EMBEDDED AND DISTRIBUTED AI - REAL TIME SLAM PROJECT REPORT

Jonatan Flyckt
School of Engineering
Jönköping University
Gjuterigatan 5, 553 18 Jönköping
fljo1589@student.ju.se

Marcus Gullstrand
School of Engineering
Jönköping University
Gjuterigatan 5, 553 18 Jönköping
guma1502@student.ju.se

May 30, 2020

ABSTRACT

Simultaneous localisation and mapping (SLAM) can be used to create maps of environments while simultaneously detecting and mapping an agent's position in these environments through the use of various sensors. The objective of our project was to create an intuitive and easy-to-use SLAM implementation that could show accurate 2D floor-plan style maps using mainly the camera sensor on mobile Android devices. By using Google's ARCore library to extract SLAM landmarks, we created an application that continuously plots and updates a 2D map using the Android Canvas library. The resulting application works well in well-lit indoor areas, as well as outdoor areas without inclination. However, the application perform poorly when there is a lot of incline, or when the scanning area is too dark to accurately detect landmarks. Our application has room for improvements with regards to processing performance, better landmark collection, and more accurate plotting. Although the application has issues, the final result is a fun and intuitive application that can be used by anyone with a mobile Android device.

1 Introduction

1.1 Background

SLAM is the technique of continuously and incrementally creating maps of environments, while also simultaneously localising the position of some agent (often a mobile robot) in these environments [1]. The SLAM problem is of great interest to the robotics community, and SLAM is used by both indoor and outdoor robots, as well as in underwater and airborn systems [1]. In recent years, SLAM has also seen successful use in commercial semi-autonomous vehicles such as Tesla cars [2]. The process of SLAM involves the agent moving through an environment and using one or more sensors to calculate an unknown number of landmarks by observing their relative position between different time frames [1]. Common sensors for SLAM are cameras, Sonars, and Lasers [3].

Figure 1 shows a simple SLAM scenario of a robot moving through an environment, simultaneously estimating its position as well as the positions of discovered landmarks. At a time instant k, there are several different data gathered. $\mathbf{x}_k$ shows the location and orientation of the robot. $\mathbf{u}_k$ shows the applied state change at time $k-1$ to move the robot to its $\mathbf{x}_k$ state. $\mathbf{m}_i$ denotes the estimated location of the observed landmarks. $\mathbf{z}_{ik}$ is the observation of the landmark i at the time k. These data quantities constitute the essence of a SLAM algorithm. [1]

For all times k, a joint posterior density probability distribution needs to be calculated to determine how accurate the mapping of an $\mathbf{m}_i$ landmark and the state $\mathbf{x}_k$ of the agent is. This is calculated using Bayes theorem recursively with the following formula: $P(\mathbf{x}_{k-1}, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})$. A map m of all landmarks can then be constructed by combining the landmarks of all the observations from different locations in the environment. [1]

Visual-Inertial SLAM (VISLAM) is a SLAM method that uses a visual sensor (camera) as well as on-device edge sensors such as gyroscope and accelerometer to accurately detect the gravitational direction, producing correct axes relative to earth. VISLAM is also generally a cheaper alternative compared to laser-based approaches. [4]

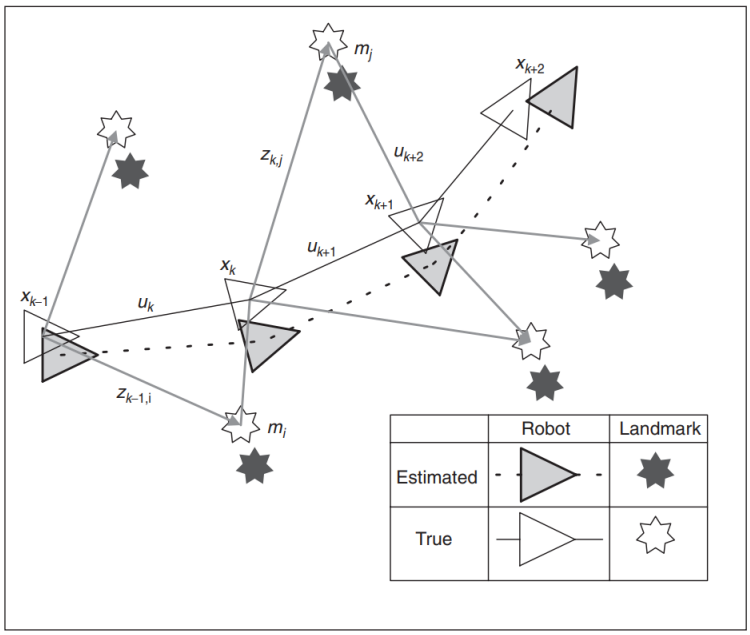

Figure 1: The SLAM problem. A robot moving through an environment and observing landmarks, estimating its own, and the landmarks' locations. Source: [1]

1.2 Problem description

In this project, we attempt to use the VISLAM implementation of Google's ARCore library [5] to map spatial environments on mobile Android devices. Mobile Android devices were chosen for several reasons:

- They are user friendly and intuitive.
- They are mobile and easy to move around with in any environment.
- Phones and other Android devices have built-in sensors in addition to camera (such as accelerometers and gyroscopes), which help in producing accurate maps of the real world with the earth as a reference point. This is often not possible when using solely a camera on a laptop or some other device.

The focus of the project is to build a user-friendly Android mobile application that produces an accurate mapping of the environment, and the device's position in that environment. We display both a live camera feed with landmarks overlayed in real-time, as well as a map plot where the landmarks have different colours based on the confidence with which the SLAM has estimated their positions. This should help the user to understand the SLAM process intuitively, and aid with gathering better landmark points, in extension generating a better overall mapping of the spatial environment.

Because we could not find any free-to-use open source 3D plotting library for Android, we elected to instead plot the map and localisation in 2D with a from-above view. This means that we have to do extensive calculations on the 3D landmark points to accurately represent the environment in a floor-plan style.

2 Method

2.1 ARCore

Google's ARCore library consists of 3 features: 'Motion tracking', 'Enviromental understanding' and 'Light estimation'. In this project we only used motion tracking, which esentially is SLAM. The other features have to do with placing 3D

objects on surfaces (e.g. tables, floors etc.) and giving them the appropriate amount of lighting or shading depending on the given lighting conditions. We excluded these two parts from the sample project that we used. [5]

Motion tracking in ARCore is the combination of processed visual information (computer vision) and spatial information about the device from inertial measurements (accelerometer data, gyroscope data, rotation etc.). The visual information, given as at least two or more feature points in an image frame, together with the phone's position and rotation relative to earth produces the combined information of points of interest in each image frame with x, y, and z axis relative to earth and the SLAM session starting direction. [5]

For every point of interest in each image frame, ARCore calculates a confidence which indicates the certainty that a point is representing an accurate description of reality. The confidence is on a scale of 0 to 1, 0 being not sure at all and 1 being so sure that it is essentially a fact. We could not find any documentation of how this confidence calculation is performed. However, the confidence values showed very good results, so we still use them in our application when deciding which landmarks to add or prune from the mapping.

2.2 Landmark processing

The landmarks are extracted from the SLAM algorithm's stream of landmarks detected by the camera sensor. Each landmark consists of four variables:

- X position in metre distance from camera origin (left and right relative to phone's starting position)
- Y position in metre distance from camera origin (forward and back relative to phone's starting position)
- Z position in metre distance from camera origin (up and down relative to the earth)
- Confidence level

Because the landmark stream updates several times per second, and can contain several hundred landmarks at a time, multiple pruning measures need to be taken when deciding which landmarks to keep and visualise. Every time a set of landmarks are extracted from the stream, they are run through a set of post-processing steps, and either removed permanently, or stored in memory and drawn on the map.

We use the calculated confidence of each landmark to decide whether it should be kept or not. However, if we used the overall confidence of all landmarks and only kept the ones with the best confidence, the map would only contain points that lie in positions that are easy to map well, removing points in hard to scan areas completely. To circumvent this problem, we divided the map into a set of grids of $0.25 * 0.25$ m, each with its own set of up to 300 landmarks. These landmarks were pruned per individual grid zone, meaning that poorly scanned areas would never be removed, but only updated if that same area was scanned again. This means that each m^2 of scanned landmark points can contain 16 grid zones and up to 4800 landmark points.

The landmarks are pruned roughly once per second. To save processing power, we keep track of which grid zones have received new landmarks since the last pruning, and only prune these zones. In addition to pruning the grid zones based on confidence, we also prune and completely remove grid zones with very few landmarks (fewer than 40). This is done to prevent incorrect outliers from being stored and displayed to the user. Another measure taken to prevent outliers is that we only add landmarks with a confidence of 0.5 or higher. We also noticed early on that sometimes landmarks that were up to 40 meters away from the scanned area could be added erronously (for instance when looking through the window of an apartment). Therefore, we only save landmarks if they are within 1.5 metres of an already existing landmark.

After pruning the new landmarks, all landmarks are iterated over to try to estimate the position of the floor in the scanned environment. We do this by identifying what Z axis 0.25 metre interval (below the average camera position) that contains the most points. This is done to make the visualisation of landmarks to the user better, and will be explained more thoroughly in the Landmark and localisation visualisation section. The step-by-step process of landmark pruning is described in the Workflow section.

2.3 Landmark and localisation visualisation

The landmarks and phone position are visualised as a 2D map in a view below the camera view in the application. Several candidate visualisation libraries (both 2D and 3D) were examined before settling on manually drawing the map for each frame with the Android standard Canvas library, which granted us the most control over what to show the user. Each landmark is drawn as a small dot on the canvas, and the position of the phone throughout a SLAM session is displayed as a line showing the movement of the phone, and the current phone position as a large dot. The canvas is redrawn 20 times per second, to make the application feel fluid. This, however, means that as more landmarks are

scanned, the application is quite draining for the phone's performance; a fully scanned environment of 50 m^2 contains 240 000 individual points. Each landmark dot is visualised on a scale from red to green, where red indicates a low confidence, and green indicates a high confidence. This is done to intuitively show the user what areas need to be scanned better.

Because the scanned area can grow continuously throughout a SLAM session, the view needs to be zoomed out as more areas are discovered. This zoom is designed to feel fluid for the user. The scale in which to display the landmarks is calculated by looking at the pixel size of the view and the extreme x and y points that have been scanned. Because landmark positions can have negative values, and are expressed as float point values as metres from the original camera position, the pixel points at which to display each single landmark needs to be recalculated for each screen update.

To show the user an accurate 2D representation, we also needed to exclude points scanned close to the floor. Without doing this, the mapping would look like a mess of points throughout the environment. To accomplish this, we calculate the average Z axis position of the phone (which is usually at roughly 1.5 metres above the floor) as well as the estimated floor position (as described in the Landmark processing section). The alpha of each point is then set to a lower value the closer to the floor the point is (with an alpha of 0 at the floor).

Figure 2 shows the application design and results from detailed scans of two separate well-lit indoor areas.

Figure 2: Screen grabs from two detailed scans of well-lit indoor areas. The results are accurate floor-plan maps where floor landmarks are excluded.

2.4 Workflow

Figure 3 shows a workflow of the running application for each landmark update iteration.

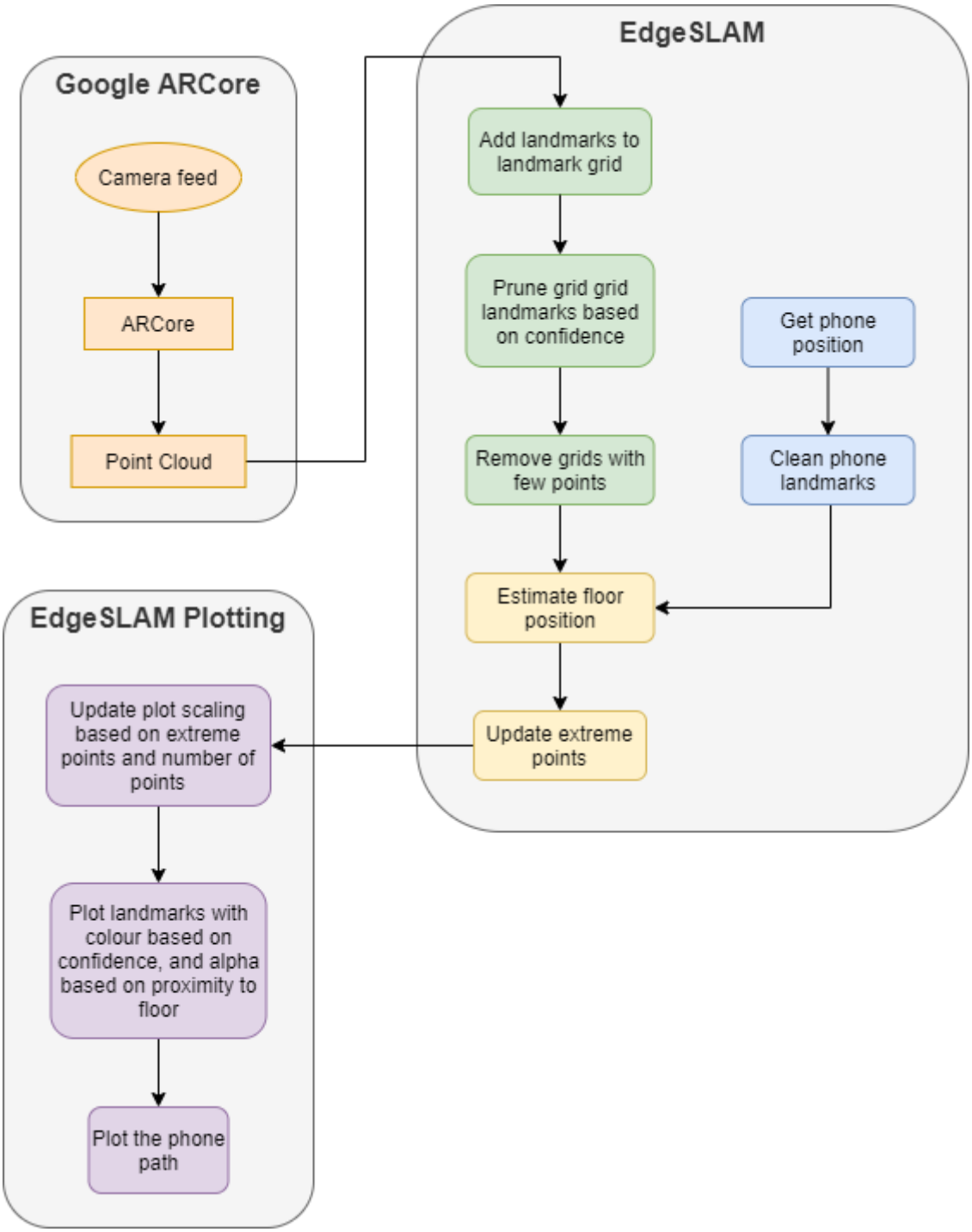

Figure 3: Workflow describing the SLAM and visualisation measures taken by our application at each iteration of landmark updates (roughly once per second depending on device).

3 Discussion and Results

3.1 Strengths and weaknesses

The overall performance of the SLAM is good when the scanning conditions are sufficient (good lighting and not a lot of dark zones). However, if the environment is too dark, the camera will be unable to accurately detect feature points (Figure 4, Figure 5). Lack of lighting is a general problem in visual-based SLAM methods [6]. The application runs smoothly on most devices. However, when the amount of scanned points reaches a certain level, the application starts to run slower.

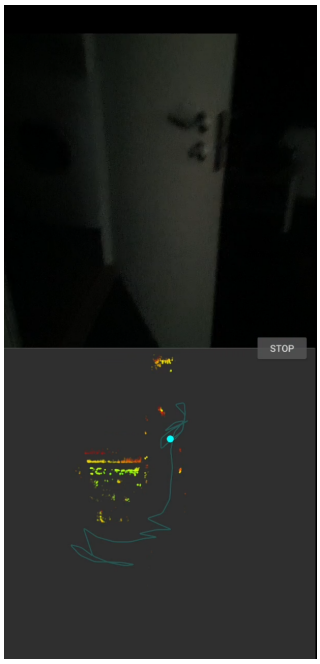

Figure 4: In dark areas, the SLAM does not manage to detect any single landmark.

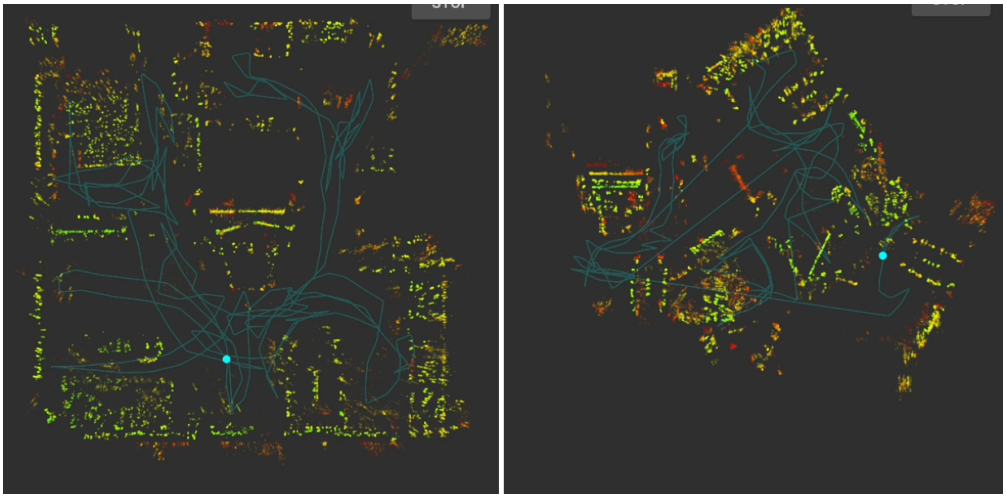

Figure 5: The difference between sufficient and insufficient lighting. The same area has been scanned, but the picture on the right has had poor lighting conditions, whereas the picture on the left has had excellent lighting conditions. The SLAM on the right has lost track of its position several times and plotted in incorrect areas. It is also clear that a lot fewer landmarks (and with worse confidence) have been picked up.

The application is very good at picking up landmark points and mapping them accurately when there are a lot of unique reference points. For instance: paintings, kitchen tiles, blankets, pillows, couches etc. all yield very good landmark points. The application is also very good at showing the user what is being scanned in the camera overlay, as well as in the real-time plot update. As with all visual based SLAMs, the performance is much worse when scanned area is "blank", i.e. it does not contain a lot of texture or colour differences. An example is a white ceiling or a white wall, which will both be poorly mapped by the SLAM algorithm.

Our application is tuned mainly to be used in flat areas, as floor level landmarks need to be excluded to produce a good map in a 2D view. This means that environments with a lot of incline can produce weird results sometimes. Because we reduce the alpha of points at an estimated floor level, incorrect reduction of alpha values can occur. This can be clearly seen in Figure 6, where the severe incline of the scanned garden causes the floor estimation to "jump" as more areas are scanned.

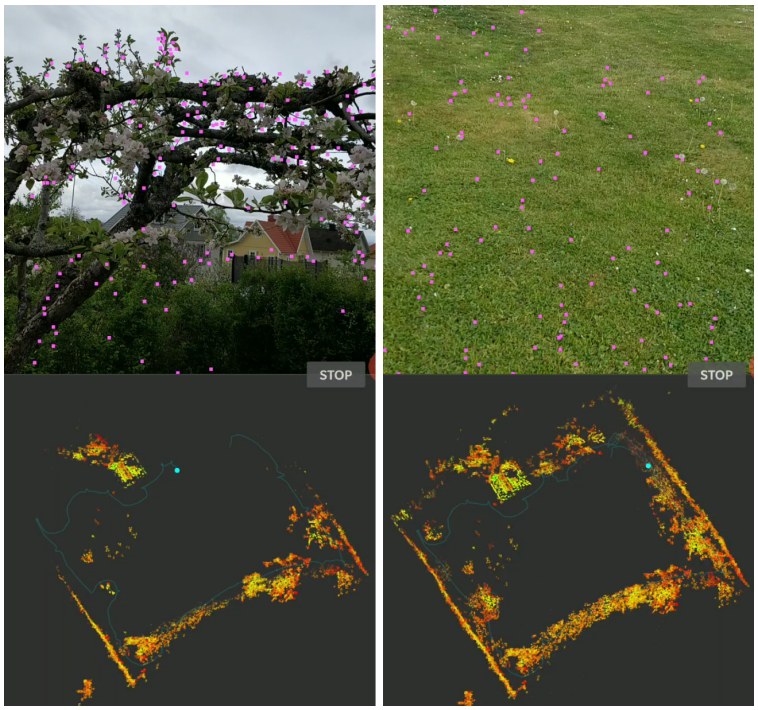

Figure 6: The effects of floor position estimation. In an area with a lot of incline, the floor position is continuously re-estimated, producing incorrect maps some of the time.

Although measures were taken to exclude outlier points scanned through windows (only points with good confidence within 1.5 metres of previously scanned areas were added), sometimes the application included outdoor points. This generally occurred when the outdoor points were on the same height as the indoor points (meaning that the points were close enough to be included) (Figure 7). Another problem with our application is that looking through mirrors caused the application to perceive the reflection as an entirely separate room. This can be seen in Figure 7. If the SLAM had been laser or sonar based, these landmarks would never be created, but we have no good suggestion for how to remove these inaccuracies with a camera-based SLAM method.

Due to time limitations, we spent limited time on examining the features of the ARCore library. ARCore performs loop closing (adjusts all landmark positions after reaching an area previously scanned), but this will not work perfectly on our application because we extract and save the landmarks in our own structure. We also have no control over the computer vision measures that ARCore takes to extract the landmarks, but have to rely on the algorithm extracting correct landmark points. ARCore also contains limited documentation as to how this is done, so it is hard to analyse how, and if, it could be done better.

Figure 7: The left image shows the SLAM inaccurately detecting mirror reflections as a separate new room. The right image shows the SLAM detecting and plotting outdoor points through a window.

3.2 Conclusions and future work

Our application is fun to use, and can be used to receive an accurate mapping of most well-lit areas, both indoor and outdoor. It is intuitive to use with points in different colours showing what areas are scanned well, and what areas need to be scanned better. The application also produces good 2D floor-plan maps by excluding floor points. However, it maps poorly in areas with a lot of incline. The application has issues when scanning dark areas. To circumvent these issues in the future, we could also make use of the phone's LED lighting automatically when dark areas are reached. This was done by [6] with good results. For the application to be useful as an instantaneous mapping for an automated vehicle or similar, the application would need to be optimised a bit more. Future work could include making the application more friendly to devices with poor hardware. We could also make use of ARCore's built-in loop closing instead of manually extracting all landmarks and processing them separately.

Acknowledgement

Thank you to Max Pettersson for assistance with editing the video in Appendix 3: Video demonstration.

References

[1] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[2] Talha Takleh Omar Takleh, N.A. Bakar, Shuzlina Rahman, Raseeda Hamzah, and Zalilah Abd Aziz. A brief survey on slam methods in autonomous vehicle. *International Journal of Engineering and Technology(UAE)*, 7:38–43, 01 2018.

[3] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics Automation Magazine*, 13(3):108–117, 2006.

[4] R. Mur-Artal and J. D. Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.

[5] Google. ARCore. `https://developers.google.com/ar/discover/concepts`, 2020. Accessed: 2020-05-23.

[6] Pei-Huang Diao and Nj Shih. Marins: A mobile smartphone ar system for pathfinding in a dark environment. *Sensors*, 18:3442, 10 2018.

Appendices

Appendix 1: Application APK

Use this link to download and install the application on an Android mobile device: EdgeSLAM APK

Appendix 2: GitHub repository

Repository link

Note: The application code is divided into two folders: "ARCore" and "EdgeSLAM". The code in the EdgeSLAM folder is produced by us (with the exception of some of the code in the SLAMActivity class), and all the code in the ARCore part is from the cloned ARCore repository.

Appendix 3: Video demonstration

YouTube video demonstrating our application running in live environments