SN Computer Science (2024) 5:748
https://doi.org/10.1007/s42979-024-03090-y

ORIGINAL RESEARCH

Fiducial Markers and Particle Filter Based Localization and Navigation Framework for an Autonomous Mobile Robot

Muhammad Shahab Alam[1] · **Ali Ihsan Gullu**[1] · **Ahmet Gunes**[1]

Received: 9 December 2023 / Accepted: 24 June 2024

Abstract

For collision-free autonomous navigation, pose estimation plays a pivotal role as it enables the robot to localize itself in the environment in which it is operating. A broad range of sensors are available for position and orientation estimation which are highly accurate and precise, but are contrariwise expensive. This work introduces a simple yet a robust method for estimating the robot pose via ArUco markers and particle filter. The proposed approach acquires position information from the *tvec* (translation vector) of the detected ArUco marker's coordinate frame. Because of the flickering that incurs as a result of various factors in the detection of the ArUco markers, the measurement error and inaccuracy in establishing a feasible robot heading vector from the ArUco marker becomes more pronounced. Therefore, instead of acquiring the orientation information head-on from *rvec*, forward filtering-backward smoothing recursions are used for generating the heading vector (and consequently for spawing steering commands) based on the observations acquired from the camera. The Q matrix values are chosen considering anticipated process noise from the target's position, speed, acceleration, heading angle, and turning rate. For the R matrix, values are selected based on the deviation in target states over time. Increasing the number of smoothing levels substantially reduces estimation errors. The smoothing filter proves to be crucial for correcting unexpected sensor information errors caused by environmental lighting conditions, system network data transfer lag, and unstable number of frames per second (fps). This study integrates the pure pursuit (PP) algorithm into the navigation framework for path following. Discretized PID control equations are employed to eliminate errors between desired and actual heading and speed of the robot. The system is simulated in a Gazebo environment and implemented on a 4-wheeled Ackermann drive mobile robot. Performance of the proposed method is evaluated using average speed, position, and heading errors. The findings showcase efficacy and robustness of the proposed method.

Keywords Fiducial markers · Localization · Mobile robots · Navigation · Particle filters

Introduction

Owing to the rapid technological advancements in artificial intelligence, sensing systems, and processing hardware, the capabilities of robots are enhancing at a great pace [1]. The increasing intelligence and robustness of robotic systems has evoked their adoption in a wide range of sectors, including manufacturing industry, logistics, defense, health care, and agriculture. In the past, mostly fixed base robots were used for performing repetitive and rule-based tasks in industries. But with the developments in technology, robots started exhibiting autonomy and locomotion in the form of autonomous mobile robots (AMRs), which further extended their capabilities beyond machine tending applications. AMRs possess the capability to navigate between distinct locations, effectively circumventing obstacles and, subsequently, determining optimal trajectories to fulfill assigned tasks. They rely on a complex system of computing hardware, sensors, software components, and control algorithms, to perceive its environment, detect obstacles, and plan appropriate paths [2]. The sensors commonly include cameras, LIDARs, ultrasonic sensors, Global Navigation Satellite System (GNSS), and infrared sensors, among others, and are critical to the

✉ Muhammad Shahab Alam
shahab@gtu.edu.tr

Ali Ihsan Gullu
aliihsangullu@gtu.edu.tr

Ahmet Gunes
ahmetgunes@gtu.edu.tr

[1] Defense Technologies Institute, Gebze Technical University, Gebze 41400, Kocaeli, Turkey

Published online: 01 August 2024

successful operation of AMRs [3]. Moreover, for solving the perception, planning, and control tasks, extensive computing hardware is required. The cost of these sensors, and computing hardware can be a significant barrier to their widespread adoption.

Additionally, the sine quo non for performing any task by an AMR is a robust navigation and localization system. A plethora of studies have been conducted to address this crucial challenge [4–9]. The most commonly used methods for localization and navigation include landmark-based approaches, odometry-based approaches, and simultaneous localization and mapping (SLAM) techniques.

Odometry-based methods [10–12] rely on motion sensors, such as wheel encoders, to estimate precise position of the robot based on its movement. However, over time, errors can accumulate due to sensor noise, slip, or other factors, leading to inaccuracies in localization and navigation. Odometry-based methods are less accurate than other techniques, such as SLAM or landmark-based approaches. This is because they rely solely on sensor data and neglect the robot's surrounding environment or external factors that may impact its movement. These methods are highly dependent on the terrain or surface over which the robot is moving. Changes in surface texture or incline can significantly impact the accuracy of odometry-based localization. Even small errors in odometry measurements can cause the robot's position estimate to drift over time. This can make it challenging for the robot to navigate accurately, particularly in dynamic or changing environments. Several reviews on Odometry-based techniques can be found in [13–15].

SLAM techniques [16–18] techniques synergistically integrate odometry-based methods with landmark-based approaches to generate a comprehensive map of the environment while concurrently estimating the precise position of the robot. Expensive laser scanners are generally used for the construction of the environment map. These techniques require significant computational power and resources, particularly when mapping large or complex environments. This can limit the real-time performance of the AMR, leading to delays or inaccuracies in localization and navigation. Various detailed surveys on SLAM techniques can be found in [19, 20].

Landmark-based approaches [21–23] rely on identifying specific features or markers in the environment and using them as reference points to estimate the robot's position. These approaches are particularly immune to the disadvantages that incur in case of the other sensors, demand minimal computing resources, are economically viable, and simple to set up. However, Landmark-based approaches can be challenging to scale for larger environments, particularly if there are few distinguishable features or landmarks. For this reason, these methods may not be suitable for localization in open or unstructured environments. Moreover, if the robot's sensors are not calibrated or the landmarks are not accurately identified, localization accuracy can suffer.

The main focus of this study is to explore the fiducial marker-based approach, which typically requires the installation of specific markers in the environment. Several types of artificial fiducial markers have been developed, including ARToolkit [24], ARTag [25], AprilTag [26], ArUco [27], TRIP [28], RUNE-Tag [29], ChromaTag [30], CCTag [31], and CALTag [32], etc. Each variant of fiducial marker exhibits its own set of distinct characteristics, strengths, and weaknesses. The selection of the appropriate fiducial marker type is contingent upon the specific requirements of each individual application. Recent studies show promising results for these marker-based systems for localization and navigation. However, there are still some limitations to these approaches, such as accuracy, scalability, and computational efficiency, that need to be addressed to enable these systems to become more robust and accurate.

In this paper a cost-effective markers-based localization and navigation approach is presented that relies on artificial fiducial markers and a vision sensor. The proposed system employs detection algorithms to identify the start and goal points, detect obstacles, and create an environment map. It then proceeds to plan a path between the start and goal points, generate motion commands for tracking the planned path, and estimate the robot's position and orientation using particle filters (PF).

The subsequent sections are organized as follows: "Methodology" presents a comprehensive description of the methodology employed, "Implementation of the Framework" presents the results and corresponding discussions, and "Experimental Validation" concludes the work with a summary of findings.

Methodology

This research work presents a mobile robot localization and navigation framework using ArUco markers and PF. The markers, are strategically positioned at the goal point and atop the mobile robot, serving as reference points to establish the robot's global position relative to the global reference frame. A vertically oriented camera module is installed on the ceiling or on a drone to capture a top-down view of the environment, encompassing the robot, markers, and obstacles, as illustrated in Fig. 1. A comprehensive description of each of the module involved in the system is provided in the subsequent sections.

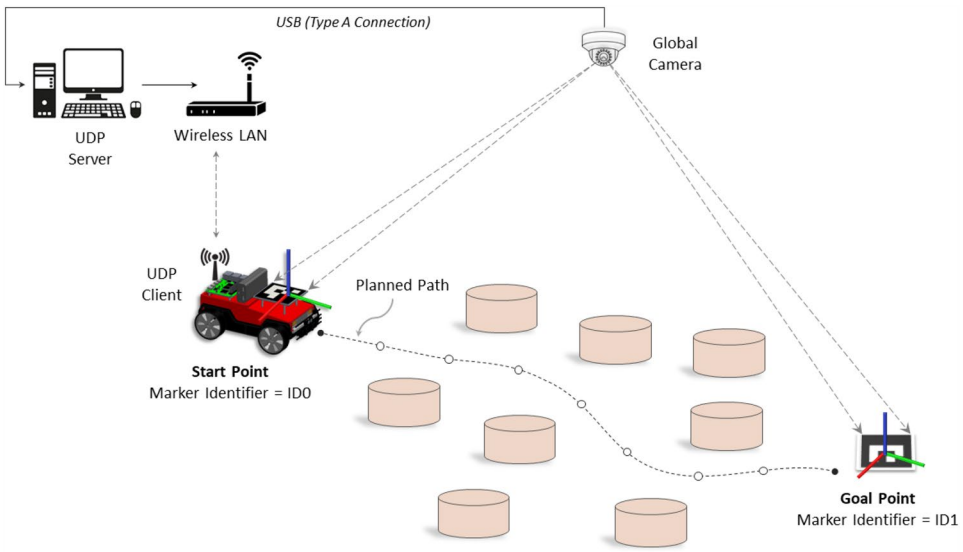

Fig. 1 Overall framework of the system

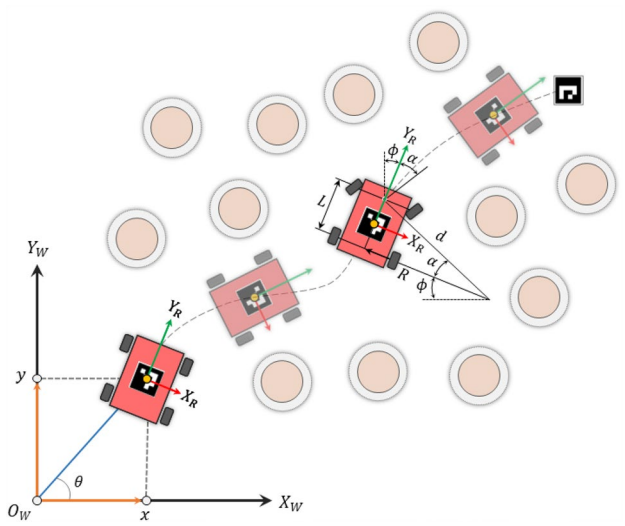

Fig. 2 Environment and robot model

Robot Pose Estimation and Control

The robot's pose in the environment is established by defining a relationship between an absolute global world reference frame $\{W\}$ in which the robot operates, and an egocentric local robot reference frame $\{R\}$ affixed to the robot, as shown in Fig. 2. A point P on robot at the center of the ArUco marker is chosen as the robot's reference point. At the origin point P, the robot's local reference frame R is positioned, featuring the two axes X_R and Y_R. This local reference frame serves as the basis for the robot's coordinate system within the environment. Position of point P in the global reference frame $\{W\}$ is established by x and y coordinates, and the angular difference θ between the global and local reference frames. Pose of the robot in world frame $\{W\}$ and local frame $\{R\}$ is represented by a vector with three components as $\xi_W = [x\ y\ \theta]^\intercal$ and $\xi_R = [0\ 0\ 0]^\intercal$, respectively. A correlation between the global and the local reference frames for establishing pose of the robot is specified by the transformation matrix R, which converts motion from the global to the local reference frame. This operation is dependent on the value of θ.

An Ackerman drive mobile robot model, shown in Fig. 2 is chosen in this study. The rear wheels are the driving wheels and the front wheels are the steering wheels. The steering angle α, measured in the counter-clockwise direction, represents the angle at which the front wheels deviate from the robot's longitudinal axis Y_R. Similarly, the heading angle ϕ, also measured in the counter-clockwise direction, signifies the angle between the longitudinal axis Y_R and the Y_W axis. The intersection point of the two lines passing through the wheel axes serves as the instantaneous center, dictating the rotation point of the robot.

The kinematic equations that govern motion of the robot in robot coordinates are comprised of

$$\begin{bmatrix} v_x \\ v_y \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 \\ v \\ \frac{v}{L}\tan(\alpha) \end{bmatrix} \tag{1}$$

Upon conversion to earth coordinates, these equations transform into

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} -v\sin(\phi) \\ v\cos(\phi) \\ \frac{v}{L}\tan(\alpha) \end{bmatrix} \tag{2}$$

To ensure smooth and gradual changes in the steering angle and velocity, the derivatives or rates of these variables are

adopted as the control signals, i.e., $\dot{\alpha} = u_1$ and $\dot{v} = u_2$. The discrete time model of the model, with a setting $t = kTs$ (where $k = 0, 1, 2, ..., n$) and sampling interval $\Delta t = Ts$ (for evaluating at discrete-time instants), can be expressed as

$$x((k+1)T_s) = x(kT_s) - \left(\frac{v_R(kT_s) + v_L(kT_s)}{2}\right) T_s sin\phi(kT_s) \tag{3}$$

$$y((k+1)T_s) = y(kT_s) + \left(\frac{v_R(kT_s) + v_L(kT_s)}{2}\right) T_s cos\phi(kT_s) \tag{4}$$

$$\phi((k+1)T_s) = \phi(kT_s) + T_s \frac{v(kT_s)}{L} tan\alpha(kT_s) \tag{5}$$

$$\alpha((k+1)T_s) = \alpha(kT_s) + T_s u_1(kT_s) \tag{6}$$

$$v((k+1)T_s) = v(kT_s) + T_s u_2(kT_s) \tag{7}$$

Image Processing for Generation of Environment Map

An initial step in robot navigation involves providing the robot with a map that accurately represents the environment in which it operates. For generating a map of the environment, the obstacles within the environment, as well as the start and goal points needs to be detected. Before delving into this, a geometric relationship between the 3D world coordinate and the camera's 2D image plane is established. This is done by utilizing the popular Zhang's [33] camera calibration method that uses a checkerboard. First, several images of a checkerboard of known size and structure at different orientations are captured. Next, the intrinsic, extrinsic parameters, as well as the lens distortion parameters of the camera are computed using a camera calibration algorithm and OpenCV library. The relationship between the world coordinates, denoted by X, and image or pixel coordinates, denoted by x, is established through perspective transformation as

$$x = K\left[R\ t\right]X \tag{8}$$

where R denotes the rotation matrix, t represents the translation vector, and K represents the camera matrix which encapsulates the intrinsic parameters. The matrix K is given by

$$K = \begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix} \tag{9}$$

where (f_x,f_y) represents the focal length, (c_x, c_y) represents the principal point offset, and s represents the axis skew. Hence, the parameters of the 3×4 matrix M of the perspective model for camera calibration are estimated as

$$M = K\left[R\ t\right] \tag{10}$$

Next, a global grid map of the real environment is generated while considering physical size of the actual robot. The global camera captures image of the scene, and the captured image is then passed to the computer/PC Laptop which utilizes obstacle and ArUo marker detection algorithms. The obstacles are detected based on their color. Next, the coordinate points of the vertices of the obstacles and the starting and destination points from the ArUco markers are extracted, and a map generator creates a complete map of the environment, as shown in Fig. 3.

Fixed cell decomposition approach is adopted for building a discretized occupancy grid map that comprises of two-dimensional arrays of cells of arbitrary resolution. Each grid cell is assigned a binary state: 0 to the empty cell and 1 to the occupied or full cells. As the actual robot is not a point, therefore, the physical size of the robot is taken into account

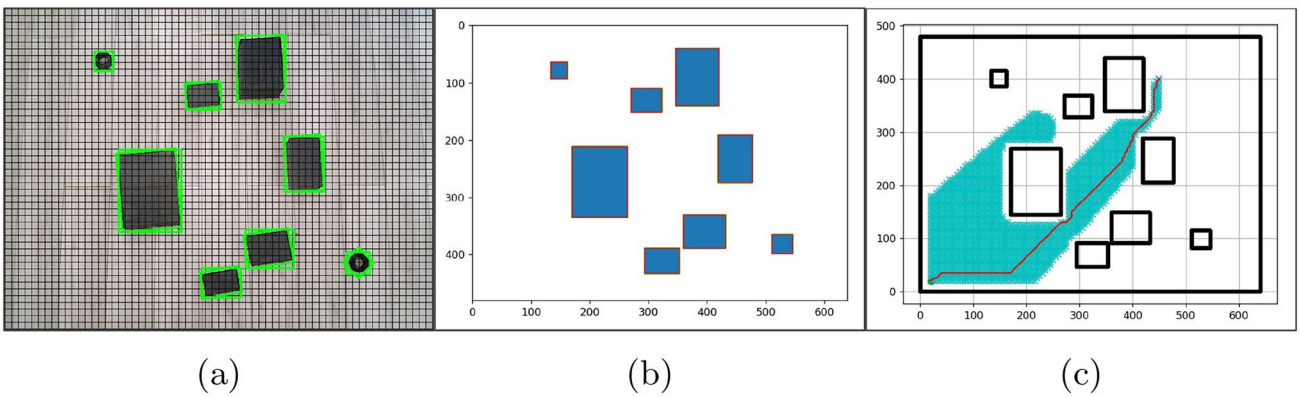

Fig. 3 Map generation: **a** detected obstacles, **b** generated map, and **c** planned path

by performing obstacle inflation according to the physical size of the robot while generating the map. The purpose of this inflation is to introduce a safety margin around obstacles, creating buffer zones between the robot and the obstacles within the environment. A cell size of approximately 10 cm is chosen for the grid map. Model of the occupancy gird cell is represented by the matrix C

$$C = \begin{bmatrix} C_{0,0} & C_{1,0} & \cdots & C_{m,0} \\ C_{0,1} & C_{1,1} & \cdots & C_{m,1} \\ \vdots & \vdots & \vdots & \vdots \\ C_{0,n} & C_{1,n} & \cdots & C_{m,n} \end{bmatrix} \tag{11}$$

where m and n indicate the maximum x and y indices of the occupancy grid.

After map generation, the markers positioned within the field of view of the imaging system, are detected. In this study, ArUco markers [27] are chosen for the proposed localization and navigation system. Among the three available mounting options, i.e., wall-mounting, floor-mounting, or ceiling-mounting, the floor mounting option is selected wherein one marker is placed on top of the robot, and the other marker is placed on the goal point. The robot/starting point is tagged with a marker ID 0, and the goal point is tagged with a marker ID 1. Each marker has a physical size of 12×12 cm, and a matrix size of 4×4.

For detecting the marker, the following steps are performed: (1) the input RGB image is converted to a grayscale image, and then into a binary image using adaptive thresholding, (2) contours are detected, and all the invalid contours are filtered out, (3) polygonal approximation is performed and only the squared shaped candidates are accepted, (4) Otsu's thresholding is performed for separating the white and black cells, (5) the image is divided into cells and the black and white pixels in each cell are tallied, and finally 6) the bits are analyzed and checked in the dictionary for identifying the marker.

The accumulation of relative pose error in a robot is a crucial factor that directly affects its successful navigation to desired locations. This discrepancy arises from the difference between the robot's actual pose and its ideal pose. It is essential to regularly estimate and correct this error to ensure accurate navigation. The relative pose between the camera and marker can be determined using six components, with three related to translation (e.g., x, y, and z) and three related to rotation (e.g., roll, pitch, and yaw angles). These components are represented in a homogeneous transformation matrix (H) consisting of a rotation matrix (R) and a translation matrix (T). However, depending on the specific application, not all six components may be required. In our case, for instance, only the x, y, and yaw parameters are extracted to estimate the pose error by calculating the variance between these parameters and the ground truth values.

The marker coordinate system is placed at the marker's center (by default), with the Z-axis pointing up. The X-axis of the marker is represented by the color red, the Y-axis by green, and the Z-axis by blue. The x-axis of the marker is set to align with the x-axis, X_R, of the robot frame $\{R\}$, and the y-axis of the marker is set to align with the y-axis, Y_R, of the robot frame $\{R\}$. The markers are detected relative to the camera with *aruco_detect* package by looking for markers in the images stream. The position and orientation of the detected markers are published in ROS topics and as TF frames.

Path Planning and Path Following

For the computation of a shortest and collision-free path from the start point to the goal point, numerous techniques are available [34, 35]. In this study we employ the A* algorithm [36], which is an extension of Dijkstra's algorithm [37]. The function $\mathcal{A}(n)$ is used to compute the shortest path between the starting and goal points as

$$\mathcal{A}(n) = \mathcal{G}(n) + \mathcal{H}(n) \tag{12}$$

where $\mathcal{G}(n)$ is the cost of moving from the starting cell to the current cell n, and $\mathcal{H}(n)$ is the heuristic value that estimates the cost of moving from the current cell n to the final goal cell. The algorithm computes the shortest path by choosing the smallest $\mathcal{A}(n)$-valued cells until it reaches the goal cell. The Euclidean distance heuristic is adopted for computing the value of $\mathcal{H}(n)$ as

$$\mathcal{H}(n) = \left\| \begin{bmatrix} x_{\text{start}} \\ y_{\text{start}} \end{bmatrix} - \begin{bmatrix} x_{\text{goal}} \\ y_{\text{goal}} \end{bmatrix} \right\| \tag{13}$$

The optimum path computed between the start and goal point in the generated map populated with obstacles is illustrated with the red line in Fig. 3c. The obstacle grid, represented by black, marks the contour area in the map where the robot cannot pass. Conversely, the free grid, represented by white, indicates the areas where the robot is able to navigate.

Once the path is planned, the next step is to follow the computed path. This study employs pure pursuit (PP) algorithm [38] for following the planned path. PP calculates angular velocity commands to guide the robot from its current position towards a look-ahead point in front of it. The robot's linear velocity is considered constant, hence, allowing for changes at any point during the path. PP continually updates the look-ahead point based on the robot's current position, moving it along the path until reaching the last point. This results in the concept of the robot consistently pursuing a point ahead of it. The parameter *LookAheadDistance* plays a crucial role in determining the placement of the look-ahead point. It specifies the distance along the path that the robot should consider when computing angular

velocity commands. Adjusting this parameter significantly impacts the robot's path following behavior. Two primary goals are associated with it: regaining the path and maintaining the path. A small *LookAheadDistance* allows the robot to quickly regain the path between waypoints, but it may cause overshooting and oscillations. Conversely, increasing the *LookAheadDistance* can help reduce oscillations, but it may also lead to higher curvatures around corners. Tuning the *LookAheadDistance* property is crucial and should be done based on the specific application requirements and the desired response of the robot. Figure 4 illustrates the concept of the look-ahead point and desired heading vector. A δ between 0° to 90° results in a turning to the right, whereas a δ between 90° to 180° results in turning to the left.

As shown in Fig. 4, the center of the robot chassis is used as the reference point on the robot. The target point is the look-ahead point for the robot that is represented by the green point on the circumference of the circle. The angle between the current heading vector and the desired heading vector is referred to as δ. The current heading vector aligns with the y-axis, Y_R of the robot frame. The goal is to make the vehicle steer at the angle δ and then proceed to that point.

The errors between the desired and actual heading and speed are eliminated by using the following discretized PID control equations

$$u_1(k) = K_p e_\phi(k) + K_i \sum_0^k e_\phi(k) T_s + K_d \frac{e_\phi(k) - e_\phi(k-1)}{T_s} \tag{14}$$

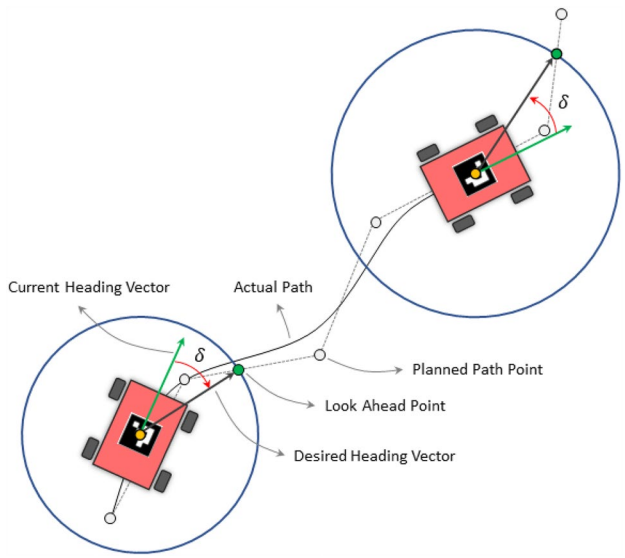

Fig. 4 Illustration of look ahead point in path following

$$u_2(k) = K_p e_V(k) + K_i \sum_0^k e_v(k) T_s + K_d \frac{e_v(k) - e_V(k-1)}{T_s} \tag{15}$$

where $e_\phi = \phi - \phi_{des}$ and $e_V = V - V_{des}$.

Particle Filtering and State Estimation

PF is used for the state estimation of dynamic nonlinear non-Gaussian systems. It is composed of two phases: prediction and update. The prediction step is based on a motion model, which defines how the vehicle moves. There are two basic branches of motion models to choose from. The first one is when the vehicle is assumed as a point without orientation. In this work, the vehicle is an extended target and its orientation is important. Hence, coordinated turn (CT) model is deployed [39]. In this model, the state vector of the vehicle is represented as

$$\mathbf{x} = \left[x, y, |v|, |a|, \phi, \omega\right]^T \tag{16}$$

where x and y are positions along x- and y-axes, respectively, $|v|$ and $|a|$ are the speed and amplitude of acceleration, and ϕ and ω are the heading and rate of turn.

The update step of PF is based on the measurement model. The measurements in this work are positions of the ArUco markers. Hence, the measurement vector is defined as

$$\mathbf{z} = \left[x, y\right]^T \tag{17}$$

Using the Chapman-Kolmogorov equation and Markov properties, the prediction step in optimal Bayes filter can be written as [40]

$$\begin{aligned} P(\mathbf{x}_k \mid \mathbf{z}_{1:k}) &= \int P(\mathbf{x}_k, \mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \\ &= \int P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{z}_{1:k-1}) P(\mathbf{x}_{k-1} \mid \mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \\ &= \int P(\mathbf{x}_k \mid \mathbf{x}_{k-1}) P(\mathbf{x}_{k-1} \mid \mathbf{z}_{k-1}) d\mathbf{x}_{k-1} \end{aligned} \tag{18}$$

The update step is based on the Bayesian rule and can be written as

$$\begin{aligned} P(\mathbf{x}_k \mid \mathbf{z}_{1:k}) &= P(\mathbf{x}_k \mid \mathbf{z}_k, \mathbf{z}_{1:k-1}) \\ &= \frac{P(\mathbf{z}_k \mid \mathbf{x}_k) P(\mathbf{x}_k \mid \mathbf{z}_{1:k-1})}{P(\mathbf{x}_k \mid \mathbf{z}_{1:k-1})} \\ &\propto P(\mathbf{z}_k \mid \mathbf{x}_k) P(\mathbf{x}_k \mid \mathbf{z}_{1:k-1}) \end{aligned} \tag{19}$$

where $\mathbf{z}_{1:k}$ denotes the set of measurements from 1st to kth step.

When the motion and measurement equations are linear, optimal filtering is possible using the Kalman filter [41].

However, the coordinated turn model is nonlinear, necessitating suboptimal approaches. PF provides a framework for nonlinear filtering problems, approximating the PDFs using weighted particles.

$$P(x) \approx \sum_{i=1}^{N} w^i \delta\left(x - x^i\right) \tag{20}$$

The weights, w, are calculated using the importance sampling [42]. In the prediction step, PF propagates the state vectors. The update state updates the weights of the particles based on the Bayes' rule.

$$w_k^i \propto w_{k-1}^i P\left(\mathbf{z}_k \mid \mathbf{x}_k^i\right) \tag{21}$$

The posterior distribution can be written as

$$P\left(\mathbf{x}_k \mid \mathbf{z}_k\right) \approx \sum_{i=1}^{N} w_k^i \delta\left(\mathbf{x}_k - \mathbf{x}_k^i\right) \tag{22}$$

As $N \to \infty$ this approximation converges to $P\left(\mathbf{x}_k \mid \mathbf{z}_k\right)$.

The pixel measurements flicker between pixels. Due to high frame per second (fps), the flicker causes high measurement errors. High measurement error does not cause serious problems in localization but the predicted heading of the vehicle is poor. A solution to this problem is smoothing [43]. The difference between filtering and smoothing is smoothing estimates $P\left(x_{k-n} \mid z_k\right)$ where $n > 0$. In other words, smoothing estimates past states using future measurements. This helps to smoothen the estimates. There are several strategies for smoothing. We have preferred forward filtering-backward smoothing recursions [43, 44] because they reduce the errors and uncertainties associated with state estimation by refining the states based on a combination of the likelihoods of the future measurements with the present predicted states, as below.

$$P\left(\mathbf{x}_{k-n} \mid \mathbf{z}_{k-n:k}\right) \propto P\left(\mathbf{x}_{k-n} \mid \mathbf{z}_{k-n-1}\right) P\left(\mathbf{z}_{k-n:k} \mid \mathbf{x}_{k-n}\right) \tag{23}$$

This formulation requires only the last n measurements to be kept in the memory. $n = 0, 5, 10, 15$ are tested for smoothing filter. The motion model for CT is defined as

$$\mathbf{x}_{k|k-1} = \begin{bmatrix} x + \frac{2|v|\Delta t + |a|\Delta t^2}{\omega \Delta t} \sin\left(\phi + \omega\frac{\Delta t}{2}\right) \sin\left(\omega\frac{\Delta t}{2}\right) \\ y + \frac{2|v|\Delta t + |a|\Delta t^2}{\omega \Delta t} \cos\left(\phi + \omega\frac{\Delta t}{2}\right) \sin\left(\omega\frac{\Delta t}{2}\right) \\ |v| + |a|\Delta t \\ |a| \\ \phi + \omega \Delta t \\ \omega \end{bmatrix} \tag{24}$$

The measurement model is linear.

$$\mathbf{z}_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{H}} \mathbf{x}_k \tag{25}$$

where $\mathbf{H}$ is the measurement matrix. The parameters of the PF are

$$\mathbf{Q} = \text{diag}[10^{-1}, 10^{-1}, 10^{-1}, 10^{-2}, 15 \times \pi/180, 2 \times \pi/180] \tag{26}$$

where *diag* indicates diagonal matrix. The measurement error matrix is $\mathbf{R}$

$$\mathbf{R} = \text{diag}[10^{-1}, 10^{-1}] \tag{27}$$

The values for the Q matrix were selected based on our expectations of process noise stemming from the target's position, speed, acceleration, heading angle, and turning rate, respectively. For instance, in terms of position, a deviation of 0.1 ms was anticipated during tracking due to model-mismatch or skidding error during take-off. As for the selection of values for the R matrix, target states over a period of time were gathered and the extent of deviation present in the measurements was observed.

To evaluate the smoothing performance, different scenarios with only one turn were setup. In each scenario, the vehicle turns to a direction. The vehicle starts the scenario moving along y-axis. Then, it turns to different directions as seen in Fig. 5. The average error at each scenario is as shown in the Table 1. The results indicate that the errors decrease with increasing smoothing amount.

To visually present the results, the predicted heading and speed values along the three scenarios, in which the vehicle moves in different directions, are presented in

Table 1 Average errors of smoothing filters with different n values for each scenario

		Scenario 1		Scenario 2		Scenario 3	
		Heading (deg)	Speed (m/s)	Heading (deg)	Speed (m/s)	Heading (deg)	Speed (m/s)
Smoothing	$n = 0$	46.34	0.96	21.84	0.70	30.93	0.72
	$n = 5$	18.58	0.24	9.33	0.13	10.96	0.13
	$n = 10$	6.50	0.07	3.68	0.02	4.17	0.02
	$n = 15$	4.98	0.02	4.58	0.02	4.61	0.02

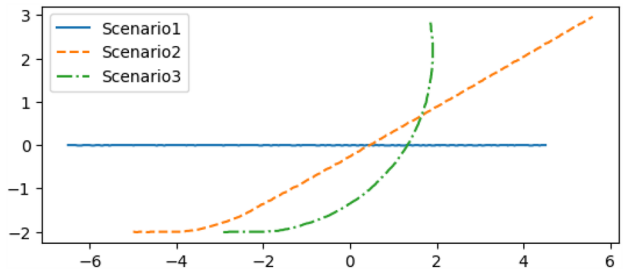

Fig. 5 Traveled paths with different amount of rotations

Figs. 6 and 7. It is seen that the filtering results are poor. However, the results indicate that $n = 15$ provides the best results. The errors for the estimations are given in Figs. 8 and 9, where GT represents the ground truth. The x-axis represents time in seconds, while the y-axis represents degrees in Fig. 6 and 8, and meters per second in Figs. 7 and 9. It is evident from these estimation errors that as the number of smoothing levels increases, estimation errors decrease significantly. The smoothing filter proves invaluable in compensating for unforeseen sensor information errors that may arise from factors like light conditioning in the environment, system network data transfer lag, and an unstable number of frames per second (fps) for each iteration of the sensor, ensuring the robustness and trustworthiness of autonomous system. It is important to note that, as mentioned earlier, the smoothing filter predicts the target states from previous times. Consequently, the smoothing level must fall within a certain range and should not exceed a level higher than the motion velocity of the target relative to the camera's frames per second in time elapsed manner. In certain scenarios, estimates from the smoothing filter may fall outside the performance range of the filter. Reliance on outdated states, instead of current information, for safety-driving maintenance can lead to significant errors such as collisions with obstacles and inaccurate localization for path following, among other issues.

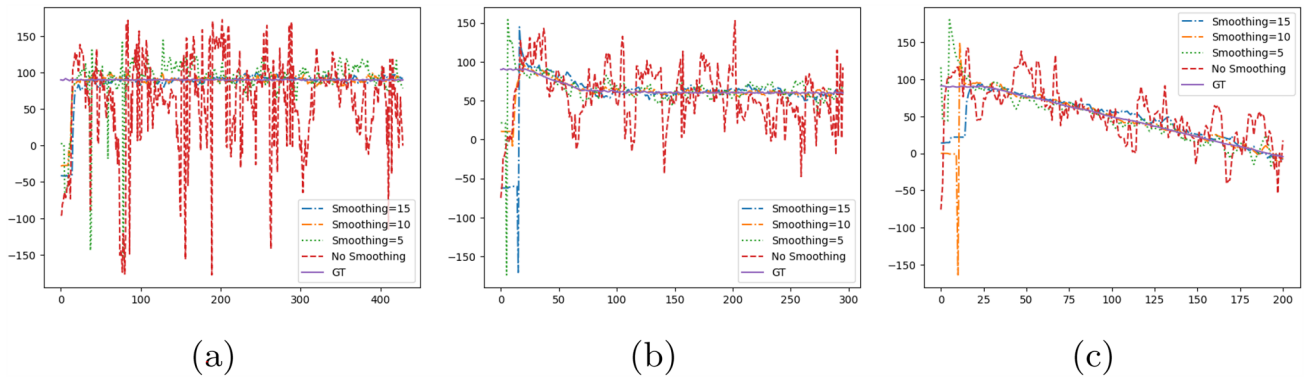

Fig. 6 Heading estimations: **a** scenario 1, **b** scenario 2, and **c** scenario 3

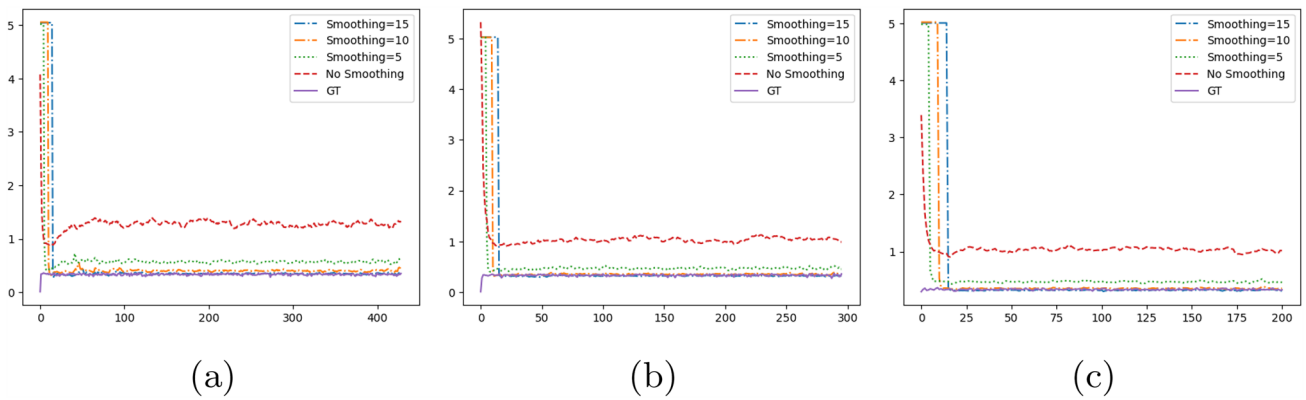

Fig. 7 Speed estimations: **a** scenario 1, **b** scenario 2, and **c** scenario 3

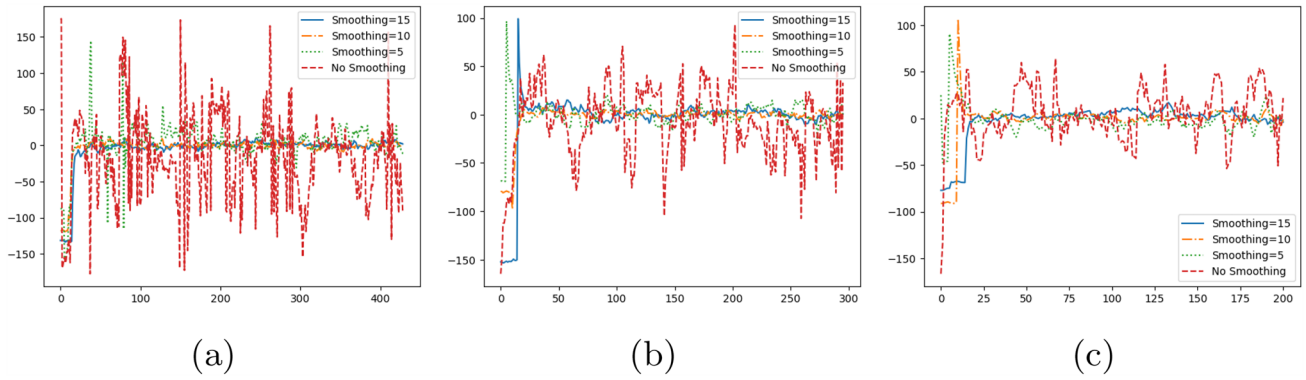

Fig. 8 Heading estimation errors: **a** scenario 1, **b** scenario 2, and **c** scenario 3

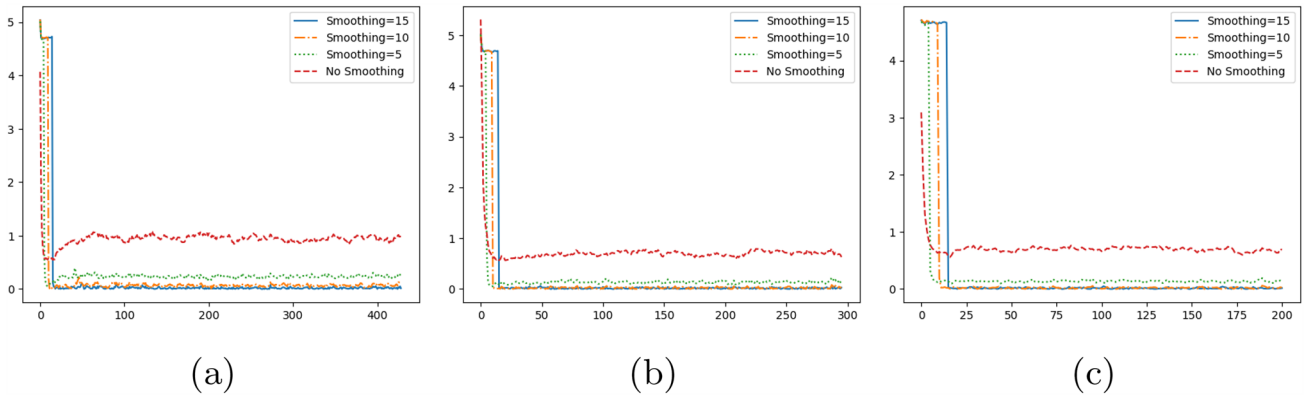

Fig. 9 Speed estimation errors: **a** scenario 1, **b** scenario 2, and **c** scenario 3

Implementation of the Framework

The complete system in its entirety is schematically illustrated in the block diagram shown in Fig. 10. Detailed working of the implemented system is provided with reference to the block diagram thereof.

A retrofitted 4-wheeled Ackermann drive vehicle, as shown in Fig. 13, is used as the mobile robot platform in this study. Hardware of the robot comprises of a Raspberry Pi 4, two 5V DC motors for driving and steering the robot, an L298D H-Bridge driver module connected with the GPIO pins of the Rasperry Pi 4 to control the DC motors, three 3.7v Lithium-ion batteries, and a 1000 mAh power bank. The Raspberry Pi 4 is connected to the laptop PC via a wireless connection on a same network. In order to avoid overhead and latency that occurs in TCP/IP communication protocol, the fast and simple UDP communication protocol is used for data transfer between the Raspberry Pi 4 and the Laptop PC. The Raspberry Pi 4 (robot) serves as the server and the Laptop PC as the client in the communication protocol. The laptop PC is equipped with an Intel(R) Core (TM) i5-10300 H @ 2.50GHz (8 CPUs), 8GB RAM, a 4GB NVIDIA GeForce GTX 1650 graphics card, and Ubuntu 20.04.5 LTS (Focal Fossa) as the operating system. All the algorithms running on the laptop PC are implemented with ROS Noetic as the middleware suite, and Python as the development language. Since the official Linux distribution, i.e., Raspian can not support ROS Noetic, therefore the Raspberry Pi 4 mounted on the robot uses Ubuntu Mate as the operating system.

A camera mounted on the ceiling at a height of 190 cm from the ground, such that it points downward and provides a visual coverage of the environment, servers as a global vision sensor for the system. The start and goal points, and the obstacles are placed inside the field of view of the camera. For detecting the start and goal points, visual markers are placed on top of the robot and the goal point. The marker on the robot serves as the starting/current point of the robot, and the marker at the goal point serves as the target point. Due to motion of the robot, the position and orientation of the robot changes and as a consequence so does the position and the orientation of the marker affixed on it. The camera captures a stream of images of the mobile robot and the

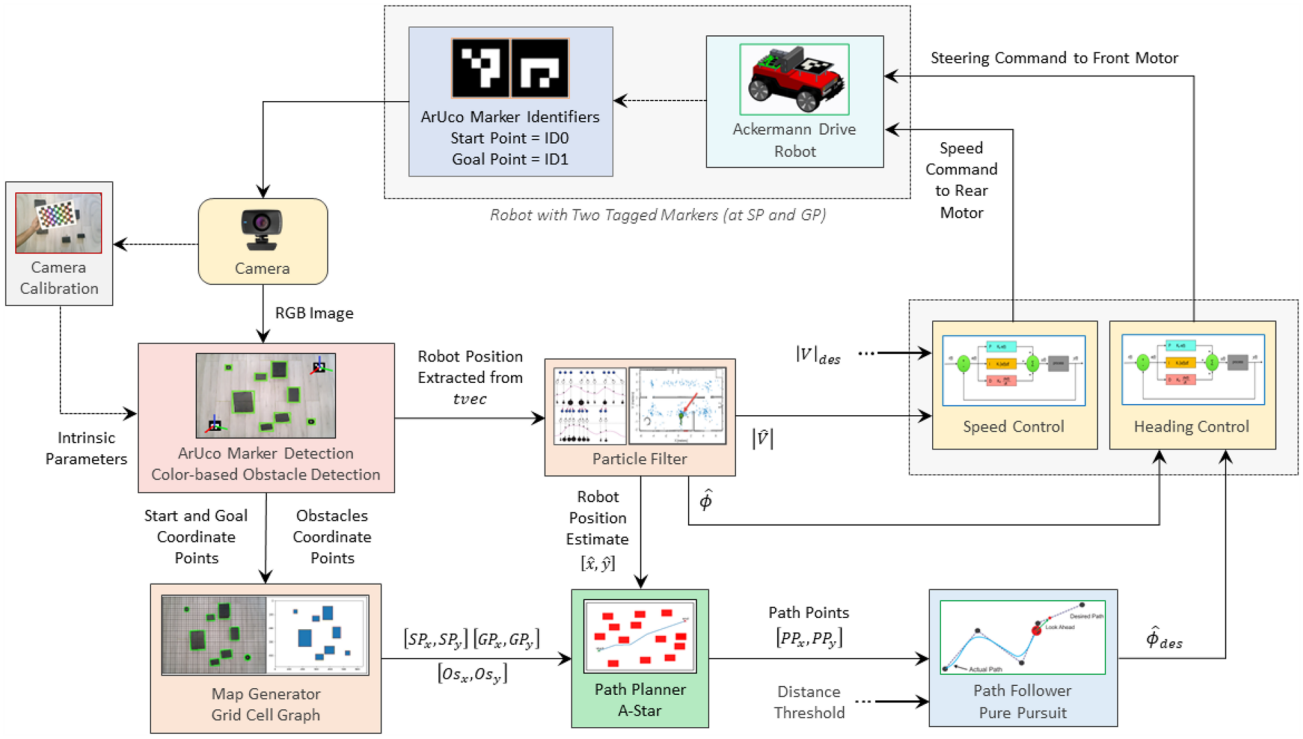

Fig. 10 Block diagram of the complete system

markers on the ground. The camera used has a resolution of 640x480 pixels and a frame rate of 30 FPS. The size of the data collected for one second is calculated. A resolution of 640 x 480 results in 307,200 different pixel values. With 30 FPS this becomes 30 x 307,200 = 921,600 pixel values. Since each pixel normally is represented by a value between [0, 255] it requires 921,600 x 8 = 73.7 MB to store one second worth of retrieved data. The camera is connected to the laptop PC via a USB connection.

Experimental Validation

The effectiveness of the proposed approach was tested through both simulation and real-world implementation scenarios. The tests were carried out in three different scenarios, with different number of obstacles placed at different locations. The entire system was simulated in Gazebo simulation environment. Three different environment scenarios were generated for conducting the simulation tests in Gazebo, as shown in Fig. 11. Figure 12 shows the PF plots acquired for the simulation runs in these three scenarios.

Moreover, to illustrate the effectiveness of the proposed method, implementation tests were carried out using a real 4-wheel Ackermann drive mobile robot, shown in Fig. 13. The implementation tests were conducted in three different environmental settings, as shown in Fig. 14. The acquired average position and heading errors for simulation and real implementation are illustrated in Table 2. For each of the three scenarios, the performance metrics were compared to a baseline method that did not utilize PF. The acquired results demonstrated that our proposed approach has superior performance over the baseline method. In Scenario 1, characterized by the presence of five obstacles along an almost straight trajectory, the proposed approach demonstrated good performance in estimating the robot's heading

Table 2 Average position and heading errors

Scenario	No. of obstacles	Simulation		Implementation	
		Position (cm)	Heading (deg)	Position (cm)	Heading (deg)
1	5	4.7	3.2	6.1	4.3
1	7	6.3	3.8	7.2	4.6
3	10	6.7	3.9	7.8	5.1

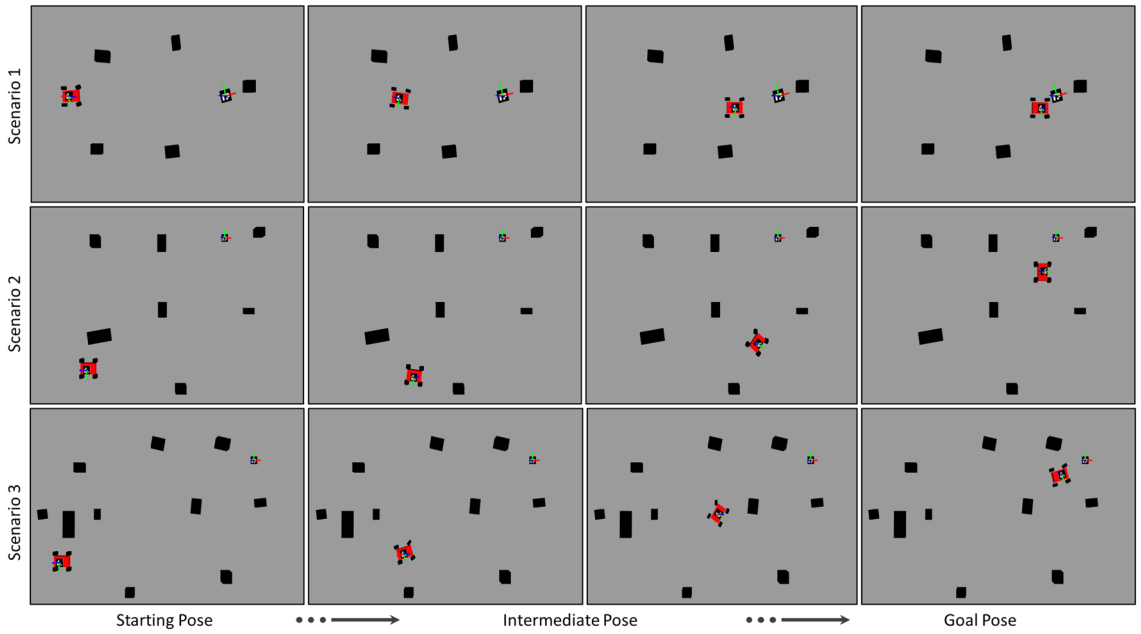

Fig. 11 Gazebo simulation in different scenarios

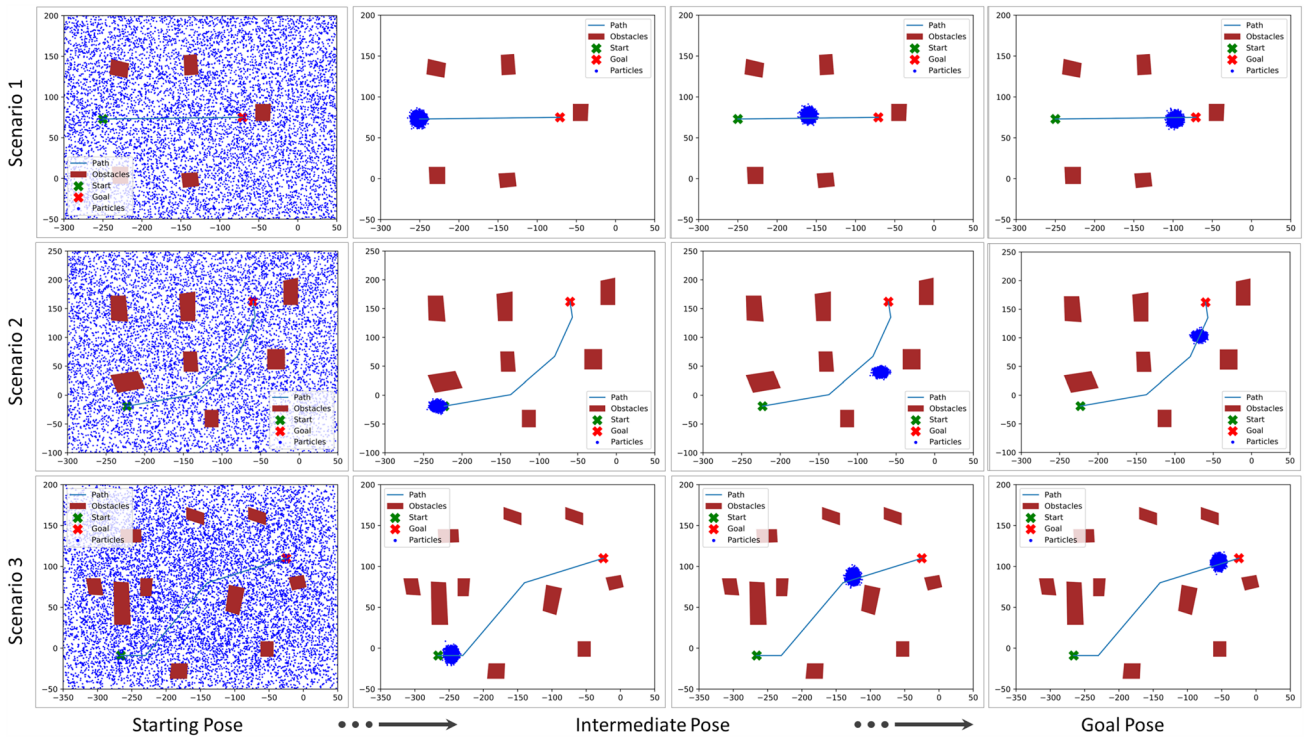

Fig. 12 PF simulation results in different scenarios

and position. In Scenario 2, marked by the presence of seven obstacles and a complex curved/bent path, the system exhibited good adaptability and precision by adhering to the curved path. In scenario 3, comprising 10 obstacles

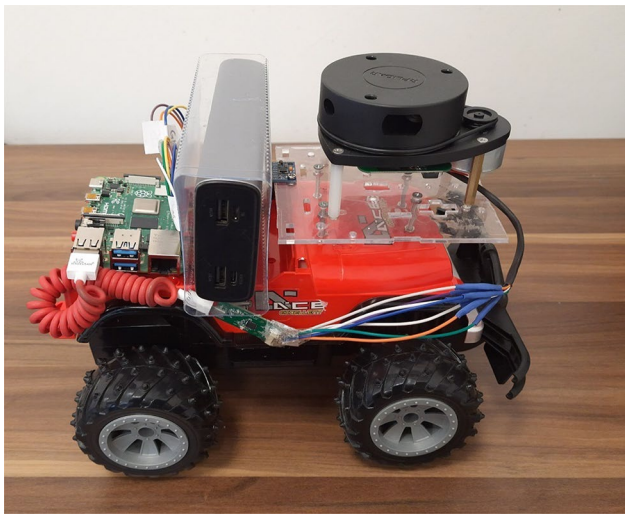

Fig. 13 Ackerman drive mobile robot used for experimentation

and a further complex path geometry with two significant bends, the average position and heading errors, increased slightly but remained in acceptable ranges. The simulation and implementation results highlight the system's ability to consistently provide accurate robot pose estimates, underscoring its reliability in navigation tasks.

Conclusion

In this work, a simple approach for pose estimation of an autonomous mobile robot is presented. The proposed approach uses ArUco markers for acquiring position information and PF for computing the desired heading vector of the robot. Due to high noise in the orientation information from the ArUco markers, orientation information is not acquired from the marker, instead PF is used to generate the heading information (for generating steering commands). The proposed system enables the robot to navigis first simulated in a Gazebo environment to assess its performance. Subsequently, the approach is implemented on a physical 4-wheeled Ackermann drive mobile robot, allowing for real-world validation of its effectiveness. The acquired findings show the robustness and effectiveness of the proposed low-cost approach for robot localization and navigation. In future work, the proposed fiducial marker-based pose estimation technique can be integrated with auxiliary sensor modalities using non-linear state estimators for more accuracy and robustness.

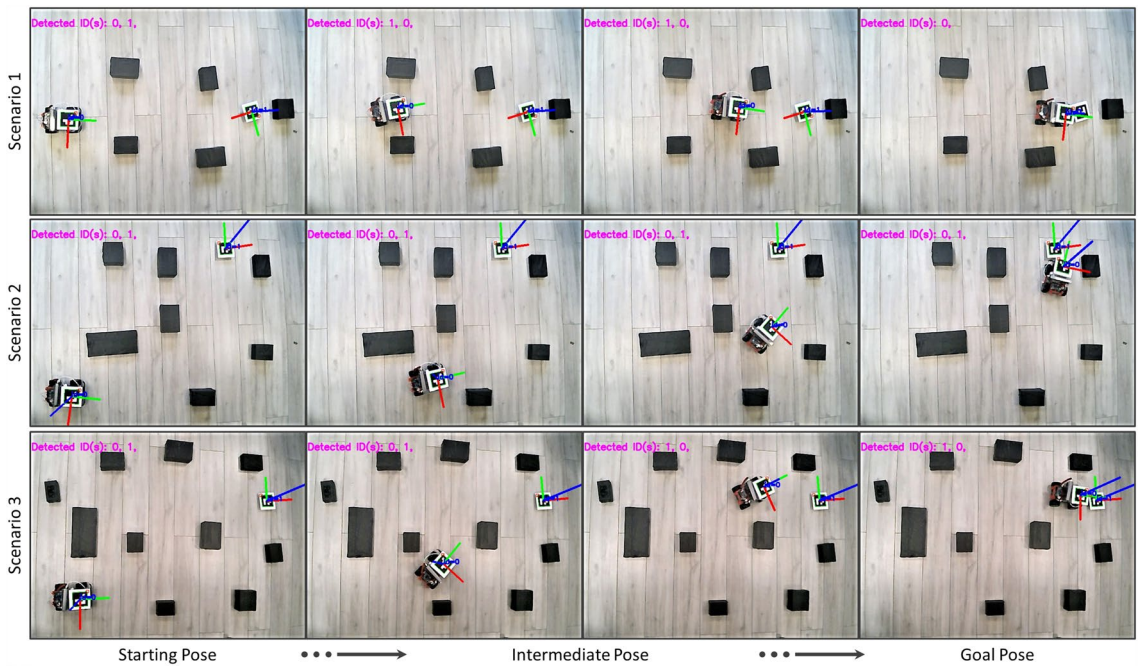

Fig. 14 Implementation results in different scenarios

Author Contributions All the others contributed equally to this work.

Funding This study did not receive any funding.

Data Availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Research involving human and/or animals Not applicable.

Informed consent Not applicable.

References

1. Andreu-Perez J, Deligianni F, Ravi D, Yang G-Z. Artificial intelligence and robotics. 2018. arXiv preprint arXiv:1803.10813.
2. Rubio F, Valero F, Llopis-Albert C. A review of mobile robots: concepts, methods, theoretical framework, and applications. Int J Adv Robot Syst. 2019;16:1729881419839596.
3. Liu L, et al. Computing systems for autonomous driving: state of the art and challenges. IEEE Internet Things J. 2020;8:6469–86.
4. Panchpor AA, Shue S, Conrad JM. A survey of methods for mobile robot localization and mapping in dynamic indoor environments. IEEE; 2018. pp. 138–144.
5. Tzafestas SG. Mobile robot control and navigation: a global overview. J Intell Robot Syst. 2018;91:35–58.
6. Roy P, Chowdhury C. A survey of machine learning techniques for indoor localization and navigation systems. J Intell Robot Syst. 2021;101:63.
7. Panigrahi PK, Bisoy SK. Localization strategies for autonomous mobile robots: a review. J King Saud Univ Comput Inf Sci. 2022;34:6019–39.
8. Gul F, Rahiman W, Nazli Alhady SS. A comprehensive study for robot navigation techniques. Cogent Eng. 2019;6:1632046.
9. Möller R, Furnari A, Battiato S, Härmä A, Farinella GM. A survey on human-aware robot navigation. Robot Autonom Syst. 2021;145: 103837.
10. Qin T, Pan J, Cao S, Shen S. A general optimization-based framework for local odometry estimation with multiple sensors. 2019. arXiv preprint arXiv:1901.03638.
11. Chuwei M, Ju H, Zhanyu Z. Localization and navigation method for omni-directional mobile robot based on odometry. IEEE; 2019. pp. 4697–4702.
12. Jaimez M, Monroy J, Lopez-Antequera M, Gonzalez-Jimenez J. Robust planar odometry based on symmetric range flow and multiscan alignment. IEEE Trans Robot. 2018;34:1623–35.
13. Mohamed SA, et al. A survey on odometry for autonomous navigation systems. IEEE Access. 2019;7:97466–86.
14. Jonnavithula N, Lyu Y, Zhang Z. Lidar odometry methodologies for autonomous driving: a survey. 2021. arXiv preprint arXiv:2109.06120.
15. Yang M, et al. Sensors and sensor fusion methodologies for indoor odometry: a review. Polymers. 2022;14:2019.
16. An J, Mou H, Lu R, Li Y. Localization and navigation analysis of mobile robot based on slam, vol. 1827. IOP Publishing; 2021. pp. 012089.
17. Kim P, Chen J, Kim J, Cho YK. Slam-driven intelligent autonomous mobile robot navigation for construction applications. Springer; 2018. pp. 254–269.
18. Mac TT, et al. Hybrid slam-based exploration of a mobile robot for 3D scenario reconstruction and autonomous navigation. Acta Polytech Hung. 2021;18:197–212.
19. Singandhupe A, La HM. A review of slam techniques and security in autonomous driving. IEEE; 2019. pp. 602–607.
20. Ahmed MF, Masood K, Fremont V. Active slam: a review on last decade. 2022. arXiv preprint arXiv:2212.11654.
21. Qu X, Soheilian B, Paparoditis N. Landmark based localization in urban environment. ISPRS J Photogramm Remote Sens. 2018;140:90–103.
22. Prasad A, Sharma B, Kumar SA. Strategic creation and placement of landmarks for robot navigation in a partially-known environment. IEEE; 2020. pp. 1–6.
23. Ramaithititima R, Bhattacharya S. Landmark-based exploration with swarm of resource constrained robots. IEEE; 2018. pp. 5034–5041.
24. Kato H, Billinghurst M. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. IEEE; 1999. pp. 85–94.
25. Fiala M. ARTag, a fiducial marker system using digital techniques, vol. 2. IEEE; 2005. pp. 590–596.
26. Olson E. AprilTag: a robust and flexible visual fiducial system. IEEE; 2011. pp. 3400–3407.
27. Garrido-Jurado S, Muñoz-Salinas R, Madrid-Cuevas FJ, Marín-Jiménez MJ. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognit. 2014;47:2280–92.
28. López de Ipina A, Mendonça PR, Hopper A, Hopper A. TRIP: a low-cost vision-based location system for ubiquitous computing. Pers Ubiquitous Comput. 2002;6:206–19.
29. Bergamasco F, Albarelli A, Rodola E, Torsello A. RUNE-Tag: a high accuracy fiducial marker with strong occlusion resilience. IEEE; 2011. pp. 113–120.
30. DeGol J, Bretl T, Hoiem D. ChromaTag: a colored marker and fast detection algorithm. 2017. pp. 1472–1481.
31. Calvet L, Gurdjos P, Griwodz C, Gasparini S. Detection and accurate localization of circular fiducials under highly challenging conditions. 2016. pp. 562–570.
32. Atcheson B, Heide F, Heidrich W. CALTag: high precision fiducial markers for camera calibration. 2010;10:41–48.
33. Zhang Z. A flexible new technique for camera calibration. IEEE Trans Pattern Anal Mach Intell. 2000;22:1330–4.
34. Alam MS, Rafique MU. Mobile robot path planning in environments cluttered with non-convex obstacles using particle swarm optimization. IEEE; 2015. pp. 32–36.
35. Alam MS, Rafique MU, Kauser Z, Saleem M. Swarm intelligence based multi-objective path planning in environments cluttered with danger sources. IEEE; 2016. pp. 1–6.
36. Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern. 1968;4:100–7.
37. Dijkstra EW. A note on two problems in connexion with graphs. 2022. pp. 287–290.
38. Coulter RC. Implementation of the pure pursuit path tracking algorithm. Tech. Rep. Carnegie-Mellon UNIV Pittsburgh PA Robotics INST. 1992.
39. Roth M, Hendeby G, Gustafsson F. EKF/UKF maneuvering target tracking using coordinated turn models with polar/Cartesian velocity. 2014. pp. 1–8.
40. Arulampalam M, Maskell S, Gordon N, Clapp T. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. IEEE Trans Signal Process. 2002;50:174–88.
41. Ristic B, Arulampalam S, Gordon N. Beyond the Kalman filter: particle filters for tracking applications. Norwood: Artech House; 2004.

42. Doucet A, Godsill S, Andrieu C. On sequential Monte Carlo sampling methods for Bayesian filtering. Stat Comput. 2000;10:197–208.
43. Doucet A, Johansen AM. A tutorial on particle filtering and smoothing: Fifteen years later. Tech. Rep. Department of Statistics, University of British Columbia. 2008.
44. Briers M, Doucet A, Maskell S. Smoothing algorithms for state–space models. Ann Inst Stat Math. 2009;62:61.