Computer Methods and Programs in Biomedicine 205 (2021) 106112

Contents lists available at ScienceDirect

Computer Methods and Programs in Biomedicine

journal homepage: www.elsevier.com/locate/cmpb

A Novel Marker Detection System for People with Visual Impairment Using the Improved Tiny-YOLOv3 Model

Mostafa Elgendy[a,b,*], Cecilia Sik-Lanyi[c], Arpad Kelemen[d]

[a] Department of Electrical Engineering and Information Systems, University of Pannonia, 8200 Veszprém, Hungary
[b] Department of Computer Science, Faculty of Computers and Artificial Intelligence, Benha University, Benha 13511, Egypt
[c] Department of Electrical Engineering and Information Systems, University of Pannonia, 8200 Veszprém, Hungary
[d] Department of Organizational Systems and Adult Health, University of Maryland, Baltimore, MD, 21201, USA

ARTICLE INFO

Article history:
Received 12 October 2020
Accepted 7 April 2021

Keywords:
Assistive technology
Visually impaired
Indoor navigation
Markers
Deep learning
Tiny-YOLOv3

ABSTRACT

Background and Objective: Daily activities such as shopping and navigating indoors are challenging problems for people with visual impairment. Researchers tried to find different solutions to help people with visual impairment navigate indoors and outdoors.

Methods: We applied deep learning to help visually impaired people navigate indoors using markers. We propose a system to help them detect markers and navigate indoors using an improved Tiny-YOLOv3 model. A dataset was created by collecting marker images from recorded videos and augmenting them using image processing techniques such as rotation transformation, brightness, and blur processing. After training and validating this model, the performance was tested on a testing dataset and on real videos.

Results: The contributions of this paper are: (1) We developed a navigation system to help people with visual impairment navigate indoors using markers; (2) We implemented and tested a deep learning model to detect Aruco markers in different challenging situations using Tiny-YOLOv3; (3) We implemented and compared several modified versions of the original model to improve detection accuracy. The modified Tiny-YOLOv3 model achieved an accuracy of 99.31% in challenging conditions and the original model achieved an accuracy of 96.11 %.

Conclusion: The training and testing results show that the improved Tiny-YOLOv3 models are superior to the original model.

1. Introduction

People with visual impairment (PVI) suffer while performing everyday activities such as navigating indoors or outdoors, identifying objects, avoiding obstacles [1,2] and shopping [3]. Therefore, developing new solutions is very important to help PVI do these activities easily and efficiently and motivate them to interact with the social environment [4,5]. Global positioning systems (GPS) are used to solve the problem of navigating outdoors. GPS uses geostationary satellite signals with an accuracy of up to several meters, sufficient for outdoor navigation. However, indoor navigation is still a big problem that needs an accurate and reliable solution. For example, in a tall building, GPS cannot be used to automatically determine which floor the user is currently on. Furthermore, the satellite signals are attenuated and scattered in tunnels or underground and by roofs, walls and other objects [6,7]. Researches have helped PVI navigate indoors and identify objects through using Computer Vision (CV) [8–10]. A typical CV navigation system uses unique installed tags such as Augmented Reality (AR) markers to help in navigating indoors and recognizing objects [11–13]. It consists of tags installed in place, a database to store tag information, a camera to capture real-time pictures, a processing unit to execute the used techniques, and a two-way communication between the system and PVI to take the input and give feedback to help them reach their destination [14,15]. However, the used markers cannot be identified in many real-life situations due to motion blur or distortion, poor lighting conditions, or too high distances from the camera [16].

In recent years, machine learning algorithms have been used in the field of CV to improve object detection software. The deep

* Corresponding author.
E-mail addresses: mostafa.elgendy@virt.uni-pannon.hu (M. Elgendy), lanyi@almos.uni-pannon.hu (C. Sik-Lanyi), kelemen@umaryland.edu (A. Kelemen).

https://doi.org/10.1016/j.cmpb.2021.106112

convolutional neural network increases the network levels, which makes the network have stronger detection capabilities. Deep learning algorithms for detection can be divided into two categories: two-stage and one-stage. Region-based Convolutional Neural Networks (R-CNNs) [17], Fast R-CNNs [18], and Faster R-CNNs [19] are two-stage algorithms that use a region proposal network to generate regions of interests in the first stage and propagate the region proposals down the pipeline for object classification and bounding-box regression. R-CNNs predict object locations using region proposal algorithms. Features are extracted from each candidate region, fed into CNNs, and finally evaluated by Support Vector Machines (SVMs). R-CNN increases the target detection accuracy, but the efficiency is very low. Faster R-CNN uses the region proposal network method to detect the region of interest in the image. Then, it uses a classifier to classify these regions of interest called bounding boxes. Faster R-CNN improves the detection accuracy, but the detection speed is slow, which is not suitable for real-time applications with high image resolution.

On the other hand, single-stage detectors such as You Only Look Once (YOLO) [20] and Single Shot Detector (SSD) [21] were proposed to improve the detection efficiency to be suitable for real-time applications by treating object detection as a simple regression problem. They take an input image and learn the class probabilities and bounding box coordinates. Such models reach lower accuracy rates but are much faster than two-stage object detectors [22]. YOLO is a CNN specifically designed for making object detection fast, accurate, and suitable for real-time usage. It uses a single convolutional neural network to predict object categories and find their locations [20]. Several versions of the YOLO model were proposed to improve the accuracy without notable effects on speed. YOLOv2 is an improvement of YOLO using higher-resolution feature maps that help the network detect objects of different scales. It also has an added batch normalization on each convolutional layer. Bounding boxes are being predicted by using anchor boxes [23]. YOLOv3 improves previous YOLO versions by using multi-scale detection, a more powerful feature extractor network, and some modifications in the loss function, which allows detecting big and small targets [24].

We choose YOLOv3 because it can balance the performance on accuracy and processing time well; however, the execution time needs to be improved when using it for real-time applications, especially on smartphones. Tiny-YOLOv3 simplified the original YOLOv3 model by reducing the number of the convolutional layers to be suitable for real-time applications without losing much accuracy.

This paper proposes a system to help PVI navigate indoors using improved versions of the Tiny-YOLOv3 model to improve the detection accuracy of the original model. In this system, Aruco markers are installed at the interest points. When markers are detected during navigation, voice feedbacks are given to guide PVI safely to their destination. The main contributions of this article are the following:

- Design of a navigation system to help PVI navigate indoors from any location inside the building to their destination using markers.
- Proposal of a novel model to improve detecting markers in different challenging situations based on Tiny-YOLOv3.
- Several modified versions of the original model were implemented and compared to improve detection accuracy. These models were tested using real test cases. They provided high accuracy and very good performance in detecting markers.

The subsequent parts of this article are structured in the following way. Section 2 reviews the relevant works related to navigation systems using markers and deep learning techniques. Section 3 explains the design of the navigation system, the original Tiny-YOLOv3, and modified versions. Section 4 presents the experimentation carried out and discusses their results. Section 5 draws conclusion and suggests some ideas for future work.

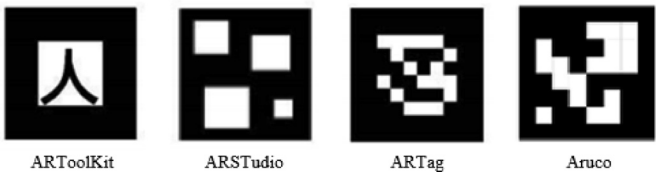


Fig. 1. Sample of different square markers.

2. Related work

In recent years, significant progress was made in the use of deep learning and CV in medical research, such as biomedical images [25], cancer prediction [26,27], object detection [28], and object recognition [29]. Smartphones have become very important because they have various capabilities including processing power, integrated cameras, and various sensors. These emerging and maturing technologies and smartphone capabilities allow researchers to construct new applications to help PVI identify objects [8,30] and safely navigate indoors [31,32]. Researchers aim to improve the quality of these applications by increasing accuracy and minimizing execution time to be suitable for real-time use. In this section, we present some of the common navigation systems using markers and deep learning.

2.1. Square markers

Square markers are square-shaped tags with a thick black border and inner region to represent a code. As shown in Fig. 1, the inner region contains pictures or binary codes represented as grids of black and white regions. In this article, Aruco markers have been used as tags in our system [13].

Al-Khalifa and Al-Razgan proposed an indoor navigation system for PVI using a smartphone and Google Glass [14]. It used a map constructed with a graph to represent interest points and the distances between them. Quick Response (QR) codes are generated and installed for each interest point. To navigate, it finds the shortest path from the current position to the destination and provides PVI with feedback during navigation. This system used Google glasses to detect QR codes which facilitate navigation without using hands to hold the smartphone. However, it fails to identify QR codes in different situations. To work, it needs to download the building maps using a stable internet connection.

Ko and Kim proposed a system to help PVI navigate in unknown environments using QR codes [33]. Location changes during navigation are continuously computed and given to PVI using text to speech. This system records the paths used by PVI during navigation to help them navigate easily during their return journey using the same routes. This system combines different types of feedback such as tactile or voice commands to minimize navigation errors. It also uses colored QR codes to make identifying them easy. However, it fails to detect QR codes in challenging situations such as motion blurring effects and long distances.

Torrado et al. proposed a system for helping PVI navigate unknown and complex environments [34]. The system uses a smartphone application to detect QR codes distributed across the floor. It calculates the distance to these QR codes and guides PVI to their destination. However, it is hard for PVI to capture real-time photos with their smartphone camera.

Elgendy et al. developed a system to help PVI in indoor navigation using markers [35]. In this system, markers were installed

at the main interest points inside a university. After the PVI selects the initial and destination points, the system finds the shortest path to that destination and gives continuous navigation feedbacks to reach it. However, it fails to detect markers in challenging situations such as motion blurring effects.

Delfa et al. proposed a system for indoor navigation using Bluetooth and a smartphone camera [36,37]. It gave a low-level accuracy position estimation using Bluetooth while increasing the accuracy by using the smartphone's camera to detect visual tags in real-time. This system used markers to estimate the PVI position and navigate to the destination. It used markers with a color different from the floor to enhance speed and efficiency. However, the system fails to detect markers from a long distance and cannot detect various markers at the same time.

Khan et al. developed a generic system for navigation using ARToolKit markers [38]. It used a smartphone's camera to recognize markers placed on the ceiling of the building. The user selects their destination and then the application determines the shortest path to the destination based on the first nearest marker. Audio feedback is given to guide PVI to reach their final point. However, this system also faces some limitations. It is difficult for PVI, who are holding the smartphone to point the camera, to detect markers on the ceiling of buildings. Finally, the system is tested only by blindfolded people.

Fusco et al. proposed an indoor localization approach to facilitate wayfinding by using a combination of computer vision and dead reckoning techniques [39]. It used markers to estimate the PVI's location and dead reckoning to track PVI movements when no markers are visible. However, the system is implemented as a logging system and is not suitable for real-time usage. Furthermore, the accuracy of marker recognition needed to be improved as they can be identified only in a small fraction of video frames. The system failed to detect markers if there is any motion blur or rapid walking speed.

Lee et al. proposed an indoor navigation system using markers and augmented reality [40]. It performs hybrid localization by using marker images as well as Inertial Measurement Unit (IMU) data from smartphones. First, an indoor map is prepared to register the positions for indoor places. Then, markers are generated and printed for these registered places. The navigation system is used to help users successfully reach their destinations. However, it cannot detect markers in a crowded environment, as the markers are installed on the floor. Furthermore, it failed to identify markers from long distances.

In addition to marker-based positioning for PVI, various studies using markers have also been conducted in robotics. Zhang et al. proposed a system using QR codes to localize a mobile robot and allow it to easily navigate [41]. A camera is attached to the robot to identify QR codes installed on the ceiling. These codes are used to identify the robot location and the shortest path to the destination is calculated using Dijkstra's algorithm. Based on the detected QR codes, the robot can navigate to the destination. The experimental results showed that the proposed method was effective and allowed the robot to navigate in indoor environments. However, square markers are more accurate than QR codes as they can be detected from a longer distance than a QR code. This system is hard for PVI to use as they need to point the camera to the ceiling. Li et al. proposed an indoor positioning system for mobile robots but they used a newly designed marker instead of QR codes [42].

2.2. Deep Learning

Dash et al. proposed an Augmented Reality (AR) system to help children learn alphabets [43]. The system used a CNN model to detect markers presented in a scene. Then, virtual objects are rendered on the top of the detected markers. To render them correctly, they should put them in front of the camera with the correct position and orientation. This system achieved high accuracy in marker identification. However, it fails to detect markers from a long distance.

Elgendy et al. proposed a system to help PVI in indoor navigation using markers [16]. The identification step was redefined as a classification problem, and a CNN is used to identify markers. Several CV techniques are used to select candidates, and then they are fed to a CNN model to classify if they are markers or not. The system helps PVI navigate indoors using markers. It achieved high accuracy. The use of CV techniques to select the candidate markers takes processing time which needs to be minimized.

Mekhalfi et al. proposed a navigation system using computer-vision technologies [44]. It included a speech recognition module to receive instructions and give voice feedback to PVI. A laser sensor was used to calculate the distance from obstacles. A set of markers and an IMU sensor were used to determine PVI location, and a path planning module was used to calculate a safe path for the user to walk through. They used a portable camera to capture the scene and forward the shots to the navigation or the recognition units. However, the size and weight of the processing unit is a big problem as PVI cannot wear it for a long time – a disadvantage compared to using a smartphone. The average processing time for recognition also needed to be minimized. Lastly, obstacle detection sensors are expensive and not available for common people.

Bazi et al. proposed a navigation system to help PVI recognize multiple objects in images using a multi-label convolutional SVM [45]. It uses a portable camera mounted on a lightweight shield worn by the user to capture images and send it via a USB wire to a laptop processing unit. To identify objects, a set of linear SVMs were used as a filter in each convolutional layer to generate a new set of feature maps. Finally, the outputs are fed again to a linear SVM classifier for carrying out the classification task. However, the size and weight of the processing unit is again a big problem for PVI. Also, it failed to detect markers from longer distances.

Kayukawa et al. proposed a collision-avoidance system for PVI using a camera that is integrated into the suitcase [46]. It used depth images to determine the risk of collision with a blind person using a CNN model for detecting objects while YOLOv2 is used to detect pedestrians using the RGB streams. This system detects individuals efficiently. However, the execution time needed to be minimized for real-time usage.

The main objective in [47] is to develop a detection method for small objects based on YOLOv3. The darknet CNN structure was modified by increasing the convolutional operation in the beginning to improve performance. The proposed method improved the performance of detecting small objects. However, it is not suitable for real-time usage by smartphones.

Tapu et al. proposed a navigational assistant prototype to increase the mobility and safety of PVI [6]. It used CV algorithms and deep CNN to detect, track, and recognize objects in real-time. It modified the YOLO algorithm by adding an object tracking procedure to impute missing information where YOLO is failing. It also introduced a detection and handling strategy to handle object occlusion and object movement or camera drift. The proposed system can process information from the environment and give feedback to PVI to avoid possible collisions. However, it is hard for PVI to carry this system on the back for a long time.

3. Proposed methodology

As navigating inside buildings have many obstacles, markers can be used to identify the exact location of PVI and allow them to navigate to their destinations. This paper proposes a system to help PVI navigate indoors using deep learning and markers. First, the building is configured by installing markers at the interest points

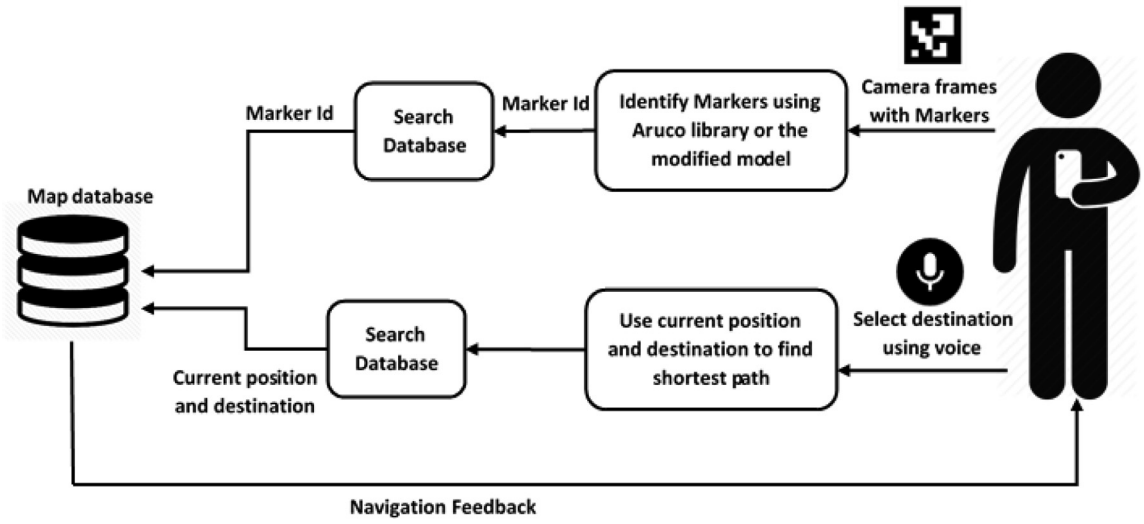


Fig. 2. System architecture that PVI should follow to reach the destination point.

such as laboratory and lecture rooms. Then, an internal map is created using a graph to save the interest points and the relation between them. Nodes in this graph represent interest points while edges are used to connect between these nodes. After configuring the environment and building the internal map, PVI can use it by following the architecture shown in Fig. 2 to reach the destination point. The main parts of this prototype are 1. How to identify markers to find PVI's location. 2. How to navigate from the starting point to the destination using the markers.

3.1. Navigation system

The main steps for using a navigation system for indoor navigation are summarized as follows: (1) the building should be well prepared for PVI by installing markers at the main interest points; (2) a map should be built to connect these points; (3) navigation commands help PVI to move from their current position and successfully reach their destination.

3.1.1. Marker Selection

Multiple solutions have been developed to help PVI navigate indoors. These solutions are divided into three categories: Tag-Based Systems, Computer Vision-Based Systems, and Hybrid Systems. We have already conducted a literature review to select which technology is the best for our system [3]. Based on the evaluation of the available technologies, we have concentrated on CV tag-based techniques. There are a lot of tags to choose from, but square markers are the most popular as they provide four correspondence points, which are enough to perform camera pose estimation [13].

From square markers, we have compared QR codes with Aruco markers by creating two applications to detect markers. The applications work as follows. First, it opens the camera to obtain a live stream of images. Then, it converts the image to grayscale using the open-source computer vision library called OpenCV and sends it to the desired library to detect and identify the marker. In the first application, we used QR codes, and an open-source library called Zxing for detecting and identifying markers. In the second application, we used Aruco markers, and an open-source library called Aruco library for detecting and identifying markers. After testing them in different situations, we found that Aruco markers can be detected from distances up to 4 meters, while QR codes were only successful up to 2 meters. So, we concluded that Aruco markers are better than QR codes. However, both cannot be detected in the following challenging conditions: long distances, blurring effect due to motion, and marker occlusion [35]. We also compared Aruco markers and another AR marker using deep learning models and found that Aruco markers give better accuracy [16]. Based on all these comparisons, we have selected to use Aruco markers for our system [48]. Fig. 3 shows examples of markers from our dataset in different conditions. The first one is the marker in normal situation -Fig. 3a. The second is a marker after applying lighting conditions to simulate illumination in the environment -Fig. 3b. The third is a motion blur due to fast camera movement -Fig. 3c. The fourth image is a motion blur and rotation by 90 degree at the same time -Fig. 3d.

3.1.2. Map Construction

Before using the navigation system, a map should be constructed for each floor in the building by sighted people. They should move inside the building to identify interest points such as laboratories and lecture rooms. Then, markers are printed and installed on the wall at the chosen places. Later, these markers help in guiding PVI to navigate inside the building. After that, an admin application is used to scan each marker and store details about it in a Firebase Database. This information includes marker id, floor number, and the name of the interest point like" Laboratory 407". This process is repeated for each floor in the building. Fig. 4 shows a blueprint of the fourth floor with interest points marked in red circles.

After that, a sighted person should explore all the available paths from each interest point to the points around it and measure the number of steps between them. For example, starting at node 7 in Fig. 4, a sighted person should count the number of steps from it to its neighbor nodes (1, 2, 6 and 8). The number of steps is counted at different distances to simulate different persons having different step lengths based on their impairment. Then, the average numbers are calculated for them. After that, a virtual map is constructed using a graph to store these points and the relations between them. In this graph, nodes represent interest points, and edges represent the connection between them. We use the average number of steps calculated between markers as the graph edges. In the above example, four edges are created to connect point 7 with the four other points around it. The first one is between points 7 and 1, where we store the number of steps on it. The second edge is between points 7 and 2, while the third edge connects points 7 with 6. The last edge connects points 7 with 8. This process is repeated for all interest points. Fig. 5 shows the constructed graph for the blueprint of the fourth floor.

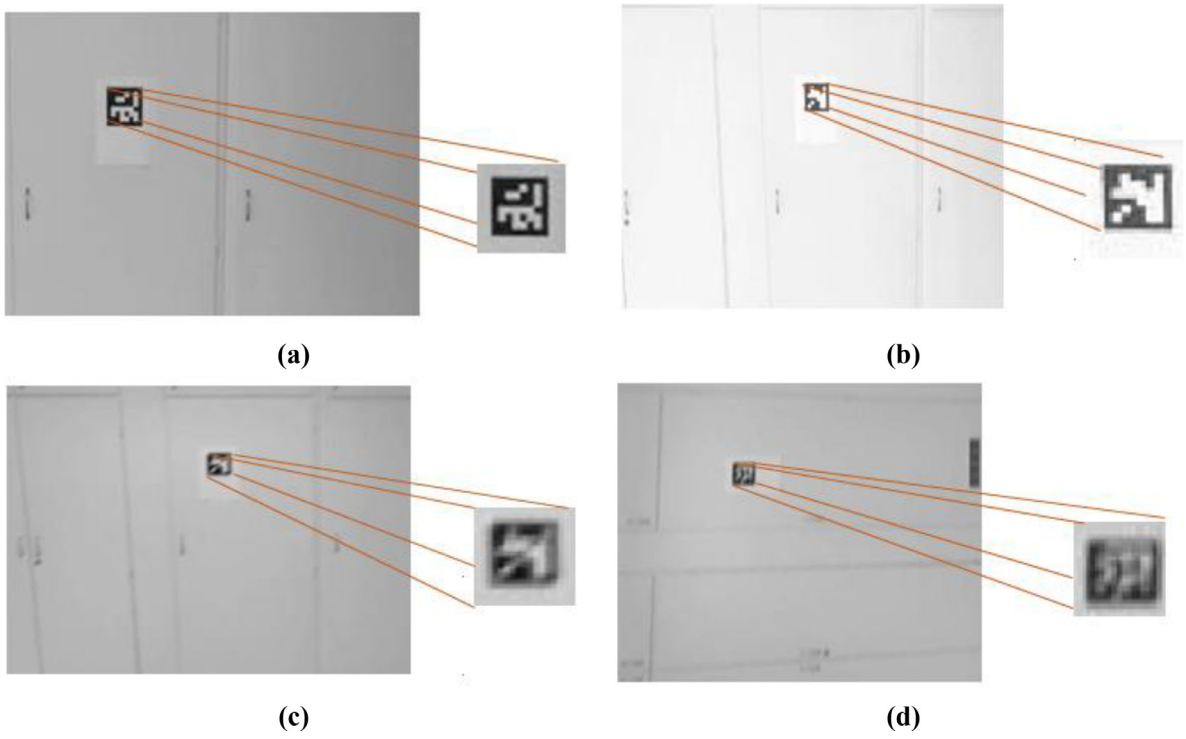


Fig. 3. Marker images obtained under challenging conditions. (a) Ideal conditions. (b) Lighting conditions. (c) Motion blur. (d) Rotation with Motion blur.

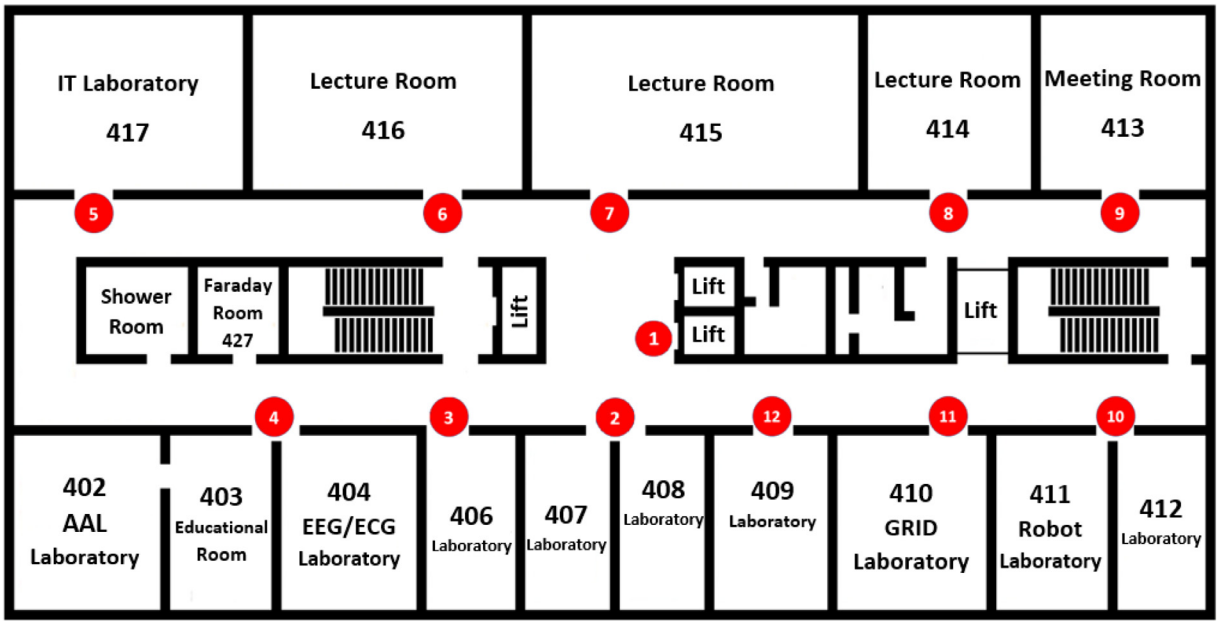


Fig. 4. The blueprint of the interest points on the fourth floor of a building. Red circles represent the interest points.

Circles represent interest points while edges represent the available paths between them. In addition to adding graph nodes to the Firebase Database, the admin application is used to store edges in it. To add this data to the database, a sighted person uses the admin application to scan two markers connected by an edge to declare it then adds the number of steps in text format. This process is repeated for all edges in the graph until the Firebase Database contains all nodes, edges, and number of steps for all edges that will be used by the navigation commands for PVI. The admin application allows removing existing markers and adding new markers. When PVI install our navigation system for the first time, it downloads the building's graph from the Firebase Database and stores it in the smartphone's local database to allow using it without Internet connection.

3.1.3. Navigation

The navigation system was designed for ease of use for PVI by using an audio interface. With a single tap on any part of the screen, the prototype application opens the camera to get a stream of frames and converts them into grayscale images. After opening the camera, an audio message asks the PVI to move the smartphone left and right to search for any marker using a Text to Speech (TTS) module. This module is used to give audio feedback to the PVI when it is needed. Table 1 lists most of the navigation

Table 1
List of the commands used during navigation and feedbacks given by our prototype.

Type	Name	Description
PVI's voice commands	"Go to" + destination	The user orders the prototype to lead them toward the predefined destinations.
	"Start"	The user orders the prototype to go to the start activity to select the start point.
	"Exit"	The user orders the prototype to exit.
Navigation instructions	"Incorrect destination, you should press on the screen and select it again"	The prototype informs the user that they should provide another destination.
	"Go straight" + number of steps	The prototype directs the user to go straight for a number of steps.
	"Turn left", "Turn right"	The prototype directs the user to turn left or right.
	"Use Elevator"	The prototype directs the user to use the elevator from one floor to another.
	"You have detected your next point, so, you should go straight to reach it"	The prototype informs the user that the next point is detected, and the user should move to it.
	"You have passed this point successfully"	The prototype informs the user that they passed this point successfully and have started navigating to the next point.
	"You have reached your destination so, go straight to it"	The prototype informs the user that they reached the desired destination.

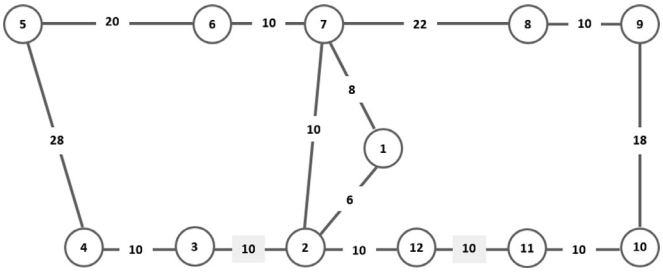


Fig. 5. A constructed graph for the blueprint. Circles represent interest points while edges represent the available paths and the number of steps between them.

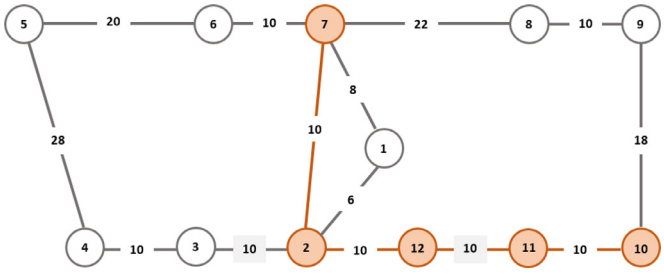


Fig. 6. The shortest path to destination example using our navigation system.

instructions used by this module. There are two fundamental components for a typical TTS model: text analysis and speech synthesis. These components convert symbols like numbers and abbreviations to written words. Then, a speech synthesis converts the navigation instructions into sounds which are understood easily by the PVI. In this module, Google TTS library is used to convert text to speech [49]. If any marker is detected, the system uses it as a starting position and then asks the PVI to select the destination point using a voice command as listed in Table 1. When the PVI communicates with the navigation system using voice command, a speech recognizer API is used to convert these commands to text using Natural Language Processing (NLP). NLP algorithms provide a way to convert voice commands correctly to text. In this article, we used the NLP algorithm found in the Google API. Then, audio feedback is given to the PVI confirming whether their command was recognized or not. If it is unrecognized, the system asks the PVI to input the destination again.

Once the starting point and destination are identified, the prototype calculates the shortest path from the initial point to the destination using Dijkstra's algorithm and instructs the PVI to start walking in the appropriate direction. This returned path is a list of marked points that the PVI should go through to reach the destination [50]. The PVI should follow the navigation commands to move from one point to the next until arriving to their destination. When the PVI reach any point by detecting the marker placed on the wall, the prototype gives navigation commands guiding them to the next point on the graph. We used the Aruco library to detect markers during navigation and calculate the distance from them to the camera. It depends on the size of the markers as seen on the captured images, so camera calibration is needed at first. With the Aruco library, we can estimate the distance between the positions of markers and the camera. If any marker is detected, this library has a function that returns two vectors to represent its position. The translation vector tells the distances between the marker and the origin of the camera coordinate system. The rotation vector describes the orientation of it. We also put markers on the wall to be seen easily from different angles, as shown in Fig. 12, to minimize the drifting error [51]. However, this library fails to detect markers under challenging conditions. So, we propose a deep learning model to solve this problem.

Fig. 6 shows an example to illustrate this process. Suppose the PVI stands in front of point 7 and wants to go to point 10. PVI taps the screen that opens the camera and moves the phone around as instructed by the application's voice command. The system detects and identifies point 7 as starting point and gives voice feedback that the initial point was selected. If more than one marker is detected simultaneously, the nearest one is selected as starting point based on distance between the phone and the marker. In this example, the PVI selects point 10 by saying "lab 411" to be their destination using voice commands. Then, the shortest path is calculated from point 7 to point 10 and returned as a list. The PVI is instructed to follow this list of points and go from point 7 through point 2, 12, 11 and 10 to reach their destination. From point 7, our system gives PVI navigation feedback to reach the next point (point 2). To reach it successfully, PVI should follow the instruction to detect the marker installed on the wall. When the marker for point 2 is detected, our application asks PVI to go towards it and gives a notification about the distance when needed. To make the system more accurate, it counts the number of steps that the PVI takes by walking from one marker to another using smartphone sensors and then compares it with the number of steps stored in the database. From point 2, the same process is repeated to guide PVI to go to the next point which is point 12. Then, go to point 11 and finally reach point 10 which is the final point. When they reach the final point, a message is given to PVI that they successfully arrived at their destination.

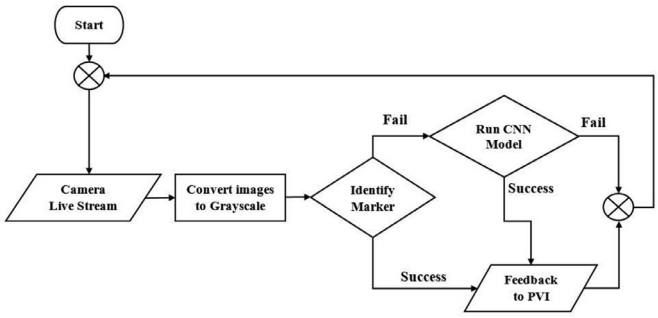


Fig. 7. Flowchart of the marker detection process.

For the navigation system to work accurately, we considered all the situations and conditions that PVI may face during navigation. We used an android smartphone (HTC desire 826) capturing videos at 30 frames per second. This means that the camera takes 30 images every second and sends it to our application for processing. Most of the images sent in a second have nearly the same scene, so if it misses detecting a marker in one frame, it will likely successfully identify it in the next ones. If the PVI finds another marker, there are two possibilities. If this marker is in the list of points to the destination, the system continues giving navigation commands from this marker to the destination point. However, if this marker is not on the list, the system searches for a new shortest path from that new marker to the destination point. If the PVI moves in a wrong direction, the camera will likely find another marker since markers cover most places inside the building. If the camera fails to detect any marker for some time, such as 30 seconds, our application gives feedback to PVI that they are walking in the wrong direction.

3.2. Detecting markers using deep learning models

We found that our system fails to detect markers in challenging conditions. So, we investigated improved versions of the Tiny-YOLOv3 model to solve this problem and improve the detection accuracy of the original model. Our research question is the following: Will this modification in the original Tiny-YOLOv3 model improve the detection accuracy and minimize the execution time? For this reason, we used the original Tiny-YOLOv3 model first to detect Aruco markers. Then, we modified the model to improve feature extraction and detection accuracy. We made the following hypotheses:

- **H1:** The modified versions of the original Tiny-YOLOv3 model will improve the detection accuracy.
- **H2:** The modified versions of the original Tiny-YOLOv3 model will lower the execution time.

With this modified model, detecting markers during navigation works as shown in Fig. 7 and is repeated until the PVI reaches their destination. After receiving a real-time stream of images and converting them to grayscale ones, there are two options to detect and identify markers. If any marker is detected using the Aruco library, voice feedback is given to the PVI. If it fails to detect markers, the image is fed into our deep learning model. A modified Tiny-YOLOv3 version is used to process the image and return the correct id if any marker is detected. However, if it fails to do so, the model concludes that no marker is available and continues processing the next image. As previously indicated, the mobile camera captures images with 30 fps. Therefore, there is no need to process all these frames while using the deep learning model, as most of them will have the same scene. The proposed model processes one frame and skips 5 frames without processing to speed up the detection algorithm.

3.2.1. Original Tiny-YOLOv3 model

Tiny-YOLOv3 model is a CNN that accepts images as an input. It consists of two main blocks: a feature extractor and detector. When a new image comes in, the feature extractor uses it as an input to extract features embedding at different scales. Then, these features feed into two branches of the detector to obtain bounding boxes and class information. The feature extractor hierarchically extracts features from pixels coming from the input layer. It uses 3×3 filters which go throughout the entire structure and max-pooling layers to reduce the dimensions of the input. The detector uses a 1×1 convolutional structure to analyze the produced results to predict the position and class of detected objects in the input image. Fig. 8 shows the original Tiny-YOLOv3 model. Given an image to this model, the final output is a list of bounding boxes along with the recognized classes. At first, the dimension of input images is reduced by a factor called the stride of the network. Then, the features are extracted by going through several convolutional layers, which makes the detection classifications. Finally, the output is given as a feature map, which represents the network class prediction. This output is converted to bounding boxes with class IDs. During training, the original Tiny-YOLOv3 model uses the same loss function used by YOLOv3, which consists of four parameters: (1) position of the prediction frame (*x, y*); (2) the prediction frame size (*w, h*); (3) the prediction *class*; (4) the prediction *confidence*.

The loss function of the original Tiny-YOLOv3 is shown in Eq. (1):

$$loss = \frac{1}{n}\sum_{k=0}^{n} loss_{xy} + \frac{1}{n}\sum_{k=0}^{n} loss_{wh} + \frac{1}{n}\sum_{k=0}^{n} loss_{class} + \frac{1}{n}\sum_{k=0}^{n} loss_{confidence} \quad (1)$$

where n is the total number of targets trained, and the loss function for each parameter in this Eq. (1) is calculated as shown in the following equations:

$$loss_{xy} = object\_mask \times (2 - w \times h) \times binary\_cross\_entropy\ (true_{xy},\ pred_{xy}) \quad (2)$$

$$loss_{wh} = object\_mask \times (2 - w \times h) \times 0.5\ \times square\ (true_{wh},\ pred_{wh}) \quad (3)$$

$$loss_{class} = object\_mask \times binary\_cross\_entropy\ (true_{class},\ pred_{class}) \quad (4)$$

$$loss_{confidence} = object\_mask \times binary\_cross\_entropy(object_{mask}, pred_{mask}) + (1 - object_{mask}) \times binary\_cross\_entropy(object_{mask}, pred_{mask}) \times ignore_{mask} \quad (5)$$

where *object_mask* is the point of the object; w and h represent the prediction box width and height respectively; *binary_cross_entropy* is a binary cross entropy function; *square* is a function of variance; $pred_{xy}$ is the predicted position and $true_{xy}$ is the actual target position; $pred_{wh}$ is the size of the prediction frame size and $true_{wh}$ is the size of actual ground truth box; $true_{class}$ and $pred_{class}$ are the actual target class and prediction class respectively; $pred_{mask}$ is the predicted object point; $ignore_{mask}$ is related to Intersection over Union (IoU) which is used for measuring the detection accuracy of corresponding objects in the dataset and is calculated using Eq. (6). If IoU is less than the specified threshold, $ignore_{mask}$ is 0.

$$\text{IoU} = \frac{\text{TP}}{\text{FP} + \text{TP} + \text{FN}} \quad (6)$$

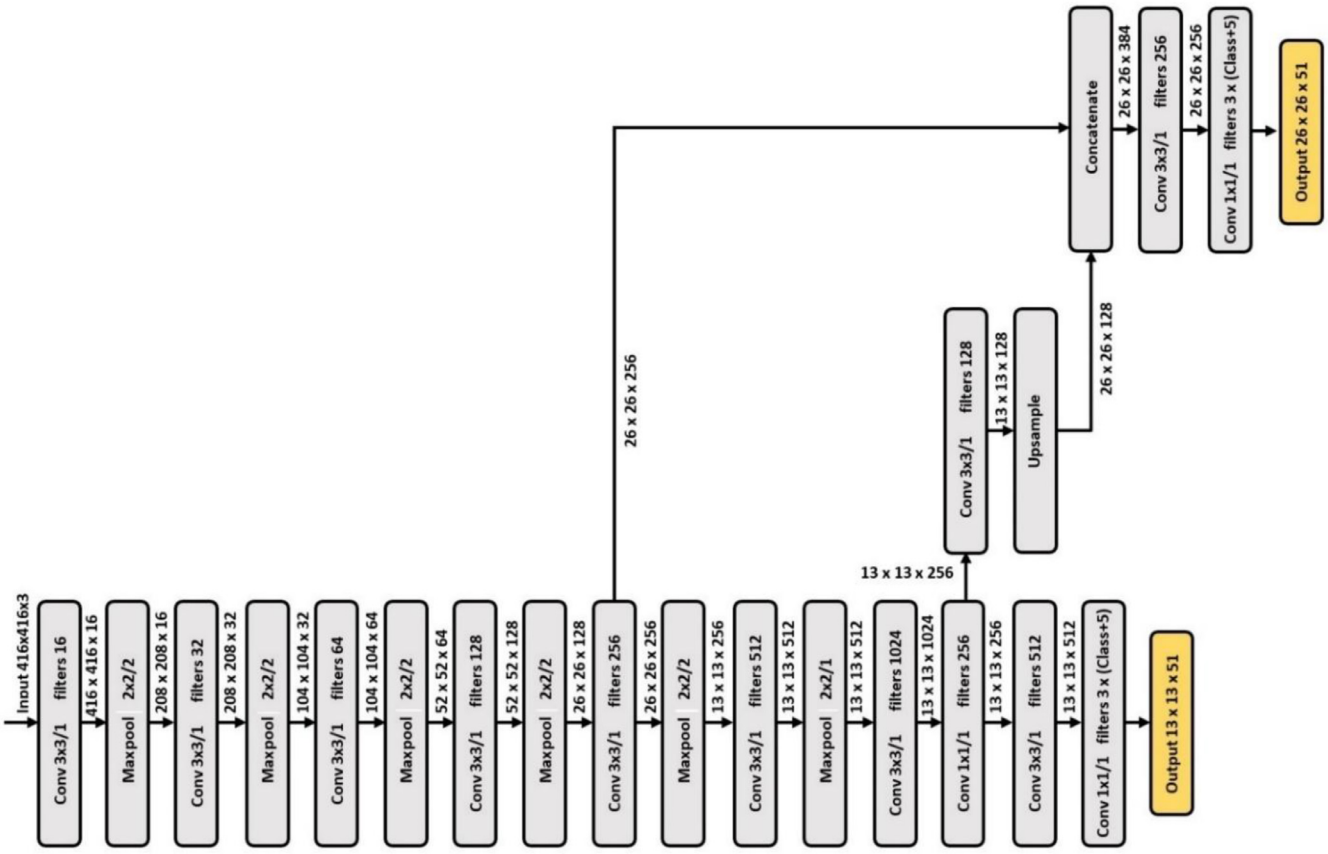


Fig. 8. The architecture of Tiny-YOLOv3 original model. It takes images with dimension 416×416 as an input. It also used layers 15 and 22 to make the final predictions and this output is shown in yellow color.

Where TP represents true positive, FP is the false positive and FN means false negative.

3.2.2. Modified Tiny-YOLOv3 Model

There are two ways to change the original Tiny-YOLOv3 to improve the accuracy of detection. The first way is to change the feature extraction module by increasing or decreasing the depth of the network. The second way is to change the detection module by adding more branches to the detector, which generates the bounding boxes and class information. In this paper, the original Tiny-YOLOv3 has been modified several times by changing the feature extraction and detection modules to improve the detection accuracy. The results of these modifications are three modified versions of the original Tiny-YOLOv3 model.

A. First version

In this version, we intended to improve model accuracy by changing the feature extraction module while keeping the detection module the same. We increased the depth of the network by adding residual network structures between the 4th to 7th layers of the original model. The roles of the added layers are to extract more features from the target and reduce information loss. The residual network uses 1×1 and 3×3 convolutional layers to extract features. The feature map of the fourth convolutional layer is concatenated with the feature map generated after adding the residual structure. Then, the output is transmitted to the fifth convolutional layer to extract features. This structure is repeated between the 4th to 7th layers, as shown in Fig. 9. The red parts are the added residual network structures to the original Tiny-YOLOv3 model.

B. Second version

The input images are down-sampled by the Tiny-YOLOv3 original model until reaching the first detection layer, where the prediction is performed at the first scale with stride 32 and 13×13 scale. Then, the output of one of the layers is up-sampled by a factor of two and concatenated with the output from one of the previous layers. The up-sampling is a simple layer with no weights that will double the dimensions of the input and can be used in a model when followed by a traditional convolutional layer. Finally, the output is used for prediction on the second scale with stride 16 and 26×26 scale. This concept is used to build the second modified version of the Tiny-YOLOv3 model with predictions across three different scales instead of two. The model makes detection at feature maps of three different sizes using strides 32, 16, 8 on scales 13×13, 26×26, and 52×52. Fig. 10 shows the architecture of the second modified version of Tiny-YOLOv3 model. The output of the convolutional layer is up-sampled and concatenated with the output from the fourth layer. Then, this output is used for the third prediction on a 52×52 scale with stride 8. This modification enriches high-level features with low-level information that helps to learn fine-grained features which are important to detect small objects.

C. Third version

As explained, the first modified Tiny-YOLOv3 version modifies the feature extraction module to improve accuracy. Meanwhile, the second modified version modifies the detection module by adding prediction at a third scale. The authors combined the two architec-

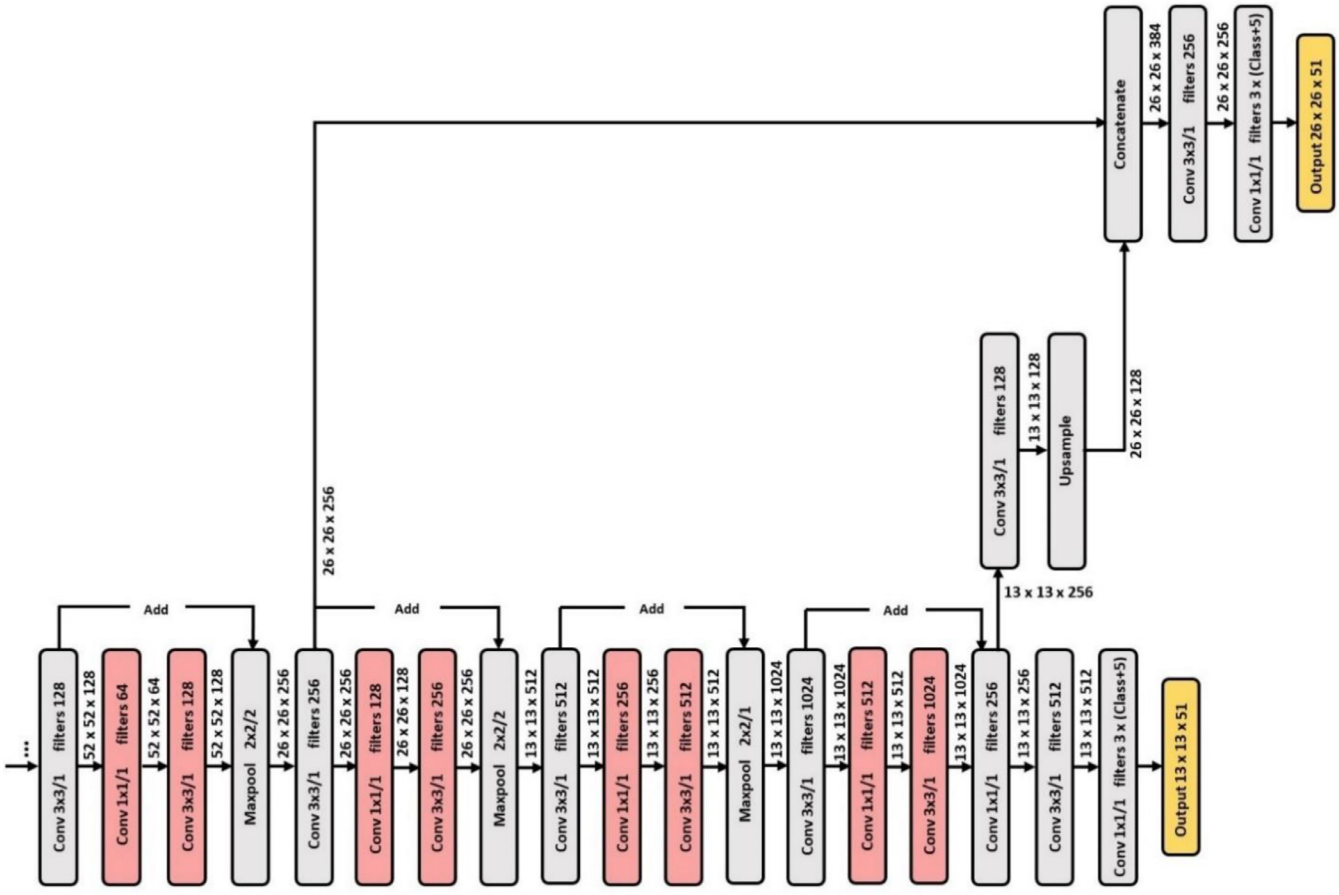


Fig. 9. The architecture of the first modified version of Tiny-YOLOv3. It takes images with dimension 416×416 as an input. Red parts are the added residual network structures to the Tiny-YOLOv3 original model.

tures to make the third modified Tiny-YOLOv3 version, as shown in Fig. 11.

4. Experiments and discussion

Our system was evaluated in three steps: First, the navigation system was evaluated using an HTC Desire 826 smartphone with 2 GB RAM, octa-core CPU and Adreno 405 GPU. Second, the performance of the proposed models was evaluated using videos on a DELL INSPIRON N5110 computer with Intel Core i7-2630 QM 2.00 GHz CPU, 6 MB cache, quad-core, and 8 GB RAM. Training our marker detection model requires a lot of resources. Therefore, we used Google Colab which leverages the power of free GPU computation for training our dataset faster. Finally, the model was uploaded and evaluated on the same HTC smartphone.

4.1. Dataset

In this study, images were collected from the testing environment using a smartphone camera. The dataset used twelve classes to represent twelve markers for the interest points on the map. For each marker, the authors used 600 images; 300 were captured from long distances between the camera and markers, while the other 300 were taken from short distances. Then, these 600 images were expanded to 7,200 using data augmentation techniques such as rotation, blur, and lighting effects to improve the detection accuracy of the neural network. To achieve this, the original images were rotated by 90, 180, and 270 degrees. These rotated images simulate holding the camera in different angles. After that, images were blurred to simulate real situations such as incorrect focus or camera movement. Lastly, several lighting effects were applied to simulate differing corridor lightings, as seen in Fig. 12, which improved detection accuracy. Including all markers, the result is a total of 86,400 images, which were divided into 57,600 images for training and 28,800 images for validation and testing. It is desirable to split the dataset into training, validation, and testing sets in a way that preserves the same proportions of examples in each class as observed in the original dataset. Training, validation, and testing sets are generally well selected to contain carefully sampled data that spans the various marker classes that the model would face when used in the real world. Finally, manual annotation was applied where bounding boxes were drawn, and categories were classified manually.

4.2. Navigation system

When selecting the starting and destination point, and during navigation, there is continuous communication between the PVI and our system, as shown in Table 1. In addition, a short introduction about how to use the application was given to the participants before using it. PVI have been trained to interact with the application for navigation and to understand the navigation voice commands. We prepared the testing environment and removed all obstacles on the way to the destination. We also asked users to hold the smartphone by hand at chest level during navigation to make sure that the smartphone camera covered the view area in front of the PVI to easily identify markers. It is also possible to mount the smartphone to the user's chest for a hands-free option. Users

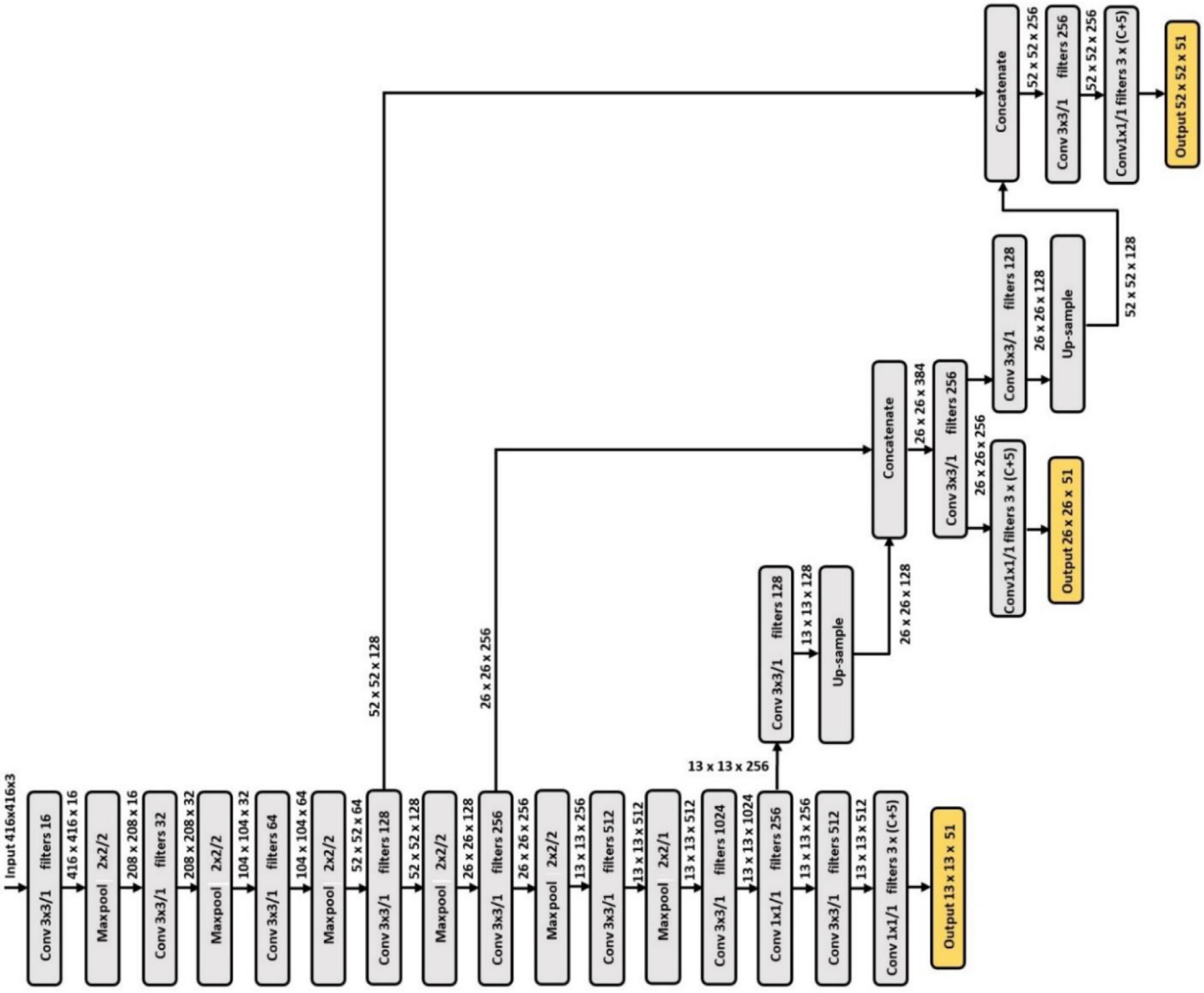


Fig. 10. The architecture of the second modified version of the original Tiny-YOLOv3 which takes images with dimension 416 × 416. It makes detection using strides 32, 16, 8 on scales 13 × 13, 26 × 26, 52 × 52.

used headphones connected to the smartphone or a smartphone's speaker to receive feedbacks during navigation. Screenshots of the testing environment is shown in Fig. 12.

Starting from the building entrance on the ground floor, the PVI selected laboratory number 404 to be the destination point, which is stored on the map as node 4 on floor number four. So, the starting point and destination are on different floors. At first, the shortest path is calculated from the entrance on the ground floor to the ground floor elevator. After reaching the fourth floor successfully, the shortest walking path between the elevator and lab number 404 is calculated. During testing, we discovered some problems: sometimes the PVI failed to understand the feedbacks so, we improved the feedback based on the comments of the PVI and found the modified audio feedback to be satisfactory. Other problems were found when: the PVI moved their hands rapidly during navigation, causing the images to be captured with a part of it is occluded; sometimes, the PVI cannot detect markers because they are moving their hands a lot and markers move out of the smartphone's camera view; markers can be captured with angles which cannot be detected correctly with our current system. We improved our environment to tackle these problems by installing eight markers with the same id at each interest point instead of adding only one, as shown in Fig. 12. This implementation makes detection easier and solved the problem of occlusion and decreased the chance for the markers to be outside of the camera view. It also helps PVI of different heights to easily detect markers. However, solving the problem of detecting markers under occlusion can be addressed by using deep learning.

4.2.1. Navigation efficiency index

Navigation efficiency index (NEI) [52] was included to evaluate the navigation performance of the systems. NEI is defined as the ratio of the actually traveled path's distance to the optimal path's distance between a source and destination. The average NEI is calculated on sub-paths, i.e., a part of the path taken by the subject while walking from the beginning to the end of the path as follows:

$$\mathrm{NEI} = \frac{1}{N} \sum_{i=1}^{N} \frac{L_A(S_i)}{L_O(S_i)} \tag{7}$$

where N is the number of sub-paths, S_i is a sub-path, L_A is the actual length traveled, and L_O is the optimal length of S_i.

We have evaluated the navigated paths using NEI. In this case, we had the main path divided into 12 sub-paths. The results are

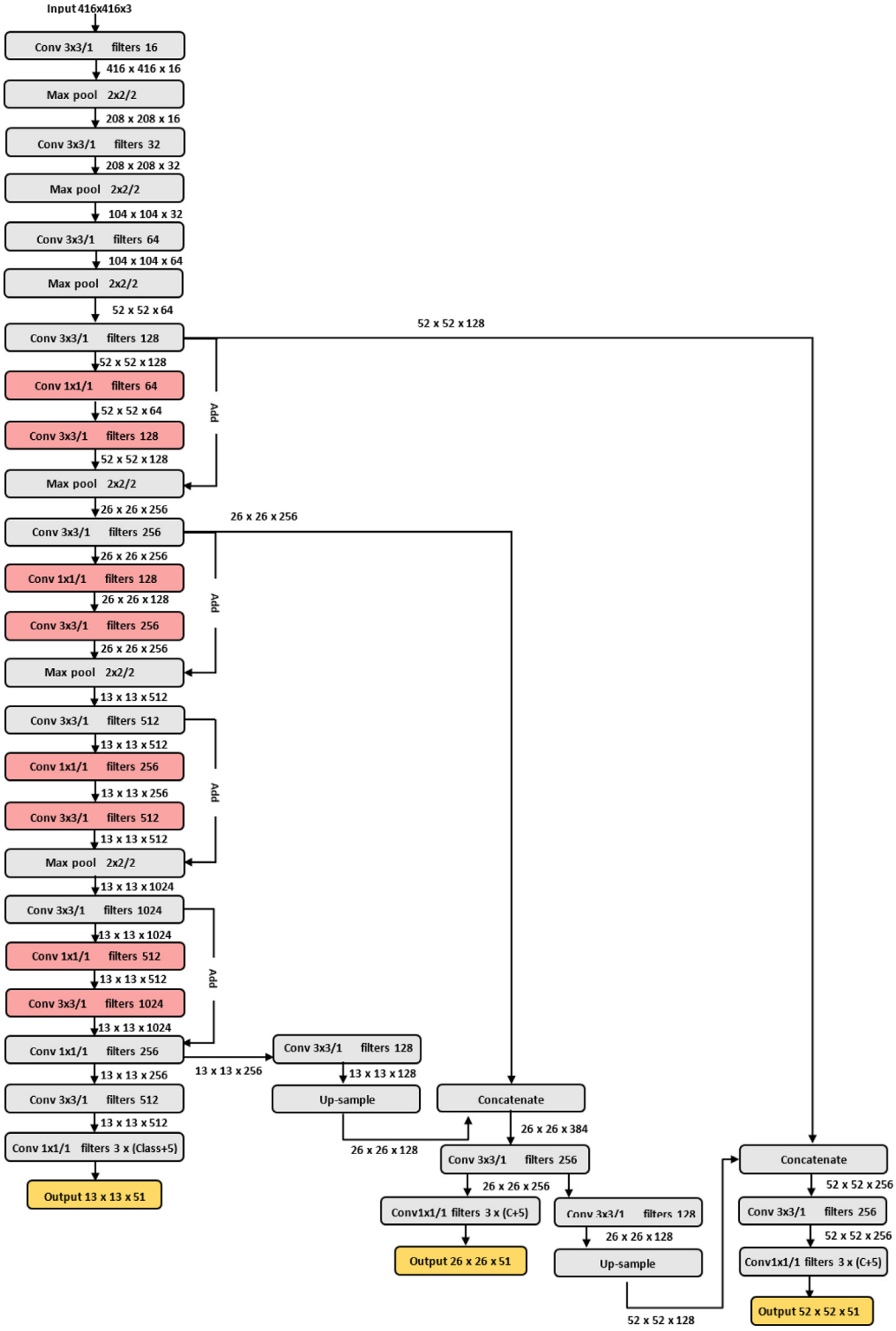


Fig. 11. The network structure of the modified version 3 which takes images with dimension 416×416. It makes detection using strides 32, 16, 8 on scales 13 × 13, 26 × 26, 52 × 52.

given in Fig. 13. The measured NEI score shows that the usability of this system is indeed acceptable in the tested indoor navigation scenarios. As shown, the low values happen when there are some turns to left or right and there are no markers in these turns. So, we will improve it by adding check-point markers at these turns to improve navigation.

4.3. Model evaluation

We trained the Tiny-YOLOv3 model, and the three modified models using the created dataset on four steps to train and test the proposed models in different parts of the datasets. The first step (Far dataset) was to evaluate the four models using a part of the full dataset that contains images captured from long distances.

Fig. 12. Screenshots of the testing environment.

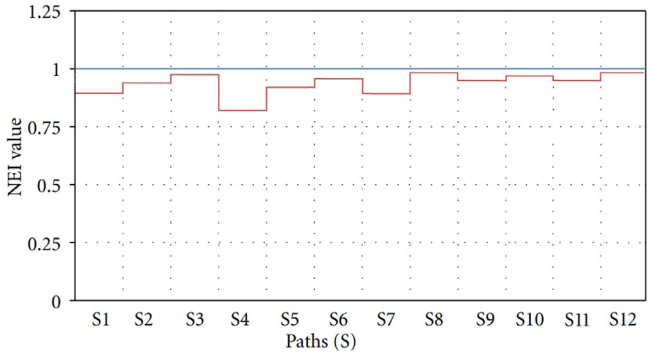


Fig. 13. Mean navigation efficiency index (NEI) versus paths (S).

We applied rotation with different angles such as 90, 180, 270 degrees. The second step (Far Challenging) was to evaluate the four models using images captured from long distances with or without applying challenging conditions such as blur and lighting effects. The third step (Full dataset) is the same as the first step, but we used images captured from long and short distances. In the last step (Full Challenging), we used the full dataset which contains images from long and short distances, rotated images with different angles, and images after applying challenging conditions like step two. We used different batch sizes and number of epochs and determined that 60 epochs in batches of size 16 yielded the best results. We used a momentum of 0.9, decay of 0.0005, Adam optimization, and a learning rate of 0.001. We found from experts' experience that these values are the best to be used for this model [24]. Models fine-tuning usually used a pre-trained model with a large dataset to get common weights and feature representation, then freeze some bottom part for further training on small or incremental data to improve and fasten the training process. So, our models used a transfer training stage from ImageNet pre-trained weights. Then, the model was unfrozen after the first 20 epochs and continued training to fine tune. In each epoch, we used 3600 iterations for training and 900 for validation. Furthermore, we calculated the training and validation losses. Every 5 epochs, precision, recall, F1 score, and mean Average Precision (mAP) were calculated to monitor the improvements during training. We used Python, TensorFlow, and the Keras framework for implementation. To evaluate these models after training, precision, recall, F1 score, average precision (AP), and mAP of the testing sets were calculated, as shown in the following subsections.

4.3.1. Loss learning curves

The loss (misclassification rate) is calculated using the number of examples that a model classifies wrong divided by the number of performed classifications. A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values. The loss of the model usually is lower on the training dataset than the validation dataset. Fig. 14 shows the training and validation loss for the four models using the full dataset in normal and challenging conditions. The loss curves are smoothly going down, indicating that our models fit better with training. The validation loss curves are slightly lower than the training loss, which indicates good model fit. Based on this, we can say that the four models were trained and validated well using our dataset.

4.3.2. Precision, Recall, and F1 Score

The analysis of precision, recall, and F1 score at different IoUs is a conventional means of evaluating object detection accuracy. If there is no detected box, but there are markers in the image, the case is considered as FN. If the detected bounding box has an IoU value greater than or equal to the predefined threshold, there are two cases. In the first case, when the predicted class is a correct marker, the box is considered a TP. In the second case, when the predicted class is not a correct marker, the box is considered a FP. Precision represents the percentage of correct predictions over the total number of predicted bounding boxes. Recall is the ratio of correctly detected boxes to the total number of markers. Precision and recall are used to evaluate the performance of any model.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

Fig. 15 shows the precision, recall, and F1 score of various Tiny-YOLOv3 marker detectors at different IoU thresholds for the full dataset. Although the graphs of a recall are nearly the same as shown in Fig. 15(b), the precision and F1 score curves of our modified Tiny-YOLOv3 version1 is the highest, as shown in (a) and (c) of Fig. 15. It is also shown that the modified Tiny-YOLOv3 version 3 is better than modified Tiny-YOLOv3 version 2 and the original Tiny-YOLOv3. This means that the modified Tiny-YOLOv3 version 1 and 3 give better performance than the other two models.

Fig. 16 shows the results of the Tiny-YOLOv3 models when using the full dataset in challenging situations. It shows that Tiny-YOLOv3 version3 and the Tiny-YOLOv3 version1 had the best precision and F1 score curves.

Table 2 shows the Precision (P), Recall (R) and F1 score at IoU = 0.5 of the four models. As shown, the results for the first modified model gives better accuracy than the other models in non-challenging conditions, as it gives a 98.50% F1 score for the Far dataset while the original model gives 97.88%. It gives 99.13% for the Full dataset, while the original model gives 87.84%. The modified version 3 is better than the original model as it gives 97.60% F1 score for the full dataset. For the dataset in challenging conditions, the modified version 3 is the best as it gives 98.40% for the far dataset and 99.31% for the full dataset while the original model gives 96.52% and 96.11% respectively. From Fig. 15, Fig. 16, and Table 2, it is seen that the modified Tiny-YOLOv3 version3 is the best choice and gives the best accuracy when used for detecting markers under challenging conditions. However, the modified Tiny-YOLOv3 version 1 is the best choice in normal conditions.

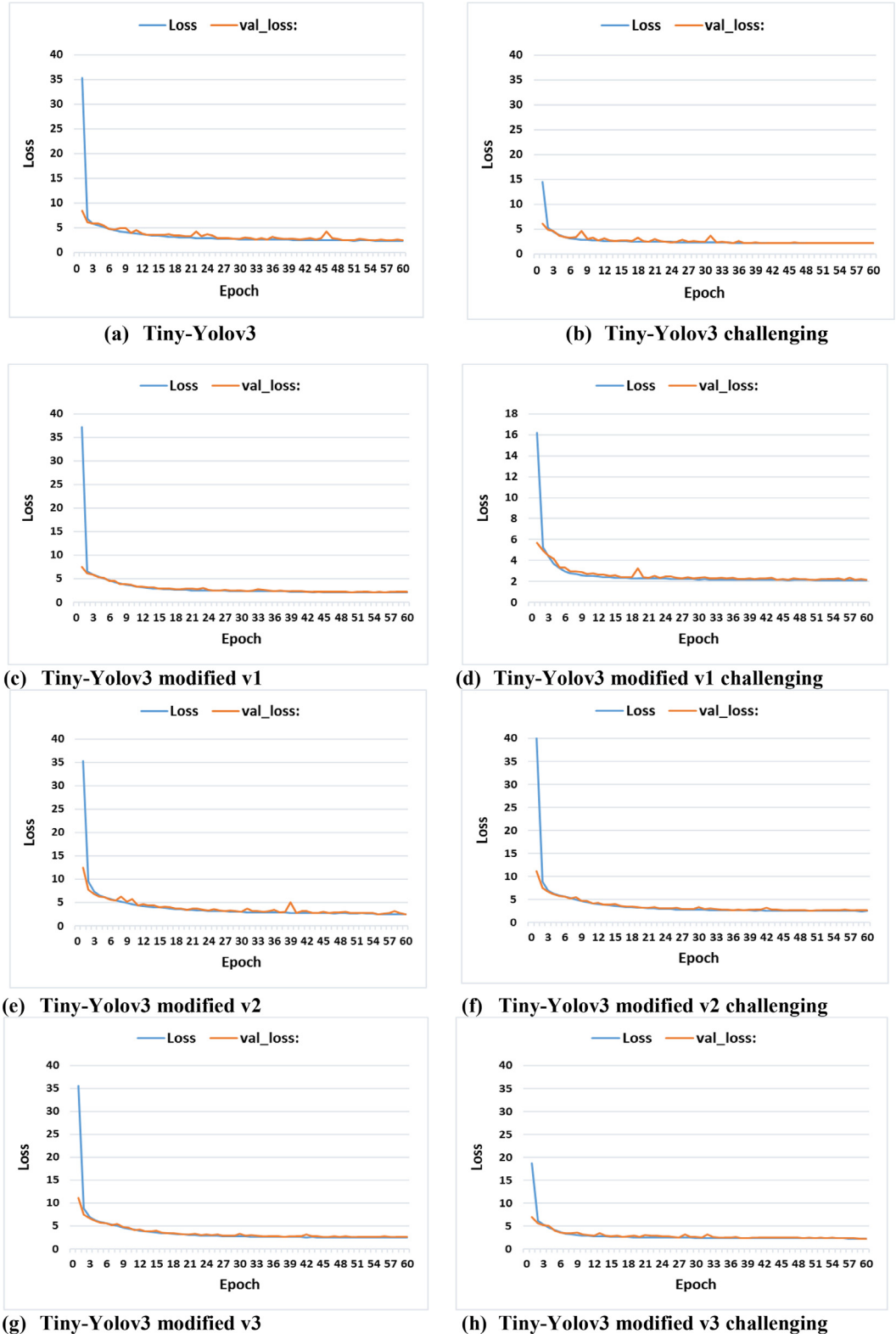


Fig. 14. Training loss and validation loss % versus epoch for the four models using the full dataset in normal and challenging conditions.

4.3.3. The mAP and AP

The average precision (AP) is a way to summarize the precision-recall curve into a single value representing the average of all precisions. The AP is calculated using a loop that goes through all precisions/recalls, the difference between the current and next recalls is calculated and then multiplied by the current precision. In other words, the AP is the weighted sum of precisions at each threshold where the weight is the increase in recall. The AP is calculated for each class, then it is used to calculate the mAP value for all classes. The mAP is an average AP value for several sets and is used for measuring detection accuracy. Fig. 17 shows the mAP curves of the four models in normal and challenging situations. It is shown

Table 2
Precision (P), Recall (R) and F1 score at IoU = 0.5 of different models.

	Far dataset			Far Challenging			Full dataset			Full Challenging		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Tiny-YOLOv3	95.84	**100**	97.88	93.29	99.99	96.52	78.32	99.98	87.84	92.52	**100**	96.11
Modified version 1	**97.21**	99.83	**98.50**	94.43	99.90	97.09	**98.27**	**100**	**99.13**	98.43	99.96	99.19
Modified version 2	79.88	99.63	88.67	93.94	99.56	96.66	89.17	100	94.28	85.82	99.96	92.35
Modified version 3	90.86	99.29	94.89	**96.85**	**100**	**98.40**	95.81	99.47	97.60	**98.97**	99.97	**99.31**

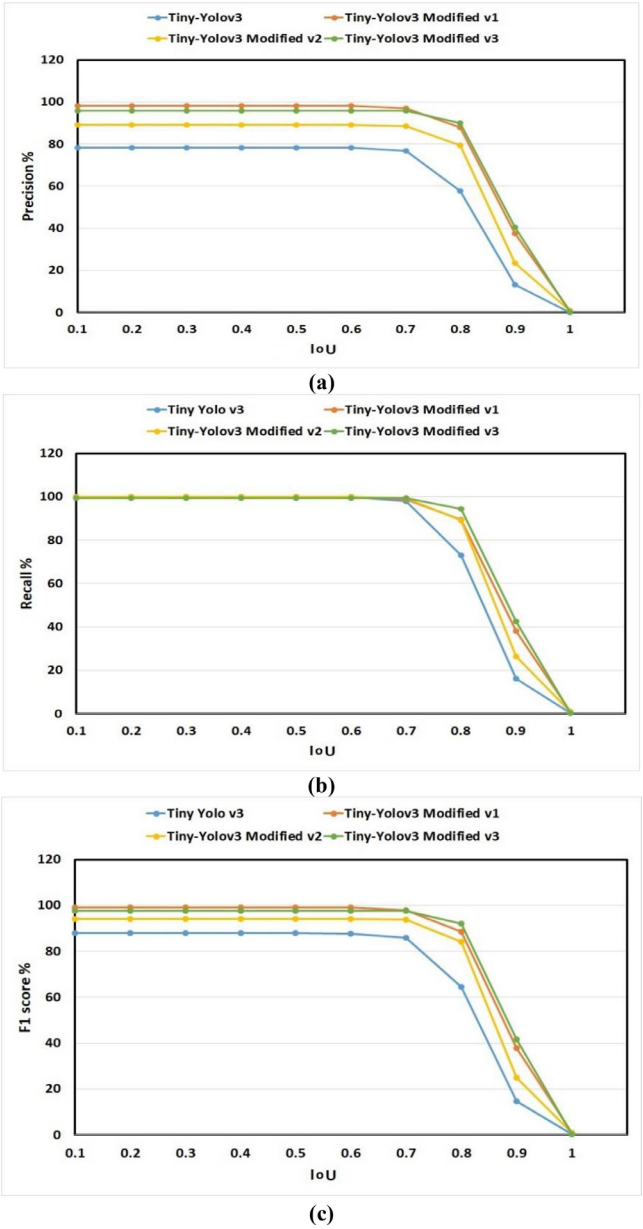


Fig. 15. Graphs for comparing (a) precision, (b) recall and (c) F1 score by different Tiny-YOLOv3 marker detector versions according to IoU threshold without challenging conditions.

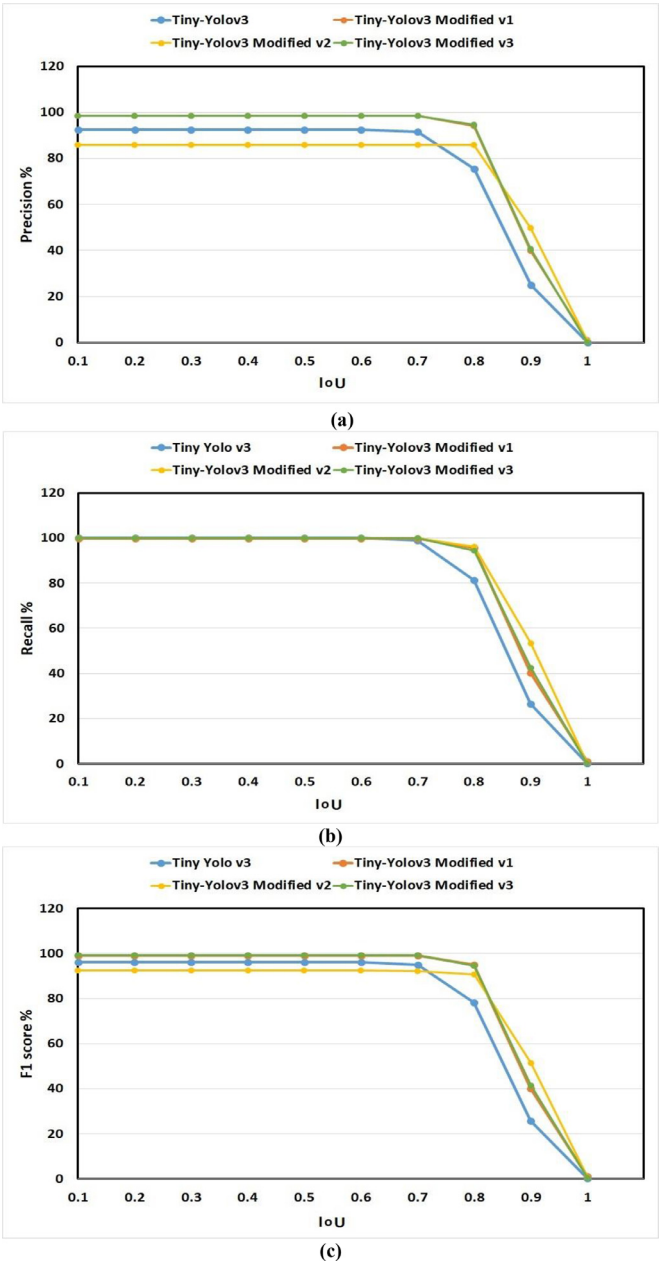
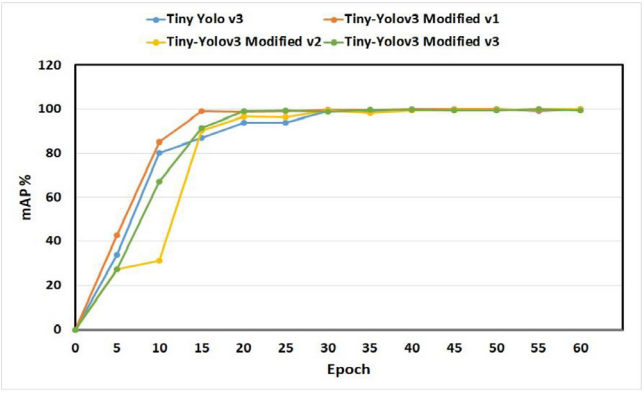


Fig. 16. Graphs for comparing (a) precision, (b) recall and (c) F1 score by different Tiny-YOLOv3 marker detector versions according to IoU threshold in challenging situation.

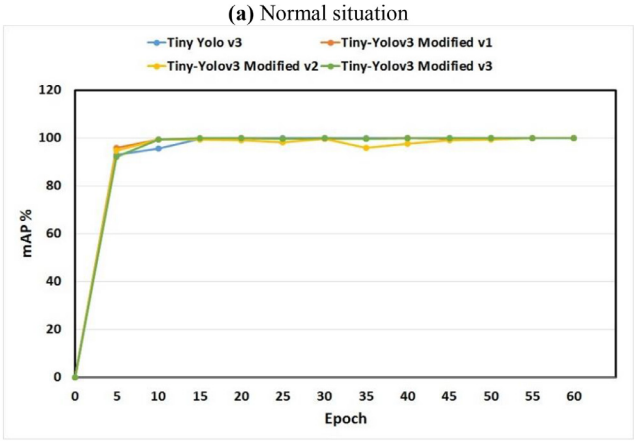
that mAP values for the four models are close to each other. So, mAP curves are not enough for comparing these models. It is also seen that the curves became more stable under challenging conditions than in normal conditions because in challenging conditions more images are used to represent situations such as rotation, blur, and lighting effects. To prove our first hypothesis, fifty sub-datasets were randomly sampled from the original test set, each with 408 images. Each model was applied on the 50 sub-datasets, and the corresponding AP, Precision, Recall, and F1 score were calculated. The p-values for each pair of methods were obtained by using the t-test. The results are analyzed at the significance level of 0.05,

Table 3
p-value for different t-tests of the original Tiny-YOLOv3 model and different modified versions.

	Modified version 1		Modified version 2		Modified version 3	
	One tail	Two tails	One tail	Two tails	One tail	Two tails
Original Version	2.86815E-09	5.7363E-09	0.378143864	0.756287728	8.53592E-09	1.70718E-08



(a) Normal situation



(b) Challenging situation

Fig. 17. Comparative graphs for the full dataset in (a) normal situation (b) challenging situation by different Tiny-YOLOv3 marker detector versions according to mAP value.

i.e., the null hypothesis $\mathbf{H_{null}}$:" there is no significant difference between the two methods" is rejected if p-value ≤ 0.05.

Table 3 shows the results of p-values when comparing the original model with each of the modified versions. We used one and two tailed t-tests. From the results, we conclude that there is a significant difference between the original model, the first, and the third modified versions. We also found that there is no significant difference between the original version and the second modified one. Based on these results, we reject the null hypothesis. We also found that there is no significant difference between the first modified version and the third modified one.

4.3.4. Execution time

In addition to detection accuracy, an important performance indicator of the target detection algorithm is processing speed. We have tested the execution time by running our models in about 15 videos. We have calculated the average execution time for the different algorithms. The mean processing time of the original Tiny-YOLOv3 model is 0.0351s while the average time for the first modified version is 0.0294s using the full dataset in challenging situations. Furthermore, the average time for the second modified version is 0.0389s and 0.0323s for the third modified version. It

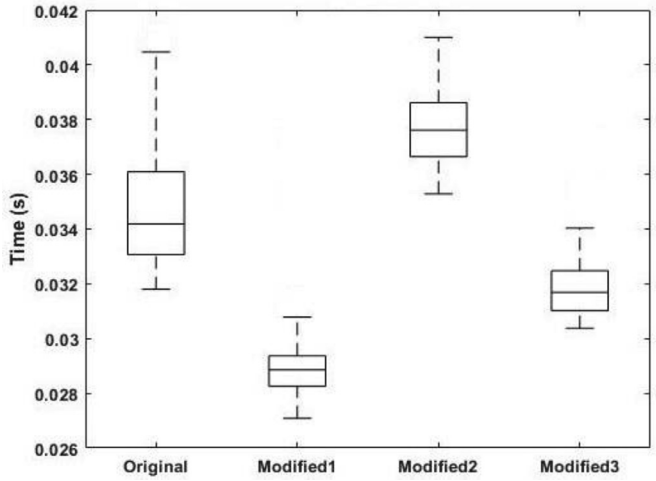


Fig. 18. Box diagrams which represent the distribution of execution times for running the four models.

is shown that the Tiny-YOLOv3 modified version 1 is the fastest model. The Tiny-YOLOv3 modified version 3 is faster than the original Tiny-YOLOv3 and the second modified version as shown in Table 4. Fig. 18 shows the box diagram which represents the distribution of execution time for running the four models. As shown, the execution time for the first and the third modified versions are the best.

We have also made a t-test for two samples to compare the running time of the original model with every one of the modified versions. We assume that the two samples have equal variances for the first test, and we make the second test for the sample mean. We assume alpha is 0.05. We state the null hypothesis as $\mathbf{H_{null}}$: the two models have the same running time. $\mathbf{H_{alt}}$: the two models have a different running time. The results are shown in Table 5. From these results, we found that there is a significant difference between the running time of the models. Based on these results, we reject null hypothesis. From evaluating the running time, we found that the first or the third modified model are the best to be used in our system.

Fig. 19 shows the marker detection examples obtained by the proposed models and the original Tiny-YOLOv3 model from different distances. The modified versions showed better results than those obtained by the original model where markers were successfully detected at both long and close distances in all cases.

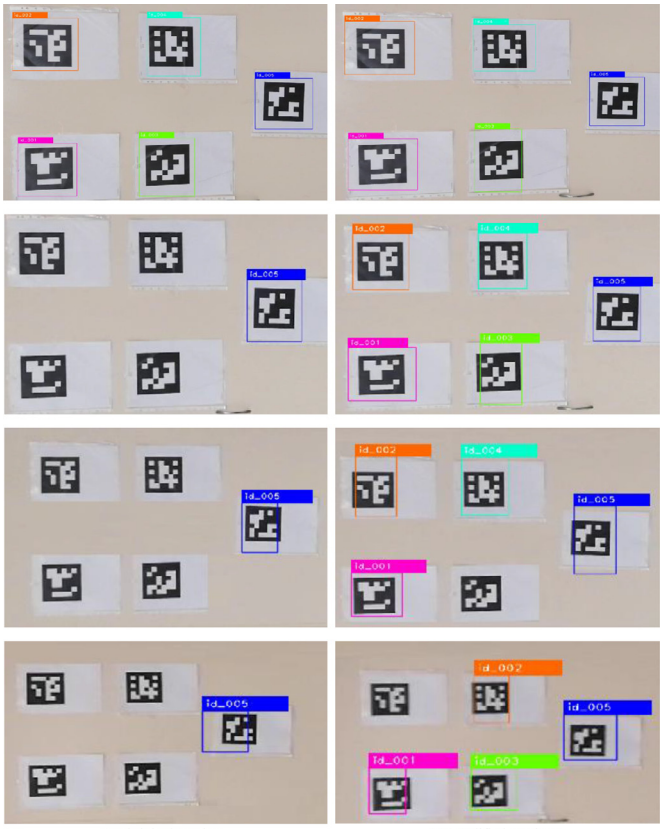
To answer the research question: we have tested and evaluated the original Tiny-YOLOv3 model and the modified version using different evaluation metrics. We have run the four models several times with different combinations of configurations like 60 or 100 epochs and batch sizes of 16 or 32. In each run, we have calculated the loss, mean precision, recall and F1-score in each epoch. From these experiments, we found that the first modified version showed the best performance in normal situations while the third modified version showed the best performance in challenging situations. From these results, Figs. 15 and 16, and our hypothesis testing, we accept the first hypothesis **H1** and use the first or the third modified versions for our system. To evaluate the second hy-

Table 4
Mean processing times of the original Tiny-YOLOv3 model and the modified models.

	Tiny-YOLOv3 Original	Tiny-YOLOv3 Modified V1	Tiny-YOLOv3 Modified V2	Tiny-YOLOv3 Modified V3
Mean time (s)	0.0351	**0.0294**	0.0389	0.0323

Table 5
p-values for different t-tests of the original Tiny-YOLOv3 model and different modified versions.

		Modified version 1		Modified version 2		Modified version 3	
		One tail	Two tails	One tail	Two tails	One tail	Two tails
Original Version	**Variance**	3.10304E-42	6.20608E-42	1.73118E-07	3.46237E-07	4.0966E-15	8.19321E-15
	Mean	5.41361E-30	1.08272E-29	5.00728E-07	1.00146E-06	6.30346E-13	1.26069E-12



(a) Original version **(b)** Modified versions

Fig. 19. Screenshots of detected markers from different distances (a) Original Tiny-YOLOv3 version (b) Modified Tiny-YOLOv3 versions.

pothesis **H2**, we have executed the original and the modified versions and calculated the inference time. It is shown that The Tiny-YOLOv3 modified version 1 is the fastest model. Also, the Tiny-YOLOv3 modified version 3 is faster than the original Tiny-YOLOv3 model and the second modified version as shown in Fig. 18 and Table 4. From these results and from our hypothesis testing, we accept the second hypothesis **H2**.

We have compared our system with the others in the related work, as shown in Table 6. In the first criterion, most solutions used deep learning to detect objects and avoid obstacles. In our system, deep learning models are used to detect markers in challenging conditions. The results give an F1 score of 99% which is evidence that the modified models are useful and appropriate for this problem. For the second criterion, some solutions used laptops as a processing unit which is heavy to carry. We used a smartphone for our system as it is easy for PVI to carry, and most of them use it for daily tasks. For the third criterion, most solutions installed QR codes or markers in the environment. However, we prefer to use Aruco markers as they are more accurate than QR codes, and our system can detect them from longer distances. Other solutions did not use any markers and only described the scenes around PVI to avoid obstacles. We use an admin application to build our virtual map in the fourth criterion. Our application constructs and updates map easily. Some solutions used a manual map creation, which has a problem of updating it if required. Other solutions did not use maps and depend on identifying the environment using computer vision techniques. In the fifth criterion, most solutions cannot detect markers in challenging conditions and from longer distances. However, there is a solution that supports identifying them in some challenging situations [43]. However, this solution was used for kids, failed to detect markers from long distances, and was developed as a desktop application. Furthermore, it used image processing techniques to select the candidate markers. These techniques take processing time that should be minimized to be suitable for real time usage. For the sixth criterion, our system and several others concentrated only on navigation. They assumed that the environment is free of obstacles or can be avoided easily using a white cane. Several solutions are used to identify objects and avoid obstacles, while others combined them. Identifying objects and avoiding obstacles are serious problems that are associated with dangerous situations. Several solutions were tested by PVI and sighted people in the seventh criterion, while others were tested only by PVI. For the last criterion, most of the accuracies were calculated for the objects' and obstacles' detection where the best one achieved 97%. In Table 6, we reported the problems found in each solution. In summary:

- Some application installed markers on the ceiling of the building which is difficult for PVI to detect.
- In some applications, markers are installed on the floor which cannot be detected in a crowded environment.
- Using markers is better than QR codes, as they can be detected from longer distances.
- Most applications failed to detect markers from long distances and in challenging conditions such as motion blur or rapid walking speed.
- Some applications use image processing techniques to select candidate markers from images and send them to classification models which takes processing time that should be minimized.
- Using IMU sensors has an acceptable positioning accuracy for only short distances.
- Some systems in the literature used expensive obstacle detection sensors which are not available for common people.
- Some applications needed a stable Internet connection to download the graph of the building from the server.
- The use of Google Glasses is an additional burden for the user and is not available for common people.

Table 6
A comparison of our system with the others in the related work.

AI		Hardware	Tags	Map Usage	challenging conditions	Function	Indented users	Accuracy	Problems
No	[14]	Google Glass Smartphone.	QR Code	Automatic	No	Indoor navigation	Sighted PVI	–	It needs Internet connection to download the graph of the building from the server. Using markers would be better than QR codes which can be detected from a long distance. The use of Google Glasses is additional burden for the user and not available for common people.
	[38]	Smartphone.	Markers	Automatic	No	Indoor navigation	Sighted PVI	–	It installed markers on the ceiling of the building which is difficult for PVI to detect. Installing markers this way lowers the aesthetic value of the building. The system was tested only by blind folded people.
	[40]	Smartphone	Markers	Manually	No	Indoor navigation	Sighted	–	Markers are installed on the floor which cannot be detected in crowded environment. It fails to detect markers from long distance. Building maps automatically would be better to enable update when needed.
	[39]	Smartphone IMU	Markers	Manually	No	Indoor navigation	PVI	–	Implemented as logging system which is not suitable for real time usage. The accuracy of markers' recognition needed to be improved as they are only visible and recognizable in a small fraction of video frames and they cannot be detected in motion blur or in rapid walking speed.
	[44]	Laptop IMU Laser sensor Portable camera	Markers	Manually	No	Indoor navigation Objects and obstacles detection	PVI	–	The size and weight of the processing unit is cumbersome for PVI to carry on the back for a long time. The obstacle detection sensor used is expensive and not available for common people. The processing time for recognition system should be minimized. Object and obstacle detection should be improved by using deep learning techniques. IMU sensors have an acceptable positioning accuracy only for a short distance since it suffers from drift error estimation over time.
Yes	[33]	Smartphone	QR Code	Yes	No	Indoor navigation Object detection.	PVI	97%	Using markers is better than QR codes which can be detected from a long distance.
	[16]	Smartphone	Markers	Manually	Yes	Indoor navigation	PVI	97%	Used CV techniques to select the candidate markers from images and sends them to classification models which takes processing time that should be minimized. Building maps automatically would be better which enables update when needed. Using markers would better than QR codes which can be detected from a long distance.
	[45]	Shield laptop	Markers	No	No	Objects and obstacles detection	- PVI	92.90	The size and weight of the processing unit is cumbersome for PVI to carry for a long time. It fails to detect markers from long distances and in challenging conditions. Using maps would be more accurate and better to help PVI navigation easily.
	[43]	Laptop	Markers	No	Yes,	Learning alphabets using AR shapes	Kids	95%.	It fails to detect markers from a long distance and was designed for kids. Implemented as a desktop application. Used CV techniques to select the candidate markers from images and sends them to classification models which takes processing time that should be minimized.
	[6]	Laptop Smartphone	–	–	No	Objects and obstacles detection	PVI	91 %.	The size and weight of the processing unit is cumbersome for PVI to carry for a long time.
	Our	Smartphone	Markers	Automatic	Yes,	Indoor navigation	Sighted PVI	99.31%.	Integrating orientation sensors to quickly warn PVI if they turn in the wrong direction would improve accuracy. Adding support to detect and avoid obstacles would be better.

- Some systems were implemented as a logging system, which is not suitable for real-time usage.
- The size and weight of the processing unit of some systems are cumbersome for PVI to carry for a long time.

5. Conclusions

Our goal is to design a real-time marker detection system. For that, we used the Tiny-YOLOv3 model. The architecture has been modified several times to increase detection accuracy. Experimentation results showed that the modified versions improve the detection accuracy. Moreover, the proposed model can be installed on different embedded solutions such as edge computing to improve inference time. A navigation system has also been built for PVI using markers. This system helped PVI to easily navigate indoors. But, before using it, a map should be constructed for each floor in the building by sighted people using an admin application. This map is a graph where nodes represent the accurate positions of the markers and edges are labelled with the number of steps and navigation instructions. PVI select the starting position and destination using a mobile camera and voice commands. Then, our system finds the shortest path from the initial point to the destination and returns it as a list of checkpoints that the PVI should walk through to arrive to their destination. While walking, continuous feedback is given to walk from one point to the next until reaching the destination. Testing our system showed that it can be easily used by PVI and blindfolded sighted people. For future work, we plan to integrate orientation sensors to quickly warn PVI if they turn in the wrong direction. Also, we will add support to detect and avoid obstacles. This system will use deep learning models installed on an embedded system to detect and identify obstacles.

Declaration of Competing Interest

None declared.

References

[1] N.A. Giudice, Navigating without vision: principles of blind spatial cognition, in: Handb. Behav. Cogn. Geogr., Edward Elgar Publishing, 2018, pp. 260–288, doi:10.4337/9781784717544.00024.

[2] Y. Zhuang, J. Yang, Y. Li, L. Qi, N. El-Sheimy, Y. Zhuang, J. Yang, Y. Li, L. Qi, N. El-Sheimy, Smartphone-Based Indoor Localization with Bluetooth Low Energy Beacons, Sensors 16 (2016) 596, doi:10.3390/s16050596.

[3] M. Elgendy, C. Sik-Lanyi, A. Kelemen, Making Shopping Easy for People with Visual Impairment Using Mobile Assistive Technologies, Appl. Sci. 9 (2019) 1061, doi:10.3390/app9061061.

[4] A. Bhowmick, S.M. Hazarika, An insight into assistive technology for the visually impaired and blind people: state-of-the-art and future trends, J. Multimodal User Interfaces. 11 (2017) 149–172, doi:10.1007/s12193-016-0235-6.

[5] E. Kostyra, S. Żakowska-Biemans, K. Śniegocka, A. Piotrowska, Food shopping, sensory determinants of food choice and meal preparation by visually impaired people. Obstacles and expectations in daily food experiences, Appetite 113 (2017) 14–22, doi:10.1016/j.appet.2017.02.008.

[6] R. Tapu, B. Mocanu, T. Zaharia, DEEP-SEE: Joint Object Detection, Tracking and Recognition with Application to Visually Impaired Navigational Assistance, Sensors 17 (2017) 2473, doi:10.3390/s17112473.

[7] R. Velázquez, E. Pissaloux, P. Rodrigo, M. Carrasco, N. Giannoccaro, A. Lay-Ekuakille, An Outdoor Navigation System for Blind Pedestrians Using GPS and Tactile-Foot Feedback, Appl. Sci. 8 (2018) 578, doi:10.3390/app8040578.

[8] K. Manjari, M. Verma, G. Singal, A survey on Assistive Technology for visually impaired, Internet of Things, 11, 2020, doi:10.1016/j.iot.2020.100188.

[9] R. Jafri, S.A. Ali, H.R. Arabnia, S. Fatima, Computer vision-based object recognition for the visually impaired in an indoors environment: a survey, Vis. Comput. 30 (2014) 1197–1222, doi:10.1007/s00371-013-0886-1.

[10] S.A.S. Mohamed, M.H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, J. Plosila, A Survey on Odometry for Autonomous Navigation Systems, IEEE Access 7 (2019) 97466–97486, doi:10.1109/ACCESS.2019.2929133.

[11] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, M.J. Marín-Jiménez, Automatic generation and detection of highly reliable fiducial markers under occlusion, Pattern Recognit 47 (2014) 2280–2292, doi:10.1016/j.patcog.2014.01.005.

[12] E. Marchand, H. Uchiyama, F. Spindler, Pose Estimation for Augmented Reality: A Hands-On Survey, IEEE Trans. Vis. Comput. Graph. 22 (2016) 2633–2651, doi:10.1109/TVCG.2015.2513408.

[13] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Generation of fiducial marker dictionaries using Mixed Integer Linear Programming, Pattern Recognit 51 (2016) 481–491, doi:10.1016/j.patcog.2015.09.023.

[14] S. Al-Khalifa, M. Al-Razgan, Ebsar: Indoor guidance for the visually impaired, Comput. Electr. Eng. 54 (2016) 26–39, doi:10.1016/j.compeleceng.2016.07.015.

[15] A. Morar, A. Moldoveanu, I. Mocanu, F. Moldoveanu, I.E. Radoi, V. Asavei, A. Gradinaru, A. Butean, A Comprehensive Survey of Indoor Localization Methods Based on Computer Vision, Sensors 20 (2020) 2641, doi:10.3390/s20092641.

[16] M. Elgendy, T. Guzsvinecz, C. Sik-Lanyi, Identification of Markers in Challenging Conditions for People with Visual Impairment Using Convolutional Neural Network, Appl. Sci. 9 (2019) 5110, doi:10.3390/app9235110.

[17] K. He, X. Zhang, S. Ren, J. Sun, Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, in: IEEE Trans. Pattern, 2014, pp. 346–361, doi:10.1007/978-3-319-10578-9_23.

[18] R. Girshick, Fast R-CNN, in: 2015 IEEE Int. Conf. Comput. Vis., IEEE, 2015, pp. 1440–1448, doi:10.1109/ICCV.2015.169.

[19] S. Ren, K. He, R. Girshick, J. Sun, R-CNN Faster, Towards Real-Time Object Detection with Region Proposal Networks, IEEE Trans. Pattern Anal. Mach. Intell. 39 (2017) 1137–1149, doi:10.1109/TPAMI.2016.2577031.

[20] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection, in: 2016 IEEE Conf. Comput. Vis. Pattern Recognit, IEEE, 2016, pp. 779–788, doi:10.1109/CVPR.2016.91.

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, A.C. Berg, SSD: Single shot multibox detector, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), Springer Verlag, 2016, pp. 21–37, doi:10.1007/978-3-319-46448-0_2.

[22] P. Soviany, R.T. Ionescu, Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction, in: 2018 20th Int. Symp. Symb. Numer. Algorithms Sci. Comput., IEEE, 2018, pp. 209–214, doi:10.1109/SYNASC.2018.00041.

[23] J. Redmon, A. Farhadi, YOLO9000: Better, Faster, Stronger, in: 2017 IEEE Conf. Comput. Vis. Pattern Recognit., IEEE, 2017, pp. 6517–6525, doi:10.1109/CVPR.2017.690.

[24] J. Redmon, A. Farhadi, YOLOv3: An Incremental Improvement, (2018). http://arxiv.org/abs/1804.02767 (accessed April 26, 2020).

[25] S. Pang, Z. Yu, M.A. Orgun, A novel end-to-end classifier using domain transferred deep convolutional neural networks for biomedical images, Comput. Methods Programs Biomed. 140 (2017) 283–293, doi:10.1016/j.cmpb.2016.12.019.

[26] Y. Xiao, J. Wu, Z. Lin, X. Zhao, A semi-supervised deep learning method based on stacked sparse auto-encoder for cancer prediction using RNA-seq data, Comput. Methods Programs Biomed. 166 (2018) 99–105, doi:10.1016/j.cmpb.2018.10.004.

[27] Y. Xiao, J. Wu, Z. Lin, X. Zhao, A deep learning-based multi-model ensemble method for cancer prediction, Comput. Methods Programs Biomed. 153 (2018) 1–9, doi:10.1016/j.cmpb.2017.09.005.

[28] S.W. Yang, S.K. Lin, Fall detection for multiple pedestrians using depth image processing technique, Comput. Methods Programs Biomed. 114 (2014) 172–182, doi:10.1016/j.cmpb.2014.02.001.

[29] J. Tang, Q. Su, B. Su, S. Fong, W. Cao, X. Gong, Parallel ensemble learning of convolutional neural networks and local binary patterns for face recognition, Comput. Methods Programs Biomed. 197 (2020) 105622, doi:10.1016/j.cmpb.2020.105622.

[30] C. González García, D. Meana-Llorián, B.C. Pelayo G-Bustelo, J.M. Cueva Lovelle, N. Garcia-Fernandez, Midgar: Detection of people through computer vision in the Internet of Things scenarios to improve the security in Smart Cities, Smart Towns, and Smart Homes, Futur. Gener. Comput. Syst. 76 (2017) 301–313, doi:10.1016/j.future.2016.12.033.

[31] B. AL-Madani, F. Orujov, R. Maskeliūnas, R. Damaševičius, A. Venčkauskas, Fuzzy Logic Type-2 Based Wireless Indoor Localization System for Navigation of Visually Impaired People in Buildings, Sensors 19 (2019) 2114, doi:10.3390/s19092114.

[32] W.C.S.S. Simões, G.S. Machado, A.M.A. Sales, M.M. De Lucena, N. Jazdi, V.F. De Lucena, A Review of Technologies and Techniques for Indoor Navigation Systems for the Visually Impaired, Mdpi.Com (2021), doi:10.3390/s20143935.

[33] E. Ko, E.Y. Kim, A Vision-Based Wayfinding System for Visually Impaired People Using Situation Awareness and Activity-Based Instructions, Sensors 17 (2017) 1882, doi:10.3390/s17081882.

[34] J.C. Torrado, G. Montoro, J. Gomez, Easing the integration: A feasible indoor wayfinding system for cognitive impaired people, Pervasive Mob. Comput. 31 (2016) 137–146, doi:10.1016/j.pmcj.2016.02.003.

[35] M. Elgendy, M. Herperger, T. Guzsvinecz, C.S. Lanyi, Indoor Navigation for People with Visual Impairment using Augmented Reality Markers, in: 2019 10th IEEE Int. Conf. Cogn. Infocommunications, IEEE, 2019, pp. 425–430, doi:10.1109/CogInfoCom47531.2019.9089960.

[36] G.C. La Delfa, V. Catania, S. Monteleone, J.F. De Paz, J. Bajo, Computer Vision Based Indoor Navigation: A Visual Markers Evaluation, in: Adv. Intell. Syst. Comput., 2015, pp. 165–173, doi:10.1007/978-3-319-19695-4_17.

[37] G.La Delfa, V. Catania, Accurate indoor navigation using smartphone, bluetooth low energy and visual tags, in: Proc. 2nd Conf. Mob. Inf. Technol. Med., 2014, pp. 5–8. https://pdfs.semanticscholar.org/5151/d3b2f8e40e09853ea81506a62c854b63a129.pdf.

[38] D. Khan, S. Ullah, S. Nabi, A Generic Approach toward Indoor Navigation and Pathfinding with Robust Marker Tracking, Mdpi.Com (2019), doi:10.3390/rs11243052.

[39] G. Fusco, J.M. Coughlan, Indoor localization using computer vision and visual-inertial odometry, in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), Springer Verlag, 2018, pp. 86–93, doi:10.1007/978-3-319-94274-2_13.
[40] G. Lee, H. Kim, A Hybrid Marker-Based Indoor Positioning System for Pedestrian Tracking in Subway Stations, Appl. Sci. 10 (2020) 7421, doi:10.3390/app10217421.
[41] H. Zhang, C. Zhang, W. Yang, C.Y. Chen, Localization and navigation using QR code for mobile robot in indoor environment, in: 2015 IEEE Int. Conf. Robot. Biomimetics, IEEE-ROBIO 2015, 2015, pp. 2501–2506, doi:10.1109/ROBIO.2015.7419715.
[42] Y. Li, S. Zhu, Y. Yu, Z. Wang, An improved graph-based visual localization system for indoor mobile robot using newly designed markers, Int. J. Adv. Robot. Syst. (2018) 15, doi:10.1177/1729881418769191.
[43] A.K. Dash, S.K. Behera, D.P. Dogra, P.P. Roy, Designing of marker-based augmented reality learning environment for kids using convolutional neural network architecture, Displays 55 (2018) 46–54, doi:10.1016/j.displa.2018.10.003.
[44] M.L. Mekhalfi, F. Melgani, A. Zeggada, F.G.B. De Natale, M.A.-M. Salem, A. Khamis, Recovering the sight to blind people in indoor environments with smart technologies, Expert Syst. Appl. 46 (2016) 129–138, doi:10.1016/j.eswa.2015.09.054.
[45] Y. Bazi, H. Alhichri, N. Alajlan, F. Melgani, Scene Description for Visually Impaired People with Multi-Label Convolutional SVM Networks, Appl. Sci. 9 (2019) 5062, doi:10.3390/app9235062.
[46] S. Kayukawa, K. Higuchi, J. Guerreiro, S. Morishima, Y. Sato, K. Kitani, C. Asakawa, BBeep: A Sonic Collision Avoidance System for Blind Travellers and Nearby Pedestrians, Proc (2019), doi:10.1145/3290605.3300282.
[47] M. Liu, X. Wang, A. Zhou, X. Fu, Y. Ma, C. Piao, UAV-YOLO: Small Object Detection on Unmanned Aerial Vehicle Perspective, Sensors 20 (2020) 2238, doi:10.3390/s20082238.
[48] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Generation of fiducial marker dictionaries using Mixed Integer Linear Programming, Pattern Recognit 51 (2016) 481–491, doi:10.1016/j.patcog.2015.09.023.
[49] G. López, L. Quesada, L.A. Guerrero, Alexa vs. Siri vs. Cortana vs. Google Assistant: A Comparison of Speech-Based Natural User Interfaces, in: Adv. Intell. Syst. Comput., Springer Verlag, 2018, pp. 241–250, doi:10.1007/978-3-319-60366-7_23.
[50] D.B. Johnson, A Note on Dijkstra's Shortest Path Algorithm, J. ACM. 20 (1973) 385–388, doi:10.1145/321765.321768.
[51] ArUco: a minimal library for Augmented Reality applications based on OpenCV | Aplicaciones de la Visión Artificial, (n.d.). http://www.uco.es/investiga/grupos/ava/node/26 (accessed December 23, 2020).
[52] A. Ganz, J. Schafer, … S. Gandhi, PERCEPT indoor navigation system for the blind and visually impaired: architecture and experimentation, Hindawi.Com (2012) https://www.hindawi.com/journals/ijta/2012/894869/abs/. (accessed August 26, 2020).