UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
GRAPHICS, MULTIMEDIA AND VIRTUAL REALITY



# Indoor and Outdoor Campus Navigation

**Graduate:**
Ioana Alexandru

**Thesis supervisor:**
Lecturer, Dr. Alexandru Grădinaru

Bucharest
2022

*One of the early challenges that most students have to face is campus navigation. In a university with multiple buildings spread out around the city, each hosting tens of different classrooms and endless corridors, figuring out when your next class is supposed to start can be rather difficult. While tools like Google Maps can be useful in navigating between some of the campus buildings, they stop helping once you need to navigate inside the buildings.*

*This paper proposes an interactive campus map of University POLITEHNICA of Bucharest (UPB) which offers indoor and outdoor navigation support, while being integrated with existing timetable and schedule management systems.*

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

A simple Google search for "getting lost on campus" nets almost 400,000,000 results - hundreds of popular news articles and blog posts telling stories about getting lost on a college campus or giving tips about avoiding such a situation. While Google [1] encourages students to use their product for campus navigation, Google Maps[1] as well as similar products (Bing Maps[2], Apple Maps[3]) are general-purpose and usually don't meet the needs of students. This property is usually either because they do not have all the updated details about campus buildings or because they do not support indoor navigation for campuses[4].

Universities around the world help their students navigate by providing a printed map. Furthermore, with the rapid advancement of technology, the majority have switched to digital maps, which are often static pictures and do not provide more guidance than a traditional paper map. Some of them have built services on top of Google Maps or similar APIs. However, very few offer much-needed indoor navigation due to its challenges.

We can therefore identify the need for specialised campus navigation tools and propose an interactive campus map offering indoor and outdoor navigation and additional features.

[1] https://www.google.com/maps
[2] https://www.bing.com/maps
[3] https://www.apple.com/maps/
[4] https://www.google.com/maps/about/partners/indoormaps/

## 1.2 Environment

### 1.2.1 University campus

According to a 2007 report [2] by ARACIS (Agenția Română de Asigurarea Calității în Învățământul Superior), the campus of **University POLITEHNICA of Bucharest (UPB)** measured around 135.678m$^2$ across four different locations in Bucharest, including 68 buildings with 141 lecture halls, 103 seminar rooms and 1025 laboratory rooms.

The university website[5] does not directly provide complete maps of the campus and buildings. However, a virtual tour which includes these maps, as well as 360° panoramas of various campus points of interest (POIs), has been available[6] since December 2020. Indoor maps of the buildings are not available on the website, allegedly due to concerns that publicly available detailed room information may encourage thievery. However, some unofficial maps made by a university professor are available in the freshmen's guide[7] for the Faculty of Automatic Control and Computer Science.

### 1.2.2 Faculty application

A group of students at the Faculty of Automatic Control and Computer Science are currently working on a collaborative application for students which aims to act as an information hub, collecting data that would be relevant for students from different sources [3]. The open-source application is available on GitHub[8] and includes a timetable (fig. 1.1) where each event has an associated location (classroom, lecture hall etc.). It is based entirely on Google technologies: it uses a cross-platform mobile technology called Flutter, which allows it to run on Android and iOS devices as well as directly on the web; the database uses Google Cloud, specifically Firebase cross-platform solutions.

[5] https://upb.ro/

[6] http://turvirtual.upb.ro/

[7] https://www2.slideshare.net/vladposea/ghidul-boboculuui-de-la-facultatea-de-automatica-si-calculatoare-vers-20112012

[8] https://github.com/acs-upb-mobile/acs-upb-mobile


Figure 1.1

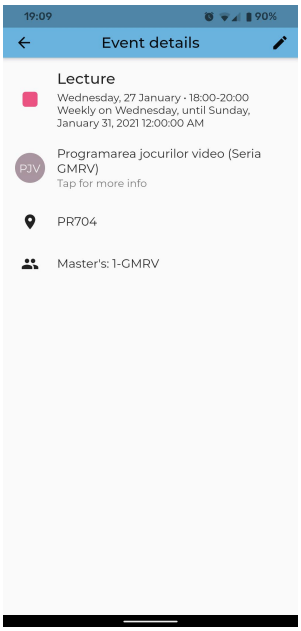


Figure 1.2

Our interactive map aims to be fully compatible with ACS UPB Mobile, such that if a user taps on a location in an event page (fig. 1.2), they would be shown the interactive map as well as navigation instructions to their desired destination.

From here on out in this paper, we will be referring to ACS UPB Mobile as *the app*. Additionally, in a similar fashion to the app's paper [3], we will refer to the larger engineering institution as *the university* or *UPB*, and to the computer science faculty as *the faculty* or *ACS*.

## 1.3 Proposed functionalities

As previously stated, the main focus of our tool is to guide students through the university campus. Our proposal as to how navigation should be achieved is described below in section 1.3.1. However, it is important to note that there is a major caveat associated with any navigation tool that works in a limited area: once the user learns to navigate on their own, the tool's usefulness decreases dramatically. A simple tool that only provides navigation would only be useful in the first few weeks of a semester while the students get used to the classes they need to be in. Subsections 1.3.2 and 1.3.3 describe additional features that can make the map useful outside of that initial time frame.

### 1.3.1 Core functionalities

At its core, the app needs to be accessible to most or all students, as well as professors and potentially campus visitors. We should not limit ourselves to a single platform, therefore we should aim for the campus navigator to be accessible both on Android as well as iOS devices. Laptops and computers are not targeted,

given that it is unlikely that someone would walk around the campus carrying a PC.

In terms of features - first and foremost, the app should provide an **interactive campus map depicting campus buildings**, and allow the user to zoom/tap on a building to view the **indoor map per floor** associated with that building. Furthermore, the **rooms should be labelled** and have an associated type (e.g. classroom, lecture hall, bathroom, office, commercial) that the user can query for. Course and seminar rooms may have additional information, such as the number of seats. Moreover, **other relevant POIs** such as coffee or food machines should also be displayed. Additionally, the user should be able to view the 360° view of an area from the **virtual tour**[9], where available. Finally, the user should be able to **save their favourite locations** as well as **select the types of markers** they wish to see on the map (e.g. hide virtual tour markers, show only bathrooms etc.).

While outdoor navigation can be easily achieved through a service such as Google Maps Platform[10], indoor navigation is difficult to achieve without specialised hardware (such as Bluetooth beacons) due to attenuation caused by construction rendering GPS unusable indoors. Since innovation in the field of indoor positioning is outside the scope of this project, the initial version of this tool (*v0*) aims to provide **navigation *without* accurate localisation**. This functionality can be achieved by allowing the user to select their starting position (listing possible options based on approximate location) and their destination. Upon starting navigation, the user can then manually mark a navigation step as complete to see the following step.

### 1.3.2 Events support

Section 1.2.2 describes the application that we wish our map to work with. One-way interaction is rather straightforward: the user presses a location name in the timetable, which queries that location in the map tool and shows a result. Similar interaction can be achieved with external apps and services as well, through intents[11] (on Android) and URL Schemes[12] (on iOS). For example, an

[9] http://turvirtual.upb.ro/

[10] https://developers.google.com/maps

[11] https://developer.android.com/training/basics/intents/filters

[12] https://developer.apple.com/documentation/xcode/allowing_apps_and_websites_to_link_to_your_content/defining_a_custom_url_scheme_for_your_app

external website or application (even a Facebook event) can display a campus location name and, if the user presses the name, open our application with the corresponding location query. Students could also use this functionality to share meeting points between them.

Two-way interaction is also possible - in the case of ACS UPB Mobile - and could significantly improve the usefulness of our map. If the map has access to timetable and event information from the app, it can display things like events happening nearby, increasing event visibility. For example, suppose a user presses on a classroom. In that case, they could see which class is currently happening there, according to the timetable[13]. Furthermore, a feature that would be particularly useful for event organisers, professors and teaching assistants would be the ability to see which class is available at a specific point in time (ideally including room size/available seating).

### 1.3.3 Additional functionalities

Attaching even more information to a location could also be helpful. For instance, a user can add a comment to a location mentioning that it is temporarily closed. Similarly, coffee and food machines could have a rating attached (users can rate them based on the quality of the product) and a label that shows whether they are out of order. Sharing documents/files/announcements in a location could also be done through integration with a project that is still under development called AirDocs. AirDocs is a middleware system that allows placing and retrieving objects or documents at different indoor locations without requiring a positioning system (see appendix A for more information).

Following Waze[14]'s example, we could also add gamification elements to our map, such as achievements for locations visited and distance travelled. Furthermore, in a similar fashion to Google Maps monthly reports, we can show statistics such as campus location where the user has spent the most time in. Furthermore, the app could support organising events such as treasure hunts/scavenger hunts on campus grounds.

[13] http://acs.pub.ro/~cpop/orare_sem2/Orar_sali_laborator_calculatoare2019_2020sem2.xls

[14] https://www.waze.com/

## 1.4 Outline

**Chapter 2** outlines the current state of the art, in terms of existing technologies, services and applications related to our project.

**Chapter 3** compares several different implementation approaches for the navigation problem.

**Chapter 4** looks at the student's opinions on the topic of campus navigation needs through the lens of our user study.

**Chapter 5** offers an overview of the learnings from our study, and discusses potential next steps.

# Chapter 2

# State of the Art

## 2.1 Positioning systems and indoor navigation

Nowadays, location tracking and outdoor navigation are commonplace, thanks to GPS technology. In order to determine the location of a mobile device, the distance between the device and several (i.e. at least three) satellites is measured, in a process called **trilateration** (fig. 2.1) [4]. For additional accuracy, a similar procedure (sometimes called **triangulation**) is done using the distance to nearby GSM (cell) towers [5].

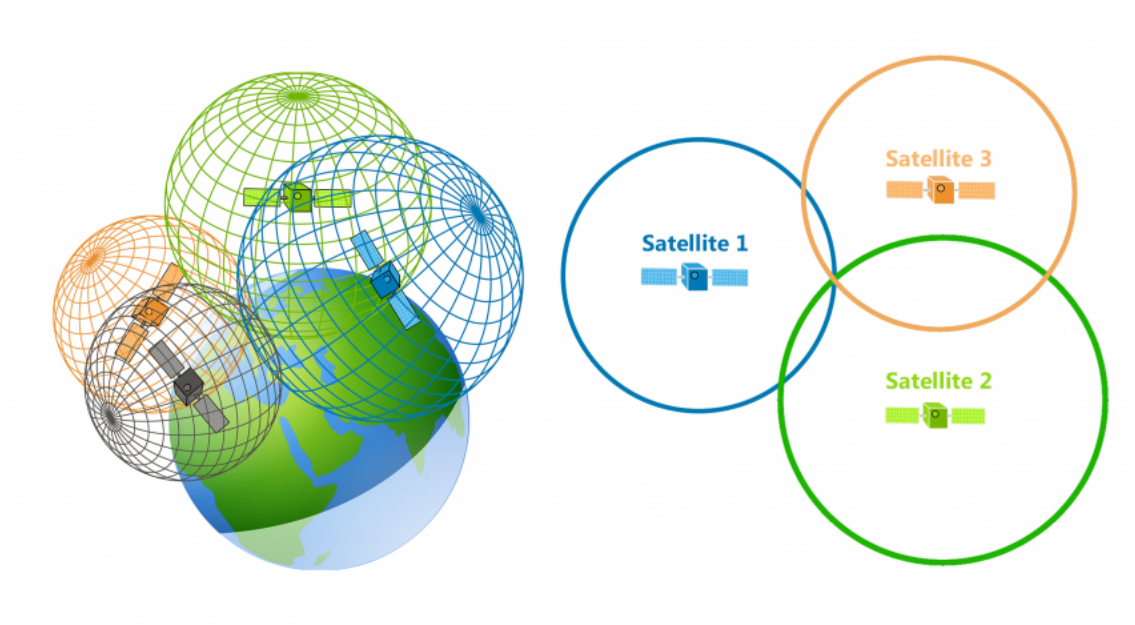


Figure 2.1: GPS trilateration

The GPS performance standard [6] promises an accuracy of <7.8m 95% of the time, with "well designed GPS receivers" achieving a horizontal accuracy of <3m. Several factors can weaken GPS signals, including mountains and thicker walls of a building. Consequently, GPS alone cannot be used in the context of indoor navigation since, in order to distinguish between different rooms, an accuracy of at least 2m would be necessary. Furthermore, vertical accuracy is approximately two times worse than horizontal accuracy, meaning that distinguishing between floors in multi-story buildings is nearly impossible using only a GPS signal.

Due to the issues underlined above, indoor navigation is a complex computing problem which requires a special network of devices used to locate a person or an object in environments where GPS technology fails, called an indoor positioning system (IPS). Any wireless signal (such as **Wi-Fi**, **Bluetooth**, **RFID** or **IR**) can be used for locating. Combining multiple different signals can lead to greater accuracy - as is the premise of the AirDocs project, described more in appendix A. Furthermore, non-radio technologies can also be used for indoor positioning. **Magnetic positioning** measures local variations in the Earth's magnetic field caused by the amount of iron inside buildings. As seen from the system described by Guido de Angelis Et Al. [7], this method can achieve an accuracy of <10cm but requires costly sensors and is not feasible in the context of a mobile device. **Inertial positioning** can use the accelerometer/pedometer of a phone to track the steps of the users and approximate the distance travelled with an accuracy of <0.5m, as described by Qiu and Mutka [8]. Finally, camera-enabled devices can use **positioning based on visual markers**, which is the technology used by Google Maps AR Navigation[1].

## 2.2 Maps APIs

The most popular Maps APIs are Google Maps[2], Bing Maps[3] and Apple Maps[4]. Out of these, Apple Maps only works on iOS and, therefore, would be a poor choice for our cross-platform application. None of the APIs provides support

[1] https://www.cnbc.com/2019/08/08/google-maps-ar-directions-released-for-iphones-and-android.html

[2] https://developers.google.com/maps/documentation

[3] https://www.bingmapsportal.com/

[4] https://developer.apple.com/maps/

for navigating inside custom buildings - indoor navigation is only allowed for specific public buildings like malls and airports.

According to Tirena Dingeldein [9], Bing offers higher image quality and a friendlier API, but Google Maps has better navigation algorithms and a better mobile experience. Furthermore, Google Maps is highly compatible with the Google-centric development environment that ACS UPB Mobile uses (see section 1.2.2).

Bryan Boyd Et Al. [10] from Texas A&M University describe a web-based navigation system based on the Google Maps API. The system does not include indoor navigation, but it accounts for different modes of transportation such as walking, driving or taking the bus. These modes are achieved through a layered roadmap (fig. 2.2), in which the roadmap is represented through a graph, where vertices are POIs and edges are roads or "transition edges" (for changing between transportation modes).

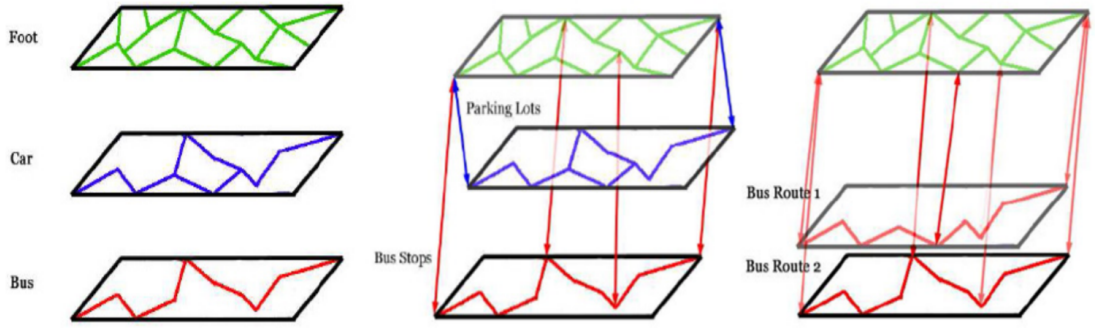


Figure 2.2: Roadmap layering, transition edges and sub-layering

## 2.3 AR indoor positioning

As previously discussed in section 2.1, accurate localization inside a building is difficult to achieve with only GPS, without additional sensors or hardware such as Bluetooth beacons or WiFi signal triangulation. A different approach to indoor navigation would be to use an AR-based solution to find the user's location (through a VPS, or Visual Positioning System) and guide them to their destination.

AR-based navigation is not a new concept, but it has only recently become

more reliable and started to find its way into the consumer market, most notably through Google Maps' Live View, first launched in 2019[11].

### 2.3.1 Visual Positioning System

While GPS (Global Positioning System) is a space-based radio-navigation system using signal from satellites, VPS (Visual Positioning System) aims to achieve positioning using an approach that is more similar to human behaviour. In the same way we look around to understand the world and our location, VPS means looking at an image of the user's surroundings and matching against its database of locations ("visual markers") to guess where the user is currently located.

Google Maps Live View's VPS works by asking the user to move their phone around them to capture the surroundings, and then comparing the camera data with its extensive Street View database in order to determine the user's location. This mostly works outdoors (and works better in daylight), but Google has slowly been expanding this functionality to some indoor spaces such as airports[12] and some transit stations in Japan[13]. One of the biggest challenges in using VPS in public spaces is the crowd - in densely circulated areas, it is unlikely that the user would be able to get a clear view of the static surroundings without a lot of people in the frame, which then have to be "erased"[14] using machine learning in order for the VPS to work its magic.

Unfortunately, the APIs used for AR navigation in Google Maps are not (yet) public. However, similar technologies by Apple - AR location markers[5] - are currently available within ARKit, the AR framework for iOS devices.

### 2.3.2 Technologies

There are currently two main platforms used for development of AR navigation solutions on mobile devices: ARKit for iOS and ARCore for Android. These development environments are responsible for translating information captured by device sensors such as the camera and gyroscope in order to draw virtual objects on the screen so that they appear to reside in the real world.

[5] https://developer.apple.com/documentation/arkit/content_anchors/tracking_geographic_locations_in_ar

#### 2.3.2.1 ARKit

According to Mobidev[15], ARKit is generally regarded as the more powerful of these two powerful mobile AR platforms, thanks to being backed up by more reliable hardware and software. Apple has full control over the production and design of its iPhone and iPad hardware as well as the OS software, whereas there is a great deal of hardware and software diversity between one Android phone to another due to the large number of different manufacturers. Because of this, ARKit's performance is better optimised.

One of the key features that puts iPhones above Android devices in the AR arena is the LiDAR sensor. This hardware can make AR navigation easier due to its depth sensing capabilities, and can reduce the need for specialised mapping hardware such as a 3D camera.

#### 2.3.2.2 ARCore

Although ARCore may not be as powerful as ARKit due to current hardware limitations and inconsistencies, it is still a very valuable platform that cannot be overlooked. Based on a survey with 200+ respondents[3], roughly 80% of students in the Faculty of Automatic Control and Computers own Android devices. Due to this majority, we cannot focus solely on an iOS-only solution, and need to find a middle ground that can work on most mobile devices.

A major advantage in focusing on ARCore is that, while ARKit only works for iOS devices, ARCore is compatible with both Android and iOS devices[6]. This makes it a good medium through which to achieve a cross-platform solution, even though it doesn't (yet) offer out-of-the-box capabilities for VPS.

#### 2.3.2.3 Other approaches

It is worth mentioning that aside from the "canonical" AR platforms, there are custom solutions (generally built on top of the above) that promise to provide AR frameworks with various features.

One such solution is used in Matthew Hallberg's demo[7] - Placenote[8] offers an AR SDK that allows users to scan their surroundings to generate a 3D model, and

[6] https://developers.google.com/ar/devices
[7] https://www.youtube.com/watch?v=VOMysKbDNxk
[8] https://placenote.com/

can be very useful for mapping an indoor space for the purpose of navigation.

Ennio Pirolo from Ambiens VR describes a different approach[9], which uses specialized ArchViz (architectural rendering) tools to create the map of the indoor space into Unity, to then use for the navigation. Specifically, it uses Revit and some special ArchViz packages for syncing Unity and Revit (AT+Sync for Unity[10] and Revit[11], AT+Explore[12]).

Yet another solution[13], from Joshua Drewlow, uses Vuforia Engine with area targets[14] - an environment tracking feature that enables you to track and augment areas and spaces, based on ARKit's location markers. This only works with an ARKit-enabled iOS device with a LIDAR sensor, in addition to a 3D camera (Matterport™ Pro2 3D) and special scanners (avVis M6 and VLX, Leica BLK360 and RTC360).

## 2.4 Existing navigation tools for our campus

As described in the previous paper[3], there have been several attempts at providing campus navigation for University POLITEHNICA of Bucharest. In April 2019, a group of students attempted to create an Android app that provides indoor navigation instructions[15], as a group project for their software engineering class. The project, however, remained in the state of a simple proof of concept (fig. 2.3), since the lack of existing digital data of the room layout in the campus buildings meant that they had to manually write the information in the form of a graph in a JSON[16] file. This process proved to be extremely tedious.

[9] https://www.youtube.com/watch?v=PQmMiB9BRYU

[10] https://assetstore.unity.com/packages/tools/integration/at-sync-v2-synchronize-from-revit-182941

[11] https://apps.autodesk.com/RVT/en/Detail/Index?id=4892322793396128687

[12] https://assetstore.unity.com/packages/tools/visual-scripting/at-explore-interactive-archviz-137202

[13] https://www.youtube.com/watch?v=F5wqfYKVbv0

[14] https://library.vuforia.com/features/environments/area-targets.html

[15] https://github.com/IrinaM09/UPB

[16] https://github.com/IrinaM09/UPB/blob/master/app/src/main/res/raw/nodes.json

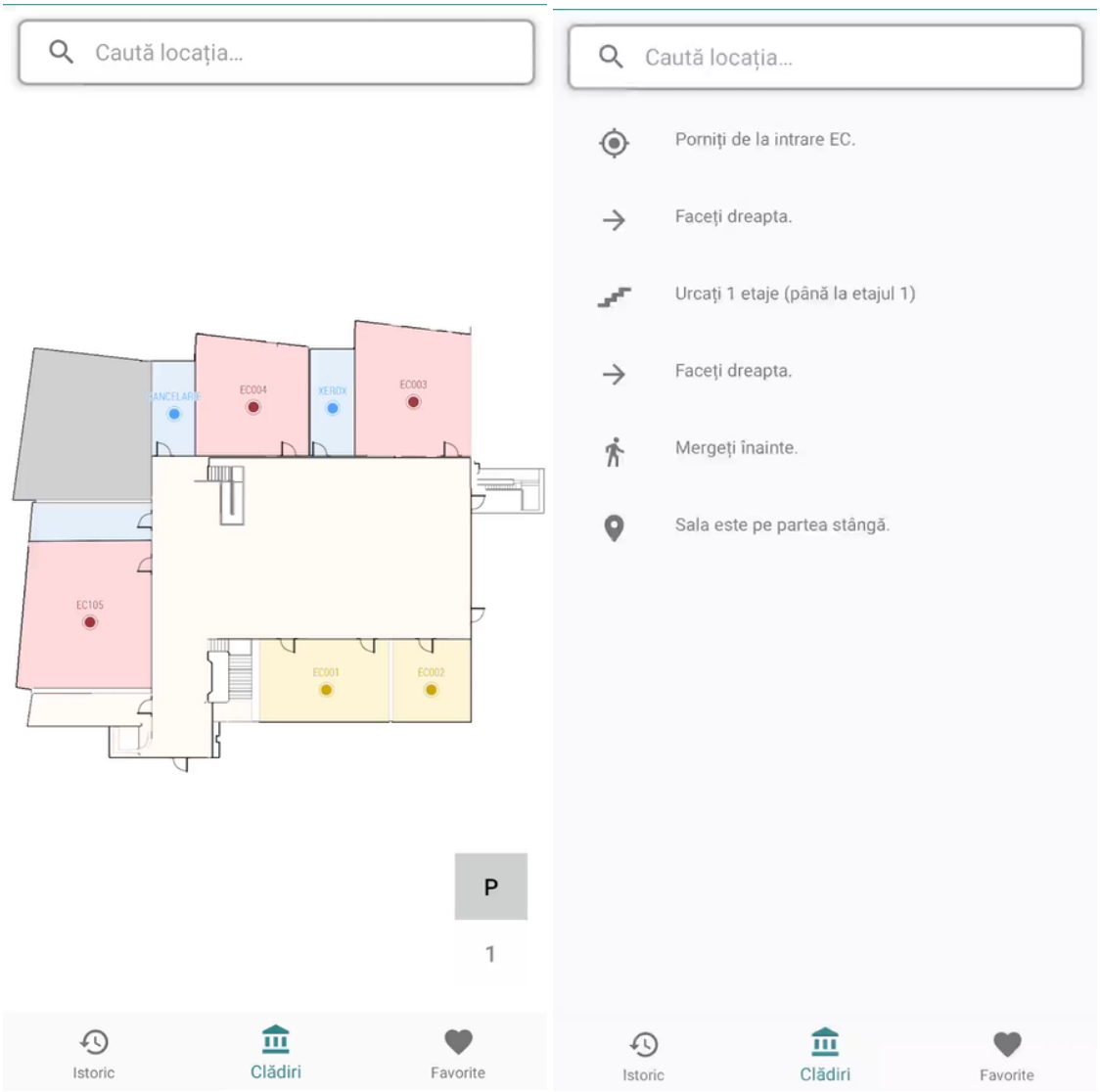


Figure 2.3: Unpublished UPB campus map application

Roughly a year later and entirely separately, a student from the Faculty of Engineering in Foreign Languages[17] within UPB, with a passion for programming, wrote a paper[16] for the UPB Students Scientific Communications Session. Consequently, they won a prize and published a campus map mobile application for both iOS and Android devices (fig. 2.4), using the Apple Maps and the Google Maps APIs, respectively (see section 2.2). It later became the official UPB app and gained many more features and a new look (fig. 2.5).

[17] http://ing.pub.ro/en/

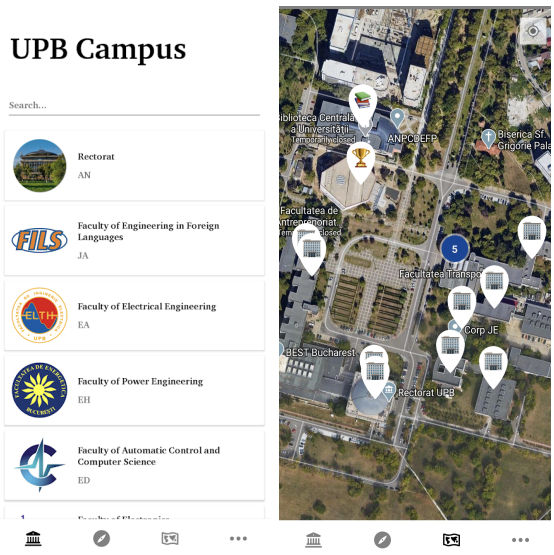


Figure 2.4: Published UPB campus map application (2020)

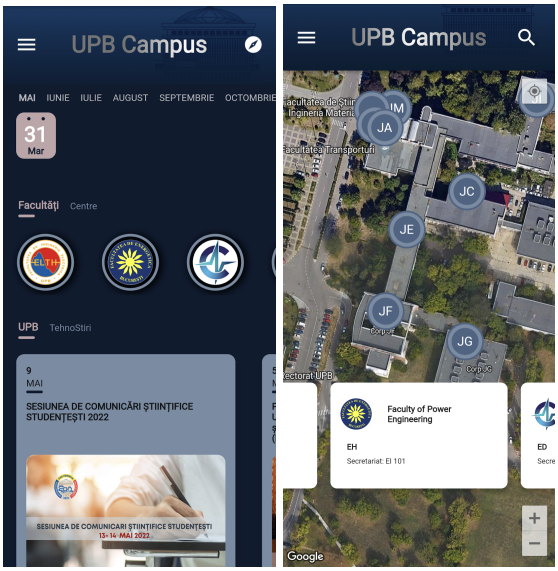


Figure 2.5: Published UPB campus map application (2022)

While admittedly not a navigation tool per se, another project that deserves a mention in this section is 3DUPB[18]. It is a computer graphics summer school involving a project that revolves around a 3D model of the Faculty of Automatic Control and Computer Science (fig. 2.6, 2.7).

Figure 2.6: 3D model of the outside of ACS (from 3DUPB)

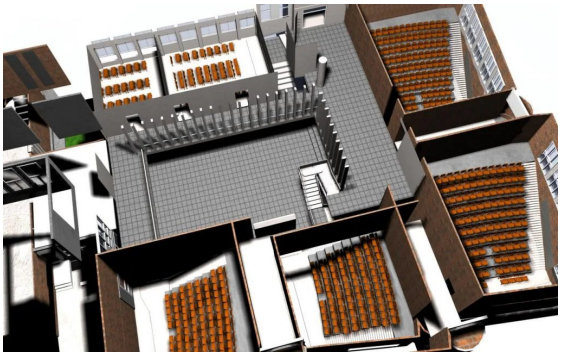

Figure 2.7: 3D model of the inside of ACS (from 3DPUB)

Given the 3D models, a 3D campus navigation system would be possible. However, several challenges are involved with such a system. The most crucial challenge is that rendering a complex 3D model consumes a lot of time and resources and may be infeasible for a mobile device. Additionally, any change in

[18] http://3d.pub.ro/

the structure of the building would require remodelling.

In 2009, Jing Wang and Zhi Lin [17] from the Asian Institute of Technology proposed a web-based 3D Campus Navigation System. In a similar fashion to the navigation system described in section 2.2, this campus navigator uses a graph to represent buildings (vertices) and roads (edges). It is described as being made up of four modules: the spatial environment module, the nodal graph module, the animation module and the behaviour module. The spatial environment module creates 3D models of each campus building in a spatial environment. The nodal graph module calculates the shortest path between the nodes defined based on the spatial environment (where each building is assigned a node near its main entrance). The animation module deals with displaying the user's 3D avatar moving through the map and showing a top-down view in the form of a mini-map. Finally, the behaviour module is responsible for user interactions.

# Chapter 3

# Implementation approaches

There are several different ways in which we can approach solving the campus navigation problem. Therefore, we would like to explore several different approaches for implementing this functionality, to compare them and decide on the best solution.

## 3.1 Alternatives considered

This section describes the variables explored when researching campus navigation solutions. In order to decide on the best alternative, we compare different approaches for things like technologies, algorithms, positioning strategies and map representations.

### 3.1.1 Integration with existing tools

As we've seen in chapter 2, section 2.4, there are already some existing solutions for offering UPB students university-related information. We would like to have a tight integration with these tools, both to avoid duplicating efforts, as well as to not overwhelm students even further with more tools that achieve the same goal.

Given that **ACS UPB Mobile** aims to be an information hub, we could potentially integrate the navigation functionality directly within the app as an additional feature. This approach could benefit from the app's involved open-source community, ensuring that the functionality will be extended and maintained over time. Alternatively, we could implement navigation as a separate app, and allow

it to be opened from **ACS UPB Mobile** via intents[1] or URLs [2]. However, even with this second approach, it would be beneficial to make use of the same Firebase endpoint in order to benefit from the data stored in the app (see the features described in 1.3.1). Furthermore, Firebase offers a Dynamic Links[3] feature, which can simplify linking between different apps.

The goals of the **UPB Campus** app are the most closely related to the goals of this study. Therefore, it comes to reason to collaborate and attempt to create a consistent user experience. In terms of integration, the easiest approach here would be again to create a separate app, and allow it to open or be opened from the **UPB Campus** app. Alternatively, there is a possibility to integrate our features directly within this app as well. However, the process would be more difficult, given that **UPB Campus** is closed-source and owned by the university.

Finally, we can use the 3D models created through the **3DUPB** program for navigation. However, simplifiying the models may be needed for smooth operation on mobile devices.

### 3.1.2 Technologies & APIs

As described in chapter 2, existing solutions are leveraging many different APIs and technologies to achieve various navigation-related functionalities. To this end, we would like to explore several different architectures and approaches in order to pick the best fit.

Firstly, we can look at the existing technologies that we would like to integrate with. Both **ACS UPB Mobile** and **UPB Campus** share a very similar technology stack, tightly-knit in the Google development ecosystem. They are cross-platform applications built using Flutter, an open-source UI software development kit developed by Google. Flutter allows having a shared code-base (written in the Dart programming language) for an application that can be compiled for multiple platforms, the most notable of which being Android and iOS. Furthermore, both apps use the Firebase platform for the back-end, leveraging services such as a non-relational database (Firestore), notifications, and cloud functions.

[1] https://developer.android.com/guide/components/intents-filters
[2] https://developer.apple.com/documentation/uikit/interprocess_communication
[3] https://firebase.google.com/docs/dynamic-links

One common denominator between the implementations described below is the chosen **game engine**: **Unity**. While our project is not a game, it requires 2D and 3D rendering capabilities that mobile frameworks on their own cannot easily perform, therefore we had to look into choosing an appropriate game engine. The two main picks were Unreal Engine and Unity, the two most popular cross-platform game engines. We chose Unity for multiple reasons. First, we want to have both a 3D and 2D version of the map, and Unity offers specialized tools for 2D game development. Second, Unity offers better support for AR functionalities, which we would like to explore for the purpose of indoor navigation. Third, Unity is compatible with Flutter through the `flutter_unity_widget`[4], and while the recommended game engine for Flutter is Flame[5], it is a minimalistic engine and not as powerful and versatile as Unity.

It is worth noting here that the primary programming language used with Unity is C#, which comes with all the features of a modern programming language and the powerful built-in Visual Studio support.

Unity also offers a platform-agnostic AR platform called **ARFoundation**. It is built on top of platforms like ARCore and ARKit (see section 2.3.2). Furthermore, it leverages common functionalities through a shared API (similar to how Flutter leverages native iOS/Android development components through a single codebase). While it also supports other frameworks (Magic Leap, HoloLens XR Plugins), we are interested in the ARCore/ARKit feature set[6] (table 3.1)..

Moreover, as we have seen in section 2.2 of chapter 2, there are several maps APIs that could be used as a source of truth for our functionality, the most notable of which being the Google Maps Platform[7].

[4] https://pub.dev/packages/flutter_unity_widget

[5] https://github.com/flame-engine/flame

[6] https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html

[7] https://developers.google.com/maps

Table 3.1: Unity feature support for ARCore/ARKit, according to documentation

| | ARCore | ARKit |
|---|---|---|
| **Device tracking** | ✔ | ✔ |
| **Plane tracking** | ✔ | ✔ |
| **Point clouds** | ✔ | ✔ |
| **Anchors** | ✔ | ✔ |
| **Light estimation** | ✔ | ✔ |
| **Environment probes** | ✔ | ✔ |
| **3D Object tracking** | ✖ | ✔ |
| **Meshing** | ✖ | ✔ |
| **2D & 3D body tracking** | ✖ | ✔ |
| **Collaborative participants** | ✖ | ✔ |
| **Human segmentation** | ✖ | ✔ |
| **Raycast** | ✔ | ✔ |
| **Pass-through video** | ✔ | ✔ |
| **Session management** | ✔ | ✔ |
| **Occlusion** | ✔ | ✔ |

### 3.1.3 Pathfinding strategy

Pathfinding has been an important topic in multiple domains such as robotics, video games and, of course, navigation. Depending on the environment that a path needs to be found in, many different strategies can be implemented, from traditional strategies such as A* and Dijkstra [18], to neural networks optimized for real-time pathfinding [19]. At its core, a pathfinding algorithm has two main requirements: the ability to travel to a specified goal in a given space, and the ability to avoid static and dynamic obstacles along the way. For our use case (campus navigation), dynamic obstacles are not a concern, therefore sophisticated real-time pathfinding algorithms are not necessary. We are only interested in taking the known path on an alley/road as well as indoors, where the obstacles to be avoided are only static (e.g. walls).

Since pathfinding is an important challenge in video games, there are several pathfinding solutions for Unity available, including the built-in NavMesh Pathfinding. While many developers implement their own solutions or use the built-in one, there are several packages on the Unity Asset Store which promise

pathfinding solutions for game developers. Some of these are specialized (e.g. A* Tile Pathfinding, PolyNav - 2D Pathfinding etc.), while others are more generic (e.g. A* Pathfinding Project, Open World PathFinding System). Among these, by far the most popular (and in the top most popular assets overall) is the A* Pathfinding Project, which offers both a full-fledged "Pro" version as well as a free option with fewer features. Table 3.2 shows a comparison of the two versions and the built-in Unity Pathfinding, based on data from the A* Pathfinding Project's website[8].

Table 3.2: Pathfinding solutions comparison

| | **A* Pathfinding Project Pro** | **A* Pathfinding Project Free** | **Unity NavMesh Pathfinding** |
|---|---|---|---|
| **Cost** | $100 | free | free |
| **Source code included** | ✔ | ✔ | ✖ |
| **Access to pathfinding data** | ✔ | ✔ | ✖ |
| **Multiple graphs** | ✔ | ✔ | ✔ |
| **Node penalties** | ✔ | ✔ | ✖ |
| **Automatic navmesh generation** | ✔ | ✖ | ✔ |
| **In-game graph recalculation** | ✔ | ✔ | ✔ |
| **Built-in editor support** | ✖ | ✖ | ✔ |

Given that the basic functionalities we are interested in are offered by both the built in solution as well as this package, and that the built in solution has official support and more available resources, we believe that the Unity Navmesh Pathfinder is the best solution for our project.

### 3.1.4 Positioning strategy

As stated previously, innovation in the field of indoor positioning is not part of the goals of this paper. However, we can explore existing state-of-the-art approaches in order to provide students with a fleshed-out user experience. We would like to visually display the user's current location on the map, even if only approximated.

[8] https://arongranberg.com/astar/

The most critical position we need is the user's starting location. Once we have that as well as the destination, we can display the navigation path. The user can then use it to navigate on their own, without requiring their permanently updated location (although it would be a quality-of-life improvement).

The most basic approach would be to simply prompt the user to pinpoint their own location on the map. This could be done through the user pressing on a specific point on the in-app map. However, this method comes with significant drawbacks. Firstly, it would require the user to actually know where they are – which, given that they need a map, is very likely not the case. Secondly, even if the user knows their current location, selecting the right spot on the map may be finicky.

A potentially more reliable way for the user to select their own current position would be for them to enter a string signifying a point of interest. We can then associate the given POI with the corresponding map coordinates, and place the user avatar accordingly.

Both approaches outlined above can be further improved by using GPS technology. Although we've already established in chapter 2 that GPS is not enough for indoor positioning, it should still provide a location accurate to approximately 5-10 meters. We can use this location to immediately estimate the user's position on the map (that the user can then further correct), or to suggest nearby POIs. Subsequently, the overhead for using the GPS location for indoor positioning is effectively zero, considering we would use it for outdoor navigation either way.

For classic map-based navigation, an approximate position may be enough, given the approaches outlined above. However, when using AR, the accuracy of the initial position may make or break the entire navigation experience. We've seen in section 2.3 that, while ARKit/ARCore can be used for the actual AR navigation, the initial positioning can still be tricky if we can only use location markers on iOS (see section 2.3.1). Since location markers and similar solutions require extensive mapping beforehand (see section 2.3.2.3), we could potentially benefit from a SLAM (simultaneous localization and mapping) solution. This involves real-time generation of a map while simultaneously tracking the agent's position within it. This can enable us to create an indoor positioning system that

doesn't require preparation beforehand, or, at the very least, requires minimal preparation as opposed to the other techniques described.

ARCore uses SLAM out of the box, so for our specific use case, a good solution would be to use SLAM to track the user's location on a known 2D map. But while ARCore can track the user's movement compared to a starting point, we still need to map this (again, critical) starting point to a known coordinate on the map. Here, we can take advantage of the fact that the user needs to open their camera for AR navigation. Therefore, one way to pinpoint the location could be by using aptly-placed QR codes at specific points in the campus, that users can scan to let the app know where their current location is. This can be especially useful for events taking place in the campus, where a QR code could be placed at the campus entrance.

### 3.1.5 Map representation

Aside from the navigation logic itself, an important problem for our project is how to represent the campus map in the app. We can draw it manually, find a way to generate it automatically, or use mixed methods.

#### 3.1.5.1 Source of truth

The "traditional" way to get the maps would be to use the official floor plan/campus images. However, these are not publicly available, and are generally inconsistent. They can be improved by manually redrawing them to be consistent and making sure they are to scale (and match the corresponding GPS coordinates). This does, however, require significant manual work if no other data is provided, as it would essentially require doing measurements by hand in the campus. Implementation #4 (see section 3.2.4) uses drawn 2D maps as the source of truth. Similarly, Implementation #2 (see section 3.2.3) also uses static, hand-crafted data in the form of campus building models from 3DUPB (see section 2.4).

Ideally, we'd also like to use some accurate cartographic data to map the GPS coordinates to the points of interest. Implementation #1 (see section 3.2.1) uses Google Maps as the source of truth by leveraging the Google Maps SDK for

Unity[9].

Implementation #3 (see section 3.2.3) explores another method, which is to use MapsModelsImporter[10], a Blender add-on for importing 3D models from Google Maps. This creates maps with a greater level of details for smaller areas, that we can then combine to create a full map of the campus.

#### 3.1.5.2 In-app representation

Now that we discussed where the mapping data could come from, we need to describe the concrete representation of this data in the app. There are two important characteristics of our maps: dimensionality (3D vs 2D maps, see figure 3.1) and level of detail. 3D maps can provide more information compared to 2D maps, but can also be more difficult to follow for some users. 2D maps can have other advantages as well, such as making it easier for users to estimate distances[20].

Figure 3.1: 3D vs 2D maps

In terms of level of detail, the more components in a map, the more resources

[9]https://developers.google.com/maps/documentation/gaming/overview_musk
[10]https://github.com/eliemichel/MapsModelsImporter

will be needed to render it, which can be particularly taxing in the context of a mobile application. From an aesthetic perspective, digital design has been shifting more and more towards a "less is more" approach, and what is called a "flat design". Therefore, maps with fewer details not only have the advantage of being easier to render (and thus speeding up an app), but they may also be perceived as "cleaner".

Ultimately, there is no ideal solution, and even Google acknowledges that and simply lets the user choose their own preference, as seen in figure 3.2.

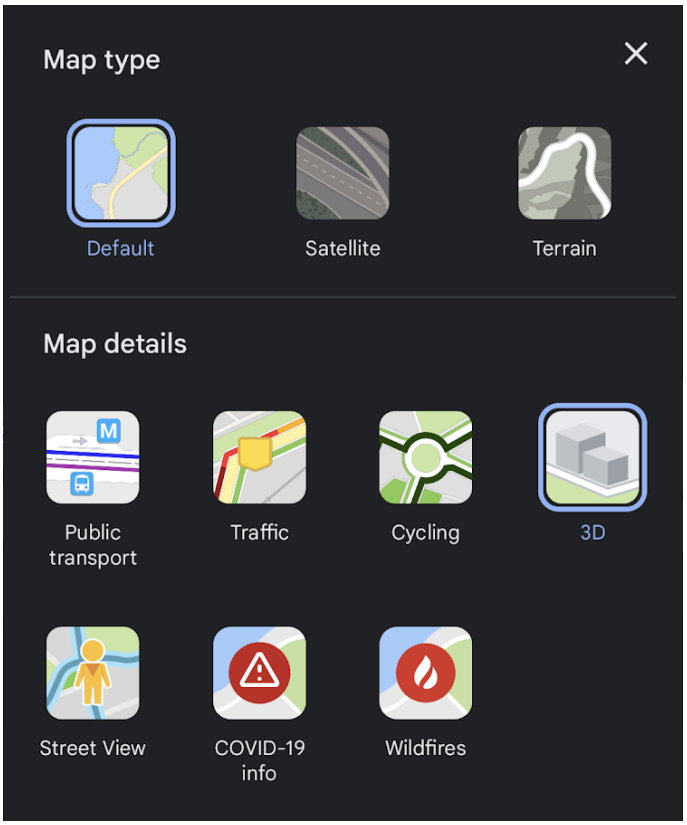


Figure 3.2: Google Maps map options

## 3.2 Implementations overview

In order to explore the different variables listed above in various combinations, we implemented multiple proofs of concept as separate Unity scenes and modules. These can be used separately or together as part of a full-fledged navigation application.

### 3.2.1 Impl. #1: Flutter & Google Maps SDK for Unity

For the first implementation, we experimented with direct Flutter integration in ACS UPB Mobile. For the source of truth, we used Google Maps SDK for Unity[11] to generate a low-detail 3D representation of the university campus.

#### 3.2.1.1 Campus map generation

Unity calls into the Google Maps API using an API key to obtain information about buildings and roads in a given area, which we then render appropriately (fig. 3.3). The Maps SDK for Unity generates multiple GameObjects as represented in fig. 3.4, with the `placeID` in brackets.

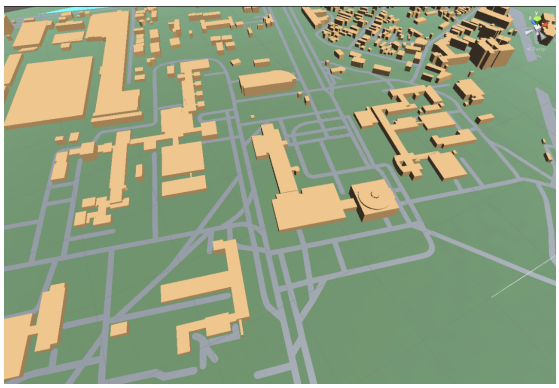

Figure 3.3: Campus buildings generated using Maps SDK for Unity


Figure 3.4: Game objects generated by Maps SDK for Unity

The metadata associated with these objects includes a `Name`, `PlaceId` and `UsageType` (one of `Unspecified`, `Bar`, `Bank`, `Lodging`, `Cafe`, `Restaurant`, `EventVenue`, `TouristDestination`, `Shopping` and `School`). As not all generated GameObjects map to a place of interest, the default labeling strategy only labels buildings where the name in the metadata is not "ExtrudedStructure" or "ModeledStructure". However, with this approach and the way our campus buildings are mapped, only three buildings in the entire campus have a label, as shown in figure 3.5.

Since the data from the Maps API is not sufficient, we could fetch the names of the campus buildings by using the Google Places API[12], which offers more infor-

[11] https://developers.google.com/maps/documentation/gaming/overview_musk
[12] https://developers.google.com/maps/documentation/places/web-service/overview

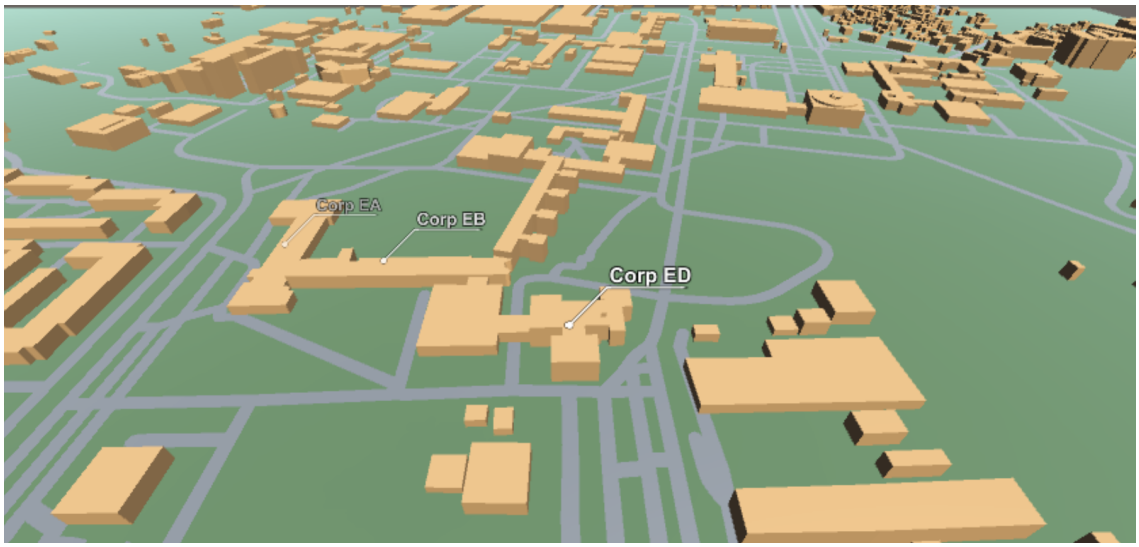


Figure 3.5: Campus building labels generated by Maps SDK for Unity

mation than the basic Maps API. We can do this by making a `GET` call and storing the response in a custom serializable class matching the response's structure):

```
// Fetch the name of a place via the Places API.
IEnumerator GetPlaceName(GameObject buildingGameObject, string
 ↪  placeId)
{
    string url =
     ↪  "https://maps.googleapis.com/maps/api/place/details/json";
    url += $"?placeid={placeId}&key={mapsService.ApiKey}";
    UnityWebRequest www = UnityWebRequest.Get(url);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
    }
    else
    {
        var attributes =
         ↪  JsonUtility.FromJson<Place>(www.downloadHandler.text);
        string displayName = attributes.result.name;
```

```
        Label label = Labeller.NameObject(buildingGameObject,
        ↪  placeId, displayName);
        if (label != null)
        {
            MapsGamingExamplesUtils.PlaceUIMarker(
                buildingGameObject,
                label.transform
            );
        }
    }
}
```

We cannot, however, call this method on every generated GameObject, as not all of them map to a place of interest and a large number of API calls leads the app to consume a lot of resources and freeze. The approach we took next was to try to filter the places we want to fetch more information for, by checking that the `UsageType` matches "School". While this does lead to more buildings being labelled than the previous approach, many are still missing and some are labelled "Bucharest" instead of a useful label, as seen in figure 3.6.

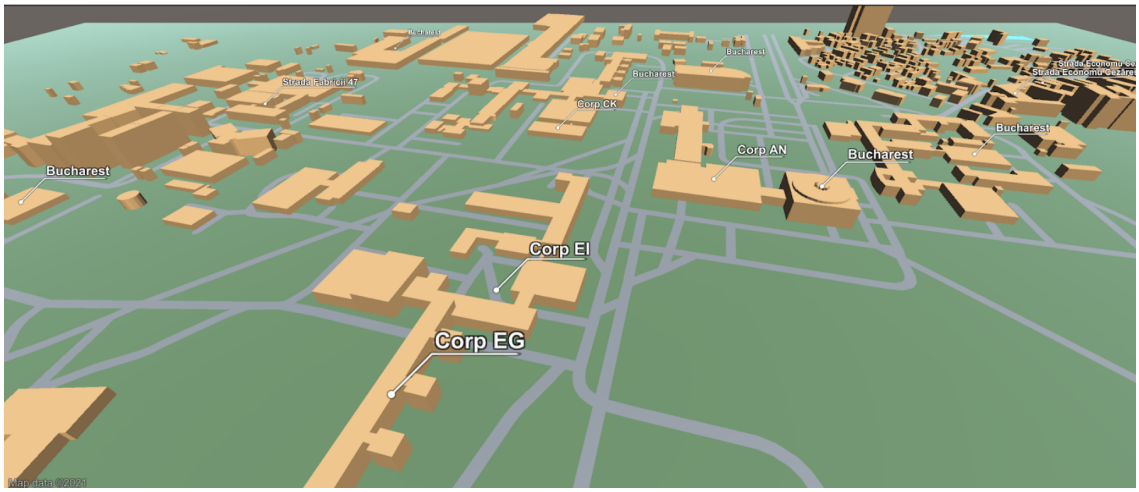


Figure 3.6: Campus building labels generated using the Places API

We therefore concluded that campus building labels cannot be reliably generated from the Maps/Places APIs. Therefore, the only reasonable solution here

is to manually mark the labels of the buildings of interest, alongside their corresponding placeID for the API. We fetch these entries from Firebase so that adding a new building label does not require uploading an app update.

#### 3.2.1.2 Flutter app integration

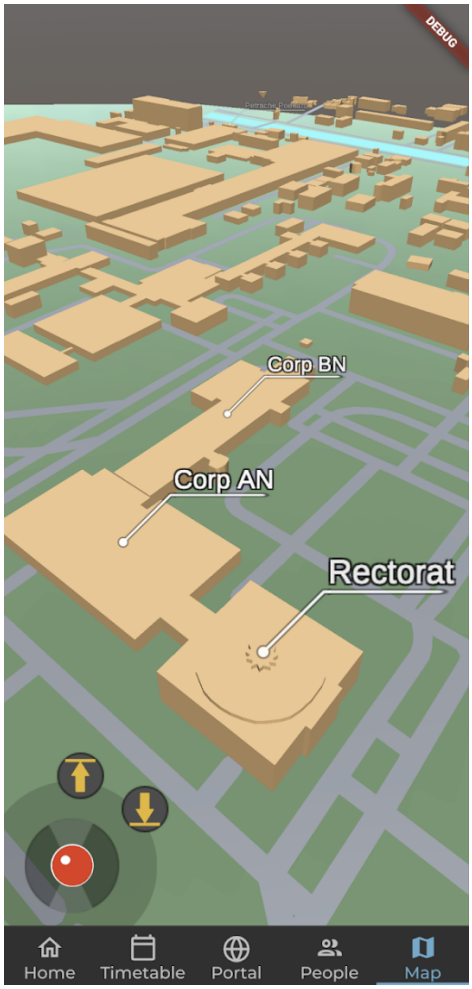


Figure 3.7: Screenshot of the Map section integrated in ACS UPB Mobile

The `flutter_unity_widget`[13] allows us to embed Unity into a Flutter app. Since Flutter allows embedding native code and binaries, the widget relies on building the Unity project for each platform (Android and iOS) and linking the result to the two Flutter build targets. The package also allows communication between Flutter and Unity through Messages, by using InteropServices[14] to import native Java (for Android) or Objective-C (for iOS) DLLs into C to allow calling native methods depending on platform. For Unity->Flutter communication, the messages themselves are serialized as JSON (JObject[15] in C#) and sent over as strings which can then be parsed in the onUnityMessage callback in Dart. For Flutter->Unity communication, the package allows us to call exposed GameObject methods directly using `postJsonMessage(String gameObject, String methodName, Map<String, dynamic> message)`, as long as the gameObject exists and is active in the scene, and the method is

[13] https://pub.dev/packages/flutter_unity_widget
[14] https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices
[15] https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JObject.htm

exposed through a `MonoBehaviour` script that uses `IEventSystemHandler`[16]. The message parameter represents a map from method parameter name to its value. We can use this messaging functionality in our app to send a query from Flutter to Unity (e.g. the destination location the user clicked on in an event page, or inputted in a Flutter search bar).

### 3.2.2 Impl. #2: Indoor (3D) navigation using NavMesh

Our next approach aims to achieve indoor navigation by drawing a path on a 3D model of the inside of a building of interest.

First, we used Blender[17] to create a simplified 3D model of a building in our university (EC) based on the floor plan. We then used Unity's built-in Navigation to bake a NavMesh onto the model, to use for navigation. The NavMesh is visible in blue in figure 3.8.

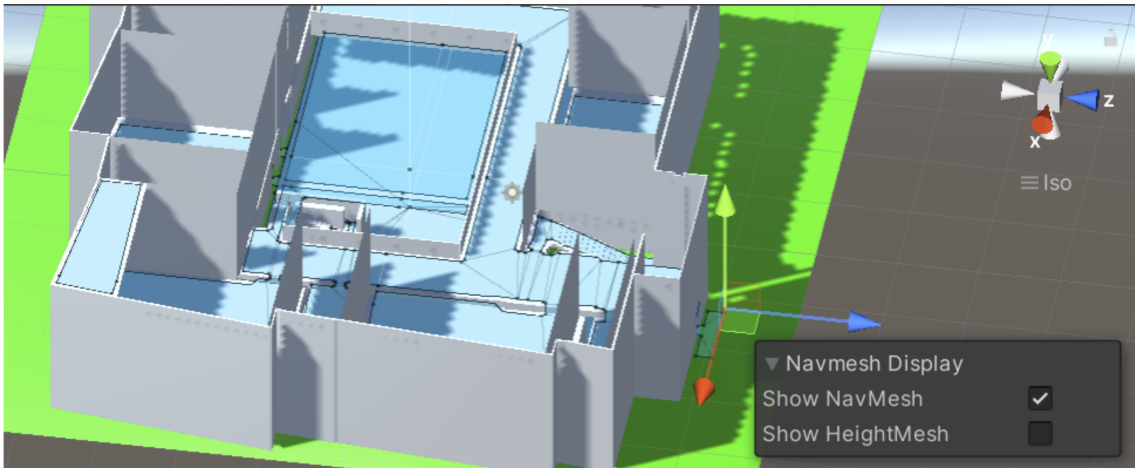


Figure 3.8: NavMesh baked onto the EC building

We then created a `NavMeshAgent` representing the user, and added support for clicking on a spot on the mesh in order to mark the destination. Once a destination is chosen, a path is drawn on the mesh between the agent and the destination, representing the recommended navigation route (fig. 3.9) .

[16] https://docs.unity3d.com/2019.1/Documentation/ScriptReference/EventSystems.IEventSystemHandler.html

[17] https://www.blender.org/

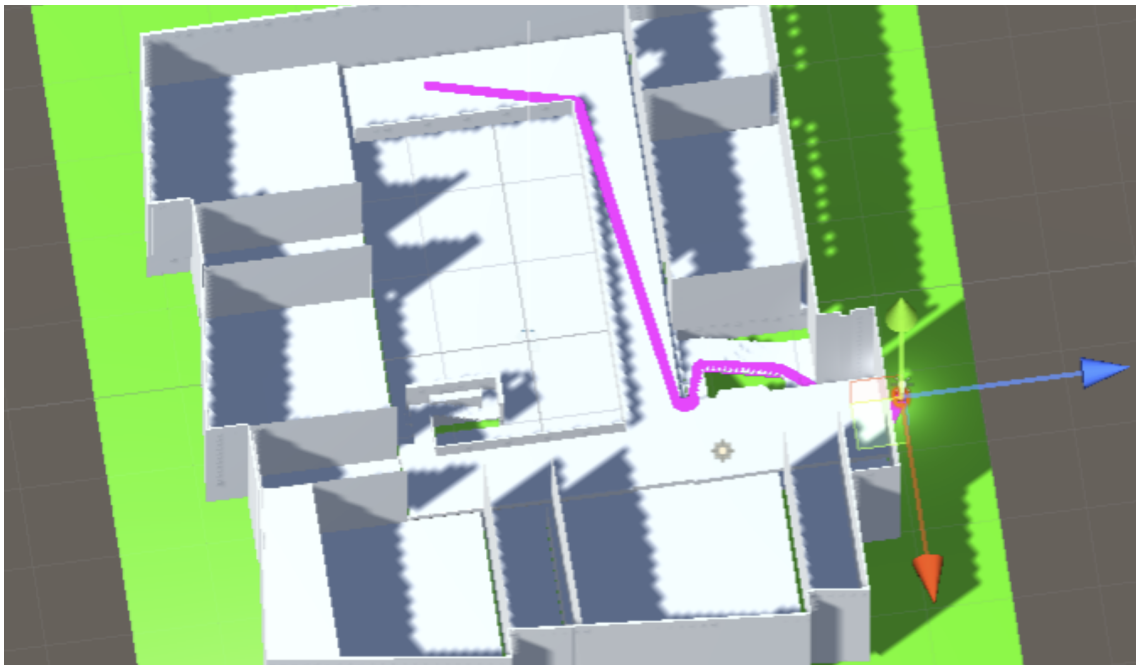

Figure 3.9: Navigation path drawn using NavMesh

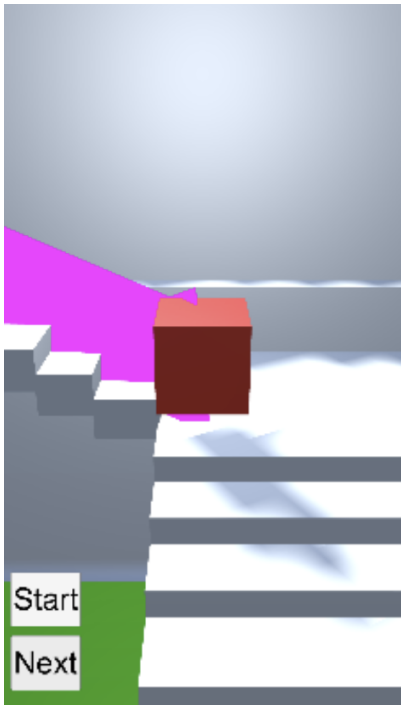


Figure 3.10: Third-person navigation view

In the 3rd person player view, for the proof of concept we added two buttons - a "Start" button which triggers drawing the navigation path to the destination, and a "Next" button which moves the agent to the next "turn" (node) in the path. As such, the user can manually select a location as the destination, as well as their current location, and be guided step by step through each segment of the path.

As an alternative to the low-fidelity 3D model built based on the floor plan, we also tried to use a model of the building from the 3DUPB archive (fig. 3.11). However, these models are designed to be as accurate as possible, including the furniture and decorative elements in the buildings. This lead to a significantly more complex NavMesh and much higher latency, making it unsuitable for a mobile application. These models could, of course, be simplified, but we have found that using a simple model made from scratch is the more scalable approach.

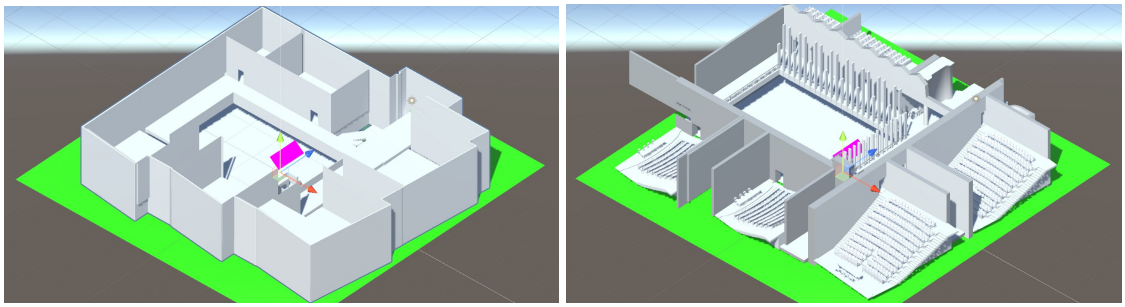

Figure 3.11: Floorplan-based low fidelity model (left) vs. 3DUPB high-detail model (right)

### 3.2.3 Impl. #3: Outdoor (3D) navigation with MapsModelsImporter

For our third implementation, we test out a different approach to outdoor navigation using a 3D map imported from Google Maps. For this, we use MapsModelsImporter[18], a Blender add-on for importing 3D models from Google Maps. As such, we imported multiple meshes corresponding to the campus map, overlapped and merged them together in Blender (fig. 3.12).

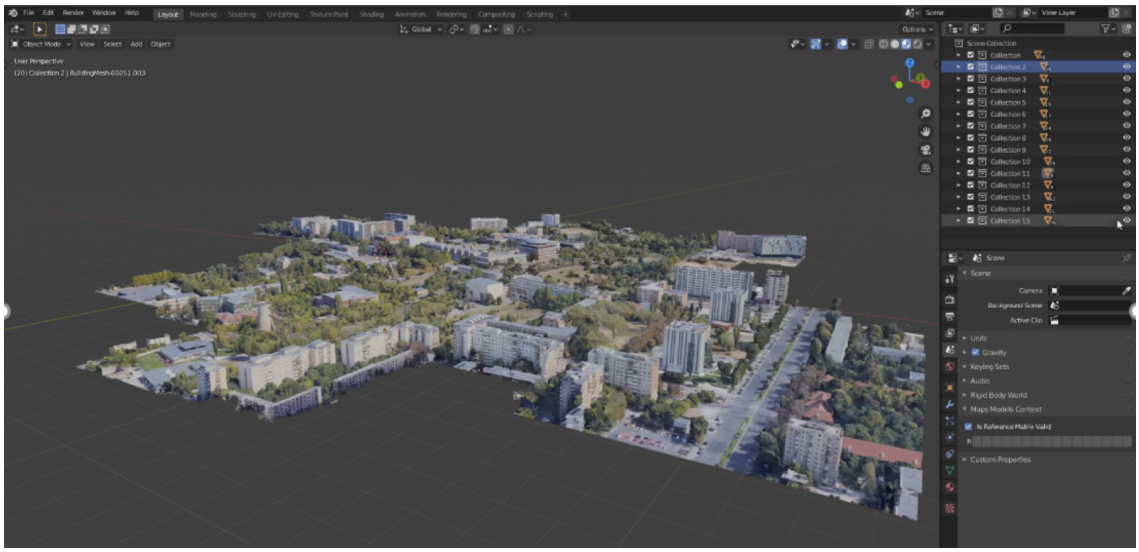

Figure 3.12: Map model imported in Blender using MapsModelsImporter

[18] https://github.com/eliemichel/MapsModelsImporter


Figure 3.13: Floorplan Unity module

The floor plan GameObject was created by drawing an SVG and importing it into Blender as a mesh. This allows us to have separate objects for each different room in Unity, in order to define on-click behaviours using ray casting.

Each room has a unique identifier, so we can also fetch additional information about it from Firebase (such as type, room capacity, building floor it belongs to etc.).

The Firebase integration allows us to easily update room information remotely, without requiring an app update. However, the same cannot be said for the actual campus maps - these are imported as Unity objects, and therefore included in the app executable. Fortunately, we don't expect the maps to change very often.

Furthermore, for this demo we also use standard GPS (location services) to place the agent (user) on the map:

```
if (Input.location.status == LocationServiceStatus.Running)
{
    // Access granted to GPS values & it has been initialised
    float x = latToX(Input.location.lastData.latitude);
    float z = lonToZ(Input.location.lastData.longitude);
    player.transform.position = new Vector3(x, 0.8f, z);
}
```

Finally, we again use Unity's NavMesh to create paths from the user's GPS location to a point of interest. We currently use the auto-generated NavMesh (fig. 3.14), but it can be improved by manually drawing the mesh over the areas representing a walkable paths (alleys, roads in the campus).

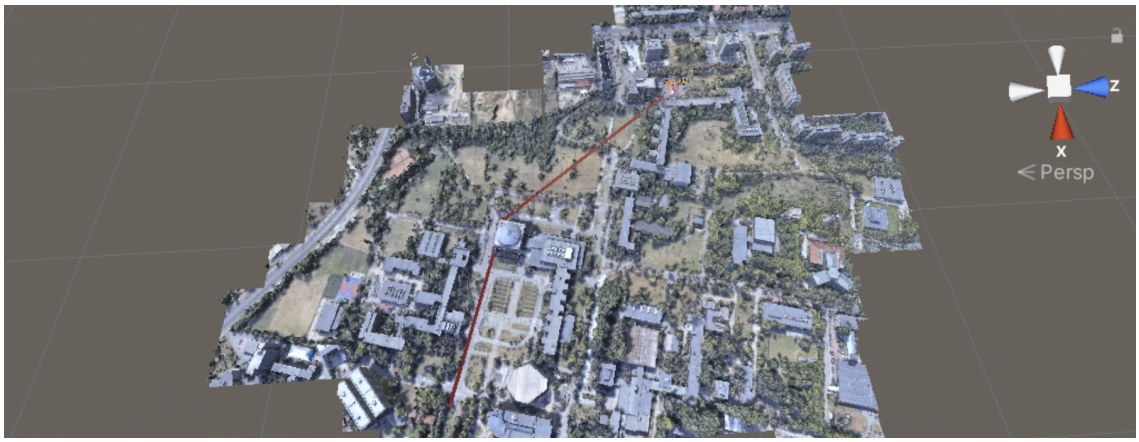


Figure 3.14: Path drawn on MapsModelsImporter map

### 3.2.4 Impl. #4: AR-powered indoor navigation

For our final indoor navigation implementation, we explore AR navigation using SLAM. This approach is based on an article by CRAFTWORKZ's Jan Hardy[21] and an article by Roberto Lopez Mendez from Arm Community[22].

Due to not having access to the campus while testing, we tested the initial proof of concept at home, using a floor plan created by a robot vacuum. However, this solution is not highly dependent on the location, and can easily be adjusted with any map, as long as it is to scale.

We started by creating a plane with the floorplan in Unity. In order for the tracking to work correctly, we also needed to make sure that the scale of the map in Unity matches real life (1 meter in real life = 1 distance unit in Unity). If this scale is not correct, the position of the user on the map will seem to be moving too slowly or too quickly.

We then used the Google ARCore Extenssions for AR Foundation[19] package to integrate AR functionality, and created a script that translates real life movement to the movement of a sphere, chosen to represent the user's location. This allows us to move the sphere on the map as the user moves with their AR camera, using SLAM.

Once the sphere can be moved according to the user's movement, we need to also make sure that the starting point corresponds to the user's starting point. GPS location is not accurate enough, so we chose to add some anchor points on the

[19] https://github.com/google-ar/arcore-unity-extensions

map that the user can start navigating from. These points have an ID associated with them, and this ID can be fetched by scanning a QR code at that specific location (fig. 3.15). For the QR scanning functionality, we use the ZXing.Net library[20].

Finally, for the actual navigation, we again use NavMesh to map the walkable area on the map and draw a path to the destination. We show this path on the 2D minimap in the app, in addition to an arrow pointing to the direction where the user has to go (fig. 3.16).


Figure 3.15: QR scanning for the initial positioning

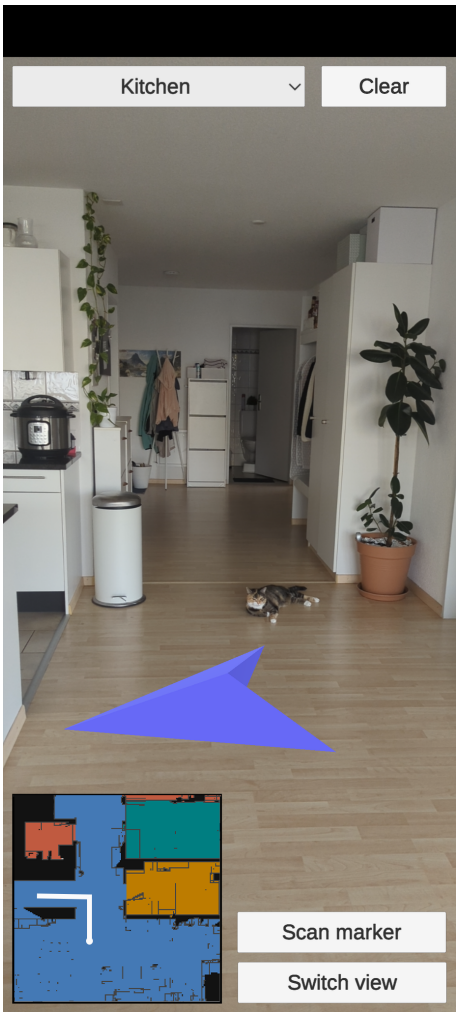


Figure 3.16: AR indoor navigation experience

[20] https://github.com/micjahn/ZXing.Net

### 3.2.5 Bonus module: 360° image viewer

As an additional feature to navigation, we created a module for viewing 360° images of various points of interest in the campus. We reached out to the photographer who created the UPB Virtual Tour[21], and he gave us access to all 360° images in the university's virtual tour collection. These images are fetched directly from our cloud storage, so as not to unnecessarily increase the app size. Users Would be able to tap certain points of interest on the map and view the relevant 360° photo taken from that point.

In terms of implementation, a spherical image viewer was implemented in Unity using the Skybox (fig. 3.17). Touch controls were also added, to allow the user to pan and zoom.

Figure 3.17: 360° image viewer

## 3.3 Comparison

Table 3.3 illustrates the variables (explained in section 3.1) selected for each of the four proofs of concept described in the previous section.

A comparison of these approaches in terms of user experience can be found in section 4.3.3 in the following chapter.

[21] http://turvirtual.upb.ro/

Table 3.3: Implementation comparison

| | **Impl. #1** | **Impl. #2** | **Impl. #3** | **Impl. #4** |
|---|---|---|---|---|
| **Goal** | outdoor navigation | indoor navigation | outdoor navigation | indoor navigation |
| **Technologies, tools and APIs** | Unity, Flutter, Google Maps SDK for Unity, Firebase | Unity, Blender, Firebase | Unity, Blender, MapsModelsImporter | Unity, ZXing, ARCore, Firebase, Photoshop |
| **Map representation** | 3D, low detail | 3D, low & high detail | 3D, high detail | 2D |
| **Integrations** | directly as a page ACS UPB Mobile | - | - | using UPB Campus room data |
| **Source of truth** | Google Maps (dynamic) | fixed building model | Google Maps (fixed) | fixed floor plan |
| **Pathfinding** | A* | NavMesh | NavMesh | NavMesh |
| **Initial positioning** | user input | user input | GPS | QR code |
| **Movement tracking** | user input | user input | GPS | AR with SLAM |

# Chapter 4

# User study

To guide future efforts, we decided to perform a user study to directly gauge the users' views. Overall, our study had three main goals:

- confirming the importance of campus navigation for certain user groups (first year students, aspiring high school students and parents, other one-time campus events participants)
- understanding the students' campus navigation needs: what they're currently using, and what they are missing
- getting feedback regarding our four prototypes, described in the previous chapter

## 4.1 Past studies

As stated previously, we have based our implementation decisions upon known and established state of the art solutions, as well as previous related studies.

### 4.1.1 ACS UPB Mobile

One notable study has been performed by us for the previous paper, "Design and Implementation of a Cross-Platform Mobile Application That Facilitates Student Collaboration"[3]. There are a few key points that we've used from this existing user study from 2019, which had over 200 answers.

Firstly, we know that, although the majority (roughly 80%) of students in our target user group use an Android phone, the iOS demographic is still significant enough to warrant investing effort in a cross-platform solution.

Secondly, over 35% of student responders prefer to use English as the main language of their mobile OS. This, paired with the fact that the faculty welcomes foreign students, is again significant enough to justify the need for a localized app that supports at least two languages: English and Romanian. Inclusiveness is a top priority, therefore students of all backgrounds should have access to our campus navigation tool.

Finally, we also have to note that, based on this prior study, a map function was second to last on a usefulness scale for student responders. As highlighted previously in section 1.3, we agree that the usefulness of a tool that simply provides navigation instructions will decrease significantly over time, as a student learns the locations of their classes. As such, in the same chapter we propose additional functionalities that would keep the tool relevant even as time passes.

### 4.1.2 UPB Campus

A similar study was also done for the UPB Campus app, also in 2019, by Diana Scurtu[16]. From there, we learn that, out of over 100 surveyed students (a majority of which were first year students), a whooping 68% consider it difficult to find some locations in the campus. In terms of methods used by students to go around campus, 44% prefer to ask for directions, while the rest are equally split (28% each) between using the campus map and using a GPS mobile app. Finally, roughly 88% of survey responders believed that an application for the university campus would be useful.

## 4.2 Methods

For our study, we chose to use a mix of research approaches. The base of our research consisted of **analysing previous studies**, as well as using **observation** directly throughout six years studying at ACS and interacting with the students and the campus. Furthermore, we sent out a **survey** that received over 100 responses, and conducted 1:1 **interviews** with university students, professors and even high school students who aim to apply at UPB.

Thanks to the mostly-online medium we've found ourselves in the past couple of years due to the pandemic, most of our communication was done over the internet. The survey consisted of a *Google Form*[1] that was shared via *WhatsApp*[2] to various students' groups. Most interviews were conducted over a *Microsoft Teams*[3] video call, with some taking place face-to-face or over a chat app (*WhatsApp*, *Facebook Messenger*[4], *Telegram*[5]).

The data from the survey was extracted into a *Microsoft Excel*[6] spreadsheet, normalised and used to generate charts and other correlational statistics (see section 4.3 below).

## 4.3 Results

### 4.3.1 Survey

Our survey included a variety of different questions, while still being short enough to not discourage students from filling it out. As such, we have received over 100 responses in the span of one week. We have used open-ended, as well as multiple choice questions (both with only one as well as with multiple possible responses) and a Likert scale.

Firstly, in order to gauge how common of an issue finding a campus location is, differently from previous questionnaires, we have asked students what are the reasons they have been late for a class. Figure 4.1 shows the number of students (out of 103 responses) who chose each answer in a multiple-choice, checkbox-style question allowing multiple answers. There, we can see that not finding the classroom is the second most common reason identified by the students with a 54% agreements, after waking up late (63%). Other reasons included the class before going overtime and being stuck in traffic, which 44% and 33% of students relating to, respectively. These results work to confirm our assumption that not being able to efficiently navigate through campus may affect the students' ability to get from one class to the other in time.

[1] http://forms.google.com/
[2] https://www.whatsapp.com/
[3] https://teams.microsoft.com/
[4] https://messenger.com/
[5] https://telegram.org/
[6] https://www.microsoft.com/en-us/microsoft-365/excel

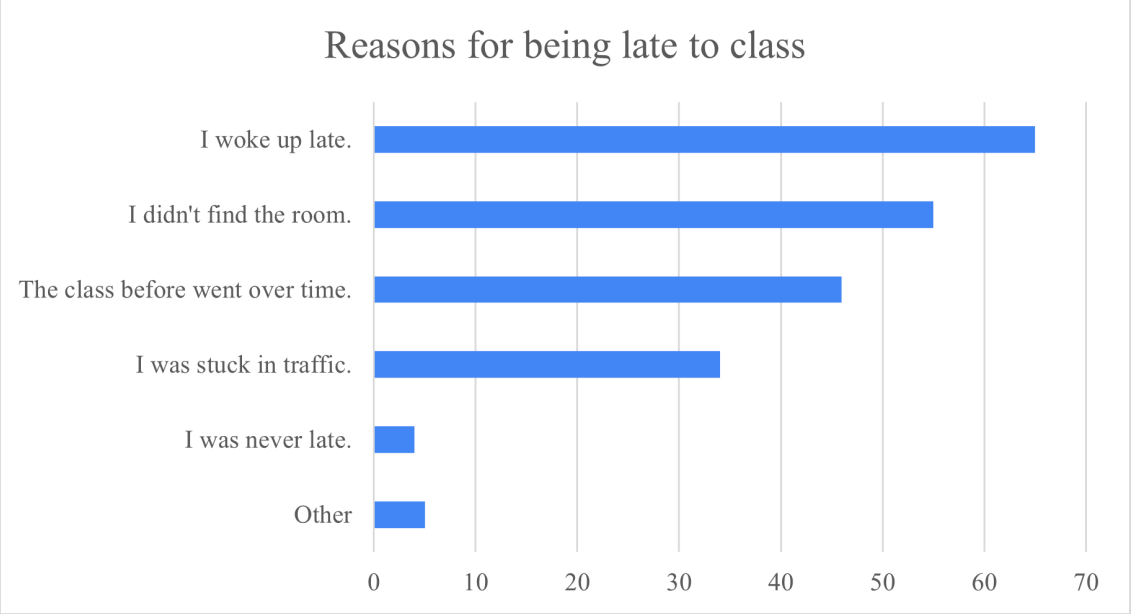


Figure 4.1: Students' reasons for being late to class

Next, we asked students what tool or approach they are using when they need to navigate through campus. The results here closely matched the one from the study made in 2019 for UPB Campus[16]. Figure 4.2 shows the result of another checkbox-style question, with the x-axis representing the number of students who checked a specific answer. Notably, many students (roughly 40%) prefer to use a mix of multiple methods rather than just one. The majority of student responders (49%) prefer to ask colleagues or someone else for directions. Some students use a photo of the map or the freshman's guide[7] (44%), while others make use of well-established navigation apps such as Google Maps (44%) and Apple Maps (3%). Finally, although it has been more than two years since the launch of the UPB Campus app, only 2 students mentioned using it for campus navigation.

Although very few ACS student responders use the UPB Campus app, more students mentioned using ACS UPB Mobile when asked (fig. 4.3). Though 64% have not heard of the app, and 17% don't find it particularly useful, 19% mentioned using it sometimes or regularly. While this is not a very significant percentage, it is still 10x more than the result for UPB Campus. This is surprising, considering UPB Campus has over 1000 downloads on the Play Store[8], while ACS

[7] https://www.slideshare.net/vladposea/ghidul-bobocului-de-la-facultatea-de-automatica-si-calculatoare-vers-20112012

[8] https://play.google.com/store/apps/details?id=com.app.upb

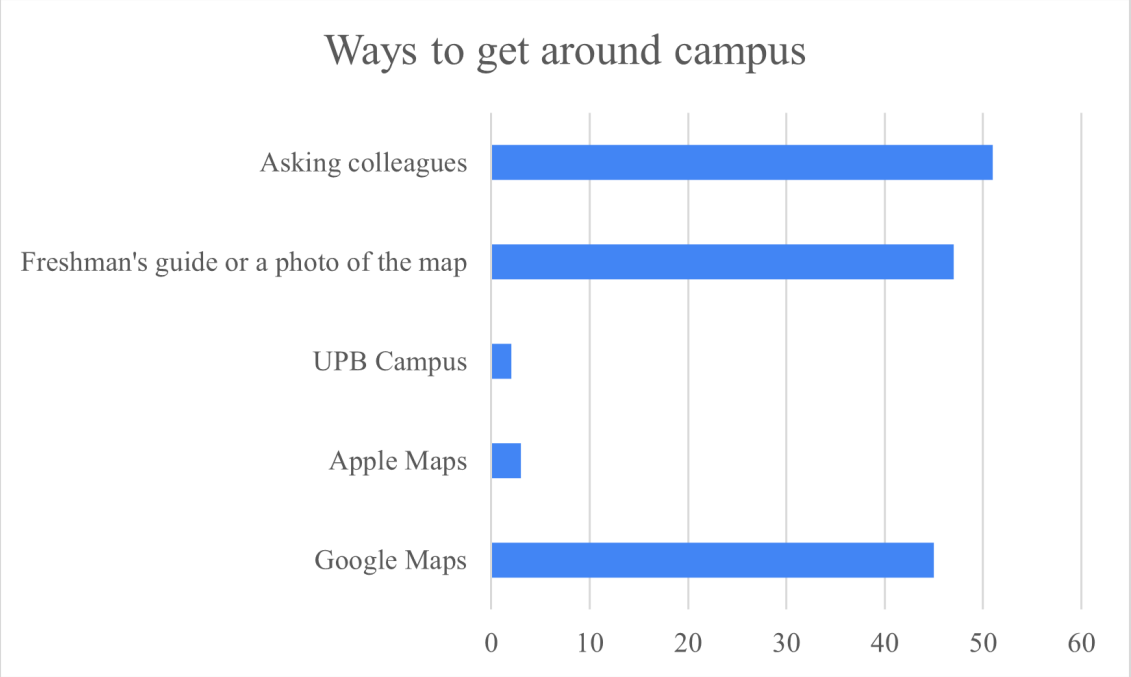


Figure 4.2: Ways to get around campus

UPB Mobile has only 500[9]. However, the target audience for UPB Campus consists of all the students at UPB, whereas ACS UPB Mobile is targeted specifically at ACS students (our survey responders).

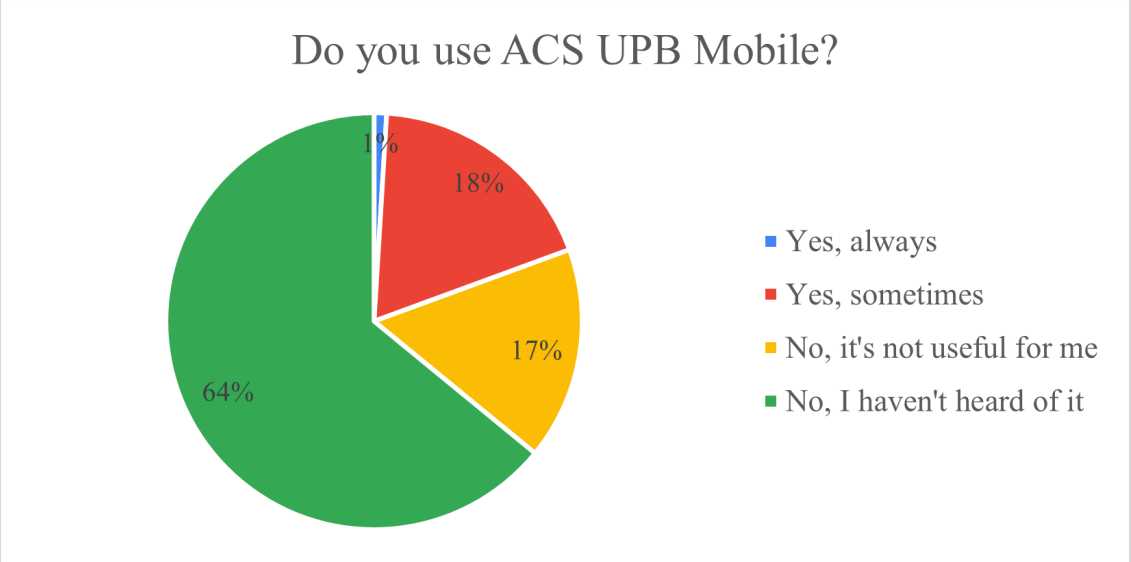


Figure 4.3: Use of ACS UPB Mobile

[9] https://play.google.com/store/apps/details?id=ro.pub.acs.acs_upb_mobile

Given the fact that classes at UPB have been mostly online in the past two years due to the pandemic, we also wanted to understand for how long our survey responders have had in-campus, physical classes. We believed this may affect their view on the need for campus navigation. As seen in figure 4.4, more than half of the students who responded have had only one semester of in-campus classes or even less. However, based on our data, we have not found a significant difference in responses from these students compared to the other groups. This indicates that issues with campus navigation are similar before and after the pandemic.

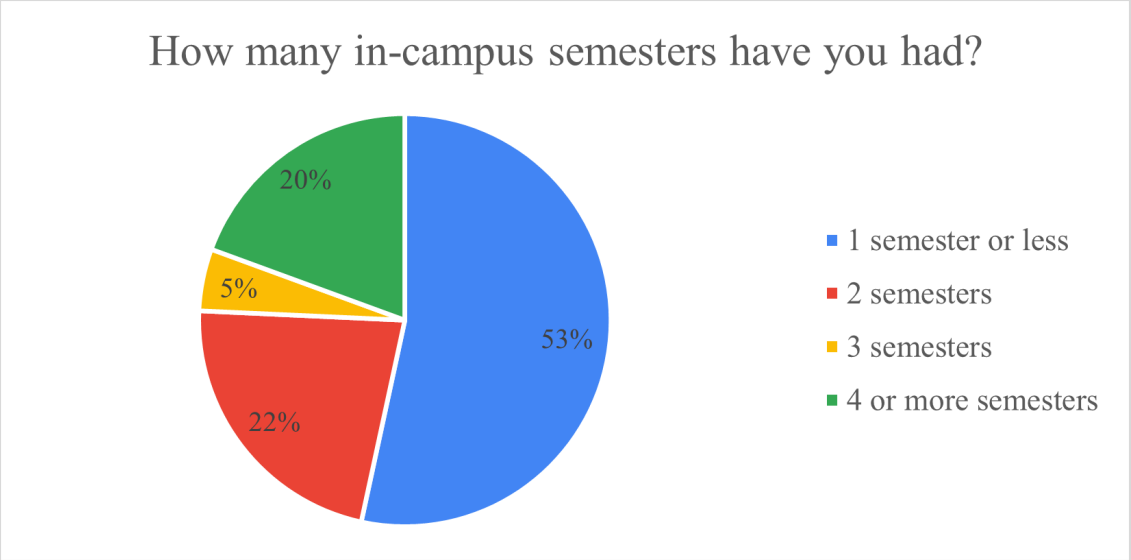


Figure 4.4: In-campus semesters for survey responders

We also used a Likert scale from *Not important* to *Very important* to assess what kind of information would be important to include in a campus map. The chart in figure 4.5 assigns a score from -1 to +2 to each level on the scale, and adds up the survey responses in order to display a reverse ranking of potential information that could be present on the map, based on the features we listed in section 1.3. The features are in descending order based on the following formula:

$$\sum_{n=1}^{103} response_n \cdot score(response_n)$$

where $response_n$ is the $n$th survey response for this question (one of *Very important*, *Important*, *Somewhat important*, *Not very important* and *Not important*), and *score* is:

$$score(response) = \begin{cases} -1, \text{ if } response = \text{"Not important"} \\ -0.5, \text{ if } response = \text{"Not very important"} \\ 0.5, \text{ if } response = \text{"Somewhat important"} \\ 1, \text{ if } response = \text{"Important"} \\ 1.5, \text{ if } response = \text{"Very important"} \end{cases}$$

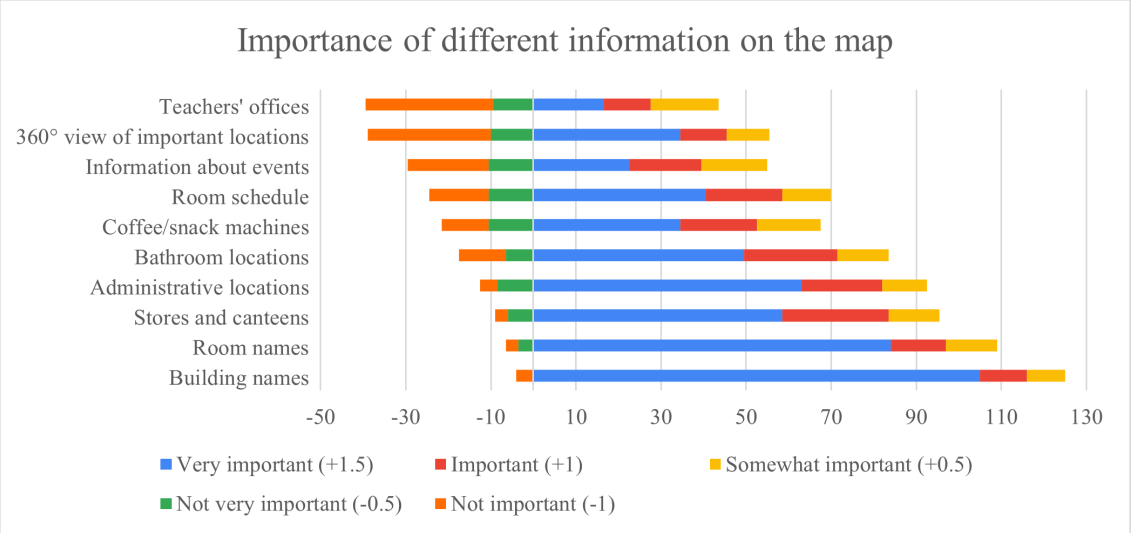


Figure 4.5: Importance of different information on the map

Based on this ranking system, by far the most important pieces of information that need to be on the map are the names of the campus buildings, corresponding to outdoor navigation. About 20 points away in the ranking are room names, corresponding to basic indoor navigation, as we have expected. In terms of POIs on the map and another 20 points away, administrative locations such as the secretariat or the dean's office have a similar score to campus shops and canteens. Moving down another 20 points would be the locations of the bathrooms, which are lower than we expected - likely because they are usually reasonably easy to find anyway. Going down another 20-points-tick, we find another POI - coffee & snack machines. These have a similar importance score with information about the schedule of a specific room (e.g. what classes are taking place there at a specific time). Yet another 20 points further down the scale, we find information about other events taking place at specific locations. Ten points lower, and on the second to last importance position, we find the 360° views of important locations in the campus. Finally, another 10 points down the ranking and on

the last position, we find information about the teacher's offices. It should be noted, however, that while the information on these last two positions are not important for students, they seem to carry more importance for aspiring high school students and university professors, respectively (see section 4.3.2).

We also asked students what other information they would like to see on a map. Some of the suggestions for points of interest were *elevators, student housing, parking spaces and print shops*.

Several students also mentioned they would like to see access points and whether they are open or not (e.g. campus gates, building entrances, the location of the guard who keeps keys to the rooms).

### 4.3.2 Interviews

We interviewed people from various user groups, including non-students:

(1) a junior high school student aspiring to apply to UPB next year (and their mother, who joined the interview as well)
(2) a professor at ACS
(3) a computer science student from another university, who participated in events in our campus such as PoliJobs
(4) a member of the ACS student league (LSAC)

Both the high school student and their parent (1) were very interested in the 360° views of campus locations, as they wanted to see what learning there would be like. None of the other interviewees found this feature particularly useful, which matches the insight from the survey. They were also interested in understanding how to enter the campus and reach the location for the extracurricular maths classes taking place as prep for the entrance exams.

The ACS professor said navigation would probably not be very useful for them as they already know most of the locations they would want to go to. However, they mentioned they sometimes forget which office some other professors are in (especially in the PRECIS building), and that having an accessible list with their office locations would be useful. They mentioned occasionally using the cs.pub.ro website to view this information. Once again, none of the other users interviewed found this information particularly useful.

Interviewee (3) mentioned they had difficulty reaching the in-campus locations for the events they participated in, especially the library. This was mostly

due to them not being familiar with the campus at all, considering they only came for a few visits. When asked if they would install an app especially to navigate in-campus at an event, they were hesitant. They eventually said they might have still resorted to asking around and using Google Maps over having to install another app.

Interviewee (4) was excited about the possibilities that a campus navigation app could have, and mentioned a recent treasure hunt that the student league organised. They suggested that having a special app that would encourage people to participate and give them points when they reach certain locations would have been very useful. Furthermore, they also suggested that the app may be useful for high school students when they come to sign up for the admittance exam.

### 4.3.3 Demos

We invited a group of 5 students to test our proof of concept applications and provide feedback. We selected a diverse group:

(1) a 1st year BSc student who hasn't used either UPB Campus or ACS UPB Mobile
(2) a 2nd year BSc student, a group leader who regularly uses ACS UPB Mobile and UPB Campus
(3) a 4th year BSc student who doesn't use either app
(4) a 4th year BSc student who contributes to the ACS UPB Mobile development
(5) a 2nd year MSc student who had most classes remote in the past two years, and doesn't use either app

#### 4.3.3.1 Impl. #1: Flutter & Google Maps SDK for Unity

First, we gave the testers a version of ACS UPB Mobile with the integrated campus map feature. The students who have not used ACS UPB Mobile before took time to explore other features in the app before getting to the map. Testers (1), (2) and (4) all noted that the map feature stands out as having a very different style and "feel" to the rest of the app. All testers noticed that the performance is pretty slow when opening up the map page. Tester (3) expressed that they believe "it's easier to just use a photo of the map". Tester (5) pointed out that ACS UPB Mobile looks like it may be useful, but the map feature itself wouldn't really be

useful for long. Finally, tester (2) said they consider UPB Campus to be easier to use and more intuitive.

#### 4.3.3.2 Impl. #2: Indoor (3D) navigation using NavMesh

Subsequently, we showed each tester a demo of the NavMesh-based 3D campus navigation. All testers expressed excitement upon seeing a 3D model of a building they are familiar with, and liked the comprehensive path it shows to the destination. They also asked to pan around the building so they can see it better. Testers (3), (4) and (5) mentioned they don't find the third-person view to be very useful, and that being able to view the building from above and pan around it would be enough to understand the path.

#### 4.3.3.3 Impl. #3: Outdoor (3D) navigation with MapsModelsImporter

We then showed the testers the demo of the MapsModelsImporter approach. Except for (5), all testers preferred the view before (the simple building shape) to this more detailed map. Some mentioned the details are too distracting, and you can't see the path very well. All testers mentioned that GPS localization should be part of every implementation, even if not entirely accurate. Tester (2) considered this to be similar to UPB Campus, but a bit more "overkill".

#### 4.3.3.4 Impl. #4: AR-powered indoor navigation

Finally, we gave our testers the demo for AR-based indoor navigation. None of them had used this kind of navigation system before. When asked whether they knew Google Maps had this feature, tester (4) answered that they were aware of it, but hadn't tried it before; all other testers hadn't heard of this feature at all. Tester (1) found this feature very useful, and shared that they consider themselves bad at reading maps in general. They found this navigation approach more intuitive than following a map. The other testers said this approach "might" be useful, but that they would mostly opt for using a regular map or one of the other approaches instead. They felt uncomfortable having to walk around with a camera pointed at their surroundings.

# Chapter 5

# Conclusion

## 5.1 Summary

Our study explores the challenge of indoor and outdoor navigation tailored for a university campus, and the needs of the students and other types of users.

In chapter 2, we describe several different state of the art approaches for indoor and outdoor navigation, from generic solutions to related studies focusing on campus navigation.

In chapter 3, we describe the four different implementations we developed in order to analyze and compare how some of these approaches stand up to the unique challenges of campus navigation. Through these implementations, we demonstrate different graphical representations of the campus, pathfinding, positioning and tracking approaches (from classic GPS to Augmented Reality solutions) as well as a combination of various technologies and APIs.

Finally, in chapter 4, we filter our assumptions and these implementations through the lens of potential users, in order to understand the benefits and shortcomings of each one. Our UX research methods include observation, user interviews, usability testing as well as an online survey.

## 5.2 Results

Based on our testing and user feedback (chapter 4), we can conclude that the "less is more" principle applies for campus navigation.

### 5.2.1 Features

Users prefer the map and navigation experience to be as uncomplicated as possible. They also don't want to see a lot of information on the map, and prefer to be able to filter specific points of interest. While a map can include data about various POIs, different types of users are interested in different types of information. For example, prospective high-school students may want to see 360° views of various locations in the campus to know what to expect. However, students are more interested in quickly navigating to their next class. Furthermore, one-time users such as those who attend an event on campus would only be interested in finding their destination, and may be less willing to install a special app for that. Therefore, it is important to create a highly customizable map that can fit the needs of different user types, without introducing a lot of noise.

While some users prefer creative alternatives to classic maps, most users found novel navigation approaches such as AR-based navigation to be too finicky and uncomfortable to use.

### 5.2.2 Source of truth

In terms of the source of truth, dynamic sources (e.g. Google Maps) can be useful for locations that tend to change relatively often. However, for a campus map that almost never changes, a static source (such as a floorplan (see implementations #3 and #4 in section 3.2) or a fixed 3D campus model (see impl. #2 and #3) are just as reliable and significantly more resource-efficient. In contrast, implementation #1 attempts to dynamically build the campus model using Google Maps SDK for Unity[1]. However, after extended testing, this approach does not scale well because the in-game generation of the map means there are many calls made to the Maps API, which not only increase latency and network usage for the user, but also leads to significant Google Cloud usage bills. For this specific approach, an alternative would be, since the buildings/alleys in the campus don't change often, to fetch this information only once and bake it into the app. However, this is also not an ideal long-term solution, since the SDK has since been deprecated in October 2021.

[1] https://developers.google.com/maps/documentation/gaming/overview_musk

### 5.2.3 Integrations

We have also discussed integrating with existing tools. Implementation #1 attempts to integrate the navigation functionality into an existing app for students - ACS UPB Mobile. This app is currently using Flutter (a cross-platform UI framework) and a server-less solution called Firebase for the back-end. The integrationis possible using flutter_unity_widget[2], which allows us to embed Unity into a Flutter app. However, this approach had some significant drawbacks:

- The app size increased tenfold with the addition of the Unity project (27MB vs 324MB)
- High latency when navigating to the Unity widget (3-4s)
- Instability: the widget is relatively new, so there is a high likelihood of crashes for more complex Unity projects

Given that, based on a survey with 200+ respondents[3] made for ACS UPB Mobile, the Map/navigation feature was the second lowest rated in terms of importance (out of 8), these drawbacks are not worth the in-app integration. Additionally, campus navigation would be a useful feature not only for students (which ACS UPB Mobile is aimed at), but also for staff (professors, administrative staff, cleaning staff etc.) as well as visitors (high school students, foreign students, parents, alumni etc.), especially given the number of events that happen on campus every year. Furthermore, ACS UPB Mobile requires authentication, which should not be necessary for someone who is just trying to find a location on campus.

Therefore, a better solution would be to have campus navigation as a separate app.This would not only allow users who are not students (and don't have a university account) to use it, but it would also not force students who use ACS UPB Mobile, but don't need the navigation assistance, to deal with the additional overhead. We can still achieve integration by allowing apps such as ACS UPB Mobile and UPB Campus to directly open a target location in our app for navigation.

[2] https://pub.dev/packages/flutter_unity_widget

## 5.3 Future improvements

The user experience of any indoor/outdoor navigation solution is highly dependent on the positioning/tracking implemented by the solution. The accuracy of the user's position on the map directly influences their ease of navigation. In our implementations, we have leveraged the GPS location (impl. #3) as well as AR-based position tracking (impl. #4). As discussed in chapter 2, GPS alone is not accurate enough for indoor positioning, and AR navigation is not widely accepted by users as a comfortable solution. Although positioning was not the main topic of this paper, it is worth noting that a campus navigation solution could greatly benefit from a more sophisticated localization system, especially for the purpose of indoor navigation.

As discussed in section 5.2.1, users want a navigation app to be as simple as possible. However, different users have different needs, and there is no "one size fits all", both in terms of navigation approaches as well as the actual information on the map (POIs). As such, displaying rich information about various locations in the campus is just as valuable as having comprehensive filtering and customization options to match a user's preferences.

# Abbreviations

**ACS** Faculty of Automatic Control and Computer Science. 6, 10, 11, 22, 46, 48, 49, 60

**API** Application Programming Interface. 9

**ARACIS** Agenția Română de Asigurarea Calității în Învățământul Superior. 10

**IPS** indoor positioning system. 16

**IR** infrared. 16

**POI** point of interest. 10, 12, 17, 29, 51, 56, 58

**RFID** radio frequency identification. 16

**SLAM** simultaneous localization and mapping. 29, 30, 41

**UPB** University POLITEHNICA of Bucharest. 2, 6, 10, 11, 20–22, 24, 46, 49, 50, 60

# Glossary

**ACS UPB Mobile** is a collaborative mobile application for students at Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest. 6, 11, 13, 17, 24, 25, 33, 36, 44, 57

**Firebase** is a mobile and web application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014. 10, 25, 57

**Flutter** is Google's open-source UI software development kit, used to develop cross-platform applications for Android, iOS, Google Fuchsia and the web. 4, 10, 25, 26, 33, 36, 44, 57

**GPS** (Global Positioning System) is satellite-based radionavigation system. 6, 15, 16, 29

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format based on defining key-value pairs. 20

**open-source** denotes software for which the original source code is made freely available and may be redistributed and modified. 10

**Unity** is a cross-platform game engine developed by Unity Technologies. 4, 26–28, 31, 33, 44

**Unreal Engine** is a cross-platform game engine developed by Epic Games. 26

# Bibliography


[1] J. Albertson. (2017, August) Explore your new campus with Google Maps. Date accessed: 20.12.2020. [Online]. Available: https://www.blog.google/products/maps/explore-your-new-campus-google-maps

[2] Agenţia Română de Asigurarea Calităţii în Învăţământul Superior, "UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI - Raport Privind Evaluarea Externă a Calităţii Academice," 2007. [Online]. Available: https://www.aracis.ro/wp-content/uploads/2019/12/BROSURA_PUBLICARE_UPB_1.pdf

[3] I. Alexandru, "Design and implementation of a cross-platform mobile application that facilitates student collaboration," 2020. [Online]. Available: https://github.com/acs-upb-mobile/paper/blob/master/Design_and_implementation_of_a_cross_platform_mobile_application_that_facilitates_student_collaboration.pdf

[4] "Trilateration vs Triangulation – How GPS Receivers Work," 2021. [Online]. Available: https://gisgeography.com/trilateration-triangulation-gps

[5] "GSM Triangulation Without GPS," 2018. [Online]. Available: https://www.radiojitter.com/gsm-trangulation-without-gps

[6] "Global positioning system - standard positioning service performance standard," 2008. [Online]. Available: https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf

[7] G. De Angelis, V. Pasku, A. De Angelis, M. Dionigi, M. Mongiardo, A. Moschitta, and P. Carbone, "An indoor ac magnetic positioning system," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 5, pp. 1267–1275, 2014.

[8] C. Qiu and M. W. Mutka, "Self-improving indoor localization by profiling outdoor movement on smartphones," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2017, pp. 1–9.

[9] T. Dingeldein, "Bing maps vs. google maps: Comparing the big players," 2019. [Online]. Available: https://blog.capterra.com/bing-maps-vs-google-maps

[10] B. Boyd, D. Haigler, R. Pearce, X. Tang, A. Patel, and N. M. Amato, "The campus navigator."

[11] M. Ranieri. (2020, October) A new sense of direction with Live View. Date accessed: 10.02.2022. [Online]. Available: https://blog.google/products/maps/new-sense-direction-live-view/

[12] D. Glasgow. (2021, March) Redefining what a map can be with new information and AI. Date accessed: 10.02.2022. [Online]. Available: https://blog.google/products/maps/redefining-what-map-can-be-new-information-and-ai/

[13] (2021, July) Google JR. Date accessed: 10.02.2022. [Online]. Available: https://japan.googleblog.com/2021/07/google-jr.html

[14] Z. Senzer and N. Sarma. (2021, October) Photobombs begone with Magic Eraser in Google Photos. Date accessed: 10.02.2022. [Online]. Available: https://blog.google/products/photos/magic-eraser/

[15] A. Makarov. (2021, June) How Augmented Reality Navigation Systems Work. Date accessed: 08.02.2022. [Online]. Available: https://mobidev.biz/blog/augmented-reality-indoor-navigation-app-developement-arkit

[16] D. Scurtu, "UPB Campus," *Sesiunea de Comunicări Științifice Studențești*, 2020.

[17] J. Wang and Z. Lin, "A general 3d campus navigator system," in *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*. IEEE, 2009, pp. 1074–1078.

[18] R. Graham, H. McCabe, and S. Sheridan, "Pathfinding in computer games," *ITB Journal*, vol. 8, pp. 57–81, 2003.

[19] ——, "Neural networks for real-time pathfinding in computer games," *ITB J*, vol. 5, no. 1, p. 21, 2004.

[20] M. Harrower. (2020, February) To 3D or not to 3D? Date accessed: 13.02.2022. [Online]. Available: https://storymaps.arcgis.com/stories/85df1e904cbb49c8ad169be4bc927016

[21] J. Hardy. (2020) Creating an ARCore powered indoor navigation application in Unity. Date accessed: 05.05.2022. [Online]. Available: https://blog.raccoons.be/arcore-powered-indoor-navigation-unity

[22] R. Lopez Mendez. (2018) Indoor Real-Time Navigation with SLAM on Your Mobile. Date accessed: 06.05.2022. [Online]. Available: https://community.arm.com/arm-community-blogs/b/graphics-gaming-and-vr-blog/posts/indoor-real-time-navigation-with-slam-on-your-mobile

# Appendices

# Appendix A

# AirDocs initial proposal

**AirDocs (Documents in the Air)** is a middleware system that allows placing and retrieving objects or documents at different indoor locations **without** requiring a positioning system. It:

1. relies on WiFi/Bluetooth infrastructure existing in most homes and institutions
2. requires only a single additional server visible in the intranet, and an app that can be installed on any smartphone.
3. does not require training of, or support from a location service
4. enables many applications that involve natural placing / retrieving documents at locations

Indoor WiFi signal propagates in an irregular fashion due to multipath and shadowing caused by the building itself, furniture, and people. But at a given location, propagation to and from access points, while not predictable, is relatively stable in time and is specific to that location. The same can be said about GSM/4G propagation, as well as sound reflections from the walls - the collection of these measurements is unique to the location they are measured at. The wireless measurements (WiFi, 2G/3G/4G, Bluetooth) are monitored permanently by all smartphones as part of their normal functioning, requiring no additional gathering effort. We aggregate all these measurements into a "rich signature" for the physical location as shown in Figure at the right.

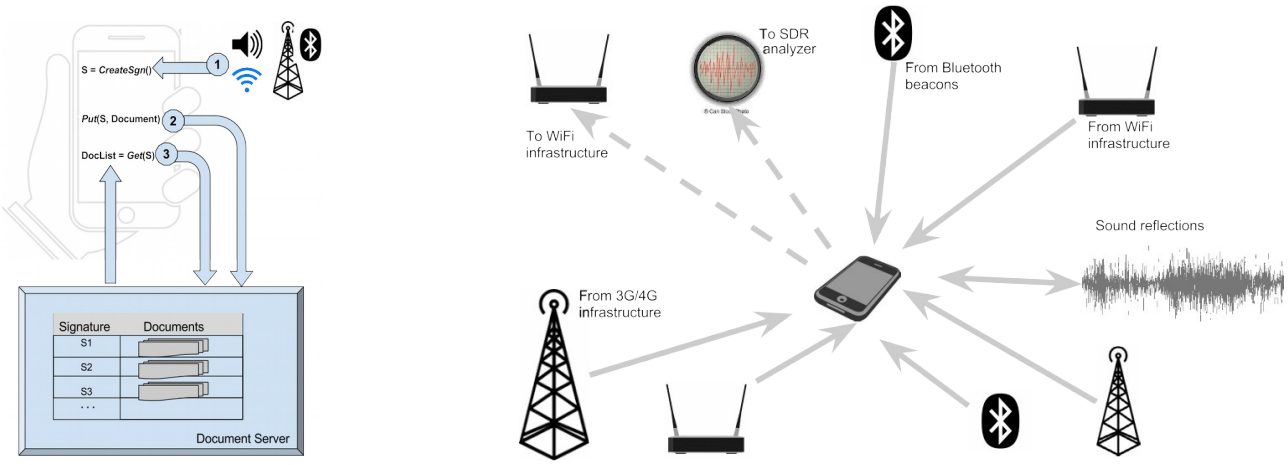


*Figure left*: Primitives used to create/use location specific signatures to index collection of documents associated with locations. *Put* function places a document "in the air" at a location identified by signature S. *Get* function retrieves document placed at signatures close to S.

As will be detailed later in the proposal, other sensors available on the smartphones can also be used: sound, magnetic, compass. The unique rich signatures obtained can then be used to manage a document collection without mapping them to geographical locations, but in fact obtaining an association between a document and its unique location in the building(but not cartesian position). The datastructure obtained is like a location indexed database, but without the actual positions which are expensive to obtain and maintain. The middleware will offer three main primitives to the applications:

1. S = *CreateSgn*() harvests location specific signal statistics from the phone sensors(WiFi, 3G, sound, etc) and creates a multidimensional signature that cannot be confused with any other location signature in the building;
2. *Put*(S, document) stores a document on the server associated with the signature S; The signature is created by a phone, but the indexing of the signatures and the document storage happen on a server in the intranet (or Internet).
3. *Get*(S) - a phone harvests its current signature, and asks the server for a list of documents that have "close" signatures, meaning documents that have been stored at the same location. The server searches in signature space, and real geographical coordinates are never stored.

What is achieved is a form of augmented reality with the users having the illusion of the documents being spread in the physical environment, only visible at certain locations. Leaving a document "in the air" allows for a natural way to use it as a wireless post-it for museums explanations, maps & directions in airports and malls, lab door announcements, restaurant menus, etc. The purpose of *AirDocs* project is to experiment with rich signatures (creation, maintenance, classification), searches in signature space, implement a proof of concept for the middleware, and several Android applications that would usually require location: wireless post-it, graffiti, guided navigation. Full blown applications will have to consider institution specific document policies (their maximum size and life length), security(who can create documents), scalability, and an appropriate human interface to facilitate production and consumption of spatial data.