HACETTEPE UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF GEOMATICS ENGINEERING

GMT404 GRADUATION PROJECT II
2021-2022
FINAL REPORT

DEVELOPMENT OF AN INDOOR POSITIONING AND NAVIGATION SYSTEM BASED ON 360-DEGREE CAMERAS

TEAM MEMBERS

Zeynep Miray ERTUNÇ	Semih GÜZEL	Emre DEMİRİZ
21733064	21733097	21733012

SUPERVISOR

Dr. Murat DURMAZ

19.05.2022

Ankara

ABSTRACT

In this project, we worked on finding the location of an AGV equipped with an omnidirectional camera in a 10x10 meter room by using the indoor positioning method and displaying its position and orientation on a visual map interface. To determine the location of the AGV, we placed markers with precise coordinates on the walls to use as reference points in the room. We detected the markers by processing the panoramic images of room obtained from the omnidirectional on the AGV. Then we used the Angle of Arrival (AOA) model, which is a positioning system, whose method is based on measuring the angular directions between one or more points with known coordinates and an unknown point to find the vehicle's position. After we produced the algorithm of the model according to our own working conditions, we gave the already known marker coordinates, the azimuth angle measurements between the markers and the AGV and the rotation angle of the AGV as input to the software and got the position of the AGV as output. As a final step, we showed the position and the orientation angle of AGV on the map produced using the Unity physic engine.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1   PROJECT INTRODUCTION

360° spherical cameras have recently started to be preferred more frequently due to the ease of use, low cost, and the development of image processing technology. The frequency of use in fields such as robotics, 3D virtual environment creation and surveillance has increased. As an example, we can give the Street View application designed by Google. Simply, with panoramic camera mounted on the car, pictures of the vehicle's surroundings are taken along the route, 3D street models are created with these pictures and served to the user via the Google Street View application. In addition, 360° spherical cameras are also preferred in indoor positioning systems where satellite technologies do not give good results. In this method, high accuracy position information can be obtained using simple equipment. In indoor positioning systems, signal-operated systems such as Wi-Fi can also be used, but as in satellite systems, this system can also be affected by signal interruptions. Camera systems stand out as they are less affected by such external factors. 360-degree camera is created by integrating more than one camera with a sum of 360° viewing angles Fig.1. In our project, an omnidirectional camera produced by stitching 2 lenses with 180° angle of view to each other was used. Fig.2. Within the scope of the project, the problem of position and orientation detection using omnidirectional camera is explained in the title 1.1.

Figure 1. *360° spherical camera created by combining 10 cameras*

Figure 2. *LG R105 omnidirectional camera*

1.1   Problem Definition and Motivation

There are many studies on indoor positioning with digital cameras in the world. The reason for this is that camera systems are less affected by external factors. To give an example, the reason why satellite systems are not sufficient for indoor positioning is that they are more affected by external factors such as signal weakening and interruptions [1]. Therefore, as we prefer, camera navigation systems can be used as an alternative for indoor positioning. In these

systems, smartphone cameras, three dimensional cameras, omnidirectional cameras etc. mostly preferred [2]. In this study, omnidirectional camera was preferred.

The main problem and motivation of the project are to find the location of an AGV with an omnidirectional camera mounted on it in a 10x10 meter indoor environment with sub-meter accuracy, its orientation with an accuracy of 2° in real-time, and display it on the map interface. In the first stage of the problem, it should be determined which method should be used to find the position and orientation of the AGV. As a result of the research, it was seen that the Angle of Arrival (AOA) method would be suitable for the problem. With this method, the position and orientation of the point whose coordinate is unknown can be calculated by measuring the azimuth and elevation angles between a point or points whose coordinates are known and a point whose coordinates are unknown. A new problem that arises in the light of this solution is to create reference points with known coordinates. Since the camera will be used in the study, the reference points to be selected must be easily distinguishable objects by the image processing algorithm to be written. For this, it was decided to place markers on the walls in a 10x10 meter room.

It is necessary to work on a simulator to test the image processing algorithm. Even if the image processing algorithm is successful in the simulation environment, it may not give the desired performance in real life. The reason for this is that the simulation environment has perfect environmental conditions. To reduce the problems that may arise in real life, it is necessary to have maximum image quality in the image processing stage. Thus, the camera to be selected should provide high-resolution images. Another problem with the camera is how to transfer the image. In this study, the captured images will be transferred to the computer for processing over the Wi-Fi connection. Therefore, a camera with Wi-Fi support should be selected. Once the appropriate camera has been selected, the system should be tested in real life. The system should be improved to reach the desired accuracy rate by using the errors obtained because of many tests. Finally, the location and orientation information of the vehicle obtained by the software should be displayed on the Map UI to be prepared.

1.2   Goals and Objectives

In this section, the goals and objectives of the project are stated. The stated goals and objectives are briefly explained in the sub-headings 1.2.1 and 1.2.2.

1.2.1 Goals

1. Developing an object detection algorithm
2. Estimating bearing angle
3. Calculating Position
4. Creating a map user interface

1.2.2 Objectives

1. Object detection algorithm will be developed with using OpenCV libraries. Thus, we will be able to detect markers from omnidirectional camera.
2. We are aiming to estimate bearing angle with under 2 ° accuracy.
3. We aim to obtain the location information of our vehicle at sub-meter precision.
4. Map user interface will be developed with using Unity physics engine.

1.3 Methodology

The project will be divided into three according to work priorities and put in order of action. These stages are determined in order of importance as camera selection, method to be used in locating, selection of objects to be used as reference, location finding algorithm, mapping, testing in simulation environment, testing in real world and finally improvement.

1.3.1 Camera Selection

A camera that will provide positioning in sub-meter sensitivity, which is the main motivation of the project, emerges as the first need. For this reason, a camera selection matrix will be produced, and the most ideal camera will be selected according to the scoring made on the table.

1.3.2 Mathematical method

Position and orientation can be obtained from spherical cameras with the epipolar geometry method. Imagine a 10x10 room with the reference point located at x0, y0. Let's assume that the position information of the uniformly distributed markers in the room is obtained from a spherical camera will be placed at this reference point. Then, a spherical camera with a distance T from our reference point is placed in the room. In this case, using the epipolar geometry method, it is possible to determine where the positions of the markers in the image obtained from the spherical camera at the reference point are in the spherical camera recently placed in the room.

Geometric Model of Spherical Camera

P is the intersection point in space of the conjugate points on two different images containing the same features, p_1 and p_2 represent the projections of this point on the spherical models seen in Figure 3.
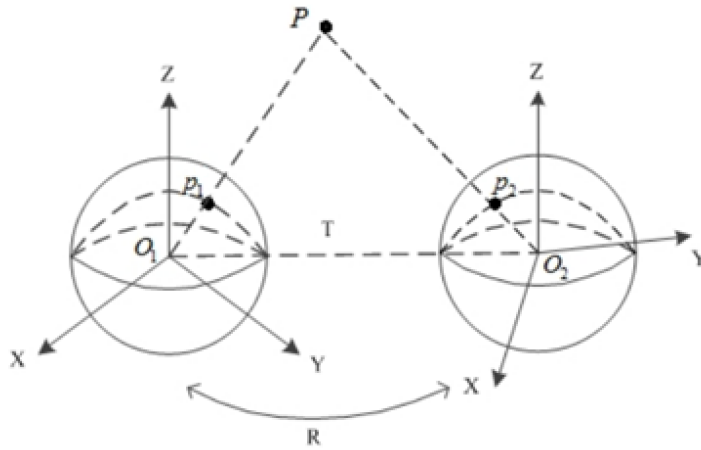
***Figure 3**. Epipolar geometry [3]*

P is the intersection point in space of the triangle plane, T is the baseline between O_1O_2, p_1 and p_2 represent the conjugate points on the images seen in Figure 3.

The rotation and coordinates of the camera O_1 at the reference point are accepted as 0. Therefore, the coordinates of p_1 on the image are known. On the image taken from the O_2 point, which is at the distance T from the O_1 camera, there is the p_2 point, which is the conjugate of the p_1 point. The vector O_1p_1 to be drawn from the center of the O_1 to the p_1 is called p_1^*. Likewise, the vector O_2p_2 to be drawn from the center of the O_2 to the p_2 is called p_2^*. To find the coordinates of the p_2 point, p_2^* must be shifted to the O_1 point. In cases where the camera plane at O_1 and the camera plane at O_2 are not parallel to each other, there will be an angle of rotation R between the two cameras. First of all, p_2^* needs to be rotated with the magnitude of R (1). Then p_2^* is shifted to the O_1 plane using the formula (2).

$$R\, p_2^* \ (1)$$

$$T \times (R\, p_2^*) \ (2)$$

When the intersections of the epipolar lines in space and the baseline line are combined, a triangular plane is formed. Since (2) is perpendicular to the co-planer, the formula (3) must be equal to 0.

$$p_1^{*T} [T \times (R\,p_2^*)] = 0 \ (3)$$

By evaluating (3), formula (4) is formed.

$$p_1^{*T} [Tx].R.p_2^* = 0 \ (4)$$

[Tx] R in formula (4) denotes Essential matrix.

$$E = [Tx].R \ (5)$$

Essential matrix can be calculated by RANSAC algorithm [4]. By using the calculated essential matrix, the rotation matrix seen in the formula (5) can be obtained with the "recoverPose" algorithm of the OpenCV library [4].

BFGS Optimization Algorithm

   The BFGS algorithm produces the hessian matrix by taking the partial second derivatives of the given inputs. In our project, the BFGS algorithm minimizes the reprojection errors in the transition from world coordinates to camera coordinates. If a function is defined according to the transformation parameters in between, this algorithm performs a kind of gradient descent search.

By using this method, the information about which direction and how much movement has been reached is reached.

1.3.3 Selecting Reference Markers

   As a result of the research done for the selection of reference points to be determined by the camera, it was decided to use markers as a reference. The markers to be used will be placed on the walls in the 10x10 room to be worked on illustrated in Figure 4. The number of markers to be used and the positions to be placed will be determined by an algorithm created on the Python software language.
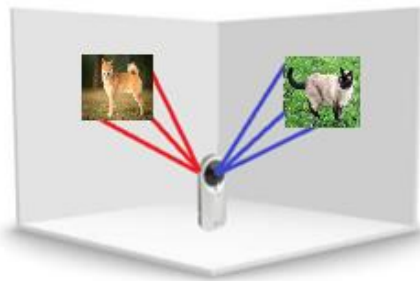
Figure 4. *Marker detection illustration*

1.3.3 Marker Detection and Position Estimation Algorithms

The OpenCV library of the Python language will be used for marker detection. Using the trackers provided by the OpenCV library, the markers on the walls will be detected by the algorithm. Then, the azimuth and elevation angles between the detected markers and the camera which connected to the AGV will be calculated from the captured image with the algorithm written in Python language, after that the location of the vehicle will be determined with mathematical formulas.

1.3.4 Mapping

After the location of the vehicle is calculated, the location information and the orientation of the vehicle will be displayed on the map user interface.

1.3.5 Testing in Simulation

With Webots, the environment to be used in real life will be transferred to the simulation environment and the written algorithms will be tested in this environment.

1.3.6 Testing in the Real Life

After the successful completion of all theoretical stages, the camera AGV system will be tested in the real working environment.

1.3.7 Improvement Phase

The tests carried out in the simulation environment will come out with high accuracy due to the perfection of the working environment, but this will not be the case in the real world. The average deviation values will be determined according to the error amounts obtained as a result of the tests performed and the improvement process will be performed according to these values.

1.4 Structure of the Report

Table 1 Structure of the report

Chapter	Title	Content
1	Introduction	What the problem is, which methods are used, and the results of the methods are explained in general.
2	Background	The information obtained as a result of the literature review about the problem was explained.
3	Requirement Analysis	All the parameters necessary for the study within the scope of the execution of the project were listed.
4	Preliminary Design of The Indoor Positioning System	The high-level design of the project was added step by step in detail. The method, mathematical formulas, software, alternative solutions and equipment to be used were demonstrated.
5	Project Planning and Management	How the project will be managed, how to establish communication between project employees, how to share information and documents, and how to publish the work were explained.
6	Implementation and Tests	Provides technical details of the implementation phase of product/solution/method.
7	Discussion	We discuss further topics related to our work and we give improvement opportunities and lessons learned from the project.
8	Conclusion	The results obtained in the first semester and the targeted studies for the next semester are explained.

2 BACKGROUND

Indoor localization is a problem that has been the subject of much research for a very long time. Today, there are many studies on solving this problem using different techniques and technologies. In this part of the report, some prominent positioning techniques and image systems that have been the subject of these studies are mentioned.

2.1 Positioning Systems

It is of great importance that a mobile robot be able to navigate safely in an unknown environment while performing the task for which it was designed. Therefore, when the robot is used in indoor positioning applications, it should be able to create a previously unknown model of this environment and use this model to predict its current position and direction. The mapping and localization of mobile robots is highly dependent on the state of the robot and known information about the environment. When the environment in which the robot will work is an indoor environment, it can be mentioned that there are basically three main techniques that are the subject of the studies.

2.1.1 Multilateration

There are many different positioning algorithms used for indoor positioning. These can be considered as triangulation and trilateration. Multilateration is based on performing the trilateration method with more than three known reference points. However, to give the technical definition of multiangulation, it should be said that multilateration is a localization technique based on distance measurements and final object location determination using the Time Difference of Arrival (TDoA) [5] methodology between the target point and at least 4 or more known reference points. An example of range-based techniques is the Received Signal Strength Indicator (RSSI). Therefore, the use of the multilateration method for RSSI-based techniques is frequently encountered in studies for indoor positioning. [6] However, RSSI methods are adversely affected by multipath propagation and shadowing that can be caused by obstacles such as furniture and human bodies indoors [7]. This can increase the positioning error. As a result of these disadvantages, the strength of wireless signals is unpredictable. Studies using log-distance models are often used to represent the signal propagation losses caused by this [8]. To mitigate the effects of these drawbacks, some studies have suggested that as a possible way to significantly improve range accuracy, the use of other physical layer measurements other than RSSI, such as time of flight of pressure waves or ancillary radio equipment such as multiple and/or directional antennas [9], maybe a solution. has been.

However, these solutions often require more energy and/or special hardware, which means more expensive devices will be needed.

2.1.2 Multiangulation

Multiangulation is a method of estimating the desired location by using reference points whose location is known and the angles between these reference points and the point whose location is desired. In a perfect test environment, it is possible to estimate the 3rd point using 2 points whose positions are known but using more reference points improves position accuracy in error-prone environments. Multiangulation has become a commonly used method for indoor positioning systems due to its characteristic features. Indoor positioning systems can be diversified as based on Computer vision and based on Communication Technology, as mentioned in [10]. Articles working on indoor positioning using the multiangulation method were generally studied in these two areas. In Computer-based systems, omnidirectional cameras are widely used. Reference points with known locations can be detected using omnidirectional cameras. From these detected reference points, it is possible to estimate the position of a mobile robot using the angle of arrival method. [11]. However, since the mathematical formulas used in this research were obtained only for this environment, they cannot be adapted to other environments.

In Communication Technologies, generally, Wi-Fi signals, Bluetooth technology, and Radiofrequency are used. Indoor positioning solutions based on the multiangulation method that covers these technologies can be produced. Angle of Arrival Cluster Forming (ACF) using radio frequencies can be implanted using phase difference in the 2.45 GHz range [12]. The authors report a localization error average of 13.5 error by using a single RFID reader and two-phase detectors. Although the obtained results are precise, the number of errors increases in the corners of the grids and in areas where the phase difference is close to 180 degrees. Martin Schüssel covered experimental results for estimating the position of a user with a smartphone[13]. They used the Angle of Arrival (AoA) method for estimating the position of the smartphone. Results have proven that the human body heavily affects the Wi-Fi signals between a smartphone and Wi-Fi access points. Therefore, the use of Wi-Fi-based AoA approaches in indoor positioning systems is not suitable to achieve high position accuracy. Achieving the automatization of industrial machines consists of a few challenges. Estimating the position and the orientation of moving machines is one of the hardest parts of this problem. This proposed solution provides a direction estimation service using Bluetooth based Angle of

Arrival (AoA) method [14]. Estimation of bearing angle between tag and anchors will be done with this technique. The proposed solution gives results as 5° maximum error of Bearing angle between anchor and tag. Albeit the proposed solution has low cost and requires limited computational power, resulted in a 5° error of bearing angle is higher than our expectations.

2.1.3   Visual Systems

Visual positioning systems work with the principle of finding positions using images taken from cameras. Markers are placed at certain locations in the environment, the visual angle between the camera and each marker is measured, and the location of the camera is estimated using the location information of the markers. Another method is storing the images previously taken by the camera in a database. Then, by matching the raw images with images in the database, the location of the camera can be found [15].

In omnidirectional vision systems, the features that appear in the images are more stable because the objects in the image stay in the field of view longer as the robot moves, which makes these systems more advantageous than traditional vision systems.

However, as with any system, in addition to the advantages of using these visual systems, there are also some disadvantages when used to model the environment. A change in the visual appearance of the work environment when the robot moves or changes in lighting conditions can cause significant changes in the appearance of the scenes. Changes may occur in the environment after the model has been created, and sometimes scenes caused by the presence of humans or other robots moving in the modeled environment may be partially covered. Therefore, important environment variables such as lighting should be considered when using this system.

System Using Interpolation

Two algorithms are used in these systems. First, The Scale-Invariant Feature Transform (SIFT) algorithm works with the principle of matching images by detecting conjugate points on images. SIFT based on the difference of Gaussians and Laplacian of Gaussian filters, the second algorithm, Speeded Up Robust Features (SURF), works for the detection of conjugate points, as in the SIFT method. Compared to SIFT, it is less accurate but a faster method [16].

In [17] the authors address image localization method for narrow corridors with smartphone camera, which adopts SIFT (Scale In-variant Feature Transform) feature to get the image closest to the input image in the database and return the position of the image. In [18], positioning with SURF, homography matrix and 3D coordinate transformation methods using images taken from users' phones is explained.

These systems, which are used for image matching in positioning systems, are not suitable for use in real time positioning systems. This is because the methods need a long time, such as a few hundred seconds, to produce results [19].

Marker Based Systems

In these systems, instead of using features in the environment, the use of highly distinguishable features added to the environment is adopted. Artificial systems that carry embedded data, such as QR code, ArUco and other fiducial markers, or traditional simple color tracking systems are preferred [20]. In [11] positioning is performed using color tracking algorithm. Different colored markers placed in 3 corners of a 100x70cm environment are detected by the object detection algorithm and the camera's position and orientation are found using the measured visual angles between the omnidirectional camera and the markers. In [21] nested hierarchical QR codes are used. The markers, whose sizes are adjusted according to the camera's viewing distance, are placed inside each other from large to small. In the Object detection section, positioning is performed using the information embedded in the QR code detected by the algorithm.

Types of Cameras Used in Visual Systems

Digital cameras such as smartphone cameras that we use in our daily life, which are getting cheaper with the developing technology, are preferred in positioning systems due to their ease of access and use. However, some cameras stand out due to their advantages. There are benefits of having a large viewing angle (FOV) of the camera to be used in the positioning system. If the viewing angle is large, the number of observed features in the environment increases. Therefore, the amount of information obtained directly increases. A standard smartphone camera's FOV is approximately 50 degrees (Figure 6). This angle of view will be insufficient in case of working in large fields [22]. For this reason, cameras with high viewing angles such as omnidirectional cameras should be used in large fields.

Figure 5. *360° omnidirectional camera illustration*

Figure 6. *Smartphone camera illustration.*

In systems based on marker detection, the camera's field of view should cover all the markers in the environment. Due to the low number of markers, the position accuracy will be low. Omnidirectional cameras solve this problem by detecting all the features in the environment with the 360-degree viewing angle they provide (Figure 5). Position accuracy below cm can be obtained by using the omnidirectional camera as seen in reference[11].

As a result, the viewing angles of the cameras to be used in visual systems should be large. Otherwise, the calculated position accuracy will decrease.

2.2 Sensors

To date, many types of sensors have been used for this purpose, such as odometry GPS, SONAR, and laser sensors. Odometry is usually calculated from the measurements of encoders installed on the wheels. It can predict the displacement of the robot, however, as a result of the accumulation of errors, it is often used in conjunction with other types of sensors. GPS (Global Positioning System) is a good choice outdoors, but there may be errors in the GPS data when used near buildings, inside buildings, or underground. SONAR (sound navigation and range) has a low cost compared to other sensors, and sound propagation of the pulses from the pulse and the receipt of the Echo is based on measuring the distance to objects in the environment. Despite this, its accuracy is relatively low, as it tends to present high angular uncertainty and some noise generated by the reflection of audio signals. Laser sensors decode the distance by measuring the time elapsed between sending and receiving laser light. Their sensitivity is higher than SONAR sensors and they can measure at large distances with better accuracy and angular resolution than SONARS. Nonetheless, the fee, weight, and power consumption of those sensors are quite excessive. Therefore, over time, vision sensors, such as omnidirectional cameras, as an alternative to these detection systems, have gained popularity due to the advantages they offer [23].

3 REQUIREMENT ANALYSIS

In this section, the usage scenarios of the project are described. Also, the requirements arising from these scenarios are mentioned. In the mentioned use case scenarios, who the actors are and the tasks they undertake are explained. Additionally, the non-functional requirements that arise in the mentioned use case scenarios are clarified.

3.1 Use Case Analysis and Scenarios

Usage scenarios consist of the installation of the system to be designed, calculating the position and orientation of the AGV, transferring the calculated position and orientation information to the AGV to provide navigation of the AGV, and finally monitoring the calculated position and orientation information with the help of a map user interface. In these usage scenarios, who the actors are, and their roles are examined under section 3.1.1.

3.1.1 Actors

In this section, the actors that will take part in the created use case scenarios are listed. These actors are Maintenance Engineer, AGV, and operator. In addition, the requirements that the actors must meet are explained.

Maintenance Engineer

The maintenance engineer is responsible for the follow-up and implementation of the tasks specified in the installation guideline, such as positioning the markers, measuring the positions where they are placed, recording the measured location in the appropriate format, and transferring them to the AGV.

AGV

In the procedure of acquiring images using a 360-degree spherical camera, the AGV carries the camera and acts as a platform to collect information in the environment. The position and orientation of the AGV must be calculated using a 360-degree spherical camera. The AGV must navigate in the working space using the location and direction information to be transmitted.

Operator

The operator must actively control the process by monitoring the position and direction of the AGV on a map user interface. Also, the operator should intervene in the accidents that may occur during the process by using the map interface.

3.1.2 Installation

In the Installation scenario, the maintenance engineer must install the developed system in the factory. It is necessary to follow the installation guide prepared for the implementation of the installation phase in an efficient and faster manner.

Installation guide

The installation guide mentioned in Section 3.1.2 includes positioning the Markers, measuring the positions at which they are placed, and recording these measured positions. In addition, throughout the user guide, the operation of the mentioned sub-scenarios and the duties of the actors who will follow the created guide are mentioned.

Marker positioning

The position and direction of the vehicle should be calculated using markers to be placed in the working environment. The placed markers need to be detected using the object detection algorithm. However, the ambient light conditions significantly affect the detectability of the markers in the object detection algorithm. Therefore, a tracker application should be developed to eliminate the problems that may arise from ambient lighting.

Measuring marker positions

The position and orientation of AGV must be derived from reference points distributed to the environment. Therefore, the true location of the markers must be known. To obtain precise location and orientation knowledge, the maintenance engineer has to measure the location markers using a total station.

Saving Measured position

The algorithm to calculate the position and direction of the AGV should take the coordinates of the markers as input. Therefore, markers whose true locations are known should be saved in an appropriate format by the maintenance engineer. Thus, the algorithm will calculate the position and direction of the AGV using markers whose positions are known.

3.1.3 Calculating Position & Orientation of AGV

First of all, the algorithm to be developed should detect the markers placed in the room from a 360-degree spherical camera. Then, a mathematical method must be established to calculate the position and direction of the AGV using the detected marks. Finally, the algorithm must be able to calculate the AGV's position with sub-meter accuracy and the bearing angle with 2 degrees precision.

3.1.4 AGV Navigation

Robotic vehicles that move autonomously without an operator or driver are called AGVs (automatic guided vehicles). To move autonomously in its environment, AGV needs real-time location and orientation information. This information calculated by the algorithm must be transferred from the computer to the AGV using communication protocols. By using this information given as input, AGV can find its own route.

3.1.5 Monitoring AGV

A map user interface is required where the operator can monitor the position and direction of the AGV. This map UI should have a label where the operator can see the location and orientation. A button should be added so that the operator can change the data collection time. In this way, it will be possible to observe the change in the position and orientation of the AGV in the desired time interval. A grid system should be displayed in the map UI to represent the size of the room. Finally, starting and stopping the algorithm must be in the operator's control. For this, it is necessary to add start and stop buttons on the Map UI. Mockup demonstration of the MAP UI can be seen in Figure 7.
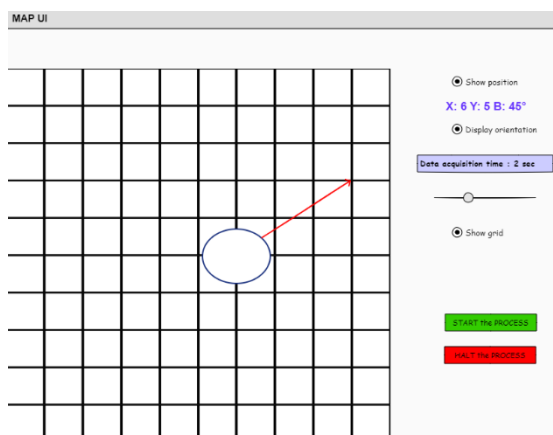
Figure 7*. Mockup Demonstration of Map UI*

3.2 Non-Functional Requirements

This section describes the non-functional requirements that occur from use cases. These requirements are clarified under the headings of usability, performance, and application requirements.

3.2.1 Usability Requirements

Selection of Marker Type: Selecting the markers in the size and shape that the camera can detect and to be placed on the wall of the working environment.

Wi-Fi Data Transfer: The images obtained from the spherical camera must be transferred to the position/orientation estimation software algorithm by means of Wi-Fi signals.

Real Time Stream: The camera to be used must be compatible with the Python OpenCV library.

3.2.2 Performance Requirements

Data acquisition: Determination of the number of positions to be calculated per unit time.

Resolution: To reduce the position accuracy in sub-meter, the resolution of the selected camera must be determined at the optimum level.

FPS: To increase the number of positions to be obtained in unit time, the number of images to be taken per second should be at the maximum level.

3.2.3 Implementation Requirements

Interior design: It must be arranged to detect markers in factory. Also, lightning of the factory must be adjusted for clear marker detection.

Library: OpenCV image processing library should be used for marker detection on images obtained from the camera.

Language: The algorithm to be used in determining the location of the AGV must be written in Python.

4   DESIGN OF THE PLACEERO

In this section, solution alternatives that can meet the requirements are explained by considering the usage scenarios. Problem-solution alternatives as pc-based solution and embedded solution were explained in sections 4.1, 4.2. A decision matrix was created to compare the solutions mentioned in section 4.3. Selection criteria were created by considering the requirements arising from the usage scenarios described in section 3.1. According to the result obtained from the solution matrix, the solution to be used in the project was selected. In section 4.4, the camera selection matrix was created by using determined criteria and the ideal camera was selected according to the result obtained from the matrix.

4.1   Pc-based Solution

Pc-based solution is a system whose components are independent and can transfer data with wi-fi signals. In this section we will briefly explain components of pc-based solution. Later, we will provide information on how to establish connections between components.

4.1.1   Components of Pc-based Solution

This section describes each component used for the PC-based solution. The components to be used are shown in Figure 8. These components are 360° camera, PC and AGV.
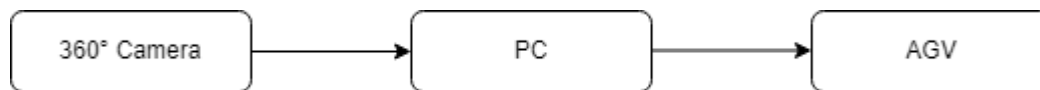
Figure 8. Components of Pc-based solution

360° Camera: It is a camera with 360-degree viewing angle that covers the entire sphere on the plane.

PC: It is used to find the location and orientation of the vehicle from the images taken by the camera.

AGV: It is a mobile vehicle which is position and orientation will be calculated, and acts as a platform to carry the camera on it.

4.1.2 Interfaces Between Components

In the first stage, it is aimed to detect the markers from the panoramic images taken from the omnidirectional camera. The images taken from the camera will be transferred to the computer via the Wi-Fi connection. The markers will be detected by the object detection algorithm that will be developed. Finally, pixel coordinates of the centers of the detected markers will be found. The Object detection algorithm will be developed using OpenCV libraries.

In the next stage, pixel coordinates of the centers of the markers will be given to the position and orientation algorithm that will be developed with Python programming language. As output, the position and orientation of the AGV will be obtained. Then the calculated data will be transferred to the AGV with the Socket communication method. External position support will be provided to the AGV thanks to the transferred data with the Socket network protocol. In this way, the navigation of the AGV will be ensured.

In final stage, the position and orientation information of the vehicle will be transferred to a map interface to be prepared on the Unity engine. Thus, the movement of the vehicle in the environment can be monitored in real time.
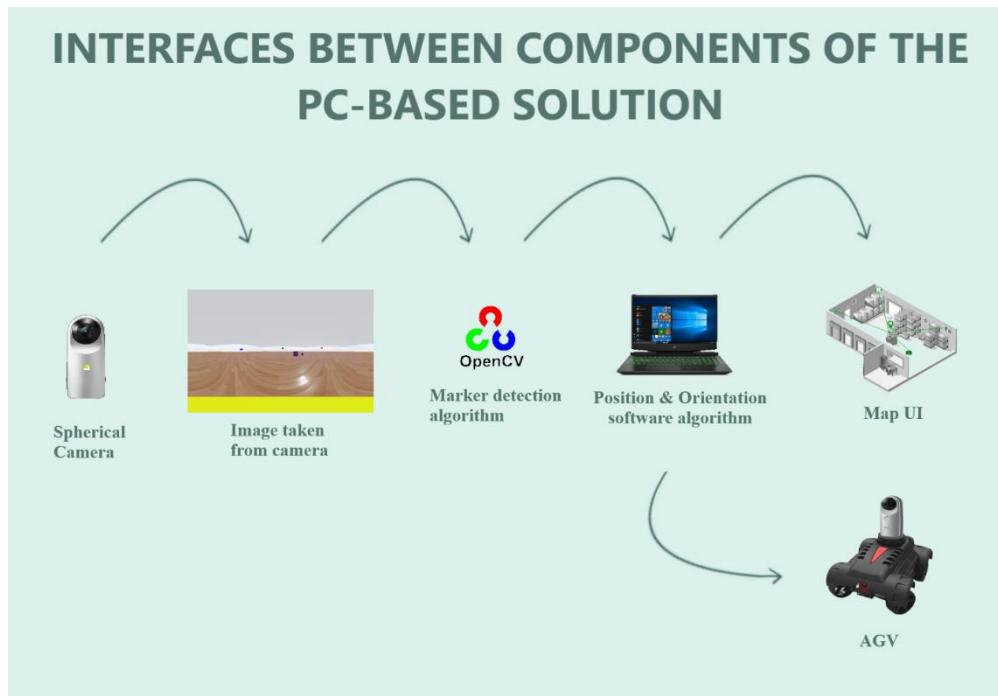
Figure 9. *Interfaces between components of the Pc-based Solution*

4.2 Embedded Solution

An embedded solution is a computing system which all components are embedded on AGV. In this section we will briefly explain components of embedded solution. Later, we will provide information on how to establish connections between components.

4.2.1 Components of Embedded Solution

This section describes each component used for the embedded solution. The components to be used are shown in Figure 10. In this system, in addition to the pc-based solution components specified in 4.1.1, a single board computer is used.
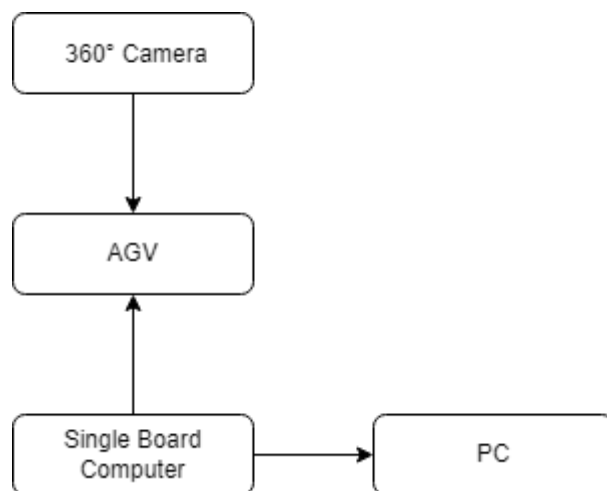
Figure 10. Components of Embedded Solution

Single Board Computer: It is used to find the location and orientation of the vehicle from the images taken by the camera.

4.2.2 Interfaces Between Components

The images taken from the camera will be transferred to the single board computer via USB connection and the markers will be detected with the object detection algorithm to be developed. The position and orientation of the AGV will be calculated using the pixel coordinates of the detected marker centers. Algorithm that will calculate position and orientation will be developed by using Python programming language. In addition, object detection algorithm will be developed with OpenCV library. Since the single board computers used in embedded solutions are integrated on the AGV, the calculated position and orientation information can be transferred to the AGV by USB cable connection.

The location and orientation information obtained on the single board computer will be transferred to the computer via Wi-Fi signals. The transferred data will be displayed in real time on the Map UI developed using the Unity physics engine.
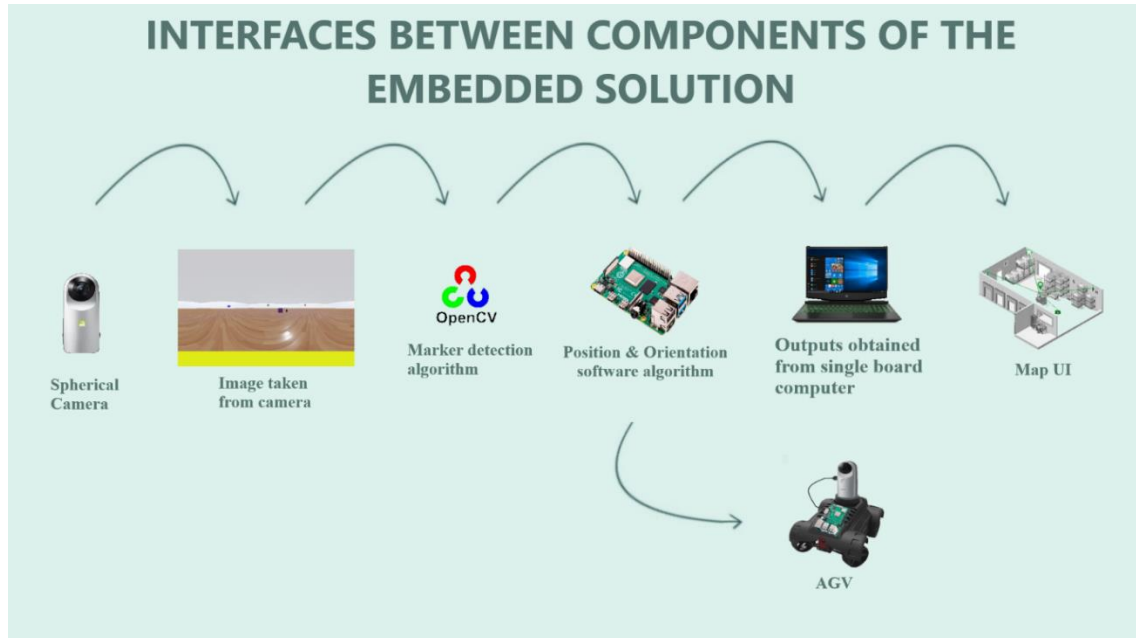
Figure 11. *Interfaces between components of the Embedded Solution*

4.3   Comparison of Solutions

This chapter compares the pc and embedded solutions described in sections 4.1 and 4.2. In order to choose the most suitable solution for the project, the necessary criteria were determined, and a decision matrix was created based on these criteria.

Table 2 *Solution Decision Matrix*

Solution Decision Matrix

Alternatives	WEIGHT	Embedded Solution	PC Solution
Requirements/ Criterias		Rating	Rating
Cost	0.4	1	4
Data transfer	0.25	4	1
Processing power	0.2	1	4
Maintenance	0.15	1	4
TOTAL	1	1.75	3.25

4.3.1 Criteria & Weight System

Weight scores were chosen according to the importance of the criteria. According to the given criteria, the system that is better than the other is represented with 4 points, and the system that is worse is represented with 1 point. The system selection was made according to the values obtained by multiplying the weight of each criteria and the rating score.

Cost

Since our project budget is low, the weight of the cost criteria was determined to be higher than the other criteria.

Considering the cost comparison of the system components in Table 3, the pc-based solution has a lower cost. Since the pc-based solution system is more suitable for our project budget, it received 4 points in the rating system.

Table 3 Cost comparison between embedded and pc-based solutions

Embedded Solution		Pc-based Solution	
Component	Cost	Component	Cost
Raspberry Pi 3B+	882₺	HP PAVILION 15-DK0005NT	15,600₺
HP PAVILION 15-DK0005NT	15,600₺	Lg -R105 Camera	899₺
Spigen Essential 10000 mAh Battery	150₺	Xbee Pro S2c Zigbee (2 pcs)	1,700₺
Cooler	140₺	Xbee Explorer USB	90₺
PiCam 360 Camera	3,700₺		
Total Cost	20,472₺		18,289₺

Data Transfer

In pc-based solution, the connection between the computer and the camera will be established over Wi-Fi signals. Data transmission speed of Wi-Fi signals may vary. In addition, packet loss may occur during data transfer. In the embedded solution, the connection between the camera and the computer is established directly with a USB cable. In this method, the data transmission rate is stable and there is less data loss. In the light of this information, 4 points were given to the embedded solution and 1 point to the pc-based solution.

The position and orientation information of the AGV will be calculated from the images

coming from the camera. Loss of data that may occur during image transfer from the camera will reduce the accuracy of position and orientation. Since the aim of the project is to achieve position accuracy in sub-meter, the data transfer criteria have been determined as the second most important criteria.

Processing Power

Since the algorithms to be used in the project do not require a very high level of processing power, the desired accuracy can be achieved with both systems. Therefore, the weight of the processing power criteria is placed in the 3rd place.

Due to the processing power of the single board computer is lower than a computer, 1 point was given to the embedded solution and 4 points to the pc-based solution.

Maintenance

Since the system to be used will not work for long period of time, the need for maintenance is the least important criteria. Therefore, its weight is given as the lowest.

In long-term use, problems such as heating, and dusting occur in embedded solution. To prevent these problems, it is necessary to perform regular maintenance on the system. However, the pc-based solution is a closed system, it requires less maintenance. Thence, 1 point was given to the embedded solution and 4 points to the pc-based solution.

4.3.2 Solution Comparison Results

According to the scoring in the decision matrix, the pc-based solution got 3.25 points out of 4, while the embedded system got 1.75 points. Since the alternative with the highest score will be chosen as the solution to the problem, the pc-based solution was preferred.

4.4 Camera Selection

Camera selection matrix was created for the selection of the camera to be used in the project. The criteria in this matrix have been determined by considering the requirements of the project. In this section, it is explained why these criteria are selected and how the criteria weights are determined.

Table 4 *Camera Decision Matrix*

Camera Decision Matrix

Alternatives	WEIGHT	Insta 360 One X2	Lg -R105	Kodak Pixpro Orbit	Gopro MAX 360°	Ricoh Theta V 360°	Samsung Gear 360	PICAM 360	GC0308 sensor 360	Xiaomi Mi 360 Camera Kit
Requirements/ Criterias		Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating	Rating
Cost	0.4	1	4	2	1	1	3	1	4	2
Resolution	0.2	4	2	4	4	4	4	2	1	4
Wi-Fi data transfer	0.2	4	4	4	4	4	4	1	1	4
FPS	0.1	3	2	2	2	2	2	2	2	2
Real Time Stream	0.1	1	4	1	1	1	1	4	4	1
TOTAL	**1**	2.37	3.4	2.64	2.24	2.24	3.1	1.63	2.63	2.7

4.4.1 Criteria & Weight System

Weight scores were chosen according to the importance of the criteria. The camera selection was made according to the values obtained by multiplying the weight of each criteria and the rating score.

Cost

Our project budget is limited to 4,000₺, the cost of the components will be used should not exceed this amount. Thence, the weight of the cost criteria was determined as the highest. Scoring of camera's costs was made according to the conditions on the Table 5.

Table 5 *Score table for camera costs*

Cost ≥ 4000	Rating score → 1
3000 ≤ Cost < 4000	Rating score → 2
2000 ≤ Cost < 3000	Rating score → 3
Cost < 2000	Rating score → 4

Resolution

In our project, it is aimed to achieve position accuracy in sub-meter. Having a high camera resolution is important to make more precise and accurate calculations. Hence, resolution was determined as the second most important criteria.

Scoring of camera's resolutions was made according to the conditions on the Table 6.

Table 6 Score table for camera resolution

Resolution ≥ 1k	Rating score → 1
1k < Resolution ≤ 2k	Rating score → 2
2k < Resolution ≤ 3k	Rating score → 3
Resolution > 3k	Rating score → 4

Wi-Fi Data Transfer

Since pc-based solution is preferred in this project, data transfer between the camera and the computer must be established with Wi-Fi signals. According to this condition, cameras with Wi-Fi module are represented with 4 points, cameras without Wi-Fi module are represented with 1 point. Considering all these conditions, the Wi-Fi Data Transfer criteria has been determined as the 3rd most important criteria.

FPS

To increase the number of positions to be obtained in unit time, the number of images to be taken per second should be at the maximum level. Consequently, resolution was determined as the fourth most important criteria.

Scoring of FPS was made according to the conditions on the Table 7.

Table 7 *Score table for camera fps*

FPS $\geq$ 20	Rating score $\rightarrow$ 1
20 < FPS $\leq$ 30	Rating score $\rightarrow$ 2
30 < FPS < 60	Rating score $\rightarrow$ 3
FPS $\geq$ 60	Rating score $\rightarrow$ 4

Real Time Stream

Object detection algorithms on OpenCV can process over real-time images or video recordings. If a Wi-Fi connection cannot be established between the camera and the computer, real-time location and orientation information will not be obtained. In this case, location and orientation information will be obtained using video recordings. Since there are two different alternatives in calculating the location and orientation information, the real time stream criteria have been determined as the criteria with the least importance. Cameras that cannot stream in real time are set as 1 point. Cameras that stream in real time are set as 4 points.

4.4.2   Camera Decision Results

For the solution of the problem, 9 different camera alternatives were selected. When the selected cameras were inserted into the camera decision matrix, the LG R105 model got the highest score of 3.1 out of 4. For this reason, the camera model to be used in the project has been determined as LG R105.

4.5   Proof of Concept Work in Simulation

In this section, it is explained that the theory that the position of the markers can be calculated in the future test and software in the solution of the problem with the image processing method can be verified with the Webots 3D robot simulator and object detection algorithm.

Step 1 - Creation of the necessary environment in the simulation:

First of all, in the Webots simulator, by selecting the wizards menu on the simulation screen, the new project directory was selected, and then by selecting the add a rectangle arena option from the options that appeared, a default new world consisting of 1 floor and 4 walls provided by the simulator was created. The created floor was converted to a size of 10x10 m using the floorSize option offered in the RectangleArena area, as shown in the Figure 12. Then, the operations were continued by playing with optional parameters such as wall height and color, which were presented in the RectangleArena option. Adding a ceiling object to completely create a room was done by sequentially selecting the add node- PROTO nodes (Webots Projects)- objects - apartment_structure - Ceiling (Solid) options. With the completion of this step, the last step of creating a room has been completed.
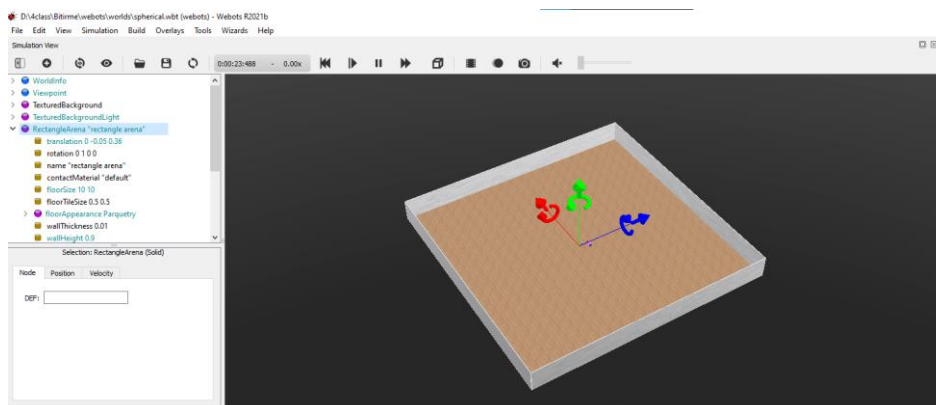
Figure 12. Created environment.

Step 2- Creation of an omnidirectional camera in simulation:

Firstly, a fixed platform was designed on which the camera would be placed in the simulator A robot node was added to the simulation to install this platform. Then camera, a child of this robot node, was added. Then the parameters were changed to create a spherical camera in the simulation. First, the spherical field from FALSE to TRUE was made TRUE by default in the camera properties. After this operation, when the spherical field was set INCORRECTLY, the value was set to 6.2831 radians for a 360-degree camera in the fieldofview field, whose value was limited to the range of 0 to π radians. Finally, the width and height areas of the camera are set to 2048 and 1080, since it has a resolution of 2k in the real world. For the camera to work in a simulation environment, a controller file was created by selecting the python software language. In this controller file, along with the name given to the camera, lines of code were written that made the camera enabled after the camera was

introduced. After saving this controller file, the file was introduced to the controller area of the robot node and the camera was made actively usable when the world was reloaded again.

Step 3- Placement of markers on the walls of the created room:

In the simulation, 8 markers that images expressing different things were placed on 4 separate walls. To create the markers, 8 separate solid nodes were added to the simulation, rectangular shapes were given to these nodes as shown in figure 13, and each different image was assigned to these rectangular shapes. The created markers must be coordinated to be used in calculations. For this reason, the markers are placed at the determined coordinates for easy tracker operation in the omnidirectional camera.

Figure 13. *Markers placed on the wall.*

Step 4 - Obtaining an image from the camera for the object detection algorithm:

Thanks to the saveImage module written using a loop to the controller file in the Webots environment, images were obtained during the simulation run. This was done for us to get photos in sync with the progress of the robot. The resulting images were automatically saved to the designated file for use in the subsequent tracking steps.
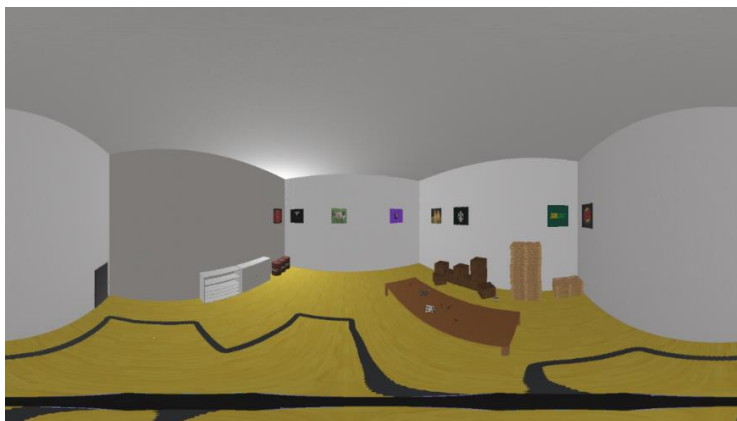
Figure 14. *Taken 360-degree camera view.*

5 PROJECT PLANING AND MANAGEMENT

The main tool used for communication is the WhatsApp application. Codes are developed using Visual Studio Code and PyCharm IDEs. The main tool ever used for document/code sharing is a workgroup channel built on the Discord platform. WeTransfer file transfer application was used for sharing that exceeded the file size restriction set for data sharing by Discord. However, in the following days, these platforms were insufficient, so the codes were shared and revised on GitHub. During the project period, the supervisor was contacted frequently, and the supervisor was informed about the progress of the project once a week or every two weeks, through face-to-face meetings or via the Zoom platform. The meetings held within the team were generally held as two or three meetings a week, and in these meetings, information was given among the teammates about the shared tasks. The meetings were mostly held on the Discord platform. How the project will be managed and who will take part in which jobs for how long are documented with the created work schedule. At the same time, risk factors that may occur during the project are listed in another table.

Table 8 *Work Schedule*

	WORK PACKAGES AND ACTIVITIES	Starting Date	Ending Date	By whom it will be done	Time	January	February	March	April	May
Work Schedule										
Preliminary Design Phase										
1.0	**Determining position**	10.01.2022	30.05.2022			11 weeks				
1.1	Establishing connection between computer and 360 camera			Zeynep Miray Ertunç	3 weeks					
1.2	Detecting markers using 360 camera using object detecting algorithms			Semih Güzel	6 weeks					
1.3	Measuring azimuth angle between vehicle and markers			Emre Demiriz	2 weeks					
2.0	**Obtaining location**	10.01.2022	30.05.2022			6 weeks				
2.1	Calculating orientation of vehicle			Zeynep Miray Ertunç	5 weeks					
2.2	Obtaining vehicle position using python			Emre Demiriz	1 week					
3.0	**Implementing map user interface**	10.01.2022	30.05.2022			8 weeks				
3.1	Creating the room plan for visual interpretation			Semih Güzel	2 weeks					
3.2	Placing room into a coordinate system			Semih Güzel	2 weeks					
3.3	Showing the orientation and the position of the vehicle on map real time			Semih Güzel	4 weeks					
4.0	**Integration & Test**	28.03.2022	30.05.2022			9 weeks				
4.1	Integrate system components			All Team Members	3 weeks					
4.2	Test & Calibration			All Team Members	6 weeks					

Table 9 *Risk Factors*

	Risk factors
1.1	Videos recorded by the camera will be used as data in case of failing to transmit real-time data from the camera using Wi-Fi signals.
1.2	In case the markers cannot be detected by the image processing algorithm, the working environment conditions will be changed (lighting quality, marker properties, etc.).
2.1	If the mathematical formula used to find the orientation contains errors in its solution, a different method will be used for the solution.
3.1	If the physics engine to be used to create a virtual working environment does not give the desired result, a different one will be preferred.
3.3	In the situation of unable to display vehicles position and orientation on map, number of images processed per second will decrease.

5.1 Work Packages

1.0 Determining Position

In this work package, the first step of the indoor positioning task to be carried out using an omnidirectional camera will be to detect the markers with the camera and to find the direction of these markers, whose positions are known, according to the vehicle position.

1.1 Establishing connection between computer and 360 Camera

In this work package, a decision will be made on how the data flow between the algorithm to be written and the camera will take place (for example, the use of the camera as a webcam or the realization of data sharing via Wi-Fi connection) and then the images will be displayed on the computer with the determined methods.

1.2 Detecting markers using 360 camera using object detecting algorithms

In this work package, markers will be detected in captured images using an object detection algorithm, such as OpenCV object tracking algorithms, as the vehicle position is determined from the directions of the markers.

1.3 Measuring azimuth angle between vehicle and markers

In this work package, since the vehicle position is determined from the directions of the markers, the angles between the vehicle and the markers must be found. This process will be calculated with the help of the image processing technique performed in 1.2.

2.0 Obtaining location

In this work package, after the parameters required for the estimation of the vehicle's location are found, the vehicle location will be determined by transferring the created mathematical formula to the software.

2.1 Calculating orientation of vehicle

Necessary analyzes and calculations will be made for the orientation of the vehicle position, and the accuracy of the calculations will be tested with the help of the algorithm.

2.2 Obtaining vehicle position using python

After calculating the orientation of the vehicle and the directions of the markers, the solution will be transferred to the algorithm that will be written in the python software language.

3.0 Implementing map user interface

In this work package, a map interface will be created to instantly monitor the location and orientation of the vehicle, a room plan will be drawn for this interface, and this room plan will be georeferenced to synchronize with the real-time application.

4.0 Integration & Testing

In this work package, it will be tested whether the system components meet the desired criteria. Improvement will be performed to meet the requirements of the system.

6 IMPLEMENTATION AND TESTS (OR APPLICATION AND RESULTS)

In this section, it is mentioned how the implementation phases are carried out by giving technical details, including the intermediate steps and also which tools are used during the processes, what kind of problems are encountered and how they are solved.

Phase 1: Establishing the simulation environment

Webots software, a 3D robot simulator, was chosen to set up the necessary environment because a 10x10 meter room and an autonomously moving robot in this room are needed to solve the problem. The factory environment provided by the Webots simulator by default was imported into the simulator, and then the dimensions of the factory were resized to be a 10x10 square meter room. In the next step, a predetermined number of rectangular markers to be placed on the walls were created. As can be seen in the Figure 15, pictures were assigned to each marker created for the detection algorithm, and these markers were fixed on the walls by changing the rotation and translation values. After meeting the room size requirements, the necessary steps were followed to create the robot that will carry the camera. Initially, the robot consists of a body and four wheels. The processes were first started by creating the body of the robot. After the body of the robot is created, as you can see in the Figure 16, the attachments required for the wheels to move with the robot, HingeJoint nodes were added, and RotationalMotor devices that provide rotational movement were added to these attachments, and the process of forming the wheels was completed. After the physical completion of the robot, a camera node was added to the robot. By default, when a camera is added, the field of view is the same as a typical camera. Therefore, to convert the camera to a spherical camera, the spherical property is set to True and the fieldOfView area is set to 6.28, the radian equivalent of 360 degrees. Finally, the width and height values of the camera are set to 1920x1080. Thus, after the camera and robot features are set, a controller file to be written in python language has been created in order to see the image given by the camera on the screen. As seen in the Figure 17, after the code lines that activate the camera were written, the controller file created was assigned to the robot and the image of the camera became visible on the screen.
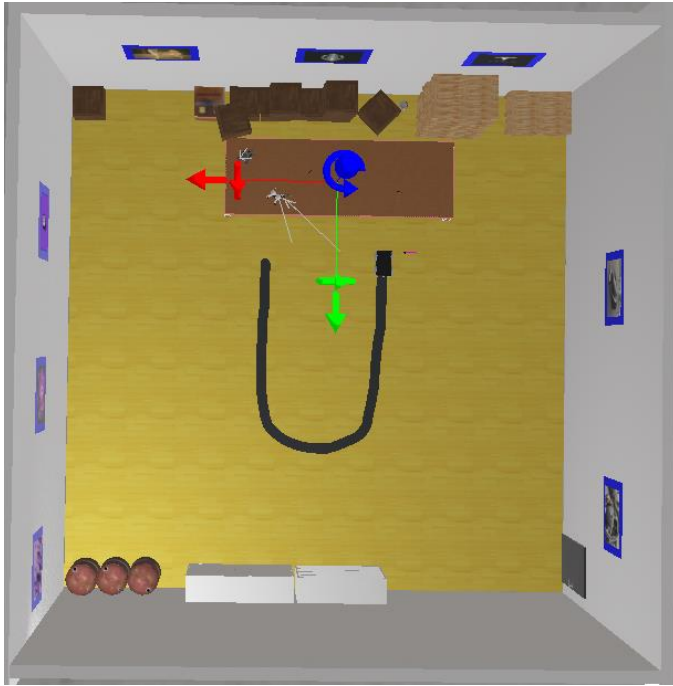
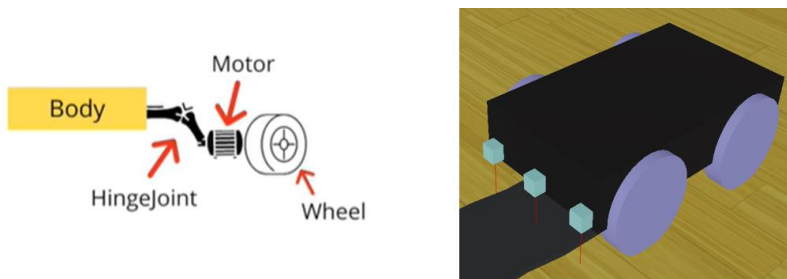*Figure 15*. *Simulation room design*

Figure 16. *Robot design*

Figure 17. *Webots controller script to active camera*

Phase 2: Taking the necessary images and measurements from the simulation environment

      The stage of taking the images started with the process of positioning the markers fixed on the wall at the determined coordinates so that they can be used in the location finding algorithms we created. The pictures were assigned a certain height, taking into account the situations such as the objects created in the factory environment may prevent the markers from appearing in the camera. At the same time, the camera height has been increased considering these situations. Then, the lines of code required for the robot to move autonomously, such as speed determination, were written to the controller file, as seen in the Figure 18. In order to obtain images synchronized with the motion during the robot's performance, a loop was used to save the images to a specified file as long as the simulation continued. GPS and IMU nodes are integrated into the robot in order to receive the coordinates and orientation information that changes during the movement of the robot, at specified second intervals. The values obtained from these GPS and IMU were pulled from the controller file and saved in text files. This process is done to compare these values with the results of the location finding algorithm, which will be mentioned in the following stages, and to measure the accuracy between the estimated and actual value.

```
24    left_front_motor = robot.getDevice('motor_1')
25    left_back_motor = robot.getDevice('motor_3')
26    right_front_motor = robot.getDevice('motor_2')
27    right_back_motor = robot.getDevice('motor_4')
28
29    left_front_motor.setPosition(float('inf'))
30    left_back_motor.setPosition(float('inf'))
31    right_front_motor.setPosition(float('inf'))
32    right_back_motor.setPosition(float('inf'))
33
34    left_front_motor.setVelocity(0.0)
35    left_back_motor.setVelocity(0.0)
36    right_front_motor.setVelocity(0.0)
37    right_back_motor.setVelocity(0.0)
38
39    gps = robot.getDevice('gps')
40    gps.enable(timestep)
41    imu = robot.getDevice('imu')
42    imu.enable(timestep)
43
44    right_ir = robot.getDevice('ds_right')
45    right_ir.enable(timestep)
46    mid_ir = robot.getDevice('ds_middle')
47    mid_ir.enable(timestep)
48    left_ir = robot.getDevice('ds_left')
49    left_ir.enable(timestep)
50
51    # Main Loop:
52    gps_values = []
53    imu_values = []
54    i=0
```

```
56    while robot.step(timestep) != -1:
57        # Read the sensors:
58        # Enter here functions to read sensor data, like:
59        left_speed = max_speed
60        right_speed = -max_speed
61        left_ir_value = left_ir.getValue()
62        right_ir_value = right_ir.getValue()
63        mid_ir_value = mid_ir.getValue()
64
65        # print("left {} mid {} right {}"\
66        # .format(left_ir_value, mid_ir_value ,right_ir_value))
67
68        left_speed = -max_speed
69        right_speed = -max_speed
70
71        if left_ir_value < 700 and right_ir_value <700 and mid_ir_value >=700:
72            left_front_motor.setVelocity(left_speed)
73            right_front_motor.setVelocity(right_speed)
74
75
76        if left_ir_value < 700 and right_ir_value >= 700 and mid_ir_value >=700:
77            left_front_motor.setVelocity(left_speed)
78            left_back_motor.setVelocity(left_speed)
79            right_front_motor.setVelocity(0)
80            right_back_motor.setVelocity(0)
81
82        if left_ir_value >= 700 and right_ir_value < 700 and mid_ir_value >=700:
83            left_front_motor.setVelocity(0)
84            left_back_motor.setVelocity(0)
85            right_front_motor.setVelocity(right_speed)
86            right_back_motor.setVelocity(right_speed)
87
88        if left_ir_value < 700 and right_ir_value < 700 and mid_ir_value <=700:
89            left_front_motor.setVelocity(left_speed)
90            right_front_motor.setVelocity(right_speed)
```

Figure 18. *The code that enables the autonomous movement of the robot*

Phase 3: Object Detection and Tracking

Classic object detection algorithms are not suitable for our project. This is because we work with a 360-degree camera. Since the images taken from the camera we use are spherical projection, some parts of the image are distorted. Object detection algorithms produced in accordance with perspective projections where distortions occur completely fail at the marker detection stage. In order to eliminate this problem, equirectangular images taken from the camera were converted to Cube Map format (Figure 19). This process was done by dividing the equirectangular image, which has 360 degrees horizontally and 180 degrees vertically, into 6 90-degree parts. Of the 6 resulting pieces, 4 represent the image on the horizontal axis (respectively: left, front, right, back) and 2 represent the image on the vertical axis (respectively: top, bottom) (Figure 20). When switching from the Equirectangular format to the Cube Map format, the picture is switched from the panoramic plane to the perspective plane. In this way, the distortions due to the very wide viewing angle on the image are eliminated.

```python
def e2c(e_img, face_w=256, mode='bilinear', cube_format='dice'):
    '''
    e_img:  ndarray in shape of [H, W, *]
    face_w: int, the length of each face of the cubemap
    '''
    assert len(e_img.shape) == 3
    h, w = e_img.shape[:2]
    if mode == 'bilinear':
        order = 1
    elif mode == 'nearest':
        order = 0
    else:
        raise NotImplementedError('unknown mode')

    xyz = utils.xyzcube(face_w)
    uv = utils.xyz2uv(xyz)
    coor_xy = utils.uv2coor(uv, h, w)

    cubemap = np.stack([
        utils.sample_equirec(e_img[..., i], coor_xy, order=order)
        for i in range(e_img.shape[2])
    ], axis=-1)

    if cube_format == 'horizon':
        pass
    elif cube_format == 'list':
        cubemap = utils.cube_h2list(cubemap)
    elif cube_format == 'dict':
        cubemap = utils.cube_h2dict(cubemap)
    elif cube_format == 'dice':
        cubemap = utils.cube_h2dice(cubemap)
    else:
        raise NotImplementedError()
```

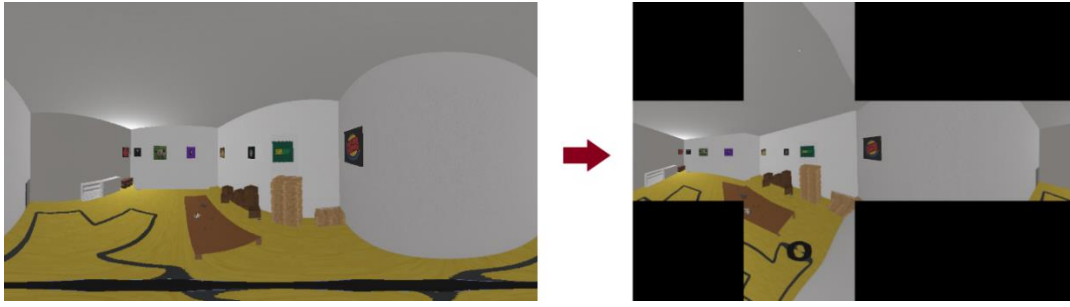*Figure 19*. *Equirectangular to Cubemap format converter code*

Figure 20. *Equirectangular to Cubemap transformation*

The images are ready for object detection by removing the distortions. For this process, methods such as machine learning and simple color and shape detection have been tried.

Object detection algorithms tried on the basis of the above-mentioned ones give low accuracy results. There are even cases where algorithms cannot detect markers in some images. Within the scope of our study, all of the markers should be detected in each image. Under these conditions, object detection algorithms will not work on images acquired in real time.

As a result of our research, it was observed that the tracking algorithm would be more suitable for our project than the object detection algorithm.

Tracker Algorithm

We preferred to use the most sensitive CSRT tracker algorithm to track the markers on the image converted to Cubemap format. The biggest problem we encountered during this period is that when the moving vehicle turns, the tracker algorithm cannot detect when the markers are passing from the left face to the back face or from the back face to the left face on the image. To solve this problem, we re-projected the cube format image at the moments when the markers would pass. We achieved the reprojection process by combining the cube face that the marker will switch to and the cube face it was on before transitioning (Figure 21).
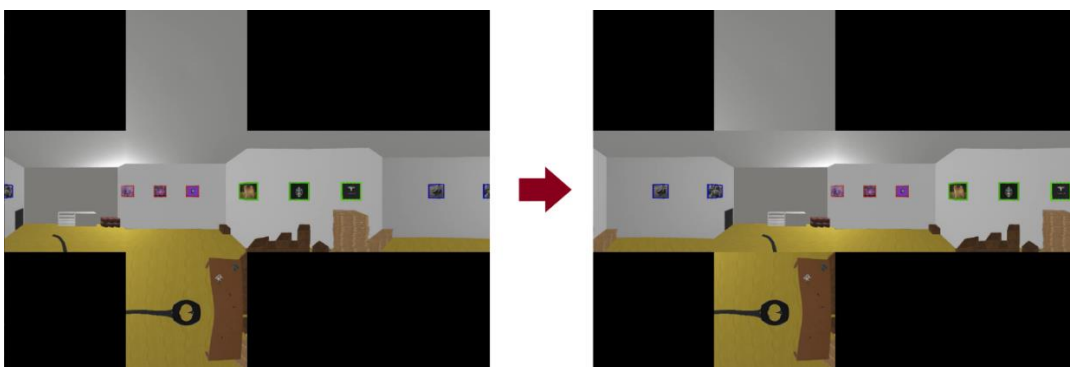

Figure 21. *Reprojection of Cubemap*

Since the places of the markers on the image have changed after the projection process, we added or subtracted 256 pixels from their old positions in order to find their new positions. However, since this process constantly reprojects the image in scenarios where the vehicle moves fast such as sudden turns, the marker positions it detects become corrupted, and for this reason, it cannot track the image after a while and the algorithm closes with an error.

For this reason, we added a blue frame to the markers so that the marker positions can be determined correctly after the reprojection process. After this process, our color detection algorithm detected the borders of each blue frame and fed the tracker algorithm (Figure 22).

Figure 22. Blue framed markers

Although adding a blue frame to the markers gave accurate results in finding their new position after reprojection, the color detection algorithm failed to give the initially determined marker order correctly, since the frame of each marker was the same color. As a final improvement, we removed the projection system and removed undetected markers from the tracker algorithm. However, since the calculated position accuracy decreases when the number of markers decreases, we ensured that the markers removed from the tracker algorithm are included in the algorithm again when they are detectable on the image. The Python code covering this part is shown on the figure (Figure 23).

```python
if int(leftmost) <= 0:
    first_sticker = boxes[0]
    stickers_deleted_left = stickers[0]
    stickers = np.delete(stickers, 0, 0)
    pixel_stickers = np.delete(pixel_stickers, 0, 0)
    boxes = np.delete(boxes, 0, 0)
    old_coordinates = boxes
    k = len(boxes)
    sticker_select = True
    scene_change = True
    add_new_marker = True
    leftest_marker = True
elif int(rightmost+rightmost_w)  >= 1024:
    last_sticker = boxes[-1]
    stickers_deleted_right = stickers[-1]
    stickers = np.delete(stickers, -1, 0)
    pixel_stickers = np.delete(pixel_stickers, -1, 0)
    boxes = np.delete(boxes, -1, 0)
    old_coordinates = boxes
    k = len(boxes)
    sticker_select = True
    scene_change = True
    add_new_marker = True
    rightest_marker = True
```

Figure 23. Python script that removes marker if it's not detectable

Phase 4: Map User Interface

We used unity physics engine to develop map UI. The first problem we encounter in the development procedure is that we use python to develop the mathematical aspect of the project and the object detection algorithm. Since Unity uses C# as its programming language, we could not directly use the python script in Unity. Although there are some alternatives where you can use python in unity, most of the libraries we worked in python did not work on Unity since the python version was 2.7x. To solve this problem, we thought we could save the AGV's coordinate in each frame in txt file format. Then we read that txt file in unity and then passed the coordinates of AGV to Map UI for visual display. However, this solution did not work well because the creation and reading of txt files affected the fps of the pointer tracking algorithm. Also, syncing this process wasn't as easy as it seemed, resulting in a delay between the coordinates we calculated and the results we saw in the MAP UI. Then we came up with a solution where we can pass the computed arguments in python to unity using the Socket communication protocol. With TCP/ICP Socket connections, you can connect both your C# and python script to the same port. After that you need to convert your string shown in Figure 24 to bytes, by doing this you can send data from python to C# script. Then in the C# script you have to encode the data obtained from the python script to string as bytes (Figure 25). The second problem we encountered during the development process was that the AGV was

teleported to the location of the first location at the start of the simulation. We solved this problem by creating an input scene shown in Figure 26. So, taking the starting position of the AGV from the user, we saved these coordinates as scriptable objects. Scriptable objects (Figure 27) are one of the properties of unity and are very useful in such situations. With these scriptable objects, you can store or render data and pass it to other scenes without losing it. So, we passed the AGV's start position to the next scene and accessed these objects and changed the AGV's view position before the scene was loaded. However, another problem arose. Even if you could set the starting position of the AGV, the python script was supposed to send you some data constantly. But in UI design, position and orientation of AGV had to be observed when the user pressed the start button. We solved this problem by making another connection with C# and python shown in Figure 28. In the input scene we send a parameter to python if the user presses the submit button, in these connection strings we pass the starting position of the AGV to python. Thus, we access the starting position of the AGV without starting the simulation. By accessing the start position, we pass the starting position of the AGV in the dialog array and send it back to C#. In this way, we observe the AGV in the same location as the starting location (Map UI display) until the user presses the start button shown in Figure 29. By pressing the start button, we are sending back a parameter to python that sets the start_clicked bool to true, as shown in Figure 24. With the help of the single if condition, we access the function that allows us to get data from the python algorithm we use to find the location and the orientation of the AGV.

```python
sock = set_connection(25001)

while has_started:

    if start_clicked:  # Unless start button clicked in unity program takes initial position of the AGV as an input
        starting_pos = calculate_pos(i)

        i += 1
        print("i", i)

    # time.sleep(0.1)  # sleep 2 sec
    posString = ','.join(map(str, starting_pos))  # Converting Vector3 to a string, example "0,0,0"
    print("Sended data to unity ...", posString)
    sock.sendall(posString.encode("UTF-8"))  # Converting string to Byte, and sending it to C#
    receivedData = sock.recv(1024).decode("UTF-8")  # receiving data in Byte from C#, and converting it to String
    print("Activation key recieved :", receivedData)

    if "1" in receivedData:
        start_clicked = True
        print("starting the User Interface")

    elif "0" in receivedData:
        print("Stop")
        sock.sendall(key.encode("UTF-8"))
        has_started = False
```

Figure 24. Map UI scene connection

```
void SendAndReceiveData()
{
    NetworkStream nwStream = _client.GetStream();
    byte[] buffer = new byte[_client.ReceiveBufferSize];

    //---receiving Data from the Host----
    int bytesRead = nwStream.Read(buffer, offset: 0, _client.ReceiveBufferSize); //Getting data in Bytes from Python
    _dataReceived= Encoding.UTF8.GetString(buffer, index: 0, count: bytesRead); //Converting byte data to string
    Debug.Log( message: "Received data :" + _dataReceived);

    if (_dataReceived.Contains("next"))
    {
        Debug.Log( message: "argument received connection has been closed ...");
        _mThread.Abort();
    }
    if (!String.IsNullOrEmpty(_dataReceived))
    {
        //---Using received data---
        _receivedPos = StringToVector3(_dataReceived); //<-- assigning receivedPos value from Python
        print( message: "received pos data, and moved the AGV!" + _receivedPos);

        //---Sending Data to Host----
        byte[] myWriteBuffer = Encoding.ASCII.GetBytes( s: "Sending data to python  ..." ); //Converting string to byte data
        nwStream.Write(myWriteBuffer, offset: 0, size: myWriteBuffer.Length); //Sending the data in Bytes to Python
    }

}
```

Figure 25. *Encoding data acquired from python*

Figure 26. *Input scene*

```
using UnityEngine;

namespace ScriptableObjects
{
    // fileName is the default name when creating a new Instance
    // menuName is where to find it in the context menu of Create

    [CreateAssetMenu(fileName = "Data", menuName = "Examples/ScriptableObject")]
    public class StartingPos : ScriptableObject
    {
        public CustomDataClass someCustomData = null;
    }

    public class CustomDataClass
    {
        public Vector3 custom;
    }
}
```

Figure 27. *Scriptable object*

```
def set_connection(port):
    # set port 49152 if receiving starting pos, set 25001 if sending calculated pos of the AGV
    # set argument as data which will be sent to unity
    host, port = "127.0.0.1", port
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host, port))
    return sock


has_started = True
start_clicked = False
starting_pos = []
key = "next connection"
i = 0
sock = set_connection(49152)
```

Figure 28. *Input scene connection*
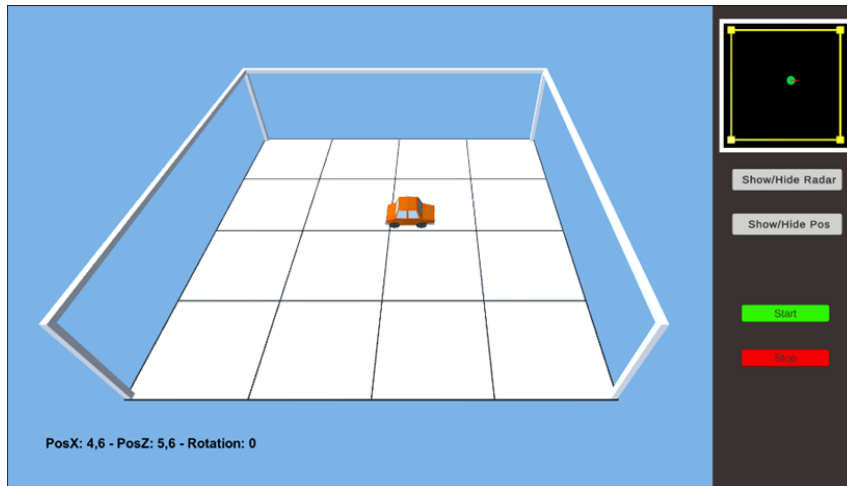
Figure 29. *Map user interface developed by Unity Physics Engine*

Phase 5: Data transfer and integration

Google Drive was used to transfer the necessary documents (Figure 30), videos and similar data during the integration process. In this way, it is ensured that the data is stored regularly and sequentially and not mixed up. In addition, the codes developed throughout the project are stored in the organization we have created on GitHub (Figure 31), and the work done since the beginning of the project has been protected in this way.

Figure 30. *Google Drive*

Figure 31. *GitHub organization*

TESTS

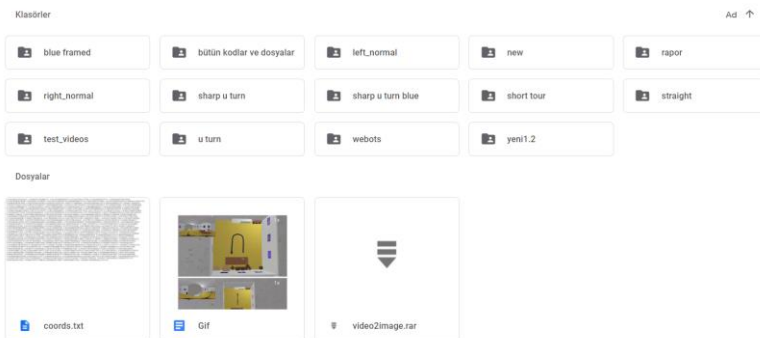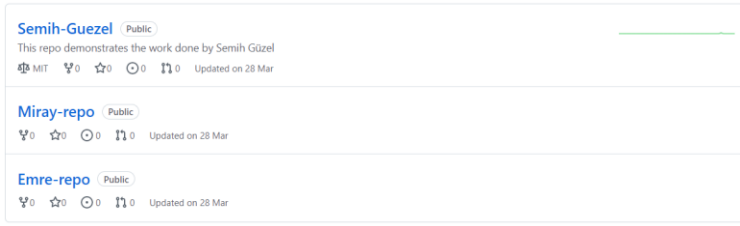In this section, we created 4 scenarios to test our object detection algorithm with the videos obtained from the simulation environment we created. these scenarios are the straight route, right-left l route, and finally the u route, respectively, and the location and orientation of the vehicle are calculated.

Test scenario 1: Straight route in 10x10 m room

      1.1 Taking the camera images:

      A straight route was determined in the 10x10-meter room arranged in the simulation environment, and image acquisition was performed on this route. The captured images were taken as a continuous recording of images to a file as mentioned in the implementation step and then converted to video. The first coordinate (5.02,8.48.4) and the last coordinate (5.05, 4.78,3.99) from which the image is taken are like this. The image acquisition phase is exemplified by a gif (Figure 32).

Figure 32. *Obtaining images from camera and robot's moving path for scenario 1.*

1.2 Results:

As seen in Table 10, the calculated error amount is within the specified 20 cm position accuracy and 2° bearing precision. The reason for the low error rates is due to the fact that the vehicle is moving on a straight route and its rotation changes in very small amounts. However, there are calculations that exceed the error limit due to the shifting of the tracker algorithm in some frames. The maximum error amounts produced in each axis are shown on the figure.

Table 10 Test results for scenario 1

BFGS Method (Coordinates)	Max errors (cm)	Root Mean Square Error for X-axis (cm)	Root Mean Square Error for Y-axis (cm)	Root Mean Square Error for Z-axis (cm)	Total Root Mean Square Error (cm)
X - axis	6.56	2.41	-	-	-
Y - axis	20.35	-	3.14	-	-
Z - axis	4.35	-	-	1.52	-
Total	-	-	-	-	2.45
BFGS Method (Rotation)	Max errors (deg)	Root Mean Square Error for X-axis (deg)	Root Mean Square Error for Y-axis (deg)	Root Mean Square Error for Z-axis(deg)	Total Root Mean Square Error (deg)
X - axis	0.70	0.21	-	-	-
Y - axis	0.21	-	0.09	-	-
Z - axis	0.55	-	-	0.29	-
Total	-	-	-	-	0.21

Test scenario 2: Right turn scenario in 10x10 m room

1.1 Taking the camera images:

A certain route was determined in the 10x10-meter room arranged in the simulation environment, and image acquisition was performed on this route. In order to test whether the locating algorithm can be successful in possible right turns while moving on the route, the route is designed in a sloping way. The captured images were taken as a continuous recording of images to a file as mentioned in the implementation step and then converted to video. The first coordinate (4.45,3.93.4) and the last coordinate (5.24, 8.39,3.99) from which the image is taken are like this. The image acquisition phase is exemplified by a gif (Figure 33).
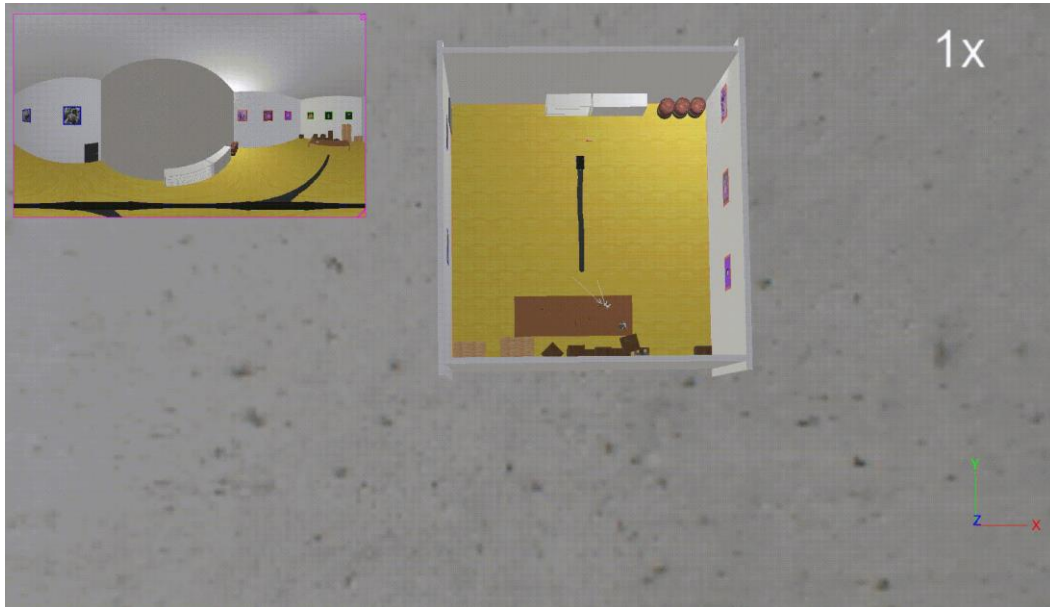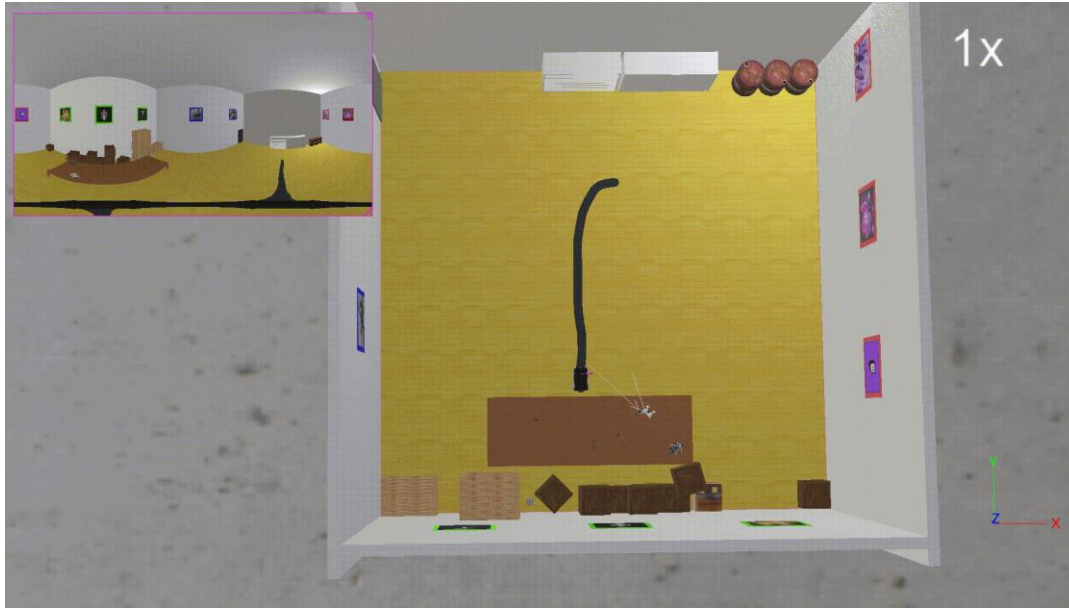
Figure 33. *Obtaining images from camera and robot's moving path for scenario 2.*

1.3 Results:

As seen in Table 11, the calculated error amount is within the specified 20 cm position accuracy and 2° bearing precision. Since the vehicle rotates on the Z axis while drawing L, the highest angle error occurred in the Z axis.

Table 11 *Test results for scenario 2*

BFGS Method (Coordinates)	Max errors (cm)	Root Mean Square Error for X-axis (cm)	Root Mean Square Error for Y-axis (cm)	Root Mean Square Error for Z-axis (cm)	Total Root Mean Square Error (cm)
X - axis	29.81	13.42	-	-	-
Y - axis	9.53	-	3.53	-	-
Z - axis	7.21	-	-	2.29	-
Total	-	-	-	-	8.12
BFGS Method (Rotation)	**Max errors (deg)**	**Root Mean Square Error for X-axis (deg)**	**Root Mean Square Error for Y-axis (deg)**	**Root Mean Square Error for Z-axis(deg)**	**Total Root Mean Square Error (deg)**
X - axis	0.76	0.35	-	-	-
Y - axis	0.95	-	0.36	-	-
Z - axis	2.07	-	-	1.24	-
Total	-	-	-	-	0.77

Test scenario 3: Left turn scenario in 10x10 m room

1.1 Taking the camera images:

In the third test scenario, a room design of 10x10 meters was used. Image acquisition was performed by moving the robot on a certain route. In order to test whether the locating algorithm can be successful in possible left turns while moving on the route, the route is designed with a left slope. The captured images were taken as a continuous recording of images to a file as mentioned in the implementation step and then converted to video. The first coordinate (4.46,3.51.4) and the last coordinate (3.57, 8.16,3.99) from which the image is taken are like this. The image acquisition phase is exemplified with a gif (Figure 34).
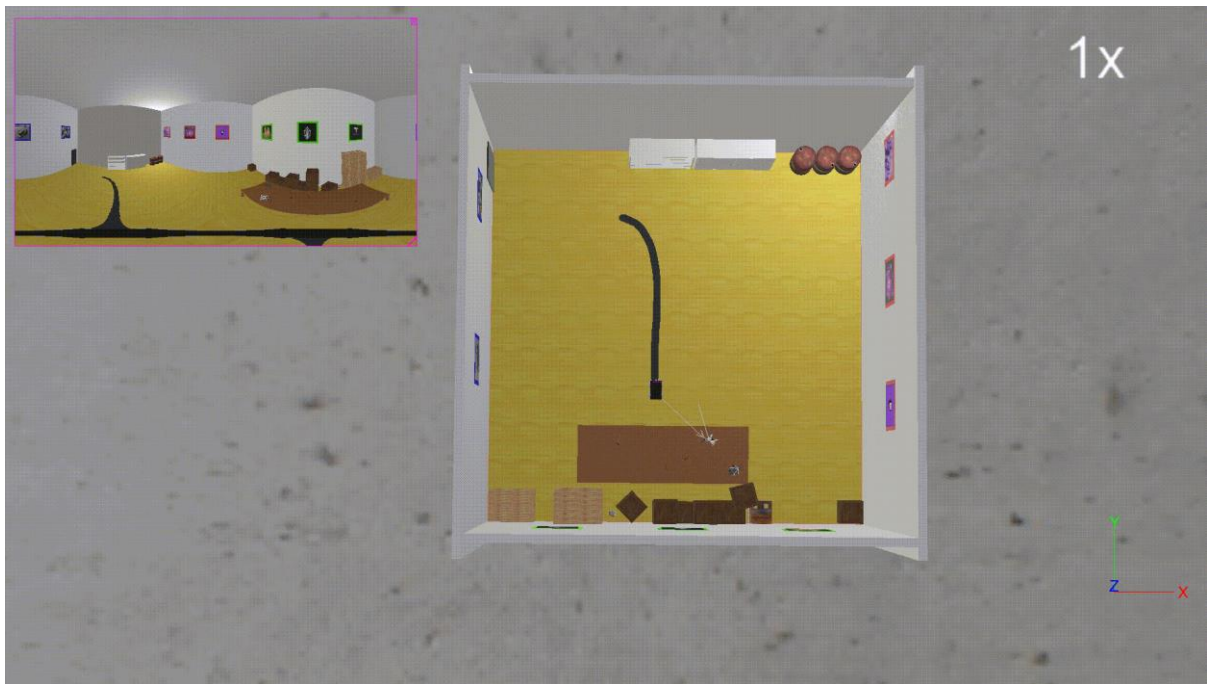

Figure 34. Obtaining images from camera and robot's moving path for scenario 3.

1.2 Results:

As seen in Table 12, the calculated error amount is within the specified 20 cm position accuracy and 2° bearing precision. Since the vehicle rotates on the Z axis while drawing L, the highest angle error occurred in the Z axis.

Table 12 *Test results for scenario 3*

BFGS Method (Coordinates)	Max errors (cm)	Root Mean Square Error for X-axis (cm)	Root Mean Square Error for Y-axis (cm)	Root Mean Square Error for Z-axis (cm)	Total Root Mean Square Error (cm)
X - axis	26.04	8.66	-	-	-
Y - axis	12.15	-	5.43	-	-
Z - axis	8.78	-	-	3.93	-
Total	-	-	-	-	6.32
BFGS Method (Rotation)	**Max errors (deg)**	**Root Mean Square Error for X-axis (deg)**	**Root Mean Square Error for Y-axis (deg)**	**Root Mean Square Error for Z-axis(deg)**	**Total Root Mean Square Error (deg)**
X - axis	0.94	0.39	-	-	-
Y - axis	0.60	-	0.23	-	-
Z - axis	3.78	-	-	1.46	-
Total	-	-	-	-	0.88

Test scenario 4: U route in 10x10 m room

1.1 Taking the camera images:

In the last test scenario, a room design of 10x10 meters was used as in the previous test scenarios. This time, image acquisition was performed by moving the robot on a route with more complex turns. Captured images were taken into a file as a continuous recording of the images as specified in the application step and then converted to video. The first coordinate (3.69, 3.42, 4) and the last coordinate (6.05, 3.58, 3.99) from which the image was taken are like this. The image acquisition phase is illustrated with a gif (Figure 35).

Figure 35. *Obtaining images from camera and robot's moving path for scenario 4.*

1.2 Results:

In the first experiment we made in this scenario, while the markers were passing from the left side of the cube to the back side of the cube, the marker at the far corner appeared as two pieces on the left and back sides of the cube. As a result, the center of the marker was detected incorrectly because the frame size of the marker followed by the tracker was reduced. Since we could not determine the center point correctly, we got rotation errors close to 180 degrees in the Z axis (Table 13).

Table 13 *Test results for scenario 4 first attempt*

BFGS Method (Coordinates)	Max errors (cm)	Root Mean Square Error for X-axis (cm)	Root Mean Square Error for Y-axis (cm)	Root Mean Square Error for Z-axis (cm)	Total Root Mean Square Error (cm)
X - axis	17.86	3.86	-	-	-
Y - axis	14.75	-	5.48	-	-
Z - axis	9.26	-	-	3.02	-
Total	-	-	-	-	4.25
BFGS Method (Rotation)	Max errors (deg)	Root Mean Square Error for X-axis (deg)	Root Mean Square Error for Y-axis (deg)	Root Mean Square Error for Z-axis(deg)	Total Root Mean Square Error (deg)
X - axis	0.73	0.28	-	-	-
Y - axis	0.85	-	0.26	-	-
Z - axis	358.49	-	-	37.02	-
Total	-	-	-	-	21.38

In our second attempt in this scenario, we narrowed the area where we detected the markers, improving the narrowing of the frame that occurred during the transition. However, since it is not a dynamic solution, it is likely to give incorrect results in different test scenarios. In case the AGV makes sharp turns on the route it follows, the number of markers detected is divided into two by going out of the frame. For this reason, a solution is produced by reducing the number of detected markers. Although the algorithm we use can work with at least 4 markers, it has been observed that the errors obtained decrease as the number of markers increases. As a result, the precision of the solution produced is affected depending on the route followed by

the AGV. Since we could not determine the center point correctly, we got rotation errors around Z axis (Table 14).

Table 14 Test results for scenario 4 second attempt

BFGS Method (Coordinates)	Max errors (cm)	Root Mean Square Error for X-axis (cm)	Root Mean Square Error for Y-axis (cm)	Root Mean Square Error for Z-axis (cm)	Total Root Mean Square Error (cm)
X - axis	12.92	4.28	-	-	-
Y - axis	19.25	-	5.17	-	-
Z - axis	8.18	-	-	2.58	-
Total	-	-	-	-	4.15
BFGS Method (Rotation)	**Max errors (deg)**	**Root Mean Square Error for X-axis (deg)**	**Root Mean Square Error for Y-axis (deg)**	**Root Mean Square Error for Z-axis(deg)**	**Total Root Mean Square Error (deg)**
X - axis	0.85	0.32	-	-	-
Y - axis	0.93	-	0.24	-	-
Z - axis	6.05	-	-	1.80	-
Total	-	-	-	-	1.06

Figure 36 shows the error distribution of the position and rotation angles for the X-Y-Z axes.
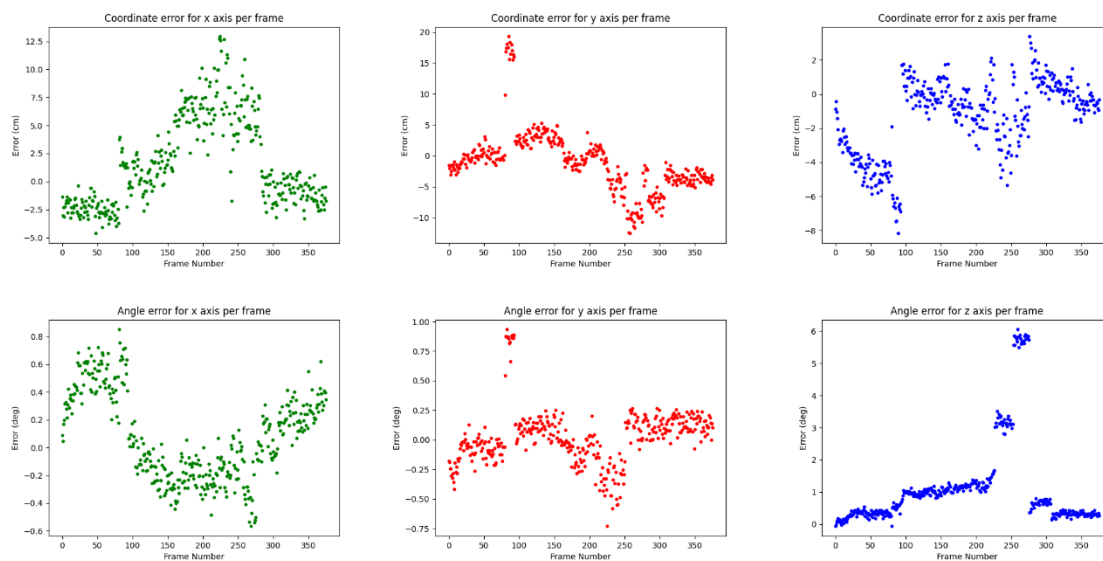

***Figure 36** Errors of X, Y, Z axis both coordinates and rotation for each frame*

Figure 37 shows how the tracker algorithm works while the vehicle is in motion. In addition, the representation of the calculated vehicle coordinates taken from the algorithm on the Map UI can be seen on the Figure 38.
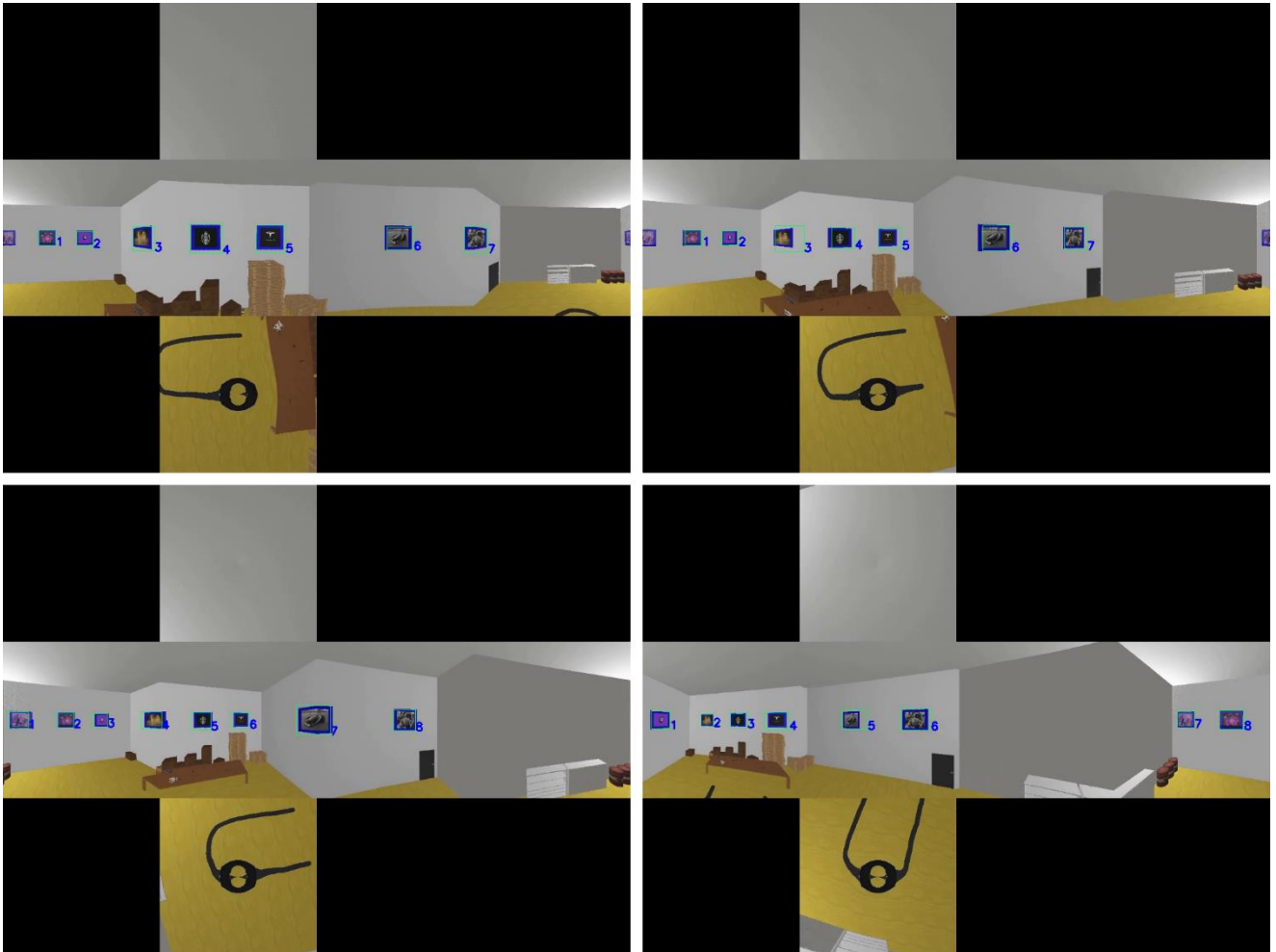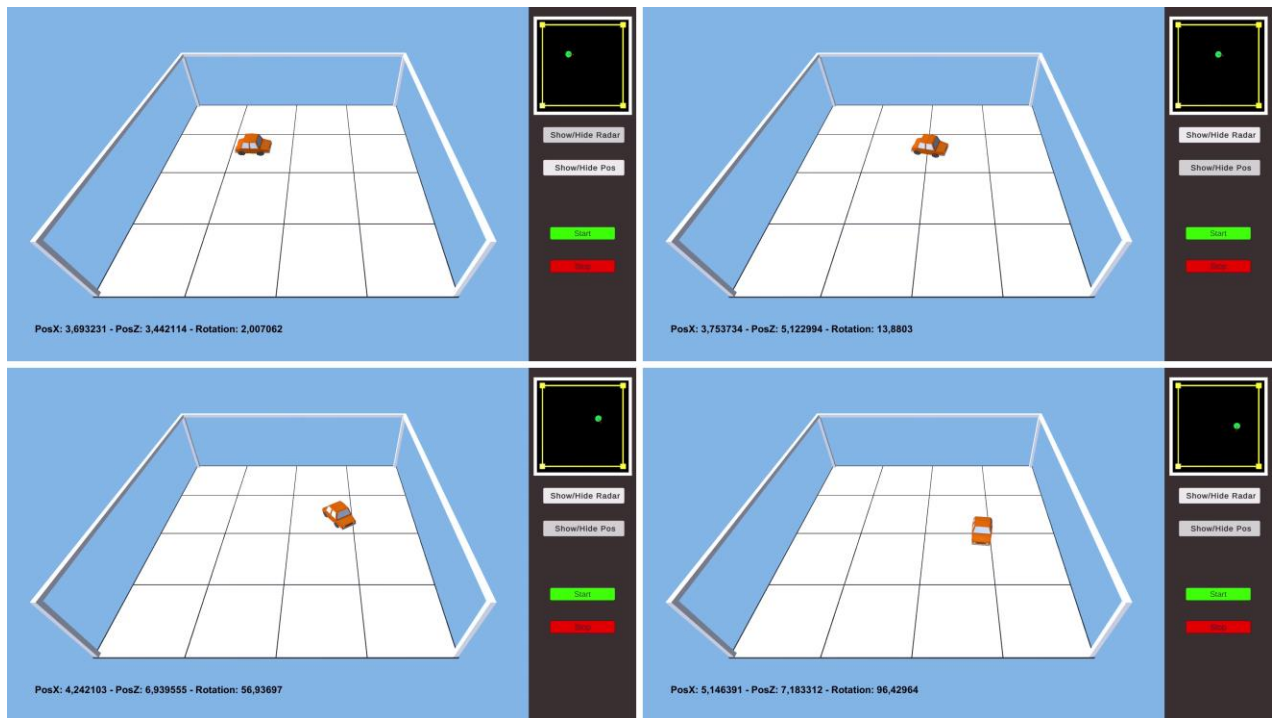

Figure 37. *Tracker algorithm illustration*

Figure 38. *Displaying the vehicles movement on unity*

In addition, the requirement to calculate 2 positions per second was met in all tests. The average number of calculated positions per second was observed as 2.1 - 3 (Figure 39).

Figure 39*. The average number of calculated positions per second*

7 DISSCUSSION

Object detection

There are many alternative methods at the stage of detecting the markers we placed in the room, which is one of the most important parts of our project. These include color and shape detection, machine learning or tracking algorithms.

Color and shape detection

Color and shape detection algorithms work on detecting the attributes of markers. These attributes are the value of each pixel of the object in the HSV color space, and its geometric shape. First of all, we tried to find the pixel coordinates of the center points of the markers using this method in our project. However, although this method is the most easily applicable method, it is vulnerable to external factors. In our tests, it has been observed that this method produces different results according to the lighting conditions in the environment, and the geometry of the detected object varies greatly depending on the light seen in Figure 40. For these reasons, the color and shape detection method were not used because it did not meet the project requirements.

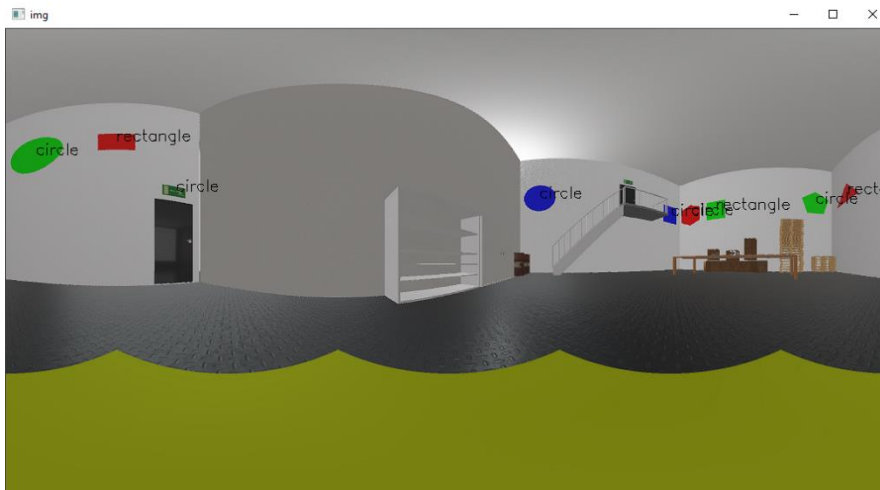***Figure 40***. *Result of the shape and color detection algorithm*

Machine Learning Method (Haar-Cascade)

This method works by applying machine learning principles. In order for the Haar-cascade algorithm to work, many images of the markers to be detected from different angles should be used for model training. Because the actual position of the markers is difficult to find, it will

be difficult to precisely determine the positions of the markers for each different location. In addition, this algorithm has many hyper parameters that need to be corrected such as overfitting and underfitting during the model training process. Due to such reasons, markers may not be detected in every frame, which will impair the accuracy of the calculated coordinates.

Figure 41. *Result of the Haar Cascade Algorithm*

As seen in Figure 41, the Haar-Cascade machine learning algorithm failed to detect some markers in the room and detected some of them incorrectly.

QR Code

QR codes are types of markers in which data can be embedded. For this reason, we aimed to solve the sequencing errors we received in the object detection part of our project by using the QR code method. We produced QR codes containing the sequence of markers that we will place at certain points in the room. However, the QR code method was unsuccessful due to the low resolution of the camera used in the detection phase and the distortions in the equirectangular projection (Figure 42).
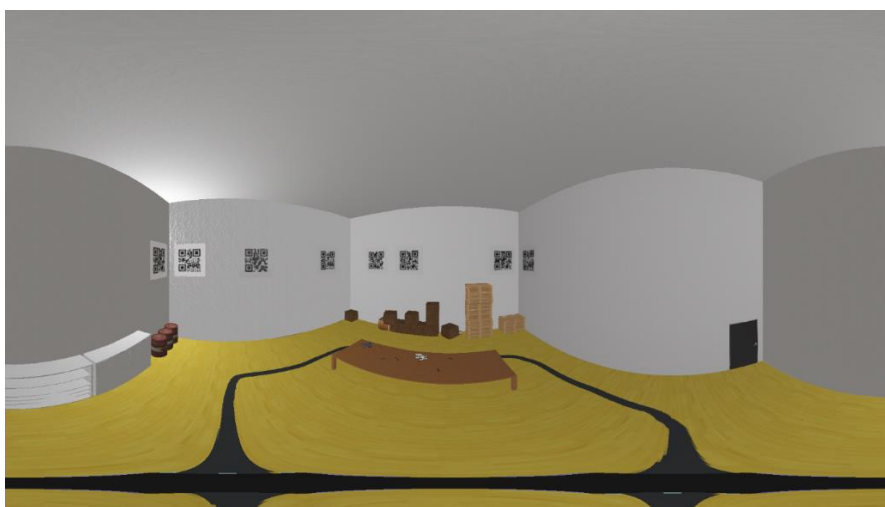

Figure 42. *Representation of how QR code seen in equirectangular projection*

Tracking

In order to calculate the coordinates and rotation of the vehicle in the room, the markers must be continuously detected in each frame of the videos to be used. If a center pixel coordinate cannot be obtained for each marker used in each frame, the calculation of the vehicle's position will fail. In the images taken while the vehicle is in motion, the coordinates of the markers on the image change. Therefore, the tracker algorithm to be used should be able to accurately calculate the change on the image and track the markers in every frame. In this context, it has been decided to use the trackers in the OpenCV library. Six of the tracker algorithms in the OpenCV library were selected and trials were conducted with each of them. These trackers are: Boosting Tracker, MIL Tracker, KCF Tracker, CSRT Tracker, MedianFlow Tracker and Mosse Tracker.

Boosting Tracker: It works similar to the machine learning method, so it is slow and not sensitive enough.

MIL Tracker: The error level is very high and also slow.

KCF Tracker: Although it is quite fast, its accuracy is low.

CSRT Tracker: CSRT works by learning the texture, color, shape, and surroundings of an object that changes over time. For this reason, it was the tracker that gave the most accurate result for this project. However, it is quite slow compared to others.

MedianFlow Tracker: This tracker gave the most accurate result after CSRT. It is quite fast, but if there is a sudden change in the image, it gives wrong results.

As a result, trackers other than the CSRT tracker could not provide 20 cm position accuracy and 2 degrees rotation sensitivity, which are the limitations of the project.

Mathematical Method

First of all, we used the decompose essential matrix function of the OpenCV library in the position and orientation calculation phase. This function requests the camera intrinsic parameters as input. However, since a 360-degree camera is used in the project, these parameters do not need to be used as inputs. Therefore, the OpenCV decompose essential matrix solution is not suitable for the intended use of the project.

Then we tried to implement the essential matrix algorithm on [24] mentioned in the project. However, we observed that the algorithm produced incorrect results in the created test scenarios.

Finally, we obtained approximate location and orientation information of the vehicle by using the BFGS optimization method. The BFGS algorithm minimizes error rate by using the partial second order derivatives of the input values.

Room Size

Since the camera used is omnidirectional, the fact that the markers are too close to each other is a situation that negatively affects the detection algorithm. Considering the dimensions determined for the easy detection of markers in the camera (the camera should be able to detect the marker even from the farthest point) and the number of markers, the reduction in the room size will make it difficult to detect the markers, thus making the algorithm difficult and a result will not be produced with the desired restrictions.

Distortion of Markers Near Poles

In an equirectangular panoramic image, areas near the poles are stretched horizontally. As shown in the Figure 43, the polar regions of an equirectangular image are extremely distorted, making these areas very difficult to retouch. In this case, the markers should not be placed too high on the wall, but it is important not to place them in a low area as objects on the ground may obscure the markers from being seen on the camera. Since the camera and markers are the same size, they will make the markers appear at the equator, thus providing convenience for detection algorithms. This is a suggestion that could solve this problem.

Figure 43. Distorted markers near poles on equirectangular image

Unity-Python Real Time Data Transfer

Real-time display of the location information obtained from the Tracker algorithm on the Map UI reduces the performance. For this reason, the 2-fps required by our project cannot be provided. We solved this problem by saving the calculated coordinates in a txt file. Then reading them with another python script and transferring them to MAP UI with TCP/ICP communication protocol.

Real Life Solution

We demonstrated that our tracker algorithm tested in the simulation works in real-world conditions, with our test in a 5x3 meter room (Figure 44). The size of the marker used in our test is A4 and it will be difficult to detect in a 10x10 meter room, which is one of the limitations of the project. The reasons why we can't produce a result in real life: the appropriate working environment and the necessary materials are not available. In addition, it is the lack of equipment to measure the position of the markers on the world with sufficient precision when calculating the location. Our project has the capacity to produce solutions in real life in case the specified deficiencies are eliminated. Real-time data transfer can be achieved if the test scenarios are performed by a computer with higher processing power.
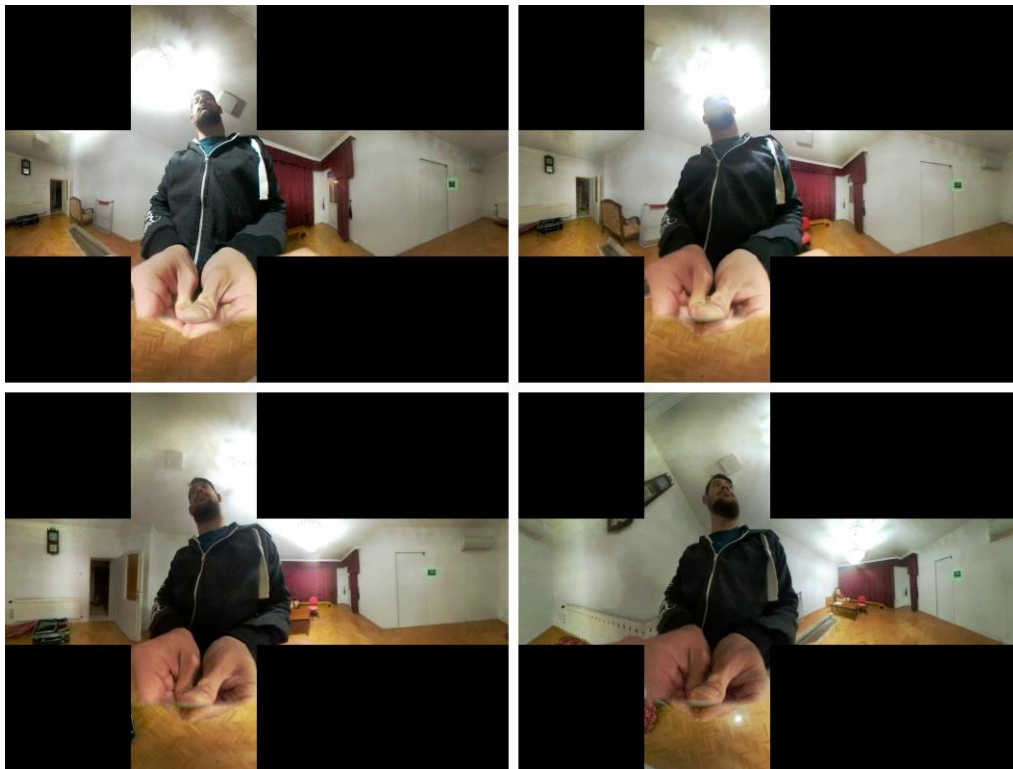

Figure 44. *Real life test*

8 CONCLUSION

In this project, an indoor positioning system using an omnidirectional camera and AoA-based position and orientation estimation was introduced. This system is based on the detection of reference markers from an omnidirectional camera and their use in the AoA based position and direction-finding algorithm. The proof-of-concept application was made in the Webots simulation environment. The results were generated in different test scenarios for the specified number of markers. The results showed that markers can be detected from an omnidirectional camera in an error-free environment (optimal lighting conditions, etc.) and positions can be generated from these markers. The videos obtained with the simulation camera have been previously recorded and given to the algorithm, but it is one piece of information that we can easily say that this algorithm can perform its function in real time with a real camera. The verification of this solution we developed in a simulation environment was successfully carried out, but unfortunately, due to the inadequacies of the environment, it could not be adapted to real life. However, it has been proven by the video given as an example that a detection algorithm can be made with a real camera.

REFERENCES

[1]   H. Liu, H. Darabi, P. Banerjee, ve J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems", *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, c. 37, sy 6, ss. 1067-1080, Kas. 2007, doi: 10.1109/TSMCC.2007.905750.

[2]   M. A. Al-Ammar *vd.*, "Comparative Survey of Indoor Positioning Technologies, Techniques, and Algorithms", içinde *2014 International Conference on Cyberworlds*, Eki. 2014, ss. 245-252. doi: 10.1109/CW.2014.41.

[3]   C. Ma, L. Shi, H. Huang, ve M. Yan, "3D Reconstruction from Full-view Fisheye Camera", Haz. 2015.

[4]   "OpenCV: Camera Calibration and 3D Reconstruction". https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html (erişim 12 Ocak 2022).

[5]   A. Makki, A. Siddig, M. Saad, J. R. Cavallaro, ve C. J. Bleakley, "Indoor Localization Using 802.11 Time Differences of Arrival", *IEEE Trans. Instrum. Meas.*, c. 65, sy 3, ss. 614-623, Mar. 2016, doi: 10.1109/TIM.2015.2506239.

[6]   M. I. M. Ismail *vd.*, "An RSSI-based Wireless Sensor Node Localisation using Trilateration and Multilateration Methods for Outdoor Environment", s. 7.

[7]   D. J. Suroso, F. Y. M. Adiyatma, A. E. Kurniawan, ve P. Cherntanomwong, "Performance Comparison of Several Range-based Techniques for Indoor Localization Based on RSSI", *Int. J. Inf. Commun. Technol. IJoICT*, c. 7, sy 1, Art. sy 1, Haz. 2021, doi: 10.21108/ijoict.v7i1.550.

[8]   M. Phunthawornwong, E. Pengwang, ve R. Silapunt, "Indoor Location Estimation of Wireless Devices Using the Log-Distance Path Loss Model", içinde *TENCON 2018 - 2018 IEEE Region 10 Conference*, Eki. 2018, ss. 0499-0502. doi: 10.1109/TENCON.2018.8650295.

[9]   G. Zanca, F. Zorzi, A. Zanella, ve M. Zorzi, "Experimental comparison of RSSI-based localization algorithms for indoor wireless sensor networks", Nis. 2008, doi: 10.1145/1435473.1435475.

[10]   J. Kunhoth, A. Karkar, S. Al-Maadeed, ve A. Al-Ali, "Indoor positioning and wayfinding systems: a survey", *Hum.-Centric Comput. Inf. Sci.*, c. 10, sy 1, s. 18, Ara. 2020, doi: 10.1186/s13673-020-00222-0.

[11]   T. Kato, M. Nagata, H. Nakashima, ve K. Matsuo, "Localization of Mobile Robots with Omnidirectional Cameras", c. 8, sy 7, s. 4, 2014.

[12]   "RFID Localization Using Angle of Arrival Cluster Forming - Waleed Alsalih, Abdallah Alma'aitah, Wadha Alkhater, 2014".

https://journals.sagepub.com/doi/10.1155/2014/269596 (erişim 05 Ocak 2022).

[13]   M. Schüssel, "Angle of Arrival Estimation using WiFi and Smartphones", *Int. Conf. Indoor Position. Indoor Navig.*, s. 4, 2016.

[14]   S. Rinaldi, P. Ferrari, E. Sisinni, A. Depari, ve A. Flammini, "An Evaluation of Low-Cost Self-Localization Service Exploiting Angle of Arrival for Industrial Cyber-Physical Systems", içinde *2021 IEEE AFRICON*, Eyl. 2021, ss. 1-6. doi: 10.1109/AFRICON51333.2021.9570985.

[15]   F. Alkhawaja, M. Jaradat, ve L. Romdhane, "Techniques of Indoor Positioning Systems (IPS): A Survey", içinde *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, Mar. 2019, ss. 1-8. doi: 10.1109/ICASET.2019.8714291.

[16]   M. Werner, M. Kessel, ve C. Marouane, "Indoor positioning using smartphone camera", içinde *2011 International Conference on Indoor Positioning and Indoor Navigation*, Eyl. 2011, ss. 1-6. doi: 10.1109/IPIN.2011.6071954.

[17]   Y. Zhang, L. Ma, ve X. Tan, "Smart phone camera image localization method for narrow corridors based on epipolar geometry", içinde *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Eyl. 2016, ss. 660-664. doi: 10.1109/IWCMC.2016.7577135.

[18]   K. Guan, L. Ma, X. Tan, ve S. Guo, "Vision-based indoor localization approach based on SURF and landmark", içinde *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Eyl. 2016, ss. 655-659. doi: 10.1109/IWCMC.2016.7577134.

[19]   H. Kawaji, K. Hatada, T. Yamasaki, ve K. Aizawa, "An image-based indoor positioning for digital museum applications", içinde *2010 16th International Conference on Virtual Systems and Multimedia*, Eki. 2010, ss. 105-111. doi: 10.1109/VSMM.2010.5665958.

[20]   A. Morar *vd.*, "A Comprehensive Survey of Indoor Localization Methods Based on Computer Vision", *Sensors*, c. 20, sy 9, Art. sy 9, Oca. 2020, doi: 10.3390/s20092641.

[21]   D.-G. Gwak *vd.*, "Marker-Based Method for Recognition of Camera Position for Mobile Robots", *Sensors*, c. 21, s. 1077, ubat 2021, doi: 10.3390/s21041077.

[22]   H. Feng, Z. Zhu, J. Xiao, ve J. Zhang, "Emerging Techniques in Vision-based Indoor Localization", 2015. https://www.semanticscholar.org/paper/Emerging-Techniques-in-Vision-based-Indoor-Feng-Zhu/050be5a5f98137270375275f7993ad038ad3060b (erişim 05 Ocak 2022).

[23]   L. Payá, A. Gil, ve O. Reinoso, "A State-of-the-Art Review on Mapping and Localization of Mobile Robots Using Omnidirectional Vision Sensors", *J. Sens.*, c. 2017, ss.

1-20, 2017, doi: 10.1155/2017/3497650.

[24]   B. Solarte, C.-H. Wu, K.-W. Lu, Y.-H. Tsai, W.-C. Chiu, ve M. Sun, "Robust 360-8PA: Redesigning The Normalized 8-point Algorithm for 360-FoV Images", içinde *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China, May. 2021, ss. 11032-11038. doi: 10.1109/ICRA48506.2021.9560888.

APPENDICES