Start navigation" shall be able to:

- Get map by scanning QR Code
- List and select destinations
- See the path from their current location to the selected destination

3.1.4. Create a new map

Users who select "Create a new map" shall be able to:

- Get unique QR Code
- Scan area
- Determine destination points
- Send created map to the back-end system

3.1.5. Server

The back-end system will be in remote server and it shall be able to:

- Stores maps of scanned areas and provide it to the front-end system.
- Create unique QR Code which gives user accessing map and initializing user's position
- Send QR Codes to the users by mail.

3.2. Nonfunctional Requirements

3.2.1. Performance

- Possible connection problems cause restarting processes therefore scanned area data will be divided 1-MB packets and they will be sent with different requests instead of one request.

3.2.2. Reliability

- Shown path may not be synchronous with real world because of ARKit library. Therefore, users will be informed about that shown destination may be at most 30 cm away from expected destination position.

3.2.3. Usability

- The purpose of the system should be easily understandable
- It must be accessible for visually impaired people.

3.2.4. Maintainability

- Different Augmented Reality libraries should be implemented easily.

3.2.5. Portability

- IOS 11 will be supported.
- Only devices with A9 or higher processors will be supported.

4. System Design

4.1. UML Use Case Diagrams for the main use cases

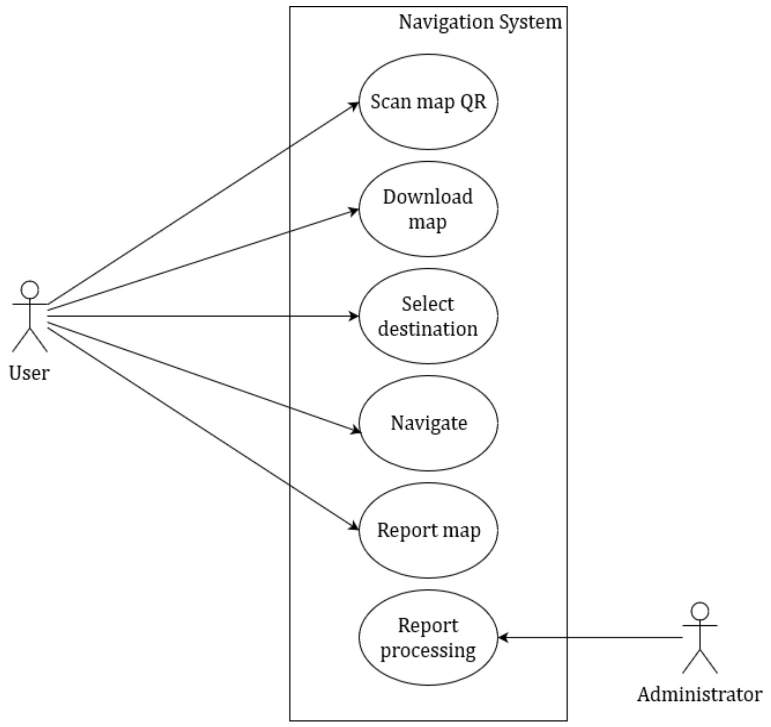


Figure 2: Use case diagram of Navigation System

User scan QR using camera. QR trigger downloading map from map storage. QR declare both map name and user start position. User should select destination point within given labels on map. Selecting label is going to trigger path calculation and path drawing on screen. If any problem occurs in this steps user can report fault.

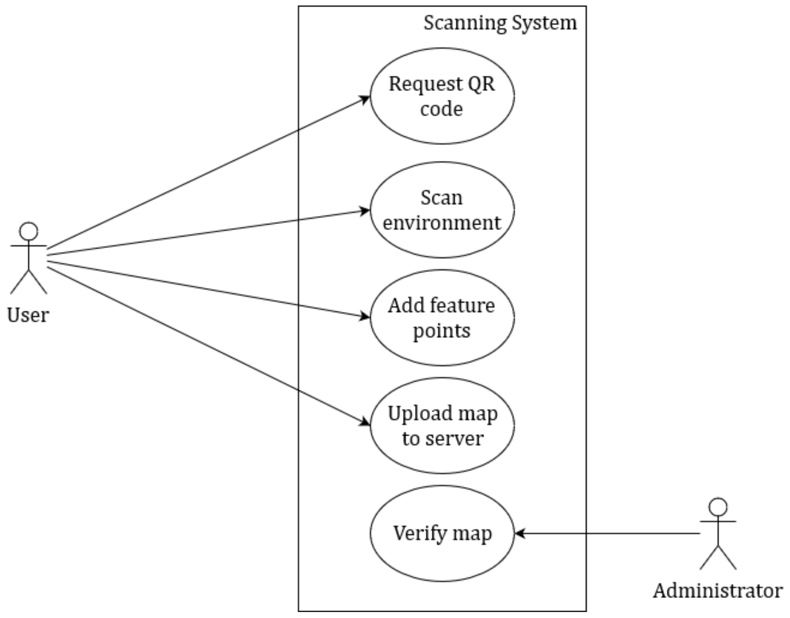


Figure 3: Use case diagram of Scanning System

Second option is to create new map. Users can create new maps and save them on map storage for further usage. First user should tell application current position and create QR code and put QR code to start position. User must start from QR to scan environment. Labels is going to be initialized in this step also. After scanning ends user send all data to storage via single upload button. If any problem occurs in this steps user can report fault.

4.2. UML Class and Database ER diagrams

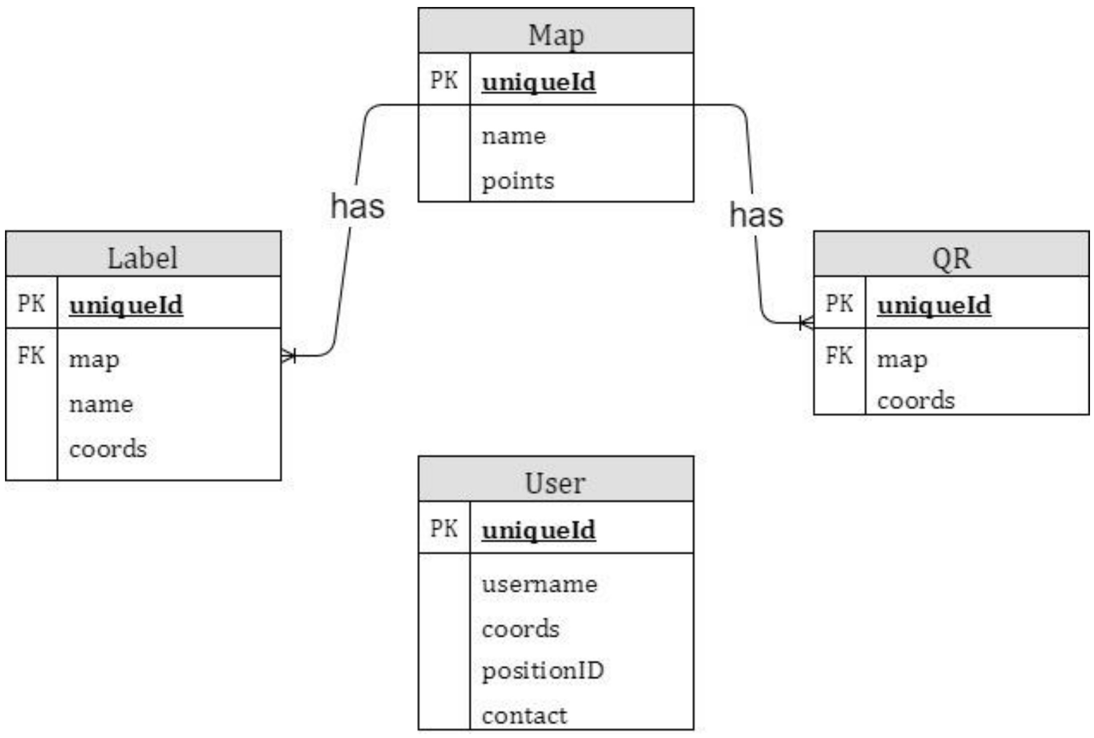


Figure 4: Database ER Diagram

Figure 4 shows Database ER Diagram. In this diagram every entity represents a table. These tables are:

- Map
    - Map represents map of scanned area.
    - Map has a unique id.
    - Map has a name.
    - Map has points which represents walkable coordinates.
- QR
    - QR represents QR Code which is scanned by user to access map and initialize their location.
    - QR has a unique id.
    - QR has a map id as foreign key. It provides accessing maps by using QR.
    - QR has a coords which represents coordinates of QR picture in map.

    - Many QR may same maps. It provides different starting points inside places instead of only one.
- Label
    - Label represents destinations which determined while creating map, and which is selected by users.
    - Label has a unique id.
    - Label has a map id as foreign key. It provides accessing destinations of map.
    - Label has a name.
    - Label has a coords which represents coordinates of destination in map.
- User
    - Users have unique id.
    - User has a coords which represents coordinates of User in map.
    - User has username.
    - User has contact to communicate with application.
    - User has positionID to define map of coords.

As can be seen in Figure 5, we are using Adapter pattern to implement ARKit, with this way we planned to create a modular system in there. So, for example to make our application compatible with Android, we can just create a new adapter with ARCore. Or in the future, we can easily implement the new technologies. To use general components such as ARKit, Network, QR Reader, we are using Service Locator pattern.

In our Main class, systems will be created and these systems can easily access any service. DataProcessSystem will process the data that is generated by ARKit in scanning side, then it will give it to UploadSystem, so it can send it to our server. In navigation side, DownloadSystem will download the map and it will give it to LoadMapSystem.

In our core design, we are mostly planning our code with Data Oriented Design instead of Object Oriented Design.

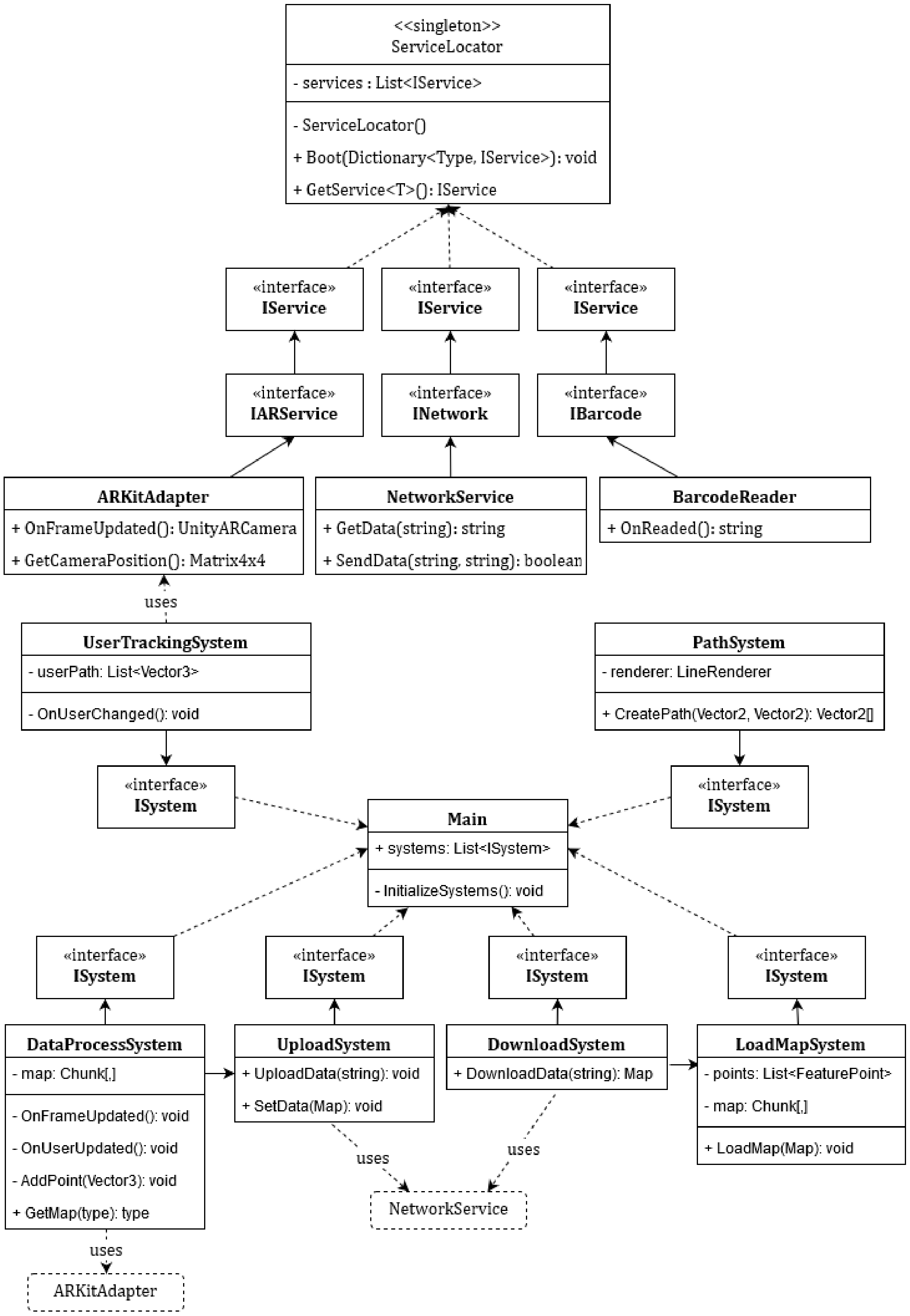


Figure 5: Main class diagram

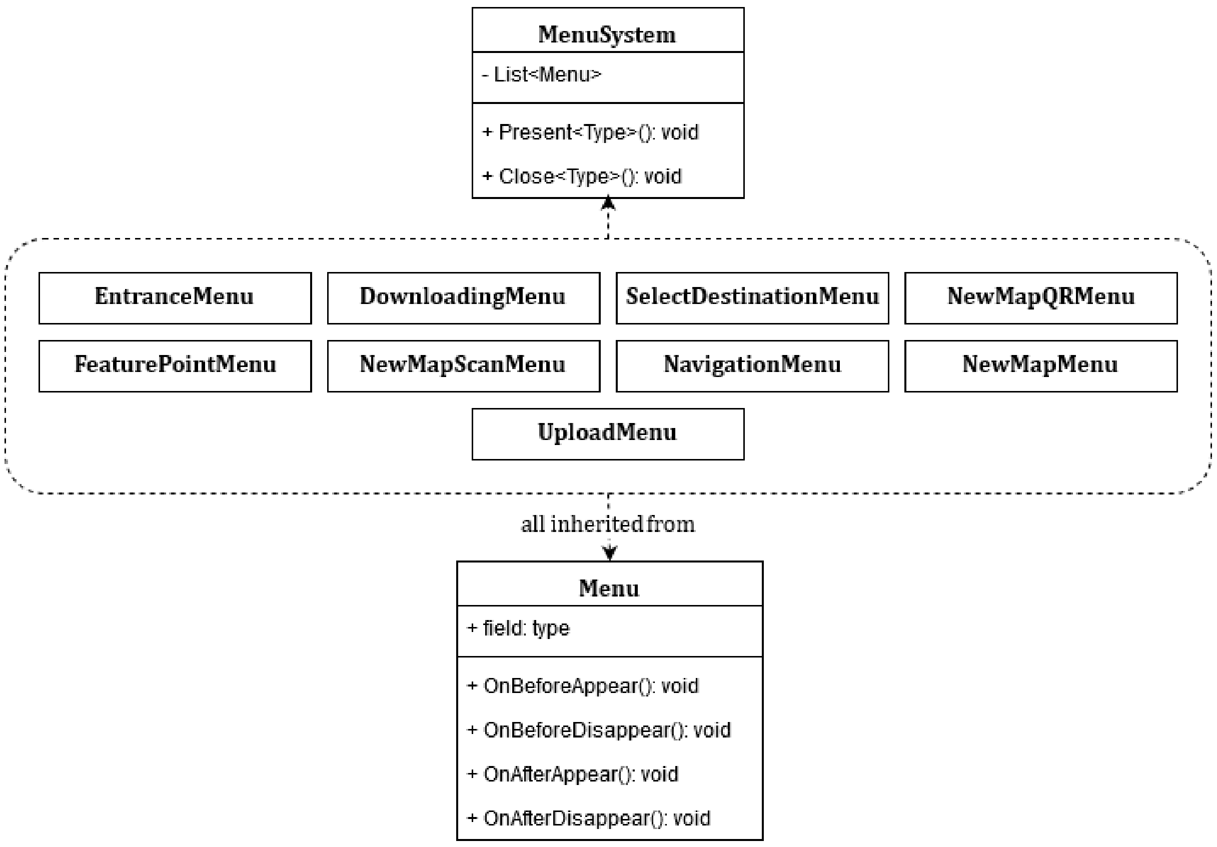


Figure 6: Menu class diagram

4.3. User Interface

User interface consist of different screens. These screens are “Main Screen”, “Select Destination Screen”, “Navigation Screen”, “Create QR Code Screen”, “Read QR Code Screen”, “Area Scan Screen” and “Upload Map Screen”.

Users access screens through the application installed on iPhone. Each screen contains different GUI elements such as input fields, buttons, text views, list views. In addition to them, some of screens use phone’s camera.

When users open the application, they will see “Main Screen”. They will see other screens depending on their selection on “Main Screen” which has two different action.

- Navigation Side
    - Select Destination Screen
    - Navigation Screen
- Creating New Map Side
    - Create QR Code Screen
    - Read QR Code Screen
    - Area Scan Screen
    - Upload Map Screen

- **Main Screen (Figure 7)**

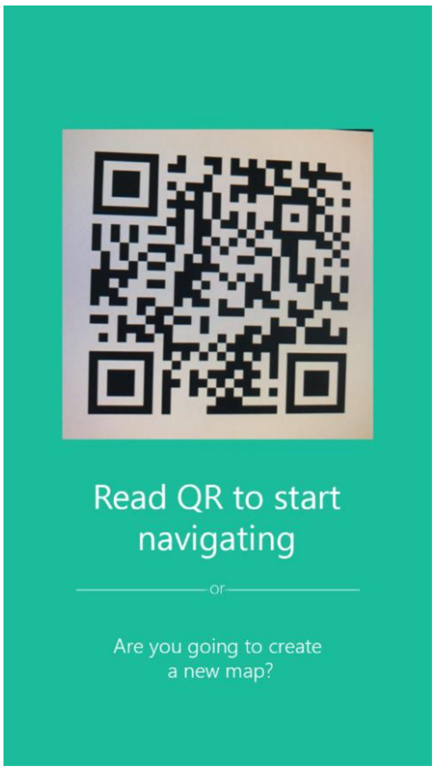

Read QR to start navigating
or
Are you going to create a new map?

Figure 7: Main Screen

If user reads a QR code, navigation side of application starts. If user taps "Are you going to create a new map?" button, creating new map side of application starts.

- **Navigation Side of Application**

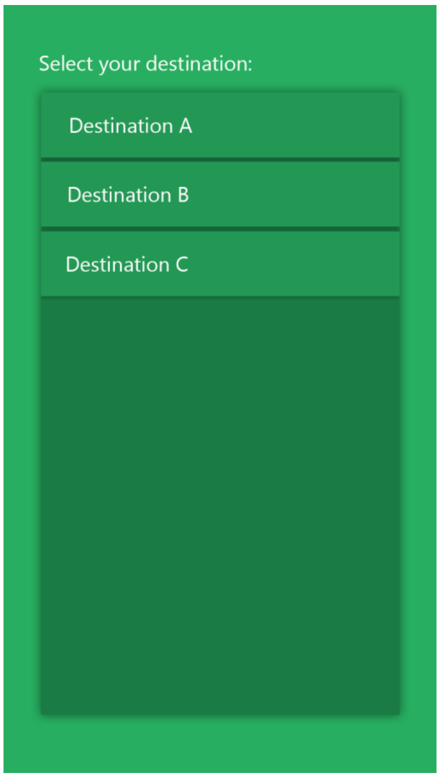


Figure 8: Select Destination Screen

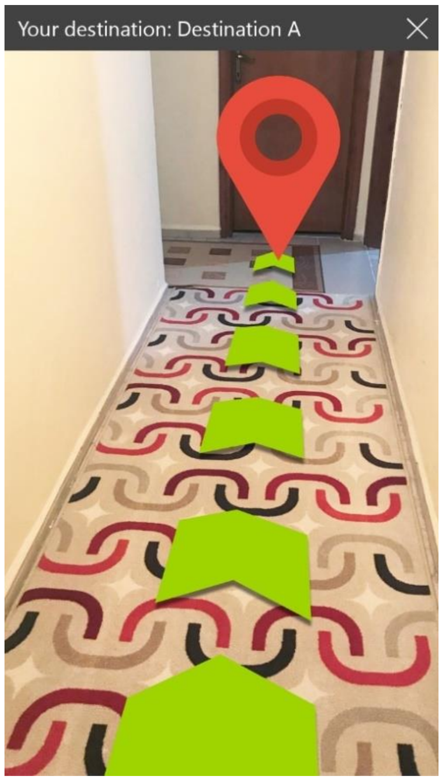


Figure 9: Navigation Screen

- **Select Destination Screen (Figure 8)**
  In this screen, destination points are listed. User can select destination point by tapping.
- **Navigation Screen (Figure 9)**
  After selecting destination point, we are going to create a path from user's position to destination position and it will be rendered to screen.

- **Creating New Map Side of application**

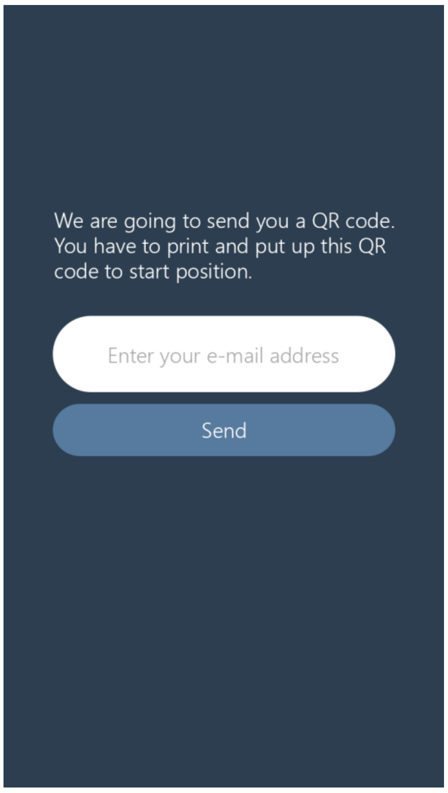

We are going to send you a QR code. You have to print and put up this QR code to start position.

Figure 10: Create QR Code Screen

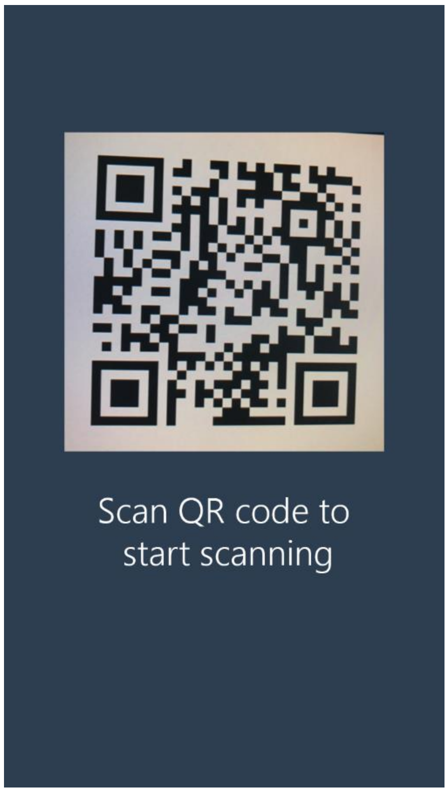


Figure 11: Read QR Code Screen

- **Create QR code screen (Figure 10)**

In this screen, we want an email address from user. We are going to send an email to user that contains a QR code. This QR code is going to contain unique map id. Users have to print this QR code to a paper and put up it to start position of map.

- **Read QR code screen (Figure 11)**

After putting up QR code to start position, user is going to read that QR code by our application. This is going to start scanning phase.

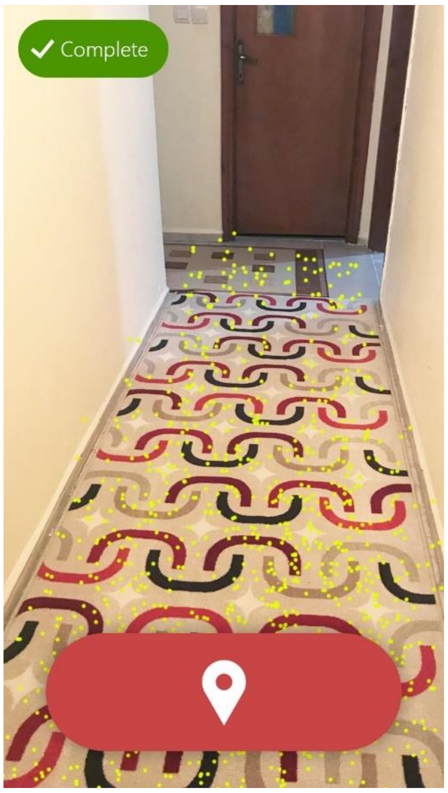


Figure 12: Area Scan Screen

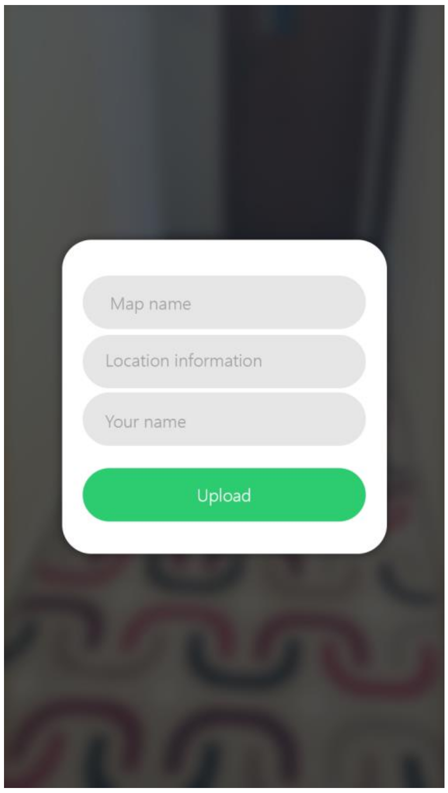


Figure 13: Upload Map Screen

- **Area Scan Screen (Figure 12)**

In this screen, user is going to start scanning. There will be camera view and two buttons, add new feature point and complete scanning buttons. When user tap adds new feature point button, if it is a valid position to add a point, our application is going to ask user for name and explanatory note of the feature point. After completing scanning, user is going to tap complete button and this will open upload screen.

- **Upload Map Screen (Figure 13)**

Upload screen is last step of scanning phase. In this screen user is going to define a name for map, some additional information about location and user's name and surname. After that user will tap upload button and application is going to send map, feature points and map information to our server.

4.4. Test Plan

Test plan consist of three parts:

- Back-end system
- Front-end system #Navigation
- Front-end system #Create a new map

Back-end system test will be made with unit test method. We assume that back-end system gets inputs from front-end system as expected. Therefore, we pass correct inputs to the functions in the back-end system and observe results.

Although, we use unit test in back-end system tests, we don't use it in front-end system tests. Because inputs which are passed in back-end are generable by us. But outputs which are generated by back-end and will be used as input in front-end systems are more complex to generate by us. Therefore, front-end system tests which need communicate with back-end system will be made after back-end system tests are successful.

Tests will be made after the developments that meet the requirement of test case are finished.

4.4.1. Back-end system

Test Case #1

Test Case Name:	Generate Unique QR Code
Requirement:	System generate unique QR Code
Procedure:	Pass parameters which are coordinates and id which represents map in database, to "generate QR Code function"
Expected Results:	Unique QR Codes should be generated with different parameters and same QR Codes should be generated with same parameters

Test Case #2

Test Case Name:	Send an Email
Requirement:	System send an email to users
Procedure:	Run "send an email" function by passing specific mail address and specific mail content
Expected Results:	Email should be arrived to passed email address with passed content

Test Case #3

Test Case Name:	Combine and Store
Requirement:	System combine parts of map and store it
Procedure:	Run “Combine parts” functions in different time by passing different walkable area coordinates and map it
Expected Results:	System should combine walkable area coordinates with passed map id and update database after all combination. Last version of combined data should be shown in database table.

4.4.2. Front-end system #Navigation

Test Case #1

Test Case Name:	QR Code Scanning
Requirement:	System shall allow user to scan QR Code
Procedure:	Scan QR Code with iPhone’s camera
Expected Results:	System accepts the user to scan QR Code

Test Case #2

Test Case Name:	Select Destination
Requirement:	System shall allow user to select destination
Procedure:	After scanning QR Code, destinations are listed and select one of them
Expected Results:	System accepts the user selection

Test Case #3

Test Case Name:	Location Synchronization
Requirement:	System should synchronize user location on the map
Procedure:	After scanning QR Code, wait until next screen is opened
Expected Results:	Map will be shown augmented reality and it should be same with walkable areas of scanned area.

Test Case #4

Test Case Name:	Path Finding
Requirement:	System should find a path from user location to selected destination
Procedure:	After selecting destination, wait until path is found
Expected Results:	Path should be shown inside walkable area and it should be shown from user location to destination even while user moves.

4.4.3. Front-end system #Create A New Map

Test Case #1

Test Case Name:	Get QR Code
Requirement:	System should give user unique QR Code
Procedure:	Select “create a new map” on main screen, fill the email address input field and send it to the back-end system
Expected Results:	System should create unique QR Code and send it to the entered email address

Test Case #2

Test Case Name:	Area Scanning
Requirement:	System shall allow user to scan area
Procedure:	Scan area with iPhone's camera
Expected Results:	Scanning points on the area should be shown on screen with yellow point

Test Case #3

Test Case Name:	Add Destination Point
Requirement:	System shall allow user to add destination point
Procedure:	Select "add destination point" button, select point on the area
Expected Results:	Selected point should be shown as augmented reality

Test Case #4

Test Case Name:	Upload Map
Requirement:	System shall allow user to upload map to the back-end system
Procedure:	Select "complete" button, fill the map name, location information, your name input fields, click "upload" button.
Expected Results:	Front-end system should send request to the back-end system, "successful" message should be shown if front-end system get response, else "error" message should be shown.

5. Software Architecture

5.1. Data Flow

In scanning phase, user requests QR code by sending email address to server and server sends this QR code via email. When user starts scanning, mobile phone of user provides two data to ARKit, camera data and inertial sensors data. ARKit creates point cloud and user movement estimation by using these data. With these data, we generate map of environment. Also, user adds feature points to map. After completing scanning, our application compresses these data and uploads it to server. Data flow diagram of these steps is shown at Figure 14.

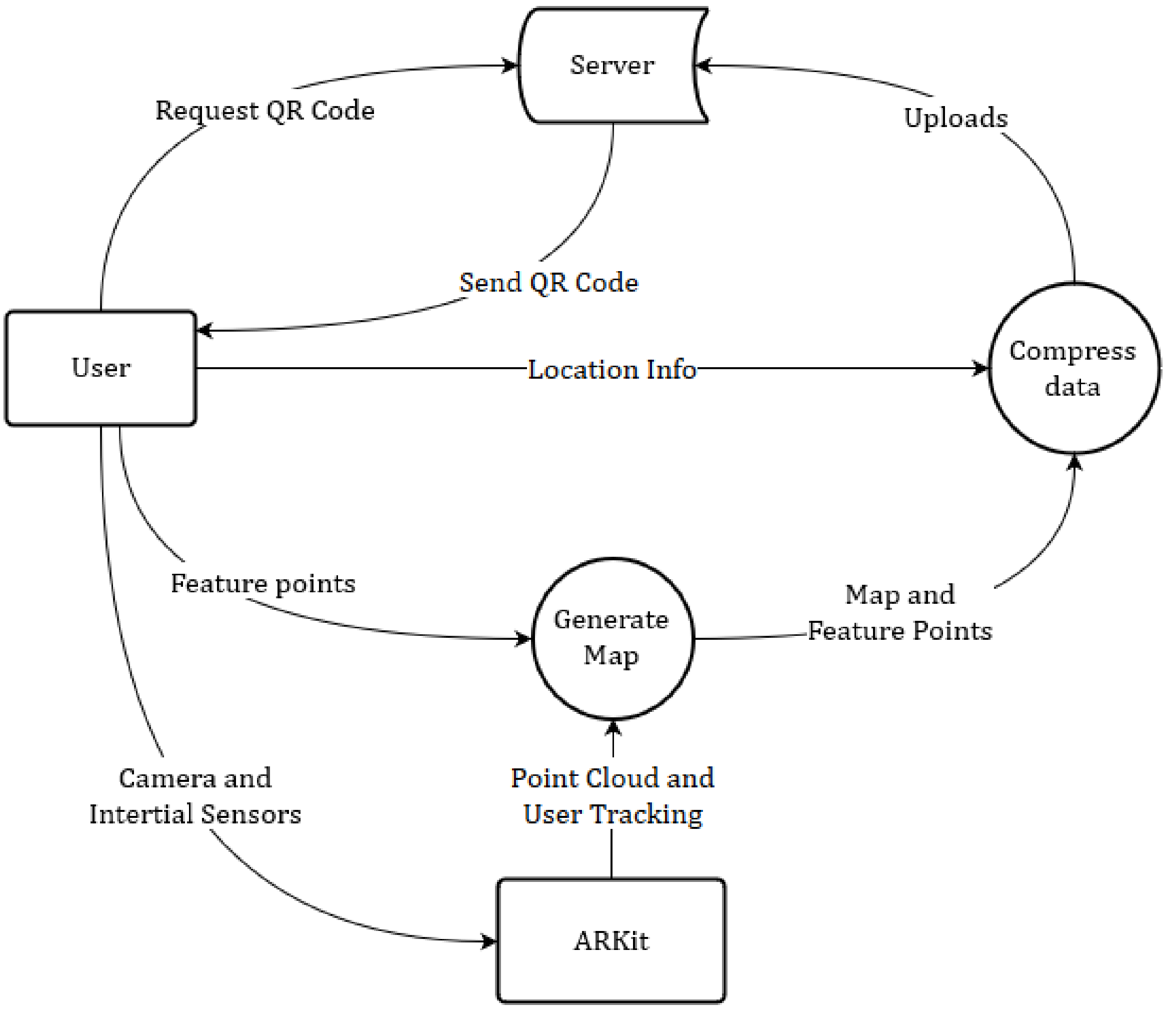


Figure 14: Data flow of scanning phase

In navigation phase, user request map data from server with map id that read from QR code. Server sends map to user and application loads this map. When user selects destination point, application plans best path from user's position to destination position and renders this path on screen. User also provides camera and inertial sensor data to ARKit and ARKit estimates user movements with these data as scanning phase. By using these data application calculates user's position and when user arrives destination point, application informs user about destination point. Data flow diagram of navigation phase is shown at Figure 15.

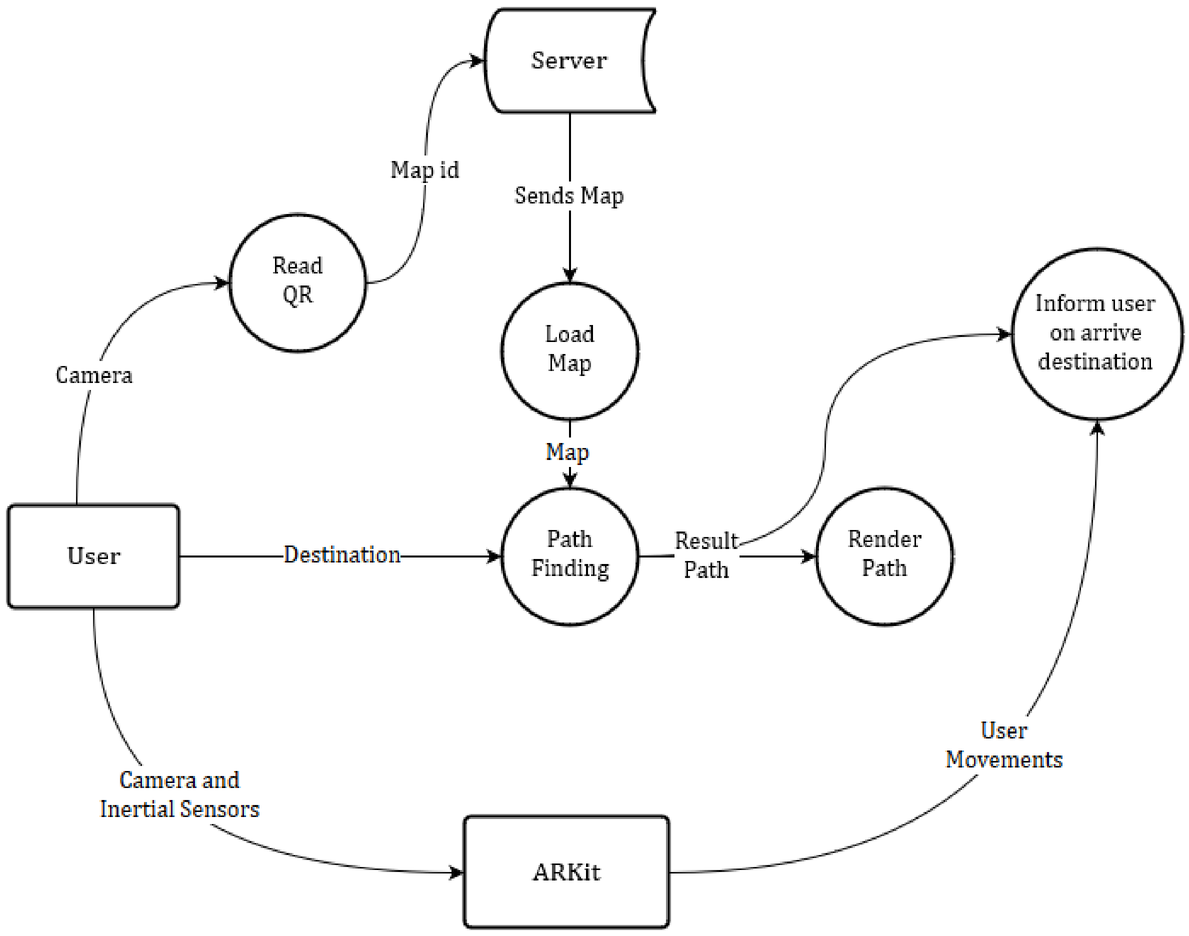


Figure 15: Data flow of navigation phase

5.2. Control Flow

ARKit provides two important information, point cloud and user movements in virtual space.

To use these data, we are going to create a reliability system on scanning. Every node in map will have a reliability value. This value will be calculated with two ways:

a. Scan by user movement

User movements are more reliable source for mapping since all of these data shows walkable areas. So, after receiving user movements our system will convert data to map data and it will set reliability of these nodes to maximum.

b. Scan by point cloud

Our system is also going to use point cloud as map data. Every time a point scanned by ARKit, we are going to find its node and increase reliability of this node. So, this means that more points for a node means more reliable node.

After completing capturing these two data, we are going to combine them and when we are combining, we are going to eliminate nodes that have reliability value less than %70 of maximum value.

By using these method, we are planning to have most reliable paths rather than shortest paths.

Processing ARKit data can be seen in Figure 16.

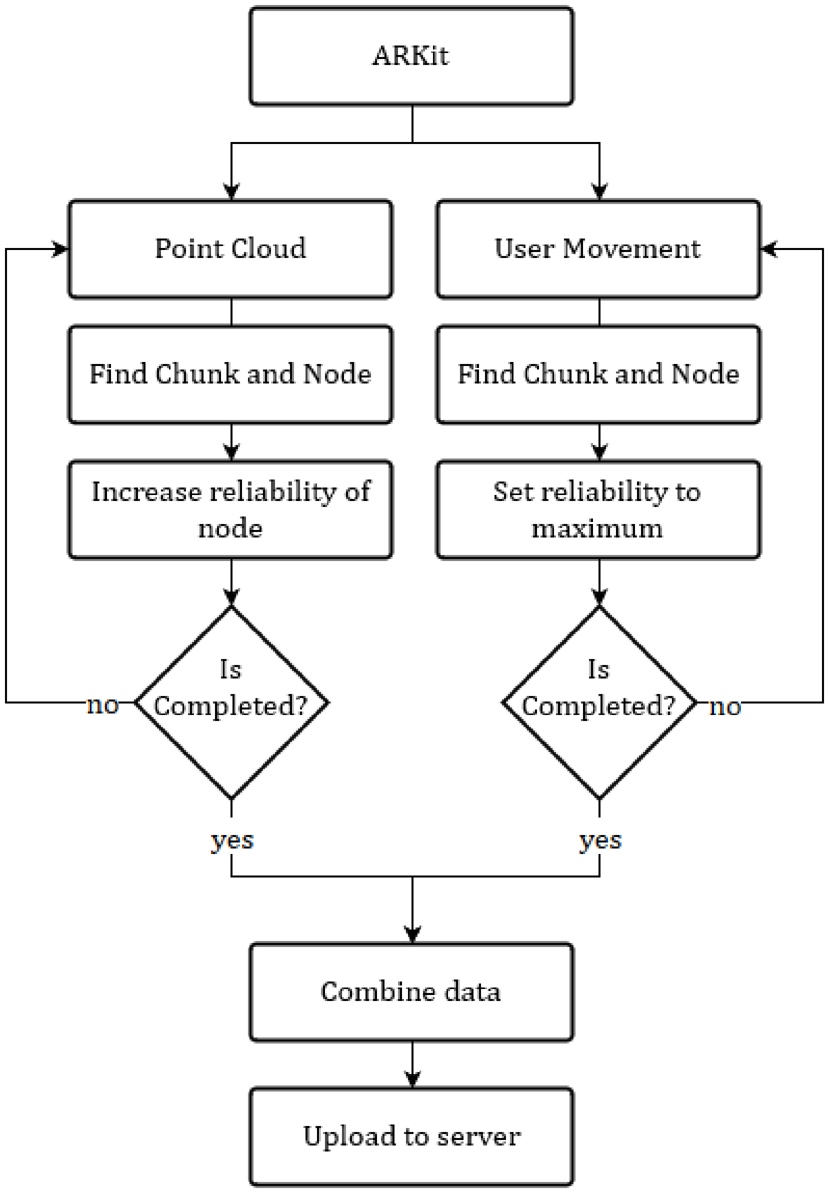


Figure 16: Control flow of processing ARKit data

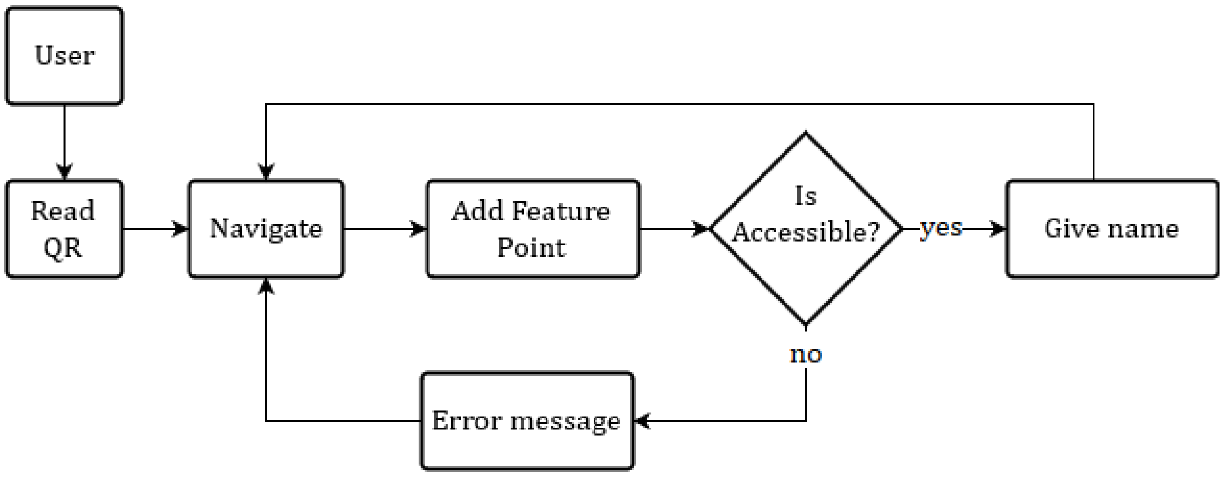


Figure 17: Control flow of scanning phase

In scanning phase, user also can add feature points. This process is shown at Figure 17. First of all, user reads QR code that given by application. This QR code defines start position of map. In navigate step, ARKit process that is shown at Figure 16 works. When navigating, if user tries to add a feature point, firstly system checks if this point is accessible from user's position. If it is not accessible, this means that we cannot create paths to this point. So, it is not a valid point and system shows an error message to user. If it is a valid point, user gives a name to this point and continues navigating.

In navigation phase, user starts with reading QR code that is put up to start position by user who scanned this place. User downloads map from server and selects a destination point from feature point list that is defined by scanner user. After that we create a path from user's position to destination point by using A* algorithm. We are using reliability of nodes as cost of nodes. So, A* creates a reliable path by using these costs.

```
initialize the open list
initialize the closed list
put the starting node on the open list (you can leave its f at zero)
while the open list is not empty
    find the node with the least f on the open list, call it "q"
    pop q off the open list
    generate q's 8 successors and set their parents to q
    for each successor
        if successor is the goal, stop the search
        successor.g = q.g + distance between successor and q
        successor.h = distance from goal to successor
        successor.f = successor.g + successor.h

        if a node with the same position as successor is in the OPEN list \
            which has a lower f than successor, skip this successor
        if a node with the same position as successor is in the CLOSED list \
            which has a lower f than successor, skip this successor
        otherwise, add the node to the open list
    end
    push q on the closed list
end
```

Algorithm 1. A* search algorithm [11]

After finding a path, we are also going to apply a line of sight algorithm to result path. By using this algorithm, we are expecting to create more realistic paths instead of unrealistic grid-based paths.

```
checkPoint = starting point of path
currentPoint = next point in path
while currentPoint->next != NULL
    if walkable (checkPoint, currentPoint->next)
        temp = currentPoint
        currentPoint = currentPoint->next
        delete temp from the path
    else
        checkPoint = currentPoint
        currentPoint = currentPoint->next
```

Algorithm 2. Line of sight algorithm [12]

Then, we are going to render this path in 3D space by using Unity3D Game Engine. When we render path, we are going to limit draw distance, so user can’t see full path.

When user arrives destination point, application shows information about feature point that is given by scanner user. After user can select new destination point again.

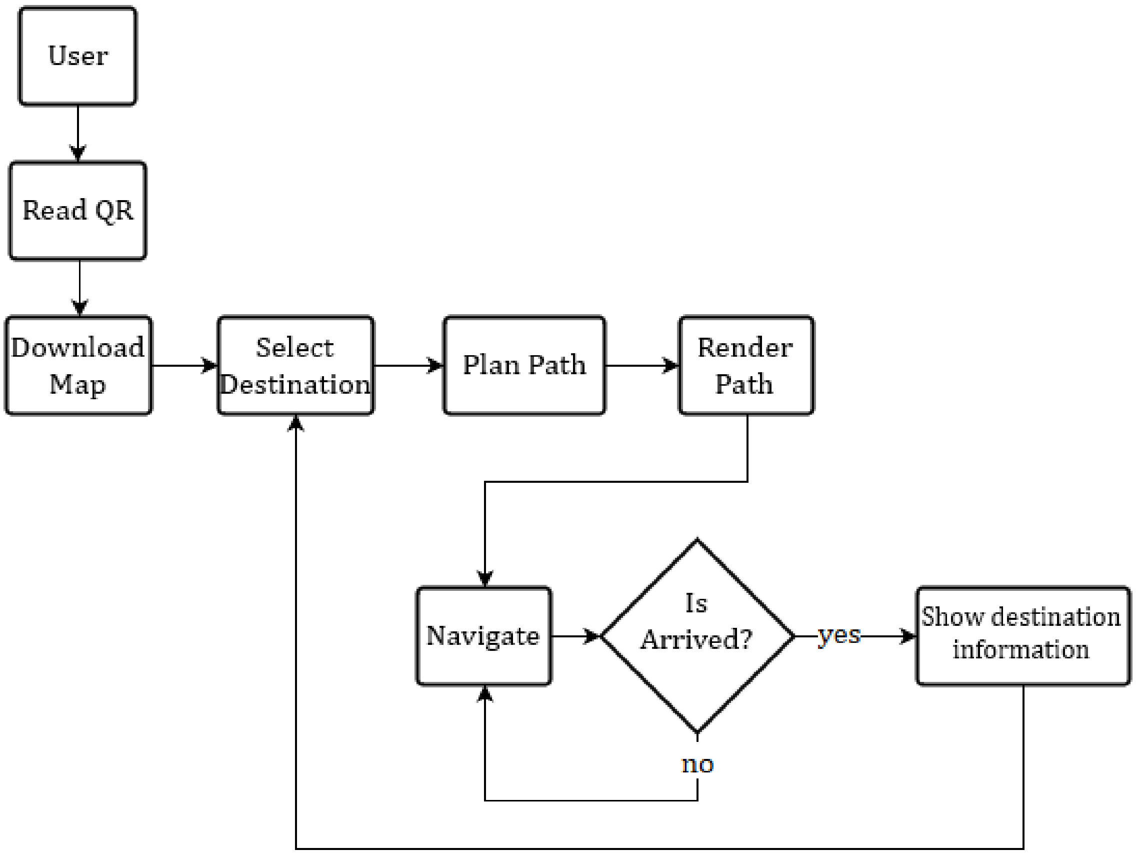


Figure 18: Control flow of navigation phase

6. Tasks Accomplished

6.1. Current state of the project

We continued with the plan that we mentioned in our Project Specification Document. We completed all the tasks in time. Tasks we accomplished are:

- Reading data from ARKit

  We are able to read point cloud and user movement data from ARKit. In Figure 19, we captured a room by using ARKit and visualized it in Unity3D Game Engine.

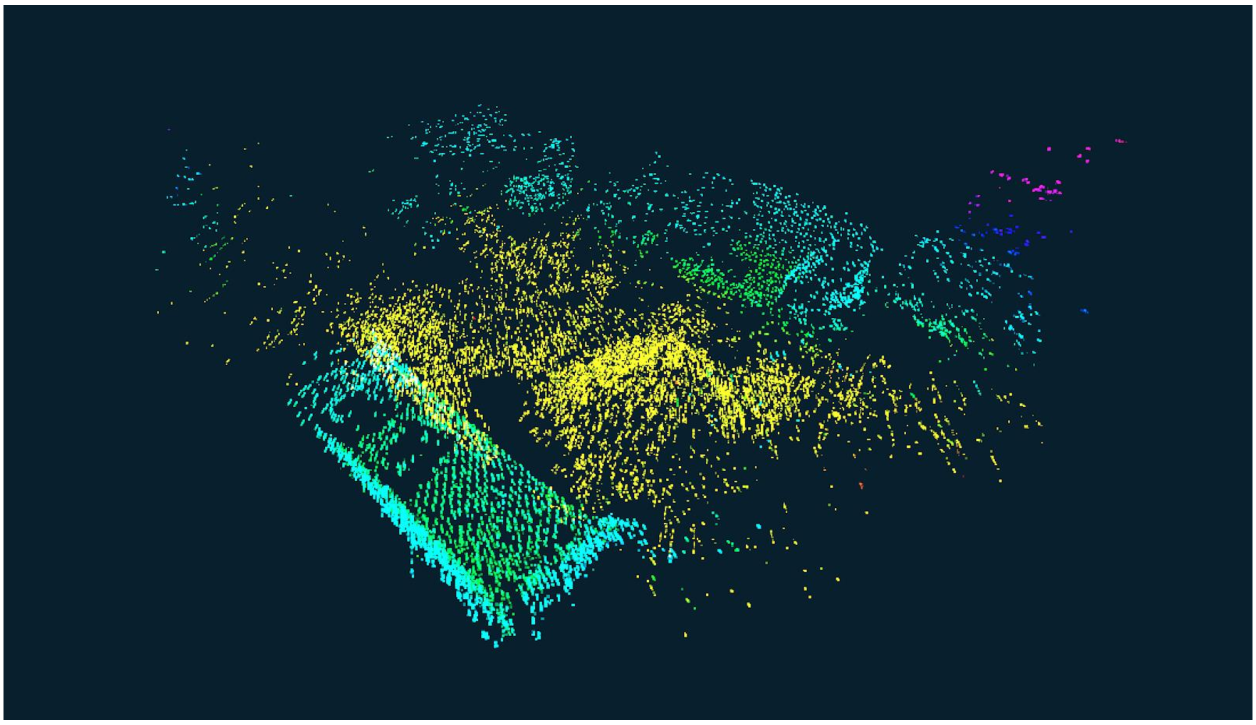


Figure 19: Visualization of a room's point cloud

- Processing this data

  After reading data, we tried to project it to a 2D array to create map of environment. There were two main problem in this step.

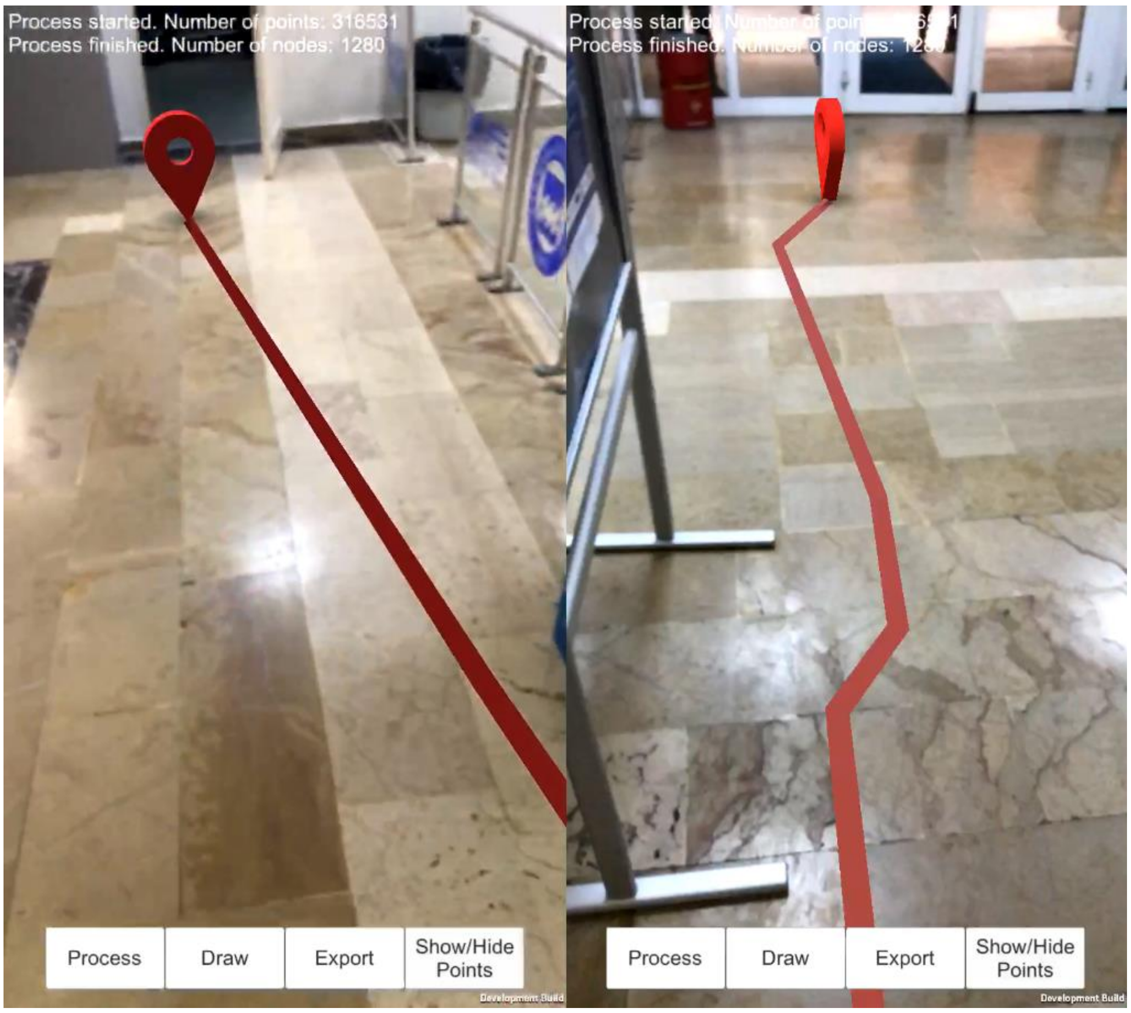
  First one is finding Z axis of floor. To overcome this problem, we take the average of Z axis of all points. This gave us an estimated value of Z axis of floor and we accepted all points that are in this average value with a small tolerance.

  Second problem is size of point cloud data. Processing all points that recognized by ARKit at the same time took some time. In larger spaces this process can take a lot more time. So, at the beginning, we tried to send data to a computer, process it there and send it back. But it didn't work either. Because file size was also too big. For example, for 240.000 points, file size was around 7.5 megabyte. So, we decided to process point clouds, when we are scanning. We are going to split an environment to parts and we are going to place all points to these parts immediately.
- Selecting start and end point and applying basic pathfinding algorithm between these points

  We also implemented small demo to data we processed. In current state, user can select start point and end point and create path between them. We used Flood Fill algorithm and basic line of sight algorithm to calculate path in current state, but we are going to change them in the future.

- Rendering result path to screen

  After creating path, we rendered it on 3D space. In Figure 20, example screenshot can be seen. This screenshot is from current state of project and all capturing, processing and pathfinding is done in runtime.



```
Process started. Number of points: 316531
Process finished. Number of nodes: 1280
```

Figure 20: Screenshot from demonstration video of presentation

6.2. Task Log

- Reading 3D point cloud data and user movements from ARKit.
  20 October 2017, 21 October 2017 - 8 Hours Total
  We successfully read point cloud and user movement data from ARKit by using an iPhone.
- Mapping this data in *Unity3D* Game Engine.
  20 October 2017, 10 November 2017, 11 November 2017 - 7 Hours Total
  We visualized the point cloud data in Unity3D Game Engine.
- Projecting 3D data to a 2D plane.
  11 November 2017, 12 November 2017 - 15 Hours Total

We projected point cloud to a 2D array. This was the hardest task we have done yet. Our solution gave some good results, but we still need to improve it.

- Implementation of pathfinding using A* search algorithm on the map that generated at previous phases.
  16 December 2017 - 4 Hours Total
  In this task, instead of A* search algorithm we implemented more basic search algorithm for fast prototyping.
- Implementation of showing generated path in user's phone.
  17 December 2017, 18 December 2017 - 7 Hours Total
  We implemented path rendering and prepared a demonstration video of our current state to show in our presentation.

6.3. Task Plan with Milestones

Description of task phases:

Phase 1: Reading 3D point cloud data and user movements from ARKit.
Phase 2: Mapping this data in *Unity3D* Game Engine.
Phase 3: Projecting 3D data to a 2D plane.
Phase 4: Implementation of pathfinding using A* search algorithm on the map that generated at previous phases and applying line of sight algorithm on it.
Phase 5: Implementation of showing generated path in user's phone.
Phase 6: Implementation of adding 3D objects to captured map.
Phase 7: Sending captured map to server and receiving it back
Phase 8: Using received map again.
Phase 9: Synchronization of user's location and received map by using QR code.
Phase 10: Implementation of user interfaces.
Phase 11: Overall testing and bug fixing

Timeline with milestones:

	October	November	December	January	February	March	April
Phase 1	🟩						
Phase 2	🟩	🟩					
Phase 3		🟪					
Phase 4			🟥				
Phase 5			🟨				
Milestone 1							
Phase 6				🟧			
Phase 7				🟦			
Phase 8				🟩	🟩		
Phase 9					🟨	🟨	
Milestone 2							
Phase 10						▨	▨
Phase 11							🟩
Milestone 3							

7. References

[1] Are you ready for Augmented Reality? Blog Post from Pointr Labs. [Online]. Available: http://www.pointrlabs.com/technology/augmented-reality/ready-augmented-reality/ (Access Date: 10.01.2018)

[2] QR code Wikipedia page. [Online]. Available: https://en.wikipedia.org/wiki/QR_code (Access Date: 10.01.2018)

[3] "What is unit testing and how do you do it?" Stack Overflow Website. [Online]. Available: https://stackoverflow.com/questions/652292/what-is-unit-testing-and-how-do-you-do-it (Access Date: 10.01.2018)

[4] Oliver, A., Kang, S., Wünsche, B.C. & MacDonald, B.: Using the Kinect as a navigation sensor for mobile robotics, Published in IVCNZ '12 Proceedings of the 27th Conference on Image and Vision Computing New Zealand Pages 509-514 (2012)

[5] Aguilar, W. G. & Morales S. G.: 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms (2016)

[6] Lavalle, S.M.: Rapidly-Exploring Random Trees: A New Tool for Path Planning (1998)

[7] Golding, A.R., Lesh, N.: Indoor navigation using a diverse set of cheap, wearable sensors (1999)

[8] Insider Navigation Website. [Online]. Available: http://insidernavigation.com/ (Access Data: 19.10.2017)

[9] Cologne Intelligence Website. [Online]. Available: https://www.cologne-intelligence.de (Access Date: 17.12.2017)

[10] Google just showed me the future of indoor navigation article by Tom Warren. [Online]. Available: http://mashable.com/2016/02/22/project-tango-indoor-navigation (Access Date: 17.12.2017)

[11] Pathfinding using A* (A-Star) Rajiv Eranki, 2002. [Online]. Available: http://web.mit.edu/eranki/www/tutorials/search/ (Access Date: 10.01.2018)

[12] Toward More Realistic Pathfinding by Marco Pinter, 2001. [Online]. Available: https://www.gamasutra.com/view/feature/131505/toward_more_realistic_pathfinding.php (Access Date: 10.01.2018)