Check for updates

Decentralized nonlinear model predictive control-based flock navigation with real-time obstacle avoidance in unknown obstructed environments

Nuthasith Gerdpratoom and Kaoru Yamamoto*

Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University,
Fukuoka, Japan

This work extends our prior work on the distributed nonlinear model predictive control (NMPC) for navigating a robot fleet following a certain flocking behavior in unknown obstructed environments with a more realistic local obstacle-avoidance strategy. More specifically, we integrate the local obstacle-avoidance constraint using point clouds into the NMPC framework. Here, each agent relies on data from its local sensor to perceive and respond to nearby obstacles. A point cloud processing technique is presented for both two-dimensional and three-dimensional point clouds to minimize the computational burden during the optimization. The process consists of directional filtering and down-sampling that significantly reduce the number of data points. The algorithm's performance is validated through realistic 3D simulations in Gazebo, and its practical feasibility is further explored via hardware-in-the-loop (HIL) simulations on embedded platforms. The results demonstrate that the agents can safely navigate through obstructed environments, and the HIL simulation confirms the feasibility of deploying this scheme on an embedded computer. These results suggest that the proposed NMPC scheme is suitable for real-world robotics deployment in decentralized robotic systems operating in complex environments.

KEYWORDS

nonlinear MPC, flocking, local obstacle avoidance, hardware-in-the-loop, distributed control

1 Introduction

In the past few decades, multi-agent systems have gained much attention in science and engineering due to the advancement of digital computers and the advantages of mimicking group behaviors to accomplish complex tasks. Our study focuses on implementing flocking behavior for a robot fleet inspired by groups of birds and fish in nature. Reynolds (1987) introduced the boid model to simulate bird flocking with cohesion, separation, and alignment. Many researchers, including Cao et al. (2010), Li et al. (2024), Tanner et al. (2003a), Tanner et al. (2003b), and Olfati-Saber (2006), have utilized Reynolds' boid model from a control perspective. Recently, more advanced strategies have been proposed, employing optimization-based control, such as model predictive control (MPC), for flocking. This control scheme involves agents predicting future states and adjusting control actions based on minimizing the cost

function, offering benefits in handling complex problems, addressing state constraints, and enabling smoother control actions over longer prediction horizons (see Yu et al., 2021). Nonlinear model predictive control (NMPC) provides more flexibility than linear MPC, accommodating complex system dynamics and constraints without compromising nonlinearities.

Recent developments in optimization-based control have addressed challenges in multi-robot systems. One is the work by Xu et al. (2023). The authors proposed MPC problems for the leader–follower structure of unmanned aerial vehicles (UAVs). The leader UAV's problem was cast to track the specified trajectory, and a fully nonlinear UAV model was utilized to constrain the optimization. Meanwhile, the followers' problem was retaining the formation among the group of UAVs with limited neighbor information to ease inter-UAV communication. The model enforced for the followers was a simplified two-layer uncoupled model, including translational and rotational motion, to reduce the computational burden. The authors implemented MPC for the followers in the linear parameter varying (LPV) fashion (LPV-MPC). Kong et al. (2023) proposed an NMPC strategy specifically for fixed-wing UAV flocking. The authors presented a three-dimensional flocking model based on the distributed NMPC, encoding flocking rules as terms in the objective function. The NMPC was then solved using a nonlinear programming solver in the CasADi optimization framework (see Andersson et al., 2019).

Safe navigation has become a hot topic in robotics and control in recent years. Mestres et al. (2024) have proposed a distributed controller for multi-robot safe navigation. The authors characterized obstacle and collision avoidance by leveraging control barrier functions (CBFs) as optimization constraints. While the presented control scheme did not explicitly consider predicted future states and inputs like in predictive controls, the forthcoming system's behavior was captured with respect to the specified safe set. Goarin et al. (2024) integrated exponential CBFs (ECBFs) into a decentralized NMPC scheme for safely controlling multi-quadcopters under thrust constraints and limited detection range situations. This control strategy enhances the reliability of CBF-based safety conditions by incorporating them within a receding horizon framework. The authors derive both conservative and practical bounds on the detection range required to preserve ECBF-based safety guarantees. The optimization-based control strategies for navigating groups of robots in obstructed environments appear effective. However, most studies assumed that obstacles were well-defined. This paper aims to close the gap between these strategies and real-world robotics applications.

This work builds upon prior studies on NMPC-based flock navigation, particularly those presented by Nag et al. (2022) and Nag and Yamamoto (2024). The authors have introduced the NMPC-based flock navigation strategy with modified flocking rules consisting of a graph-distance hierarchy and a cohesion/alignment dynamic trade-off to navigate the fleet smoothly through an obstructed environment. The NMPC formulation allows one to explicitly impose each rule in the optimal control formulation, and it was successfully evaluated via numerical simulation and laboratory experimentation. However, in these works, it was assumed that an analytic expression of the obstacles (as a set of smooth inequalities proposed by Sathya et al. (2018)) becomes available to each agent as soon as they are in their detection range. In

other words, the obstacles were well-defined and did not reflect the obstacle representation in the actual robotics implementation. Therefore, this work further develops the theoretical foundations by investigating such practical aspects, which were previously neglected. In particular, we consider the situation in which the obstacles are perceived by local onboard sensors such as LiDAR and incorporate the whole process of obstacle detection, data suppression, and obstacle avoidance. We focus specifically on point cloud data, which represents the common information type for mobile robots' perception of the environment. The obstacle-avoidance algorithm based on a point cloud has been well studied. The most widely used algorithm, the artificial potential field proposed by Song and Kumar (2002), has been utilized in many mobile robot projects, such as the collision-free backup controller for a micro aerial vehicle based on 2D point cloud (see Lindqvist et al. (2020)) or the decentralized control of a robot group for manipulation tasks (see Barraquand et al. (1991)). Another reactive approach is vector field histogram (VFH), which maps the radial point cloud information in a polar coordinate into a histogram and selects the obstacle-free direction for the robot to steer to (see Liang et al. (2023)).

In this study, we incorporate obstacle information in point cloud data type into the optimal control formulation, adopting a strategy to minimize the amount of data input required by the solver to reduce the computational burden for both two and three-dimensional point clouds. The problem is solved using the proximal averaged Newton-type method for optimal control (PANOC), as introduced by Stella et al. (2017) via Optimization Engine (OpEn) code generation in Rust, developed by Sopasakis et al. (2020). The study is evaluated through a close-to-real-world simulation, Gazebo (see Koenig and Howard (2004)), where physical uncertainties are considered. The algorithm is executed in a distributed manner asynchronously with the robot operating system (ROS) framework, and the robot fleet navigates safely in an obstructed environment.

Unlike the experimental setup of Nag and Yamamoto (2024), in which the NMPC algorithm was executed in a decentralized manner on a station computer, providing optimal control actions to each robot, we further study the feasibility of implementing the algorithm in the embedded platforms, where computational resources are limited, through hardware-in-the-loop (HIL) simulation (see Brayanov and Stoynova (2019) and MihaliÄ et al. (2022)). We connect Raspberry Pis, widely used in low-cost robotics-embedded platforms, to a computer running a realistic simulation. The NMPC algorithm is run in a more distributed fashion inside the target hardware while simultaneously monitoring computational loads and assessing trajectory quality.

1.1 Contributions

There are two main contributions in this work. First, we introduce obstacle avoidance based on local environmental information that fits our original NMPC formulation. We also propose point cloud processing that can significantly reduce the computational burden for both two-dimensional and three-dimensional point clouds. Second, we conduct an HIL simulation to investigate the feasibility of implementing NMPC-based flock navigation on Raspberry Pi 4 in a fully distributed manner.

1.2 Outline

Section 2 briefly outlines the NMPC-based flock navigation with modified flocking rules, serving as a self-contained context for readers. Section 3 discusses the main results of this work, consisting of point cloud data processing, obstacle-avoidance constraints, and NMPC formulation featuring a local obstacle-avoidance strategy. Section 4 presents the evaluation of the algorithm through realistic simulation scenarios, as well as an HIL simulation conducted on embedded platforms.

2 Preliminaries

In this section, we will briefly explain our prior work on distributed NMPC-based flock navigation with modified flocking rules. The optimal control problem formulation and modified flocking rules will be discussed in the following subsections. A detailed explanation can be found in the works done by Nag et al. (2022) and Nag and Yamamoto (2024).

2.1 Setting

A system consisting of N agents in an n_p-dimensional space, categorized into leaders and followers, is considered. Leaders are given trajectories, while followers only react to their immediate surroundings and lack knowledge of any predefined destination. At time step t, let $\mathcal{N}_i^t$ be the index set of neighbors, including itself (Agent i), and let $\overline{\mathcal{N}}_i^t$ be the index set of neighbors without itself. Let $x_i^t \in \mathbb{R}^n$ be the state vector of Agent i and let y_i^t be the vector that contains Agent i's position $p_i^t \in \mathbb{R}^{n_p}$ and velocity $v_i^t \in \mathbb{R}^{n_p}$ in the global frame defined as $y_i^t = [p_i^{t\top}, v_i^{t\top}]^\top$. The variable sequences defined along the prediction horizon T are bold-faced. For instance, the sequence of the control input $u_i^t \in \mathbb{R}^{n_u}$, computed at time t, is $\boldsymbol{u}_i^t := u_i^{t|t} \dots u_i^{t+T-1|t}$. Similarly, the sequence of the predicted state of Agent i is $\boldsymbol{x}_i^t$. The nonlinear discrete-time state equation for each agent $f_i{:}\mathbb{R}^n \times \mathbb{R}^{n_u} \to \mathbb{R}^n$ can be described by Equation 1:

$$x_i^{t+k+1|t} = f_i\left(x_i^{t+k|t}, u_i^{t+k|t}\right), \quad k = 0, \dots, T-1. \tag{1}$$

The goal is for followers to find optimal actions $u_i^{t|t}$ for navigating through obstructed environments while minimizing deviation from the proposed flocking rules, thus ensuring fleet connectivity.

2.2 Objective function

To achieve our main objective, we introduced a quadratic minimum-effort cost function given by

$$J(\boldsymbol{u}_i^t) = \|\boldsymbol{u}_i^t\|_{R_i}^2 + \sum_{k=0}^{T-1} \gamma^k \|y_i^{t+k+1|t} - \bar{y}_i^{t+k+1|t}\|_{Q_i^t}^2, \tag{2}$$

where $R_i \in \mathbb{R}^{n_u \times n_u}$, and $Q_i^t \in \mathbb{R}^{2n_p \times 2n_p}$ are positive semi-definite diagonal weight matrices corresponding to the first and second term. Let $y_{j|i}^t$ be the output of Agent $j \in \overline{\mathcal{N}}_i^t$, detected by Agent i at time

instant t, and let $\bar{y}_i^{t+k|t} = [\bar{p}_i^{t+k|t\top}, \bar{v}_i^{t+k|t\top}]^\top$ be the weighted average of $y_{j|i}^{t+k|t}$, where

$$\begin{aligned} \bar{p}_i^{t+k|t} &= \sum_{j \in \mathcal{N}_i^{t+k|t}} w_{\mathrm{p},j|i}^{t+k|t} p_{j|i}^{t+k|t}, \\ \bar{v}_i^{t+k|t} &= \sum_{j \in \mathcal{N}_i^{t+k|t}} w_{\mathrm{v},j|i}^{t+k|t} v_{j|i}^{t+k|t}, \end{aligned} \tag{3}$$

for $k = 0, \dots, T-1$. The weights $w_{\mathrm{p},j|i}^{t+k|t}$ and $w_{\mathrm{v},j|i}^{t+k|t}$ will be discussed in the modified flocking rules section (Section 2.4). The errors of the predicted states can be accumulated during the prediction. Hence, a discount factor $\gamma \in (0, 1]$ is introduced by prioritizing the near-future prediction.

2.3 Optimization constraints

2.3.1 State and input constraints

The input and state constraints, excluding separation and obstacle avoidance, are assigned as

$$x_i^{t+k+1|t} \in \mathcal{X}_i \quad \text{and} \quad u_i^{t+k|t} \in \mathcal{U}_i, \tag{4}$$

where $\mathcal{X}_i \subset \mathbb{R}^n$ and $\mathcal{U}_i \subset \mathbb{R}^{n_u}$ are the feasible sets for the states and inputs of Agent i, respectively.

2.3.2 Separation constraint

To ensure separation between agents, both hard and soft constraints are considered to address the accumulated error during prediction. That is, in the early stages, separation is enforced by a hard constraint, and after a predefined time $T_{\mathrm{sep}} \in (0, T]$, it switches to a soft constraint. For a detailed justification, the reader is referred to Nag and Yamamoto (2024). Hard separation constraints are imposed as

$$d_{i,\mathrm{sep}} - d_{j|i}^{t+k+1|t} \le 0, \quad k = 0, \dots, T_{\mathrm{sep}} - 1, \tag{5}$$

where $d_{j|i}^{t+k+1|t} := \|p_i^{t+k+1|t} - p_{j|i}^{t+k+1|t}\|^2$ for $j \in \overline{\mathcal{N}}_i^{t+k|t}$, and $d_{i,\mathrm{sep}}$ is a separation distance. To add the soft constraint to the cost function, we define the penalty function as Equation 6:

$$P(d_i^t) = \begin{cases} \left(d_{i,\mathrm{sep}} - d_{j|i}^t\right)^2 & \text{if } d_{i,\mathrm{sep}} > d_{j|i}^t, \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

Then the objective function, Equation 2, will be modified as shown by Equation 7:

$$\tilde{J}(\boldsymbol{u}_i^t) = J(\boldsymbol{u}_i^t) + \rho_{\mathrm{sep}} \sum_{k=T_{\mathrm{sep}}+1}^{T-1} \gamma^k \sum_{j \in \overline{\mathcal{N}}_i^{t+k|t}} P\left(d_{j|i}^{t+k|t}\right), \tag{7}$$

where $\rho_{\mathrm{sep}} \in \mathbb{R}_+$ acts as a penalty weight for a soft separation constraint.

2.4 Modified flocking rules

This section will discuss the subjective neighbor weights and the weighted matrix Q_i^t. The effect of each rule was explained in the prior work (see Nag and Yamamoto, 2024).

2.4.1 Leader−follower graph-distance hierarchy

A hierarchy level subjected to each agent $\pi_i^t \in \mathbb{N}$ was introduced. In the initialization process, each leader l is set a constant hierarchy level $\pi_l^t = 0$. Each follower is assigned an upper-bound hierarchy level $\bar{\pi} \in \mathbb{N}_{\geq 1}$, which can be defined as a flock parameter set by the designer. Then, the hierarchy level of each follower will be updated in every sample by the following equation:

$$\pi_i^t = \min \left\{ \bar{\pi}, 1 + \min_{j \in \overline{\mathcal{N}}_i^t} \pi_j^{t-1} \right\}, \qquad (8)$$

while the leaders' hierarchy levels remain zero. In this way, each agent can estimate its current hierarchy level in a completely distributed manner. By communicating hierarchy levels between neighbors, each agent can identify the more important agents (e.g., leaders or followers near a leader). This information is incorporated into the position update rule by setting weight $w_{p,j|i}^{t+k|t}$ in Equation 3 according to Equation 9:

$$w_{p,j|i}^{t+k|t} = \frac{2^{-\pi_j}}{\sum_{\ell \in \mathcal{N}_i^{t+k|t}} 2^{-\pi_\ell}}. \qquad (9)$$

2.4.2 Allocating weights by travel vector

The alignment weight should be prioritized for the agents in the front with respect to travel direction. This can be assessed by examining the inner product between its velocity and the relative position vector. The alignment weight $w_{v,j|i}^{t+k|t}$ can be defined as

$$w_{v,j|i}^{t+k|t} = \begin{cases} 1 & \text{if } \langle v_i^{t|t}, p_{j|i}^{t-1|t-1} - p_i^{t|t} \rangle \geq 0 \\ \beta_i & \text{otherwise} \end{cases}, \qquad (10)$$

where $\beta_i \in [0,1]$.

2.4.3 Cohesion/alignment dynamic trade-off

For each agent to have the ability to determine whether to prioritize cohering or aligning with its neighbor, the cohesion and alignment dynamic trade-off is implemented by dynamically adjusting the weight matrix $Q_i^t \in \mathbb{R}^{2n_p \times 2n_p}$ according to Equation 11:

$$Q_i^t := \text{diag}\left(1 - q_i^t, \ldots, 1 - q_i^t, q_i^t, \ldots, q_i^t\right) \qquad (11)$$

with

$$q_i^t := \frac{q_{i,\text{st}}}{1 + c_i \|p_i^{t|t} - \bar{p}_i^{t|t}\|^2}, \qquad (12)$$

where $c_i \in \mathbb{R}_+$, and $q_{i,\text{st}} \in (0,1)$.

3 Obstacle avoidance based on local sensor

Point cloud data from the depth camera or LiDAR sensor represents an environment around it in a robot's body frame, including obstacles and neighboring agents. For two-dimensional LiDAR, the number of data points can be large, depending on the sensor specification, and the number is squared for 3D LiDAR. In the optimization problem, it is not always possible to impose a high number of constraints due to the limitation of computational power. Hence, in this section, we present a point cloud processing technique for reducing the number of data points to a feasible range. Then, we define an obstacle-avoidance constraint tailored to this processed data, aligning with our previously developed NMPC-based algorithm. The NMPC problem formulation with the proposed strategy is shown at the end of the section.

3.1 Processing of point cloud information

In point cloud processing, we utilize directional filtering and down-sampling, which can significantly reduce the number of data points while preserving vital features of the point cloud raw data. Because we do not consider the detected neighboring agents to be obstacles, the neighbor exclusion will be discussed in this section. The details for each process will be described in the following subsections.

We denote a set of processed point cloud's index perceived by Agent i at time t that will be fed into the NMPC solver as $\mathcal{O}_i^t$. The symbol $\mathcal{O}_{\text{raw},i}^t$ refers to the index set of raw point cloud data from Agent i's sensor before any processing. Furthermore, we define $\mathcal{O}_{f,i}^t$ and $\mathcal{O}_{s,i}^t$ as the index sets of the processed point cloud after directional filtering and down-sampling, respectively.

Because most of the sensors perceive the environment in their body frame, the point data are analyzed in the agent's body frame, where they are defined as follows: the x-axis points forward in the direction of its primary movement or orientation, representing its heading. The z-axis points upward, perpendicular to the plane of movement or operation. The y-axis completes the right-hand coordinate system, pointing to the robot's left side, orthogonal to the x-axis (forward) and the z-axis (up). Consequently, the robot's orientation can be briefly defined using the Euler angle (ϕ, θ, ψ), where ϕ_i^t denotes roll (rotation about the x-axis), θ_i^t signifies pitch (rotation about the y-axis), and ψ_i^t represents yaw (rotation about the z-axis) of the ith agent at time t. Furthermore, vectors observed from the agents' body frame are denoted with the subscript b.

3.1.1 Directional filtering

Directional filtering is a step in the proposed point cloud processing to reduce the number of point cloud data points to be processed, allowing each agent to selectively process only the subset of environmental data relevant to its current motion intent. This strategy improves computational efficiency and robustness in dynamic and cluttered environments.

Each follower Agent i moves by minimizing a local objective function (2) that encourages alignment with a desired direction of travel, represented by the weighted average position $\bar{p}i^{t|t}$ of its neighboring agents $\mathcal{N}i^t$. Intuitively, Agent i should prioritize environmental features that lie in the direction of this intended movement and disregard those in the opposite direction.

To formalize this directional prioritization, we define a reference plane in the body frame of Agent i. The normal vector of this plane is given by Equation 13:

$$\bar{p}_{b,i}^{t|t} = M(\phi_i^t, \theta_i^t, \psi_i^t) \cdot \left(\bar{p}_i^{t|t} - p_i^t\right), \qquad (13)$$

where $M(\phi_i^t, \theta_i^t, \psi_i^t)$ is a rotation matrix corresponding to the Euler angles of the ith agent with respect to the global frame. This

FIGURE 1
The illustration of the reference plane in the directional filtering process.
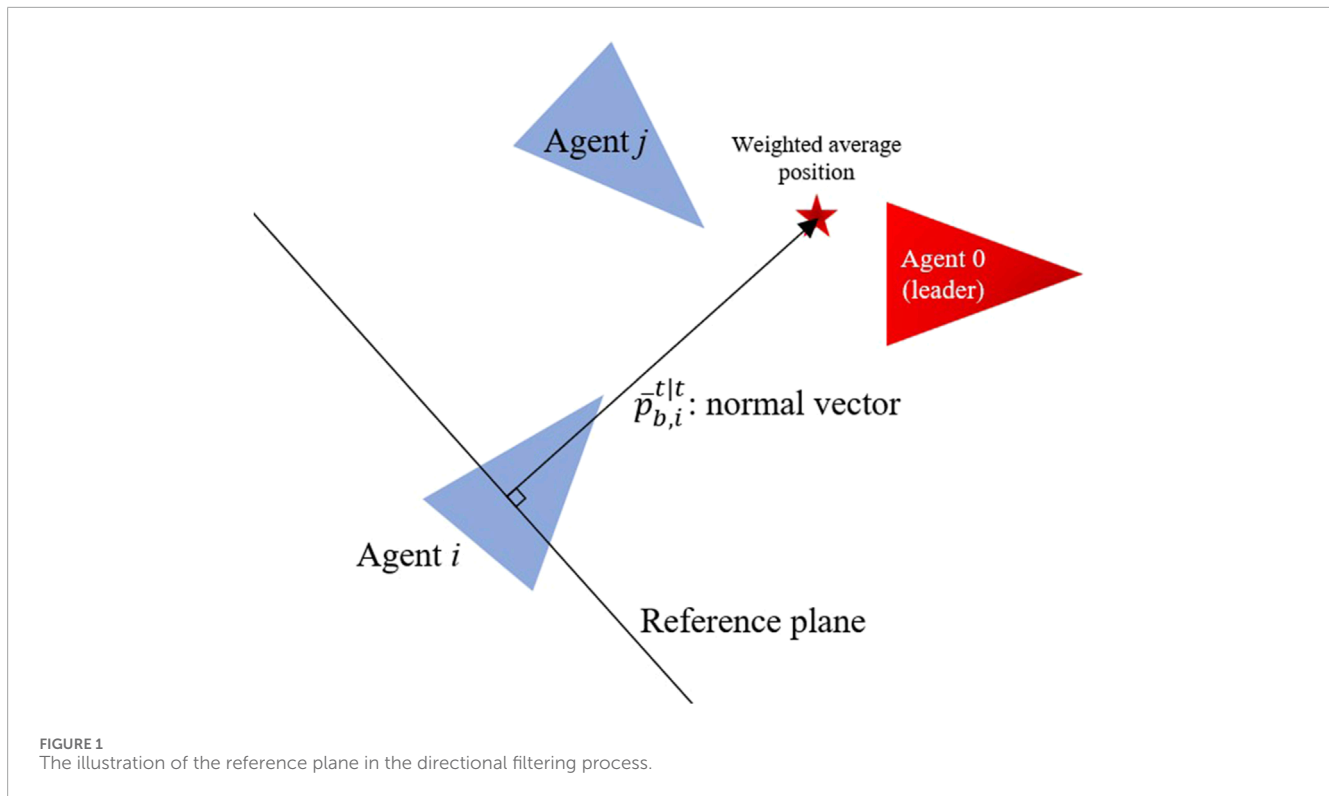
transforms global frame vectors into Agent i's body frame. The reference plane passes through the origin of the body frame (i.e., the agent's center) and is orthogonal to $\bar{p}_{b,i}^{t|t}$. Data points in front of this plane (in the general direction of motion) are considered relevant.

Consider $p_{b,q|i}^t \in \mathbb{R}^{n_p}$ as the position vector of a data point detected by Agent i's sensor in its body frame, for $q \in \mathcal{O}_{raw,i}^t$. Subsequently, we define a new subset $\mathcal{O}_{f,i}^t \subset \mathcal{O}_{raw,i}^t$, representing the index set of point cloud data located on the positive side of a reference plane with normal vector $\bar{p}_{b,i}^{t|t}$. Accordingly, a data point $p_{b,q|i}^t$ belongs to $\mathcal{O}_{f,i}^t$ if and only if its inner product with $\bar{p}_{b,i}^{t|t}$ is positive. A filtered set of indices is defined according to Equation 14:

$$\mathcal{O}_{f,i}^t = \left\{ q \in \mathcal{O}_{raw,i}^t \,\middle|\, \langle \bar{p}_{b,i}^{t|t}, p_{b,q|i}^t \rangle \geq 0 \right\}, \qquad (14)$$

which includes only the points located in front of the reference plane. This dot-product condition ensures that only the data aligned with or ahead of the movement direction are retained.

The illustration portraying the reference plane and normal vector in the directional filtering process is displayed in Figure 1, while a comparison of point cloud visualization in RViz between the before and after situations is depicted in Figure 2.

3.1.2 Down-sampling

In this process, data points are uniformly neglected by grouping nearby points and selecting the one closest to the robot from each group. Then, the indices of the selected points are assigned to a down-sampled point cloud index set $\mathcal{O}_{s,i}^t \subset \mathcal{O}_{f,i}^t$.

For two-dimensional point clouds, such as laser scan data, the index $q \in \mathcal{O}_{f,i}^t$ is organized based on spatial positioning. That is, the grouping process can be done by partitioning the polar coordinated laser scan data into sectors and selecting the point closest to Agent

i. A down-sampled index set can be represented as follows:

$$\mathcal{O}_{s,i}^t = \left\{ \arg\min_q \left\{ r_q \,\middle|\, q \in \mathcal{O}_{f,i}^t[i_s{:}i_s+f_s] \right\} \,\middle|\, i_s \in \{0, f_s, 2f_s, \dots, n_s f_s\} \right\}, \qquad (15)$$

where $f_s \in \mathbb{R}$ is the down-sample factor and $n_s = \lfloor (|\mathcal{O}_{f,i}^t| - 1)/f_s \rfloor$. The index set $\mathcal{O}_{f,i}^t[i_s{:}i_s+f_s]$ denotes the segment for f_s consecutive index elements in $\mathcal{O}_{f,i}^t$ starting at index i_s and ending with $i_s + f_s$. The distance r_q ranges from the center of Agent i to the qth point data.

For three-dimensional point clouds, where the data points can be gathered from a depth camera or 3D LiDAR, the perceived data are unstructured, unlike in the two-dimensional case. A widely used method for down-sampling such data is voxel grid filtering, proposed by Shi and Luo (2024) and Xu et al. (2021). In this work, a modified voxel grid filtering technique will be utilized by prioritizing the closest point in each voxel. The algorithm creates voxel volume grids over the point cloud data, and the down-sampled point for each grid will be the one closest to the ith agent. Let $p_{b,q|i}^t = \left[x_{b,q|i}^t, y_{b,q|i}^t, z_{b,q|i}^t \right]^\top$ be a position of the obstacle $q \in \mathcal{O}_{f,i}^t$ in three-dimensional space observed from Agent i's body frame and let $V{:}\mathbb{N} \to \mathbb{Z}^3$ be a function that maps the obstacle's index to its corresponding voxel coordinates in a three-dimensional grid space defined according to Equation 16:

$$V(q) = \left[\left\lfloor \frac{x_{b,q|i}^t}{\delta_x} \right\rfloor, \left\lfloor \frac{y_{b,q|i}^t}{\delta_y} \right\rfloor, \left\lfloor \frac{z_{b,q|i}^t}{\delta_z} \right\rfloor \right]^\top, \qquad (16)$$
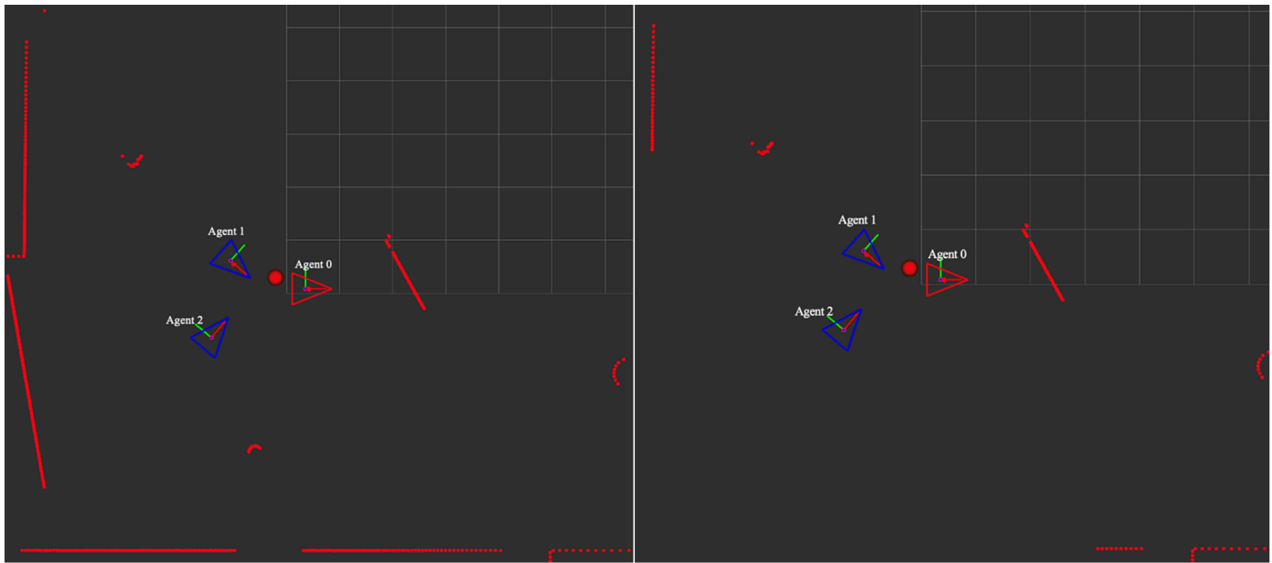
FIGURE 2
Agent 2's raw point cloud visualization (left). Agent 2's point cloud visualization after employing directional filtering (right). A red circle represents the weighted average among three robots, where the position of the leader agent (red triangle) is given more weight.
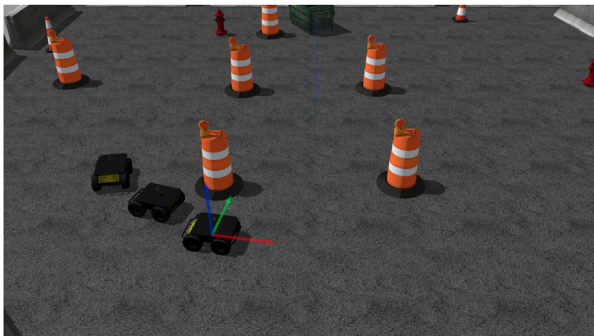

FIGURE 3
The Gazebo 3D simulation of three Husky UGVs from Clearpath Robotics (2015) in an obstructed environment by Mukherjee (2015).

where δ_x, δ_y, and δ_z denote voxel grid size. The set expression can be represented as

$$\mathcal{O}_{s,i}^t = \left\{ q \in \mathcal{O}_{f,i}^t \,|\, \nexists q' \in \mathcal{O}_{f,i}^t : \quad (V(q) = V(q')) \wedge \quad (q \neq q') \wedge \left(\|p_{b,q'|i}^t\|^2 < \|p_{b,q|i}^t\|^2 \right) \right\}. \quad (17)$$

According to the above equation, the set $\mathcal{O}_{s,i}^t$ consists of the indices q for which there is no other index q' in the set $\mathcal{O}_{f,i}^t$ (after the directional filtering process) such that the obstacle corresponding to index q' is in the same voxel as obstacle q, and is closer to Agent i.

3.1.3 Neighbor exclusion

The point cloud data perceived via a sensor contain obstacles and neighboring agents. The neighbors could be considered dynamic

obstacles, as presented by Lindqvist et al. (2021). However, in this work, the imposed separation rule was highlighted. Thus, the detected neighbors in the point cloud index can be neglected to reduce the number of data points.

This is the final stage of the point cloud processing, where the set of processed data $\mathcal{O}_i^t$ will be obtained and then used in the optimal control problem. The set expression of the processed point cloud is expressed as Equation 18:

$$\mathcal{O}_i^t = \left\{ q \in \mathcal{O}_{s,i}^t | \|p_{q|i}^t - p_{j|i}^{t|t}\| > r_b, j \in \overline{\mathcal{N}}_i^{t|t} \right\}, \quad (18)$$

where r_b is the farthest distance within the agent's body from its center.

3.2 Obstacle-avoidance constraint

To ensure obstacle-free trajectories along the finite prediction horizon for each agent, we impose an obstacle-avoidance constraint. Because obstacles are defined as processed point clouds in n_p-dimensional space, let $p_{m|i}^t \in \mathbb{R}^{n_p}$ be an mth obstacle's position sensed by the ith agent, where $m \in \mathcal{O}_i^t$, and let $r_s \in \mathbb{R}_+$ be a safety distance away from an obstacle. The obstacle can be expressed as

$$h\left(p_i^t, p_{m|i}^t\right) = r_s^2 - \|p_i^t - p_{m|i}^t\|^2. \quad (19)$$

The obstacle-avoidance condition is that the trajectory of Agent i along the NMPC's prediction horizon must be completely outside the sphere or circle defined by Equation 19. Thus, the obstacle-avoidance constraint can be defined as

$$h\left(p_i^{t+k+1|t}, p_{m|i}^t\right) \leq 0, \quad k = 0, \dots, T-1. \quad (20)$$
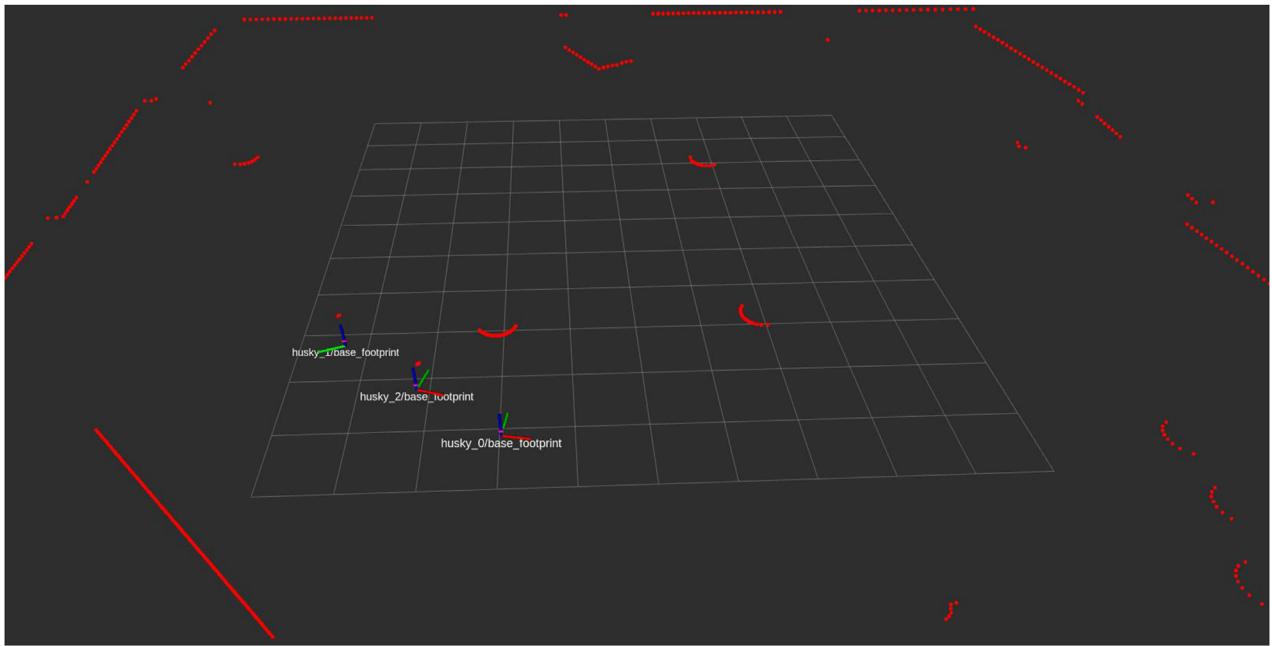
FIGURE 4
The visualization of laser scan point cloud data from UST10 simulated 2D LiDAR and the robots' frames in RViz.

3.3 NMPC problem

From the objective function (Equation 7) and the constraints Equations 4, 5, and 20 described earlier, the optimization problem can be formulated as Equation 21:

$$\underset{u_i^t}{\text{Minimize}} \ \tilde{J}\!\left(u_i^t\right) \qquad (21)$$

subject to Equation 22:

$$
\left.
\begin{array}{l}
x_i^{t+k+1|t} = f_i\!\left(x_i^{t+k|t}, u_i^{t+k|t}\right) \\[4pt]
u_i^{t+k|t} \in \mathcal{U}_i \\[4pt]
x_i^{t+k+1|t} \in \mathcal{X}_i \\[4pt]
h\!\left(p_i^{t+k+1|t}, p_{m|i}^t\right) \le 0, \quad m \in \mathcal{O}_i^t
\end{array}
\right\} \ \text{for } k = 0, \ldots, T-1
$$

$$d_{j|i}^{t+k+1|t} \ge d_{i,\text{sep}}, \quad j \in \overline{\mathcal{N}}_i^{t+k+1|t} \quad \text{for } k = 0, \ldots, T_{\text{sep}} - 1. \qquad (22)$$

This problem will be fed into the OpEn (see Sopasakis et al., 2020) framework to generate Rust code that solves the constrained optimization problem using PANOC (see Stella et al., 2017) with augmented Lagrangian and penalty methods.

4 Simulation

4.1 Simulation setup

In this work, the proposed optimal control problem is demonstrated through a close-to-real-world simulation in Gazebo (see Koenig and Howard, 2004), which incorporates physical quantities and three-dimensional dynamics, such as friction and inertia. This simulation environment is well integrated with
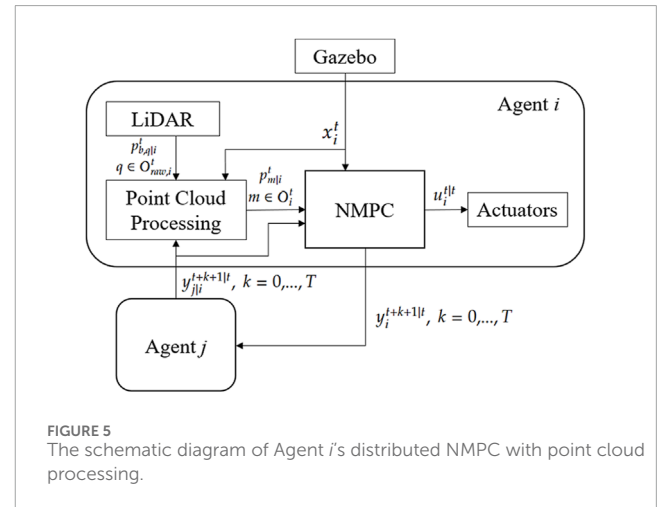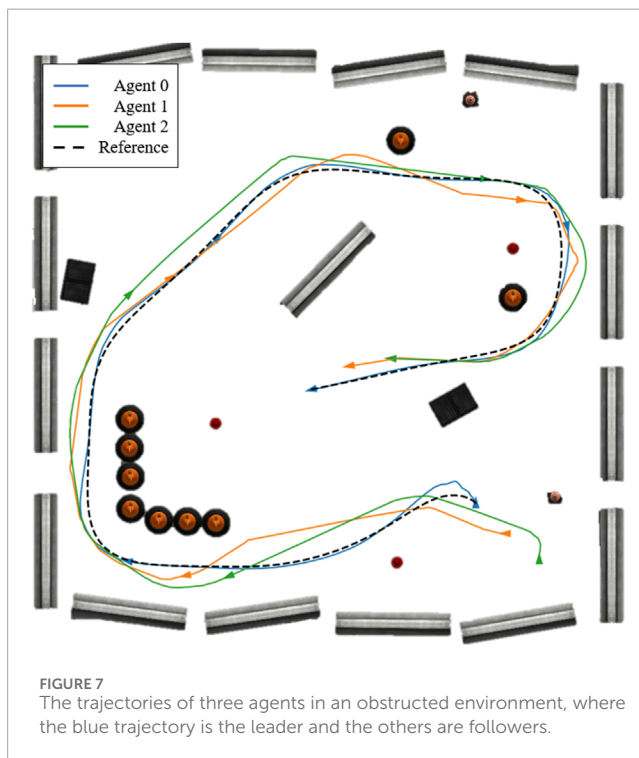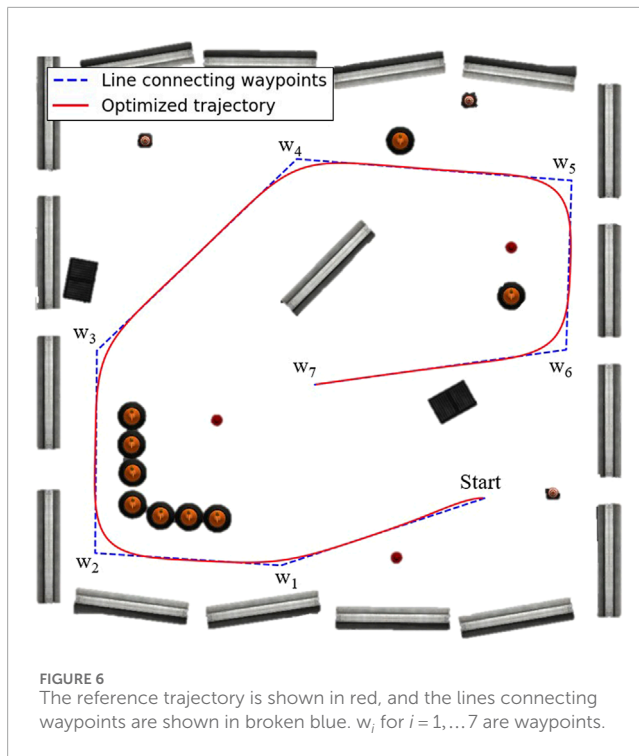

FIGURE 5
The schematic diagram of Agent *i*'s distributed NMPC with point cloud processing.

the ROS. The agent model in our study is the unicycle ground vehicle model, chosen for its representation of most mobile robots and its inherent nonlinear properties as a nonlinear input-affine system.

The continuous-time state equation of the unicycle ground vehicle model is given by Equation 23:

$$
\begin{bmatrix} \dot{p}_{i,x}(t) \\ \dot{p}_{i,y}(t) \\ \dot{\psi}_i(t) \end{bmatrix} =
\begin{bmatrix} \cos \psi_i(t) & 0 \\ \sin \psi_i(t) & 0 \\ 0 & 1 \end{bmatrix}
\begin{bmatrix} v_i(t) \\ \dot{\psi}_i(t) \end{bmatrix}, \qquad (23)
$$

where $v_i(t)$ and $\dot{\psi}_i(t)$ are the system inputs. The discrete-time state update equations can be derived through a forward Euler discretization. Additionally, the extra states, which are translational

FIGURE 6
The reference trajectory is shown in red, and the lines connecting waypoints are shown in broken blue. w_i for $i = 1, \ldots 7$ are waypoints.

FIGURE 7
The trajectories of three agents in an obstructed environment, where the blue trajectory is the leader and the others are followers.

velocities with respect to the global frame, are added so that the alignment with its neighbors will be explicitly determined. Hence, the system's state becomes $x_i^k = \left[p_{i,x}^k, p_{i,y}^k, \psi_i^k, v_{i,x}^k, v_{i,y}^k \right]^\top \in \mathbb{R}^5$, and the system's input is $u_i^k = \left[v_i^k, \dot{\psi}_i^k \right]^\top \in \mathbb{R}^2$. The discrete-time state equation

can be expressed as

$$
\begin{bmatrix}
p_{i,x}^{k+1} \\
p_{i,y}^{k+1} \\
\psi_i^{k+1} \\
v_{i,x}^{k+1} \\
v_{i,y}^{k+1}
\end{bmatrix}
=
\begin{bmatrix}
p_{i,x}^{k} \\
p_{i,y}^{k} \\
\psi_i^{k} \\
0 \\
0
\end{bmatrix}
+
\begin{bmatrix}
\Delta t \cos \psi_i^k & 0 \\
\Delta t \sin \psi_i^k & 0 \\
0 & \Delta t \\
\cos \psi_i^k & 0 \\
\sin \psi_i^k & 0
\end{bmatrix}
\begin{bmatrix}
v_i^k \\
\dot{\psi}_i^k
\end{bmatrix}, \qquad (24)
$$

where Δt is the sampling time in seconds. This discretized state equation will be used in NMPC to update the agent's trajectory along the finite horizon period.

We utilized the Husky UGV from Clearpath Robotics (2015) as agents in the 3D Gazebo simulation. Figure 3 displays the simulated Husky UGVs and environment in our Gazebo 3D simulation. The robots are equipped with a UST10 simulated 2D LiDAR at their body center. The LiDAR has a default range of 5 m, a 360° angular range starting from the robot heading, and rotates counterclockwise, with 720 sampling points per round in the robot's body frame polar coordinate. The output information from the LiDAR contains a time stamp, frames, sensor configurations, and ranges of detected obstacles, packed in the laser scanner message type in the ROS convention, which are two-dimensional point clouds. These raw data will be fed into the point cloud processor to produce the processed point data and then provided to the NMPC solver. The demonstration of the point cloud data is shown through RViz (ROS visualization) in Figure 4. The video of the 3D simulation with visualized point cloud data can be accessed via the following link: https://youtu.be/APg52Rw725M.

Note that the state Equation 24 represents the kinematic model of a unicycle mobile robot, which omits physical characteristics such as the relationship between motor voltage, current, torque, and speed. In other words, this model serves as a simplified representation of the proposed NMPC, where the controller generates high-level commands in the form of translational and angular velocities. In a real-world implementation, these commands would typically be passed to a low-level controller, such as a proportional–integral–derivative (PID) controller, which regulates motor actuation by adjusting electrical inputs accordingly. In this study, the high-level commands are sent to a virtual low-level system that converts the unicycle model's control inputs into individual wheel speeds and torques. In the simulation, the wheels' speeds are sensed and torques are regulated using well-tuned low-level PID controllers. A complete model incorporating other dynamic characteristics could be considered for more accurate state prediction in an NMPC scheme. However, such a model requires system identification techniques and is not within our scope.

In the simulation, we consider the system with three agents, consisting of one leader and two followers. The leader will accept the arbitrary input command and predict its translational position and velocity trajectory within the horizon period T based on the instantaneous inputs. The predicted trajectory will be shared with the followers within the detection range, which is assumed to be the same as the LiDAR default range. The followers also share their predicted trajectories from the NMPC optimizer with one another. The schematic diagram of follower Agent i is shown in Figure 5, where Agent j is the neighbor within Agent i's detection range $\left(j \in \overline{\mathcal{N}}_i^{t+k|t} \right)$.
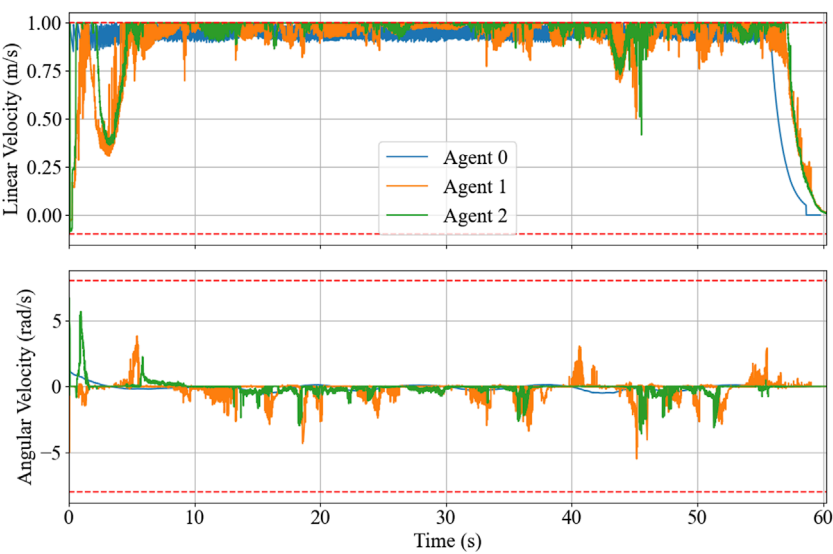
FIGURE 8
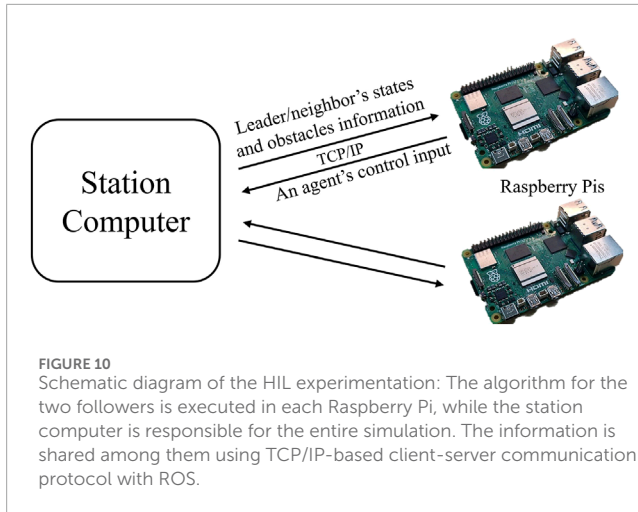Time-series control inputs with constraint boundary shown as broken red lines.
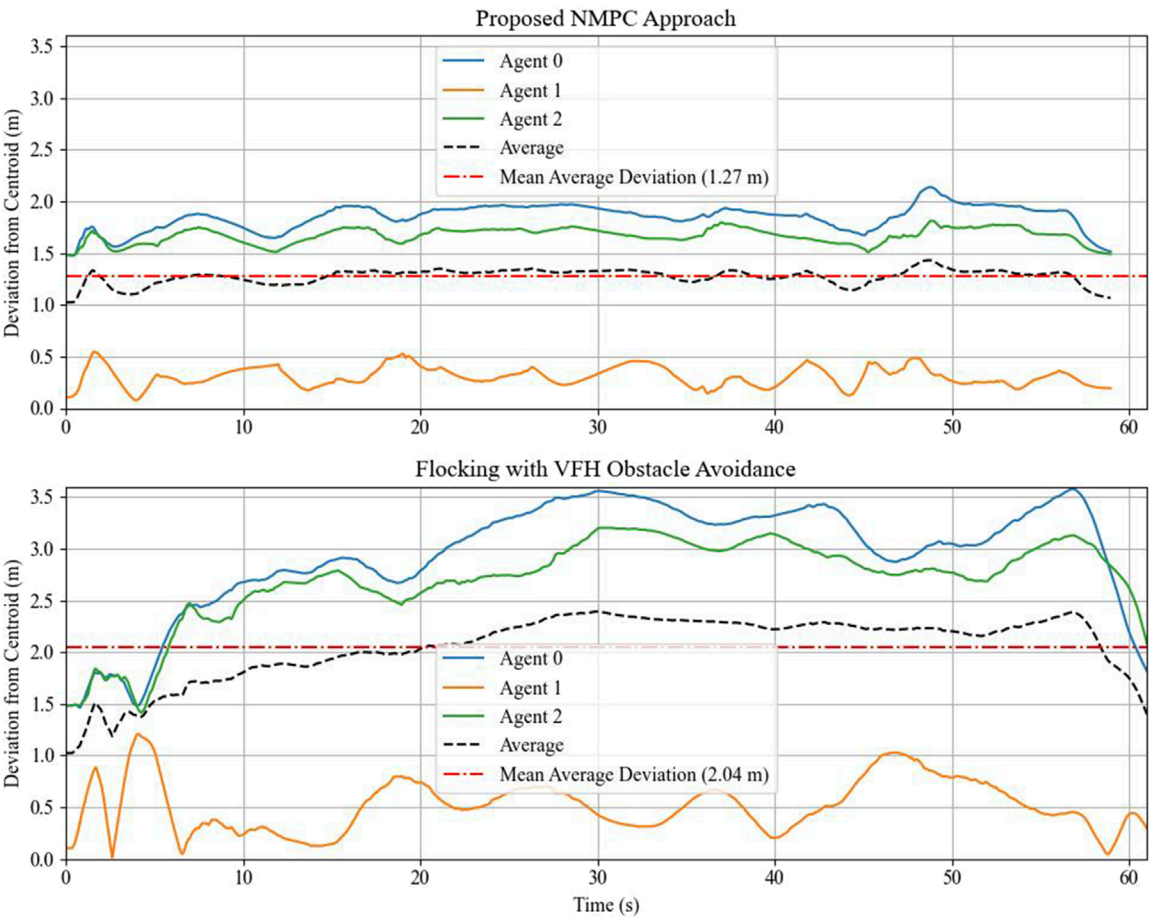
FIGURE 9
The plot of the deviation from the centroid of the two approaches.

Schematic diagram of the HIL experimentation: The algorithm for the two followers is executed in each Raspberry Pi, while the station computer is responsible for the entire simulation. The information is shared among them using TCP/IP-based client-server communication protocol with ROS.

HIL simulated trajectories of three agents in an obstructed environment, where the blue trajectory is the leader's and the rest are followers.

The MPC horizon for all agents is set to $T = 10$ steps. The separation horizons are set as $T_{sep} = 5$ steps, and the prediction of collision avoidance beyond T_{sep} is softly constrained with a penalty parameter of $\rho_{sep} = 20$. The discount factor γ is set to 0.8. For the alignment weights of agents positioned behind, as stated in Equation 10, a value of $\beta_i = 0.5$ is adopted. In the trade-off rule described in Equation 12, a default static coefficient of $q_{i,st} = 0.5$ is utilized alongside a dynamic weight of $c_i = 10$. An upper limit of $\bar{\pi} = 3$ is set to be equal to the total number of agents (including the leader) for the hierarchy levels according to Equation 8. The sampling time for state equation discretization Δt is 0.1 s. Because we employ a two-dimensional laser scanner as a sensor in our simulation, the down-sampling processing is

utilized based on Equation 15, where the down-sampling factor f_s is set to 4.

We employed rectangular sets for the inputs and state constraints. The input boundaries for translational velocity are $[-0.1, 1.0]$ m/s and $[-8, 8]$ rad/s for rotational velocity. The state constraint can be interpreted as a square workspace in a two-dimensional plane originating at $(0,0)$, in which the robots can move regardless of obstacles. We define the spatial limit as $p_x = p_y = \pm 10$ meters. For obstacle-avoidance constraint, the inequality in Equation 20 is reformulated to $\min\{0, h(p_i^{t+k+1|t}, p_{m|i}^t)\} = 0$ for $k = 0, \ldots, T - 1$ and $m \in \mathcal{O}_i^t$. The obstacle-avoidance constraint is implemented through OpEn's penalty method, while the state and separation (for $k < T_{sep}$) constraints are handled using the augmented Lagrangian method. The solver parameters are configured as follows: the solution tolerance ϵ and δ are set to 1×10^{-5} and 1×10^{-4}, respectively. The initial penalty λ_0 is set to 1×10^{-2}. The penalty update coefficient ρ, which is different from the separation soft constraint coefficient, is selected to be 5.

4.2 Navigate in the obstructed environment

We tested the proposed optimal control formulation and algorithm by letting the leader agent track a reference trajectory in an obstructed environment created by Mukherjee (2015) and evaluating the followers' trajectories resulting from the NMPC problem formulated in Section 3.3. The environment contains various obstacles, such as barriers, fire hydrants, dumpsters, and construction cones. The reference trajectory can be obtained by first specifying waypoints. Then, create points between the two immediate waypoints, where the distances between points are constant except for the remaining segments. Suppose that T_p is the total number of points, and $\widehat{P} = \{\widehat{p}^0, \ldots, \widehat{p}^{T_p}\}$ is a set of points from the given process. The reference trajectory for the leader agent $\mathrm{ref}P = \{p^0, \ldots, p^{T_p}\}$ is defined as

$$
\begin{aligned}
^{\mathrm{ref}}P = \arg\min_{p^k, u^k} \sum_{k=0}^{T_p - 1} \left(q_p \|p^k - \widehat{p}^k\|^2 + q_u \|u^k\|^2 \right) + q_T \|p^{T_p} - \widehat{p}^{T_p}\|^2, \\
k = 0, \ldots, T_p,
\end{aligned}
\tag{25}
$$

subjected to the state Equation 24, where $q_p \in \mathbb{R}$, $q_T \in \mathbb{R}$, and $q_u \in \mathbb{R}$ are weighted constants. The optimized trajectory, a so-called admissible path for unicycle mobile robots, is depicted in Figure 6. The leader agent will track this reference trajectory in Equation 25 employing a simple reactive controller, and the followers will preserve the fleet connectivity using the proposed NMPC scheme. Let us say the agent index for the leader is 0. Its control law is defined below.

$$
\begin{aligned}
\nu_0^k &= K_\nu \|\widehat{p}^k - p_0^k\|^2 \\
\dot{\psi}_0^k &= K_\psi \cdot \mathrm{wrap}\left(\widehat{\psi}_0^k - \psi_0^k \right), \quad k = 0, \ldots, T_p,
\end{aligned}
\tag{26}
$$

where $K_\nu \in \mathbb{R}$ and $K_\psi \in \mathbb{R}$ are the proportional gains for each control channel. $\widehat{\psi}^k$ is defined by Equation 27:

$$
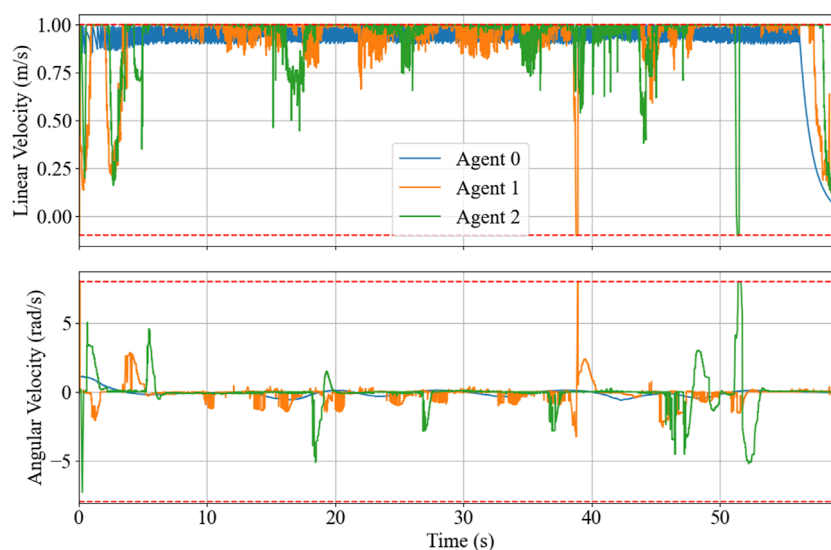\widehat{\psi}_0^k = \arctan\left[\frac{(\widehat{p}^k - p_0^k)_y}{(\widehat{p}^k - p_0^k)_x} \right].
\tag{27}
$$

FIGURE 12
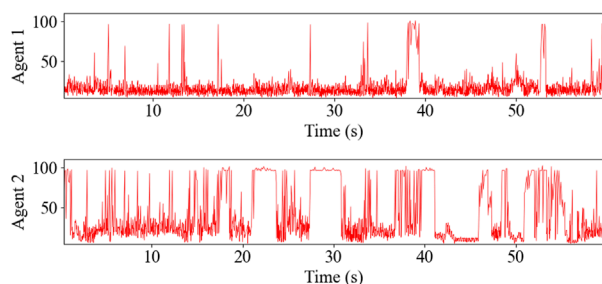HIL simulation's time-series of the control inputs with boundary, depicted in broken red.

FIGURE 13
NMPC's solver solving time on Raspberry Pi 4B: Agent 1 averages 17.19 ms and peaks at 95 ms, while Agent 2 averages 24.6 ms and also peaks at 95 ms.
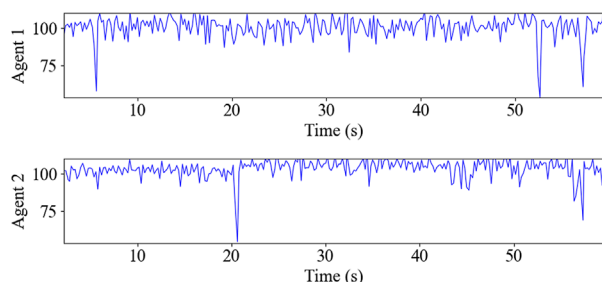

FIGURE 14
The Raspberry Pi's NMPC solver node's CPU usage percentages for Agents 1 and 2 are as follows: peak percentages of 116.2% and 117.3% and average percentages of 98.2% and 97.82%, respectively.

A function wrap:$\mathbb{R} \to \mathbb{R}$ is a function that wraps the angle to be within the range of $[0, \pi)$, defined by Equation 28:

$$\text{wrap}(\theta) = \theta - 2\pi \cdot \left\lfloor \frac{\theta}{2\pi} \right\rfloor. \tag{28}$$

The initial poses of the robots are: Agent 0 at $(5.86, -5.13)$ with 0 radians (the same pose as the initial waypoint in Figure 6), Agent 1 at $(6.85, -6.25)$ with 1.5 radians, and Agent 2 at $(7.87, -7.35)$ with 0 radians, where Agent 0 is the leader, and the others are followers. The resulting trajectories are shown in Figure 7, and the reference trajectory is illustrated as a broken black path. The control inputs throughout the experimentation are displayed in Figure 8 with the control inputs' boundary shown as broken red lines. The simulation video can be accessed through the following link https://youtu.be/u8TcpLy7NKI. In the simulation, the leader is given a control input (Equation 26) to track the optimized path, and the two followers can maintain fleet connectivity and separation while avoiding obstacles using local information. In the demo, each follower recognized up to two neighbors if they were in the sensor's range, regardless of the point cloud processing, which only simplifies the obstacle-avoidance constraint. The algorithm is scalable, but we only showed three robots for visualization purposes.

A comparison experiment is conducted to benchmark the proposed NMPC algorithm. For this purpose, the vector field histogram (VFH) algorithm, tailored for flock navigation, is employed. VFH is a simple reactive obstacle-avoidance strategy well-suited for a two-dimensional LiDAR sensor, where obstacle information is stored in polar coordinate form. The algorithm converts obstacle data into a one-dimensional histogram and selects a sector that is both obstacle-free and closest to the target, allowing the robot to navigate accordingly. See Liang et al. (2023) for more details on this algorithm. The simulation is performed in a similar manner, where the leader agent tracks the reference path (Equation 25), and the followers execute the

tailored VFH algorithm to maintain fleet connectivity. The fleet connectivity is quantified by the deviation from the average position of all agents, that is, the deviation from the centroid. The plot comparing the deviation from the centroid for both the proposed method and the VFH algorithm is shown in Figure 9. The results indicate that the proposed algorithm navigates the robot fleet through an obstructed environment with better connectivity preservation.

4.3 Hardware-in-the-loop simulation

In the previous subsection, the optimal control problem was solved successfully in a distributed manner on a station computer. However, the embedded processor might not have enough computational power to solve such a problem smoothly. Thus, in this section, we examine the feasibility of implementing the proposed NMPC on Raspberry Pi by connecting it to a station computer and performing a hardware-in-the-loop (HIL) simulation. In this setting, we use two Raspberry Pis to represent the two followers, Agents 1 and 2. The board model we use is Raspberry Pi 4 B, with a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor (ARM architecture), 1500 MHz clock speed, and 4 GB of RAM. The HIL simulation is conducted through a master–slave communication scheme, where the station computer is assigned to be a master in an ROS fashion. We connect the two Raspberry Pis to the same network as the simulation computer and create the communication bridge. Then, ROS topics published by either the master or the slaves can be subscribed to by every machine within this network. The NMPC solvers operate independently on each target board while exchanging information. The time required for the optimizer to attain the solution is crucial. The longer it takes, the more the performance degrades. To ensure that the solution for each optimizer is available within the discretization sampling period, set at 100 ms, we set the solve time cut-off to 95 ms, meaning that the solver will stop the operation once the solving time reaches the specified cut-off, and the lowest cost value in the last iteration will be the result. The leader is given a reference trajectory to track, akin to the simulation in the preceding section, utilizing the robots' initial poses from the previous section. The schematic diagram of the experimentation is depicted in Figure 10.

The resulting trajectories of the HIL simulation are shown in Figure 11, and the time-series data of control inputs are depicted in Figure 12. Figure 13 displays the solving time in milliseconds (ms). The average and peak solving times are 17.19 ms and 95 ms for Agent 1 and 24.6 ms and 95 ms for Agent 2, respectively. The results show that the solver occasionally reached the cut-off time limit, particularly for Agent 2. However, this does not indicate a failure in controlling the robots. Rather, it means that the optimizer did not fully converge to the optimal solution within the allotted time. Nevertheless, the suboptimal solutions returned at the cut-off were sufficiently effective, as demonstrated in Figure 11. Figure 14 shows the CPU usage percentage for the NMPC solver node in each Raspberry Pi. Because it is a quad-core processor, the percentage can be up to 400%. For Agents 1 and 2, the peak percentages are 116.2% and 117.3%, respectively,

while the average percentages are 98.2% and 97.82%, respectively. This means that the NMPC solver utilized approximately one core on average.

5 Conclusion

In this study, we introduced a local obstacle-avoidance strategy tailored to our distributed NMPC-based flock navigation with modified flocking rules. We proposed a point cloud processing technique, including directional filtering and down-sampling, that significantly reduces the computational burden. This technique is applicable to both two-dimensional and three-dimensional point cloud data commonly utilized in modern robotic sensors to store obstacle information, such as LiDAR and depth cameras. Then, we introduced a local obstacle-avoidance constraint and integrated it into our framework. The real-time solving of the NMPC problem was achieved using PANOC via the OpEn code generator. Simulation results demonstrated the successful navigation of the robot fleet through unknown obstructed environments. Furthermore, an HIL simulation using Raspberry Pi 4 was conducted to assess feasibility. The solving time for the NMPC optimizer was analyzed, revealing peak values reaching the maximum solving time for a few samples, with an average of less than 25 m. We acknowledge that if the sampling time gets smaller, the chance of hitting the solve time cut-off would increase, which could lead to performance degradation. Thus, further studies can be considered. For example, instead of treating each point in the point cloud separately, we can group nearby points into a single object. This would reduce the number of obstacle constraints from approximately 100 to less than 10, which would significantly accelerate the solver and improve real-time performance. On average, the optimizer occupied one core during the HIL simulation. The trajectories in the HIL simulation showed that the robot fleet could navigate through an unknown obstructed environment successfully. However, if the number of detected agents increases, the solving time will also increase, leading to a situation where the optimizer cannot reach the cost value tolerance and therefore generates undesirable trajectories. Therefore, future research might also explore algorithms that limit the number of agents and prioritize them accordingly.

Data availability statement

Publicly available datasets were analyzed in this study. These data can be found here: https://github.com/husky/husky/blob/noetic-devel/husky_gazebo/worlds/clearpath_playpen.world.

Author contributions

NG: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Writing – original draft, and Writing – review and editing. KY: Conceptualization, Formal analysis, Funding acquisition, Project administration, Resources, Supervision, Validation, Writing – original draft, and Writing – review and editing.

Funding

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Generative AI was used in the creation of this manuscript.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). CasADi: a software framework for nonlinear optimization and optimal control. *Math. Program. Comput.* 11, 1–36. doi:10.1007/s12532-018-0139-4

Barraquand, J., Langlois, B., and Latombe, J.-C. (1991). Numerical potential field techniques for robot path planning. *Fifth Int. Conf. Adv. Robotics 'Robots Unstructured Environ.* 2, 1012–1017 vol.2. doi:10.1109/icar.1991.240539

Brayanov, N., and Stoynova, A. (2019). Review of hardware-in-the-loop – a hundred years progress in the pseudo-real testing. *Electrotech. and Electron.* 54, 70–84.

Cao, H., Chen, J., Mao, Y., Fang, H., and Liu, H. (2010). "Formation control based on flocking algorithm in multi-agent system," in *2010 8th world congress on intelligent control and automation*, 2289–2294.

Clearpath Robotics, (2015). Husky UGV tutorials 1.0.0. Available online at: https://clearpathrobotics.com.

Goarin, M., Li, G., Saviolo, A., and Loianno, G. (2024). "Decentralized nonlinear model predictive control for safe collision avoidance in quadrotor teams with limited detection range," in *IEEE international conference on robotics and automation (ICRA) 2025. arXiv preprint arXiv:2409.17379.*

Koenig, N., and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ Int. Conf. Intelligent Robots Syst. (IROS) (IEEE Cat. No.04CH37566)* Vol. 3, 2149–2154. doi:10.1109/IROS.2004.1389727

Kong, F., Chen, H., Li, H., Yan, J., Wang, X., and Fang, J. (2023). "Flocking with obstacle avoidance for fixed-wing unmanned aerial vehicles via nonlinear model predictive control," in *2023 42nd Chinese control conference (CCC)*, 5957–5962. doi:10.23919/CCC58697.2023.10240689

Li, C., Yang, Y., Jiang, G., and Chen, X. (2024). A flocking control algorithm of multi-agent systems based on cohesion of the potential function. *Complex and Intelligent Syst.* 10, 2585–2604. doi:10.1007/s40747-023-01282-2

Liang, Q., Wang, Z., Yin, Y., Xiong, W., Zhang, J., and Yang, Z. (2023). Autonomous aerial obstacle avoidance using lidar sensor fusion. *Plos one* 18, e0287177. doi:10.1371/journal.pone.0287177

Lindqvist, B., Mansouri, S. S., Kanellakis, C., and Nikolakopoulos, G. (2020). "Collision free path planning based on local 2d point-clouds for mav navigation," in *2020 28th mediterranean conference on control and automation (MED)*, 538–543.

Lindqvist, B., Sopasakis, P., and Nikolakopoulos, G. (2021). "A scalable distributed collision avoidance scheme for multi-agent UAV systems," in *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 9212–9218.

Mestres, P., Nieto-Granda, C., and Cortés, J. (2024). Distributed safe navigation of multi-agent systems using control barrier function-based controllers. *IEEE Robotics Automation Lett.* 9, 6760–6767. doi:10.1109/LRA.2024.3414268

Mihalič, F., Truntič, M., and Hren, A. (2022). Hardware-in-the-loop simulations: a historical overview of engineering challenges. *Electronics* 11, 2462. doi:10.3390/electronics11152462

Mukherjee, P. (2015). "clearpath_playpen.world," in (Kitchener, ON, Canada: Clearpath Robotics Inc). Available online at: https://github.com/husky/husky/blob/noetic-devel/husky_gazebo/worlds/clearpath_playpen.world.

Nag, A., Huang, S., Themelis, A., and Yamamoto, K. (2022). "Flock navigation with dynamic hierarchy and subjective weights using nonlinear MPC," in *2022 IEEE conference on control technology and applications (CCTA)*, 1135–1140.

Nag, A., and Yamamoto, K. (2024). Distributed control for flock navigation using nonlinear model predictive control. *Adv. Robot.* 38, 619–631. doi:10.1080/01691864.2023.2299859

Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Trans. Automatic Control* 51, 401–420. doi:10.1109/TAC.2005.864190

Reynolds, C. W. (1987). Flocks, herds and schools: a distributed behavioral model. *SIGGRAPH Comput. Graph.* 21, 25–34. doi:10.1145/37402.37406

Sathya, A., Sopasakis, P., Van Parys, R., Themelis, A., Pipeleers, G., and Patrinos, P. (2018). "Embedded nonlinear model predictive control for obstacle avoidance using PANOC," in *2018 European control conference (ECC)*, 1523–1528. doi:10.23919/ECC.2018.8550253

Shi, L., and Luo, J. (2024). A framework of point cloud simplification based on voxel grid and its applications. *IEEE Sensors J.* 24, 6349–6357. doi:10.1109/jsen.2023.3320671

Song, P., and Kumar, V. (2002). A potential field based approach to multi-robot manipulation. *Proc. 2002 IEEE Int. Conf. Robotics Automation (Cat. No.02CH37292)* 2, 1217–1222. doi:10.1109/ROBOT.2002.1014709

Sopasakis, P., Fresk, E., and Patrinos, P. (2020). OpEn: code generation for embedded nonconvex optimization. *IFAC-PapersOnLine* 53, 6548–6554. doi:10.1016/j.ifacol.2020.12.071

Stella, L., Themelis, A., Sopasakis, P., and Patrinos, P. (2017). "A simple and efficient algorithm for nonlinear model predictive control," in *IEEE conference on decision and control (CDC)*, 1939–1944.

Tanner, H., Jadbabaie, A., and Pappas, G. (2003a). Stable flocking of mobile agents, part I: fixed topology. *42nd IEEE Int. Conf. Decis. Control (IEEE Cat. No.03CH37475)* 2, 2010–2015. doi:10.1109/CDC.2003.1272910

Tanner, H., Jadbabaie, A., and Pappas, G. (2003b). Stable flocking of mobile agents part II: dynamic topology. *42nd IEEE Int. Conf. Decis. Control (IEEE Cat. No.03CH37475)* 2, 2016–2021. doi:10.1109/CDC.2003.1272911

Xu, T., Liu, J., Zhang, Z., Chen, G., Cui, D., and Li, H. (2023). Distributed mpc for trajectory tracking and formation control of multi-uavs with leader-follower structure. *IEEE Access* 11, 128762–128773. doi:10.1109/ACCESS.2023.3329232

Xu, Y., Tong, X., and Stilla, U. (2021). Voxel-based representation of 3d point clouds: methods, applications, and its potential use in the construction industry. *Automation Constr.*, 126. doi:10.1016/j.autcon.2021.103675

Yu, S., Hirche, M., Huang, Y., Chen, H., and Allgöwer, F. (2021). Model predictive control for autonomous ground vehicles: a review. *Aut. Intell. Syst.* 1, 4–17. doi:10.1007/s43684-021-00005-z