A single camera 360-degree real time vision-based localization method with application to mobile robot trajectory tracking

Khomsun Singhirunnusorn | Farbod Fahimi | Ramazan Aygun

Abstract

A method is proposed for real-time vision-based localization in the 360° area around a three-dimensional (3D) reference object with a single camera. The problem is split into three subproblems. First, 360° 3D object recognition is proposed, in which a computer vision solution can recognize a reference object from all possible 360° locations. Second, 360° pose estimation is presented, in which the pose of a robot at all locations is estimated. Third, a 360° localization application is integrated with a closed-loop real-time trajectory tracking controller. The successful results of simulations and real experiments of trajectory tracking are also presented.

1 INTRODUCTION

Vision is an important sensor that can be used to perceive the environment. Cameras are cheap and provide information about the shape, size and texture of an object. They are needed by the vision-based control unit of a mobile robot. There are many studies on visual robot control or visual servoing. Visual servoing has been introduced to control the trajectory of a robot with respect to a target object using feedback obtained by a vision system [1]. The objective of closed-loop visual servoing is to minimize the difference between the actual pose and the desired pose using feedback. Existing approaches can be classified into two main categories: (1) position-based visual servoing, in which feedback is defined in terms of three-dimensional (3D) Cartesian information derived from the image(s), and (2) image-based visual servoing, in which where feedback is defined directly in the image in terms of image features [1]. Before one can guide a mobile robot to the desired path, the self-localization problem must be solved. In this research, a robot uses an external object as its reference point to determine its pose and location. This research proposes an alternative vision-based localization with a single camera. Our research team developed a pose estimation method called Mirage. Using a desired pose and the current image of a 3D object, Mirage provides the estimated 6-degrees-of-freedom (6DOF) pose of the robot with minimum errors in real time. Mirage has been studied for a number of problems including robot trajectory tracking control [2], pose estimation with one camera [3], two cameras [4], and pose recovery [5]. We propose solutions to eliminate the 3D object recognition problem caused by the relative location of the robot with respect to the object, such as when the robot moves behind the reference 3D object. The outcome of the simulations and real experiments demonstrate small pose estimation errors in all 360° coverage around the 3D reference object.

The work focuses on a landmark-based system in which an autonomous system can determine its pose or follow a trajectory based on a landmark. In a landmark-based system, autonomous systems move, drive, or fly with respect to a known landmark. Hence, the landmark should always be detected by these systems to be able to function normally. For example, consider a case in which an unmanned aerial system (UAS) needs to gather information for a police station in a police car. The UAS may use a landmark on the police car or even the police car itself as a landmark. It may fly on a desired trajectory with a given pose, collect the information, and fly back using the landmark. Images of the landmark taken on the wireless camera on the UAS could be sent to a ground station for real-time processing of the UAS’s pose. Then, the UAS’s trajectory tracking controller can estimate the pose at which to fly the UAS accurately on the desired path.

Our contributions can be summarized thus:

• Existing methods assumed that the same side of the landmark object is always in view of the camera. The
authors eliminate that assumption and propose a method for recognising the landmark object from all sides, leading to the ability of robot localization irrespective of the location of the object around the landmark object. The authors propose an efficient method for recognising what side of the object is in view. This method is simpler to implement and is computationally efficient compared with existing methods.

The rest of this work presents the material as follows. Section 2 presents a review of studies in vision-based camera localization. Section 3 explains details of the 360° vision-based localization method with a single camera system. In Sections 4 and 5, the results of both simulations and experiments are presented. Finally, the last section concludes the work.

2 RELATED WORK

Vision-based camera localization has two categories related to environments: one with unknown environments and another with known environments. Methods for unknown environments include online and real-time environment mapping called simultaneous localization and mapping (SLAM) [6] and the methods without online and real-time environment mapping, called structure from motion (SFM) [7].

SLAM focuses on visual sensors called Visual-SLAM. Visual-SLAM uses images as external information to estimate the location of the robot and constructs the environment map [8]. Visual-SLAM has gradually become the preferred choice for lunar rover positioning [8]. Visual-SLAM, which uses a single camera to localize a mobile robot, is a challenging problem because the image from the monocular camera lacks depth information. Therefore, multisensor approaches were introduced, which improved robustness and gave a higher accuracy compared with that of inertial measurement units (IMUs) and global positioning systems (GPSs). Piao and Kim [9] proposed the method of combining a monocular camera and IMU sensor in mobile devices to implement a more accurate and stable SLAM system. Shi et al. [10] presented a framework for the GPS to support a visual-SLAM with bundle adjustment using a rigorous sensor model in a panoramic camera. Camera technologies such as infrared and deep cameras were introduced to contribute to the pose estimation approaches. Su et al. [11] and Xie et al. [12] proposed a system for unmanned aerial vehicle (UAV) 6DOF pose estimation. Su et al. [11] used an infrared active marker and a colored passive marker to mark the relative pose of the target. With two different types of sensors, a particle swarm optimization (PSO) algorithm was employed to solve the optimal pose of a UAV. Xie et al. [12] also used a PSO for the correspondence search between landmarks and image detections, and then applied the Kalman filter to track and predict pose of the UAV in space.

For SFM approaches, Irshara et al. [7] presented a fast location recognition technique based on structure from motion point clouds. Burschk a and Mair [13] proposed an SFM method to calculate 6DOF pose estimation directly from one camera with two images. Camposeco et al. [14] proposed the pose estimation of calibrated pinhole and generalized cameras with respect to an SFM model by leveraging both 2D-3D correspondences as well as 2D-2D correspondences. In stages of the SFM development, there were more works on global pose solving. In addition, Li et al. [15] estimated a full 6DOF camera pose with respect to a large geo-registered 3D point cloud. It combined research on image localization, landmark recognition, and 3D pose estimation. Park et al. [16] estimated the camera direction of a geotagged image using Google Street View and Google Earth satellite images. Cui and Tan [17] presented a global method for camera pose registration.

Camera localization with a known environment or 3D space points is called a perspective-n-point (PnP) problem. The goal of a PnP problem is to find 6DOF pose parameters. There are no solutions for the PnP problem when \( n = 1, 2 \) because in that case, the number of constraints is less than the number of parameters. Many solutions have been proposed to solve the PnP pose estimation problem when \( n \geq 3 \). The minimal case is the P3P [18] problem, which uses only three matching points to solve for unknown pose parameters. However, this may yield two to four solutions. All solutions are not viable at the same time, which causes ambiguity [19]. This ambiguity could be solved by using more feature points. For example, Guo et al. [20] proposed P4P, whereas Tang et al. [21] used P5P to find the pose parameters. Theoretically, the pose estimation should be solved by using a small number of feature points. However, Zheng et al. [22] suggested that using many feature points reduced the side effects of noise issues.

Our research team proposed Mirage-S (Mirage for single camera) [3]. Mirage-S works well using eight feature points and has superior performance compared with eight well-known PnP methods [3]. Mirage-S yields the minimum average error less than 6 cm in translation and the minimum average rotational error less than \( 2^\circ \).

3 METHOD

In this section, we present a 360° pose estimation method with a single camera. The goal is to find the actual pose anywhere around a 360° area. There are three approaches to achieving this goal.

3.1 Three hundred sixty-degree three-dimensional object recognition

The first stage is to recognize the side of the reference 3D object faced by the camera from any point of view. The vision system needs a fixed 3D reference object in the real world. A geometry of two octagonal prisms, in which the small one is on top of the big one, is defined as the 3D reference object, shown in Figure 1. This 3D object is an inertial coordinate system to calculate the actual pose of the mobile robot. It has eight sides; each side has one face. Each face has a unique...
pattern of eight circles, as shown in Figure 2. The centers of these circles are used as feature points because Mirage-S yields good results using at least eight feature points. To recognize the 3D object, the system must match the feature points in the 3D model to the feature points on the image captured by the camera. The two main processes in this stage that are presented are feature extraction (explained in Section 3.1.1) and feature matching and correspondence (covered in Section 3.1.2).

3.1.1 Feature extraction using MATLAB image processing toolbox

This stage is composed of detecting feature points by finding circles on the 3D object and then determining the face(s) viewed by the camera using the patterns of feature points. Our goal in this stage is to recognize the face(s) of the 3D reference object in the actual images captured by the robot in real time.

**Detecting Feature Points.** To find the feature points, we must first detect circles on the image. After capturing a grayscale image, Otsu’s method [23] is used to find a single-intensity threshold. This threshold was used to separate pixels into two classes (foreground and background) to generate a binary image. Based on our 3D model, we defined the size of the circle area between 150 pixels and 440 pixels assuming the distance to the 3D object to be 4–7 ft. By using the `regionprops` function in the MATLAB image processing toolbox [24], a set of regions of interest (ROIs) or blobs from the binary image is obtained to determine candidate circles of the object. In the ROIs, we determined all possible circles in the image by calculating the simple circularity of blob(e) [25] as in Equation (1):

\[
c = \frac{4\pi \text{area}}{\text{perimeter}^2}
\]

where `area` and `perimeter` respectively represent the area and perimeter of the blob. The circularity of a circle (in the reference object) is 1 or close to 1. For our empirical experimental results, c should be greater than 0.75. Then, we eliminated all possible circles that are outliers. The center points of detected circles are used as feature points.

**Identifying Face(s).** Because each face of the 3D reference object has eight feature points, viewing one face of the object should be enough for Mirage-S, because it requires at least eight feature points. However, if the number of feature points detected is less than eight, the system should ignore that image and retake a new image. It is also likely that the neighboring faces might be visible in the camera owing to the object’s octagonal prism shape, leading to 24 feature points detectable by the system. Labeling of the circles is important for clustering of feature points on the same face. We used MATLAB function `bwlabel` to label all circles. This function searches and labels all circles from top to bottom and left to right. In case the mobile robot trajectory tracking does not support this function because of its complexity, a similar, simple labeling method is used to replace this function. Because each pattern is composed of four feature points on the faces of the object, the detected feature points are clustered into groups of four feature points. If the number of features is
a multiple of 4, such as 8, 12, 16, 20, or 24, the system could straightforwardly cluster them into two, three, four, five, or six groups, respectively, by having four features in each group. However, there are some cases in which the camera might see a face partially and not all feature points on that face are detected, leading to a number of feature points that are not multiples of 4 (e.g., 9, 10, 11, 13, 15). In such a case, the highest multiple of 4 less than the number of detected feature points is used for analysis. For example, if the number of feature points is 13, the system will use only the first 12 points and cluster them into three groups. For each group, the system should seek the closest point for each point and draw the centerline through each point. The centerlines of all four points form the pattern that is required in Section 3.1.2.

3.1.2 Feature matching and correspondence problem for recognizing the face of the object

The patterns of feature points are determined by the angles between feature points. Hence, we applied angle matching to recognize the face of the object. Figure 3 illustrates two vectors that are connected among three points. First, we compute the angle in degrees (d) between two vectors, as in Equation (3):

\[ d = \arccos \left( \frac{\overline{AB} \cdot \overline{BC}}{\| \overline{AB} \| \| \overline{BC} \|} \right) \]  
\[ a = 180 - d \]  

Four points are connected by three lines and there are two angles among these three lines.

We maintain a ground-truth database of angles for patterns on the faces of the object. Because each face has patterns on the upper and lower prism of the object, there are two groups of angles for each face (one for each prism). To solve the feature matching and correspondence problem, we compared the angles from the actual image with the angles in the database. We calculated the sum of absolute error between them and compared it with a defined threshold. The threshold is set to five per group in our experiments. If the error is less than or equal to the threshold, it is an indicator of the correct side or pattern. As mentioned earlier, the camera may view one, two, or three faces simultaneously. In the case of three sides, possible face trios are F8-F1-F2, F1-F2-F3, F2-F3-F4, ..., F7-F8-F1, where \( F_i \) indicates the \( i \)th face. In the case of two sides, possible face pairs are F1-F2, F2-F3, F3-F4, ..., F8-F1. In the case of one side, target patterns are F1, F2, F3, F4, ..., F8. Each case has its own angle database, in which each face has two groups (for each prism) and each group has two angles. The angle database is provided in the Appendix.

The number of circles is used as an indicator of the number of faces seen by the camera. Registering 24, 16, and eight circles are possible indicators of the visibility of three faces, two faces and one face, respectively. Hence, if the number of circles is eight, 16, or 24, we apply feature matching with one, two, or three sides, respectively. In other cases, at least one face is detected partially. If the number of circles is 12, the system first checks with the case with two sides. If it does not match, the first 8 points are selected for feature matching for the case with one side. If the number of circles is 20, the system first checks for feature matching with the case with three sides. Twenty points may not match exactly to three neighboring faces. In such a case, the first 16 points are selected and feature matching is applied for case with two sides. To minimize the processing time in the mobile robot trajectory tracking, we eliminate the number of circles that are 12 and 20.

3.1.3 Experimental results

Figures 4 and 5 show some results from stage 1. Our algorithm works well under various conditions. We detect all circles in the actual images in Figure 4. Our method works even with a blurred image as well. We also tested our algorithm in a variety of cases that could happen in real situations. Figure 5 shows the incomplete outcomes. Sometimes our method failed to extract all of the features or made an incorrect pattern. However, our method can still recognize the face despite the noise or other incomplete faces. These results show that our method yields reliable feature matching from different angles. As an additional experiment in stage 1, we moved the robot...
around the 3D reference object in a 5-ft radius manually and recorded the videos with a frame rate of 10 frames/s as the robot was moving. This experiment was set up to observe real-time image processing in the vision system with continuous frames from nine videos. We compared our results with the SqueezeNet-V1.1 [26] and ResNet-50 [27] convolutional neural networks. In the case of deep learning methods, we randomly classified 75% into training and 25% into testing datasets. Table 1 shows the comparison of the percentage of accuracy rate among SqueezeNet-V1.1, ResNet-50 and our proposed method. Their average percentages of accuracy rate are 91.38%, 95.52% and 96.96%, respectively. These results indicate that our 3D object recognition method and the deep learning methods yield high accuracy rates.

### Table 1: Comparison of accuracy of state-of-art deep learning methods and proposed method

| Videos | SqueezeNet V1.1 | ResNet-50 | Proposed Method |
|--------|-----------------|-----------|-----------------|
| 1      | 91.38           | 93.10     | 94.83           |
| 2      | 90.67           | 97.58     | 96.89           |
| 3      | 91.46           | 95.20     | 96.26           |
| 4      | 90.31           | 96.92     | 98.92           |
| 5      | 91.71           | 92.50     | 97.29           |
| 6      | 90.61           | 95.55     | 97.53           |
| 7      | 93.90           | 98.74     | 97.67           |
| 8      | 91.51           | 95.67     | 97.00           |
| 9      | 90.87           | 94.43     | 96.28           |

3.2 | Three hundred sixty–degree mobile robot pose estimation

The second stage is pose estimation with respect to the reference object. In this stage, we need to determine how the robot is positioned and oriented in the inertial Euclidean space. We propose calculating the pose of the robot using 2D images of reference points on the 3D reference object. In the literature, terms such as PnP or localization problem have been used to describe this problem. The 6DOFs that describe the pose of the mobile robot in 3D space are \( q = [x, y, z, \text{roll}, \text{pitch}, \text{yaw}] \). We propose to calculating the robot pose using Mirage-S pose estimation [3].

We investigated the performance of our method with a real experiment in the laboratory using a high-precision Vicon motion capture system. Vicon is a key player in optoelectronic motion capture systems based on markers [28]. Owing to its ability to provide precise 3D positions and orientations, we used the Vicon results as ground pose truth. We set the mobile robot in 64 different locations around the reference object. We used our proposed 360° vision-based pose estimation method to calculate the actual pose of the robot at the same 64 locations as shown in Figure 6. Then, we calculated the pose error by comparing the Vicon’s ground truth poses with the outcomes of our pose estimation method.

#### 3.2.1 Experimental results

Figures 7 and 8 illustrate the results of the calculated errors, along with Tables 2 and 3. The average translational errors along the \( x \), \( y \), and \( z \) axes show small errors of about 1.0–1.7 cm. The observation showed that the standard deviation of translational error along the \( z \) axis is greater than that of the \( x \) and \( y \) axes; therefore, translational error along \( z \) axis is slightly more distributed compared with that of the \( x \) and \( y \) axes, as shown in Figure 7(a)–(c). Table 3 indicates that the average rotational errors around the \( x \), \( y \), and \( z \) axes show small errors of about 0.7° to 2°, whereas error around the \( x \) axis is greater than that of the \( y \) and \( z \) axes. When we consider the standard deviation of rotational errors, the \( y \) axis yields a greater value than the \( x \) and \( z \) axes. As a result, rotational error around the \( y \)-axis shows more distribution than that of the \( x \) and \( z \) axes, shown in Figure 8(a)–(c).

#### 3.3 | Mobile robot 360° vision-based trajectory tracking control

We demonstrated the real-time capabilities of the proposed method and measured the performance of our method by controlling a mobile robot on a commanded desired path using a closed loop feedback controller. We defined a desired path of the mobile robot as a complete circular motion around the 3D reference object with a constant speed. By integrating the closed loop feedback controller with the proposed vision-based pose estimation, the mobile robot automatically drives on the desired path around the 3D object with constant speed in real time without human control. The vision data are acquired from a camera mounted directly on the mobile robot (eye-in-hand). Visual features are extracted from the image and used by Mirage-S to calculate the actual pose. The closed loop controller uses the error between the desired pose and the actual pose to command the wheel speeds that correct the error. The control law is defined in terms of the 3D Euclidean space. The performance of the controller was studied using
simulations and real setting experiments. The vision-based trajectory tracking method and the robot model’s details are presented. Then, the results of the simulations and the real experiments are reported and discussed.

### 3.3.1 Vision-based trajectory tracking

Figure 9 provides an overview of vision-based trajectory tracking. We define the circular path around the 3D reference object with radius $R$. The motion follows a circular path with a constant angular rate of rotation $\dot{\alpha}$. We calculate the desired 3D pose $\vec{q}^d = [x^d, y^d, \theta]^T$, where $x^d$ and $y^d$ are translational pose parameters and $\theta$ is the rotational pose parameter. We can compute $\vec{q}^d$ as a function of time using Equation (4):

$$
\vec{q}^d = \begin{bmatrix} x^d \\ y^d \\ \alpha t + \frac{\pi}{2} \end{bmatrix}
$$

where $R$ is the circular path’s radius, $t$ is the given time, $\dot{\alpha}$ is the constant angular rate of rotation, and $\alpha = \dot{\alpha} t$. We also calculate the desired velocity vector $(\vec{u}^d = [v^d, \omega]^T)$ from $\vec{q}^d$, as seen in Equation (5):

$$
\vec{u}^d = \begin{bmatrix} v^d \\ \omega \end{bmatrix} = \begin{bmatrix} \sqrt{(\dot{x}^d)^2 + (\dot{y}^d)^2} \\ \dot{\alpha} \end{bmatrix}
$$

where $\dot{x}^d = \frac{dx^d}{dt} = -R(\sin \alpha)\dot{\alpha}$, and $\dot{y}^d = \frac{dy^d}{dt} = R(\cos \alpha)\dot{\alpha}$. The desired 3D pose $\vec{q}^d$, $\vec{u}^d$, and actual 3D pose $\vec{q}^m$ are sent to the position-based controller to calculate $\vec{u} = [v, \omega]^T$, whereas $v$ and $\omega$ are the translational and angular velocities, respectively. Velocity vector $\vec{u}$ is transferred to the robot model to compute 3D pose $\vec{q}$ by integrating Equation (6):

$$
\vec{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cos \theta \\ v \sin \theta \\ \omega \end{bmatrix}
$$

The mobile robot moves to pose $\vec{q}$ and the camera will take image $\vec{p}$ as the feedback control signal. Then, the feature extraction unit will detect the feature points from the image. The 2D image space of feature points $s$ is sent to the Mirage module to estimate the actual 3D pose $\vec{q}^m$ for the next motion.

### 3.3.2 Two-wheeled nonholonomic mobile robot

A two-wheeled nonholonomic mobile robot is depicted in Figure 10, where $(x, y)$ is the wheel axis center position and $\theta$ is the robot’s orientation. We focus on the kinematic model, which does not consider the mass or force that caused the motion. The kinematic motion equations of a mobile robot are equivalent to a unicycle model. The robots have a nonholonomic constraints, as in the equation:

$$
\begin{bmatrix} -\sin \theta(t) & \cos \theta(t) \end{bmatrix} \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = 0
$$

where $\dot{x}(t) = \frac{dx(t)}{dt}$ and $\dot{y}(t) = \frac{dy(t)}{dt}$. According to this assumption, the robot cannot move in the lateral direction [29]. In our experiments, a mobile robot moves along a desired trajectory using a single camera mounted on the robot body. Each time the robot verifies its actual pose with respect to the desired pose, it moves to reduce the pose error between the actual pose and desired pose.
3.3.3 | Error model of mobile robot

The posture error model from our experiment is given in the global coordinate system. Global posture error \( \vec{E} = \vec{q}^d - \vec{q}^m = [E_x, E_y, E_z]^T \) should be transformed to the local coordinate system as in Equation (8). The error model in the local coordinate system \( \vec{e} = [e_x, e_y, e_\theta]^T \), where \( e_x \) gives the error in the direction of driving, \( e_y \) gives

| Parameters | X     | Y     | Z     |
|------------|-------|-------|-------|
| Minimum    | -11.7755 | -7.1215 | -25.1268 |
| Maximum    | 8.3167  | 16.1208 | 4.7144  |
| Average (\( \mu \)) | -1.1829 | 1.6962  | -1.0467 |
| SD (\( \sigma \)) | 3.5937  | 4.2697  | 7.1377  |
the error in the lateral direction, and \( e_\theta \) gives the error in the orientation [29]:

\[
\begin{bmatrix}
    e_x \\
    e_y \\
    e_\theta
\end{bmatrix} =
\begin{bmatrix}
    \cos \theta & \sin \theta & 0 \\
    -\sin \theta & \cos \theta & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    q^d_x \\
    q^d_y \\
    q^d_\theta
\end{bmatrix} -
\begin{bmatrix}
    q^m_x \\
    q^m_y \\
    q^m_\theta
\end{bmatrix}
\]

(8)

As illustrated in Figure 9, the desired pose \( \overrightarrow{q}^d \), actual pose \( \overrightarrow{q}^m \), and desired velocity \( \overrightarrow{u}^d = [v^d, \omega^d]^T \) are transferred to the position-based control unit to compute the actual velocity vector \( \overrightarrow{u} \). Kannayama et al. [30] presented the control \( \overrightarrow{u} \) to solve the trajectory tracking problem:

\[
\overrightarrow{u} = \begin{bmatrix}
    v \\
    \omega
\end{bmatrix} = \begin{bmatrix}
    v^d \cos e_\theta + \omega^b_x \\
    \omega^d + \omega^b_y
\end{bmatrix}
\]

(9)

where \( \overrightarrow{u}^b = [v^b, \omega^b] \) is the feedback signal to control the mobile robot on the desired path. Blazic [29] proposed the feedback signal control law:

\[
\begin{align*}
    v^b &= k_x e_x \\
    \omega^b &= k_v^d e_y (1 + \frac{e_c^2}{a}) + k_\omega e_x [(1 + \frac{e_c^2 n}{a})]
\end{align*}
\]

(10)

where \( e_x = \sin e_\theta, e_c = \cos e_\theta - 1, a > 2, k > 0, k_x \) and \( k_\omega \) are positive functions, whereas \( n \in \mathbb{Z} \). For practical cases, \( n \) is a small number (usually \(-2, -1, 0, 1 \) or \(2\) are good choices) [29].

| Parameters | roll          | pitch         | yaw          |
|------------|--------------|---------------|--------------|
| Minimum    | -9.7062      | -14.3854      | -12.1252     |
| Maximum    | 10.7991      | 20.4855       | 11.2680      |
| Average    | -1.9604      | -0.9170       | -0.6529      |
| SD (\(\sigma\)) | 4.6604      | 6.0245        | 4.4717       |

**TABLE 3** Rotational errors (deg)

We simulated closed loop control by moving the robot around the reference 3D object with a radius of 5.7 ft (174 cm), 9.8 ft (300 cm) and 19.7 ft (600 cm). Our simulations were implemented in MATLAB, running on MATLAB 2018a on a desktop computer (processor: AMD A4-6300B; memory: 16 GB; hard drive: 930 GB). We developed programs that follow the diagram in Figure 9. The 3DOF desired pose \( \overrightarrow{q}^d = [x, y, \theta]^T \) is defined by the function of time as shown in Equations (11)–(13):

\[
x = R \cos (\dot{\theta} t + \pi)
\]

(11)

\[
y = R \sin (\dot{\theta} t + \pi)
\]

(12)

\[
\theta = \dot{\theta} t - \frac{\pi}{2}
\]

(13)

where \( R = 174 \text{ cm (5.7 ft), 300 cm (9.8 ft) and 600 cm (19.7 ft)}, \) \( \dot{\theta} = 0.1795 \text{ rad (10°)}, \) and the sampling time is 0.1 s. We simulated the mobile robot by moving it in a counterclockwise direction on a circular path. We performed simulations in

**FIGURE 10** Two-wheeled nonholonomic mobile robot

**FIGURE 9** Overview of trajectory tracking
which the mobile robot’s starting position was set outside the initial desired position. We altered the x coordinate by −12, −40, and −85 cm in respect to the simulations, where \( R = 174, 300, \) and \( 600 \) cm. The images are generated by using actual poses, 64 feature points on a 3D model, and intrinsic and extrinsic camera parameters. The Mirage uses the desired pose \( \mathbf{q}^\text{wd} \) and a 2D feature points image to calculate the actual pose \( \mathbf{q}^\text{em} \). The trajectory of the mobile robot results from the simulations indicating accurate tracking for \( R = 174, 300 \) and \( 600 \) cm, as presented in Figure 11(a)–(c), respectively. In addition, we generated a visualization of the simulations, such as the robot moving on the circular path, and 3D reference model in a camera view to verify the proposed method in the simulations.

5 | EXPERIMENTS

After verifying our method through the simulations, we conduct two types of experiments to show the validity of our method. First, we deploy our algorithm on a robot with a Logitech c525 installed Web camera, shown in Figure 12 and evaluate its performance. Second, we compare our method with one of the deep pose learning algorithms, PoseNet.

Before performing pose estimation experiments, the intrinsic and extrinsic camera parameters must be computed. The intrinsic and the extrinsic parameters of the camera were found in two separate stages. In the first stage, the camera calibration toolbox for MATLAB [31] was used to estimate intrinsic parameters \( (f_x, f_y, \alpha_x, \alpha_y) \), where \( f_x, f_y \) are focal lengths and \( (\alpha_x, \alpha_y) \) is the principle point. The calibration toolbox requires a \( 7 \times 9 \) square checkerboard image, as shown in Figure 13(a). An undistorted camera was assumed. In the second stage, extrinsic camera parameters were estimated using a high-precision Vicon motion capture system. Two similar quadrilateral shapes using reflective Vicon markers were placed at the center of the camera and the mobile robot, shown in Figure 13(b). After results were obtained from the lab, the optimal rotation and translation of the camera or extrinsic parameters \([x, y, z, \theta, \phi, \psi]\) relative to the mobile robot were calculated.

5.1 | Real-world experiments

First, we modified the simulation code. We replaced the simulated camera and robot model with the actual camera and robot. The modified code was integrated with Simulink Real Time running on MATLAB 2018a with the same desktop computer used to perform the simulations. Second, we established a connection between the host computer (desktop computer) and the embedded target computer (processor: Intel Atom 1.8 GHz; memory: 4 GB; hard drive: 80 GB), which is

![FIGURE 11](image1.png) Trajectory of mobile robot results in simulations

![FIGURE 12](image2.png) Nonholonomic mobile robot employed in real experiments
the computer on the robot, via the Ethernet cable. Third, we compiled, built, and transferred the execution code in a binary format to the embedded target computer. Finally, we ran the code on the embedded target computer and obtained results from the Simulink files scopes to process and interpret the data. In addition, the robot’s wheel speed control hardware executes the closed loop controller’s outputs. To function, the robot speed controller hardware requires the velocity of the left and right wheel. As mentioned in Section 3.3.3, the feedback controller calculated the actual velocity vector \( \vec{v} = [v, \omega] \), where \( v \) = velocity (cm/s) and angular velocity \( \omega \) (rad/s). Therefore, \( v \) and \( \omega \) must be converted to the velocities of the left and right wheels. Equation (14) shows the formulas to calculate velocities \( (v_L, v_R) \) of the left and right wheels:

\[
\begin{align*}
v_L &= v - \left( \frac{T}{2} \right) \omega \\
v_R &= v + \left( \frac{T}{2} \right) \omega
\end{align*}
\]  

where \( T \) is the distance between the centers of the wheels.

We performed experiments on closed loop control using vision signals. As shown in Figure 12, the mobile robot was commanded to move along the circular path. The starting and ending poses were \([-174, 0, -\frac{\pi}{2}]^T\); the camera faced the 3D model as the robot headed on the path. The controller gain was selected to be \( k_x = 2.5, k = 0.7, k_2 = 3.5, a = 3, \) and \( n = 2 \). The robot started at 0 s and began moving along the circular track. The robot continued on the circular path and passed the end point, which is the same as the starting point, at approximately 35 s. We used Simulink to measure the processing time for image acquisition, image processing, Mirage pose estimation, and control blocks; it yielded 5.345, 25.128, 57.982, and 8.266 ms, respectively. The total processing time was estimated at about 100 ms. Therefore, the real-time program was set to run at 10 Hz.

The trajectory of the mobile robot result for the experiment is presented in Figure 14. In addition, the controller inputs for the robot’s motion are presented in Figure 15. The translational velocity \( v \) varies with a root-mean-square (RMS) equal to 31.2430 cm/s, as in Figure 15(a). Angular velocity \( \omega \) showed a variation with an RMS of 0.1804 rad/s, as in Figure 15(b). \( v_L \) and \( v_R \) in Figure 15(c) showed variations in the RMS of 27.4068 and 35.1082 cm/s, respectively. The experiment’s results show that the mobile robot successfully moved on the desired trajectory using the proposed method.

5.2 Comparison with well-known perspective-n-point methods and PoseNet

Well-known PnP methods such as the Pose from Orthography and Scaling with Iteration (POSIT) [32], Lu-Hager-Mjolness (LHM) [33], and the direct least-squares (DLS) method [34], were tested along with PoseNet [35] and Mirage-S.

POSIT [32] is a well-known and popular method that approximates the perspective projection with a scaled orthographic projection and finds the rotation matrix and translation vector of the object by repeatedly solving a linear system. DLS [34] estimates pose without iterations by using initiation parameters. DLS employs the camera measurement equation and formulates a non-linear least-squares cost function. LHM [33] uses object space collinearity error and derives an iterative method to compute the orthogonal rotation matrix with a global convergence.

PoseNet is the first learning-based architecture that introduces the idea of regressing the absolute pose with a deep architecture. Kendall et al. [35] suggested learning the localization pipeline in its entirety. Starting from an RGB image, a network can learn to regress the camera pose in an end-to-end supervised manner. PoseNet is a modification of a GoogLeNet architecture [36], a deep convolutional network with 22 layers. The regression architecture can be abstracted into three components: (1) an encoder that generates the visual encoding vector.

**Figure 13** (a) A 7 × 9 square checkerboard; (b) Markers on mobile robot and camera

**Figure 14** Trajectory of mobile robot result in real experiment
(v), (2) a localizer that outputs (produces) the localization features vector (u) and (3) a regressor that regresses pose (p). PoseNet was trained with stochastic gradient descent to minimize the pose error.

The experiments were set up in the Vicon lab to train PoseNet for our purposes. We moved the robot around the 3D reference object manually within a radius of 4, 5 and 6 ft. While moving the robot, we recorded a video with a camera on the robot. Simultaneously, the pose data are recorded by Vicon at a frame rate of 60 frames/s. The precise 3D positions and orientations from Vicon were used as ground truth data. After that, we extracted all frames from the video and obtained the pose data from Vicon. Because they were recorded simultaneously, the frame or image and pose data match properly. POSIT, LHM, DLS and Mirage-S used the images and the pose data from Vicon as the desired poses to calculate the estimated poses. In the case of PoseNet, we randomly classified 80% of images with pose data into training and 20% of data into testing datasets. The code of the TensorFlow PoseNet framework was provided by Dr. Mukherjee from the University of Alabama in Huntsville. This code is modified from GitHub’s website and published by Kent Sommer [37]. There are two subprocesses for PoseNet: (1) Training: We trained the network with the training dataset and the initial weights. The training process was run for 100, 1000, and 10,000 iterations, in which each iteration had a batch size equal to 25. The training process yielded the adjusted weights used for testing. (2) Testing: We tested the network with the testing dataset and the adjusted weights. We compared the estimated pose and ground truth data by computing the absolute value of the errors on translation and rotation. Tables 4, 5, and 6 present the median translational and rotational errors of PoseNet, POSIT, LHM, DLS and Mirage-S. In a comparison of the results of the well-known PnP methods and Mirage-S, the outcomes indicate that Mirage-S performed with minimal errors (Table 4).

PoseNet yields the minimum value while running 10,000 iterations for each radius. The median translational error is about 3–5 cm and the median rotational error is around 3°. PoseNet also yields consistent errors for different radii. This indicates that PoseNet produces robust results. On the other hand, Mirage-S yields a median translational error of around 5–6 cm, whereas its median rotational error is close to the rotational error.

**FIGURE 15** Controller inputs for the robot’s motion. (a) Velocity (v); (b) omega (ω); and (c) Left and right wheel velocities (cm/s)

**TABLE 4** Median errors within radius of 4 ft (1.2192 m)

| Methods  | Translational Error (m) | Rotational Error (degrees) |
|----------|-------------------------|----------------------------|
| PoseNet  | 0.464001                | 5.759196                   |
|          | 0.175353                | 3.320341                   |
|          | 0.032449                | 2.566116                   |
| POSIT    | 0.562689                | 5.231871                   |
| LHM      | 0.675482                | 7.812309                   |
| DLS      | 0.652341                | 6.954327                   |
| Mirage-S | 0.055164                | 3.412325                   |

Abbreviations: DLS, direct least-squares; LHM, Lu-Hager-Mjølness; POSIT, pose from orthography and scaling with iteration.
of PoseNet. Mirage-S also yields consistent errors for different radii. Results from the experiments show that PoseNet and Mirage-S yield low median errors (Tables 5 and 6).

When we considered the processing time, PoseNet performs pose estimation at around 5 ms for testing, whereas Mirage-S completes pose estimation in about 100 ms. The major advantage of Mirage-S compared with PoseNet is that errors can be tracked for Mirage-S. Because PoseNet uses a deep learning architecture, it is hard to verify what it learned or did not. PoseNet is recommended to be trained on a computer with a GPU, for performance reasons, whereas Mirage-S does not require a GPU or a high-end computer.

PoseNet works well when the training and testing datasets are from the same source data. Sattler et al. [38] proposed the limitations of a convolutional neural network–based absolute camera pose regression technique using different training and testing datasets. The estimated pose results from PoseNet are significantly noisier and do not lie on the testing poses; instead, they lie on the training poses [38]. In addition, the absolute camera pose regression technique is not guaranteed to generalize from the training data in practical scenarios [38].

### 6 CONCLUSIONS

We proposed a method for real-time vision-based localization in the full 360° area around a 3D reference object. There are three main subproblems. First is the 360° object recognition problem. To solve this issue, we applied a computer vision solution to recognize a reference 3D object from all possible 360° views. The reference 3D object is defined by different faces and patterns in which the robot can see from anywhere around it by using feature extraction and feature matching based on angles from patterns on the object. The experimental results in a variety of cases show that the solution can recognize the unique face(s) despite noise or other incomplete patterns, and it yields a high accuracy rate. Second is the 360° pose estimation problem, in which we use Mirage pose estimation with a single camera to calculate the pose of a robot at all possible 360° locations around the reference 3D object. The results of the experiment with the camera poses indicated small translational and rotational pose errors. Third is the application of 360° localization. We integrated 360° localization with a closed loop real-time trajectory tracking controller. The mobile robot is expected to move on a desired path covering the whole 360° area around the 3D reference object. The simulation results indicated minimum average errors on 3DOF (x axis, y axis, and heading). The results of experiments show that the mobile robot successfully moves along the desired circular track. The simulations and real experiments confirmed the feasibility and performance of the proposed approach.

**ORCID**

Khomsun Singhirunnusorn 🐦 https://orcid.org/0000-0003-2378-6379
REFERENCES

1. Kazemi, M., Gupta, K., Mehrandezh, M.: Path-planning for visual servoing: a review and issues. In: Chesi, G., Hashimoto, K. (eds.) Path-Planning for Visual Servoing: A Review and Issues, pp. 189–207. Springer London, London (2010). https://doi.org/10.1007/978-1-84996-089-2-11

2. Dine, S., Fahimi, F., Aygun, R.: Vision-based trajectory tracking for mobile robots using mirage pose estimation method. IET Comput. Vis. 10(5), 450–458 (2016)

3. Singhirunnusorn, K., Fahimi, F., Aygun, R.: Single-camera pose estimation using mirage. IET Comput. Vis. 12(5), 720–727 (2018)

4. Dine, S., Fahimi, F., Aygun, R.: Mirage: an O(θ²) time analytical solution to 3D camera pose estimation with multi-camera support. Roboticsa. 35(12), 2278–2296 (2017)

5. Dine, S., Aygun, R.S., Fahimi, F.: An efficient pose recovery algorithm for vision based trajectory tracking applications. In: 2018 International Conference on Information and Computer Technologies, pp. 107–110. ICICT (2018). https://doi.org/10.1109/ICICT2018.8356850

6. Durrant Whyte, H., Bailey, T.: Simultaneous localization and mapping: part I. IEEE Robot. Autom. Mag. 13(2), 99–110 (2006)

7. Ischara, A., et al.: From structure-from-motion point clouds to fast location recognition. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2599–2606. (2009). https://doi.org/10.1109/CVPR.2009.5206587

8. Wang, Y., Zhang, W., An, P.: A survey of simultaneous localization and mapping on unstructured lunar complex environment. AIP Conference Proceedings. 1890(1), 030010. (2017). https://aip.scitation.org/doi/abs/10.1063/1.5005198

9. Piao, J.C., Kim, S.D.: Adaptive monocular visual–inertial SLAM for real-time augmented reality applications in mobile devices. Sensors. 17(11), 2567 (2017). http://www.mdpi.com/1424-8220/17/11/2567

10. Shi, Y., et al.: GPS motion by similarity averaging. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 119–136. (2019). https://doi.org/10.1109/CVPR.2019.882201/13/1/119

11. Su, W., et al.: UAV pose estimation using ir and rgb cameras. In: 2017 IEEE/SICE International Symposium on System Integration (SII), pp. 151–156. (2017). https://doi.org/10.1109/SII.2017.8279204

12. Shi, Y., et al.: GPS-supported visual SLAM with a rigorous sensor model for a panoramic camera in outdoor environments. Sensors. 13(1), 119–136 (2013). http://www.mdpi.com/1424-8220/13/1/119

13. Burschka, D., Mair, E.: Direct pose estimation with a monocular camera. J. Math. Imag. Vis. 45(2), 186–198 (2013)

14. Camposeco, F., et al.: Hybrid camera pose estimation. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 136–144. (2018)

15. Li, Y., et al.: A novel on-board UAV pose estimation system based on monocular camera. In: 2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC). 02, pp. 504–508. (2016). https://doi.org/10.1109/IHMSC.2016.180

16. Burschka, D., Mair, E.: Direct pose estimation with a monocular camera. J. Math. Imag. Vis. 45(2), 186–198 (2013)

17. Bouguet, J.Y., Bouguet, J.Y. (eds.) Camera Calibration Toolbox for MATLAB. The Mathworks, Inc., Natick (2018)

18. Bouguet, J.Y., Bouguet, J.Y. (eds.) Camera Calibration Toolbox for MATLAB. The Mathworks, Inc., Natick (2018)

19. Petersen, T.: A Comparison Of 2D-3D Pose Estimation Methods. Aalborg University, Aalborg (2008)

20. Guo, Y.: A novel solution to the P4P problem for an uncalibrated camera. J. Math. Imag. Vis. 45(2), 186–198 (2013)

21. Tang, J., Chen, W-S., Wang, J.: A novel linear algorithm for P3P problem. Appl. Math. Comput. 205(2), 628–634 (2008)

22. Zheng, Y., et al.: Revisiting the PnP problem: a fast, general and optimal solution. In: Proceedings of the 2015 IEEE International Conference on Computer Vision, pp. 2344–2351. Sydney (2015)

23. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. 9(1), 62–66 (1979)

24. MATLAB: 2017b: MATLAB (eds.) MATLAB version 9.3.0.713579 (R2017b). The Mathworks, Inc., Natick (2017)

25. Balter, A., Balter, A. (eds.) How To Calculate Circularity. https://sciencing.com/calculate-circularity-5138742.html (2017). Accessed 10 May 2021

26. Meriaux, P., et al.: A study of vicon system positioning performance. Sensors. 17(7), 1591 (2017)

27. Zheng, Y., et al.: Revisiting the PnP problem: a fast, general and optimal solution. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. (2015). https://doi.org/10.1109/CVPR.2015.90

28. Meriaux, P., et al.: A study of vicon system positioning performance. Sensors. 17(7), 1591 (2017)

29. Blažič, S.: A novel trajectory-tracking control law for wheeled mobile robots. Robot. Autonom. Syst. 59(11), 1001–1007 (2011). https://doi.org/10.1109/ROBOT.2011.606006

30. Bouguet, J.Y., Bouguet, J.Y. (eds.) Camera Calibration Toolbox for MATLAB. The Mathworks, Inc, Natick (2018)

31. Bouguet, J.Y., Bouguet, J.Y. (eds.) Camera Calibration Toolbox for MATLAB. The Mathworks, Inc. (2018)

32. Dementhon, D.F., Davis, L.S.: Model-based object pose in 25 lines of code. Int. J. Comput. Vis. 15(1), 123–141 (1995)

33. Lu, C-P., Hager, G.D., Mjolsness, E.: Fast and globally convergent pose estimation from video images. IEEE Trans. Pattern Anal. Machine Intell. 22(6), 610–622 (2000)

34. Hesch, J.A., Roumeliotis, S.I.: A direct least-squares (DLS) method for PnP. In: 2011 IEEE International Conference on Computer Vision (ICCV), 383–390 (2011)

35. Kendall, A., Grimes, M., Cipolla, R.: Convolutional Networks for Real-Time 6-DOF Camera Relocalization. CoRR (2015). abs/1505.07427. http://arxiv.org/abs/1505.07427

36. Szegedy, C., et al.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9. (2015). https://doi.org/10.1109/CVPR.2015.7298594

37. Botts, M., et al.: Understanding the limitations of cnn pose estimation methods. In: 2019 IEEE/RSJ Conference on Intelligent Robots and Systems, 1, pp. 384–389. (1990). https://doi.org/10.1109/ROBOT.1990.126006

38. Seth, S., et al.: Understanding the limitations of cnn-based absolute camera pose regression. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3297–3306. (2019)

How to cite this article: Singhirunnusorn K, Fahimi F, Aygun R. A single camera 360-degree real time vision-based localization method with application to mobile robot trajectory tracking. IET Cyber. Syst. Robot. 2021; 14. https://doi.org/10.1049/esy2.12021
APPENDIX

Angle database in degree for feature matching with one, two, and three sides, respectively.

**TABLE A1** Angles database for one-side case

| Faces | Group 1        | Group 2        |
|-------|----------------|----------------|
| 1     | (173.68, 176.89) | (178.75, 178.48) |
| 2     | (131.17, 132.39) | (132.63, 129.93) |
| 3     | (102.88, 98.54)  | (86.75, 83.33)  |
| 4     | (147.25, 179.92) | (126.65, 177.48) |
| 5     | (154.90, 146.26) | (143.51, 133.13) |
| 6     | (177.55, 144.31) | (178.63, 128.99) |
| 7     | (140.19, 137.21) | (125.27, 123.77) |
| 8     | (151.18, 120.48) | (82.82, 85.71)  |

**TABLE A2** Angles database for two-sides case

| Faces | Group 1        | Group 2        | Group 3        | Group 4        |
|-------|----------------|----------------|----------------|----------------|
| 1-2   | (173.96, 176.79) | (178.77, 178.55) | (136.14, 134.87) | (135.51, 132.60) |
| 2-3   | (132.38, 133.58) | (132.53, 130.06) | (109.78, 106.11) | (70.89, 86.44)  |
| 3-4   | (105.78, 101.04) | (72.34, 86.43)  | (151.05, 179.86) | (124.38, 176.94) |
| 4-5   | (146.78, 179.69) | (124.80, 177.85) | (157.19, 149.17) | (146.70, 130.42) |
| 5-6   | (151.13, 146.35) | (143.40, 134.09) | (176.61, 145.01) | (179.83, 121.77) |
| 6-7   | (177.72, 143.48) | (178.40, 128.44) | (144.35, 138.68) | (122.71, 127.90) |
| 7-8   | (140.19, 137.21) | (125.27, 123.77) | (116.73, 120.12) | (79.75, 80.28)  |
| 8-1   | (113.25, 118.71) | (79.26, 82.32)  | (174.94, 177.72) | (179.92, 179.48) |

**TABLE A3** Angles database for three-sides case

| Faces | Group 1        | Group 2        | Group 3        | Group 4        | Group 5        | Group 6        |
|-------|----------------|----------------|----------------|----------------|----------------|----------------|
| 8-1-2 | (132.78, 138.26) | (111.88, 116.10) | (173.54, 177.36) | (178.57, 179.34) | (142.03, 140.16) | (142.78, 140.49) |
| 1-2-3 | (175.39, 177.48) | (178.40, 178.55) | (131.05, 131.83) | (132.08, 129.61) | (118.11, 115.09) | (79.63, 95.63)  |
| 2-3-4 | (137.41, 139.62) | (139.64, 138.01) | (102.29, 98.04)  | (65.50, 80.25)   | (162.84, 179.82) | (137.16, 177.60) |
| 3-4-5 | (117.46, 112.87) | (89.30, 102.60)  | (147.34, 179.85) | (122.61, 176.96) | (162.8, 157.98)  | (151.90, 136.69) |
| 4-5-6 | (150.63, 179.84) | (133.78, 178.39) | (152.77, 144.54) | (138.75, 127.86) | (177.26, 155.89) | (179.18, 131.72) |
| 5-6-7 | (159.32, 151.41) | (151.03, 142.23) | (176.81, 141.70) | (179.19, 121.64) | (158.01, 151.87) | (136.05, 142.91) |
| 6-7-8 | (178.58, 148.58) | (178.35, 138.16) | (140.12, 135.39) | (121.19, 123.76) | (135.86, 137.41) | (102.97, 101.89) |
| 7-8-1 | (143.73, 141.54) | (131.95, 128.07) | (111.10, 115.52) | (74.31, 75.72)   | (177.76, 178.92) | (179.35, 179.39) |