applied sciences

MDPI

Article

Mapping Server Collaboration Architecture Design with OpenVSLAM for Mobile Devices

Jooeun Song [1] and Joongjin Kook [2,*]

[1] Department of Electronics Information System Engineering, Sangmyung University, 31 Sangmyungdae-gil, Dongnam-gu, Cheonan-si 31066, Chungcheongnam-do, Korea; jueun5840@naver.com
[2] Department of Information Security Engineering, Sangmyung University, 31 Sangmyungdae-gil, Dongnam-gu, Cheonan-si 31066, Chungcheongnam-do, Korea
* Correspondence: kook@smu.ac.kr

Abstract: SLAM technology, which is used for spatial recognition in autonomous driving and robotics, has recently emerged as an important technology to provide high-quality AR contents on mobile devices due to the spread of XR and metaverse technologies. In this paper, we designed, implemented, and verified the SLAM system that can be used on mobile devices. Mobile SLAM is composed of a stand-alone type that directly performs SLAM operation on a mobile device and a mapping server type that additionally configures a mapping server based on FastAPI to perform SLAM operation on the server and transmits data for map visualization to a mobile device. The mobile SLAM system proposed in this paper mixes the two types in order to make SLAM operation and map generation more efficient. The stand-alone type of SLAM system was configured as an Android app by porting the OpenVSLAM library to the Unity engine, and the map generation and performance were evaluated on desktop PCs and mobile devices. The mobile SLAM system in this paper is an open-source project, so it is expected to help develop AR contents based on SLAM in a mobile environment.

Keywords: mobile SLAM; mobile AR; markerless AR; SLAM; visual SLAM

Citation: Song, J.; Kook, J. Mapping Server Collaboration Architecture Design with OpenVSLAM for Mobile Devices. *Appl. Sci.* **2022**, *12*, 3653. https://doi.org/10.3390/app12073653

Academic Editor: Cheonshik Kim

Received: 11 March 2022
Accepted: 2 April 2022
Published: 5 April 2022

1. Introduction

Simultaneous localization and mapping (SLAM) technology belongs to the information communication technology (ICT) application field, and it is a technology in which a mobile device creates an environment map and calculates a location within the environment using the generated map at the same time. The field of SLAM is growing remarkably due to advances in computer vision and robotics, and there are various types, such as extended Kalman filter (EKF) SLAM [1], rapid SLAM [2], and graph-based SLAM [3,4]. Among the various SLAM technologies, visual SLAM based on machine learning and computer vision, which uses camera image data to track feature points in images and create maps, is considered to be the next-generation technology that supports industries such as automobiles, drones, and mixed reality. In particular, thanks to the spread of remote and non-contact culture caused by COVID-19, augmented reality (AR) and virtual reality (VR) application technologies, such as digital twins, are attracting attention. In addition, as the demand for indoor applications where maps do not exist and the demand for drones in military and commercial applications increase, the need for SLAM technology development is also growing.

A SLAM technology that uses various sensors that can replace markers to search for information, estimate one's location in an arbitrary space, and generate a spatial map has emerged [5]. The markerless AR system recognizes an object by detecting feature points of an object or image without prior knowledge of the environment, such as a wall or intersection, and, through this, 3D contents can be placed at a specific location. The markerless AR technology not only improves the accuracy of image analysis with the

development of SLAM technology, which is a simultaneous localization and mapping technology, but also detects the surroundings and synchronizes the current location of the targets through various sensor data [6]. In AR applications, the real-time camera pose and the distance between the camera and the object are more important, and the accuracy of SLAM system mapping and global positioning is relatively low [7].

Creating a map of SLAM requires solutions to problems such as localization. Mapping and path planning are required, and robust path planning depends on successful localization and mapping. Both problems can be overcome with SLAM techniques. Since sequential sensor information is required for SLAM, eliminating these sensor noises is crucial for the next measurement and prediction [8]. Visual SLAM requires camera-based visual SLAM technology or SLAM technology using initial measurement unit (IMU) sensors or LiDAR sensors because UAVs used in indoor spaces cannot be positioned using GPS [9].

SLAM systems can be divided into direct SLAM, which uses every pixel of the camera frame, and indirect SLAM, which uses a series of key frames and feature points. Direct SLAM has a slow processing speed, but it is possible to model the surrounding environment closely and has excellent performance, even in a featureless environment. On the other hand, indirect SLAM a has fast processing speed because it estimates the camera location using certain pixels in the image and creates a 3D map [10].

In the field of AR, ORB-SLAM [11–13] and OpenVSLAM [14], which are indirect SLAM systems that can be mainly used for robots or autonomous driving, are used. However, since these SLAM libraries are mainly developed for robots related to autonomous driving, although they are open-source software, they are rarely used for the purpose of realizing AR in mobile devices or augmented reality headsets. It is especially difficult to find open SLAM libraries for mobile devices. Therefore, AR contents for mobile devices are mostly developed using solutions provided with software development kits (SDK) by companies such as Google's ARCore, Apple's ARKit, and Maxst's SensorFusionSLAM.

In this paper, we propose a design method for an AR system that can be used in mobile devices or augmented reality headsets through a markerless-based SLAM algorithm. For the implementation of the mobile SLAM system, OpenVSLAM, which can save and load the generated maps among various open-source SLAMs and provides web visualization functions, was chosen. In consideration of real-time performance and scalability, the stand-alone type, which independently performs SLAM on a mobile device, and the mapping server type, which receives the result of SLAM from the server and processes only the visualization for the purpose of not adding to the SLAM operation overhead on the mobile device, were designed at the same time. Through the dualization of SLAM performance, users can continuously track their location along with a map generated in advance for the indoor/outdoor environment they are looking at and are able to build a service that can provide AR contents based on a more accurate location. To enable the development of visual-slam-based AR applications in a mobile environment, Unity, which is widely used as a visualization engine for Android apps, was applied to evaluate the performance of the slam system.

To verify the function and the performance of the mobile SLAM system proposed in this paper, comparisons between the stand-alone type and mapping server type were made on desktops and mobile devices using the EuRoC datasets, and the abilities for each type to generate maps in real environments were evaluated.

The remainder of this paper is organized as follows. In the following section, we describe the related works on various SLAM methods. In Section 3, the OpenVSLAM-based mobile SLAM system structure and main functions are described. In Section 4, we evaluate the results of map creation and visualization and localization accuracy using real mobile devices. Finally, we provide our conclusions in Section 5.

2. Related Works

The SLAM system proposed in this paper is a markerless-based system. The stand-alone type generates a map by directly performing SLAM operation on the mobile device, and the mapping server type performs SLAM operation and transmits the information for visualization to the mobile device by transmitting the camera image to the FastAPI-based server.

2.1. Markerless-Based AR

In the early AR systems, markers were used to connect the real world and the virtual world. The marker-based AR system recognizes a marker from an image input by a camera and represents it by overlaying 2D or 3D contents related to the marker on an image output to a display device. Simple patterns, such as QR codes or logos, are mainly used for the shape of the marker, and when the marker is recognized, the location or direction of the marker is also calculated, allowing digital 3D contents to be placed in an augmented reality application. However, the marker-based AR requires a prior appointment for a marker, and it has technical limitations, such as making tracking impossible when the camera leaves the marker. In the marker-based AR system, the three steps of marker recognition, reasoning, and representation can be configured as shown in [15]. On the other hand, the markerless-based AR system provides contents by recognizing a specific object or space instead of a pre-arranged marker, so it requires that the marker recognition, reasoning, and representation must be made on the mobile devices themselves, unlike the marker-based AR system.

In order to overcome the limitations of marker-based AR technology, a SLAM technology has emerged that uses various sensors that can replace markers to search for information, estimate one's location in an arbitrary space, and generate a spatial map. The markerless AR system recognizes an object by detecting feature points of an object or image without prior knowledge of the environment, such as a wall or intersection, and, through this, 3D contents can be placed at a specific location. The markerless AR technology not only improves the accuracy of image analysis with the development of SLAM technology, which is a simultaneous localization and mapping technology, but also detects the surroundings and synchronizes the current location of the targets through various sensor data. Representative SLAM-based markerless AR development tools include Google's ARCore, Apple's ARKit and RealityKit2, and PTC's Vuforia.

Google's ARCore provides SDKs that support multiple platforms, including mobile platforms, such as Android and iOS, to build an AR system, allowing mobile devices to detect their surroundings and interact with information about their surroundings. ARCore uses three main functions to integrate virtual contents with the real world viewed through a mobile device's camera: motion tracking, environmental understanding, and lighting estimation. Basically, ARCore performs two tasks: it tracks the location of moving mobile devices and builds its own environment. ARCore's tracking technology uses a mobile device's camera to identify specific points and track how those points move over time, combining image data and IMU data from the mobile device to track its location and direction as the mobile device moves through space. In addition to identifying feature points, flat surfaces can be detected, and the average illumination of the surrounding area can also be estimated, making it possible to build map data for the surrounding environment on its own. These functions allow you to build a totally new AR system or improve existing applications [16].

Apple's ARKit is a software framework for making AR applications, which was unveiled at WWDC17 and which enables AR technology to be implemented and used in Apple products, such as the iPhone, iPad, and Mac. Recently, Apple released RealityKit2, which integrates ARKit and SWIFT API, and these are being used in the 'Measure' application, an app for distance measurement, and the App Store app for providing Apple product information based on AR. In addition, Apple is widely applying more precise AR technology by additionally using depth information of pixels depending on the changes

in the surrounding environment, such as the spatial audio function added to the AirPod Pro and the LiDAR sensor installed in iPhone 12 Pro and iPad Pro [17].

2.2. Visual SLAM

The representative open-source-based visual SLAM algorithms are Kimera [18], PTAM [19], DSO [20], LSD-SLAM [21], ORB-SLAM2 [12], ORB-SLAM3 [13], SVO [22], and OpenVSLAM [14]. If these SLAM algorithms are classified based on the method, Kimera, PTAM, ORB-SLAM2, ORB-SLAM3, and OpenVSLAM correspond to the feature-point-based indirect SLAM algorithm. DSO and LSD-SLAM are direct SLAM algorithms that do not need to explicitly extract feature points from the image, and the SVO algorithm uses both direct and indirect SLAM methods (Figure 1).

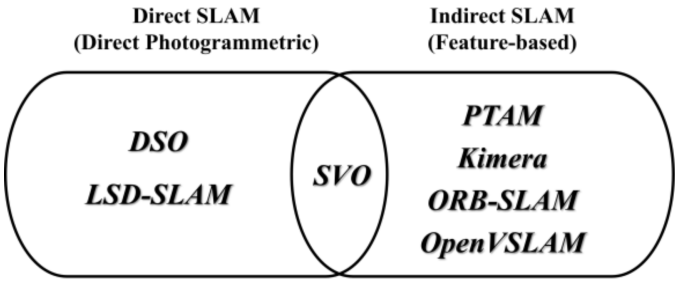


Figure 1. Classification of SLAM algorithms depending on map configuration method.

In the case of classification based on map density, there are sparse types with low map density, such as DSO, ORB-SLAM2, ORB-SLAM3, SVO, and OpenVSLAM; dense types with dense map density, such as Kimera and PTAM; and semi-dense types, such as LSD-SLAM, whose map densities are mostly higher than the sparse types and whose processing speeds are faster than most dense types. Since the comparison target is visual SLAM, the sensors of all the compared algorithms are cameras and can be divided into mono/stereo/RGBD/IMU depending on the algorithm. PTAM, DSO, and SVO do not have loop closure and optimization functions because they are VO (visual odometry). Finally, most of the point cloud maps are drawn in the form of points, but only Kimera uses a polygon mesh method (Table 1).

Table 1. Open-source-based visual SLAM algorithms.

Category	Kimera	PTAM	DSO	LSDSLAM	ORB-SLAM2	ORB-SLAM3	SVO	OpenVSLAM
Method	Indirect	Indirect	Direct	Direct	Indirect	Indirect	Hybrid	Indirect
Density	Dense	Dense	Sparse	Semi-Dense	Sparse	Sparse	Sparse	Sparse
Sensor	Mono Stereo IMU	mono	Mono Stereo IMU	mono	Mono Stereo RGBD	Mono Stereo RGBD IMU	Mono Stereo RGBD	Mono Stereo RGBD IMU
Backend	GTSAM	-	-	g2o	g2o	g2o	-	g2o
Geometry	Mesh/ TSDF	points	points	points	points	points	points	points

The adaptive monocular visual-inertial SLAM system is a system similar to the mobile AR system proposed in this paper, and it is a system that can develop real-time AR applications for mobile devices using ORB-SLAM [7]. In that study, a visual inertial odometry (VIO) method that combines a camera and an IMU sensor was designed, and a high-speed visual odometry (VO) module was designed to support the fast tracking of AR applications in a mobile environment. An adaptive execution method for selecting a tracking module was proposed by combining with the existing ORB-SLAM system and by

changing the IMU sensor value. In that paper, the system was designed for the purpose of fast tracking with lower optimization, while our paper has the purpose of accurate mapping and localization to implement an AR system in a mobile environment. Therefore, a fast SLAM system adding optimization to the VIO method is needed, and a sophisticated SLAM system that can save and load maps to continue drawing previous maps is also needed.

Therefore, in this paper, among the various open-source visual SLAM algorithms, the OpenVSLAM algorithm is used. It is (1) an indirect SLAM system that can be mainly used for robots or autonomous driving, (2) a sparse type with high speed, and (3) has the advantage of being easy to extend through the network by providing a web visualization through SocketIO communication.

OpenVSLAM is a visual SLAM system based on the graph-based indirect SLAM algorithm and based on the sparse type with low-density maps, such as ORB-SLAM, ProSLAM [23], and UcoSLAM [24]. As a feature point detector, oriented FAST and rotated BRIEF (ORB), which improved the problem of the non-directional FAST detector, was adopted as a representative image feature. The OpenVSLAM algorithm was designed by encapsulating multiple functions in components separated with a clear API (application programming interface) divided into tracking, mapping, and global optimization modules in a completely modular way. The tracking module estimates camera poses for all frames sequentially input to OpenVSLAM through key point matching and pose optimization and determines whether to insert a new keyframe. If it is determined that the current frame is suitable for a new key frame, it is transmitted to the mapping module and the global optimization module. In the mapping module, new 3D points are triangulated using inserted keyframes to create and extend a map, and a local bundle adjustment module is performed. Loop closure detection, pause graph optimization, and global bundle adjustment are performed in the global optimization module, and trajectory drift is resolved through the pose graph optimization implemented with g2o.

BA uses g2o, a library for nonlinear optimization, to calculate the pose of the current frame as a motion-only BA, the pose of a recent key frame as a local BA, and the overall optimization as a pose graph optimization of loop closing and global BA. All four optimizations run independently on different threads. Motion-only BA optimizes only the current pose with the key points on the map fixed, and local BA optimizes the pose of the co-visibility keyframe and the key points visible in the corresponding pose when a new keyframe is created. Loop closing optimizes the pose graph with only the pose, except for key points from the co-visibility graph. After optimizing the pose graph, key frames and key points newly added through the global BA are updated [10–14]. Table 2 describes terms and explanations for the main procedures of BA.

Table 2. Bundle Adjustment.

Type	Description
Motion-only BA	Optimize the pose of the current frame using the g2o library
Local BA	pose optimization of co-visibility keyframes and new keyframes
Loop Closing	pose graph optimization
Full BA	added keyframes and key point updates

OpenVSLAM is mainly implemented in C++ and uses several libraries to implement SLAM functions. Typically, it includes Eigen [25] for matrix calculation, OpenCV2 for I/O operation of image and feature point extraction, g2o [26] for map optimization, and FBoW [27] for compressing the feature point descriptions for the image to quickly use image data in loop closure. Eigen supports all matrix sizes, from small fixed-size matrices to arbitrarily large matrices, and supports all standard number types, including the complex class in the C++ standard library for representing complex numbers and the integer class for representing integers. It is also easily extensible with user-defined number types and supports geometric and decomposition functions of various matrices. OpenCV2 is a computer vision library focused on real-time image processing. By using the cv::Mat class

instead of the existing IplImage and CVMat structures for image data management, data processing or pixel access can be performed quickly and conveniently. If using the C++ API of OpenCV2 instead of the C API of OpenCV, the size and type of image data returned from the function are automatically assigned, depending on the input parameters. In addition, by automatically releasing the memory of each image datum in the class destructor, the image data management is convenient. g2o is a C++ framework for optimizing graph-based nonlinear error functions, providing solutions to several variants of SLAM and BA by finding the configuration of parameters or state variables that maximize a set of measurements affected by Gaussian noise. FBoW is a version of the DBow2/DBow3 library that is optimized to speed up the creation of a bag of words using AVX, SSE, and MMX commands, and, when loading vocabulary, it is about 80 times faster than DBoW2.

The OpenVSLAM system is compatible with various types of camera models, including fisheye, square, mono/stereo/RGBD/IMU cameras, and can be customized for the camera model and provides a cross-platform viewer running in a web browser for user convenience. The generated map can be saved and loaded, so new images can be localized using pre-made maps. Since it is implemented almost the same in perspective cameras and in fisheye cameras, AR systems can be implemented on mobile devices as well. In addition, SLAM's spatial recognition technology and map creation technology can be identified using the OpenGL-based Pangolin viewer and the WebGL-based viewer running on a web browser through NodeJS's web socket. However, since the system of this paper targets the mobile platform, we would like to check SLAM's key frame, camera pose, and point cloud by using the application through Unity Engine's Android SDK and NDK.

2.3. FastAPI-Based Mapping Server Architecture

FastAPI is one of the high-performance web frameworks that are fast enough to be comparable to NodeJS or Go based on Starlette and Pydantic and is designed based on JSON Schema, an open API standard. In addition, FastAPI has fewer bugs due to the minimization of code duplication, so debugging time can be reduced, and code can be written quickly. [28] shows the results of several tests that take multiple rows from a simple database table and serialize them into JSON messages in order to compare the performance of various web frameworks. Per request, 1, 5, 10, 15, and 20 queries were tested. FastAPI showed the second-best performance after atreugo, and, in this paper, FastAPI implemented in Python was used for development convenience.

Figure 2a shows the relationship between Uvicorn, an ASGI server, and FastAPI, one included ASGI framework. FastAPI can be composed of Starlette, a microweb framework, and Pydantic for data validation. Starlette, a subclass of FastAPI, is a web application that can be executed asynchronously. It runs on the Uvicorn server and uses Pydantic, the fastest Python validation tool, for all data processing of FastAPI. Uvicorn is a high-speed ASGI web server that performs asynchronous processing based on uvloop in a single process. Because parallel processing is not possible with Uvicorn alone, FastAPI uses Gunicorn, which can handle everything that happens between a web server and a web application as a server and process manager, making parallel processing possible. Figure 2b shows the multi-process architecture based on Gunicorn. FastAPI based on parallel processing using Gunicorn has all the performance advantages of Uvicorn and can also execute ASGI applications [28].

To implement the OpenVSALM algorithm in the server, instead of the communication method using Node.js and SoketIO, this paper focused on the ease of maintenance and on the asynchronous process communication for the background process implementation of the server. This requires a server with the same level of performance as Node.js, which was used in the existing OpenVSLAM. In the case of FastAPI, the SLAM algorithm composed of C++ is applicable, and it is an asynchronous process communication and a high-performance Python web framework. Therefore, it meets the requirements of the system in this paper.

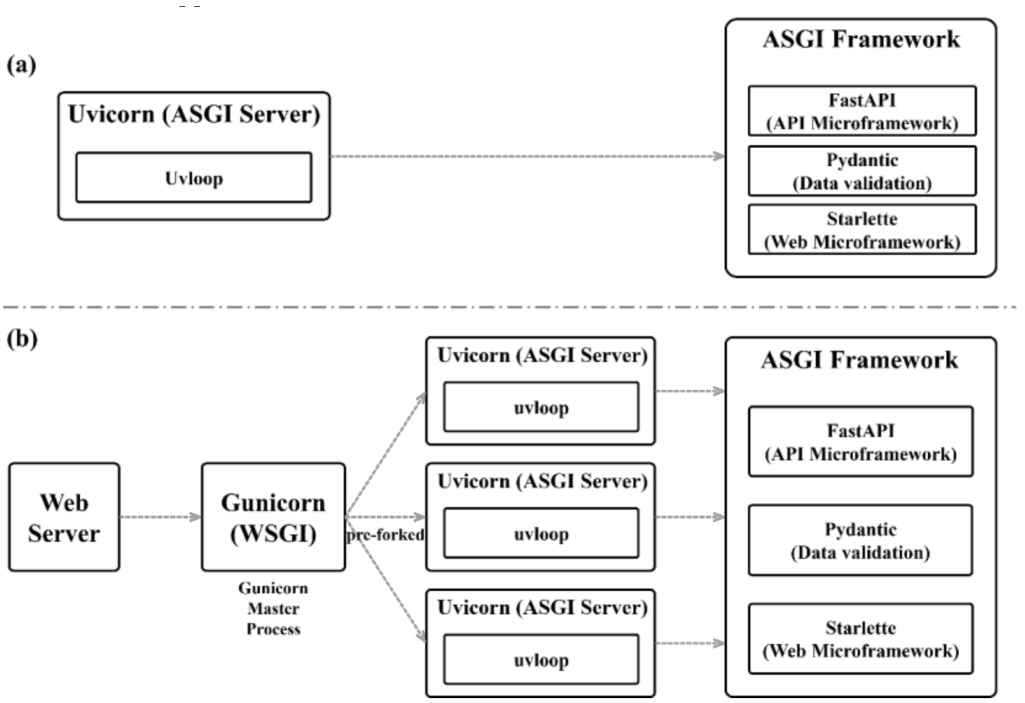


Figure 2. FastAPI-based single-process and multi-process models; (**a**) Single Process Model; (**b**) Multi-process Model.

3. Visual-SLAM-Based Mobile AR System

The mobile SLAM system proposed in this paper is a stand-alone type that directly performs SLAM operation on a mobile device and a mapping server type that transmits data for map visualization to a mobile device after performing SLAM operation on the server through a FastAPI-based mapping server. The mapping server type is a mixed architecture. Through this, real-time performance can be improved by alleviating the burden of SLAM computation on mobile devices.

3.1. OpenVSLAM-Based Mobile AR System Architecture

The mobile AR system proposed in this paper consists of sensors, networks, a Unity engine, and a FastAPI server. The image and IMU data collected by the camera of the mobile device are serialized through ProtoBuffer and transmitted to the SLAM system through Socket.IO. At this time, the SLAM system can be a SLAM system or a FastAPI server built into the Unity Android app in the form of a plugin. In the former case, SLAM is performed in the mobile device app, whereas in the latter case, SLAM can be performed through the multi-process environment considering multiple clients.

The SLAM system running on the Android app performs a function corresponding to Google ARCore and is used as an algorithm to track the camera movement and demonstrate AR technology. The SLAM system running on the FastAPI server is designed to merge the maps of multiple clients and perform localization to provide the same AR experience to multiple users by integrating data received from multiple clients. Figure 3 shows an integrated mobile SLAM architecture of the stand-alone type and mapping server type.

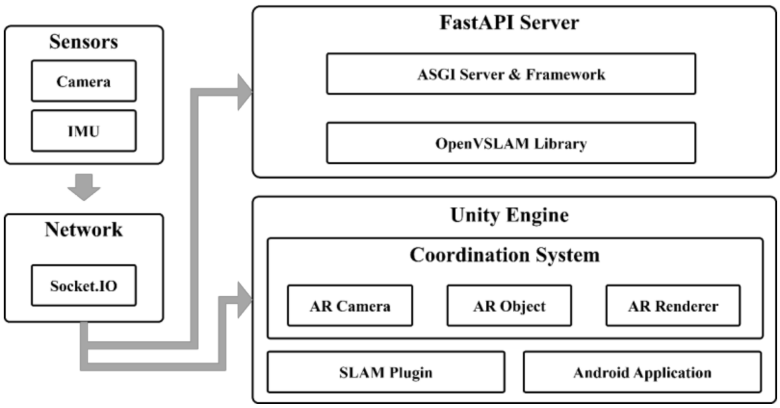


Figure 3. Integrated mobile SLAM architecture of stand-alone type and mapping server type.

3.2. Android-Based SLAM System Design

The overall workflow of the mobile SLAM system is shown in Figure 4. First, continuous image data and IMU data are collected using the camera of the mobile device. The collected data are serialized through Protobuffer and transmitted to the SLAM system inside the Unity engine through Socket.IO. The Unity engine's SLAM system estimates the approximate camera pose by extracting the ORB feature for each image datum and calculating the location of the feature point by comparing it with the nearest key frame. In order to determine the correlation with the real environment based on the camera's pose information and create a map that is as similar to the real one as possible, we draw a local map that is the map data at the time the camera is viewing using the BA (bundle adjustment) co-visibility graph that represents the relationship between key frames. To remove the accumulated drift, loop closure detection is performed on the local map, and, after optimization to reduce the map points on the global map that are obtained by adding all the local maps, localization is performed with only the camera pose, excluding the map points. BA (bundle adjustment) uses the g2o library for non-linear optimization to calculate the current frame's pose with motion-only BA, the recent key frame's pose with local BA, and the overall optimization with loop closing's pose graph optimization and full BA, and the four optimization tasks are performed independently through each thread. Motion-only BA optimizes only the current pose while fixing the key points of the map, and local BA optimizes the pose of the co-visible keyframe and the key points visible in the corresponding pose when a new keyframe is created. Loop closing performs pose graph optimization as a way of optimizing with only the pose perspective, excluding key points from the co-visibility graph, and when the pose graph optimization is finished, the added key frames and key points are updated through full BA [10–14].

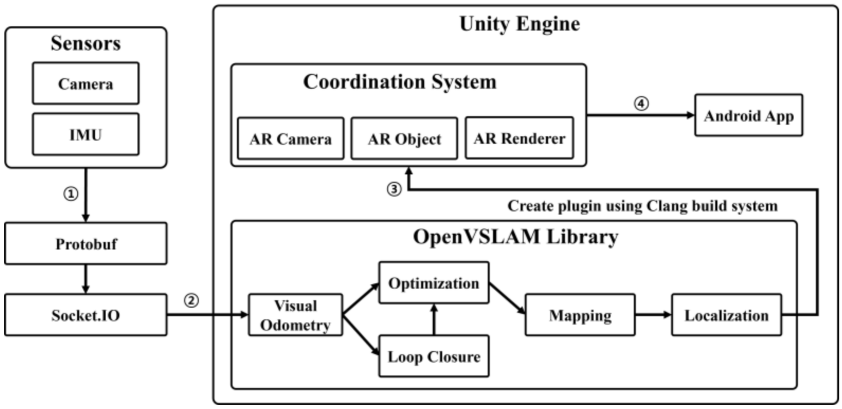


Figure 4. Mobile SLAM system architecture for stand-alone type.

In order to apply the SLAM algorithm to the Unity engine, which can only support dynamic libraries, the SLAM system must be libraryized through the Clang build of LLVM and converted into a Unity plug-in. LLVM [29] is the basis of the compiler and makes it easy to optimize programs, regardless of the writing language at compile time, link time, and runtime environment. Clang is a front-end of a compiler for programming languages such as C, C++, Objective-C, and Objective-C++ that uses LLVM as a backend and has faster compilation speed, faster execution speed, and more accurate tracking of the error location and of the error information than the GCC compiler to increase the convenience of development. In this paper, the SLAM plug-in built with Clang is imported into the Unity engine's 'DllImport ("PluginName.dll")' function to use it in the Unity engine, and the SLAM system is controlled by using the SLAM system initialization function, setting value, and function for image transfer. In this way, AR technology was implemented through the SLAM plug-in built into the Unity engine.

Pose data and point cloud data from the continuous image data collected by mobile devices are first delivered to the SLAM library to be finally provided to the Unity engine. The camera pose data may represent the camera movement as a 4×4 matrix, and the point cloud data may represent a location in space as 1×3 vector data. Through these two types of data, it is possible to visualize the current camera pose and the surrounding environment, which are saved inside SLAM.

3.3. Visualization of a Map-BasedPoint Cloud

The existing OpenVSLAM uses Pangolin to visualize SLAM's keyframes, camera poses, and point clouds based on a pre-prepared dataset to show how SLAM works. Pangolin is a lightweight and fast 3D visualization library for managing OpenGL displays and abstracting image input that is widely used in computer vision as a means to remove platform-specific boilerplate and easily visualize data [30]. In this paper, to design a SLAM system targeting the Android platform, the Unity engine was used instead of Pangolin, and the Unity API was used to control the camera and render the AR contents. The Unity engine is an authoring tool that provides a development environment for 3D and 2D video games and an integrated development tool for creating interactive contents, such as 3D animation, architectural visualization, virtual reality, and AR. It supports various platforms, such as Windows, MacOS, iOS, and Android.

The point cloud map visualization performance of OpenVSLAM was compared using the EuRoC Machine Hall 03 dataset in the existing Pangolin environment and the Unity-based environment of the mobile system. Table 3 shows the experimental environment and device specifications for comparison.

Table 3. Desktop and mobile device specifications for performance comparison of SLAM visualization.

Item	Desktop	Mobile Device
OS	Ubuntu 21.04	Android 10
CPU	AMD Ryzen5 3600X 6-Core 3.79 GHz	ARM Cortex-A77 MP1 3.09 GHz ARM Cortex-A77 MP3 2.42 GHz ARM Cortex-A55 MP4 1.80 GHz
Memory	16 GB	12 GB

The EuRoC MH 03 dataset is a visual inertial dataset collected from a micro aerial vehicle (MAV), and it contains stereo images, synchronized IMU measurements, accurate motion, and structural measurement data. It was recorded using two pinhole cameras and an inertial sensor, which were mounted on an Asctec Firefly hex-rotor drone [31]. The existing Pangolin viewer was run in the Ubuntu desktop environment, and the Unity-based mobile viewer proposed in this paper was run on a Galaxy Note 20 Ultra 5G.

Because the Pangolin viewer and the mobile viewer used the same dataset to visualize the SLAM system, the number of points in the point cloud is the same, at about 32,000. The visualization result for the EuRoC MH 03 dataset is shown in Figure 5. Figure 5a

is the visualization result using the Pangolin library on the desktop, and Figure 5b is the visualization result based on Unity in the mobile environment. Although there is a difference in the size of the point cloud and the scale of the entire map in Figure 5b, the mobile-based visualization result, it can be seen that the overall map shape is the same as in Figure 5a, the desktop result. It shows that the OpenVSLAM-based SLAM system can be visualized using Unity in the Android environment.

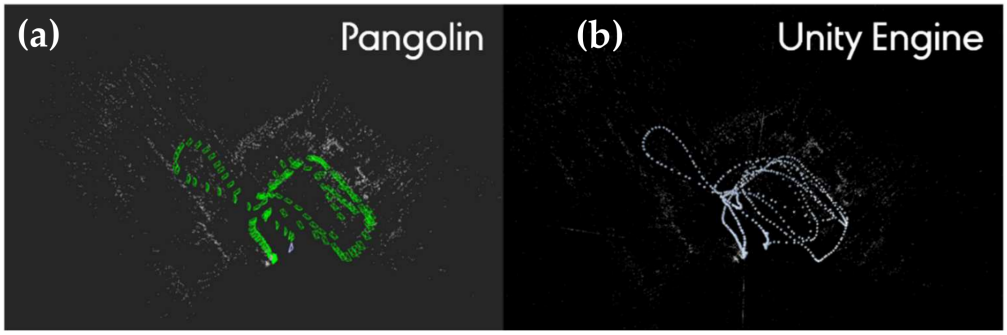


Figure 5. The result of SLAM visualization; (**a**) for desktop (Pangolin); (**b**) mobile device (Unity).

Table 4 shows the results of comparing CPU usage, memory usage, resolution, and fps for SLAM visualization performance using OpenVSLAM in the Pangolin-based desktop environment and the Unity-based mobile environment. When running the Pangolin viewer, the CPU usage was 99% and the memory usage was 23.1%, whereas in the Unity-based mobile environment, CPU and memory usage were less, 28% and 8.3%, respectively.

Table 4. Comparison of SLAM visualization performance on desktop and mobile devices.

Item	Desktop (Pangolin)	Mobile Device (Unity)
CPU Usage (%)	99	28
Memory Usage (%)	23.1	8.3
Screen Resolution	1920 × 1080	1280 × 720
Frame rate (FPS)	Avg. 30	Avg. 25

This experiment shows that SLAM visualization, even in a Unity-based mobile environment, is almost equivalent to visualization using OpenVSLAM in a Pangolin-based desktop environment. It is expected to be efficient from a multitasking perspective due to low load on CPU. However, the frame rate is slightly lower than that of the desktop environment, resulting in poor real-time performance.

3.4. FastAPI Server

SocketIO is a library that enables two-way event-based communication between a browser and a server and provides functions such as stability, automatic reconnection, packet buffering, and multiplexing through web sockets. The client sets a websocket connection, a communication protocol that provides a full-duplex and low-latency channel between the server and the browser and replaces it with HTTP long polling when a websocket cannot be connected [32]. Google's Protobuffer, which stands for protocol buffer, is a useful data structure for serializing structured data and provides SDKs for various programming languages. Data serialization is a process mainly required when developing a communication program for the purpose of wired communication or data storage. To use Protobuffer, you need to install the protocol compiler required to compile the proto file and the Protobuffer runtime for the programming language you want to use.

Figure 6 is the overall workflow diagram of the FastAPI server system. On the server side, it is more convenient to implement in the Python server rather than using the spatial

data processing, visualization, and reconstruction library implemented as C++ in Node.js. Therefore, in this paper, in order to apply the OpenVSLAM-based mobile mixed reality system as a plug-in, the Socket.IO communication module was configured in the same way as the existing Node.js event handler and was used in FastAPI. The data collected by mobile is serialized through Protobuffer, and data communication between the mobile application and the SLAM algorithm of the server is implemented using FastAPI server and SoketIO. It receives image and IMU data from the client through Socket.IO, deserializes it with Protobuffer, puts the deserialized data to the input data of the OpenVSLAM algorithm, receives camera pose data, and sends it back to the client through Socket.IO.

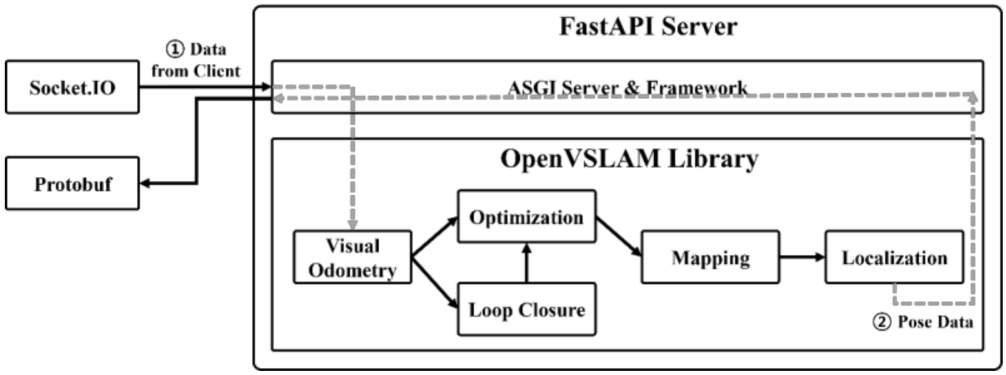


Figure 6. SLAM procedure by FastAPI-based mapping server.

Figure 7 is the overall workflow of the OpenVSLAM-based mobile AR system. Images and IMU data are collected by the user's client's mobile device camera and transmitted to the Python FastAPI server for mapping. On the mapping server, map data and camera pose data are obtained through the OpenVSLAM and transmitted to the Android application of the user client's Unity engine to visualize the global map and location.

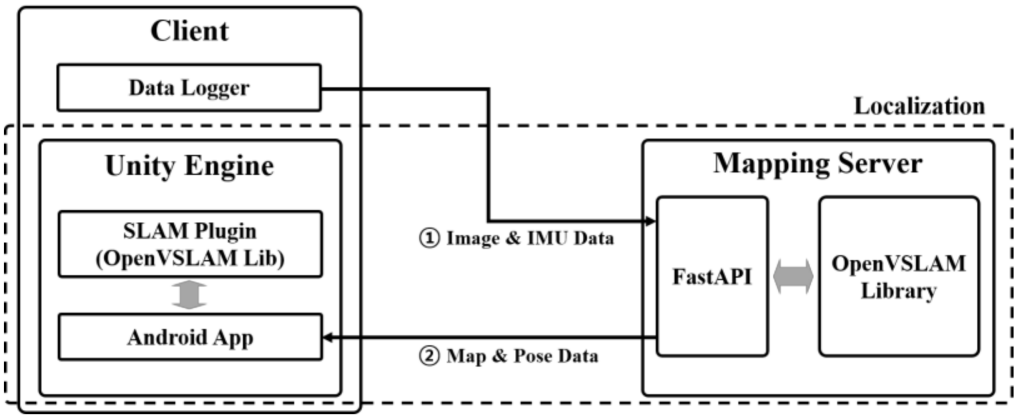


Figure 7. Interaction between mobile device and mapping server for performing SLAM based on mapping server.

4. Experiments and Results

To evaluate the function and performance of the mobile SLAM system proposed in this paper, SLAM was performed in a real indoor space and the visualization results and performance were evaluated.

4.1. Experimental Environments

Through the mobile SLAM system proposed in this paper, we tried to verify by creating and localizing a map in the real environment. To this end, image data and IMU data were collected by capturing the experimental environment in advance.

Figure 8 shows the indoor (office) experimental environment used for the experiment. The indoor space has an area of about 65 m^2, and it was captured using the camera of the Galaxy Note 20 Ultra 5G.

Figure 8. Indoor environment for mobile SLAM system evaluation.

4.2. Experimental Results

The captured image was saved as Frames.m4v. cam0 was converted to an image based on the timestamp (Frames.txt). GPS.txt was the saved location data. Accel.txt was the acceleration data. Gyro. txt was the angular velocity data. IMU.txt was the saved inertia data obtained by calculating the acceleration data and angular velocity data. config.yaml contained the camera settings. The sparse map (.bin) and sparse point cloud (.ply) files were created through the Unity engine with the image data and IMU data captured by the mobile camera. Table 5 is a description of the main files.

Table 5. SLAM performance-related files.

Filename	Description
Frames.m4v	captured image
Frames.txt	time stamp
cam0	image converted form Frames.m4v based on Frames.txt
GPS.txt	pose data
Accel.txt	acceleration data
Gyro.txt	angular velocity data
IMU.txt	inertia data
config.yaml	camera setting value
Sparse_Map_Unity.bin	sparse map created in Unity engine
Sparse_PointCloud.ply	sparse point cloud

Figure 9 is the result of comparing the partial capture of a point cloud map that matches the actual indoor space by capturing the indoor space of Figure 8 for about 2 min and 30 s in advance.

The resolution was 1280 × 720, and point clouds in the captured indoor space numbered about 54,000. The average fps was measured to be 40, and the battery consumption was very low, about 1% or less (Table 6).

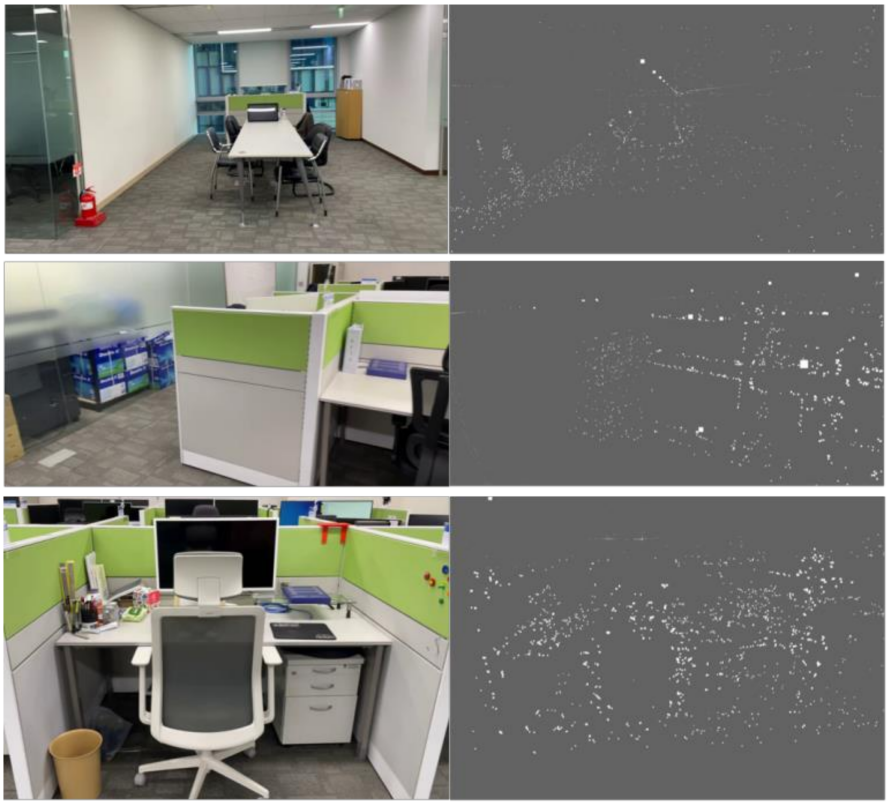


Figure 9. Map creation results by mobile SLAM system.

Table 6. Mobile SLAM system performance.

Item	Description
Display Resolution	1280 × 720
Recording Time (s)	150
Number of Points	about 54,000
Frame rate (FPS)	avg. 40

The overall consistency of the trajectory estimated in the visual SLAM system is an important factor. The absolute trajectory error (ATE) is evaluated by comparing the absolute distance between the estimate and the actual measurement. Since both trajectories can be specified in any coordinate frames, alignment is required first, in which the method of Horn [33] to find the rigid transformation, S, corresponding to the least-squares solution mapping the estimated trajectory, $P_{1:n}$, to the actual trajectory, $Q_{1:n}$, can be used. The ATE at time step i can be calculated as follows [34].

$$F_i := Q_i^{-1} S P_i \tag{1}$$

For each temporal index of the converted components, the root-mean-squared error (RMSE) can be calculated as follows.

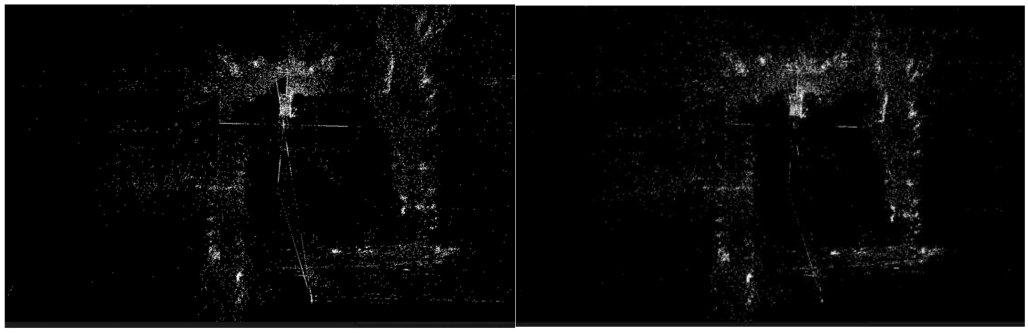
$$RMSE(F_{1:n}) := \left(\frac{1}{n} \sum_{i=1}^{n} \| \, trans(F_i) \, \|^2 \right)^{1/2} \tag{2}$$

The accuracy of the localization was evaluated by comparing it with the ground-truth data for the monocular of the EuRoC dataset, and the comparison results are shown in Table 7. The average RMSE of the monocular mobile SLAM is about 2.017, but it is not a problem for the purpose of displaying AR contents through a mobile app.

Table 7. Absolute translation RMSE, mean, and standard deviation.

Dataset	RMSE (m)	MEAN (m)	STD (m)
MH_01(Easy)	3.110	2.863	1.213
MH_03(Medium)	3.562	3.230	1.502
MH_04(Difficult)	3.308	2.976	1.445
V1_01(Easy)	0.912	0.832	0.375
V1_02(Medium)	1.796	1.675	0.650
V1_03(Difficult)	1.589	1.412	0.729
V2_01(Easy)	1.611	1.559	0.406
V2_02(Medium)	1.327	1.244	0.463
V2_03(Difficult)	0.939	0.854	0.390
Average	2.017	1.849	0.797

Figure 10 is a picture comparing the results before and after the optimization. When loop closure was detected, the point cloud map was optimized close to the scale of the real space centered on the current location, as shown in Figure 10b. Since the point cloud map represents a sparse map reconstructed with a monocular system, it may be difficult to identify objects with the naked eye. For such a problem, it is necessary to maintain the color of the point cloud and improve it so that it can be reconstructed as a close point cloud map. In addition, if the map is taken using a sensor that can obtain depth data, such as a stereo camera or RGB-D camera, it is expected that mapping and localization will be more accurate.



(**a**) (**b**)

Figure 10. Optimization results: (**a**) before optimization; (**b**) after optimization.

To verify this, the RTAB-Map application was installed on an iPhone 12 Pro with a LiDAR sensor, and the indoor environment shown in Figure 11. The RTAB-Map application for iOS was released in June 2021 and is an RGB-D based on loop closure detection, stereo, and a LiDAR graph-based ROS SLAM system. Loop closure detection uses the BoWs approach to determine the likelihood that a new image comes from a previous or new location. In addition, when loop closure is detected, it detects the previous location and draws a map, enabling real-time mapping for a large-scale environment [32]. Since the RTAB-Map SLAM system can draw a more accurate map through a depth sensor, it is estimated that adding depth data to the system in this paper will enable a clearer and more accurate AR system to be implemented in a mobile environment. Therefore, in this paper, further research and development will be conducted using the iPhone's LiDAR sensor by changing the existing Android-based data collector to an iOS-based data collector.

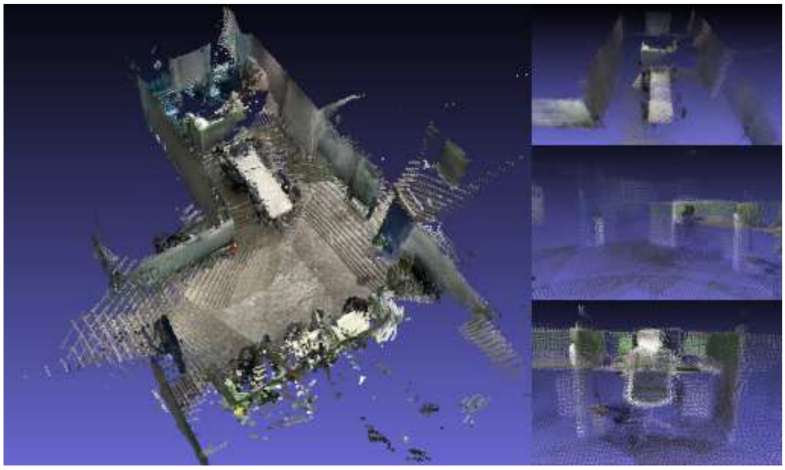


Figure 11. SLAM visualization results by RTAB-Map application.

5. Conclusions

In this paper, to reduce unnecessary self-research by researchers and contribute to the rapid growth of the SLAM field, an OpenVSLAM-based AR system was designed and implemented in a mobile environment for the purpose of open-source software, and its function and performance were evaluated not only in an actual indoor space but also using the EuRoC datasets. To implement this system, image data and IMU data were collected using the camera on a mobile device. In addition, SLAM plugins were created to use SLAM functions, including the tracking, regional and global map generation, loop closure detection, and localization functions in Unity-based mobile applications.

By building a mapping server based on the FastAPI server, implementing SLAM functions that require a lot of computation, such as mapping and optimization, on the server, and transmitting it to the mobile app, a method to reduce the overhead of SLAM operation in a mobile environment was proposed. This allows users to continuously track their location along with a map generated in advance for the indoor and outdoor environment they are looking at and to build a service that can provide AR contents based on a more accurate location. When the accuracy was evaluated by comparing the RMSE of the trajectory based on the monocular camera with the ground-truth data of the EuRoC dataset, the average of 2.017 was shown, and it was conformed that materialization of the main object is possible through the visualization of the point cloud for the actual indoor space.

Through the system in this paper, the cost of constructing an AR system for a wide range of environments can be reduced with only a mobile device and a laptop without using a separate camera or other expensive equipment. We hope that this will serve as a basis for applying SLAM technology to various application fields, such as safe and reliable autonomous driving technology for drones and vehicles, multi-access AR system implementation technology, and indoor space navigation.

In the future, based on this system, we plan to use the LiDAR sensor by changing the Android-based data collector to an iOS-based data collector. It is expected to implement a lighter system by providing a localization function to Android-based mobile devices without depth sensors after taking a map of the surrounding environment using an iOS-based mobile device.

Author Contributions: Conceptualization, J.S. and J.K.; methodology, J.S. and J.K.; software, J.S.; validation, J.S. and J.K.; formal analysis, J.S.; investigation, J.S.; resources, J.S.; data curation, J.S. and J.K.; writing—original draft preparation, J.S. and J.K.; writing—review and editing, J.K.; visualization, J.S.; supervision, J.K.; project administration, J.K.; funding acquisition, J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was carried out with the support of Green Convergence Professional Manpower Training Program of the Korea Environmental Industry and Technology Institute funded by the Ministry of Environment.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Esparza-Jiménez, J.O.; Devy, M.; Gordillo, J.L. Visual EKF-SLAM from Heterogeneous Landmarks. *Sensors* **2016**, *16*, 489. Available online: http://lps3.www.ncbi.nlm.nih.gov.libproxy.smu.ac.kr/pubmed/27070602 (accessed on 3 February 2022). [CrossRef] [PubMed]
2. Chul Roh, H.; Hun Sung, C.; Jin Chung, M. *Rapid SLAM using Simple Map Representation in Indoor Environment*; IEEE: Piscataway, NJ, USA, 2013; pp. 225–229.
3. Jung, S.; Choi, D.; Song, S.; Myung, H. Bridge Inspection Using Unmanned Aerial Vehicle Based on HG-SLAM: Hierarchical Graph-based SLAM. *Remote Sens.* **2020**, *12*, 3022. Available online: http://lps3.doaj.org.libproxy.smu.ac.kr/article/b28d031fbd6a40e59e70f9563ae11c95 (accessed on 3 February 2022). [CrossRef]
4. Fan, T.; Wang, H.; Rubenstein, M.; Murphey, T. CPL-SLAM: Efficient and Certifiably Correct Planar Graph-Based SLAM Using the Complex Number Representation. *TRO* **2020**, *36*, 1719–1737. Available online: http://lps3.ieeexplore.ieee.org.libproxy.smu.ac.kr/document/9143200 (accessed on 3 February 2022). [CrossRef]
5. Durrant-Whyte, H.; Bailey, T. Simultaneous localization and mapping: Part I. *MRA* **2006**, *13*, 99–110. Available online: http://lps3.ieeexplore.ieee.org.libproxy.smu.ac.kr/document/1638022 (accessed on 3 February 2022). [CrossRef]
6. Peddie, J. Types of Augmented Reality. In *Augmented Reality*; Springer International Publishing: Cham, Switzerland, 2017; pp. 29–46.
7. Piao, J.-C.; Kim, S.-D. Adaptive Monocular Visual–Inertial SLAM for Real-Time Augmented Reality Applications in Mobile Devices. *Sensors* **2017**, *17*, 2567. [CrossRef] [PubMed]
8. Protobuf. Available online: https://developers.google.com/protocol-buffers (accessed on 3 February 2022).
9. Aslan, M.F.; Durdu, A.; Yusefi, A.; Sabanci, K.; Sungur, C. A Tutorial: Mobile Robotics, SLAM, Bayesian Filter, Keyframe Bundle Adjustment and ROS Applications. In *Robot Operating System (ROS)*; Springer International Publishing: Cham, Switzerland, 2021; pp. 227–269.
10. Servières, M.; Renaudin, V.; Dupuis, A.; Antigny, N. Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking. *J. Sens.* **2021**, *2021*, 1–26. Available online: http://lps3.dx.doi.org.libproxy.smu.ac.kr/10.1155/2021/2054828 (accessed on 3 February 2022). [CrossRef]
11. Mur-Artal, R.; Montiel, J.M.M.; Tardos, J.D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *TRO* **2015**, *31*, 1147–1163. Available online: http://lps3.ieeexplore.ieee.org.libproxy.smu.ac.kr/document/7219438 (accessed on 3 February 2022). [CrossRef]
12. Mur-Artal, R.; Tardos, J.D. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *TRO* **2017**, *33*, 1255–1262. Available online: http://lps3.ieeexplore.ieee.org.libproxy.smu.ac.kr/document/7946260 (accessed on 3 February 2022). [CrossRef]
13. Campos, C.; Elvira, R.; Rodríguez, J.J.G.; Montiel, J.M.M.; Tardós, J.D. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *IEEE Trans. Robot.* **2021**, *37*, 1874–1890. Available online: https://arxiv.org/abs/2007.11898 (accessed on 3 February 2022). [CrossRef]
14. Sumikura, S.; Shibuya, M.; Sakurada, K. OpenVSLAM, ACM. In Proceedings of the 27th ACM International Conference on Multimedia, Nice, France, 15 October 2019; pp. 2292–2295.
15. El Filali, Y.; Krit, S. Augmented Reality Types and Popular Use Cases. In *Proceedings of the 1st International Conference of Computer Science and Renewable Energies—ICCSRE*; IEEE Digital Library: Piscataway, NJ, USA, 2019; pp. 107–110. ISBN 978-989-758-431-2. [CrossRef]
16. Anonymous ARCore. Available online: https://developers.google.com/ar/develop (accessed on 3 February 2022).
17. Apple ARKit. Available online: https://developer.apple.com/documentation/arkit (accessed on 3 February 2022).
18. Rosinol, A.; Abate, M.; Chang, Y.; Carlone, L. *Kimera: An Open-Source Library for Real-Time Metric-Semantic Localization and Mapping*; IEEE: Piscataway, NJ, USA, 2020; pp. 1689–1696.
19. Klein, G.; Murray, D. *Parallel Tracking and Mapping for Small AR Workspaces*; IEEE: Piscataway, NJ, USA, 2007; pp. 225–234.
20. Engel, J.; Koltun, V.; Cremers, D. Direct Sparse Odometry. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *40*, 611–625. Available online: http://lps3.ieeexplore.ieee.org.libproxy.smu.ac.kr/document/7898369 (accessed on 3 February 2022). [CrossRef] [PubMed]
21. Engel, J.; Schöps, T.; Cremers, D. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *Computer Vision—ECCV 2014*; Springer International Publishing: Cham, Switzerland, 2014; pp. 834–849.
22. Forster, C.; Pizzoli, M.; Scaramuzza, D. *SVO: Fast Semi-Direct Monocular Visual Odometry*; IEEE: Piscataway, NJ, USA, 2014; pp. 15–22.

23. Schlegel, D.; Colosi, M.; Grisetti, G. *ProSLAM: Graph SLAM from a Programmer's Perspective*; IEEE: Piscataway, NJ, USA, 2018; pp. 3833–3840.
24. Muñoz-Salinas, R.; Medina-Carnicer, R. UcoSLAM: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *Pattern Recognit.* **2020**, *101*, 107193. Available online: https://lps3.dx.doi.org.libproxy.smu.ac.kr/10.1016/j.patcog.2019.107193 (accessed on 3 February 2022). [CrossRef]
25. Anonymous Eigen. Available online: https://eigen.tuxfamily.org/index.php?title=Main_Page (accessed on 3 February 2022).
26. Rainer Kümmerle, g2o. Available online: https://github.com/RainerKuemmerle/g2o (accessed on 3 February 2022).
27. Muñoz-Salinas, R. Available online: https://github.com/OpenVSLAM-Community/FBoW (accessed on 3 February 2022).
28. FastAPI. Available online: https://fastapi.tiangolo.com/ (accessed on 3 February 2022).
29. Hsu, M. *LLVM Techniques, Tips, and Best Practices Clang and Middle-End Libraries*; Packt Publishing, Limited: Birmingham, UK, 2021.
30. Pangolin. Available online: https://github.com/uoip/pangolin (accessed on 3 February 2022).
31. Burri, M.; Nikolic, J.; Gohl, P.; Schneider, T.; Rehder, J.; Omari, S.; Achtelik, M.W.; Siegwart, R. The EuRoC micro aerial vehicle datasets. *Int. J. Robot. Res.* **2016**, *35*, 1157–1163. [CrossRef]
32. Socket.IO. Available online: https://socket.io (accessed on 3 February 2022).
33. Horn, B.K.P. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A Opt. Image Sci. Vis.* **1987**, *4*, 629. [CrossRef]
34. Aslan, M.F.; Durdu, A.; Sabanci, K.; Ropelewska, E.; Gültekin, S.S. A Comprehensive Survey of the Recent Studies with UAV for Precision Agriculture in Open Fields and Greenhouses. *Appl. Sci.* **2022**, *12*, 1047. [CrossRef]