IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 June 2024, accepted 11 July 2024, date of publication 17 July 2024, date of current version 26 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3429506

RESEARCH ARTICLE

ASAP: Agile and Safe Pursuit for Local Planning of Autonomous Mobile Robots

DONG-HYUN LEE[1,2], SUNGLOK CHOI[3], (Member, IEEE), AND KI-IN NA[4]

[1]School of Electronic Engineering, Kumoh National Institute of Technology, Gumi-si, Gyeongbuk 39177, Republic of Korea
[2]Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi-si, Gyeongbuk 39177, Republic of Korea
[3]Department of Computer Science and Engineering, Seoul National University of Science and Technology (SEOULTECH), Nowon-gu, Seoul 01811, Republic of Korea
[4]Mobility Robot Research Division, Electronics and Telecommunications Research Institute (ETRI), Yuseong-gu, Daejeon 34129, Republic of Korea

Corresponding author: Ki-In Na (kina4147@etri.re.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by Korean Government through MSIT under Grant RS-2024-00337584.

ABSTRACT This paper presents a novel local planning approach called Agile and SAfe Pursuit (ASAP) for autonomous mobile robots. It aims to enable agile path following and safe collision avoidance in cluttered environments, while ensuring computational efficiency for real-time performance in embedded systems with limited computational power. For agile path following, the proposed approach utilizes a local path that includes a line path, arc path, and in-place rotation, and generates a target velocity based on the kinematic constraints of the robot. For safe collision avoidance, the proposed approach uses obstacle information to generate safety corners, which represent points in free space to circumvent obstacles with arbitrary shapes, and selects the best safety corner with the minimum travel time. To reach the target velocity as quickly as possible, the proposed approach uses a normalized velocity space to calculate control velocity that achieves the ratio of the linear and angular components of the target velocity in the shortest possible time period. For end-users to easily adapt a robot's behavior to different environments, the proposed approach is designed to require only a few tuning parameters. The proposed algorithm's agile control, rigorous collision avoidance, and computational efficiency were demonstrated through experimental results from hardware-in-the-loop simulations under various scenarios and real-robot tests in cluttered environments. Remarkably, the proposed approach achieves computational speeds that are 25 to 200 times faster than other existing algorithms.

INDEX TERMS Autonomous mobile robot, collision avoidance, local planner, path following.

I. INTRODUCTION

Autonomous mobile robots have gained widespread popularity in various automation applications, such as smart factories and last-mile delivery, owing to advancements in deep learning-based perception, the computational power of embedded systems, and sensor technologies [1], [2], [3]. This growing interest in autonomous mobile robots has driven extensive research aimed at enhancing their functionality and performance across diverse environments and tasks. Numerous studies have explored different types of mobile robots, each offering unique capabilities. For instance, Arif et al. [4] designed an amphibious spherical robot that achieves high torque, versatile motion, compactness, and stability without relying on optimal control. Kim et al. [5] introduced a mobile platform with transformable wheels capable of navigating steps and stairs in indoor environments. Additionally, Jiang et al. [6] proposed a novel sliding mode control method for four-wheel omnidirectional mobile robots, effectively compensating for lumped disturbances in harsh terrain conditions.

Among the various mechanisms used in autonomous mobile robots, the differential-drive mechanism stands out due to its simplicity in kinematics and hardware, as well as its ability to rotate in narrow spaces. In mobile robot applications, autonomous navigation is a critical technology that combines a global planner for path planning and a local planner for path following and collision avoidance [7], [8],

The associate editor coordinating the review of this manuscript and approving it for publication was Jamshed Iqbal.

[9]. Typically, global planners determine a feasible path to the target position using a given map without considering the kinematic constraints of the robots or unknown obstacles not present on the map. In contrast, local planners generate control velocities to follow the global path while avoiding collisions by considering the kinematic constraints of the robot and newly detected obstacles. Mobile robot systems with limited computational power require a computationally efficient local planner that accurately follows a given global path while safely avoiding collisions.

There are three major approaches in the local planning of mobile robots: velocity space optimization, trajectory space optimization, and learning-based approaches. Velocity space optimization approaches, such as the curvature velocity method (CVM) and the dynamic window approach (DWA), have been widely explored for local planning. These methods generate collision-free and feasible velocities without requiring trajectory optimization but are often limited by their dependence on multiple parameter tuning and sampling-related parameters [10], [11], [12], [13]. Trajectory space optimization methods, including Timed-Elastic-Band (TEB) and Model Predictive Control (MPC), aim to find near-optimal trajectories but are computationally expensive and require accurate dynamic models, making them less practical for real-time applications [14], [15]. Learning-based approaches leverage machine learning algorithms to learn motion commands directly from sensory input, offering adaptability and robustness in complex environments. However, they face challenges related to sensor data reliability, generalization, and computational complexity [16], [17], [18].

In this paper, we present a novel local planning approach called Agile and SAfe Pursuit (ASAP) for autonomous mobile robots. The motivation behind this work stems from the need for more robust and versatile navigation algorithms in autonomous robotics. The primary objective of this work is to develop a navigation algorithm for autonomous robots that can efficiently and safely navigate complex environments. The proposed algorithm aims to address the limitations of existing methods by optimizing local planning and obstacle avoidance in real-time. Traditional methods often struggle with scalability, adaptability, and efficiency in complex environments. Our contributions include the development of the ASAP algorithm, which improves upon these aspects, and extensive validation through simulations and real-world experiments.

The proposed approach uses velocity space optimization to determine control velocity and employs a local path of lines and arcs for accurate path following. It solves the local planning problem without using objective functions or sampling, avoiding associated parameters. This method selects control velocity from a dynamic window considering kinodynamic constraints, avoiding the need for a predictive dynamic model. It uses safety corners to navigate obstacles, bypassing the need for proximity-based constraints. The approach runs in real-time on embedded systems using lightweight processes, avoiding complex models like deep neural networks. It is robust and adaptable for various mobile robots with minimal parameter adjustments. The proposed approach has three primary objectives:

- Enable agile global path following and safe collision avoidance in cluttered environments.
- Ensure computational efficiency for real-time performance on embedded systems with limited resources.
- Allow end-users to operate it in a specific mission environment using only a minimal set of intuitively tunable parameters.

The remainder of this paper is organized as follows. Section II provides an overview of previous works with a discussion of their ideas and limitations. Section III describes the proposed local planner in detail. Section IV presents the experimental results obtained from hardware-in-the-loop simulations and real robot systems. Section V discusses the experimental results of the proposed approach, and finally, Section VI presents the conclusions of this study.

II. RELATED WORKS

In this section, we review the existing approaches for local planning in robot navigation, categorized into three main types: velocity space optimization approaches, trajectory space optimization approaches, and learning-based approaches. Each subsection will provide an overview of these approaches, their strengths, and their limitations, highlighting how our proposed method differs and improves upon them.

A. VELOCITY SPACE OPTIMIZATION APPROACHES

Velocity space optimization approaches formulate local planning as a constrained optimization problem in the velocity space of the robot. The CVM selects the best point that maximizes the objective function in the velocity space [10], [12]. The objective function typically refers to optimizing metrics such as path length, travel time, energy consumption, distance from obstacles or a combination of these factors. The lane curvature method (LCM) and beam curvature method (BCM) combines the lane and beam methods to CVM [19], [20]. Unlike the CVM-related approaches that require obstacle maps, the DWA selects the control velocity with the lowest cost by calculating the cost function for each sampled control velocity from the dynamic window [11], [13], [21].

Velocity space optimization approaches can generate collision-free and feasible velocities for robots, without requiring trajectory optimization. However, they only consider local paths as arcs and select the one that maximizes the objective function, requiring the tuning of multiple parameters to combine optimization elements with different units and ranges. Additionally, the effectiveness and computational cost of sampling-based approaches, such as DWA, depend significantly on sampling-related parameters, such as the number of samples and simulation time.

The proposed approach adopts a velocity space optimization approach to determine the control velocity from the dynamic window. However, this approach differs from the aforementioned approaches because it employs a local path comprising a line and arc, including in-place rotation, to enable agile and accurate path following. Considering the kinematic constraints, current velocity, and target angle error, the proposed approach deterministically solves the local planning problem without utilizing an objective function or sampling, which results in not having any parameters associated with sampling or trajectory scoring.

B. TRAJECTORY SPACE OPTIMIZATION APPROACHES

Trajectory space optimization methods search for an optimal trajectory in a space of all possible trajectories that satisfy the specific limitations and conditions, such as kinodynamic constraints on robot velocity and acceleration, as well as environmental constraints like avoiding obstacles. TEB, which is an extension of the elastic-band (EB) algorithm [22], incorporates temporary information on kinodynamic constraints, such as limited robot velocity and acceleration using a weighted multi-objective optimization framework [14], [23]. MPC is another trajectory space optimization approach that repeatedly solves an optimization problem to compute a sequence of control actions over a finite horizon [15], [24], [25]. MPC is a flexible framework that allows the combination of various algorithms to improve its performance in robot navigation, such as an ancillary state feedback controller, hybrid PID controller, and primal-dual neural network [26], [27], [28].

Although trajectory space optimization approaches can generate near-optimal trajectories that satisfy constraints or objectives, they are computationally expensive, particularly for large optimization problems or long horizons. Additionally, accurate models of robot dynamics and the environment are required for good performance, which can be challenging to obtain in practice. Furthermore, these approaches are sensitive to parameter selection and require exhaustive tuning to achieve a satisfactory performance.

The proposed approach differs from the trajectory space optimization approaches in that it selects the control velocity from a dynamic window in the velocity space that considers the kinodynamic constraints rather than generating the control velocity as a solution for constrained optimization. This feature makes it unnecessary for the proposed approach to have an accurate predictive dynamic model for the robot and avoids parameters related to optimization. Moreover, instead of using the proximity to obstacles as a constraint and finding an optimal solution that avoids obstacles, the proposed approach employs safety corners, which are points in free space extracted from obstacles, to circumvent obstacles and rapidly reach the global path.

C. LEARNING-BASED APPROACHES

Learning-based approaches use machine learning algorithms, such as deep reinforcement learning and neural network, to learn the mapping between the sensory input of a robot and its motion commands without relying on a predefined model of the environment or robot dynamics [16], [29]. End-to-end learning-based local planners learn mapping directly from sensory input to motion commands [17], [30]. Hybrid learning-based local planners combine learning-based approaches with traditional methods [31], [32], [33]. These approaches use machine learning models to learn a correction term or modify the output of the traditional planner based on current sensory input. To enhance the performance of learning based navigation, an input representation of LiDAR readings is proposed in [34]. Another hybrid approach involves using a trained agent to switch between multiple different planners based on sensor data observations [18], [35].

Learning-based local planners have shown promise in mobile robot navigation and autonomous driving in complex environments. However, these remain an active area of research, and there are many open challenges, such as dealing with noisy or incomplete sensor data, handling uncertainty, data requirements, generalization, adaptability, safety, interpretability, and complexity.

The proposed approach can be executed in real-time on embedded systems because it avoids utilizing complex models, such as deep neural networks, and instead employs three lightweight and interpretable processes. It is sufficiently robust for handling environmental changes and unexpected events, and can be adapted for mobile robots by adjusting a small number of parameters.

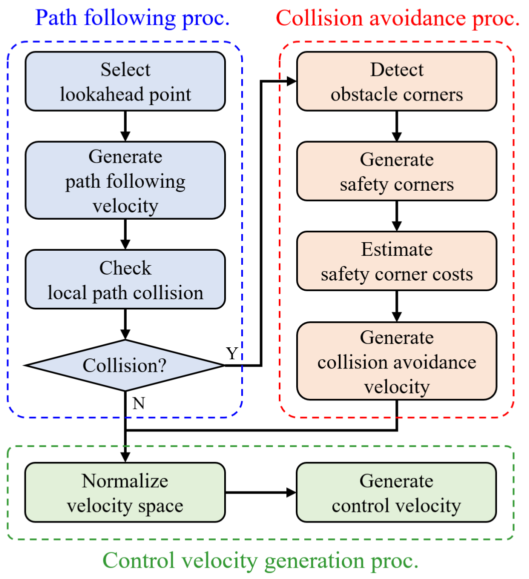

Path following proc.
Collision avoidance proc.
Select lookahead point
Generate path following velocity
Check local path collision
Collision?
Y
N
Detect obstacle corners
Generate safety corners
Estimate safety corner costs
Generate collision avoidance velocity
Normalize velocity space
Generate control velocity
Control velocity generation proc.

FIGURE 1. Flow chart of the proposed local planner.

III. AGILE AND SAFE PURSUIT (ASAP)

The proposed approach consists of three primary processes: path following, collision avoidance, and control velocity generation processes, as shown in Fig. 1. In the path following process, an adaptive lookahead point along the global path is selected while considering the current velocity, minimum distance from obstacles, and the stopping distance. Subsequently, the path following velocity is determined from the velocity space, which reflects the kinematic constraints of the robot. Using the lookahead point and path following velocity, a local path comprising line and an arc paths is generated. If no collision is predicted on the local path, the path following velocity is used as the target velocity. Otherwise, the collision avoidance process is triggered to generate the collision avoidance velocity, which is used as the target velocity.

The collision avoidance process is initiated by generating safety corners that act as temporary target points used to avoid arbitrarily shaped obstacles while considering their safety distance from them. Subsequently, the collision avoidance point is computed by selecting the best safety corner that can lead the robot to reach the global path in the shortest time without collisions. A collision avoidance velocity is generated to enable the robot to rapidly reach the collision avoidance point.

The control velocity generation process determines the target velocity path required to reach the turning radius of the target velocity within the shortest time in the normalized velocity space. It then computes the control velocity to follow the target velocity path and reach the target velocity in the shortest possible time considering the velocity window of the robot.

A. PATH FOLLOWING PROCESS

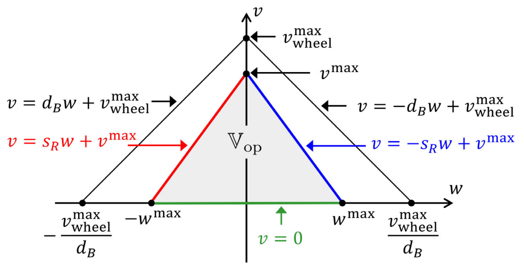
1) LOOKAHEAD POINT SELECTION

The lookahead distance determines the extent to which a lookahead point is located on a global path. The lookahead point is a temporary target position on the global path required by the robot to converge smoothly to a global path over time. In the proposed algorithm, the lookahead distance is determined by considering the current robot velocity, minimum distance from the closest obstacle, minimum distance to stop, and distance to the final goal position.

The maximum and minimum lookahead distances $d_l^{\max}$ and $d_l^{\min}$ are defined as

$$d_l^{\max} = \begin{cases} d_O^{\max}, & d_{\text{free}} \leq d_O^{\max} \\ d_F^{\max}, & d_{\text{free}} > d_O^{\max} \end{cases}$$
$$d_l^{\min} = s_L d_l^{\max} \tag{1}$$

where d_{free} is the minimum distance between the robot and the inflated obstacles with inflation distance d_R, $d_O^{\max}$ is the maximum range for considering the obstacles, $d_F^{\max}$ is the maximum lookahead distance for path following when there are no obstacles in the range of $d_O^{\max}$, and s_L ($0 < s_L \leq 1$) is a constant scaler that determines the ratio of $d_l^{\min}$ to $d_l^{\max}$. The inflation distance d_R can be determined as the radius of the circle that encompasses the robot with a safety margin. In (1), $d_O^{\max}$ must be sufficiently large for the local planner to forecast any collisions. However, a large



v
$v_{\text{wheel}}^{\max}$
$v^{\max}$
$v = d_B w + v_{\text{wheel}}^{\max}$
$v = -d_B w + v_{\text{wheel}}^{\max}$
$v = s_R w + v^{\max}$
$\mathbb{V}_{\text{op}}$
$v = -s_R w + v^{\max}$
w
$-\frac{v_{\text{wheel}}^{\max}}{d_B}$
$-w^{\max}$
$v = 0$
$w^{\max}$
$\frac{v_{\text{wheel}}^{\max}}{d_B}$

FIGURE 2. **A collection of all admissible velocities $\mathbb{V}_{\text{op}}$ is determined by s_R, which is the ratio of the maximum linear velicity $v^{\max}$ to the maximum angular velocity $w^{\max}$.**

lookahead distance increases the cross track error and may cause the robot to move inside the corner, which is called the "cutting corners" problem [36]. Thus, using a large lookahead distance is inefficient for agile path following when there are no obstacles within the specified range. For agile motion control to follow the global path with small cross track error and avoid the cutting corner problem, $d_F^{\max}$ ($d_F^{\max} < d_O^{\max}$) is used as the maximum lookahead distance when no obstacles exist within $d_O^{\max}$.

The lookahead distance d_l is defined as

$$d_l = \min(\max(d_l^v, d_{\text{stop}}), d_g) \tag{2}$$

where d_g denotes the distance to the goal, d_l^v is the velocity scaled lookahead distance, and d_{stop} is the minimum distance required for the robot to stop. To adjust d_l according to the current velocity of the robot d_l^v is defined as

$$d_l^v = (d_l^{\max} - d_l^{\min})\frac{v_r}{v^{\max}} + d_l^{\min} \tag{3}$$

where v_r and $v^{\max}$ are the current and maximum linear velocities of the robot, respectively. For the robot to stop before hitting an obstacle when the lookahead point is in the collision area, the stop distance d_{stop} is defined as

$$d_{\text{stop}} = \frac{v_r^2}{2\dot{v}^{\max}} + d_R \tag{4}$$

where $\dot{v}^{\max}$ is the maximum linear acceleration, and d_R is added to the safety margin. Using d_l, the lookahead point $\mathbf{p}_l$ is defined as

$$\mathbf{p}_l = f_{\text{look}}(d_l) \tag{5}$$

where $f_{\text{look}}(d_l)$ returns a single point toward the goal position on the global path that is d_l away from the robot.

2) PATH FOLLOWING VELOCITY GENERATION

The proposed approach only considers the operation of the robot in the forward direction, including in-place rotation. Thus, a collection of all admissible velocities $\mathbb{V}_{\text{op}}$, that the robot can attain during its operation is determined, as shown in Fig. 2, where $s_R = \frac{v^{\max}}{w^{\max}}$, and $v_{\text{wheel}}^{\max}$ and d_B are the maximum linear velocity of the wheel and the half distance between the left and right wheels, respectively.

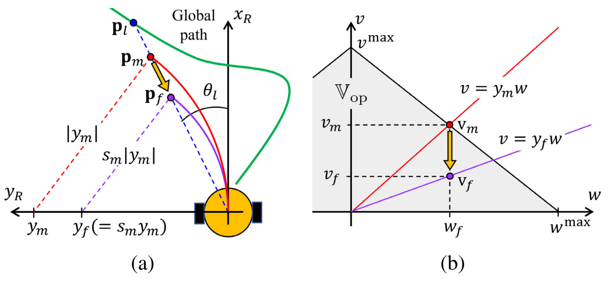


FIGURE 3. **Proposed path following velocity generation process. (a) Turning radius y_m is scaled down to y_f by the linear velocity scaler, s_m, when the robot is moving slowly or is stationary. (b) v_m is scaled down to v_f by s_m in the velocity space.**

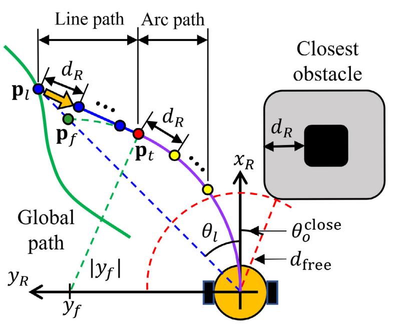


FIGURE 4. **Collision check of the local path starts at $\mathbf{p}_l$ and moves toward the robot with a step size of d_R. It terminates when a collision is detected or when it reaches the collision-free circle with a radius of d_{free}.**

The path following velocity generation process produces a target velocity for the agile path following by considering differential-drive kinematics. As the angle between the lookahead point and the forward direction of the robot approaches the threshold angle, the turning radius decreases. If it is greater than or equal to the threshold angle, the turning radius becomes zero for the in-place rotation.

Fig. 3 illustrates the proposed process for generating path following velocities, where θ_l denotes the lookahead angle and $\mathbf{p}_m$ $(0 \leq \|\mathbf{p}_m\| \leq d_l)$ refers to the intersection point between an arc of a radius y_m and a line connecting the lookahead point and the robot, as depicted in Fig. 3(a). Fig. 3(b) illustrates the determination of $\mathbf{v}_m$ $(= (w_m, v_m))$ by intersecting the line $v = y_m w$ and one of the edges of $\mathbb{V}_{\text{op}}$. The path following velocity is obtained by scaling y_m using a linear velocity scaler, which decreases the turning radius when v_r is small or θ_l is large.

The maximum threshold angle for path following $\theta_F^{\max}$ $(0 < \theta_F^{\max} \leq \pi)$ is defined to reduce the turning radius as $|\theta_l|$ approaches $\theta_F^{\max}$ and to drive the robot to rotate in place when $|\theta_l|$ exceeds $\theta_F^{\max}$. This can be tuned according to the task environment of the robot. For example, if a robot operates in an office environment with narrow corridors, then a small $\theta_F^{\max}$ is preferable, such that the robot can turn with a small radius. On the other hand, if the robot operates in a wide open space and a large turning radius does not affect the task performance, a large $\theta_F^{\max}$ value is preferable for turning corners smoothly at high speeds. Using $\theta_F^{\max}$, the normalized lookahead angle $\hat{\theta}_l$ is defined as

$$\hat{\theta}_l = f_{\text{norm}}(\theta_l, \theta_F^{\max}) \tag{6}$$

where

$$f_{\text{norm}}(\theta, \theta^{\max}) = \begin{cases} \dfrac{\pi}{2}\dfrac{\theta}{\theta^{\max}}, & |\theta| < \theta^{\max} \\ \dfrac{\pi}{2}, & \theta \geq \theta^{\max} \\ -\dfrac{\pi}{2}, & \theta \leq -\theta^{\max}. \end{cases} \tag{7}$$

From d_l and $\hat{\theta}_l$, y_m is generated as

$$y_m = f_{\text{rad}}(d_l, \hat{\theta}_l) \tag{8}$$

where $f_{\text{rad}}(d, \hat{\theta})$ is the radius generation function in (34) as described in the Appendix. As $|\theta_l|$ approaches $\theta_F^{\max}$, y_m decreases for the robot to enable a rotation with a smaller radius. When $|\theta_l|$ is equal to or greater than $\theta_F^{\max}$, y_m is zero for an in-place rotation. On the other hand, as $|\theta_l|$ approaches zero, $|y_m|$ approaches ∞ such that the robot moves straight to $\mathbf{p}_l$. Using θ_l and y_m, $\mathbf{v}_m$ $(= (w_m, v_m))$ is defined as

$$w_m = \frac{v^{\max}}{y_m + \text{sgn}(\theta_l)s_R}, \quad v_m = y_m w_m \tag{9}$$

where $\text{sgn}(\theta)$ is the sign function of θ and $s_R (= \frac{v^{\max}}{w^{\max}})$ is the ratio of $v^{\max}$ to $w^{\max}$.

If the robot moves slowly or is stationary, a small turning radius turn toward the goal can prevent collisions with obstacles around the robot. Therefore, the linear velocity scaler s_m $(0 \leq s_m \leq 1)$ is defined as

$$s_m = 1 - \frac{2}{\pi}|\hat{\theta}_l| f_{\text{hys}}(v_r) \tag{10}$$

with

$$f_{\text{hys}}(v_r) = \begin{cases} 1, & v_r < v_{\text{hys}}^{\min} \\ 0, & v_r > v_{\text{hys}}^{\max} \\ h^-, & v_{\text{hys}}^{\min} \leq v_r \leq v_{\text{hys}}^{\max} \end{cases} \tag{11}$$

where h^- is the previous output of $f_{\text{hys}}(v_r)$, and $v_{\text{hys}}^{\min}$ and $v_{\text{hys}}^{\max}$ are the lower and upper hysteresis thresholds, respectively. From $\mathbf{v}_m$ and s_m, the path following velocity $\mathbf{v}_f$ $(= (w_f, v_f))$ is defined as

$$\begin{aligned} w_f &= w_m, \\ v_f &= y_f w_m = s_m y_m w_m = s_m v_m \end{aligned} \tag{12}$$

where y_f $(= s_m y_m)$ denotes the turning radius, w_f is the angular velocity, which is the same as w_m, and v_f is the linear velocity obtained by scaling v_m into s_m. When v_r is higher than $v_{\text{hys}}^{\max}$, y_f is the same as y_m because s_m is one. If v_r is lower than $v_{\text{hys}}^{\min}$, then y_f becomes lower than y_m, as shown in Fig. 3, since s_m is less than one. This allows the robot to turn with a small radius when it moves slowly or remains stationary.

3) LOCAL PATH COLLISION CHECK

The local path to reach $\mathbf{p}_l$ consists of the line and arc paths, as shown in Fig. 4, where θ_o^{close} is the angle between the robot and the closest obstacle, and $\mathbf{p}_t$ is the tangent point between the line and arc paths with a turning radius y_f. The line path for the path following is the tangent line connecting $\mathbf{p}_l$ and $\mathbf{p}_t$. For the collision check of the local path in the path following process, the set of sampling points $\mathbb{P}_f$ is defined as

$$\mathbb{P}_f = f_{\text{path}}(\mathbf{p}_l, y_f) \tag{13}$$

where $f_{\text{path}}(\mathbf{p}, y)$ starts the sampling points from $\mathbf{p}_l$ and moves along the local path toward the robot with a step size of d_R until it reaches the collision-free circle of radius d_{free}. If all points in $\mathbb{P}_f$ are in a free configuration space $\mathbb{C}_{\text{free}}$, that is, $\mathbb{P}_f \subset \mathbb{C}_{\text{free}}$, the control velocity generation module calculates the control velocity using $\mathbf{v}_f$. Otherwise, the collision avoidance process is triggered.

B. COLLISION AVOIDANCE PROCESS

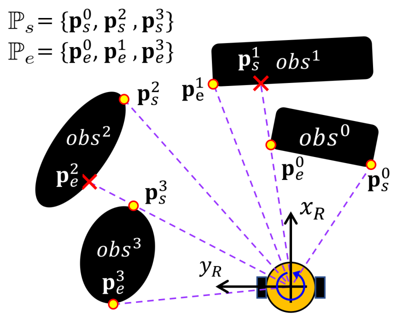
1) OBSTACLE CORNER DETECTION

The first step in the collision avoidance process is to detect the corners of the obstacles that are discontinuous points detected by distance measurement sensors, such as laser scanners, time-of-flight sensors, stereo cameras, and single image sensors with depth estimation. The corners are simply defined as discontinuous points in the distance between the robot and obstacles, thus requiring no information about the shape, number, or overlap of obstacles. Additionally, corners are exclusively used by the proposed algorithm to generate temporary target points to avoid collisions when predicting and preventing collisions.

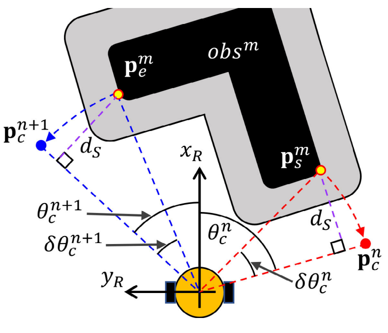
The corner detection is illustrated in Fig. 5(a) where $\mathbf{p}_s^n$ and $\mathbf{p}_e^n$ are the start and end corners of the n-th obstacle, respectively. The start corner set $\mathbb{P}_s$ and the end corner set $\mathbb{P}_e$ store the start and end corners, respectively, which are not occluded by other obstacles. When a previously detected obstacle occludes a new obstacle, only the end corner of the previously detected obstacle is added to $\mathbb{P}_e$, and the start corner of the new obstacle is ignored. For example, obs^0 occludes obs^1 in Fig. 5(a), so only $\mathbf{p}_e^0$ is added to $\mathbb{P}_e$, and $\mathbf{p}_s^1$ is ignored. On the other hand, if a previously detected obstacle (e.g., obs^2) is occluded by a new obstacle (e.g., obs^3), then only the start corner of a new obstacle (e.g., $\mathbf{p}_s^3$) is added to $\mathbb{P}_s$ and the end corner of the previous obstacle (e.g., $\mathbf{p}_e^2$) is ignored. For example, obs^2 is occluded by obs^3, as shown in Fig. 5(a), so only $\mathbf{p}_s^3$ is added to $\mathbb{P}_s$ and $\mathbf{p}_e^2$ is ignored.

2) SAFETY CORNER GENERATION

The collected corners in $\mathbb{P}_s$ and $\mathbb{P}_e$ can be utilized for the robot to bypass the obstacles. In the safety corner generation process, the corners of $\mathbb{P}_s$ and $\mathbb{P}_e$ are rotated clockwise and counterclockwise by a safety angle, respectively, to maintain a safety distance d_S ($d_S > d_R$) from the obstacles. For $\mathbf{p}_o(= d_o\angle\theta_o) \in \mathbb{P}_s \cup \mathbb{P}_e$, the polar form of the safety corner



(a)



(b)

FIGURE 5. (a) Start and end corners are stored to the start corner set, $\mathbb{P}_s$, and the end corner set, $\mathbb{P}_e$, that are not occluded by other obstacles, respectively. (b) Safety corners are generated by rotating the corners to keep the safety distance, d_S, from the obstacles.

$\mathbf{p}_c(= d_c\angle\theta_c)$ is defined as

$$d_c = d_o,\ \theta_c = \theta_o + \delta\theta_c \tag{14}$$

where

$$\delta\theta_c = \begin{cases} -\sin(\dfrac{d_S}{d_o}), & \mathbf{p}_o \in \mathbb{P}_s \\ \sin(\dfrac{d_S}{d_o}), & \mathbf{p}_o \in \mathbb{P}_e. \end{cases} \tag{15}$$

If $\mathbf{p}_c \in \mathbb{C}_{\text{free}}$ is satisfied, $\mathbf{p}_c$ is added to the safety corner set $\mathbb{P}_c$.

The generation of a safety corner is illustrated in Fig. 5(b), where $\mathbf{p}_c^n$ and $\delta\theta_c^n$ are the n-th safety corner and angle, respectively. The m-th obstacle obs^m has a start corner $\mathbf{p}_s^m$ and end corner $\mathbf{p}_e^m$. Because they are not occluded by other obstacles, $\mathbf{p}_s^m$ and $\mathbf{p}_e^m$ are stored in $\mathbb{P}_s$ and $\mathbb{P}_e$, respectively. For each corner to be away from the obstacle, the distances d_S, $\mathbf{p}_s^m$, and $\mathbf{p}_e^m$ are rotated to $\delta\theta_c^n$ clockwise and $\delta\theta_c^{n+1}$ counterclockwise to move to $\mathbf{p}_c^n$ and $\mathbf{p}_c^{n+1}$, respectively.

3) SAFETY CORNER COST ESTIMATION

The safety corner cost represents the estimated minimum time required for a robot to pass through the safety corner and reach the global path. Determining the cost of moving from a safety corner to a global path, a collision-free temporary lookahead point on the global path $\mathbf{p}_l^\star$ is defined as

$$\mathbf{p}_l^\star = f_{\text{look}}(nd_l) \tag{16}$$

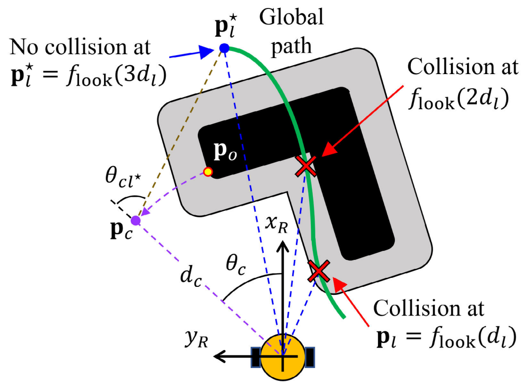

No collision at $\mathbf{p}_l^\star = f_{\text{look}}(3d_l)$
$\mathbf{p}_l^\star$
Global path
Collision at $f_{\text{look}}(2d_l)$
$\mathbf{p}_o$
$\theta_{cl^\star}$
$\mathbf{p}_c$
d_c
θ_c
x_R
y_R
Collision at $\mathbf{p}_l = f_{\text{look}}(d_l)$

FIGURE 6. Cost of the safety corner, $\mathbf{p}_c$, is the estimated minimum time for the robot to arrive at the temporary lookahead point, $\mathbf{p}_l^\star$, via $\mathbf{p}_c$.

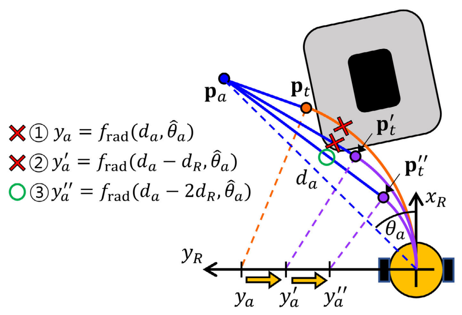

$\mathbf{p}_a$
$\mathbf{p}_t$
$\mathbf{p}_t'$
$\mathbf{p}_t''$
✗ ① $y_a = f_{\text{rad}}(d_a, \hat{\theta}_a)$
✗ ② $y_a' = f_{\text{rad}}(d_a - d_R, \hat{\theta}_a)$
○ ③ $y_a'' = f_{\text{rad}}(d_a - 2d_R, \hat{\theta}_a)$
d_a
x_R
θ_a
y_R
y_a
y_a'
y_a''

FIGURE 7. Local paths at the first and second iterations of collision check with d_a and $d_a - d_R$ as the avoidance lookahead distances are expected to collide. At the third iteration with $d_a - 2d_R$, the local path is collision-free, and thus y_a'' is used to generate the collision avoidance velocity.

where n is defined as a minium positive integer that guarantees a collision-free point of $\mathbf{p}_l^\star$. The cost of $\mathbf{p}_c$ with $\mathbf{p}_l^\star$ as the temporary lookahead point $f_{\text{cost}}(\mathbf{p}_c, \mathbf{p}_l^\star)$ is defined as

$$f_{\text{cost}}(\mathbf{p}_c, \mathbf{p}_l^\star) = \frac{d_c + \|\mathbf{p}_l^\star - \mathbf{p}_c\|}{v^{\max}} + \frac{|\theta_c| + |\theta_{cl^\star}|}{w^{\max}} \tag{17}$$

where

$$\theta_{cl^\star} = \cos^{-1}\left(\frac{\mathbf{p}_c \cdot (\mathbf{p}_l^\star - \mathbf{p}_c)}{\|\mathbf{p}_c\|\|\mathbf{p}_l^\star - \mathbf{p}_c\|}\right). \tag{18}$$

The first and second terms on the right side of (17) represent the estimated minimum time required for the robot to travel by distance and rotate by angle, respectively.

Fig. 6 shows an example of the safety corner cost generation for $\mathbf{p}_c$, where $\mathbf{p}_o \in \mathbb{P}_e$ is rotated counterclockwise to move toward $\mathbf{p}_c$. Considering n as the minimum integer that ensures a collision-free point for $f_{\text{look}}(nd_l)$, $\mathbf{p}^\star l$ is determined as $f_{\text{look}}(3d_l)$. For the robot to reach $\mathbf{p}_c$, the travel distance and angle are d_c and θ_c, respectively. For a robot to travel from $\mathbf{p}_c$ to $\mathbf{p}_l^\star$, the travel distance and angle are $\|\mathbf{p}_l^\star - \mathbf{p}_c\|$ and $\theta_{cl^\star}$, respectively. Thus, the estimated minimum time for the total travel distance with $v^{\max}$ and angle with $w^{\max}$ is $\frac{d_c+\|\mathbf{p}_c-\mathbf{p}_l^\star\|}{v^{\max}}$ and $\frac{|\theta_c|+|\theta_{cl^\star}|}{w^{\max}}$, respectively.

4) COLLISION AVOIDANCE VELOCITY GENERATION

The collision avoidance velocity generation process estimates the collision avoidance velocity to bypass obstacles and quickly reach a global path, as described in Algorithm 1. As shown in line 9 of Algorithm 1, the best safety corner with the lowest cost $\mathbf{p}_c^\star$ is selected to determine the collision avoidance point $\mathbf{p}_a$. The avoidance lookahead distance d_a is calculated in line 10 of the algorithm, where $d_a^{\min}$ in line 7 is the minimum avoidance lookahead distance. The avoidance scaler s_a ($0 \leq s_a \leq 1$) adjusts d_a and $d_a^{\min}$ using d_{free} and θ_a^{close}. As the robot approaches the obstacle than a distance of d_S and is heading toward the obstacle, s_a is decreased to zero to avoid collisions and rotate in place to head to $\mathbf{p}_a$.

The processes of adjusting d_a and checking the collision of the local path are described in lines 13 through 22. The turning radius y_a for collision avoidance is calculated in line 15, where $\hat{\theta}_a$ is the normalized angle θ_a with $\theta_A^{\max}$ ($0 < \theta_A^{\max} \leq \frac{\pi}{2}$) as the maximum threshold angle for collision avoidance. The sampled point set, $\mathbb{P}_a$, from the local path with $\mathbf{p}_a$ and y_a is calculated in line 16 to detect collisions along the

Algorithm 1 Collision Avoidance Velocity Generation

```
Input  : $\mathbb{P}_c$, $\mathbf{p}_l^\star$, $\theta_o^{\text{close}}$, $d_f$, $d_l$
Output: $\mathbb{V}_a$, $\mathbf{v}_a$
1  $w_a \leftarrow 0,\ v_a \leftarrow 0$
2  $\mathbf{v}_a \leftarrow (w_a, v_a),\ \mathbb{V}_a \leftarrow \{\mathbf{v}_a\}$
3  $s_o \leftarrow \min(\max(\frac{d_{\text{free}}}{d_S}, s_o^{\min}), 1.0)$
4  $s_a \leftarrow 1.0$
5  if $d_{free} < d_S$ then
6      $s_a \leftarrow \min(\frac{2|\theta_o^{\text{close}}|}{\pi}, 1.0)$
7  $d_a^{\min} \leftarrow \min(d_l, \max(s_a d_S, d_{\text{free}}))$
8  while $\mathbb{P}_c \neq \emptyset$ do
9      $\mathbf{p}_c^\star \leftarrow \underset{\mathbf{p}_c \in \mathbb{P}_c}{\text{argmin}} f_{\text{cost}}(\mathbf{p}_c, \mathbf{p}_l^\star)$
10     $d_a \leftarrow \min(d_l, s_a \max(d_c^\star - d_a^{\min}, 0.0) + d_a^{\min})$
11     $\theta_a \leftarrow \theta_c^\star$
12     $\mathbf{p}_a \leftarrow d_a \angle \theta_a$
13     while $d_a \geq d_a^{\min}$ do
14         $\hat{\theta}_a = f_{\text{norm}}(\theta_a, \theta_A^{\max})$
15         $y_a \leftarrow f_{\text{rad}}(d_a, \hat{\theta}_a)$
16         $\mathbb{P}_a \leftarrow f_{\text{path}}(\mathbf{p}_a, y_a)$
17         if $\mathbb{P}_a \subset \mathbb{C}_{free}$ then
18             $\mathbb{V}_a \leftarrow \{\mathbf{v} \in \mathbb{V}_{\text{op}} |\ v \leq s_o(s_R w + v^{\max}), v \leq -s_o(s_R w - v^{\max})\}$
19             $w_a \leftarrow \frac{s_o v^{\max}}{y_a + \text{sgn}(\theta_a) s_o s_R},\ v_a \leftarrow y_a w_a$
20             $\mathbf{v}_a \leftarrow (w_a, v_a)$
21             return $\mathbb{V}_a$, $\mathbf{v}_a$
22         $d_a \leftarrow d_a - d_R$
23     $\mathbb{P}_c \leftarrow \mathbb{P}_c \setminus \{\mathbf{p}_c^\star\}$
24 return $\mathbb{V}_a$, $\mathbf{v}_a$
```

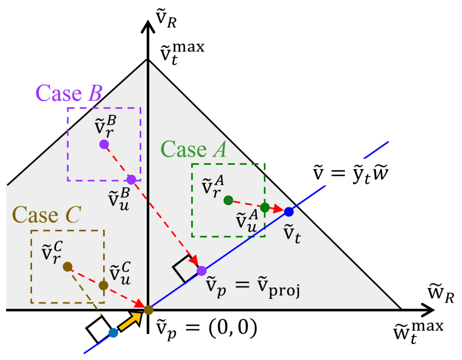


FIGURE 8. Three example cases (*A*, *B*, and *C*) of generating control velocities with different current velocities ($\mathbf{v}_r^A$, $\mathbf{v}_r^B$, and $\mathbf{v}_r^C$).

local path. If all the points in $\mathbb{P}_a$ are collision-free, the feasible velocity space for collision avoidance, $\mathbb{V}_a$, is calculated as indicated in line 18, where s_o ($s_o^{\min} \leq s_o \leq 1$) reduces $v^{\max}$ up to $s_o^{\min}$ times as the robot approaches the obstacles for safety. From $\mathbb{V}_a$ and y_a, the collision avoidance velocity $\mathbf{v}_a$ is determined as the cross-point between the linear function $v = y_a w$ and one of the edges of $\mathbb{V}_a$, as shown in line 19. Algorithm 1 is terminated by returning $\mathbb{V}_a$ and $\mathbf{v}_a$, as shown in line 21. However, if any collision is detected in the local path, d_a decreases by d_R each time, as indicated in line 21. If a collision-free local path is not found until $d_a < d_a^{\min}$, then $\mathbf{p}_c^\star$ is removed from $\mathbb{P}_c$, as shown in line 22, and the new best safety corner is selected for the subsequent iterations.

Fig. 7 illustrates the process in lines 12 through 22 of Algorithm 1. For the first trial with d_a, the local path crosses the obstacle because of a large y_a. Similarly, the local path from the second trial with $d_a - d_R$ passes through the obstacle. For the third trial with $d_a - 2d_R$, the local path is collision-free, and thus the turning radius, y_a'' is used to generate $\mathbb{V}_a$ and $\mathbf{v}_a$, as shown in lines 18 and 19, respectively, of Algorithm 1.

C. CONTROL VELOCITY GENERATION MODULE

1) VELOCITY SPACE NORMALIZATION

During the control velocity generation process, the target velocity space, target velocity, current velocity, and the maximum acceleration of the robot are considered to generate the control velocity that can be achieved within one control cycle time. The target velocity space $\mathbb{V}_t$ and the target velocity $\mathbf{v}_t$ are determined as

$$(\mathbb{V}_t, \mathbf{v}_t) = \begin{cases} (\mathbb{V}_{\text{op}}, \mathbf{v}_f), & \mathbb{P}_f \subset \mathbb{C}_{\text{free}} \\ (\mathbb{V}_a, \mathbf{v}_a), & \text{otherwise.} \end{cases} \tag{19}$$

If all the sampled points on the local path to the lookahead point are collision-free, $\mathbb{V}_{\text{op}}$ and $\mathbf{v}_f$ from the path following process are used as as $\mathbb{V}_t$ and $\mathbf{v}_t$, respectively. Otherwise, $\mathbb{V}_a$ and $\mathbf{v}_a$ from the collision avoidance process are used as $\mathbb{V}_t$ and $\mathbf{v}_t$, respectively.

Based on the current velocity and the maximum acceleration of the robot, a velocity window, which is a subset of $\mathbb{V}_t$ that can be reached in one control cycle time $\mathbb{V}_t^d$, is defined as

$$\mathbb{V}_t^d = \{\mathbf{v} \in \mathbb{V}_t | \, |w - w_r| \leq \delta w, |v - v_r| \leq \delta v\} \tag{20}$$

with

$$\delta w = \dot{w}^{\max} T_C, \quad \delta v = \dot{v}^{\max} T_C \tag{21}$$

where δw and δv are the angular and linear velocity steps, respectively, in one control cycle time T_C. $\dot{w}^{\max}$ and $\dot{v}^{\max}$ are the maximum angular and linear accelerations, respectively.

Because an arc path with turning radius $y_t = \frac{v_t}{w_t}$ guarantees collision-free travel, the control velocity must first reach the line $v = y_t w$ in the shortest time, and then move along the line to reach $\mathbf{v}_t$. To determine the target velocity path that guides the current velocity to the line $v = y_t w$ in the shortest time, the target velocity space is normalized by the velocity window. The normalized target velocity space, $\tilde{\mathbb{V}}_t$, is defined as

$$\tilde{\mathbb{V}}_t = \{\tilde{\mathbf{v}} = (\tilde{w}, \tilde{v}) | \, \tilde{w} = \frac{w}{\delta w}, \tilde{v} = \frac{v}{\delta v}, \mathbf{v} \in \mathbb{V}_t\}. \tag{22}$$

The target velocity path in $\tilde{\mathbb{V}}_t$ is the shortest line from the normalized current velocity $\tilde{\mathbf{v}}_r$ to the normalized line $\tilde{v} = \tilde{y}_t \tilde{w}$, where $\tilde{y}_t = \frac{\delta w}{\delta v} y_t$.

2) CONTROL VELOCITY GENERATION

In the normalized target velocity space, the normalized temporary target velocity $\tilde{\mathbf{v}}_p$ on the line $\tilde{v} = \tilde{y}_t \tilde{w}$ is defined as

$$\tilde{\mathbf{v}}_p = \begin{cases} \tilde{\mathbf{v}}_{\text{proj}}, & \tilde{v}_{\text{proj}} > 0 \\ (0, 0), & \tilde{v}_{\text{proj}} \leq 0 \end{cases} \tag{23}$$

where the projected point $\tilde{\mathbf{v}}_{\text{proj}}$ from $\tilde{\mathbf{v}}_r$ to the line $\tilde{v} = \tilde{y}_t \tilde{w}$, is defined as

$$\tilde{\mathbf{v}}_{\text{proj}} = \frac{\tilde{\mathbf{v}}_r \cdot \tilde{\mathbf{v}}_t}{\|\tilde{\mathbf{v}}_t\|^2} \tilde{\mathbf{v}}_t. \tag{24}$$

Because the proposed algorithm only considers forward motion, including in-place rotation, $\tilde{\mathbf{v}}_p$ is set to $(0, 0)$ when $\tilde{v}_{\text{proj}} \leq 0$. From $\tilde{\mathbf{v}}_p$, the temporary target velocity $\mathbf{v}_p$ is obtained as

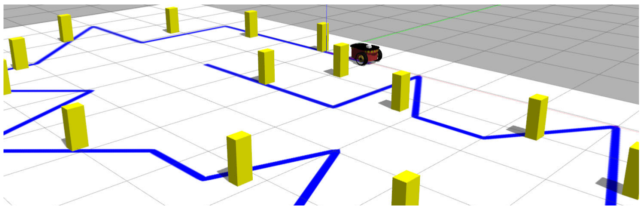
$$\mathbf{v}_p = (\delta w \tilde{w}_p, \delta v \tilde{v}_p). \tag{25}$$

From $\mathbf{v}_t$ and $\mathbf{v}_p$, the control velocity $\mathbf{v}_u$ is defined as

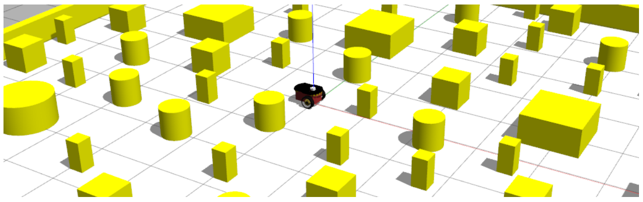
$$\mathbf{v}_u = \begin{cases} \underset{\mathbf{v} \in \mathbb{V}_t^d}{\operatorname{argmin}} \|\mathbf{v}_t - \mathbf{v}\|, & \mathbb{V}_t^d \cap \{\mathbf{v} | \, v = y_t w\} \neq \emptyset \\ \underset{\mathbf{v} \in \mathbb{V}_t^d}{\operatorname{argmin}} \|\mathbf{v}_p - \mathbf{v}\|, & \mathbb{V}_t^d \cap \{\mathbf{v} | \, v = y_t w\} = \emptyset. \end{cases} \tag{26}$$

If the current velocity reaches to the line $v = y_t w$ in one control cycle time, the point closest to $\mathbf{v}_t$ in the velocity window is selected as $\mathbf{v}_u$. Otherwise, the point closest to $\mathbf{v}_p$ is used as the control velocity to reach the line $v = y_t w$ in the shortest time and then moves on the line to finally reach $\mathbf{v}_t$.

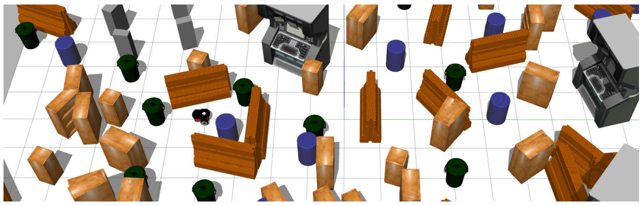
Fig. 8 shows the three example cases of generating control velocities for different current velocities $\mathbf{v}_r^A$, $\mathbf{v}_r^B$, and $\mathbf{v}_r^C$, and their normalized velocity windows are indicated by dashed



(a)



(b)



(c)

FIGURE 9. Screenshots of simulation scenarios: (a) Scenario 1, (b) Scenario 2, and (c) Scenario 3.

squares in the normalized target velocity space. In the case of A, $\mathbf{v}_t^A$ reaches line $v = y_t w$ in T_C. Thus, the control velocity $\mathbf{v}_u^A$ is selected from the velocity window $\mathbf{v}_r^A$ which can reach $\mathbf{v}_t$ within the shortest time. Cases B and C show that $v = y_t w$ cannot be reached from their current velocities in T_C such that both cases must use $\mathbf{v}_p$ as the temporary target velocity until their current velocities reach $v = y_t w$. For B, $\tilde{\mathbf{v}}_p$ is obtained by projecting $\tilde{\mathbf{v}}_r^B$ on the line $\tilde{v} = \tilde{y}_t \tilde{w}$ in the normalized target velocity space. However, for C, $\tilde{v}_{\text{proj}}$ is less than zero, and thus $(0, 0)$ is used as $\tilde{\mathbf{v}}_p$. Finally, cases B and C select the control velocities from their velocity windows that can reach $\mathbf{v}_p$ in the shortest time.

IV. EXPERIMENTS

In this section, we evaluate the performance of the proposed algorithm through a series of experiments. We conduct both simulation and real-world experiments to provide a comprehensive analysis. In both the simulation and real-world experiments, the robot navigates in an unknown environment without any prior knowledge to evaluate the robustness and effectiveness of the local planners. The simulation experiments are designed to compare our approach with existing methods in different scenarios, while the real robot experiment demonstrates the practicality of our approach in a real-world environment.

A. SIMULATION EXPERIMENTS

1) SIMULATION SCENARIOS

To compare the performances of four local planners (DWA [7], TEB [14], MPC [15], ASAP (proposed)), simulations were conducted with three scenarios. The screenshot and summary of the three scenarios are shown in Fig. 9 and Table 1, respectively.

TABLE 1. Characteristics of the three simulation scenarios.

	Scenario 1	Scenario 2	Scenario 3
Global path planner	Reference path	Straight-line	$A^\star$
Obstacle density	Low	High	High
Concave obstacles	×	×	◯

In Scenario 1, a predefined reference path consisting of a straight path and a sharp corner was given to the robot, with obstacles adjacent to the reference path, as shown in Fig. 9(a). Therefore, Scenario 1 was used to evaluate the local planners' performance in terms of reference path following and collision avoidance in a low obstacle density environment.

In Scenario 2, the robot was required to visit the sequentially provided waypoints in a cluttered environment, as shown in Fig. 9(b). For path planning, a straight-line (SL) planner was used to periodically generate a straight path from the robot to the waypoint. Because SL planner was not capable of generating collision-free paths, the local planners were responsible for the collision avoidance. Therefore, Scenario 2 was used to evaluate the collision avoidance ability of the local planners in a high obstacle density environment.

In Scenario 3, the robot was required to sequentially visit the provided waypoints in a cluttered environment with concave obstacles, as shown in Fig. 9(c). To prevent the local planners from being trapped by concave obstacles, an $A^\star$ planner was used for path planning. Although collision-free paths were provided by the $A^\star$ planner, robot's kinodynamic constraints were not considered during the path generation. Moreover, the path was updated rapidly upon the discovery of unknown obstacles. Therefore, Scenario 3 was used to evaluate the ability of local planners to quickly follow a given collision-free path, considering the robot's kinodynamic constraints and ensuring safe collision avoidance.

The performances of the local planners were compared for the three scenarios by computing the average and standard deviation of the evaluation metrics after executing each local planner ten times for each scenario. To ensure performance consistency, each local planner used the same parameters for all scenarios, where the parameters were tuned to ensure that each local planner could complete all scenarios.

2) SIMULATION SETUP

As illustrated in Fig. 10, the hardware-in-the-loop (HIL) simulation setup consists of a workstation to simulate the four scenarios and an embedded board to execute the navigation algorithms. In the experiment, NVIDIA Jetson Xavier NX was used as the embedded system. The workstation ran a virtual differential-drive robot equipped with a 2D LiDAR in a simulated environment using the Gazebo simulator, which is an open-source 3D robotics simulator, and transferred the

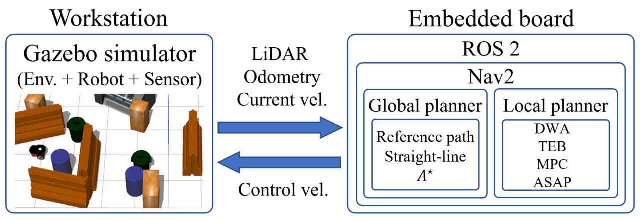
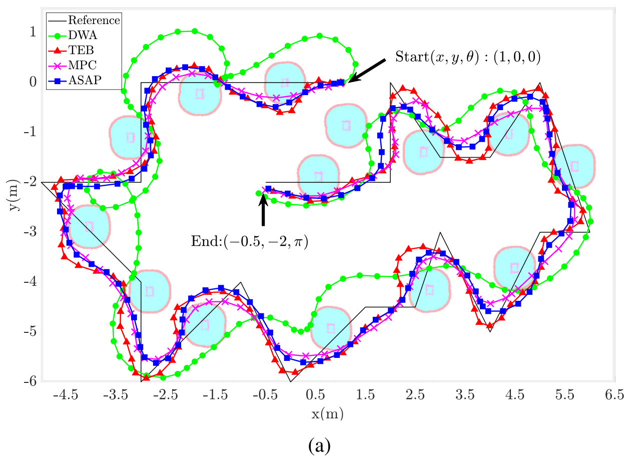


FIGURE 10. Hardware-in-the-loop simulation setup consists of a workstation to run the Gazebo simulator and an embedded board to run navigation algorithms.



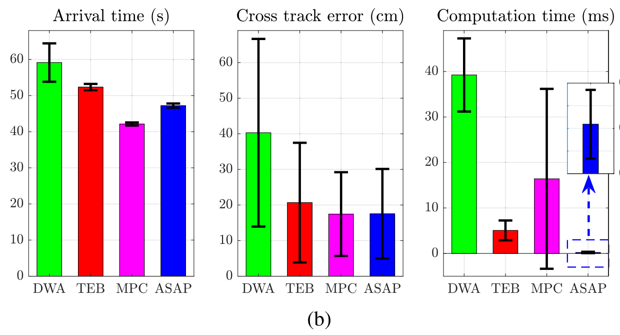


FIGURE 11. Simulation results from Scenario 1. (a) Robot trajectories and (b) arrival time, cross track error, and computation time of four local planners.

LiDAR measurement data, odometry, and velocity of the robot to the embedded board. The embedded board ran the global planner and local planner using Nav2, which is the navigation stack of robot operating system 2 (ROS 2), and transferred the control velocity to the workstation. The workstation and embedded board were connected to the same WiFi network and communicated with each other using the data-distribution service (DDS).

3) SIMULATION RESULTS FROM SCENARIO 1

In Scenario 1, the arrival time, cross track error (CTE), and computation time were used as performance metrics of the local planners. The arrival time is the time required for the robot to travel from the starting point to the end point along the reference path, as shown in Fig. 11(a). The CTE, which is the shortest distance from the robot to the reference path, was measured every 0.1 seconds until the robot reaches the end point. The average and standard deviation of CTE were used for the evaluation. The computation time is the time required by the local planner to generate the control velocity.

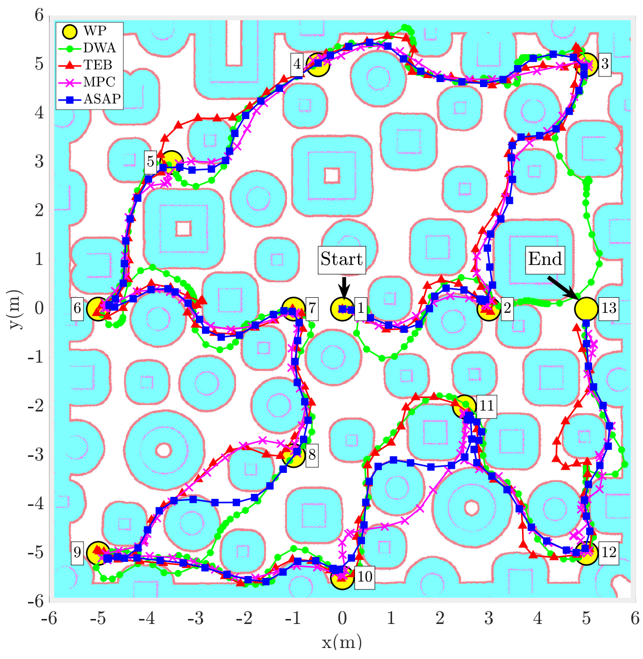
Fig. 11(a) shows the robot's trajectory using the four local planners. In the case of DWA, the turning radius was larger than that of the other local planners, which resulted in inefficient trajectories. In addition, DWA struggled to generate control speeds to avoid obstacles when in front of the robot. Consequently, DWA showed a larger arrival time and CTE than the other algorithms, as shown in Fig. 11(b). MPC showed superior results in terms of arrival time and CTE compared with the other algorithms. However, since MPC required more time to generate collision-free local paths through optimization when encountering obstacles, the standard deviation of the computation time was larger than that of the other algorithms. On the other hand, the proposed approach only ran the collision avoidance process when a collision was expected along the local path and chose the best safety corner to bypass obstacles and quickly converge to the reference path. As a result, the proposed approach exhibited a low CTE and a significantly faster computation time than DWA, TEB, and MPC, approximately 200, 25, and 80 times faster, respectively, as shown in Fig. 11(b).

4) SIMULATION RESULTS FROM SCENARIO 2

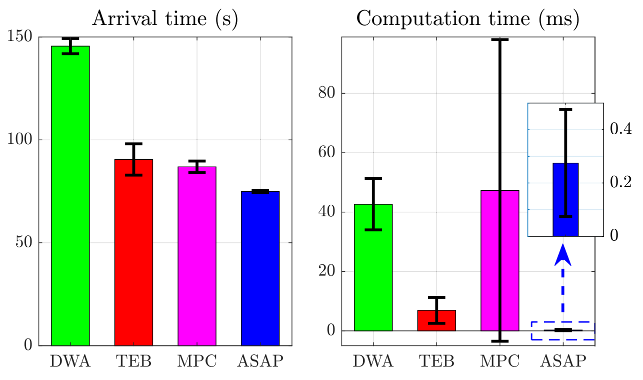
In Scenario 2, the arrival and computation times of the local planners were used as evaluation metrics. Because there was no reference path to follow, the CTE was excluded from the evaluation. Fig. 12(a) shows the robot's trajectory of visiting 13 waypoints using four local planners. Similar to the results of Scenario 1, DWA rotated with a large turning radius, resulting in a long arrival time, as shown in Fig. 12(b). In contrast, TEB, MPC, and the proposed approach rotated agilely depending on the waypoint location, as observed at waypoints 2, 6, and 9 in Fig. 12(a). In terms of computation time, TEB and MPC increased by more than two times compared with Scenario 1, as shown in Fig. 12(b). This is because both algorithms generated collision-free local paths through nonlinear optimization, and the computational load increases as the obstacle density increases. Similarly, the computation time of the proposed approach increased by approximately 1.5 times compared with Scenario 1. However, it still showed computation time that was 155, 25, and 170 times faster than DWA, TEB, and MPC, respectively, as shown in Fig. 12(b).

5) SIMULATION RESULTS FROM SCENARIO 3

In Scenario 3, the arrival and computation times of the local planners were used as evaluation metrics as in Scenario 2. However, unlike Scenarios 1 and 2, Scenario 3 contained more complex (irregularly shaped) obstacles, such as concave obstacles, that could cause the local planners to trap in local minima. To prevent this problem, an $A^\star$ planner that periodically generates collision-free paths was used.



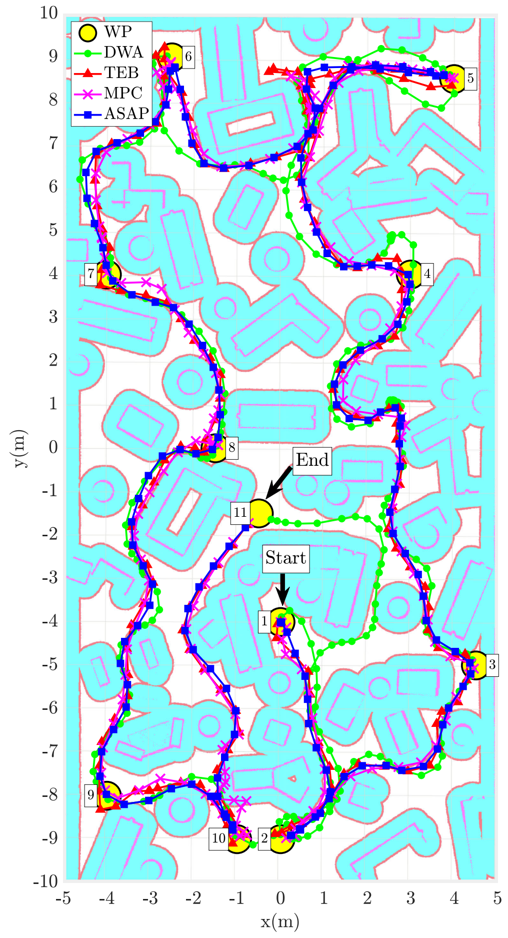
(a)



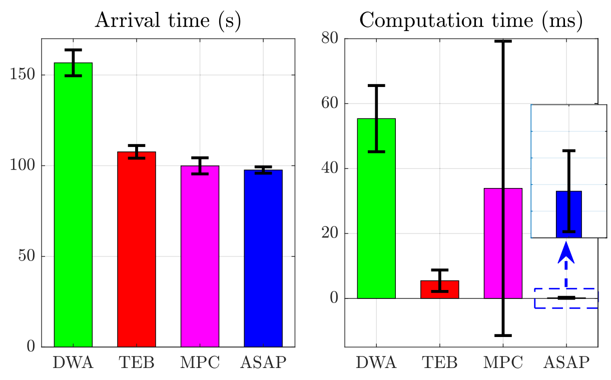
(b)

FIGURE 12. Simulation results from Scenario 2. (a) Robot trajectories and (b) arrival time, cross track error, and computation time of four local planners.

Fig. 13(a) shows the trajectory of a robot visiting 11 waypoints using the four local planners. Unlike the other local planners, DWA resulted in an inefficient trajectory when moving from waypoint 10 to 11 because of its large turning radius. Consequently, the DWA exhibited a longer arrival time than the others, as shown in Fig. 13(b). It can be observed from Fig. 13(b) that the computation times of TEB, MPC, and the proposed approach decreased by approximately 0.8, 0.7, and 0.6 times, respectively, compared with Scenario 2. This is because $A^{\star}$ provided collision-free paths to the local planner, thereby reducing the computational burden on the local planners to calculate collision-free local paths. On the other hand, the computation time of DWA increased by approximately 1.3 times. This demonstrated that even though a collision-free paths were provided, DWA suffered from generating a collision-free arc path in a cluttered environment. As in the computation time results of Scenarios 1 and 2, the proposed approach showed a significantly faster computation time than DWA, TEB, and MPC by approximately 316, 31, and 194 times, respectively, as shown in Fig. 13(b).



(a)

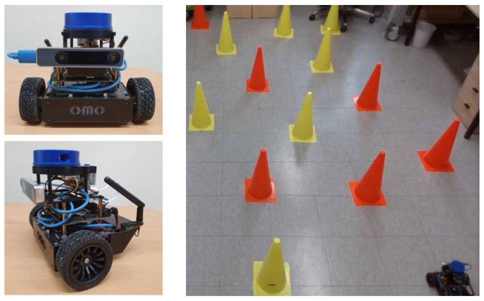


(b)

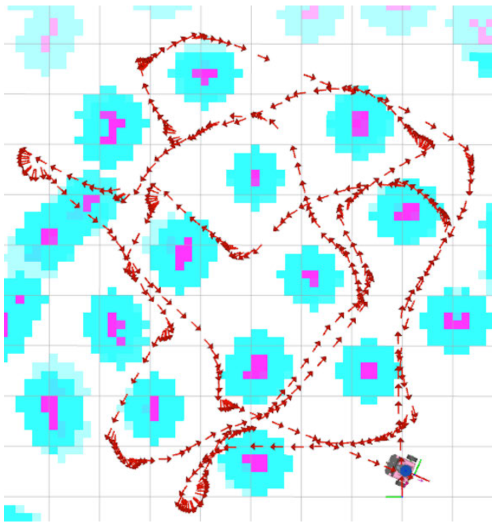
FIGURE 13. Simulation results from Scenario 3. (a) Robot trajectories and (b) arrival time, cross track error, and computation time of four local planners.

B. REAL ROBOT EXPERIMENT

The proposed algorithm was implemented in a mobile robot system as shown in Fig. 14(a). The system comprised



(a) (b)



(c)

FIGURE 14. (a) Differential-drive robot platform, (b) a cluttered environment with 20 obstacles and (c) robot trajectories from the experiment.

a mobile robot platform with differential-drive kinematics (OMO R1mini), an embedded computer (NVIDIA Jetson Xavier NX), a 2D LiDAR (YDLIDAR X4), and a tracking camera (Intel RealSense T265). For autonomous navigation, the SL planner and the proposed approach were used as the global and local planners, respectively. For localization, only visual-inertial odometry from the tracking camera was used without a prior map. To construct the cluttered experimental environment, 20 obstacles were evenly placed in the 5 m × 5 m area, as shown in Fig. 14(b). The human operator used the ROS visualization tool (Rviz) from a remote computer to send the target positions to the robot. When the target position was sent to the robot from the operator, the embedded computer on the robot ran the SL planner and the proposed approach to reach the target while avoiding collisions.

During the experiment, the human operator randomly selected 12 target positions and provided the subsequent target locations one by one when the robot reached the target. A video of the experiment is available in [37]. The trajectories of the robot is illustrated in Fig. 14(c), and the experiment result demonstrated that the proposed approach could reach all target positions in a cluttered environment quickly, smoothly, and safely, without collisions. When the target location was provided while the robot was stationary, it rotated in place and moved smoothly along the corner as the angle with the target location decreased. When collisions were expected, the robot decreased its linear velocity and moved toward a safety corner with a short radius for safe collision avoidance.

V. DISCUSSION

It was observed from the results of the three scenarios that the arrival time, CTE, and computation time of the DWA were higher than those obtained using the other algorithms. This is due to the fact that DWA necessitates a large number of velocity samples and a lengthy simulation time determining a local path that avoids collisions. Conversely, TEB requires less computation than DWA because it generates collision-free paths by adjusting the elastic band. However, TEB requires solving a non-convex optimization problem, which increases the computation when the obstacle density is high. Similarly, MPC is computationally expensive because it solves a constrained optimization problem in real-time to obtain the optimal control input sequence over a finite prediction horizon. Unlike other algorithms, the proposed approach separates path following from collision avoidance. It accurately follows the global path when there are no obstacles on the local path and utilizes safety corners to avoid obstacles, while quickly returning to the global path when a collision is detected along the local path. Moreover, the proposed approach does not rely on computationally expensive techniques such as velocity sampling and non-linear optimization, which makes it faster than the other algorithms.

While the proposed approach demonstrates promising performance in various scenarios, several limitations have been identified that suggest directions for future research and improvements. The algorithm is primarily designed for differential-drive robots capable of in-place rotation, making it less suitable for robots with Ackermann steering mechanisms. For such robots, the algorithm would need modifications to account for maximum steering angles and the possibility of reversing maneuvers. Addressing this limitation would involve developing an adapted version of the algorithm that can handle the kinematic constraints of Ackermann steering systems. The algorithm also assumes a collision-free global path and generates reactive velocity commands based on current sensor information. Consequently, if the global path is not collision-free and crosses concave obstacle areas, the robot can fall into a local minimum, leading to deadlock. To mitigate this, future enhancements could include implementing deadlock detection mechanisms and incorporating strategies such as global path re-planning or wall-following behaviors when a local minimum is detected. Addressing these limitations will be crucial for enhancing the versatility and robustness of the proposed algorithm.

VI. CONCLUSION

This study presents a novel local planning approach for autonomous mobile robots with three primary objectives, enabling agile path following and safe collision avoidance, ensuring computational efficiency for real-time performance in embedded systems, and facilitating intuitive parameter tuning for end-users. For path following, the proposed approach considers the kinematic constraint of the robot and generates the path following velocity based on the turning radius and hysteresis thresholding. For collision avoidance, the proposed approach generates safety corners, incurs the cost of bypass obstacles, and reaches the global path rapidly. From the target velocity, a control velocity is generated to rapidly reach the turning radius to the target from the velocity space based on the velocity window and kinodynamics of the robot. The HIL simulation experiments with thee scenarios demonstrated that the proposed approach is superior to DWA, TEB, and MPC in path following and collision avoidance with a significantly shorter computation time. Notably, the proposed algorithm demonstrates a computational speed that is 25 to 200 times faster compared to other existing algorithms. The real embedded robot experimental results also demonstrated its effectiveness in cluttered environments, indicating its potential for real-world applications.

Future work will include improving the proposed approach for applications in mobile robots with diverse drive mechanisms, such as Ackermann steering and omni-wheeled mobile robots. Modeling and avoiding dynamic obstacles are other major future research topics for robust local planning in crowds.

APPENDIX

The radius generation function $f_{\text{rad}}(d, \hat{\theta})$ generates the turning radius of the local path for a differential-drive robot to reach the goal point $\mathbf{p}$ $(= d\angle\theta)$ by considering $\theta^{\max}$ as the maximum angle. For an accurate and collision-free path following, $y = f_{\text{rad}}(d, \hat{\theta})$ must satisfy the following conditions:

1)

$$|y| \leq \frac{d}{2|\sin(\theta)|} \tag{27}$$

2)

$$\lim_{\hat{\theta}\to+0} y = \infty \text{ and } \lim_{\hat{\theta}\to-0} y = -\infty \tag{28}$$

3) For $y_a = f_{\text{rad}}(d, \hat{\theta}_a)$ and $y_b = f_{\text{rad}}(d, \hat{\theta}_b)$,

$$y_a \geq y_b \geq 0 \text{ for } 0 < \hat{\theta}_a \leq \hat{\theta}_b \leq \frac{\pi}{2} \text{ and}$$
$$y_a \leq y_b \leq 0 \text{ for } -\frac{\pi}{2} \leq \hat{\theta}_b \leq \hat{\theta}_a < 0 \tag{29}$$

The first condition guarantees that the arc path does not pass over the goal point, and the second condition indicates that the robot moves straight when θ approaches zero. In the third condition, $|y|$ decreases to zero as $|\theta|$ approaches $\theta^{\max}$, and the robot rotates in place when $|\theta| \geq \theta^{\max}$. To satisfy the aforementioned conditions, $f_{\text{rad}}(d, \hat{\theta})$ is defined as

$$f_{\text{rad}}(d, \hat{\theta}) = \alpha \cot(\hat{\theta}) \tag{30}$$

where α is the value of $f_{\text{rad}}(d, \hat{\theta})$ to satisfy all conditions. The remaining conditions, except for the first one, are satisfied as long as α is non-negative. When $|\hat{\theta}|$ is $\frac{\pi}{2}$, the first condition is satisfied regardless of α because $\cot(\pm\frac{\pi}{2})$ is zero. As $\hat{\theta}$ approaches zero, the leftside of (27) using (30) can be approximated as

$$|f_{\text{rad}}(d, \hat{\theta})| = \alpha|\cot(\hat{\theta})| = \alpha\left|\frac{\cos(\hat{\theta})}{\sin(\hat{\theta})}\right| \simeq \alpha\frac{1}{\frac{\pi}{2}\frac{|\theta|}{\theta^{\max}}} = \alpha\frac{2\theta^{\max}}{\pi|\theta|}. \tag{31}$$

Similarly, the rightside of (27) is approximated as $\frac{d}{2|\theta|}$. By substituting the leftside and rightside of (27) into (31) and $\frac{d}{2|\theta|}$, respectively, (27) can be approximated as

$$\alpha\frac{2\theta^{\max}}{\pi|\theta|} \leq \frac{d}{2|\theta|}. \tag{32}$$

Thus, α must satisfy the following inequality:

$$0 < \alpha \leq \frac{d\pi}{4\theta^{\max}}. \tag{33}$$

For the end of the arc path to reach the goal point as $|\hat{\theta}|$ approaches zero, $f_{\text{rad}}(d, \hat{\theta})$ is defined as

$$f_{\text{rad}}(d, \hat{\theta}) = \frac{d\pi}{4\theta^{\max}}\cot(\hat{\theta}). \tag{34}$$

REFERENCES

[1] K.-C. Chen, S.-C. Lin, J.-H. Hsiao, C.-H. Liu, A. F. Molisch, and G. P. Fettweis, "Wireless networked multirobot systems in smart factories," *Proc. IEEE*, vol. 109, no. 4, pp. 468–494, Apr. 2021.

[2] D. Lee, G. Kang, B. Kim, and D. H. Shim, "Assistive delivery robot application for real-world postal services," *IEEE Access*, vol. 9, pp. 141981–141998, 2021.

[3] M. M. Madebo, C. M. Abdissa, L. N. Lemma, and D. S. Negash, "Robust tracking control for quadrotor UAV with external disturbances and uncertainties using neural network based MRAC," *IEEE Access*, vol. 12, pp. 36183–36201, 2024.

[4] M. A. Arif, A. Zhu, H. Mao, X. Zhou, J. Song, Y. Tu, and P. Ma, "Design of an amphibious spherical robot driven by twin eccentric pendulums with flywheel-based inertial stabilization," *IEEE/ASME Trans. Mechatronics*, vol. 28, no. 5, pp. 1–13, 2023.

[5] Y. Kim, Y. Lee, S. Lee, J. Kim, H. S. Kim, and T. Seo, "STEP: A new mobile platform with 2-DOF transformable wheels for service robots," *IEEE/ASME Trans. Mechatronics*, vol. 25, no. 4, pp. 1859–1868, Aug. 2020.

[6] L. Jiang, S. Wang, Y. Xie, S. Xie, S. Zheng, J. Meng, and H. Ding, "Decoupled fractional supertwisting stabilization of interconnected mobile robot under harsh terrain conditions," *IEEE Trans. Ind. Electron.*, vol. 69, no. 8, pp. 8178–8189, Aug. 2022.

[7] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *Proc. ICRA Workshop Path Planning Costmaps*, 2008.

[8] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A navigation system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 2718–2725.

[9] W. Ayalew, M. Menebo, L. Negash, and C. M. Abdissa, "Solving optimal path planning problem of an intelligent mobile robot in dynamic environment using bidirectional rapidly-exploring random tree star-dynamic window approach (BRRT*-DWA) with adaptive Monte Carlo localization (AMCL)," *TechRxiv*, Dec. 2023.

[10] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, Sep. 1996, pp. 3375–3382.

[11] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[12] J. López, P. Sánchez-Vilariño, R. Sanz, and E. Paz, "Efficient local navigation approach for autonomous driving vehicles," *IEEE Access*, vol. 9, pp. 79776–79792, 2021.

[13] S. Yasuda, T. Kumagai, and H. Yoshida, "Safe and efficient dynamic window approach for differential mobile robots with stochastic dynamics using deterministic sampling," *IEEE Robot. Autom. Lett.*, vol. 8, no. 5, pp. 2614–2621, May 2023.

[14] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robot. Auto. Syst.*, vol. 88, pp. 142–153, Feb. 2017.

[15] C. Rösmann, A. Makarow, and T. Bertram, "Online motion planning based on nonlinear model predictive control with non-Euclidean rotation groups," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2021, pp. 1583–1590.

[16] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 3052–3059.

[17] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with AutoRL," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2007–2014, Apr. 2019.

[18] L. KU+000E4stner, J. Cox, T. Buiyan, and J. Lambrecht, "All-in-one: A DRL-based control switch combining state-of-the-art navigation planners," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 2861–2867.

[19] N. Yong Ko and R. G. Simmons, "The lane-curvature method for local obstacle avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. Innov. Theory, Pract. Appl.*, Sep. 1998, pp. 1615–1621.

[20] J. López, P. Sanchez-Vilariño, M. D. Cacho, and E. L. Guillén, "Obstacle avoidance in dynamic environments based on velocity space optimization," *Robot. Auto. Syst.*, vol. 131, Sep. 2020, Art. no. 103569.

[21] D. H. Lee, S. S. Lee, C. K. Ahn, P. Shi, and C.-C. Lim, "Finite distribution estimation-based dynamic window approach to reliable obstacle avoidance of mobile robot," *IEEE Trans. Ind. Electron.*, vol. 68, no. 10, pp. 9998–10006, Oct. 2021.

[22] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proc. IEEE Int. Conf. Robot. Autom.*, Aug. 1993, pp. 802–807.

[23] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *Proc. ROBOTIK 7th German Conf. Robot.*, May 2012, pp. 1–6.

[24] A. Richards and J. P. How, "Robust distributed model predictive control," *Int. J. Control*, vol. 80, no. 9, pp. 1517–1531, Sep. 2007.

[25] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey," *Robotica*, vol. 33, no. 3, pp. 463–497, Mar. 2015.

[26] F. Ke, Z. Li, and C. Yang, "Robust tube-based predictive control for visual servoing of constrained differential-drive mobile robots," *IEEE Trans. Ind. Electron.*, vol. 65, no. 4, pp. 3437–3446, Apr. 2018.

[27] S. Peicheng, L. Li, X. Ni, and A. Yang, "Intelligent vehicle path tracking control based on improved MPC and hybrid PID," *IEEE Access*, vol. 10, pp. 94133–94144, 2022.

[28] H. Xiao, Z. Li, C. Yang, L. Zhang, P. Yuan, L. Ding, and T. Wang, "Robust stabilization of a wheeled mobile robot using model predictive control based on neurodynamics optimization," *IEEE Trans. Ind. Electron.*, vol. 64, no. 1, pp. 505–516, Jan. 2017.

[29] L. Kästner, C. Marx, and J. Lambrecht, "Deep-reinforcement-learning-based semantic navigation of mobile robots in dynamic environments," in *Proc. IEEE 16th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2020, pp. 1110–1115.

[30] H. Shi, L. Shi, M. Xu, and K.-S. Hwang, "End-to-end navigation strategy with deep reinforcement learning for mobile robots," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, pp. 2393–2402, Apr. 2020.

[31] L. Kästner, T. Buiyan, L. Jiao, T. A. Le, X. Zhao, Z. Shen, and J. Lambrecht, "Arena-rosnav: Towards deployment of deep-reinforcement-learning-based obstacle avoidance into conventional autonomous navigation systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 6456–6463.

[32] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 5113–5120.

[33] K. E. Dagher, R. A. Hameed, I. A. Ibrahim, and M. Razak, "An adaptive neural control methodology design for dynamics mobile robot," *TELKOMNIKA (Telecommun. Comput. Electron. Control)*, vol. 20, no. 2, p. 392, Apr. 2022.

[34] W. Zhang, Y. Zhang, N. Liu, K. Ren, and P. Wang, "IPAPRec: A promising tool for learning high-performance mapless navigation skills with deep reinforcement learning," *IEEE/ASME Trans. Mechatronics*, vol. 27, no. 6, pp. 5451–5461, Dec. 2022.

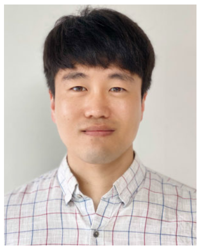
[35] W. Zhang and Y. F. Zhang, "Behavior switch for DRL-based robot navigation," in *Proc. IEEE 15th Int. Conf. Control Autom. (ICCA)*, Jul. 2019, pp. 284–288.

[36] A. Lombard, X. Hao, A. Abbas-Turki, A. E. Moudni, S. Galland, and A.-U.-H. Yasar, "Lateral control of an unmaned car using GNSS positionning in the context of connected vehicles," *Proc. Comput. Sci.*, vol. 98, pp. 148–155, Jan. 2016.

[37] D.-H. Lee, S. Choi, and K.-I. Na. (Jun. 9, 2022). *Video Demonstration of the Experiment*. Accessed: Dec. 6, 2022. [Online]. Available: https://youtu.be/1xq0Em9IJgo

DONG-HYUN LEE received the B.S. degree in electrical engineering from Kyungpook National University, Daegu, Republic of Korea, in 2007, and the M.S. and Ph.D. degrees in electrical engineering from KAIST, Daejeon, Republic of Korea, in 2009 and 2015, respectively. From 2015 to 2016, he was a Postdoctoral Researcher with the Intelligent Robotics Group (IRG), NASA Ames Research Center, Moffett Field, CA, USA. Since 2016, he has been with the School of Electronic Engineering and the Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi-si, Republic of Korea. His research interests include autonomous navigation, human–robot interaction, and multi-robot systems.

SUNGLOK CHOI (Member, IEEE) received the B.S. degree in mechanical and aerospace engineering from Seoul National University, Seoul, Republic of Korea, in 2006, and the M.S. and Ph.D. degrees in robotics from KAIST, Daejeon, Republic of Korea, in 2008 and 2019, respectively. From 2008 to 2021, he was a Research Scientist with the Electronics and Telecommunications Research Institute (ETRI), Daejeon. Since 2021, he has been with the Department of Computer Science and Engineering, Seoul National University of Science and Technology (SEOULTECH). His research interests include autonomous navigation, 3D computer vision, and robust regression.



KI-IN NA received the B.S. degree in mechanical engineering from Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea, in 2009, and the M.S. and Ph.D. degrees in robotics program from KAIST, Daejeon, Republic of Korea, in 2011 and 2022, respectively. Since 2011, he has been a Research Scientist with the Electronics and Telecommunications Research Institute (ETRI), Daejeon. His current research interests include detection and tracking of moving objects, socially-aware navigation, human–robot interaction, and artificial intelligence for real applications.