Indoor Localization and Tracking Based On Neural Network

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Lu Liang

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2021

Indoor Localization and Tracking Based on Neural Network

By Lu Liang

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Prof. Lei Wang Examination Committee Chairperson	Date
Prof. David Mathias Examination Committee Member	Date
Prof. Mao Zheng Examination Committee Member	Date

Abstract

This manuscript describes a neural network-based indoor localization and tracking method. Most localization technologies rely on GPS signals. However, in an indoor environment, high-frequency GPS signals are blocked by solid objects, such as walls and buildings. In this project, a low-cost, neural network-based vision tracking system that does not rely on GPS is proposed, implemented, and deployed on a smartphone (Nokia 7.2). We use ArUco code to provide reference positions (landmarks) for the system and use neural networks to improve the landmark detection rate. To produce smooth tracking results, we also integrate the inertial measurement unit (IMU)-based tracking and vision-based positioning with the Kalman filter. Through testing and experimental results, we show that the proposed system significantly improves the accuracy, anti-interference ability, and real-time performance of indoor tracking systems.

Acknowledgements

I would like to express my sincere appreciation to my project advisor Dr. Lei Wang for his invaluable guidance and patience when the project is not progressing. Dr. Wang provided the great ideas, and gave me clear guidance when I got trouble. I would also like to express my thanks to the Department of Computer Science at the University of Wisconsin–La Crosse for providing the learning materials and computing environment for my project. Finally, I would like to thank my parents, whose company has gave me great courage to overcome various difficulties in the process of completing this project.

Table of Contents

List of Tables

List of Figures

List of Code

Glossary

Computer Vision

Computer vision is the science of how to make machines "see," and furthermore, it refers to the use of cameras and computers instead of the human eye to identify, track and measure targets and other machine vision, and further do image processing, using the computer processing to become more suitable for human eye observation or transmitted to the instrument to detect the image.

Machine Learning

Machine learning is the study of computer algorithms that improve automatically through experience and by the use of data.

Neural Network

The field of machine learning and cognitive science is a mathematical or computational model that mimics the structure and function of biological neural networks (the central nervous system of animals, especially the brain) to estimate or approximate functions.

ArUco Code

The ArUco code is a reference marker placed on the object or scene being imaged. It is a binary square with a black background and border that is uniquely identified by the white pattern generated within it.

Pespective-n-Point(PnP)

The PnP solving algorithm is an algorithm that solves the external camera reference by minimizing the reprojection error through multiple pairs of 3D and 2D matching points with known or unknown camera internal reference.

Light of Sight

The straight path between a transmitting antenna (as for radio or television signals) and a receiving antenna when unobstructed by the horizon.

Field of View (FOV)

FOV is the open observable area a person can see through his or her eyes or via an optical device.

Inertial measurement unit (IMU)

IMU is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers.

Kalman Filter

The Kalman filter is an efficient recursive filter (autoregressive filter) that estimates the state of a dynamic system from a series of incomplete and noise-inclusive measurements. The Kalman filter considers the joint distribution at each time based on the values of each measurement at different times and then generates an estimate of the unknown variables, which is, therefore, more accurate than an estimate based on only a single measurement.

Accelerometer

An accelerometer is a type of sensor that can measure acceleration. It usually consists of a mass block, a damper, an elastic element, a sensitive element, and a tuning circuit. The sensor obtains the acceleration value by measuring the inertial force on the mass block during acceleration, using Newton's second law.

Gyroscope

The gyroscope, a device used to sense and maintain direction, was designed based on the theory of angular momentum indestructibility. Once the gyroscope starts to rotate due to the angular momentum of the wheel, the gyroscope tends to resist the change of direction. People use it to maintain the direction according to this theory, and the manufactured thing is called a gyroscope.

Magnetometer

Magnetometer devices convert changes in the magnetic energy of sensitive elements caused by magnetic fields turn into electrical signals and detect the corresponding physical quantities in this way.

1. Introduction

1.1. Overview

The Global Positioning System (GPS) has been used for positioning and navigation since its invention. The GPS is a navigation system that uses satellites, receivers, and algorithms to synchronize position, speed, and time data for air, sea, and land travel. Satellites orbiting the earth send signals to be read and interpreted by a GPS device on or near the earth's surface. GPS is a powerful and reliable tool for businesses and organizations in many diverse industries. Some people who use GPS daily are scientists, pilots, ship captains, and first responders. Such individuals tend to use GPS information to prepare accurate surveys and maps, make precise time measurements, track locations or positions, and navigate [1] [2]. Although GPS can provide people with convenient navigation and positioning services, it has limitation in the indoor environments. GPS signals propagate through waves at frequencies that do not readily move through solid objects. At the same time, GPS devices rely on a series of satellites to determine their physical location. When users are using GPS inside a building, various physical obstacles and potential sources of interference make it difficult for the device to determine its location accurately. GPS works better when the device has a clear Line-of-Sight (LOS) to the sky. The more GPS satellites a user' device can access, the more accurate the result will be. When indoors, there is usually no direct line between the satellite signal and the device. The signal will weaken or distort as it travels through the building to reach the device, and will operate incorrectly. Consequently, this situation creates a development gap between indoor and outdoor localization systems.

With the popularity of intelligent terminals, indoor positioning technologies such as Wi-Fi, Bluetooth, Radio frequency identification technology (RFID), and Ultra Wide Wave Technology (UWB) have emerged. Wi-Fi positioning system (WPS) uses the characteristics of nearby Wi-Fi hotspots and other wireless access points to discover where a device is located [3] [4]. The disadvantage of Wi-Fi-based indoor positioning technology is that due to the complex structure of indoor space, there is also a shadowing effect and multi-path propagation effect during the propagation of radio waves in indoor space. It is challenging to model radio-frequency signal propagation in an indoor environment. Our proposed method remains effective in complex indoor environments because we need not receive signals emitted from a specific source. Bluetooth was concerned with distance rather than exact location. Originally, Bluetooth was not intended to provide a fixed location like GPS. However, it was called a geofencing or micro-fencing solution, making it an indoor proximity solution rather than an indoor location solution [5]. The disadvantage is that a large number of beacon devices need to be deployed indoors to achieve 1-2 meters of positioning accuracy. This will increase costs, while our proposed method is very economical as it does not require additional equipment assistance [6].

The existing indoor localization and tracking systems require either the priori knowledge of the environment, such as the building plan, the locations of WiFi access points, Bluetooth beacons, and prebuilt RF fingerprints database, or expensive onboard equipment, such as 3D lidar, depth camera, or omnidirectional camera[7]. Google indoor maps [8], for example, can triangulate user's approximate location (the position where user are standing) in an indoor shopping mall using the nearby WiFi spots, user device Bluetooth, and user device built-in GPS. But it is only available in public buildings such as airports, malls, stadiums, and train stations, as these are shared spaces, and their infrastructure information is publicly available. However, in practices, indoor positioning is needed not only in public buildings but also in ordinary buildings. For example, a robot delivers food in a residential building. To navigate the robot through gate, access elevator, and avoiding obstacles, precise indoor positioning is necessary. It is challenging to obtain infrastructure information due to security considerations in residential buildings or even some of the commercial buildings. In contrast, our proposed method does not require infrastructure environmental information and can me applied widely in most indoor localization scenarios.

For existing computer vision technology, the method of localization using only vision has a large amount of image processing leading to poor real-time performance [9][10]. Moreover, such methods are affected by light and cannot work in poor light conditions. We found that the traditional vision localization algorithm has poor anti-interference ability and limited real-time problems. We proposed a deep learning-based vision localization method and thus designed an indoor localization system based on neural networks and sensor fusion. We improved the accuracy of vision localization by using neural network-based object detection and made the system highly efficient by feeding results in real-time.

The vision-based indoor localization is used to extract the information about the three-dimensional (3D) world from the two-dimensional (2D) images captured by the camera [11]. Discovering the correspondence between 3D points in the real-world environment and their 2D image projections is the most critical and complex step in the process. In our project, we used the ArUco code to aid localization. The advantage of this marker is that a single marker provides enough correspondence information to calculate the camera's pose. Besides, the internal binary encoding of the ArUco code allows the marker to maintain specific stability in terms of checks and corrections. We used neural networks to improve the accuracy of detecting markers under complex conditions such as fast movement and light changes.

In detecting markers, we may encounter situations where we cannot capture the markers due to the limited view angle of the camera. The sensors that come with mobile phones can then be used as an alternative localization method when the markers are not detected. In IMU-based localization, we need to double integrate the measured acceleration values. When we integrate the accelerations, we also integrate the noise, which causes a drift in the calculated trajectory. A certain distance exists between each marker in our experimental setting, creating a gap in detecting two markers. However, the acquisition frequency of the cell phone sensor is very high, and it can be used to fill this gap. As mentioned earlier, just using the sensor for localization can lead to an accumulation of errors. Therefore, we also introduce the Kalman filter to correct this error. The phone's position obtained by vision localization is an absolute position that can be a measurement value to update the position of the phone obtained by using sensor-based localization. Updating the prediction is the process of the Kalman filter to reduce accumulation errors.

The whole process of localization is first to detect the marker that appears in each frame of the video, and the resulting information is the ID of the marker and the 2D coordinates of each of its corners. Then, using this information and the 3D coordinates of the corners measured, the 3D coordinates of the camera are obtained by solving the PnP (Perspective-n-Point) problem. At the same time, the sensors are working. The fusion of the accelerometer and gyroscope also allows obtaining the current position. Consequently, this result will be used to fill the gap where no marker is detected. Finally, based on the calculated 3D coordinates of the device, a 3D trajectory of the device's movement can be plotted.

1.2. ArUco Code

The ArUco marker was originally developed in 2014 by S. Garrido-Jurado et al [12]. Figure 1 shows the example of ArUco marker. Pose estimation is fundamental in many computer vision applications: robot navigation, augmented reality, and so on. The process is based on discovering the correspondence between 3D points in the natural environment and their 2D image projections. This is often difficult, so synthetic or benchmark markers are often used to simplify the operation. The use of ArUco markers is usually one of the common ways. The main advantage of this marker is that a single marker provides enough correspondence information to obtain the camera pose. Simultaneously, the internal binary encoding of the notation allows the marker to maintain some stability in terms of error checking and correction. The black borders speed up the detection of the marker in the image, and the internal 2D encoding uniquely identifies the marker for both error detection and error repair. The size of the marker determines the size of the internal matrix. We chose the ArUco code to aid the visual positioning of our system over the more common QR codes for reasons that can be derived from the comparison below.

Type / Functionality	**ArUco code**	**QR code**
Information storage	Only simple id numeric information can be stored	Can store any custom information
Detection difficulty	Detection is simple, fast, and highly robust	Easy to lose at long distances and large declination angles

Table 1. Comparison between ArUco code and QR code

In indoor positioning, we do not need to store too much information on the marker but only have an ID that can distinguish the marker, and the ArUco code satisfies this condition well. Because it does not store much information, it is less computationally intensive during detection, which helps to improve the real-time performance of the system.

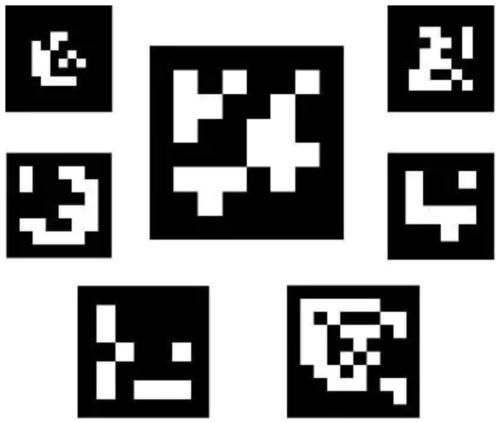


Figure 1. The example of ArUco markers

1.3. YOLO

A neural network model for detecting ArUco marker is a object detection model. Object detection is the most common problem in machine vision. It is a kind of image segmentation based on the object's geometric and statistical features, which combines the segmentation and recognition of the object into one, and its accuracy and real-time is an essential capability of the whole system. In recent years, object detection has been widely used in artificial intelligence, face recognition, unmanned vehicles, and other fields.There exist many kinds of neural networks for object detection. Among them, the convolutional neural network (CNN) has become a research hotspot in the field of image recognition, and its weight-sharing network structure makes it more similar to the biological neural network, reducing the complexity of the network model and the number of weights. Its advantages are more apparent when the input of the network is a multidimensional image, which enables the image to be directly used as the input of the network, avoiding the complex feature extraction and data reconstruction processes in traditional recognition algorithms. The CNN is a multilayer perceptron designed to recognize two-dimensional shapes. This network structure is highly invariant to translation, scaling, tilting, or other forms of deformation.[13] YOLO is a object detection algorithm that uses a separate CNN model.

The YOLO was chosen as the object detection model of this project. The full name of YOLO is you only look once, referring to the fact that you only need to browse once to identify the category and location of an object in an image. Because you only need to look once, YOLO is known as a Region-free method, as opposed to a Region-based method, which does not need to find the region where the object may exist in advance. That is, a typical region-based method flows like this: first, the image is analyzed by computer graphics (or deep learning) to find a number of regions where objects may exist, and these regions are cropped and placed into an image classifier, which classifies them. Because a region-free method like YOLO requires only one scan, it is also known as a single-stage model.[14] In this project, the YOLO model takes real-time frames as input and then outputs the detected ArUco markers' location, including the coordinates of the four corners of the markers.

In general, single-stage detectors tend to be less accurate than tow-stage detectors but are significantly faster. Suitable for real-time system. The objective of this project is to implement a real-time indoor positioning and tracking system, so the requirement for detection speed is very high. Therefore, YOLO algorithm is the best choice for this project. In this project, we specifically use YOLO version 3 (YOLOV3) [15]. The backbone network used in YOLOV3 is Darknet-53, which has more network layers, introduces a more advanced residual network, and replaces max pooling layers with convolutional layers. This will result in greater accuracy. Section 4. will describe how YOLOV3 is applied to this project.

2. Software Development Life Cycle Model

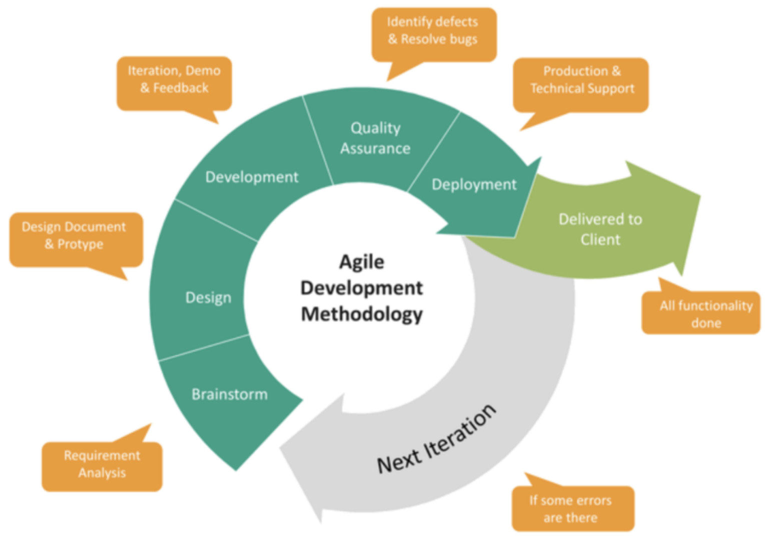
2.1. Overview

The software life cycle model refers to the typical practices of the software life cycle that people have summarized to develop better software. The software life cycle model's primary role is to determine the sequence of implementation of individual software development activities. An appropriate software life cycle model should be selected in the early stages of software development to achieve the goal of simplifying the software development process and reducing the difficulty of software development. The software life cycle model can help developers improve software development efficiency and software quality, reduce software development costs, monitor and control the software development process, and reduce risk. Therefore, it is essential to choose the appropriate software life cycle model according to the specific situation of the software development project[16]. The agile development model is used in this project.

2.2. Agile Development Model

Agile development is a human-centered, iterative, step-by-step development approach. In agile development, the build of a software project is sliced into multiple sub-projects, and the results of each sub-project are tested and have integrated and runnable characteristics. In other words, a large project is divided into multiple interconnected but also independently runnable sub-projects and completed separately, with the software continuously in a usable state during the process.[17]

We chose the agile development model because we can easily cope with the uncertainty of changing requirements during the development process. Another reason is that with agile development, we have a clear picture of the progress of the project. In terms of how we implemented agile development, we divided the whole project into four parts and completed them one by one. They are vision-based indoor localization, neural network-based indoor localization, sensor fusion-based indoor localization, and a combination of the three. In addition to that, Dr. Lei Wang and I have a weekly meeting that allows me to keep him informed of progress. Based on his feedback, I will adjust the development method for the next week to achieve our project goals.



Source: https://orionadvisortech.com/blog/why-were-moving-to-agile-software-development/?print=print

Figure 2. Agile Development Process

3. Requirements

This project aims to implement a neural network-based indoor visual localization system that can be instantiated and implemented on an mobile phone. When visual positioning is not available, i.e., when no marker is detected, this system can also call on the sensors that come with the Android phone for indoor positioning. Eventually, the device's movement trajectory can be drawn in a graph with a 3D coordinate system. We evaluate the performance of our proposed system under different environmental conditions. For example, the light conditions (sufficient light and insufficient light) and movement speed.

3.1. Functional Requirements

The functional requirements of this project can be summarized by the following four tasks:

- This system can calibrate camera.
- This system can detect ArUco markers with neural network.
- This system can use the detected markers to localization mobile device, which is vision-based localization.
- This system can use inertial sensors to aid vision-based localization.

This project focuses on collecting datasets for training neural networks, training the network, and implementing it on Android phones. Therefore, it is not the same as a typical user-based application with no specific user or user operation. Instead, the main work of this project is to turn the theory of neural network-based indoor visual localization into a working Android application.

3.2. Non-functional Requirements

We also have non-functional requirements to judging the operations of the system.

1. Reliability.
   Reliability is the extent to which the software system consistently performs the specified functions without failure. Our system needs to ensure that we can still use the IMU for localization if the marker is not detected rather than crashing.
2. Effectiveness.
   Effectiveness indicates a system's resulting performance in relation to effort. Since our system needs to detect markers and output the results in real-time, effectiveness is an essential requirement for our system.

4. Design

4.1. Overview

The goal of this project is to locate and track a mobile device and to plot the 3D trajectory of mobile device. The input to this system is the video frame captured by the mobile device, measured values of inertial sensor, gyroscope and magnetometer, and the output is the 3D movement trajectory of it. To accomplish the goals of this project, we have divided the system into five modules. The entire process of this system exhibits in Figure 3. The whole system is divided into five modules. The first module is the camera calibration. The results obtained from this module are the intrinsic and extrinsic parameters of the camera. This result will be used as input for the second module. The second module is to use YOLO to detect the markers. The detection results are the 2D coordinates of the four corners of the marker and its unique ID. When the system does not detect the marker, it goes to the third module, which is IMU-based localization. This module takes the values obtained from the IMU and fuses them to calculate the displacement of the device. When the system detects the marker, it goes to the fourth module, which is to use the PnP algorithm to estimate the device pose. Also, the results obtained in this step are used as input to the Kalman filter in the third module, which is used to update the results of the sensor fusion calculation. The main task of the last module is to present the results of the previous calculations and plot a 3D trajectory diagram.

4.2. Camera Calibration

The first module is the camera calibration. In machine vision applications, a geometric model of camera imaging is required to determine the 3D geometric position of a point on the surface of a spatial object and its corresponding point in the image. These geometric model parameters are the camera parameters. In most conditions, these parameters must be obtained through experiments and calculations, and this process of solving the parameters (intrinsic, extrinsic and distortion parameters) is called camera calibration [18].

The first step in camera calibration requires converting the world coordinate system to the camera coordinate system. The world coordinate system (X_W, Y_W, Z_W), also known as the measurement coordinate system, is a three-dimensional right-angle coordinate system to which the spatial position of the camera and the object to be measured can be described. The position of the world coordinate system can be freely determined according to the actual situation. The camera coordinate system (X_C, Y_C, Z_C), also a three-dimensional right-angle coordinate system, the origin is located at the optical center of the lens, x and y axes are parallel to the two sides of the phase plane, z-axis for the lens optical axis, and perpendicular

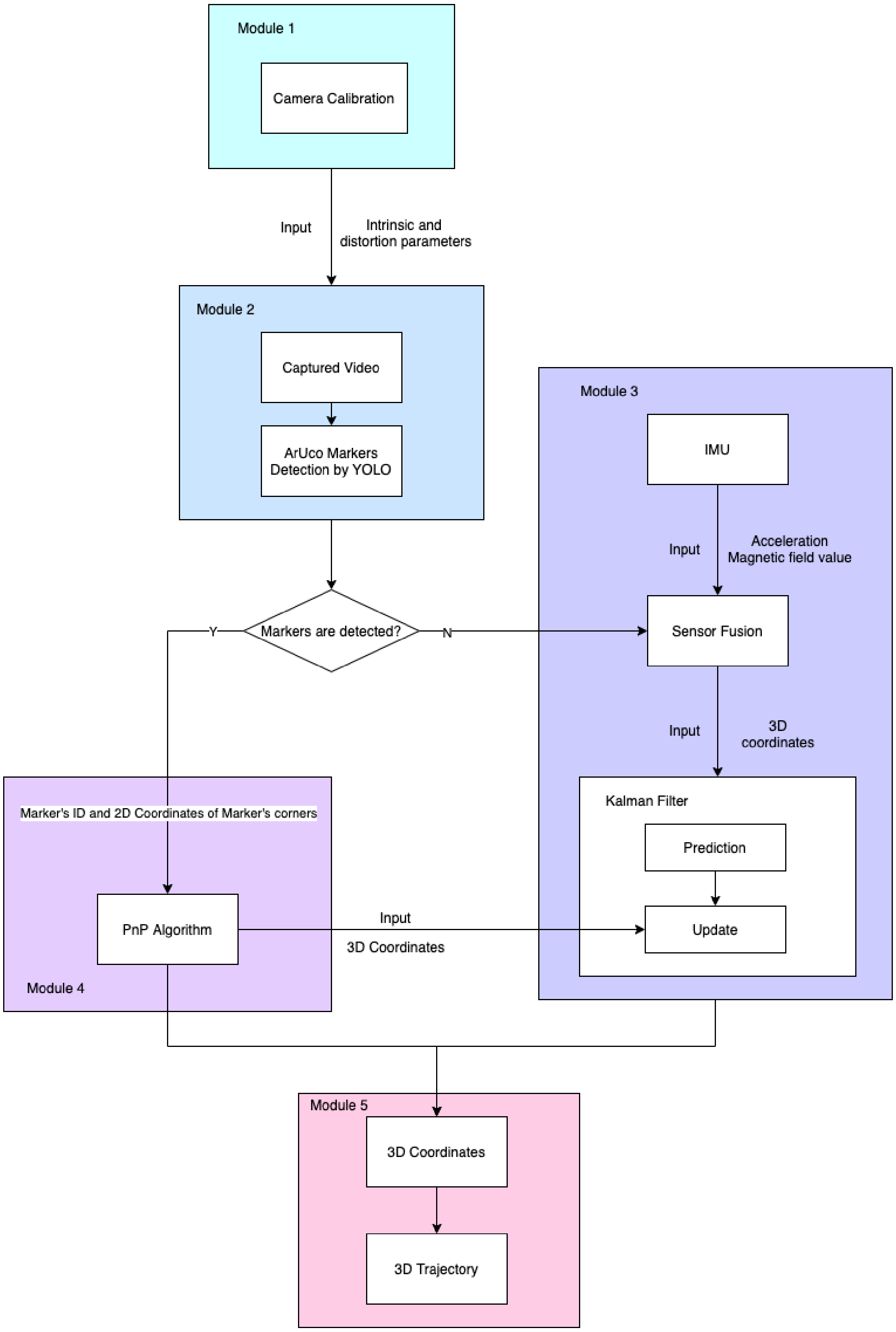


Figure 3. The entire process of system

to the image plane. The conversion process is shown in Equation (1).

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{1}$$

where $\mathbf{R}$ is a 3×3 rotation matrix, $\mathbf{t}$ is a 3×1 translation vector, $(X_C, Y_C, Z_C)^T$ and $(X_W, Y_W, Z_W)^T$ are the homogeneous coordinates of camera coordinate system and world coordinate system, respectively.

The next step is the conversion of pixel coordinates and image coordinates. As shown in Figure 4, the pixel coordinate system uov is a two-dimensional right-angle coordinate system that reflects pixel arrangement in the camera chip. The origin $\mathbf{o}$ is located in the upper left corner of the image, and the u and v axes are parallel to the two sides of the image plane, respectively. The units of the axes in the pixel coordinate system are pixels. The pixel coordinate system is not conducive to coordinate transformation, so it is necessary to establish the image coordinate system XOY. The unit of its coordinate axis is usually millimeters (mm). The origin is the intersection of the camera optical axis and the phase plane (called the principal point), which is the center of the image. X-axis and Y-axis are parallel to the u-axis and v-axis, respectively. Therefore, the two coordinate systems are translational, i.e., they can be obtained by translation. This conversion can be done by Equation (2).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/dX & 0 & u_0 \\ 0 & 1/dY & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2}$$

where, dX, dY are physical dimensions of the pixel in the X and Y-axis directions, respectively. u_0 and v_0 are the coordinates of the principal point.

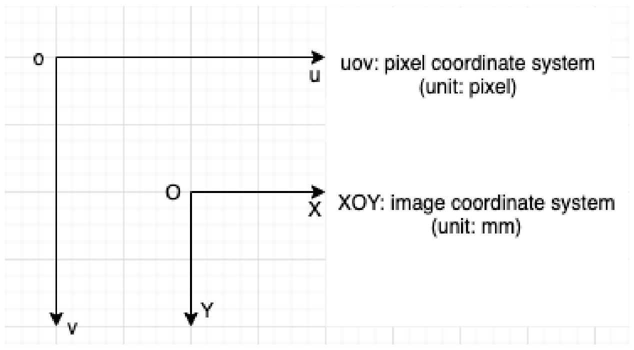


Figure 4. Pixel coordinate and image coordinate

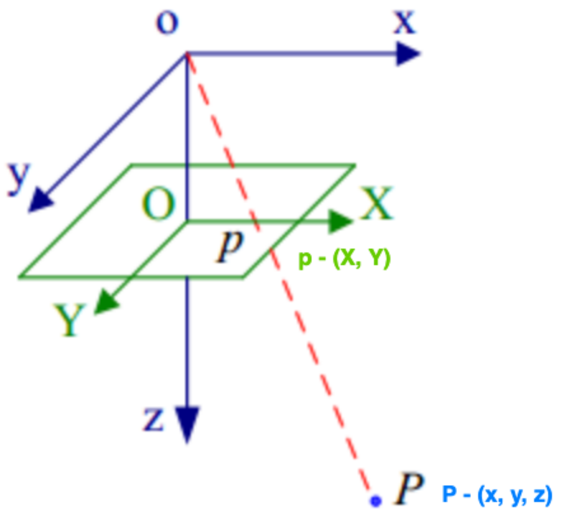


Figure 5. Pinhole imaging principle

Figure 5 illustrates the relationship between any point **P** in space and its image point **p**. The line between **P** and the camera optical center **o** is **oP**, and the intersection of **oP** and the image plane **p** is the projection of the point **P** in space on the image plane. This process is perspective projection, as represented by the following matrix:

$$s \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3}$$

where, s is the scale factor (s is not zero), f is the effective focal length (the distance from the optical center to the image plane). $(x, y, z, 1)^T$ is the homogeneous coordinates of the spatial point **P** in camera coordinate system xoy, and $(X, Y, 1)^T$ is the homogeneous coordinates of the image point **p** in the image coordinate system XOY. Combining Equations (1) to (3) we can get the intrinsic and extrinsic parameters of camera.

$$\begin{aligned} s\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} &= \begin{bmatrix} 1/dX & 0 & u_0 \\ 0 & 1/dY & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} a_x & 0 & u_0 & 0 \\ 0 & a_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} = \boldsymbol{M}_1\boldsymbol{M}_2\boldsymbol{X}_w \end{aligned} \tag{4}$$

where, $a_x = f/dX, a_y = f/dY$, are called the scale factors of u and v axes. M_1 and M_2 are the intrinsic and extrinsic parameters of camera, respectively.

4.3. YOLO

The second module uses YOLO to detect markers. As we mentioned earlier, YOLO is a single-stage detector, which is fast and ideal for applications in real-time systems. The following will describe how YOLOV3 is applied to this project.

When a frame is passed into YOLOV3, this image is first resized to 416x416 grids, and a gray bar is added around the image to precent distortion. YOLOV3 then splits the images into 13x13, 26x26, and 52x52 grids, which are used to detect large, medium, and small objects, respectively. Each grid point is responsible for detecting its lower right corner area, and if the object's center point falls in a grid, then the object's position will be determined by that grid point. In the example given in Figure 6, an image containing the object to be detected, i.e., the ArUco marker, is input into the YOLOV3 neural network and then surrounded by gray bars. In this image, the ArUco belongs to a large object, so that a 13×13 grid image will detect the result.

Since there is no open-source ArUco dataset, we need to collect, process, and label the dataset ourselves. For the training part of YOLO, we choose to train on Google Colab. The implementation of these tasks will be described in detail in Section 5..

4.4. Camera Pose Estimation

The two-dimensional coordinates obtained from the detection of the markers will be used for the estimation of the camera pose. The camera pose estimation is mainly based on the PnP (Perspective-n-point) algorithm. General conditions for the PnP problem [19]:

- Coordinates of the n 3D reference points in the world coordinate system.

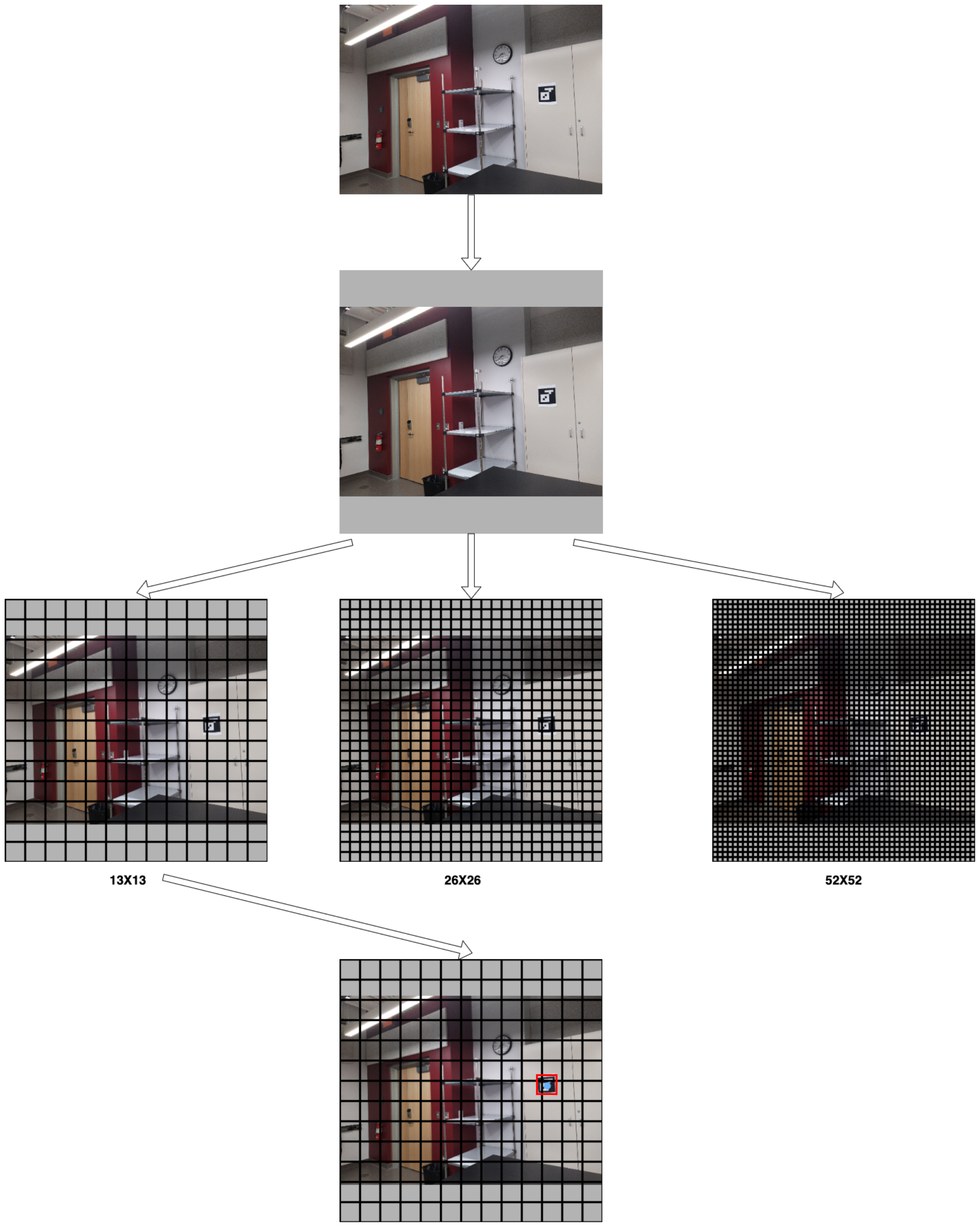


Figure 6. The training process of YOLOV3

- Corresponding to these n 3D points, the coordinates of the 2D reference point projected on the image.
- The intrinsic parameters of the Camera, denoted by $\boldsymbol{M}_1$.

Based on our experimental results (for details, see Section 6.), the EPnP algorithm is of the highest accuracy among the existing PnP algorithms.

Most non-iterative PnP algorithms will first solve for the depth of the feature point to obtain the 3D coordinates of it in the camera coordinate system. The EPnP algorithm, on the other hand, represents the 3D coordinates in the world coordinate system as a weighted sum of a set of virtual control points. For the general case, the EPnP algorithm requires the number of control points to be four, and these four control points cannot be coplanar. Because the camera's extrinsic parameters are unknown, the coordinates of these four control points under the camera reference coordinate system are unknown. Furthermore, if we can solve the coordinates of these four control points under the camera reference coordinate system, we can calculate the camera's pose [20].

4.4.1. Control Points and Barycentric Coordinates

In this paper, the superscript ^{w} and ^{c} are used to denote coordinates in the world and camera coordinate systems, respectively. Then, the coordinates of the 3D reference points in the real world frame are $\boldsymbol{p}_i^w, i = 1, ..., n$, the coordinates in the camera frame are $\boldsymbol{p}_i^c, i = 1, ..., n$. The four control points in the world coordinate system are $\boldsymbol{c}_j^w, j = 1, ..., 4$, the coordinates in the camera reference coordinate system are $\boldsymbol{c}_j^c, j = 1, ..., 4$.

The EPnP algorithm expresses the coordinates of the reference point as a weighted sum of the coordinates of the control point:

$$\boldsymbol{p}_i^w = \sum_{j=1}^{4} a_{ij}\boldsymbol{c}_j^w, with \sum_{j=1}^{4} a_{ij} = 1 \tag{5}$$

where the a_{ij} are homogeneous barycentric coordinates. They are unique and can easily be estimated. In the camera coordinate system, the same relationship exists:

$$\boldsymbol{p}_i^c = \sum_{j=1}^{4} a_{ij}\boldsymbol{c}_j^c \tag{6}$$

Assuming that the extrinsic parameters (rotation matrix R and translation vector t) of the camera are $[\boldsymbol{R}, \boldsymbol{t}]$ then a relationship exists between the virtual control points $\boldsymbol{c}_j^w$ and $\boldsymbol{c}_j^c$:

$$\boldsymbol{c}_j^c = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_j^w \\ 1 \end{bmatrix} \tag{7}$$

Considering that the EPnP algorithm expresses the reference point coordinates as a weighted sum of the control point coordinates, then can get:

$$\boldsymbol{p}_i^c = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} p_i^w \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} \sum_{j=1}^4 a_{ij} c_j^w \\ 1 \end{bmatrix} \tag{8}$$

Further,

$$\boldsymbol{p}_i^c = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} \sum_{j=1}^4 a_{ij} \boldsymbol{c}_j^w \\ 1 \end{bmatrix} = \sum_{j=1}^4 a_{ij} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \begin{bmatrix} c_j^w \\ 1 \end{bmatrix} = \sum_{j=1}^4 a_{ij} \boldsymbol{c}_j^c \tag{9}$$

In the above derivation, the important constrain $\sum_{j=1}^4 a_{ij} = 1$ of EPnP on the weight a_{ij} is used. Without this constraint, the above derivation will not hold. Putting the four control point constraints together yields the following equation:

$$\begin{bmatrix} \boldsymbol{p}_i^w \\ 1 \end{bmatrix} = \boldsymbol{C} \begin{bmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \\ a_{i4} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1^w & \boldsymbol{c}_2^w & \boldsymbol{c}_3^w & \boldsymbol{c}_4^w \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \\ a_{i4} \end{bmatrix} \tag{10}$$

Obviously, $[\boldsymbol{p}_i^{W^T} 1]^T$ and $[\boldsymbol{c}_j^{W^T} 1]^T$ are both isometric coordinates. The equation (5) in the reference paper, however, is essentially a linear combination of the isometric coordinates of 3D reference points with the isometric coordinates of the control points. Thus, we also get barycentric coordinates computed as follows:

$$\begin{bmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \\ a_{i4} \end{bmatrix} = \boldsymbol{C}^{-1} \begin{bmatrix} p_i^w \\ 1 \end{bmatrix} \tag{11}$$

4.4.2. Selection of Control Points

A specific method for determining the control points is given in here. The set of 3D reference pints is $P_i^W, i = 1, ..., n$ and the barycentric coordinates of the 3D reference points is chosen as the first control point:

$$\boldsymbol{c}_1^w = \frac{1}{n} \sum_{i=1}^n \boldsymbol{p}_i^w \tag{12}$$

This leads to the matrix:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{p}_1^{w^T} - \boldsymbol{c}_1^{w^T} \\ \dots \\ \boldsymbol{p}_n^{w^T} - \boldsymbol{c}_1^{w^T} \end{bmatrix} \tag{13}$$

Donating the characteristic value of $\boldsymbol{A}^T\boldsymbol{A}$ as $\lambda_{C,i,i=1,2,3}$, the corresponding feature vector is $v_{c,i,i=1,2,3}$. Thus, the remaining three control points can then be determined by the following formula:

$$\boldsymbol{c}_j^w = \boldsymbol{c}_1^w + \lambda_{c,j-1}^{\frac{1}{2}} v_{c,j-1}, j = 2, 3, 4 \tag{14}$$

4.4.3. Solve for the coordinates of the control point in camera coordinates

$\boldsymbol{u}_i, i = 1, ..., n$ is the 2D projection of the reference point $\boldsymbol{p}_i, i = 1, ..., n$, then,

$$\forall i, w_i \begin{bmatrix} \boldsymbol{u}_i \\ 1 \end{bmatrix} = K\boldsymbol{p}_i^C = K\sum_{j=1}^{4} a_{ij}\boldsymbol{c}_j^C \tag{15}$$

Substitute $\boldsymbol{c}_j^C = [x_j^C, y_j^C, z_j^C]^T$ into the above equation and write K in the form of focal length f_u, f_v and optical center (u_c, v_c), then,

$$\forall i, w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} a_{ij} \begin{bmatrix} x_j^C \\ y_j^C \\ z_j^C \end{bmatrix} \tag{16}$$

Two linear equations can be obtained from Equation 16:

$$\sum_{j=1}^{4} a_{ij} f_u x_j^C + a_{ij}(u_c - u_i) z_j^C = 0 \tag{17}$$

$$\sum_{j=1}^{4} a_{ij} f_v y_j^C + a_{ij}(v_c - v_j) z_j^C = 0 \tag{18}$$

Concatenating all n reference points, we can obtain a linear system of equations:

$$\boldsymbol{M}\boldsymbol{x} = 0 \tag{19}$$

where $\boldsymbol{x} = [\boldsymbol{c}_1^{cT}, \boldsymbol{c}_2^{cT}, \boldsymbol{c}_3^{cT}, \boldsymbol{c}_4^{cT}]$, $\boldsymbol{x}$ is the coordinate of the control point in the camera coordinate system, which is a 12×1 vector and $\boldsymbol{x}$ is in the right null space of $\boldsymbol{M}$, or $x \in ker(\boldsymbol{M})$. $\boldsymbol{M}$ is a $2n \times 12$ matrix. Hence,

$$\boldsymbol{x} = \sum_{i=1}^{N} \beta_i \boldsymbol{v}_i \tag{20}$$

In the equation above, $\boldsymbol{v}_i$ is the N eigenvector corresponding to the N null eigenvalues of $\boldsymbol{M}$. For the i-th control point:

$$\boldsymbol{c}_j^C = \sum_{k=1}^{N} \beta_k \boldsymbol{v}_K^{[i]} \tag{21}$$

where $\boldsymbol{v}_k^{[i]}$ is i-th 3×1 sub-vector of eigenvector $\boldsymbol{v}_k$. Then we can obtain $\boldsymbol{v}_i$ by computing the eigenvectors of $\boldsymbol{M}^T\boldsymbol{M}$.

The next step is to calculate $\beta_{i,i=1,...,N}$. Because the extrinsic parameters of the camera describe only coordinate transformations and do not change the distance between control points, thus:

$$\left\| \boldsymbol{c}_i^C - \boldsymbol{c}_j^C \right\|^2 \tag{22}$$

$$\left\| \textstyle\sum_{k=1}^N \beta_k \boldsymbol{v}_K^{[i]} - \sum_{k=1}^N \beta_k \boldsymbol{v}_K^{[j]} \right\|^2 = \left\| \boldsymbol{c}_i^C - \boldsymbol{c}_j^C \right\|^2 \tag{23}$$

This is a linear equation for $\beta_{ij,i,j=1,...,N}$.In EPnP algorithm [20], four cases $N = 1, 2, 3, 4$ are discussed. When N takes different values, the number of unknowns of the linear equation are:

- N = 1, the unknowns number is 1
- N = 2, the unknowns number is 3
- N = 3, the unknowns number is 6
- N = 4, the unknowns number is 10

When $N = 4$, the number of equations is 6 and the number of unknowns is more than the number of equations. By commutativity of the multiplication, we have

$$\beta_{ab}\beta_{cd} = \beta_a\beta_b\beta_c\beta_d = \beta_{a'b'}\beta_{c'd'} \tag{24}$$

where $\{a', b', c', d'\}$ represents any permutation of the integers $\{a, b, c, d\}$. Then we can reduce the number of unknowns. For example, if we solve for $\beta_{11}, \beta_{12}, \beta_{13}$, then we get $\beta_{23} = \frac{\beta_{12}\beta_{13}}{\beta_{11}}$.

4.4.4. Gauss-Newton Optimization

The objective function of the optimization is:

$$Error(\beta) = \sum_{(i,j s.t. i<j)} (\left\| \boldsymbol{c}_i^C - \boldsymbol{c}_j^C \right\|^2 - \left\| \boldsymbol{c}_i^W - \boldsymbol{c}_j^W \right\|^2)^2 \tag{25}$$

4.4.5. Calculating the Camera's pose

The calculation of the pose of the camera in the EPnP algorithm is as follows.

1. Calculate the coordinates of the control point in the camera reference coordinate system.

$$\boldsymbol{c}_i^c = \sum_{j=1}^N \beta_k v_k^{[i]}, i = 1, 2, 3, 4 \tag{26}$$

2. Calculate the coordinates of the 3D reference point in the camera reference coordinate system.

$$\boldsymbol{p}_i^c = \sum_{j=1}^4 a_{ij}\boldsymbol{c}_j^c, i = 1, ..., n \tag{27}$$

3. Calculate the barycentric coordinates $\boldsymbol{p}_0^w$ of $\boldsymbol{p}_i^w, i = 1, ..., n$ and matrix **A**:

$$\boldsymbol{p}_0^w = \frac{1}{n}\sum_{i=1}^{n} p_i^w \tag{28}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{p}_1^{w^T} - \boldsymbol{p}_0^{w^T} \\ ... \\ \boldsymbol{p}_n^{w^T} - \boldsymbol{p}_0^{w^T} \end{bmatrix} \tag{29}$$

4. Calculate the barycentric coordinates $\boldsymbol{p}_0^c$ of $\boldsymbol{p}_i^c, i = 1, ..., n$ and matrix **B**:

$$\boldsymbol{p}_0^c = \frac{1}{n}\sum_{i=1}^{n} p_i^c \tag{30}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{p}_1^{c^T} - \boldsymbol{p}_0^{c^T} \\ ... \\ \boldsymbol{p}_n^{c^T} - \boldsymbol{p}_0^{c^T} \end{bmatrix} \tag{31}$$

5. Calculate **H**:

$$\boldsymbol{H} = \boldsymbol{B}^T\boldsymbol{A} \tag{32}$$

6. Calculating the singular value decomposition (SVD) of **H**:

$$\boldsymbol{H} = U\sum V^T \tag{33}$$

7. Calculate the rotation **R** in the pose:

$$\boldsymbol{R} = UV^T \tag{34}$$

8. Calculate the translation **t** in the pose:

$$\boldsymbol{t} = \boldsymbol{p_0^c} - \boldsymbol{Rp_0^w} \tag{35}$$

$\boldsymbol{l}$ represents the camera's position in the real world coordinate system:

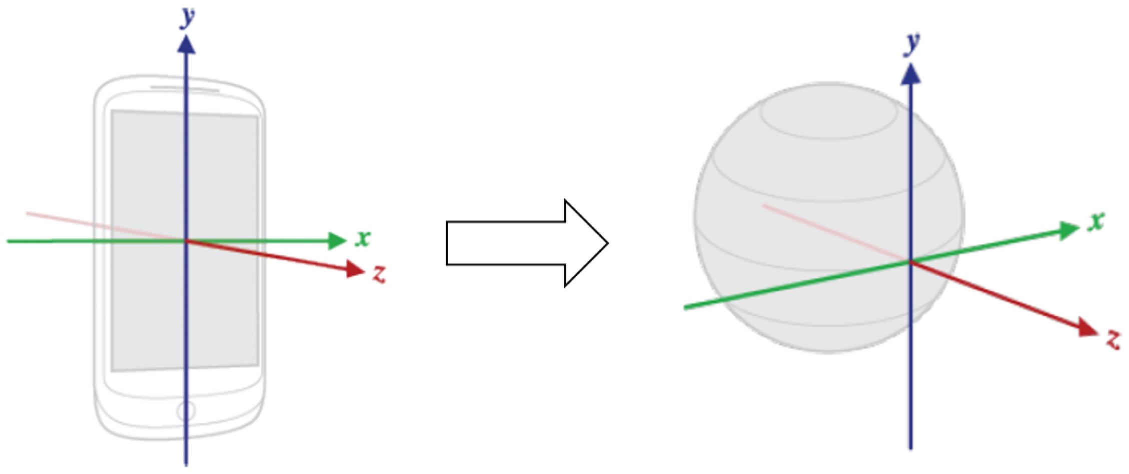
$$\boldsymbol{l} = -\boldsymbol{R}^{-1}\boldsymbol{t} \tag{36}$$

4.5. Sensor Fusion

The third module, is the IMU-based localization method. This method is used as a replacement when visual localization is not available. As can be seen in Figure 3, when the marker is not detected, the system uses the IMU measurements to calculate the 3D coordinates of the mobile device. The sensors used in this system are mainly accelerometer, gyroscope, and magnetometer, all of which are currently equipped in smartphones. Before we can use the sensor for localization, we need to convert the phone's coordinate system.

4.5.1. Coordinate System Conversion

The acceleration sensor of an Android phone refers to its coordinate system when measuring acceleration. Therefore, we need to convert the phone's coordinate system to an inertial, non-rotating coordinate system, which is the Earth coordinate system. This conversion will make it possible to hold the Android phone in any orientation and measure the correct acceleration vectors to calculate the phone's trajectory in the earth coordinate system. Figure 7 displays this transformation. This transformation[21] is done with formula (37) as shown below.

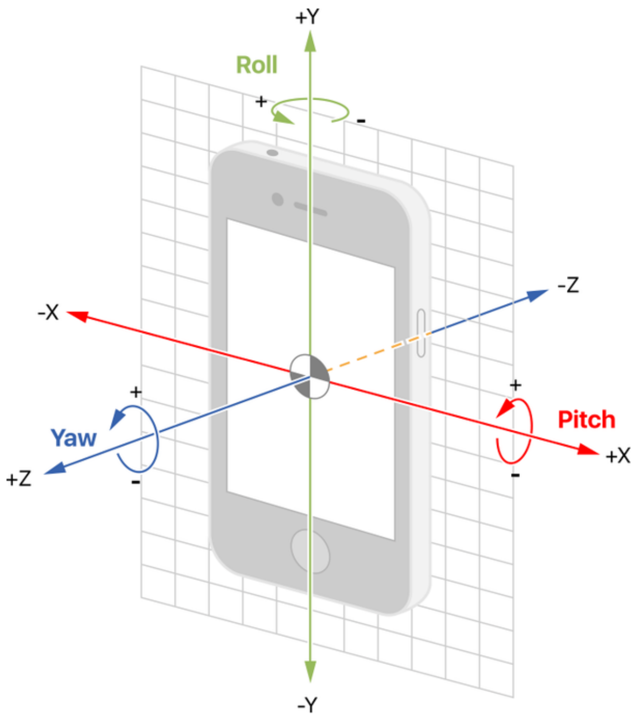


Source: https://journals.plos.org/plosone/article/figure?id=10.1371/journal.pone.0174959.g002

Figure 7. The transformation from Phone coordinate to Earth's

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_z(\psi)R_y(\theta)R_x(\phi)\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{37}$$

Where $[X, Y, Z]$ are the phone's coordinate system linear accelerations, and R_z, R_y, R_x are the rotation matrices for each axis in order to rotate the [X, Y, Z] over to earth's $[x, y, x]$ axes. The Euler angles (ψ, θ, ϕ) correspond to the angles about the pitch, roll, and yaw, which are shown in Figure 8. The difference between the acceleration in the earth coordinate system (The green line) and the Android phone coordinate system (The gray line) can be seen in Figure 9. The vertical coordinate in the figure represents the value of the sensor, and the horizontal coordinate represents the sampling times (we used a sampling rate of 100 Hz). After the conversion, the Z-axis's gravity stays near 9.8 meters per second, while the gravity in the X-axis and Y-axis stays near 0 meters per second. This result is the same as what we know as common sense.



Source: https://www.quora.com/How-does-smartphone-sensor-orientation-work

Figure 8. The phone's axes

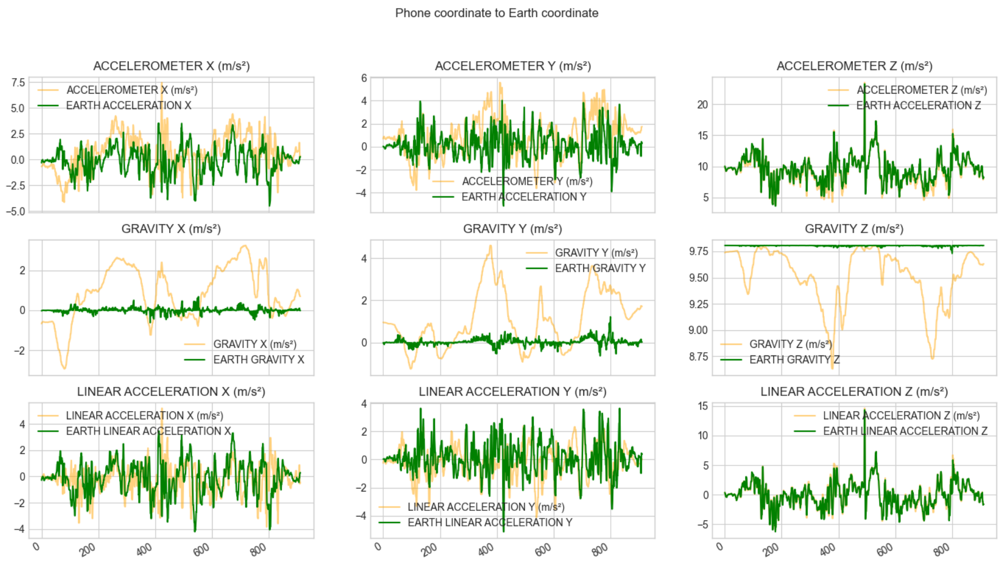


Figure 9. Change in sensor values before(orange) and after(green) coordinate system conversion

4.5.2. Displacement

Acceleration is the rate of change of an object's velocity. At the same time, velocity is the rate of change of the position of the same object. In other words, velocity is the derivative of position, and acceleration is the derivative of velocity. Therefore, the following equation is available,

$$\vec{a} = \frac{d\vec{v}}{dt} \tag{38}$$

$$\vec{v} = \frac{d\vec{s}}{dt} \tag{39}$$

Hence,

$$\vec{a} = \frac{d(d\vec{s})}{dt^2} \tag{40}$$

The integral is the opposite of the derivative. If the acceleration of an object is known, then we are able to obtain the position of the object using the double integral. Assuming that the initial condition is 0, then there is the following equation.

$$\vec{v} = \int (\vec{a}) \, \mathrm{d}t \tag{41}$$

$$\vec{s} = \int (\vec{v}) \, \mathrm{d}t \tag{42}$$

Hence,

$$\vec{s} = \int (\int (\vec{a}) \, \mathrm{d}t) \, \mathrm{d}t \tag{43}$$

Where $\vec{v}$, $\vec{a}$ and $\vec{s}$ are velocity, acceleration and displacement respectively. Therefore, the displacement in the x-axis is shown in the following equation,

$$\vec{s_x} = \int (\int (\vec{a_x}) \, \mathrm{d}t) \, \mathrm{d}t \tag{44}$$

Similar expression for displacement in y-axis and z-axis.

4.5.3. Noise Filtering

When we integrate the accelerations, we also integrate the noise — which causes a drift in the calculated trajectory. Therefore, we introduce a low-pass filter to attenuate the noise. First, we perform a Fourier analysis of the accelerometer values to obtain the spectrum of the noise. Figure 10 illustrates the noise spectrum for each axis. We can see that the noise is concentrated in the low-frequency band, and it looks like we should use a high-pass filter to be the right choice. However, the signal we are supposed to measure is also concentrated in the low-frequency band, so a low-pass filter is the genuinely right choice. Using a low-pass filter can make the signal smoother. Figure 11 shows the variation of the acceleration values after the low-pass filter. The vertical coordinate in the figure represents the value of the sensor, and the horizontal coordinate represents the sampling times (we used a sampling rate of 100 Hz). As can be seen from the figure, the values on each axis are attenuated, which corresponds to a partial attenuation of the noise.

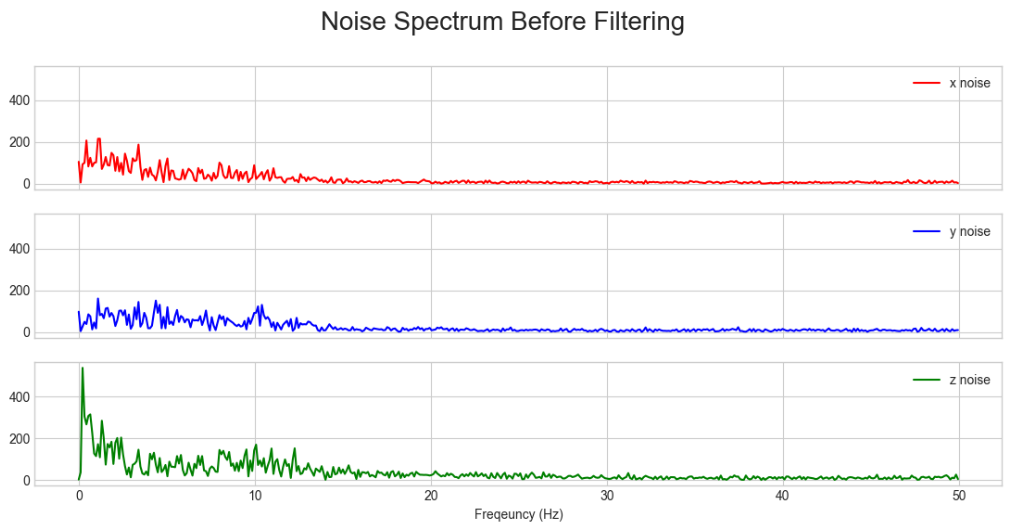


Figure 10. Noise spectrum of each axis

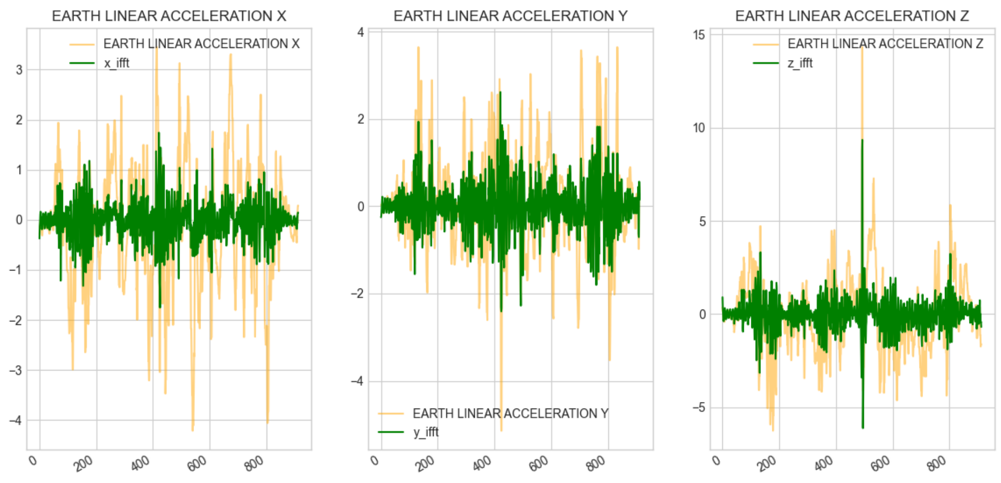


Figure 11. Comparison of accelerometer measurements before(orange) and after(green) filter

4.6. Kalman Filter

As mentioned earlier, when no marker appears on the screen or the system cannot detect a marker that appears on the screen, the system will call the IMU for localization. This is module 4 of the system. The inertial sensors that come with mobile phones are relatively cheap and thus prone to generating a lot of noise. When we use inertial sensors to estimate the position of a moving phone, we need to double integrate the acceleration. However, at the same time, we also integrate the noise. This leads to an accumulation of errors, which in turn leads to incorrect localization and tracking of mobile phones. In our proposed method, the absolute position obtained by vision-based localization is used to update the position prediction obtained by IMU-based tracking via Kalman filter.

Kalman filtering [22] is mainly divided into two steps, prediction plus correction. Prediction is the estimation of the current state based on the state of the previous moment, and correction is the integrated analysis based on the observation of the current state and the estimation of the previous moment to estimate the system's optimal state value. Then the process is repeated the next moment. The Kalman filter iterates continuously, it does not require a large number of particle state inputs, only process quantities, so it is fast and well suited for state estimation of linear systems.

Applying Kalman filtering to this project uses the position information obtained from vision-based localization to update the position information obtained from sensor-based localization. The process of constructing the Kalman filter state space for this project is described in detail below.

The position and velocity of the device at moment t can be deduced from the position and velocity at moment $t-1$.

$$P_t = P_t - 1 + V_t - 1 \times \delta t + \frac{\delta t^2}{2} a_t \tag{45}$$

$$V_t = V_t - 1 + \delta t \times a_t \tag{46}$$

Where P, V, a, and δt, are position, velocity, acceleration, and time interval, respectively. According to the state prediction formula of the Kalman filter,

$$\widehat{X}_t^- = \boldsymbol{A}\widehat{X}_{t-1}^- + \boldsymbol{B}a_t \tag{47}$$

Then combining equations (45) and (46), in a three-dimensional coordinate system, we can obtain,

$$\widehat{X}_t^- = \boldsymbol{A}\widehat{X}_{t-1}^- + \boldsymbol{B}a_t \rightarrow \begin{bmatrix} P_t^x \\ P_t^y \\ P_t^z \\ V_t^x \\ V_t^y \\ V_t^x \end{bmatrix} = \begin{bmatrix} P_{t-1}^x + V_{t-1}^x \times \delta t + \frac{\delta t^2}{2} a_t^x \\ P_{t-1}^y + V_{t-1}^y \times \delta t + \frac{\delta t^2}{2} a_t^y \\ P_{t-1}^z + V_{t-1}^z \times \delta t + \frac{\delta t^2}{2} a_t^z \\ V_{t-1}^x + \delta t \times a_t^x \\ V_{t-1}^y + \delta t \times a_t^y \\ V_{t-1}^z + \delta t \times a_t^z \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{t-1}^x \\ P_{t-1}^y \\ P_{t-1}^z \\ V_{t-1}^x \\ V_{t-1}^y \\ V_{t-1}^z \end{bmatrix} + \begin{bmatrix} \frac{\delta t^2}{2} & 0 & 0 \\ 0 & \frac{\delta t^2}{2} & 0 \\ 0 & 0 & \frac{\delta t^2}{2} \\ \delta t & 0 & 0 \\ 0 & \delta t & 0 \\ 0 & 0 & \delta t \end{bmatrix} \begin{bmatrix} a_t^x & a_t^y & a_t^z \end{bmatrix} \tag{48}$$

Therefore, the state transition matrix $\boldsymbol{A}$ should then be

$$\begin{bmatrix} 1 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{49}$$

The control matrix should be

$$\begin{bmatrix} \frac{\delta t^2}{2} & 0 & 0 \\ 0 & \frac{\delta t^2}{2} & 0 \\ 0 & 0 & \frac{\delta t^2}{2} \\ \delta t & 0 & 0 \\ 0 & \delta t & 0 \\ 0 & 0 & \delta t \end{bmatrix} \tag{50}$$

The measurement should be a 3×1 matrix, that is

$$\begin{bmatrix} P_t^x \\ P_t^y \\ P_t^z \end{bmatrix} \tag{51}$$

Thus the measurement matrix can be derived as

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{52}$$

5. Implementation

5.1. Overview

This section will describe how we implemented this multi-technology based indoor localization technology.

5.2. Dataset Collection

In this project, we need to collecting dataset by ourselves since no open-source dataset is available. Our dataset is divided into a training set and a test set. The training set is used to train the model and tune the parameters of the model. The test set is used to evaluate the detection ability of the model.

5.2.1. Data Processing

The camera that comes with Nokia 7.2 was used to capture the images. The Nokia 7.2 provides a triple camera, which including a 48 MP main camera and ZEISS Optics. With the 118 wide-angle lens and ZEISS Optics, it can capture the entire scene [23]. With this camera, we can have a larger field of view, which means we will miss fewer markers during the detection process. The locations were the lab inside Prairie Springs Science Center. We printed out several ArUco markers and placed them in various locations around the room. The number of markers that appear in the images we took varies to meet the need for the model to detect different numbers of markers.

In order to satisfy that the model can detect markers under different light conditions, the images in the dataset also have to include both sufficient and insufficient light conditions. As shown in Figure 12(c), it is an insufficient light image. The model also needs to be able to detect markers that are blurred due to rapid camera movement. As shown in Figure 12(b), it is a blurred image caused by the rapid movement of the camera. Therefore, the dataset should also include pictures of the blurred markers.

To meet these needs, we first took a large number of images and then processed them further. We used a combination of Python and OpenCV to add image effects. The following code implements adding motion blur to the image and adjusting the brightness of the image, respectively.

```python
import numpy as np
import cv2

def motion_blur(input_img_path, output_img_path, degree=12, angle=45):
    image = cv2.imread(input_img_path)

    image = np.array(image)

    # Here the matrix of motion blur kernel is generated for any angle, the larger the degree
    # The higher the degree of blur
    M = cv2.getRotationMatrix2D((degree / 2, degree / 2), angle, 1)
    motion_blur_kernel = np.diag(np.ones(degree))
    motion_blur_kernel = cv2.warpAffine(motion_blur_kernel, M, (degree, degree))

    motion_blur_kernel = motion_blur_kernel / degree
    blurred = cv2.filter2D(image, -1, motion_blur_kernel)

    # convert to uint8
    cv2.normalize(blurred, blurred, 0, 255, cv2.NORM_MINMAX)
    blurred = np.array(blurred, dtype=np.uint8)
    cv2.imwrite(output_img_path, blurred)

# Get the path of the image to be transformed and generate the target path
image_filenames = [(os.path.join(dataset_dir, x), os.path.join(output_dir, x))
                   for x in os.listdir(dataset_dir)]
# Convert all images
for path in image_filenames:
    motion_blur(path[0], path[1])
```

Listing 1. Code for adding motion blur to an image

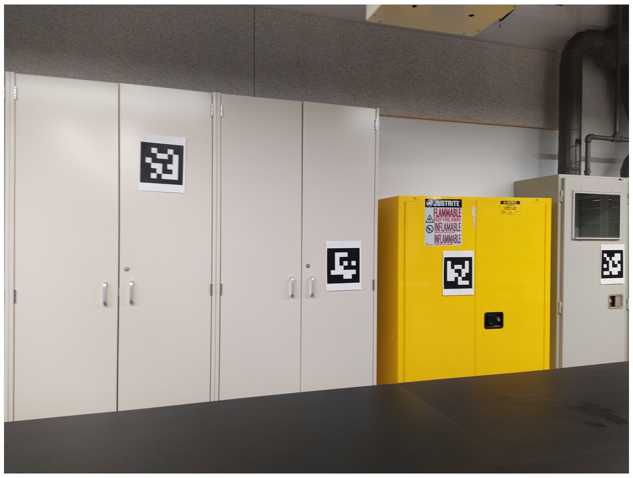
```python
import numpy as np
import cv2
import os

MAX_VALUE = 100

def adjustment(input_img_path, output_img_path, lightness, saturation):
    """
    For modifying the lightness and saturation of images
    :param input_img_path
    :param output_img_path
    :param lightness
    :param saturation
    """

    # Load image, read color image normalized and converted to floating point
    image = cv2.imread(input_img_path, cv2.IMREAD_COLOR).astype(np.float32) / 255.0

    # Color space conversion BGR to HLS
    hlsImg = cv2.cvtColor(image, cv2.COLOR_BGR2HLS)

    # Adjust brightness (linear transformation)
    hlsImg[:, :, 1] = (1.0 + lightness / float(MAX_VALUE)) * hlsImg[:, :, 1]
    hlsImg[:, :, 1][hlsImg[:, :, 1] > 1] = 1
    # Saturation
    hlsImg[:, :, 2] = (1.0 + saturation / float(MAX_VALUE)) * hlsImg[:, :,
    2]
    hlsImg[:, :, 2][hlsImg[:, :, 2] > 1] = 1
    # HLS2BGR
    lsImg = cv2.cvtColor(hlsImg, cv2.COLOR_HLS2BGR) * 255
    lsImg = lsImg.astype(np.uint8)
    cv2.imwrite(output_img_path, lsImg)

dataset_dir = 'DATASET_DIR_PATH'
output_dir = 'OUTPUT_PATH'

# parameters adjustment
lightness = int(input("lightness(lightness-100~+100):"))
saturation = int(input("saturation(saturation-100~+100):"))

# Get the path of the image to be transformed and generate the target path
image_filenames = [(os.path.join(dataset_dir, x), os.path.join(output_dir,
    x))
                   for x in os.listdir(dataset_dir)]
# Convert all images
for path in image_filenames:
    adjustment(path[0], path[1], lightness, saturation)
```

Listing 2. Code for lightness adjustment

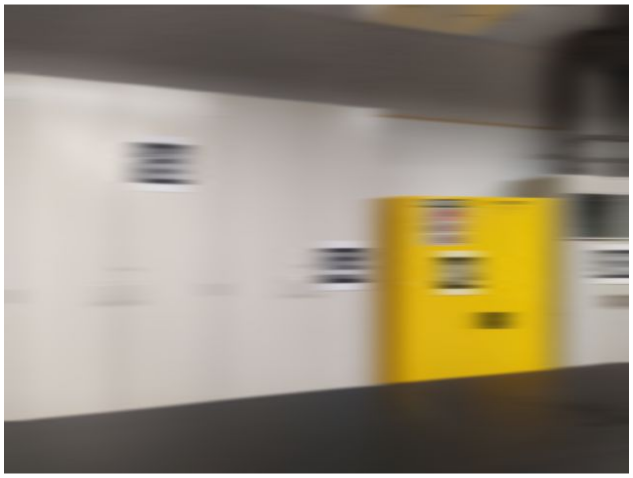
5.2.2. Data Labeling

In machine learning, data labeling is used to identify raw data (images, text files, videos, and so on.) and add one or more meaningful tags of information to provide context so that machine learning models can learn from it. For example, tags can indicate whether a photo contains a bird or a car, which words are pronounced in a recording, or whether an X-image contains a tumor. Data labeling is required for various use cases, including computer vision, natural language processing, and speech recognition. It is also an indispensable part of making a dataset. *LabelImg* [24] is a handy tool that helps me to finish this task. It is a graphical image annotaion tool and written in Python and uses Qt for its graphical interface.

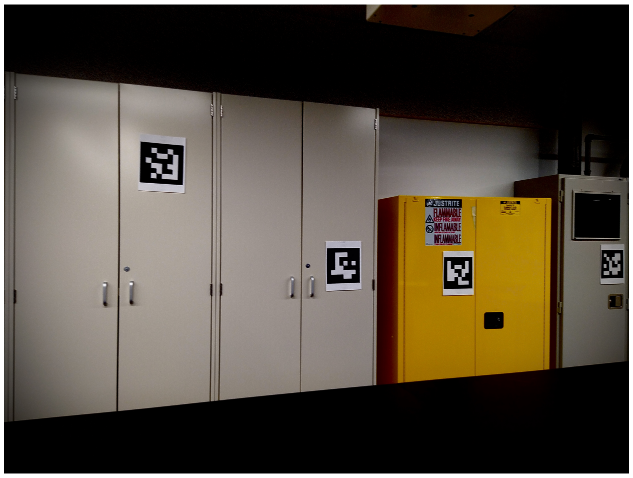
LabelImg supports the YOLO format. In the YOLO dataset format, a file with the same name is created for each image file in the same directory. Each file contains the corresponding image file: object class, object coordinates, height, and width. Figure 13 displays the data labeling process of this project. The results of labeling can be saved as XML files or TXT files. This labeling process requires us to mark out the boxes that enclose the markers manually. After drawing a box each time, we have to mark what object the box belongs to. As shown in Figure 13, we have marked four boxes, and the right side shows the categories of the boxes, all of which are ArUco codes. After completing the annotation of one image, we can import the following image and cycle through the process. Images are classified based on their different conditions, such as lighting and blurring. The number of training sets accounted for 80% of the total number of data sets, while the number of test sets accounted for 20%. As can be seen from the Figure 14, this dataset does not have a validation set. The reason is that the validation set is automatically divided from the training set at the



(a) The original image



(b) The blurred image



(c) The insufficient illumination image

Figure 12. The example after adjusting images

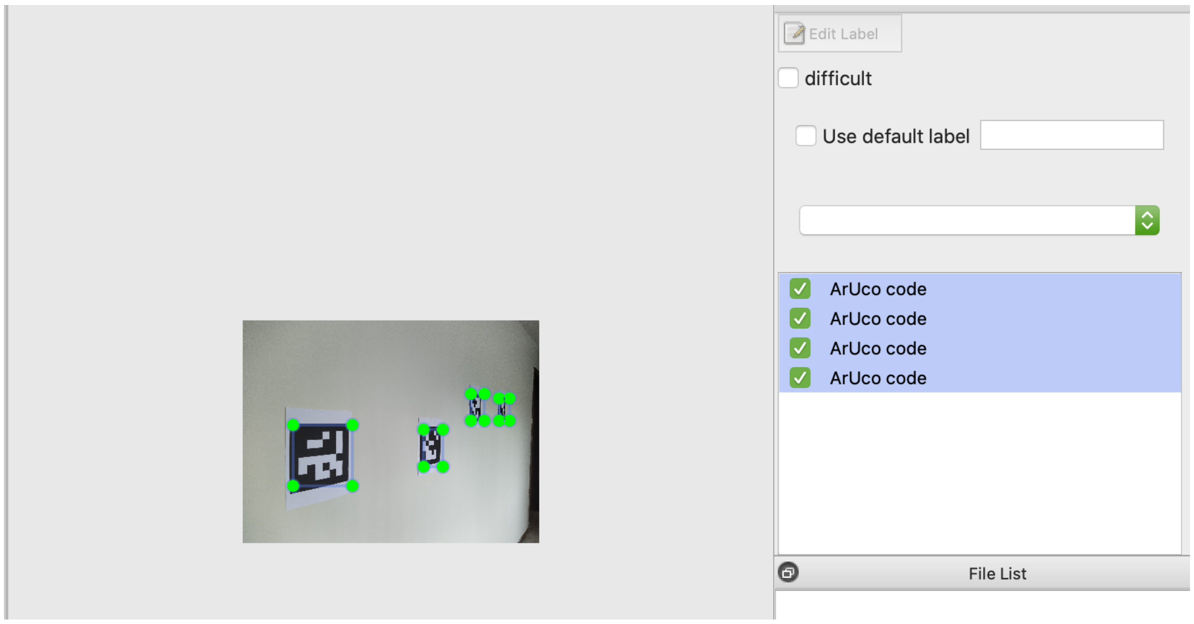


Figure 13. The processing of data labeling

beginning of model training. The test set is needed to evaluate the model performance. Figure 14 illustrates the data set structure.

5.3. ArUco Marker Generation

ArUco code can be generated in multiple sizes. The size is selected based on the object size and the scene for successful detection. If very small markers go undetected, simply increasing their size can make their detection easier. These markers can be easily generated using OpenCV. ArUco module in OpenCV has 25 predefined marker dictionaries. All markers in the dictionary contain the same number of blocks or bits (4×4, 5×5, 6×6 or 7×7), and each dictionary contains a fixed number of markers (50, 100, 250 or 1000). The following is an example of batch generating ArUco markers and detecting the markers.

After generating the markers, we print out and take a picture of them for detection testing, which is shown as Figure 16 and 17

Given an image in which the ArUco markers are visible, the detection program should return a list of detected markers. Each detected marker includes:

- Position of the four corners of the image (in the original order)
- Id of marker

5.4. Camera Calibration

The calibration steps are introduced as follows. First, we need to fix the ArUco checkerboard figure onto a flat surface and then take at least 20 photos from different angles and positions using the cell phone camera. Second, The *Aruco.calibrateCameraAruco()* function will

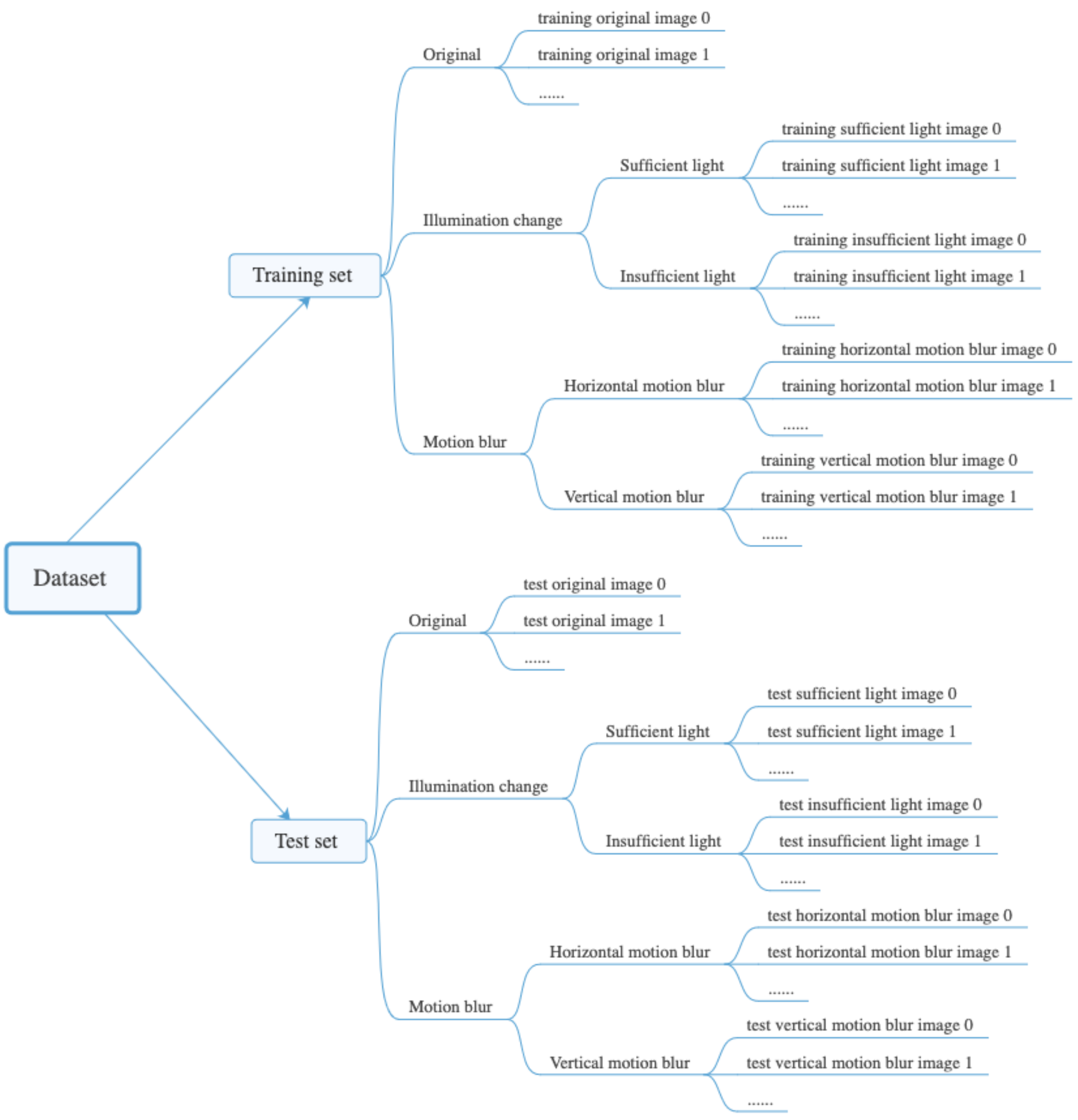

Dataset
Training set
Original
training original image 0
training original image 1
......
Illumination change
Sufficient light
training sufficient light image 0
training sufficient light image 1
......
Insufficient light
training insufficient light image 0
training insufficient light image 1
......
Motion blur
Horizontal motion blur
training horizontal motion blur image 0
training horizontal motion blur image 1
......
Vertical motion blur
training vertical motion blur image 0
training vertical motion blur image 1
......
Test set
Original
test original image 0
test original image 1
......
Illumination change
Sufficient light
test sufficient light image 0
test sufficient light image 1
......
Insufficient light
test insufficient light image 0
test insufficient light image 1
......
Motion blur
Horizontal motion blur
test horizontal motion blur image 0
test horizontal motion blur image 1
......
Vertical motion blur
test vertical motion blur image 0
test vertical motion blur image 1
......

Figure 14. The dataset structure

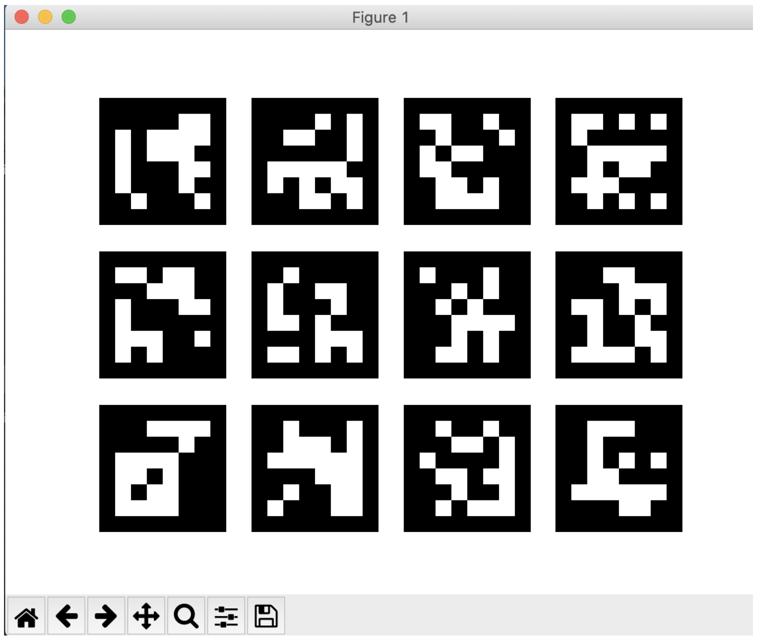


Figure 15. Batch-generated markers containing a 6x6-bit binary

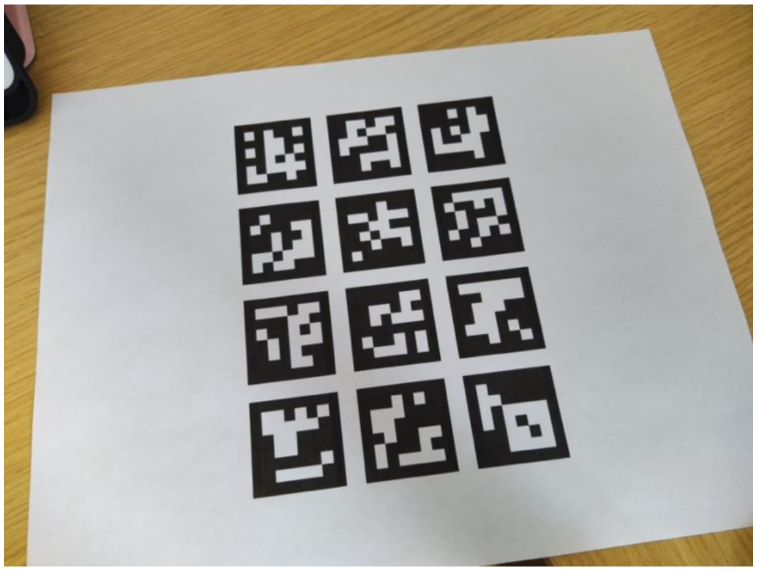


Figure 16. The original image of markers

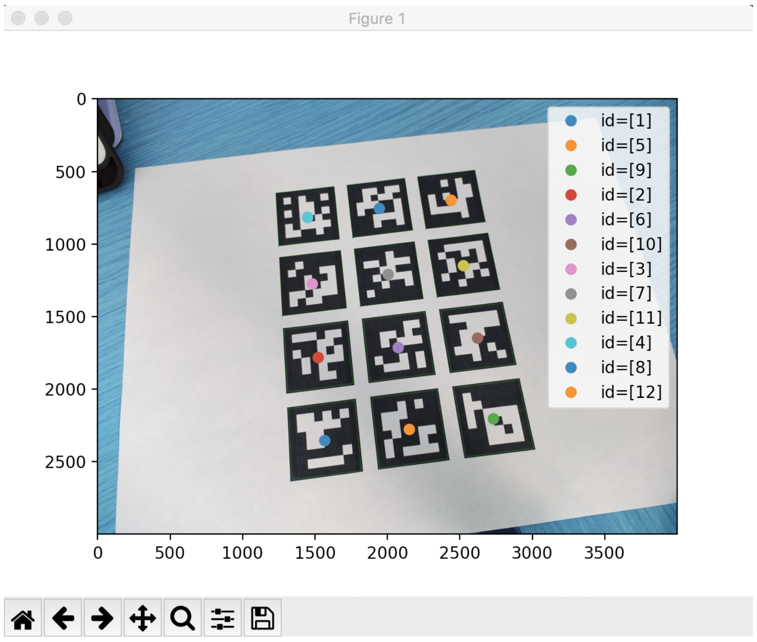


Figure 17. IDs of the detected markers

calibrate camera. Finally, the calibration results are saved and will be used in the detection of the marker object. The following shows the part of code for calibrating the camera and saving the calibration results.

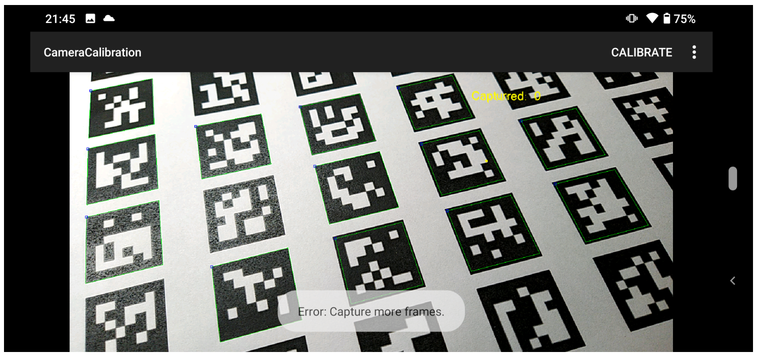
```java
public double calibrate(){
    List<Mat> rvecs=new LinkedList<>();
    List<Mat> tvecs=new LinkedList<>();

    return Aruco.calibrateCameraAruco(
            allCornersConcatenated,
            allIdsConcatenated,
            markerCounterPerFrame,
            board,
            size,
            cameraMatrix,
            distCoeffs,
            rvecs,
            tvecs
    );
}
```

Listing 3. Partial Code for camera calibration

```java
public static void save(Activity activity, Mat cameraMatrix, Mat
   distortionCoefficients) {
    SharedPreferences sharedPref = activity.getPreferences(Context.
   MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPref.edit();

    double[] cameraMatrixArray = new double[CAMERA_MATRIX_ROWS * CAMERA_MATRIX_COLS];
    cameraMatrix.get(0,  0, cameraMatrixArray);
    for (int i = 0; i < CAMERA_MATRIX_ROWS; i++) {
        for (int j = 0; j < CAMERA_MATRIX_COLS; j++) {
            Integer id = i * CAMERA_MATRIX_ROWS + j;
            editor.putFloat(id.toString(), (float)cameraMatrixArray[id]);
        }
    }

    double[] distortionCoefficientsArray = new double[DISTORTION_COEFFICIENTS_SIZE];
    distortionCoefficients.get(0, 0, distortionCoefficientsArray);
    int shift = CAMERA_MATRIX_ROWS * CAMERA_MATRIX_COLS;
    for (Integer i = shift; i < DISTORTION_COEFFICIENTS_SIZE + shift; i++) {
        editor.putFloat(i.toString(), (float)distortionCoefficientsArray[i-shift]);
    }

    editor.commit();
    Log.i(TAG, "Camera matrix: " + cameraMatrix.dump());
    Log.i(TAG, "Distortion coefficients: " + distortionCoefficients.dump());
}
```

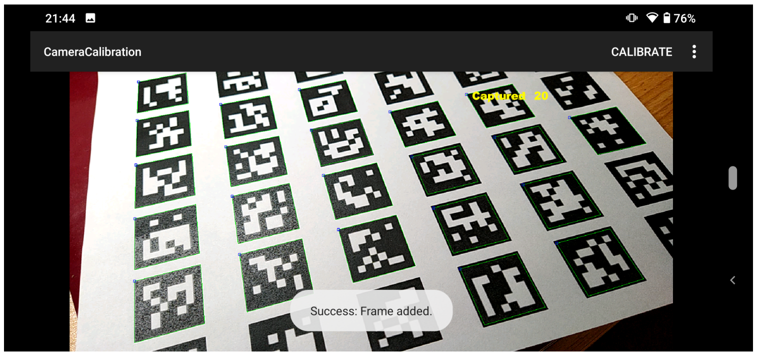
Listing 4. Partial Code for saving camera calibration's result

When we launch this application, we can capture pictures with just a simple tap on the screen. As mentioned earlier, at least 20 pictures have to be captured before the camera calibration can be done. The top right corner of the screen shows the number of pictures currently captured. If we click on the calibration button without taking enough pictures, a prompt will pop up at the screen's bottom, as shown in Figure 18. Also, a corresponding prompt will pop up for each successful picture , as shown in Figure 19. As displayed in Figure 20 and Figure 21, after enough pictures have been taken, we can click the calibration button in the upper right corner, and the calibration results will be displayed at the bottom of the screen and stored in this phone, waiting to be called.



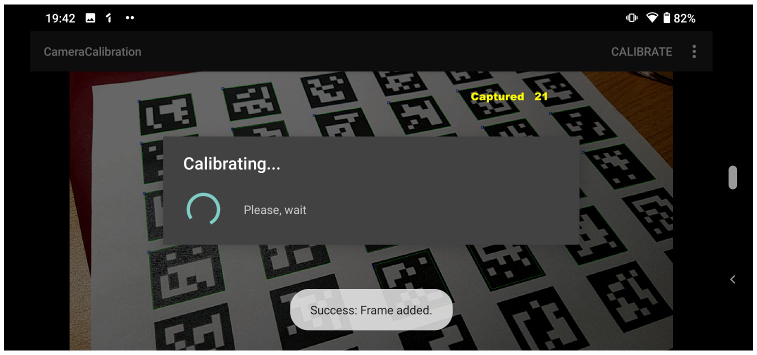
Error: Capture more frames.

Figure 18. Screenshot of the prompt for insufficient frames



Success: Frame added.

Figure 19. Screenshot of the prompt for successfully picture captured

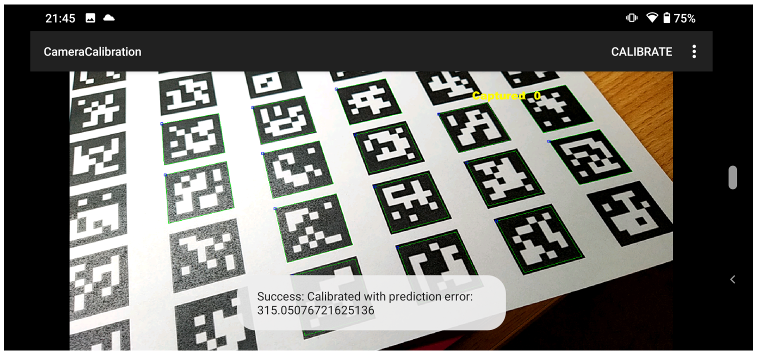


Calibrating...

Please, wait

Success: Frame added.

Figure 20. Screenshot of after clicking the 'Calibrate' button



Success: Calibrated with prediction error: 315.05076721625136

Figure 21. Screenshot of the prompt for successfully calibrate

5.5. ArUco Marker Detection

Before detecting the ArUco markers, we need to load the calibration results. The *Camera-Parameter* class is responsible for this task, and the code for this class is shown below.

```java
public static boolean tryLoad(Activity activity, Mat cameraMatrix, Mat distCoeffs) {
    try{
        Context context = activity.createPackageContext(CAMERA_CALIBRATION_PKG, Context.CONTEXT_IGNORE_SECURITY);
        SharedPreferences cameraPrefs = context.getSharedPreferences(CAMERA_CALIBRATION_PREFS, Context.MODE_WORLD_READABLE);

        if(cameraPrefs.getFloat("0",-1)==-1)
            return false;
        else
            load(cameraPrefs,cameraMatrix,distCoeffs);

    }
    catch(PackageManager.NameNotFoundException e){
        e.printStackTrace();
    }
    return true;
}

private static void load(SharedPreferences sharedPreferences, Mat cameraMatrix, Mat distCoeffs) {
    double[] cameraMatrixArray = new double[CAMERA_MATRIX_ROWS * CAMERA_MATRIX_COLS];

    for (int i = 0; i < CAMERA_MATRIX_ROWS; i++) {
        for (int j = 0; j < CAMERA_MATRIX_COLS; j++) {
            int id = i * CAMERA_MATRIX_ROWS + j;
            cameraMatrixArray[id] = sharedPreferences.getFloat(Integer.toString(id), -1);
        }
    }
    cameraMatrix.put(0,0,cameraMatrixArray);

    double[] distortionCoefficientsArray = new double[DISTORTION_COEFFICIENTS_SIZE];
    int shift = CAMERA_MATRIX_ROWS * CAMERA_MATRIX_COLS;

    for(int i = shift; i < DISTORTION_COEFFICIENTS_SIZE + shift; i++){
        distortionCoefficientsArray[i-shift] = sharedPreferences.getFloat(Integer.toString(i),-1);
    }
    distCoeffs.put(0,0,distortionCoefficientsArray);
}
```

Listing 5. Partial code for loading the result of camera calibration

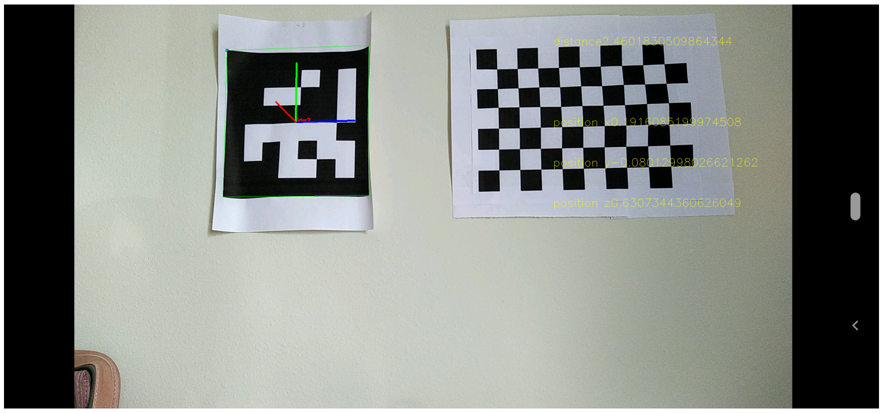
The dictionary of ArUco code we use is 6x6x250, which is required to be declared before detection. Since we need to use the camera's live image, we need to request permission to open the camera when the program launches. As each frame is passed into the system, the RGB image will be converted to grayscale before the markers are detected. When a marker is detected, the system frames the detected marker and draws the coordinate axis and its ID. The camera's footage will occupy the majority of the screen for display. Figure 22 shows the screenshot of the current system.

```java
// Get the permission of camera
if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
        PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[] {Manifest.
permission.CAMERA},
    50); }
```

Listing 6. Partial code for getting camera permission

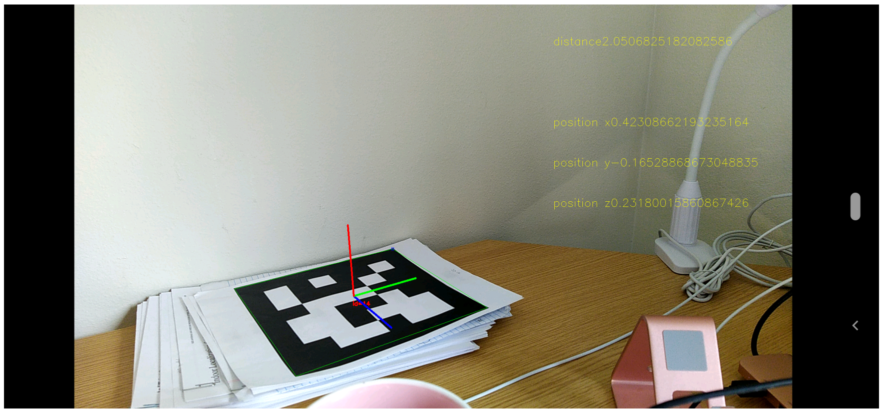
```java
// Image grayscale
Imgproc.cvtColor(inputFrame.rgba(), rgb, Imgproc.COLOR_RGBA2RGB);
gray=inputFrame.gray();
// Create a one-centered matrix object to store the IDs of the detected
markers
ids = new MatOfInt();
corners.clear();
// Marker detection
Aruco.detectMarkers(gray, dictionary, corners, ids, parameters);

if(corners.size()>0){
    // Basket out the markers
    Aruco.drawDetectedMarkers(rgb, corners, ids);

    rvecs=new Mat();
    tvecs=new Mat();

    Aruco.estimatePoseSingleMarkers(corners,0.025f,cameraMatrix,distCoeffs
,rvecs,tvecs);

    for(int i=0; i<ids.toArray().length; i++){
        transformModel(tvecs.row(0),rvecs.row(0));
        Aruco.drawAxis(rgb, cameraMatrix, distCoeffs, rvecs.row(i), tvecs.
row(i),0.01f);
    }
}
```

Listing 7. Partial code for drawing axis



```
distance2.4601830509864344
position x0.1916085199974508
position y-0.08012998026621262
position z0.6307344360626049
```

(a) The first screenshot



```
distance2.0506825182082586
position x0.42308662193235164
position y-0.16528868673048835
position z0.23180015860867426
```

(b) The second screenshot

Figure 22. Screenshot of the current system detecting the marker

5.6. YOLOV3

5.6.1. Training

We train the YOLOV3 on Google Colab. The framework we use is Darknet, which we use to train our custom dataset. In the first step, we need to modify the file that defines the network structure according to our self-define dataset. Since our dataset contains only one category, which is the ArUco code, the corresponding number of filters can be calculated as 18 according to this formula (53) below.

$$filters = (classes + 5) \times 3 \tag{53}$$

where filters, classes indicates the number of filters and number of classes.
The following is a portion of the document that defines the network structure.

```
[net]
# Testing
# batch=1
# subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 2000
policy=steps
steps=1600, 1800
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky
```

Listing 8. Partial document for network structure

After uploading the dataset and the corresponding files to Google Colab, we can start training the model. A weight file is obtained at the end of training and will be used in the process of detecting markers.

5.6.2. Markers Detection by YOLOV3 in Android

After training the neural network, our task becomes to make the neural network detect markers on the phone. We use OpenCV to help with this task. First, we need to add OpenCV as a dependency of the project. This step was done in the previous camera calibration. Place the weight file obtained after training into the Android project and load the weight file with the function *getAssetsFile()*. The next step is to pass the camera frame into the network for detection. Finally, the coordinate points of the four corners of the detected markers are returned. Some of the code to accomplish this task is shown below, and the complete code can be seen in the appendix.

```java
public List<Corners> markerDetection(CameraBridgeViewBase.
   CvCameraViewFrame inputFrame) {

    List<Corners> detectionResult = new ArrayList<>();

    classNames = readLabels("labels.txt", this);
    for(int i=0; i<classNames.size(); i++) {
        colors.add(randomColor());
    }
    // Loading network
    String modelConfiguration = getAssetsFile("yolov3_aruco.cfg", this);
    String modelWeights = getAssetsFile("yolov3_aruco_final.weights", this
    );
    yolo = Dnn.readNetFromDarknet(modelConfiguration, modelWeights);

    Mat frame = inputFrame.rgba();
    Imgproc.cvtColor(frame, frame, Imgproc.COLOR_RGB2RGBA);
    Size frame_size = new Size(416, 416);
    Scalar mean = new Scalar(127.5);

    ......
}
```

Listing 9. Partial code for markers detection by yolov3 in Android

5.7. Vision-Based Localization

Once we have obtained the two-dimensional coordinate points of the markers, we can move to the next stage, solving the device's position in the three-dimensional coordinate system. That is vision-based indoor localization. The known 3D coordinates of the marker are stored in a class called Coordinates, and the corresponding 3D coordinates are obtained based on the marker's ID.The *Calib3d.solvePnP()* function can then be used to calculate the device's rotation and translation matrices. These two matrices will be used to calculate the position of the device. The following code fragment shows how to calculate a device's location and finally return the result to the function that calls it.

```
private Mat getCameraPose(MatOfPoint2f imagePoints, int id) {
    Mat cameraPosition;

    MatOfPoint3f mObjectPoints = new MatOfPoint3f();
    List<Point3> objectPoints = new ArrayList<>();
    double[][] coordinates = Coordinates.getCoordinates(id);
    objectPoints.add(new Point3(coordinates[0][0], coordinates[0][1],
    coordinates[0][2]));
    objectPoints.add(new Point3(coordinates[1][0], coordinates[1][1],
    coordinates[1][2]));
    objectPoints.add(new Point3(coordinates[2][0], coordinates[2][1],
    coordinates[2][2]));
    objectPoints.add(new Point3(coordinates[3][0], coordinates[3][1],
    coordinates[3][2]));
    mObjectPoints.fromList(objectPoints);

    double[] distortionCoefficientsArray = {1.15562467e-01, -1.87996508e
    -01,  1.54664020e-04, -2.89903773e-04, 2.00265575e-02};
    MatOfDouble mDistCoeffs = new MatOfDouble();
    mDistCoeffs.fromArray(distortionCoefficientsArray);

    // Get the rotation vector of current camera pose
    Calib3d.solvePnP(mObjectPoints, imagePoints, cameraMatrix, mDistCoeffs
   , rvecs, tvecs, false, Calib3d.SOLVEPNP_EPnP);

    Mat rotationMatrix = new Mat();
    // Convert rotation vector to ratation matrix
    Calib3d.Rodrigues(rvecs, rotationMatrix);
    rotationMatrix = rotationMatrix.inv();
    // Calculate camera position, corresponding to equation (25) in
   section 4
    cameraPosition = star(star(rotationMatrix, -1), tvecs);

    return cameraPosition;
}
```

Listing 10. Partial code for calculating device's 3D coordinates

5.8. Sensor Fusion-Based Localization

As mentioned in section 4.5.1., we need to find the rotation matrix before converting the phone coordinate system to the Earth coordinate system. Android provides a function called *getRotationMatrix()*, which can be used to find the rotation matrix. This function requires the use of accelerometer and magnetometer. Once we have the acceleration values on the Earth coordinate system, we can start calculating the Android device's position. The 3D coordinates of the Android device can be obtained by double integration of the acceleration on each of the three axes according to the method mentioned in the section 4.5.2..

```
final SensorEventListener myListener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
            magneticFieldValues = event.values;
        }
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            accelerometerValues = event.values;
        }
        float[] rotationMatrix = calculateRotationMatrix();
        acceleration.add(times(rotationMatrix, accelerometerValues));
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
};

private float[] calculateRotationMatrix() {
    float[] values = new float[3];
    float[] rotationMatrix = new float[9];
    SensorManager.getRotationMatrix(rotationMatrix, null,
   accelerometerValues, magneticFieldValues);
    SensorManager.getOrientation(rotationMatrix, values);

```

```java
        return rotationMatrix;
}
```

5.9. Kalman Filter

As mentioned in the section 4.6., Kalman filtering is mainly divided into two parts: prediction and update. Since we need information about the device's location in space, this Kalman filter will be a 3D Kalman filter.

```java
public void kalmanFilter(float x, float y, float z) throws Exception {
    JKalman kalman = new JKalman(6, 3);

    Matrix stateMatrix = new Matrix(6, 1); // [x, y, z, dx, dy, dz]
    Matrix correctMatrix = new Matrix(6, 1);
    Matrix measurement = new Matrix(3, 1); // measurement [x, y, z]

    // x, y, z is the coordinates from vision-based localization
    measurement.set(0, 0, x);
    measurement.set(1, 0, y);
    measurement.set(2, 0, z);

    double[][] tr = { {1, 0, 0, 1, 0, 0},
            {0, 1, 0, 0, 1, 0},
            {0, 0, 1, 0, 0, 1},
            {0, 0, 0, 1, 0, 0},
            {0, 0, 0, 0, 1, 0},
            {0, 0, 0, 0, 0, 1} };
    kalman.setTransition_matrix(new Matrix(tr));
}
```

Listing 11. Partial code for implementing Kalman filter

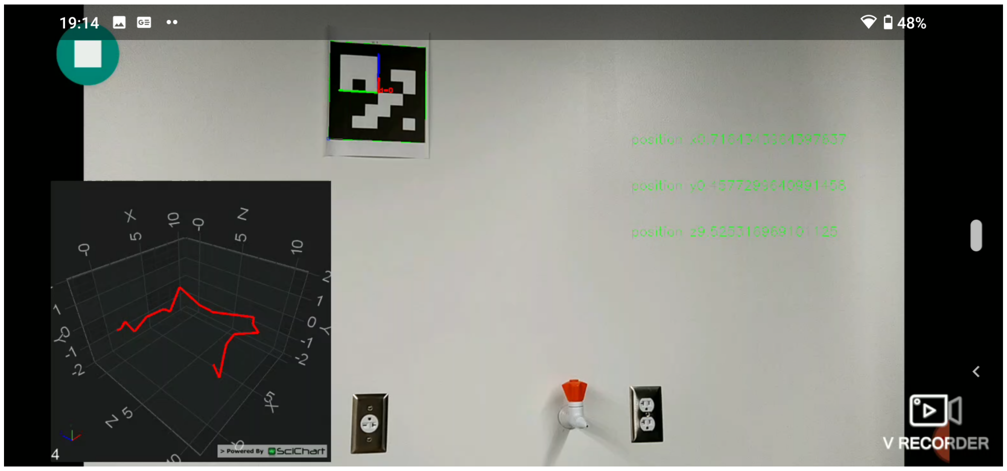
5.10. Trajectory Plotting

With the previously mentioned calculation process, we get the coordinates of the device during the movement. Now we can start plotting the trajectory of the device's movement. In this section, we use a third-party library called SciChart, which provides many drawing tools, and these tools are available for Android, which is very helpful for our drawing process on this system. We first need to obtain its license and write it into the code to use this third-party library. After this step of preparation, we can use the tool classes provided by SciChart to draw the 3D trajectory of the device. The following code snippet shows how to get permission and draw. The drawn 3D trajectory will be displayed on the screen in real-time. Figure23 illustrates how the 3D trajectory looks like. As can be seen, the calculated three coordinate points are displayed in real-time at the top right of the screen.

```java
// Set your license code here
try {
    com.scichart.charting.visuals.SciChartSurface.setRuntimeLicenseKey(
            "LICENSE");
    } catch (Exception e) {
        Log.e("SciChart", "Error when setting the license", e);
    }

private void drawTrajectory() {
    SciChart3DBuilder.init(this);
    final SciChart3DBuilder sciChart3DBuilder = SciChart3DBuilder.instance
   ();

    final DataManager dataManager = DataManager.getInstance();

    final Camera3D camera3D = sciChart3DBuilder.newCamera3D().build();

    final NumericAxis3D xAxis = sciChart3DBuilder.newNumericAxis3D().build
   ();
    final NumericAxis3D yAxis = sciChart3DBuilder.newNumericAxis3D().build
   ();
    final NumericAxis3D zAxis = sciChart3DBuilder.newNumericAxis3D().build
   ();

    xAxis.setVisibleRange(new DoubleRange(-2.0, 11.0));
    yAxis.setVisibleRange(new DoubleRange(-2.0, 2.0));
    zAxis.setVisibleRange(new DoubleRange(-2.0, 11.0));

    xAxis.setAxisTitle("X");
    yAxis.setAxisTitle("Y");
    zAxis.setAxisTitle("Z");

    zAxis.setFlipCoordinates(true);

    final XyzDataSeries3D<Double, Double, Double> xyzDataSeries3D = new
   XyzDataSeries3D<>(Double.class, Double.class, Double.class);
    final PointMetadataProvider3D metadataProvider = new
   PointMetadataProvider3D();

    final List<PointMetadataProvider3D.PointMetadata3D> medatata =
   metadataProvider.metadata;

    for (int i = 0; i < xCoordinates.size(); i++) {
        final double x = xCoordinates.get(i);
        final double y = yCoordinates.get(i);
        final double z = zCoordinates.get(i);

        xyzDataSeries3D.append(x, y, z);

        final int color = Color.RED;
        final float scale = dataManager.getRandomScale();
        medatata.add(new PointMetadataProvider3D.PointMetadata3D(color,
   scale));
    }

    final SpherePointMarker3D pointMarker = sciChart3DBuilder.
   newSpherePointMarker3D()
            .withFill(ColorUtil.Red)
```

```java
                .withSize(0.1f)
                .build();

     final PointLineRenderableSeries3D rs = sciChart3DBuilder.newPointLinesSeries3D()
                .withDataSeries(xyzDataSeries3D)
                .withPointMarker(pointMarker)
                .withStroke(ColorUtil.Green)
                .withStrokeThicknes(2f)
                .withIsAntialiased(false)
                .withIsLineStrips(true)
                .withMetadataProvider(metadataProvider)
                .build();

     UpdateSuspender.using(surface3D, new Runnable() {
         @Override
         public void run() {
                surface3D.setCamera(camera3D);
                surface3D.setXAxis(xAxis);
                surface3D.setYAxis(yAxis);
                surface3D.setZAxis(zAxis);

                surface3D.getRenderableSeries().add(rs);

                surface3D.getChartModifiers().add(sciChart3DBuilder.newModifierGroupWithDefaultModifiers().build());
                }
     });
}
```

Listing 12. Partial code for drawing device's 3D trajectory



```
position x0.71643413661397637
position y0.45772996409891458
position z9.525316969101125
```

Figure 23. Screenshot of running system

5.11. Experimental Data Collection

To be able to analyze the results of the experiment afterward, we also need to collect data from the experiment. We chose to save the experimental data on the computer rather than on the mobile device. To achieve this goal, we used a MySql database. After calculating the 3D coordinates of the device, these data are also stored in the database in real-time. This operation requirement is that the mobile device and the computer need to be in the same network environment. The following code snippet shows how to store the 3D coordinates in the database. The complete code can be found in the appendices.

```java
new Thread(new Runnable() {
    @Override
    public void run() {
        Connection connection = DBOpenHelper.getConnection();
        String sql = "INSERT INTO vision1_fifteen (x, y, z) VALUES (?,?,?)
    ";
        PreparedStatement preparedStatement;
        try {
            preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setDouble(1, cameraPosition.get(0,0)[0]);
            preparedStatement.setDouble(2, cameraPosition.get(1,0)[0]);
            preparedStatement.setDouble(3, cameraPosition.get(2,0)[0]);
            preparedStatement.execute();
            preparedStatement.close();
            connection.close();
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
}).start();
```

Listing 13. Partial code for data collection

6. Testing

6.1. Overview

In this section, the testing design and the analysis of each testing result are presented. The purpose of the testing is to verify that our proposed approach and the implemented system meet the requirements we mentioned in Section 3. We test the trained neural network using a test set to evaluate the detection accuracy and efficiency of the neural network. Then we perform *fifteen* experiments on the IMU-based localization method, vision-based localization method, and our proposed method, respectively. We also analyze the results of each experiment, thus verifying the superiority of our proposed method.

6.2. Markers Detection with Neural Network

We use 20% of the dataset, i.e., 200 images, as a test set to validate the detection performance of YOLOV3. The test set has 200 files and contains 784 ArUco code markers. Then we use the trained YOLOV3 to detect the test set. The detection results show that out of 784 objects to be detected, 368 were detected incorrectly.

6.2.1. Evaluation Metrics

Based on the above detection results, we can then evaluate of the neural network model we have trained. We use mean Average Precision(mAP) to evaluate our training result. Before that, we need to introduce a few metrics.

1. Precision.
   Precision measures how accurate is model's predictions. i.e. the percentage of model's prediction are correct.
   $$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{54}$$
2. Recall.
   Recall measures how good model find all the positives.
   $$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{55}$$
3. IoU (Intersection over union). IoU measures the overlap between two boundaries. We use that to measure how much model predicted boundary overlaps with ground truth (the real object boundary).
   $$IoU = \frac{AreaOfOverlap}{AreaOfUnion} \tag{56}$$
4. AP (Average precision). AP computes the average precision value for recall value over 0 to 1. The general definition for the AP is finding the area under the precision-recall

curve above. Precision and recall are always between 0 and 1. Therefore, AP falls within 0 and 1 also.

$$AP = \int_0^1 p(r)\,\mathrm{d}r \tag{57}$$

5. mAp. The mAp is the average of AP. Since we only have one class which is the ArUco code, so mAp is equal to AP.

6.2.2. Result

Figure 24, 25 and 26 show the precision, recall and mAP of our trained model, respectively. As can be seen from the results, our model has achieved a 99.87% mAP. This detection accuracy is very high and satisfies our requirements of marker detection for indoor localization.

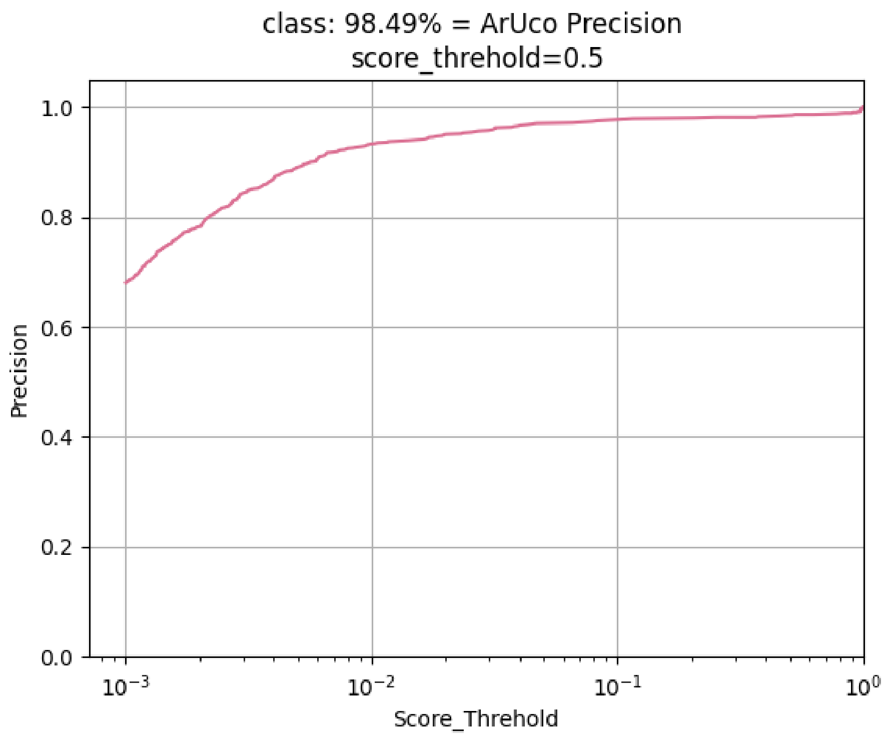


Figure 24. The precision of our trained model

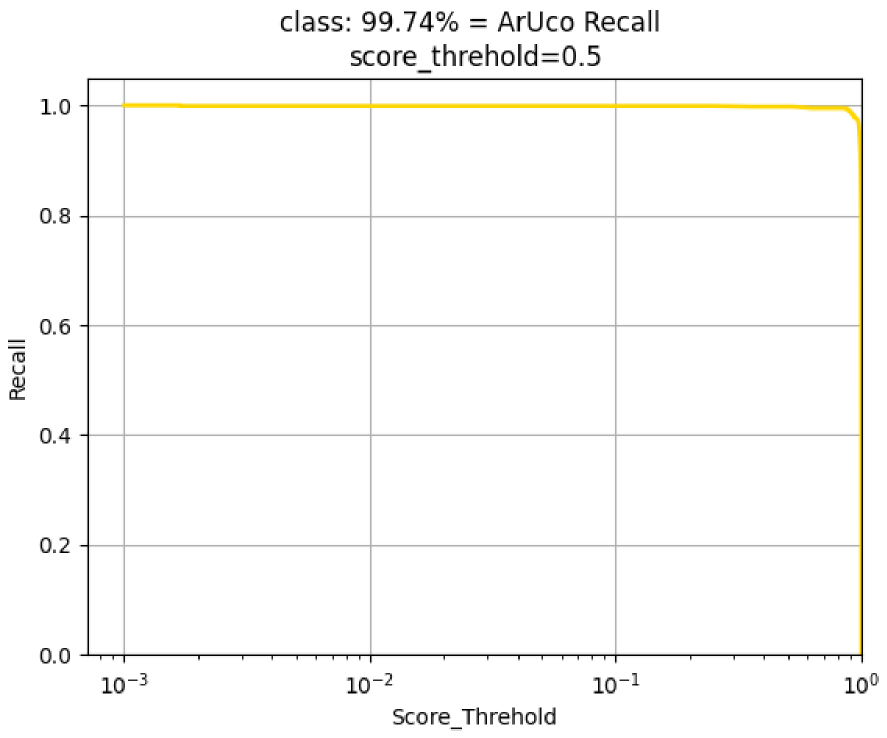


Figure 25. The recall of our trained model

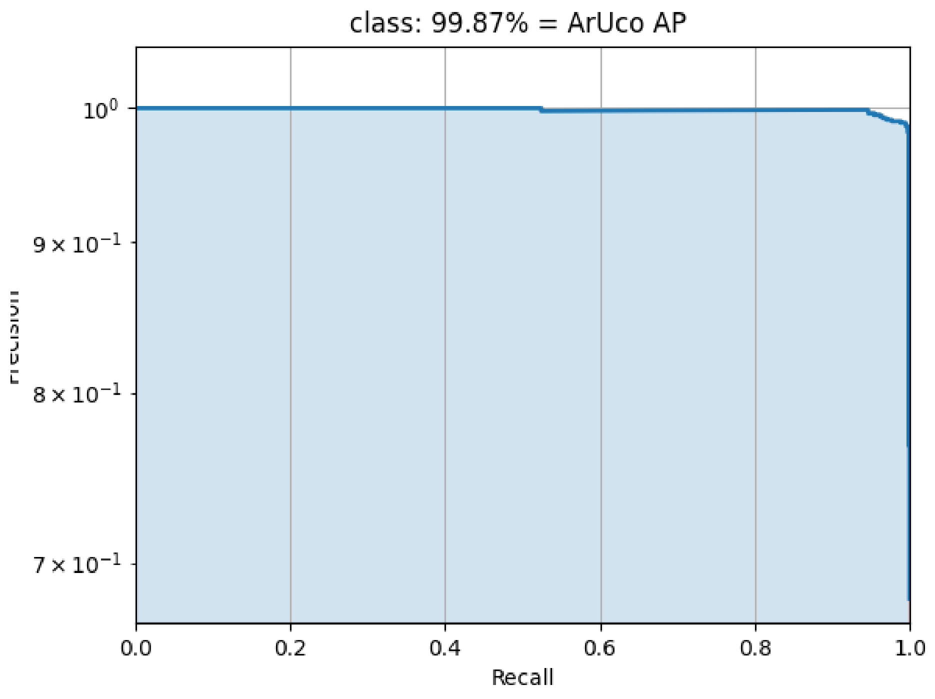


Figure 26. The mAP of our trained model

6.3. Indoor Localization and Tracking System Testing

To demonstrate that our proposed method is superior to both sensor-based localization only and vision-based localization only, we first tested the effect of applying only one of these methods for localization and tracking. Lastly, we tested our complete system, i.e., applying the IMU to support visual localization.

Our experimental setup is as follows:

- Location: The laboratory in the Prairie Springs Science Center
- Device: Nokia 7.2
- Measuring tool: Tape measure
- Conditions: Sufficient and insufficient light; shade and no shade on markers

We choose a point in the lab as the origin of the world coordinates, and then affix ArUco markers at different heights and on different surfaces. Each marker's location information was recorded to observe the difference between the system test results and the actual data results. Figure 27 shows the environment of the laboratory. We take the center point of the first marker as the origin of the world coordinate system. The laboratory's floor plan and the actual walking route during the experiment is shown in Figure 28. The small black squares in the figure show the approximate location of the markers in the laboratory.

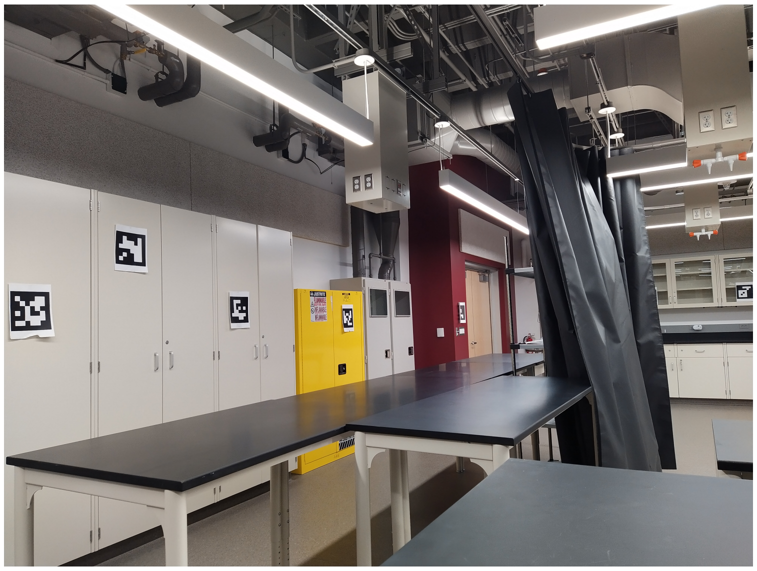


Figure 27. The laboratory's environment

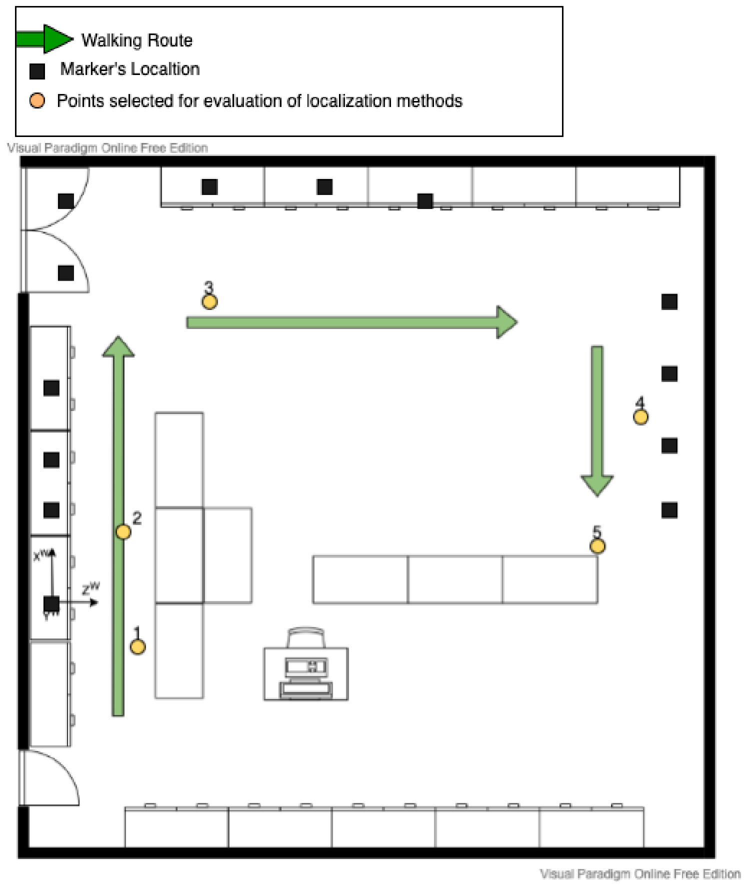

Walking Route
Marker's Localtion
Points selected for evaluation of localization methods
Visual Paradigm Online Free Edition
1
2
3
4
5
x^w
z^w
y^w
Visual Paradigm Online Free Edition

Figure 28. The laboratory's floor plans and actual walking route

In our indoor localization and tracking experiment, fifteen trails are conducted. Figure 29 is the actual 3D moving trajectory. The actual path shows the route of our experiment. The true path contains two corners, as well as a movement in the vertical direction. The maximum distance of motion in the x-axis direction reached 7 meters, and the maximum length of motion in the z-axis direction was 10 meters.

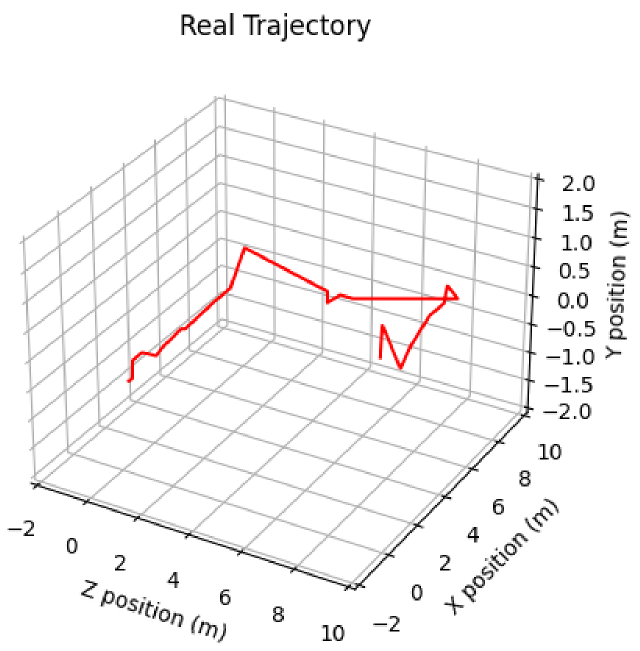


Figure 29. The real trajectory

6.3.1. Testing of IMU-based Indoor Tracking

We first conduct *fifteen* experiments on the IMU-based indoor localization method to evaluate the effectiveness of this method. Figure 30 displays the results of the sensor fusion-based method via fifteen experiments. Figure 31 shows an example from those experiments on the sensor-based localization method to localize and track mobile devices. It can be seen from the figure that the sensor-based approach, which is based on the IMU, has a significant drift in obtaining the movement trajectory. The reason for such a significant error in the IMU-based method is that the interference of the gravitational acceleration in the vertical direction cannot be completely eliminated, and residuals exist. Moreover, the double integration of acceleration leads to the accumulation of errors. Another reason is the drift of the sensor during the measurement. Specifically, when a moving device goes from rest to motion and back to rest, the inertia sensor readings are not zero.

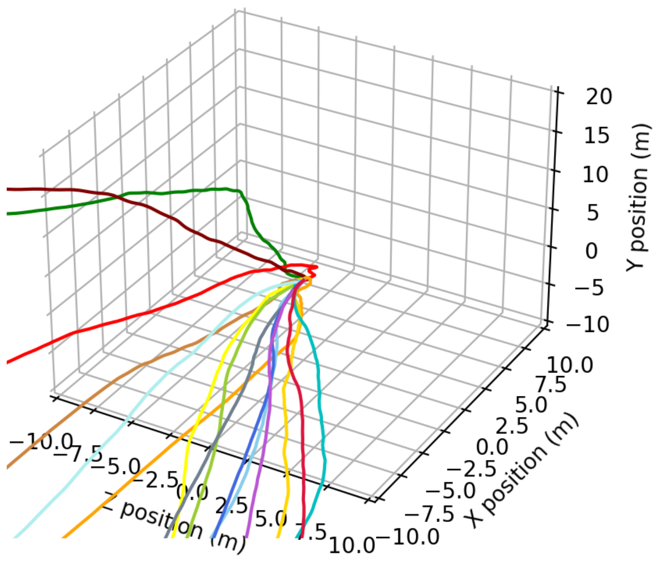


Figure 30. Fifteen trajectories obtained by sensor fusion-based method

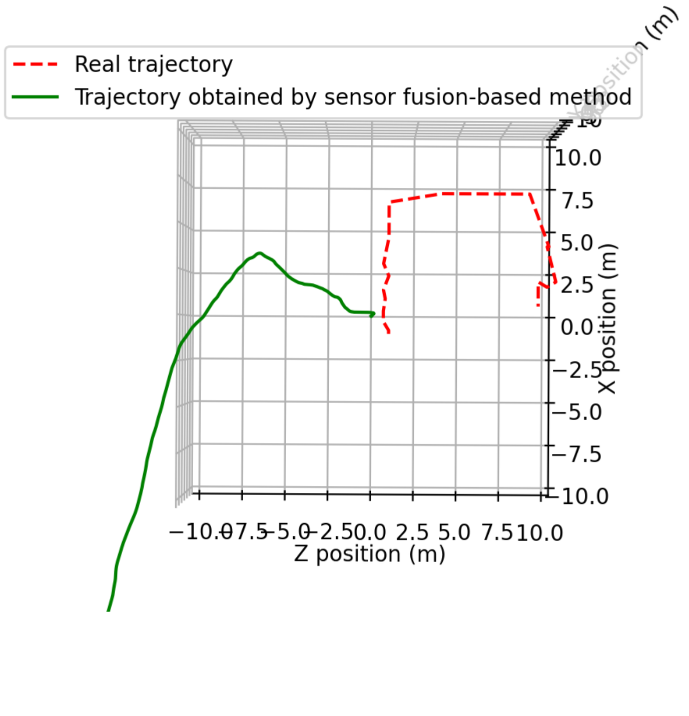


Figure 31. Comparison of real trajectory and trajectory obtained by sensor fusion-based method

6.3.2. Testing of the Vision-based Tracking

Our system still uses mainly vision-based localization methods when markers can be detected. This part of the experiment tests the effectiveness of our proposed method of using neural networks to aid visual localization.

A. PnP Algorithm Comparison

We need to verify that the EPnP algorithm is superior to other PnP algorithms. Therefore, we designed this experiment to compare the difference in accuracy and speed of computation between EPnP, P3P and Iterative algorithms.

In this experiment, the origin of our world coordinate system is the center of the marker, with the X-axis pointing to the right, the Y-axis pointing up, and the Z-axis facing outward perpendicular to the wall. We took pictures of different markers from different angles (Front, below, above, left and right side) and different distance (0.6, 1.2, 1.8, 2.4 meters), and noted the camera's position at the time the picture was taken. Then, We use three different methods - EPNP, P3P, and Iterative - to calculate the camera position in world coordinates. Therefore, the camera had a total of 20 different positions, and we took 30 images for each of these different positions and then performed *thirty* experiments for each position to calculate the average error. Figures 32 to 36 show the error comparison of different PnP algorithms for calculating mobile devices at different angles and different distances.

Figure 32 compares the average error of different algorithms for estimating the camera position at different distances when the mobile device is located directly in front of the marker. The Iterative and P3P algorithms perform similarly, with the EPnP algorithm showing a clear advantage. The errors of all three algorithms increase with increasing distance.

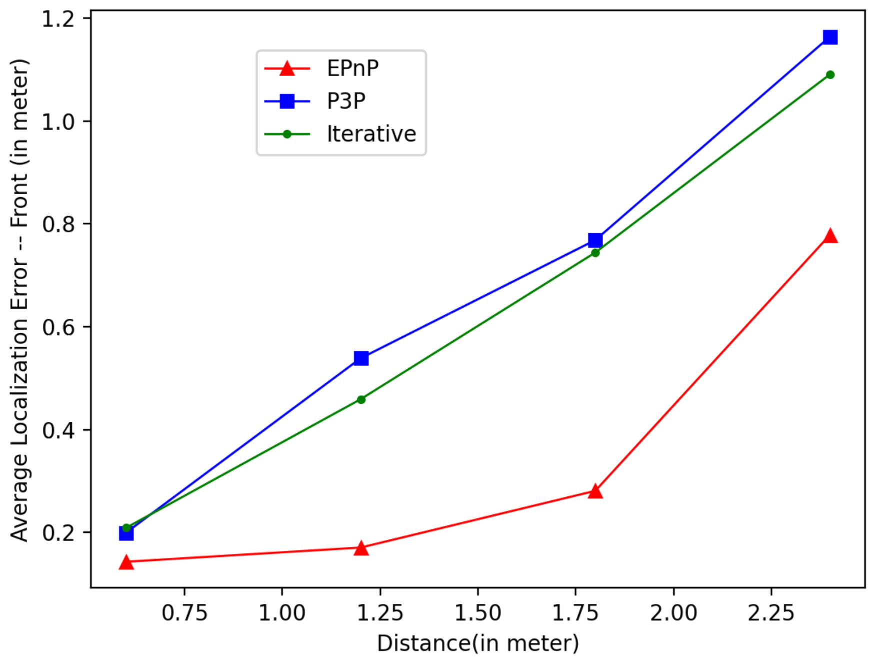


Figure 32. Comparison of different algorithm's average localization error – Front

Figure 33 shows the performance of the different algorithms when the camera is located directly below the marker. The error of the three algorithms still increases with distance, and the performance of the three algorithms is very similar. However, the EPnP algorithm still has a slight advantage.

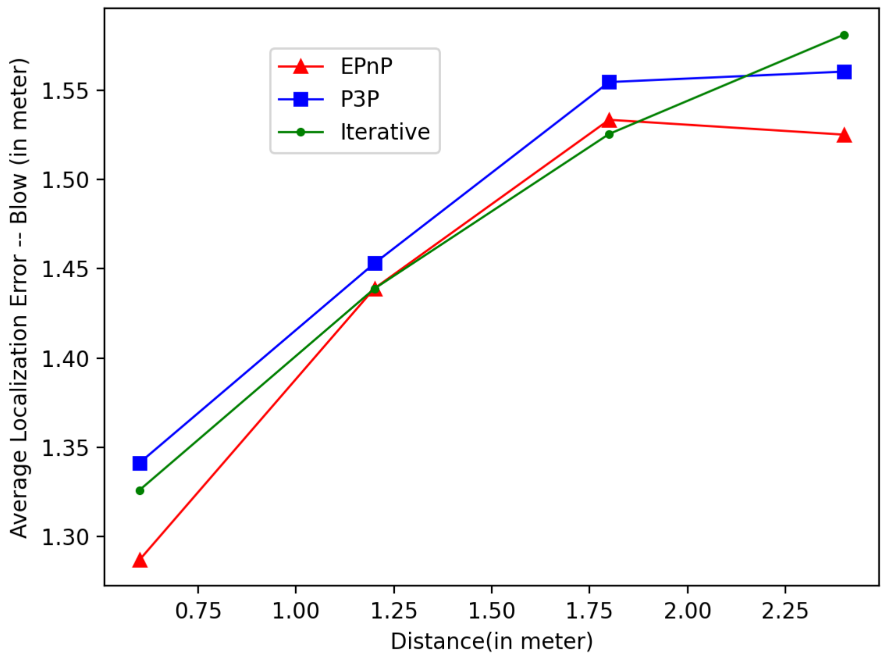


Figure 33. Comparison of different algorithm's average localization error – Blow

Figures 34 and 35 show the comparison of the errors in estimating the camera position by different algorithms when the marker is located directly above the camera and to the left of the marker, respectively. The errors of the P3P and Iterative algorithms are very similar when estimating these two different angles, and the error curves almost overlap. Again, the computed error is positively correlated with the distance. EPnP algorithm has an obvious advantage over the other two algorithms.

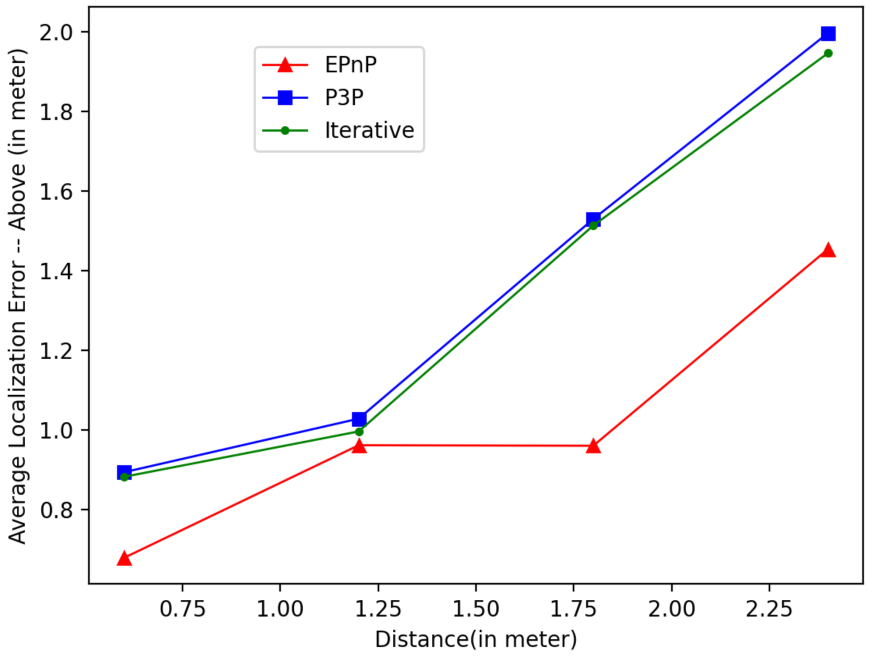


Figure 34. Comparison of different algorithm's average localization error – Above

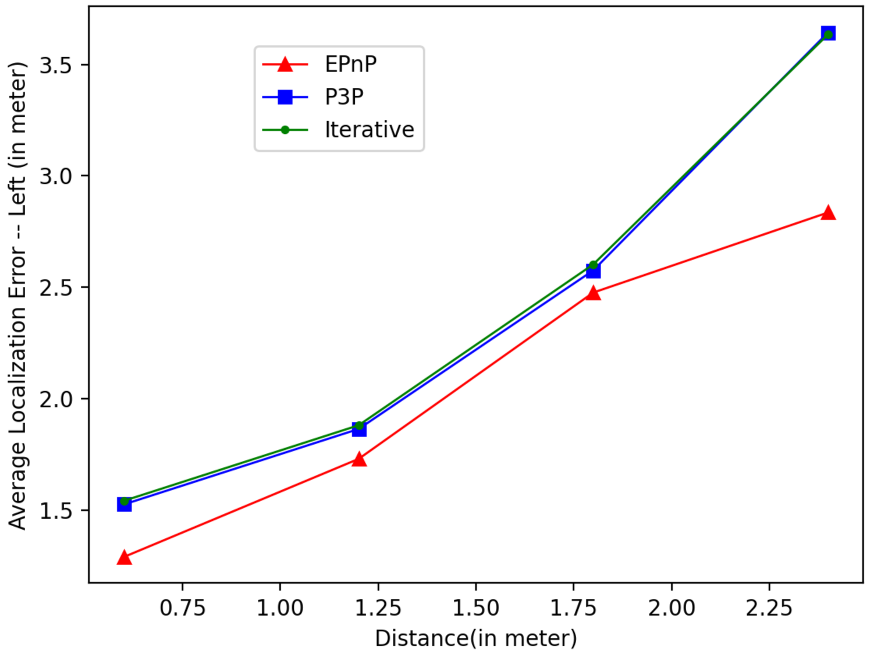


Figure 35. Comparison of different algorithm's average localization error – Left

Figure 36 compares the performance of different algorithms when the camera is on the right side of the marker. The difference between here and the previous one is that the EPnP algorithm is not significantly affected by the distance.

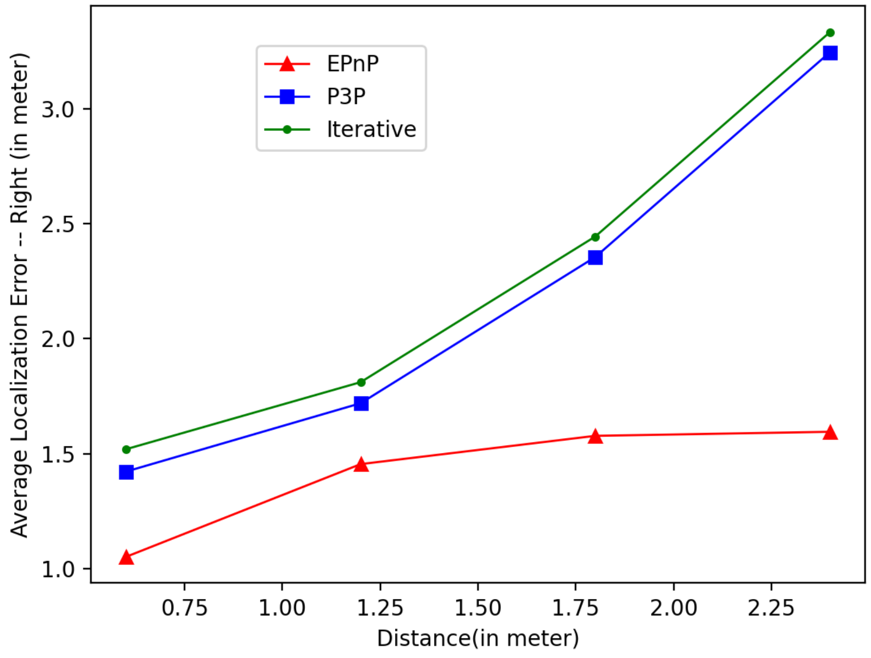


Figure 36. Comparison of different algorithm's average localization error – Right

As can be seen from this experimental results, the computational error of each algorithm increases with the distance. The reason is that when the marker is farther away from the camera, the smaller the imaging area on the image. The x, y coordinate information in the physical world becomes a minimal number of pixels in the image. This causes an error in the z-axis position. The closer the marker is to the camera, the larger the imaging area and the greater the accuracy. In addition, the lighting condition at the time the photo was taken can also affect the calculation results. In general, the EPnP algorithm has the slightest error, and the superiority of the EPnP algorithm is better demonstrated when the distance is farther.

Besides, we also performed performance tests on the three algorithms. We executed the same program 1000 times with each of the three algorithms to obtain the running time of the algorithms. Table 2 shows the results of the three different algorithms. From the results, it is clear that the EPnP algorithm is the fastest, followed by the P3P algorithm. The iterative method is the slowest.

Algorithm	Time
EPnP	223.658ms
P3P	657.869ms
Iterative	1110.194ms

Table 2. Performance comparison of different PnP algorithms

B. Vision-based Localization and Tracking Result

The second part of the experiment is for the vision-based localization method. Similarly, this part of the experiment was repeated *fifteen* times. Figure 37 shows the trajectories obtained by vision-based localization method. Figure 38 shows the comparison between the real trajectory and the trajectory obtained from one of the experiments. As can be seen from the figures, the vision-based approach provided trajectories are not smooth as it calculates the absolute positions of the mobile device. This is because vision-based localization methods rely on markers as reference positions. When there is no marker captured in FoV, no camera pose and position could be estimated. In this case, there will be a gap in the trajectory, positioning will be continued once at least one marker is detected. Therefore, the estimated movement trajectory moves suddenly from the previous position to the current position.

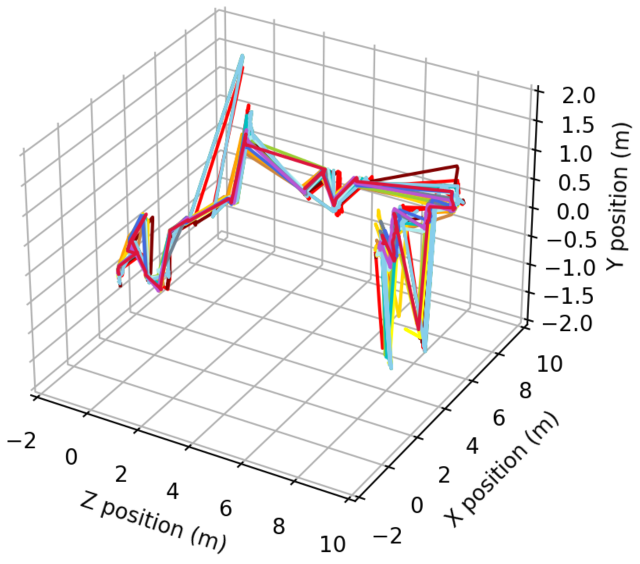


Figure 37. Fifteen trajectories obtained by vision-based method

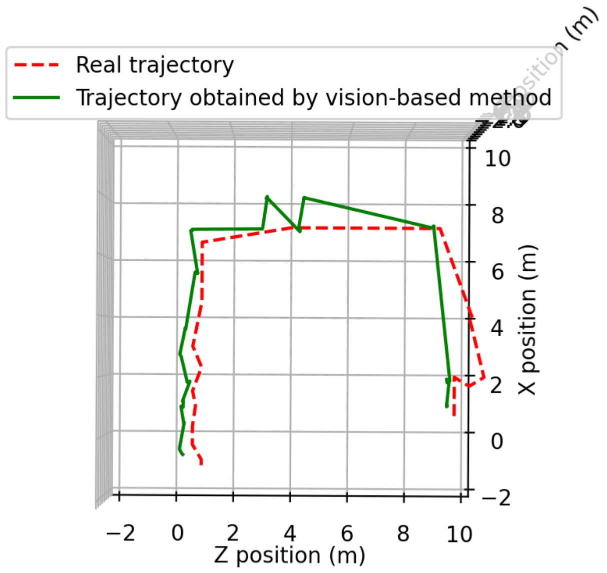


Figure 38. Comparison of real trajectory and trajectory obtained by vision-based method

6.3.3. Testing of Our Proposed Method

The last part of the experiment is to evaluate our proposed approach and test our system's reliability and effectiveness. The experiment has also been repeated *fifteen* times. The results obtained by our system are shown in Figure 39. Figure 40 shows the comparison of one of the paths obtained by our system with the real path in the experiment. From the figure 40, we find that the mobile phone's movement trajectory obtained through our system is very close to the actual trajectory. Moreover, there are no large jumps in the obtained trajectories, meaning that sensor-based localization successfully fills the gap when vision-based localization cannot be used. These fifteen experiments demonstrate that our proposed method combines the advantages of vision-based and sensor-based localization and compensates each other for the disadvantages of these two methods, yielding good results.

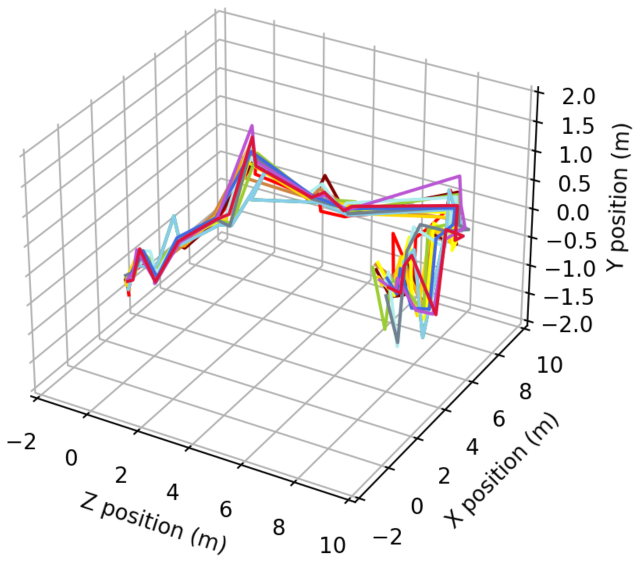


Figure 39. Fifteen trajectories obtained by our system

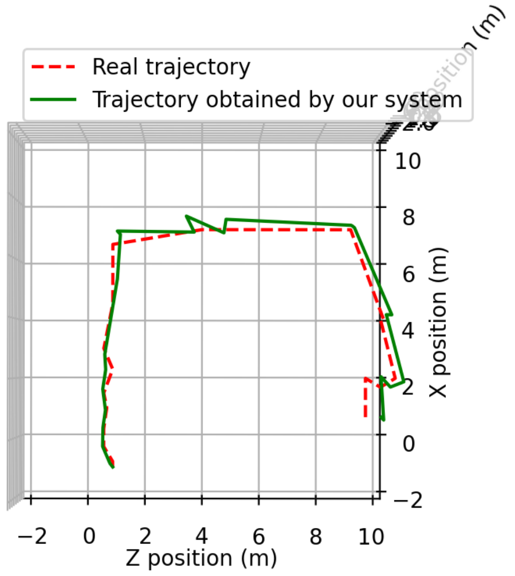


Figure 40. Comparison of real trajectory and trajectory obtained by our system

Table 3 shows the example of location errors evaluated at the selected positions (as indicated in figure 28). The yellow dots in the figure 28 represent these selected points. The location error comparison of our proposed method, vision-based method, and sensor fusion-based method is shown in table 4. On average, our proposed method improves 66.4%, 65%, and 61.6% in the x-axis, y-axis, and z-axis, respectively, with respect to the vision-based method. Furthermore, our proposed method improves much more for the sensor-based method. The improvement is 93.6%, 80% and 84.4% in x-axis, y-axis and z-axis, respectively.

No.	**Real**	**Our system**	**Vision-based**	**Sensor fusion-based**
1	-1	-1.4011	-1.6379	-8.9199
	-0.15	-0.5655	1.0470	-2.2626
	1	1.3943	1.3849	-3.8799
2	1	1.3202	2.0098	-6.4524
	-0.2	-0.5938	1.6148	-2.3476
	0.8	1.1437	1.3625	-2.8963
3	7	7.3121	7.8599	0.3548
	0	0.4543	1.1423	-1.3658
	4	4.1368	5.2847	1.8462
4	4	4.4975	2.9463	-3.4625
	1.2	1.6831	2.2486	-2.3746
	10	10.4304	10.2109	0.4625
5	0.6	1.1250	2.1186	-7.3624
	1	1.3554	2.2250	-2.4762
	9.5	10.3051	10.1139	5.7264

Table 3. Examples of localization using different methods (unit in meter)

No.	**Our system error**	**Vision-based error**	**Sensor fusion-based error**	**Impr**[1]	**Impr**[2]
1	0.4011	1.3621	7.9199	71%	95%
	0.4155	1.1030	2.1126	69%	80%
	0.4057	1.6159	4.8799	75%	92%
2	0.4798	1.0980	7.4524	56%	94%
	0.4062	1.8148	2.1476	78%	81%
	0.4675	1.4375	3.6963	67%	87%
3	0.4897	1.1401	6.6452	57%	93%
	0.4543	1.1423	1.3658	60%	66%
	0.6632	1.2847	2.1574	48%	69%
4	0.4975	3.0537	7.4625	83%	93%
	0.4831	1.0486	3.5746	54%	86%
	0.4304	1.7810	9.5375	76%	95%
5	0.5250	1.5186	7.9624	65%	93%
	0.4446	1.2250	3.4762	64%	87%
	0.8051	1.3864	3.7736	42%	79%

1 Improvement of our proposed method over vision-based method

2 Improvement of our proposed method over sensor-based method

Table 4. Examples of localization error comparison (unit in meter)

The tracking errors of these fifteen experiments on the three axis are shown in figure 41, figure 42, and figure 43, respectively. The experimental results show that IMU-based indoor localization methods have the worst performance. In the fifteen experiments we repeatedly performed, the average error of this method was around 8 meters. Vision-based localization methods perform much better, with an average error of less than 1 meter. Our system performed the best, with the highest accuracy in localization, within an average error of 0.5 meters. The error in the mobile phone movement trajectory obtained by our system is relatively large on the Y-axis compared to the other two axes. The reason for this phenomenon may be that the markers are closer together in the vertical direction, causing duplication of the detection. Overall, our system outperforms both vision-based and IMU-based localization methods, and it meets the accuracy requirements for indoor localization.

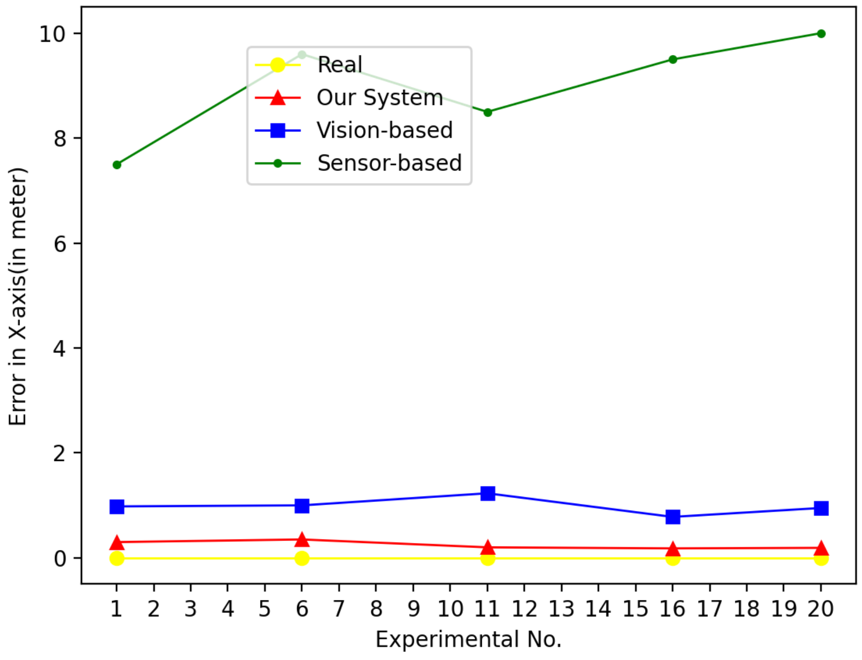


Figure 41. Errors in the X-axis for different localization methods

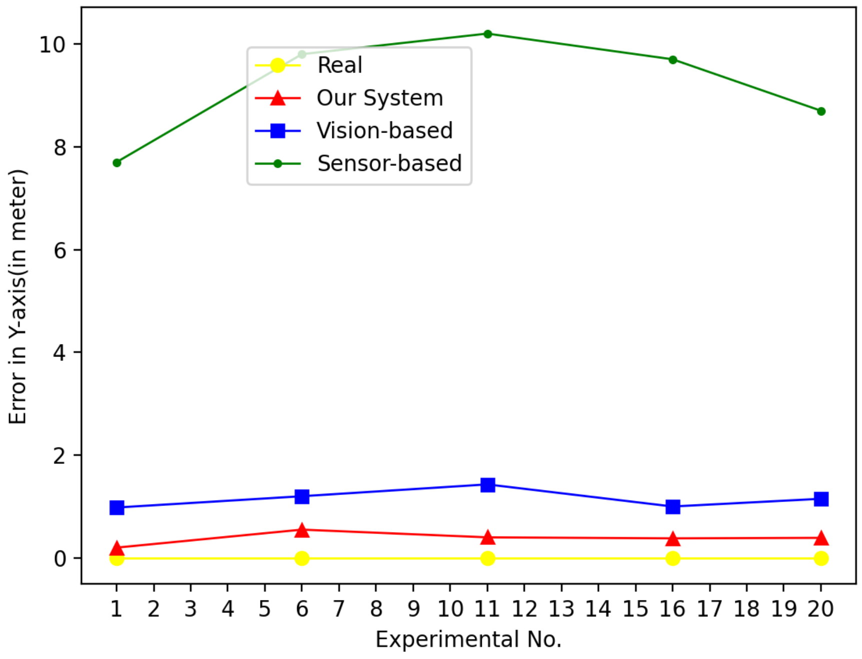


Figure 42. Errors in the Y-axis for different localization methods

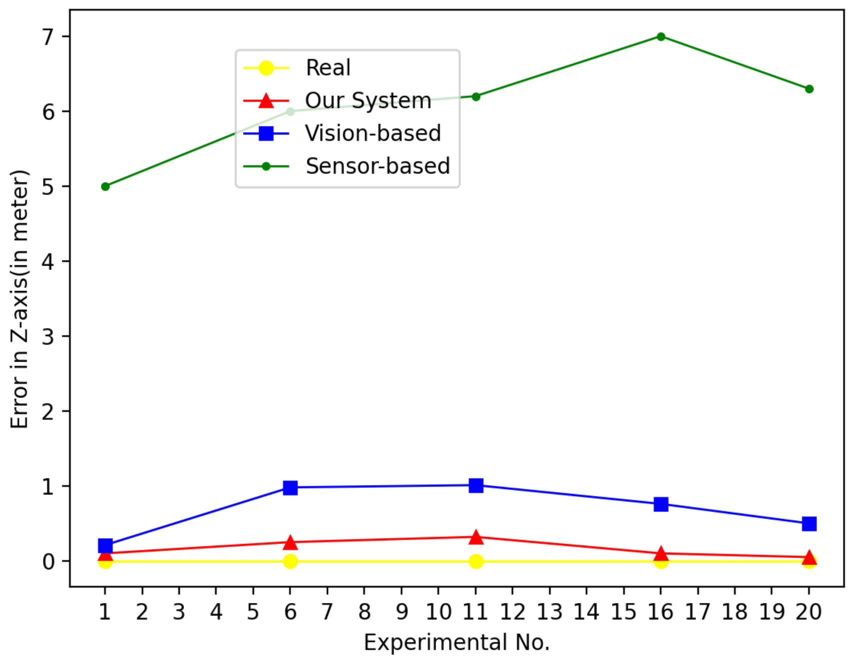


Figure 43. Errors in the Z-axis for different localization methods

7. Conclusion and Future Works

Our proposed neural network-based indoor localization method improves stability of indoor tracking system. It is improved by introducing inertial sensors to assist vision-based localization. Compared to vision-based localization without the assistant of inertial sensors, our system avoids the inability to localize due to missing markers. The proposed method does not rely on any expensive depth camera and can be easily planted to a mobile device. We evaluate and validate our method with the prototype implementation on the smartphone platform. The experimental results show that the system has strong robustness to the complex indoor environment, strong anti-interference ability, high accuracy, and fast processing speed, which meets the demand for indoor localization. The neural network model we trained explicitly for detecting ArUco code has a breakneck detection speed, taking only 0.164 seconds to detect a single frame. Overall, The proposed method demonstrates a tracking accuracy of under 0.5 meters.

In terms of future work, we plan to increase further the number and diversity of datasets, which will significantly improve the accuracy of the neural network in detecting markers. This improves the accuracy of vision-based location detection of markers, which in turn improves the accuracy of our system's location. We also plan to include hardware devices, such as depth cameras, in the approach. This will allow our visual localization method to no longer rely on markers and use objects already present in the indoor environment for localization.

8. References

[1] Elliott Kaplan and Christopher Hegarty. *Understanding GPS: principles and applications*. Artech house, 2005.

[2] Ahmed El-Rabbany. *Introduction to GPS: the global positioning system*. Artech house, 2002.

[3] Thomas Lindner, Lothar Fritsch, Kilian Plank, and Kai Rannenberg. Exploitation of public and private wifi coverage for new business models. In *Building the E-Service Society*, pages 131–148. Springer, 2004.

[4] George E Violettas, Tryfon L Theodorou, and Christos K Georgiadis. Netargus: An snmp monitor & wi-fi positioning, 3-tier application suite. In *2009 Fifth International Conference on Wireless and Mobile Communications*, pages 346–351. IEEE, 2009.

[5] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, and Laurie Cuthbert. Bluetooth positioning using rssi and triangulation methods. In *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, pages 837–842. IEEE, 2013.

[6] Peter Ayuba, Tella Yohanna, and Gabriel Lazarus Dams. An overview of indoor localization technologies and applications. 2015.

[7] Jayakanth Kunhoth, AbdelGhani Karkar, Somaya Al-Maadeed, and Abdulla Al-Ali. Indoor positioning and wayfinding systems: a survey. *Human-centric Computing and Information Sciences*, 10:1–41, 2020.

[8] GOOGLE MAPS. Google indoor maps. Website, 2021. `https://www.google.com/maps/about/partners/indoormaps/`.

[9] Faheem Zafari, Athanasios Gkelias, and Kin K Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599, 2019.

[10] Priya Roy and Chandreyee Chowdhury. A survey of machine learning techniques for indoor localization and navigation systems. *Journal of Intelligent & Robotic Systems*, 101(3):1–34, 2021.

[11] Nathan Piasco, Désiré Sidibé, Cédric Demonceaux, and Valérie Gouet-Brunet. A survey on visual-based localization: On the benefit of heterogeneous data. *Pattern Recognition*, 74:90–109, 2018.

[12] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[13] Maria V Valueva, NN Nagornov, Pave A Lyakhov, Georgiy V Valuev, and Nikolay I Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020.

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[16] Scott W Ambler. Agile model driven development (amdd). *XOOTIC MAGAZINE, February*, 2007.

[17] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[18] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[19] A Alan B Pritsker. *Introduction to Simulation and SLAM II*. Halsted Press, 1984.

[20] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[21] Wikipedia contributors. Conversion between quaternions and euler angles — Wikipedia, the free encyclopedia, 2021. [Online; accessed 12-April-2021].

[22] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[23] NOKIA. Smartphones nokia 7.2. Website, 2019. `https://www.nokia.com/phones/en_us/nokia-7-2?gclid=CjwKCAjw7J6EBhBDEiwA5UUM2sKQg0Vj6UsGLhOxGSwXb-U4v0_6-CjoCRMoe46zOPWwXo2RMUqnIxoCe0wQAvD_BwE&gclsrc=aw.ds`.

[24] Tzutalin. Labelimg. `https://github.com/tzutalin/labelImg`, 2015.

[25] Yueping Li and Nan Jiang. Comparison and evaluation of pnp algorithms of monocular vision. In *AOPC 2015: Image Processing and Analysis*, volume 9675, page 967528. International Society for Optics and Photonics, 2015.

[26] Danying Hu, Daniel DeTone, and Tomasz Malisiewicz. Deep charuco: Dark charuco marker pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8436–8444, 2019.

[27] Tzu-Han Chou, Chuan-Sheng Ho, and Yan-Fu Kuo. Qr code detection using convolutional neural networks. In *2015 International conference on advanced robotics and intelligent systems (ARIS)*, pages 1–5. IEEE, 2015.

[28] Tim Kulich. Indoor navigation using vision-based localization and augmented reality, 2019.

[29] Mostafa Elgendy, Tibor Guzsvinecz, and Cecilia Sik-Lanyi. Identification of markers in challenging conditions for people with visual impairment using convolutional neural network. *Applied Sciences*, 9(23):5110, 2019.

[30] Yihong Wu, Fulin Tang, and Heping Li. Image-based camera localization: an overview. *Visual Computing for Industry, Biomedicine, and Art*, 1(1):1–13, 2018.

[31] Hajime Taira, Masatoshi Okutomi, Torsten Sattler, Mircea Cimpoi, Marc Pollefeys, Josef Sivic, Tomas Pajdla, and Akihiko Torii. Inloc: Indoor visual localization with dense matching and view synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7199–7209, 2018.

[32] Federico Boniardi, Abhinav Valada, Rohit Mohan, Tim Caselitz, and Wolfram Burgard. Robot localization in floor plans using a room layout edge extraction network. *arXiv preprint arXiv:1903.01804*, 2019.

[33] Meng-Jiun Chiou, Zhenguang Liu, Yifang Yin, An-An Liu, and Roger Zimmermann. Zero-shot multi-view indoor localization via graph location networks. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 3431–3440, 2020.

[34] Maximilian Kloock, Patrick Scheffe, Isabelle Tülleners, Janis Maczijewski, Stefan Kowalewski, and Bassam Alrifaee. Vision-based real-time indoor positioning system for multiple vehicles. *arXiv preprint arXiv:2002.05755*, 2020.

[35] Boxin Zhao, Tianjiang Hu, Daibing Zhang, Lincheng Shen, Zhaowei Ma, and Weiwei Kong. 2d monocular visual odometry using mobile-phone sensors. In *2015 34th Chinese Control Conference (CCC)*, pages 5919–5924. IEEE, 2015.

[36] Chaobin Wang, Huawei Liang, Xinli Geng, and Maofei Zhu. Multi-sensor fusion method using kalman filter to improve localization accuracy based on android smart phone. In *2014 IEEE International Conference on Vehicular Electronics and Safety*, pages 180–184. IEEE, 2014.

[37] Daniel Kold Hansen, Kamal Nasrollahi, Christoffer Bøgelund Rasmussen, and Thomas B Moeslund. Real-time barcode detection and classification using deep learning. In *International Joint Conference on Computational Intelligence*, pages 321–327. SCITEPRESS Digital Library, 2017.

9. Appendices

```java
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
    Imgproc.cvtColor(inputFrame.rgba(), rgb, Imgproc.COLOR_RGBA2RGB);
    gray=inputFrame.gray();

    ids = new MatOfInt();
    corners.clear();

    Aruco.detectMarkers(gray, dictionary, corners, ids, parameters);

    if(corners.size()>0){
        Aruco.drawDetectedMarkers(rgb, corners, ids);

        rvecs=new Mat();
        tvecs=new Mat();

        Aruco.estimatePoseSingleMarkers(corners,0.025f,cameraMatrix,distCoeffs,rvecs,tvecs);

        for(int i=0; i<ids.toArray().length; i++){
            transformModel(tvecs.row(0),rvecs.row(0));
            Aruco.drawAxis(rgb, cameraMatrix, distCoeffs, rvecs.row(i), tvecs.row(i),0.01f);
        }

        MatOfPoint2f mImagePoints = Mat2MatOfPoint2f(corners.get(0));
        Mat cameraPosition = getCameraPose(mImagePoints, ids.toArray()[0]);

        if (cur-prev > 3*1000) {
            xCoordinates.add(cameraPosition.get(0,0)[0]);
            yCoordinates.add(cameraPosition.get(1,0)[0]);
            zCoordinates.add(cameraPosition.get(2,0)[0]);

            prev = cur;

            new Thread(new Runnable() {
                @Override
                public void run() {
                    Connection connection = DBOpenHelper.getConnection();
                    String sql = "INSERT INTO vision1_fifteen (x, y, z) VALUES (?,?,?)";
                    PreparedStatement preparedStatement;
                    try {
                        preparedStatement = connection.prepareStatement(sql);
                        preparedStatement.setDouble(1, cameraPosition.get(0,0)[0]);
                        preparedStatement.setDouble(2, cameraPosition.get(1,0)[0]);
                        preparedStatement.setDouble(3, cameraPosition.get
(2,0)[0]);
                        preparedStatement.execute();
                        preparedStatement.close();
                        connection.close();
                    } catch (SQLException throwables) {
                        throwables.printStackTrace();
                    }
                }
            }).start();

        }

        Imgproc.putText(rgb, "position x" + cameraPosition.get(0,0)[0], new Point(rgb.cols()/3*2,rgb.rows()* 0.3), Core.FONT_HERSHEY_SIMPLEX,1.0,new Scalar(0,255,0));
        Imgproc.putText(rgb, "position y" + cameraPosition.get(1,0)[0], new Point(rgb.cols()/3*2,rgb.rows()* 0.4), Core.FONT_HERSHEY_SIMPLEX,1.0,new Scalar(0,255,0));
        Imgproc.putText(rgb, "position z" + cameraPosition.get(2,0)[0], new Point(rgb.cols()/3*2,rgb.rows()* 0.5), Core.FONT_HERSHEY_SIMPLEX,1.0,new Scalar(0,255,0));
        drawTrajectory();

        cur = System.currentTimeMillis();

    }
    return rgb;
}
```

Listing 14. Code for detecting marker and data collection

```
public List<Corners> markerDetection(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {

    List<Corners> detectionResult = new ArrayList<>();

    classNames = readLabels("labels.txt", this);
    for(int i=0; i<classNames.size(); i++) {
        colors.add(randomColor());
    }
    // Loading network
    String modelConfiguration = getAssetsFile("yolov3_aruco.cfg", this);
    String modelWeights = getAssetsFile("yolov3_aruco_final.weights", this);
    yolo = Dnn.readNetFromDarknet(modelConfiguration, modelWeights);

    Mat frame = inputFrame.rgba();
    Imgproc.cvtColor(frame, frame, Imgproc.COLOR_RGB2RGBA);
    Size frame_size = new Size(416, 416);
    Scalar mean = new Scalar(127.5);

    Mat blob = Dnn.blobFromImage(frame, 1.0/255.0, frame_size, mean, true, false);
    yolo.setInput(blob);
```

```java

        List<Mat> result = new ArrayList<>();
        List<String> outBlobNames = yolo.getUnconnectedOutLayersNames();

        yolo.forward(result, outBlobNames);
        float confidenceThreshold = 0.5f;

        for (int i = 0; i < result.size(); ++i) {
            Mat level = result.get(i);
            for (int j = 0; j < level.rows(); ++j) {
                Mat row = level.row(j);
                Mat scores = row.colRange(5, level.cols());
                Core.MinMaxLocResult mm = Core.minMaxLoc(scores);
                float confidence = (float) mm.maxVal;
                Point classIdPoint = mm.maxLoc;
                if (confidence > confidenceThreshold) {

                    int centerX = (int) (row.get(0, 0)[0] * frame.cols());
                    int centerY = (int) (row.get(0, 1)[0] * frame.rows());
                    int width = (int) (row.get(0, 2)[0] * frame.cols());
                    int height = (int) (row.get(0, 3)[0] * frame.rows());

                    int left = (int) (centerX - width * 0.5);
                    int top =(int)(centerY - height * 0.5);
                    int right =(int)(centerX + width * 0.5);
                    int bottom =(int)(centerY + height * 0.5);

                    Point left_top = new Point(left, top);
                    Point right_bottom = new Point(right, bottom);
                    Point right_top = new Point(left + width, top);
                    Point left_bottom = new Point(right - width, bottom);
                    Point label_left_top = new Point(left, top-5);
                    DecimalFormat df = new DecimalFormat("#.##");

                    int class_id = (int) classIdPoint.x;
                    String label= classNames.get(class_id) + ": " + df.format(confidence);
                    Scalar color= colors.get(class_id);

                    Imgproc.rectangle(frame, left_top,right_bottom , color, 3, 2);
                    Imgproc.putText(frame, label, label_left_top, Core.FONT_HERSHEY_SIMPLEX, 1, new Scalar(0, 0, 0), 4);
                    Imgproc.putText(frame, label, label_left_top, Core.FONT_HERSHEY_SIMPLEX, 1, new Scalar(255, 255, 255), 2);

                    Corners corners = new Corners();
                    corners.setLeft_top(left_top);
                    corners.setRight_top(right_top);
                    corners.setLeft_bottom(left_bottom);
                    corners.setRight_bottom(right_bottom);
                    detectionResult.add(corners);
                }
            }
```

```
    }
    return detectionResult;
}
```

Listing 15. Code for markers detection by yolov3 in Android