Variable Graph Control: ROS2-Based Implementations of Cooperative Distributed Algorithms

Ishan Agrawal

May 5, 2025

Thesis Advisor:
Dr. Dan Guralnik

Abstract

This thesis presents Variable Graph Control (VGC), a generalized ROS2-based framework for implementing distributed multi-agent control algorithms with communication graph maintenance guarantees. Building upon the Plug-and-Play (PnP) controller paradigm [1], VGC enables empirical investigation of theoretical bounds on control parameters in complex navigation scenarios. The framework employs navigation fields to generate distributed control laws that preserve inter-agent communication links while navigating through environments with obstacles.VGC facilitates methodical testing across different environment types, including sphere worlds and non-convex domains.

The implementation leverages ROS2's Data Distribution Service capabilities to support genuinely distributed computation without centralized coordination. The modular architecture decouples environment representation, navigation field computation, and control law implementation, providing an extensible platform for comparative evaluation of multi-agent coordination strategies. This work contributes to bridging the divide between theoretical models and practical robotic implementations, offering insights into more efficient parameter selection for real-world deployment of distributed multi-agent systems.

Acknowledgements

I could not have asked for a better undergraduate advisor than Dr. Dan Guralnik. Within weeks of my first semester, you engaged deeply with a clueless, overly enthusiastic eighteen-year-old — amid a global pandemic — while I was still living at home in India. That early mentorship radically changed the trajectory of my life.

I am also deeply grateful to Dr. Warren Dixon for providing me with countless opportunities to grow as an engineer and researcher along with funding me through this journey. Your guidance pushed me to take on ambitious challenges and pursue work I once thought beyond my ability.

Over the past four years, I have been incredibly fortunate to be part of the Nonlinear Controls and Robotics Lab. Working alongside brilliant colleagues and close friends has been an unforgettable experience. Thank you for your support, your ideas, and your friendship.

Most importantly, I would like to express my deepest appreciation to my parents and my brother. Your patience, sacrifices, and unwavering belief in me made this thesis — and this entire journey — possible.

Contents

CHAPTER 1

Introduction

Multi-agent systems (MAS) provide a robust framework enabling cooperative execution of complex tasks through decentralized coordination among autonomous agents. Each agent within a MAS is capable of sensing its environment, making localized decisions, and executing control actions to achieve collective objectives [1]. The application domains for MAS technologies span spacecraft formation flying [3, 4], autonomous vehicle platooning [5, 6], distributed sensing [7, 8], and cooperative robotics [9, 10].

A critical architectural decision in MAS design involves choosing between centralized and distributed control approaches. Centralized schemes utilize a master node aggregating global information to compute system-wide control actions. Conversely, distributed architectures decentralize decision-making authority, allowing agents to independently calculate their control actions based on locally available information communicated by neighbors [11]. Distributed architectures present significant advantages, notably eliminating single points of failure, enhancing scalability, and reducing computational burdens through parallelism. However, these benefits introduce unique challenges for coordination, especially in scenarios where agents communicate over distance-limited networks represented as dynamic graphs [1].

1. Background and Motivation

Robust navigation remains a fundamental task for MAS operating in complex environments. Classical navigation approaches, such as the Rimon-Koditschek Navigation Functions (RKNFs), provide guaranteed obstacle avoidance and goal convergence within known environments [12, 13]. These approaches have evolved along several key dimensions. For time-critical applications, Loizou's Navigation Transformation [14] introduces methodologies for precise temporal scheduling, enabling robots to reach destinations at predetermined times without requiring integration over flow lines. Extensions to broader domains include Filippidis and Kyriakopoulos's work [15] on navigation in "everywhere partially sufficiently curved worlds" and Paternain et al.'s investigations [16] of navigation functions for convex potentials. Reactive navigation strategies have further extended these guarantees to unknown or partially known settings. Arslan and Koditschek [17] developed sensor-based navigation methods suitable for convex (sphere world) environments, while separately, earlier work introduced coordinated navigation approaches [18]. A significant advancement came through Vasilopoulos and Koditschek's work [2], which introduced a framework for navigation in environments with non-convex obstacles. Their key insight involves using diffeomorphisms (smooth deformations with smooth inverses) to transform non-convex obstacles into convex ones, where existing reactive controllers can be applied, then pulling this solution back to the physical space to generate actual robot commands. This approach along with nonholonomic navigation methods [19, 20] extends to frameworks like motivational dynamics [21], which enables composing a variety of navigational tasks such as alternating between targets, following patrol routes, or reaching opportunistic targets.

Translating these single-agent navigation strategies to distributed multi-agent scenarios represents an emerging frontier. MAS must solve coordination tasks such as rendezvous, formation control, and consensus using only locally accessible information. Practical MAS implementations must contend with communication constraints, typically modeled by agents exchanging information within a fixed communication radius R. Consequently, preserving the underlying communication graph dynamically during operation is crucial.

This challenge motivates the graph maintenance problem: designing distributed control laws that preserve communication links (graph edges) while enabling safe navigation through obstacle-rich environments [1].

Figure 1 conceptually illustrates a distributed MAS graph, where nodes represent agents, solid lines indicate edges (communication links), and dashed circles show the agents' limited communication radii. Agents maintain graph connectivity by staying within communication range, highlighting the intrinsic coupling between local navigation tasks and global communication constraints.

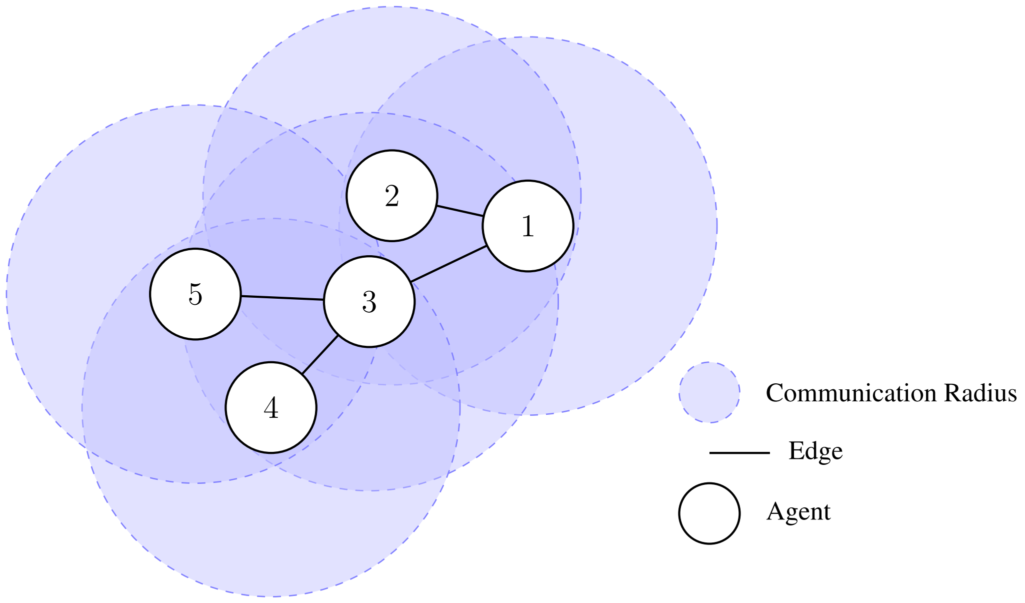

FIGURE 1. Visualization of a distributed MAS graph highlighting agent nodes, inter-agent edges, and communication radii. Maintaining the connectivity of this graph is crucial for coordination. Here i denotes the agents.

This thesis introduces a generalizable framework called **Variable Graph Control (VGC)**, leveraging navigation fields to achieve distributed MAS coordination. In this thesis, VGC employs, but is not limited to, the recently developed "plug-and-play" (PnP) controller paradigm [1], where a closed-form MAS controller is constructed from "black-box" single-agent navigation fields.

The PnP controller provides mathematical guarantees for connectivity maintenance during navigation through complex environments, contingent on the navigation field satisfying certain properties. In particular, the field must satisfy the (R, δ)-goodness condition. However, in practice, the theoretical constraints prove to be significantly more conservative than required for successful operation.

The (R, δ)-goodness constraint, while theoretically sound, presents a significant practical challenge. This constraint requires that for any two points within distance R of each other, the navigation field must make an angle with the direct line between them whose cosine is at least δ [1]. The critical challenge lies not in satisfying this condition for some arbitrary $\delta \in (0, 1)$, but rather in achieving it with δ sufficiently close to 1 such that the necessary inequalities for graph maintenance guarantees are satisfied. As noted in the appendix of [1], enforcing this constraint with the theoretically required high values of δ is hard to achieve in a generic workspace except through reducing the communication radius.

The authors acknowledge that the theoretical bounds are significantly more conservative than what works in practice, with simulation results showing successful performance using parameter values several orders of magnitude less stringent than those required by theory [1]. This gap between theory and practice presents a compelling research opportunity to investigate —empirically as well as theoritically—how these bounds might be refined while maintaining performance guarantees.

2. Software Development

Implementing theoretical control methods in the real world Translating theoretical control methodologies to real-world implementations in a machine-agnostic manner is a critical challenge in MAS research.

The VGC framework is designed to be a modular ROS 2-based software environment aimed explicitly at experimental validation and visualization of graph-maintaining MAS controllers, focusing on the need for robust implementation.

The VGC framework leverages the Data Distribution Service (DDS) model inherent in ROS 2, enabling fully decentralized message-passing between agents, eliminating the reliance on a centralized communication master node [22, 11]. Each agent independently manages state broadcasting, neighbor subscriptions, control computation, and local actuation, aligning closely with the principles of distributed control.

Key capabilities include dynamically adjusting the inter-agent communication graph during operation via `addEdge` and `rmEdge` services. These features envision future runtime experimentation with adaptive graph topologies, facilitating an examination of tradeoffs between connectivity preservation, task execution efficiency, and required control effort. This modular structure decouples control logic from communication constraints, enhancing system flexibility and analytical clarity.

By studying system behavior across various parameter settings, the testbed may uncover patterns leading to refined theoretical bounds or practical guidelines for real-world implementation. Overall, the VGC testbed facilitates future validation of the theoretical contributions of [1] but also establishes a solid foundation for continued exploration of distributed control under uncertainty and dynamic network configurations, including under other distributed control laws, and in a manner independent of low-level controllers deployed by individual agents.

The complete implementation of the VGC framework described in this thesis is publicly available as open-source software in the following GitHub repository: https://github.com/kotmasha/variable_graph_MAS. This repository contains all source code, configuration files, and documentation necessary to reproduce the experiments and extend the framework for further research.

CHAPTER 2

Single Agent Navigation

1. Environment and Agent Models

DEFINITION 1 (Workspace). The workspace $\Omega \subset \mathbb{R}^d$, $d \geq 2$ is a compact domain with piecewise-smooth boundary, $\partial\Omega$, having finitely many components.

1.1. Agent Dynamics. The focus is initially on holonomic (fully actuated) agents with first-order dynamics. This choice is motivated by the significant body of navigation field methods originally developed for holonomic systems [**12**, **17**, **21**, **16**, **23**], while acknowledging that these methods have been extended to non-holonomic systems as well [**19**, **20**, **2**]. The holonomic model provides a simplified foundation upon which more realistic (e.g., second-order) navigation frameworks can be built, see Figure 3. For a holonomic agent with first-order dynamics, the equations of motion are:

$$\dot{z} = u \tag{1}$$

where $z \in \Omega$ represents the agent's position and $u \in \mathbb{R}^d$ is the control input.

1.2. Environment Model. In practice, navigation needs to account for the presence of obstacles in the workspace: objects the robot should not collide with.

DEFINITION 2 (Obstacles). The connected components of $\mathbb{R}^d \setminus \Omega$ are referred to as obstacles in the workspace. The collection $\mathcal{O}$ of all obstacles is finite, allowing one to write $\mathcal{O} = \{O_i\}_{i=1}^{b}$.

1.2.1. *Extended Obstacles.* The physical agent occupies space and cannot overlap with obstacles. Unless the agent has complex articulated appendages and effectors, it is reasonable to represent it with a circular disk of radius $r > 0$ circumscribed about its center of mass, see Figure 1. In order to benefit from a presentation of the agent as a point particle, it is common practice to inflate the obstacles rather than the agent. To account for the agent's physical dimensions for the purpose of obstacle avoidance, each obstacle O_i is

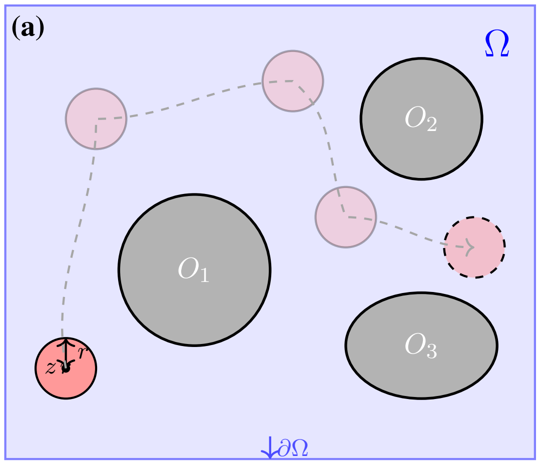

FIGURE 1. Disk-shaped agent of radius $r > 0$ in a planar workspace Ω with three obstacles, as per Definitions 1 and 2.

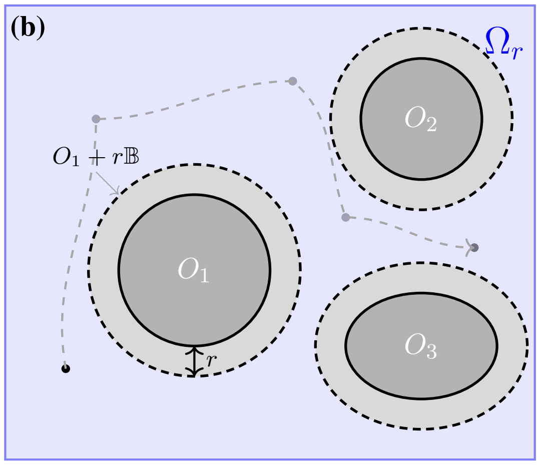

FIGURE 2. Point robot in inset workspace Ω_r with obstacles expanded by radius r.

replaced with a thickened version $O_i + r\mathbb{B}$ (where $\mathbb{B}$ is the unit ball), and the workspace Ω with an inset version $\Omega_r \triangleq \mathbb{R}^2 \setminus \bigcup_{i=1}^{b}(O_i + r\mathbb{B})$. This equivalent formulation simplifies the collision avoidance problem while preserving all the necessary safety constraints (compare Figure 1 with Figure 2).

1.3. Sphere Worlds. Sphere worlds were the first general class of navigation environments for which navigation functions were produced in closed form.

DEFINITION 3 (Sphere World [12]). A sphere world is a workspace obtained by subtracting a finite collection of disjoint open balls $\{c_i + r_i\mathbb{B}\}_{i=1}^{b}$ from a larger closed ball $c_0 + r_0\mathbb{B}$ containing them:

$$\Omega = (c_0 + r_0\mathbb{B}) \setminus \bigcup_{i=1}^{b}(c_i + r_i\mathbb{B}), \tag{2}$$

where $c_i \in \mathbb{R}^d$, $r_i > 0$ are selected accordingly.

The sphere world abstraction is valuable for its elegant closed-form navigation functions [12] and serves as a model space onto which more complex environments can be mapped through diffeomorphisms [2].

2. The Navigation Problem in Ω

Given a workspace Ω, obstacles O_i, and a target position $x^* \in \operatorname{int}(\Omega)$, the navigation problem is to find a Lipschitz continuous controller $u : \Omega \to \mathbb{R}^d$ that

(1) keeps the robot within the workspace Ω (safety/obstacle avoidance);
(2) drives the robot to the target position x^* (convergence).

Due to topological constraints, conditions (1) and (2) cannot simultaneously be globally guaranteed in non-contractible workspaces with continuous controllers. In other words, noncontractible domains necessarily force the existence of additional equilibrium points. This topological obstruction is a classical result in the robot-navigation literature; see [12] for a rigorous treatment using the Poincaré–Hopf index theorem and related arguments.

2.1. Path Planning vs. Reactive Navigation. To address the navigation problem, two primary methodological approaches have emerged: path planning and reactive navigation. These approaches differ fundamentally in how they generate control actions and respond to environmental changes.

2.1.1. *Path Planning.* A path planner $\mathcal{P} : \Omega \times \Omega \to C([0,1], \Omega)$ receives the initial position $z_0 \in \Omega$ and target position $z^* \in \Omega$ and produces a continuous parameterized path

$\sigma : [0,1] \to \Omega$, such that

$$\sigma(0) = z_0, \quad \sigma(1) = z^*, \quad \text{and} \quad \sigma(s) \in \Omega \text{ for all } s \in [0,1]. \tag{3}$$

It is important to note that a path planner is **not** a state feedback controller. Instead, the produced path is inherently time-dependent and precomputed—meaning that time is an explicit parameter within the plan itself. This results in a non-autonomous differential equation when implementing a controller to track the path, contrasting with the autonomous system arising from state feedback controllers [24]. The complexity of physical interactions between robots and their environments leads to accumulation of uncertainty in the system state over time. When a robot attempts to track a precomputed path, unexpected disturbances and modeling errors may force repeated replanning as state estimates deteriorate. This is particularly problematic for mobile robotic platforms with limited energy reserves and computational resources.

Limitations:

(1) Cannot easily adapt to dynamic or uncertain environments;
(2) Does not allow for computations using only local state information;
(3) Sensitive to modeling errors.

2.1.2. *Reactive Planning.* In contrast, reactive navigation produces control inputs based solely on the current state, broadly interpreted to include both environmental measurements and internal state variables. In a reactive approach (e.g., using navigation functions as in [12]), the control law is generated in real time by continuously evaluating the current position, without explicitly planning a time-dependent path beforehand.

Rimon and Koditschek proposed replacing path-following controllers with a new paradigm based on artificial potentials, also known as navigation functions [12]. By leveraging suitable Morse functions on the workspace, the robot performs gradient descent on a function $\varphi : \Omega \to [0,1]$ having the target as its unique minimum, with the workspace boundary represented as the level set $\varphi^{-1}(1)$, which guarentees obstacle-avoidance.

This approach is particularly valuable in settings where the workspace is noncontractible—implying that there exists no state-feedback law that globally stabilizes all trajectories. The reactive approach guarantees convergence to the target from almost all initial conditions (except a set of measure zero, typically corresponding to saddle points or other nongeneric configurations that are structurally unstable under perturbation [12]) requiring only continual real-time computation of a single vector field rather than frequently recalculating complete path plans [1]. The practical advantages of reactive navigation include:

(1) Resilience to uncertainty and disturbances;
(2) Reduced computational requirements, critical for energy-constrained mobile platforms;
(3) Real-time adaptability to changing environments;
(4) Capability to operate with only locally available sensory information

2.2. Navigation Functions. The reactive approach directly generates control inputs based on the current state, without explicit path planning as shown in [12]. A primary example of reactive navigation is generated by the following notion.

DEFINITION 4 (Navigation Function). A C^2 smooth function $\varphi : \Omega \to [0,1]$ is a navigation function on workspace Ω with goal $x^* \in \operatorname{int}(\Omega)$ if:

(i) $\varphi^{-1}(0) = \{x^*\}$ and $\varphi^{-1}(1) = \partial\Omega$ — this ensures a single minimum at the goal and repulsion at workspace boundaries, under the dynamics (4) below;
(ii) Critical points of φ are non-degenerate — this guarantees that all equilibria of the gradient system are isolated, facilitating convergence analysis;
(iii) x^* is the unique global attractor in $\operatorname{int}(\Omega)$ — this property ensures almost global asymptotic stability of the target position.

Using a navigation function, the robot control law is defined as:

$$u(x) = -\nabla\varphi(x) \tag{4}$$

The Rimon-Koditschek navigation function (RKNF) construction takes the form:

$$\varphi = \left(\frac{\gamma^k}{\gamma^k + \prod_{i=1}^{b} \beta_i}\right)^{1/k} \tag{5}$$

where:

- $\gamma : \Omega \to [0,\infty)$ is a cost function with $\gamma^{-1}(0) = \{x^*\}$;
- $\beta_i : \Omega \to [0,\infty)$ are barrier functions with $\beta_i^{-1}(0) = \partial O_i$ defining obstacles;
- $k > 0$ is a tuning parameter that must be sufficiently large.

The RKNF works by combining attractive forces toward the goal with repulsive forces from obstacles. The parameter k acts as a "sharpening" factor that, when sufficiently large, eliminates spurious local minima and ensures that only necessary critical points (dictated by topological constraints) remain. For sphere worlds, it can be mathematically proven that for large enough k, the resulting dynamical system $\dot{x} = -\nabla\varphi(x)$ has only the desired stable equilibrium at x^* and unstable equilibria at saddle points determined by the workspace topology. The construction creates a "steeper" gradient near obstacle boundaries while maintaining a smooth attractive basin toward the goal, effectively resolving the navigation problem with almost global convergence guarantees.

For sphere worlds, the cost function is typically chosen as $\gamma(x) = \|x - x^*\|^2$, measuring the squared Euclidean distance to the target. The barrier functions β_i are designed as $\beta_i(x) = \|x - c_i\|^2 - r_i^2$, representing the squared distance from point x to the boundary of obstacle O_i. These choices ensure that β_i vanishes precisely at the boundary of obstacle O_i while remaining positive elsewhere in the workspace.

2.3. Extensions of RKNFs. The classical Rimon–Koditschek Navigation Functions (RKNFs) are originally formulated for sphere worlds. However, many practical workspaces do not satisfy this restrictive geometric structure. Fortunately, extensions have been developed that allow RKNFs to be applied in more general settings. In what follows we briefly describe some of these extensions, which in turn motivate the design of navigation fields in the next section.

- **Star-Shaped Obstacles:** In many real-world applications, obstacles are not perfect spheres but are instead star-shaped. It is often possible to construct a diffeomorphism
$$h : \Omega \to \Omega_{sph},$$
which maps the original workspace Ω, containing star-shaped obstacles, onto a sphere world Ω_{sph} where standard RKNFs are valid. If φ is a navigation function on a sphere world Ω_{sph}—say, and RKNF—and $h : \Omega \to \Omega_{sph}$ is a diffeomorphism, then $\varphi \circ h$ is a navigation function on Ω, enabling navigation of a fully actuated robot to the target by setting $u = -\nabla(\varphi \circ h)(x)$.

  The navigation field is then recovered in the original coordinates by
$$\mathfrak{n}(y, z) = Dh(z)^{-1} \cdot \mathfrak{n}_{sph}(h(y), h(z)). \tag{6}$$

This approach, following [13] extends the applicability of RKNFs to a broader class of obstacle geometries. An example construction is included in the appendix.

- **Topological Sphere Worlds:** Even when the workspace does not have the exact geometric properties of a sphere world, it is often still topologically equivalent to one. In such *topological sphere worlds*, a continuous diffeomorphism exists that transforms the workspace into a sphere world. This transformation preserves the key properties needed for convergence, allowing the design of navigation functions (and hence navigation fields) with guarantees analogous to those in the classical setting.
- **Complex 3D Domains:** Extensions of the RKNF framework have also been explored for three-dimensional environments. For instance, when dealing with knot complements in $\mathbb{R}^3$, one imposes curvature conditions on the workspace boundary. A typical condition is that the Gaussian curvature satisfies

  $$\kappa(\partial\Omega) < K_{\text{crit}}, \tag{7}$$

  where K_{crit} is a critical threshold. Such conditions help ensure the reactive navigation approach remains feasible even in highly complex or topologically intricate 3D domains [15].

These extensions highlight that—even though the traditional RKNF is tailored to simple sphere worlds—the underlying principles can be adapted to much more general environments via appropriate transformations. However, limitations in constructing exact diffeomorphisms for arbitrary environments led researchers like Arslan and Vasilopoulos to develop more reactive and locally-defined methods. These approaches move away from global potential fields toward sensor-based, real-time reactive strategies that can handle partial knowledge of the environment and do not require explicit construction of a diffeomorphism. This shift emphasizes practical implementability in real-world robotic systems where complete geometric information may not be available.

3. Navigation Fields

Navigation fields generalize the gradient-based dynamics generated by navigation functions, providing locally Lipschitz continuous, vector-valued controllers:

DEFINITION 5 (Navigation Field [1]). A navigation field on Ω is a locally Lipschitz continuous map $\mathfrak{n} : \Omega \times \Omega \to \mathbb{R}^d$ satisfying the following conditions for every $y \in \operatorname{int}(\Omega)$:

(1) $\langle \mathfrak{n}(y, z) \,|\, \nabla_z \beta(z) \rangle \geq 0$ everywhere on $\partial\Omega$ — this ensures workspace invariance, guaranteeing that trajectories never leave the safe region;
(2) $z = y$ is the unique stable equilibrium of $\mathfrak{n}(y, -)$ — this provides a unique convergence point for any desired target;
(3) For almost all initial conditions $z(0) \in \Omega$, the solutions $z(t)$ of $\dot{z} = \mathfrak{n}(y, z)$ converge to y as $t \to \infty$ — this establishes almost global asymptotic stability;

For example, a navigation field can be constructed from a navigation function via

$$\mathfrak{n}(y, z) := -\nabla_z \varphi_y(z) \tag{8}$$

where φ_y is a navigation function with unique minimum at y, with a smooth dependency on the parameter y.

3.1. Invariance and Safety.

3.1. Invariance and Safety. A critical property of navigation fields is their ability to guarantee that trajectories remain within the workspace, ensuring collision avoidance with obstacles. This property can be formally characterized through the concept of positive invariance.

DEFINITION 6 (Positive Invariance). A set $S \subset \mathbb{R}^n$ is said to be positively invariant with respect to a dynamical system $\dot{z} = f(z)$ if for any initial condition $z(0) \in S$, the solution $z(t)$ remains in S for all $t \geq 0$.

For navigation problems, we require Ω to be positively invariant under the dynamics $\dot{z} = \mathfrak{n}(y, z)$ for any target $y \in \Omega$. This ensures that agents never leave the workspace or collide with obstacles. Equivalently, forward-invariance of Ω ensures the agent does not collide with obstacles.

3.1.1. *Nagumo's Condition.* The condition for invariance of a set under a dynamical system was first characterized by Nagumo's Lemma:

THEOREM 7 (Nagumo [25]). *Let $S = \{x \in \mathbb{R}^n : \beta(x) \geq 0\}$ where $\beta : \mathbb{R}^n \to \mathbb{R}$ is a C^1 function with $\nabla\beta(x) \neq 0$ for all but finitely many $x \in \partial S = \{x : \beta(x) = 0\}$. Then S is positively invariant under the dynamical system $\dot{x} = f(x)$ if and only if*

$$\langle f(x) \,|\, \nabla\beta(x) \rangle \geq 0 \quad \textit{for all } x \in \partial S. \tag{9}$$

This theorem was generalized to non-smooth boundaries by Bony and Brezis [26, 27]. In the context of navigation fields, this theorem directly relates to condition (1) in Definition 5. When the workspace is represented as $\Omega = \{z \in \mathbb{R}^d : \beta(z) \geq 0\}$, the condition:

$$\left\langle \mathfrak{n}(y,z) \,\middle|\, \nabla_z \beta(z) \right\rangle \geq 0 \quad \text{for all } z \in \partial\Omega \tag{10}$$

ensures that trajectories under the dynamics $\dot{z} = \mathfrak{n}(y,z)$ cannot leave Ω.

3.1.2. ***Global Convergence Properties***. While the first condition of a navigation field ensures safety through workspace invariance, conditions (ii-iv) collectively ensure almost global convergence to the target. Classical analysis of such systems often uses Lyapunov functions, but an alternative approach is through Rantzer's dual Lyapunov theorem.

THEOREM 8 (Reformulation of Rantzer's Theorem [28]). Consider a system $\dot{x} = f(x)$ where f is continuously differentiable on $\mathbb{R}^n \setminus \{a\}$. Suppose there exists a non-negative integrable function $\rho : \mathbb{R}^n \setminus \{a\} \to \mathbb{R}$ such that $\rho(x)f(x)$ is locally integrable on $\mathbb{R}^n \setminus \{a\}$ and:

$$\nabla \cdot (\rho(x) f(x)) > 0 \quad \text{for almost all } x \in \mathbb{R}^n \setminus \{a\}, \tag{11}$$

Where $(\nabla\cdot)$ denotes the divergence operator. Then, for almost all initial states $x(0)$, the solution $x(t)$ of the system converges to a as $t \to \infty$.

For navigation fields derived from navigation functions (i.e., $\mathfrak{n}(y,z) = -\nabla_z \varphi_y(z)$), a suitable density function is $\rho(z) = \frac{1}{\varphi_y(z)}$, as demonstrated in [15]. This approach provides an elegant analytical tool for establishing almost global convergence properties of navigation fields.

The combination of invariance (safety) and almost global convergence makes navigation fields particularly well-suited for reactive navigation in complex environments, as they provide strong theoretical guarantees while remaining computationally tractable.

4. Examples Implemented in VGC

4.1. Example 1: Construction of navigation field $\mathfrak{n}_{sph}$.

Let $O_i \triangleq x_i^* + \rho_i \mathbb{B}^d$, $i = 1, \ldots, b$, be a collection of pairwise-disjoint closed balls serving as obstacles,contained in a bounded convex environment provided as d-dimensional polytope in the form $E \triangleq \{x \in \mathbb{R}^d : Mx \leq C\}$, where $M = [m_1 | \cdots | m_a]^\top \in \mathbb{R}^{a \times d}$ and $C = [c_1, \ldots, c_a]^\top \in \mathbb{R}^a$. Also, let $n_j \triangleq -\frac{m_j}{\|m_j\|}$ be an inward facing normal at any point $z \in \partial E$ satisfying $\langle m_j, z \rangle = c_j$.

For each $i = 1, \dots, b$, let $o_i(z) \triangleq x_i^* + \rho_i \frac{z - x_i^*}{\|z - x_i^*\|}$ and let

$$H_i(z) \triangleq \{x \in \mathbb{R}^d : \|x - z\| \leq \|x - o_i(z)\|\} \tag{12}$$

be the perpendicular bisector half-space separating z from $o_i(z)$. The polytope

$$P(z) \triangleq E \cap \bigcap_{i=1}^{b} H_i(z), \tag{13}$$

may be regarded as a neighborhood of z within which navigation is inherently safe.

Following [17], a navigation field $\mathfrak{n}_{sph}$ on $\Omega \triangleq E \setminus \bigcup_{i=1}^{b} O_i$ is constructed by setting

$$\mathfrak{n}_{sph}(y, z) \triangleq \pi_z(y) - z, \tag{14}$$

where $\pi_z : \mathbb{R}^d \to P(z)$ is the nearest-point projection of y onto the polytope $P(z)$.

4.1.1. *Computing $\mathfrak{n}_{sph}$ using Quadratic Programming.* Computing this field efficiently requires formulating a quadratic program (QP) that determines the nearest-point projection $\pi_z(y)$.

First, examining the half-space constraint from (12) algebraically reveals that $x \in H_i(z)$ holds if and only if

$$\left\langle x \,\middle|\, x_i^* - z \right\rangle \leq \frac{\|x_i^* - z\|^2 - \rho_i \|x_i^* - z\|^2}{2} + \left\langle z \,\middle|\, x_i^* - z \right\rangle, \tag{15}$$

where x_i^* is the center of obstacle O_i and ρ_i is its radius.

This inequality represents a linear constraint on x, expressible in the form $\left\langle A_i \,\middle|\, x \right\rangle \leq D_i + \left\langle A_i \,\middle|\, z \right\rangle$, where:

$$A(z)_i \triangleq (x_i^* - z)^\top, \tag{16}$$

$$D(z)_i \triangleq \frac{\|x_i^* - z\|^2 - \rho_i \|x_i^* - z\|^2}{2}. \tag{17}$$

The nearest-point projection $\pi_z(y)$ is then the solution to the following quadratic program:

$$\pi_z(y) = \arg\min_x \|x - y\|^2 \text{ s.t. } \begin{cases} Mx \leq C, \\ Ax \leq D + Az, \end{cases} \tag{18}$$

where the first line of constraints represents the environment boundaries, and the second line represents the obstacle avoidance constraints.

4.1.2. *Implementation.* To efficiently implement the computation of $\mathfrak{n}_{sph}(y, z)$, a strategic change of variables is beneficial. Setting $\xi \triangleq x - y$ reformulates the QP as:

$$\mathfrak{n}_{sph}(y, z) = \arg\min_{\xi} \|\xi\|^2 \text{ s.t. } \begin{cases} M\xi \leq C - My, \\ A\xi \leq D + A(z - y), \end{cases} \tag{19}$$

This formulation offers several computational advantages:

- The objective function $\|\xi\|^2$ has a simple form without cross-terms
- All constraints remain linear in the new variable ξ
- The solution ξ^* directly gives the navigation vector $\mathfrak{n}_{sph}(y, z)$
- The QP can be solved using standard convex optimization techniques

4.1.3. *Performance.* The QP formulation in (19) is particularly well-suited for real-time computation using standard optimization libraries. Pseudocode for implementation:

Algorithm 2-1 Computing $\mathfrak{n}_{sph}(y, z)$

```
1: procedure ComputeNavField($y, z, \{x_i^*, \rho_i\}_{i=1}^b$, $M$, $C$)
2:     Construct matrix $A$ with rows $(x_i^* - z)^\top$ for $i = 1, \ldots, b$
3:     Construct vector $D$ with elements $D_i = \frac{\|x_i^* - z\|^2 - \rho_i \|x_i^* - z\|^2}{2}$
4:     $\tilde{C} \leftarrow C - My$
5:     $\tilde{D} \leftarrow D + A(z - y)$
6:     Solve QP: $\xi^* = \arg\min_{\xi} \|\xi\|^2$ s.t. $M\xi \leq \tilde{C},\ A\xi \leq \tilde{D}$
7:     return $\xi^*$                                   ▷ This equals $\mathfrak{n}_{sph}(y, z)$
```

This approach offers several important advantages:

(1) **Computational efficiency**: The QP has a simple quadratic objective and linear constraints, making it amenable to efficient solution methods.
(2) **Local information**: The construction only requires knowledge of the obstacles and boundaries that are relevant to the current agent position.
(3) **Theoretical guarantees**: The resulting navigation field $\mathfrak{n}_{sph}$ satisfies the properties required for the PnP controller, including (R, δ)-goodness under appropriate conditions.

(4) **Obstacle avoidance**: By construction, following the navigation field guarantees collision-free motion as long as the obstacles remain fixed.

In multi-agent implementation, each agent computes this navigation field independently based on local information, and the PnP controller coordinates their movements while maintaining communication connectivity.

4.2. Example 2: Differential Drive Robot Navigation. The navigation field approach can be extended to non-holonomic robots, which are subject to motion constraints. A common example is the differential drive robot, widely used in mobile robotics applications. While the navigation field $\mathfrak{n}_{sph}$ generates control inputs suitable for fully-actuated systems, additional transformations are needed to apply these control strategies to differential drive robots.

4.2.1. *Differential Drive Kinematics.* A differential drive robot is characterized by the following kinematic model:

$$\dot{z_1} = v \cos\theta \tag{20}$$

$$\dot{z_2} = v \sin\theta \tag{21}$$

$$\dot{\theta} = \omega \tag{22}$$

where (z_1, z_2) represents the position of the robot's center, θ is its orientation, v is the linear velocity, and ω is the angular velocity. The control inputs are $u = [v, \omega]^\top$, rather than direct control of $\dot{z}_1$ and $\dot{z}_2$ as in the fully-actuated case.

4.2.2. *Near-Identity Diffeomorphism Approach.* To bridge the gap between the navigation field output and the differential drive control inputs, a near-identity diffeomorphism approach can be employed, as proposed by Olfati-Saber [29]. This method enables the conversion of control commands designed for fully-actuated systems to suitable inputs for non-holonomic vehicles.

Given a navigation field output $\mathfrak{n}_{sph}(y, \mathbf{z}) = [u_1, u_2]^\top$, the conversion to differential drive commands $[v, \omega]^\top$ is performed as follows:

$$v = u_1 \cos\theta + u_2 \sin\theta \tag{23}$$

$$\omega = \frac{1}{\epsilon}(-u_1 \sin\theta + u_2 \cos\theta) \tag{24}$$

where $\epsilon > 0$ is a small positive constant that determines the aggressiveness of the angular correction.

This transformation effectively projects the desired velocity vector onto the robot's heading direction to determine the linear velocity v, while the angular velocity ω is designed to align the robot's heading with the desired direction of motion.

The separation offset from the robot's center of mass carries with it two non-negligible costs: First, the $\frac{1}{\epsilon}$ factor in ω implies (arbitrarily) high angular acceleration if ϵ is small, resulting in a demand for very powerful actuators, which may be impractical. Second, offsetting the robot's center of mass by ϵ means Ω_r needs to be replaced with $\Omega_{r+\epsilon}$ to maintain forward-invariance, restricting the set of positions accessible to the robot, as well as the set of allowable targets.

4.2.3. *Implementation Results.* Figure 3 shows a simulation of a differential drive robot navigating through a sphere world using the navigation field $\mathfrak{n}_{sph}(y, z)$ combined with the near-identity diffeomorphism transformation. The blue arrows represent the navigation field, while the orange trajectory shows the path taken by the differential drive robot from its starting position (yellow) to the target (red X).

Despite the additional kinematic constraints, the robot successfully navigates around multiple obstacles while maintaining a smooth trajectory toward the goal. The vector field visualization demonstrates how the navigation field guides the robot through the complex environment, providing obstacle avoidance guarantees even under non-holonomic constraints.

This case study demonstrates that the navigation field approach can be effectively extended to non-holonomic systems through appropriate transformations, making it applicable to a wide range of practical mobile robot platforms.

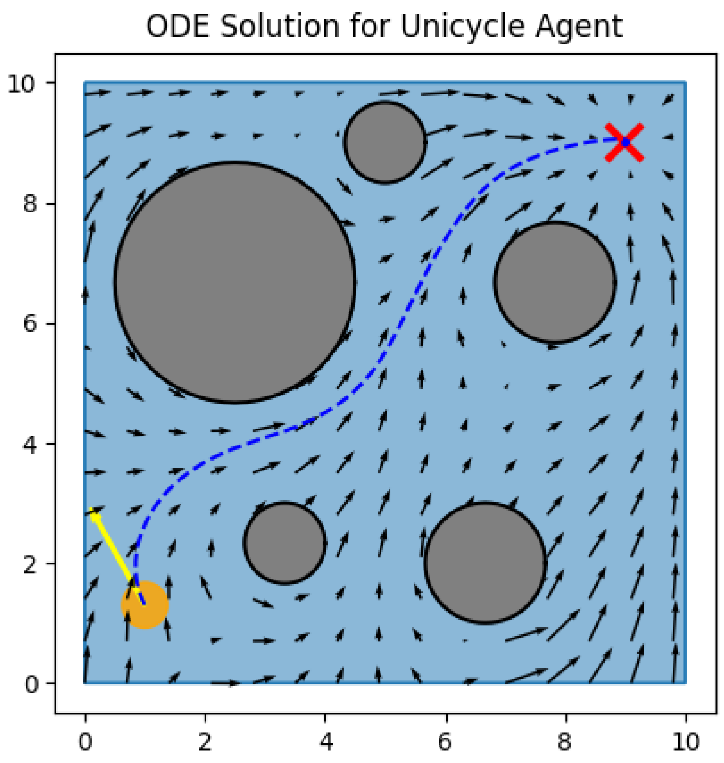

FIGURE 3. Differential drive agent navigating in a sphere world using the computed navigation field. The agent follows trajectories determined by $\mathfrak{n}_{sph}(y, z)$ transformed via near-identity diffeomorphism, safely avoiding obstacles while moving toward the target (red $\times$). The blue arrows show the navigation field, while the orange trajectory shows the robot's path from its starting position (yellow).

CHAPTER 3

Multi-agent Systems

1. Control Objective and Design Assumptions

DEFINITION 9 (Agent Set). $\mathcal{V}$ is a nonempty finite set indexing the agents of a multi-agent system.

DEFINITION 10 (Configuration). A configuration is a vector $\mathbf{x} \triangleq (x_p)_{p\in\mathcal{V}} \in (\mathbb{R}^d)^{\mathcal{V}}$ representing the states of all agents.

DEFINITION 11 (s-graph of a Configuration). For any $s > 0$ and configuration $\mathbf{x}$, the s-graph of $\mathbf{x}$ is defined as $G_s(\mathbf{x}) = (\mathcal{V}, \mathcal{E}_s(\mathbf{x}))$, where

$$\mathcal{E}_s(\mathbf{x}) \triangleq \{pq : \|x_p - x_q\| \leq s\} \tag{25}$$

is the set of edges available for communication when the communication radius is s.

DEFINITION 12 (s-admissible Configurations). For any graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $s > 0$, the set of s-admissible configurations is:

$$\mathcal{C}_s(\mathcal{G}) \triangleq \{\mathbf{x} \in \Omega^{\mathcal{V}} : \mathcal{E} \subseteq \mathcal{E}_s(\mathbf{x})\} \tag{26}$$

This represents all configurations where communication according to $\mathcal{G}$ is possible.

Agent Dynamics. Each agent $p \in \mathcal{V}$ has state $x_p \in \mathbb{R}^d$ with $d \geq 2$, evolving according to first-order dynamics:

$$\dot{x}_p = u_p, \quad u_p \in \mathbb{R}^d \text{ continuous control input.}$$

Communication Constraints. Two agents $p, q \in \mathcal{V}$ can communicate only if $\|x_p - x_q\| \leq R$. Communication may occur exclusively along edges of a prescribed connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Initial configurations $\mathbf{x}(0)$ must lie in the interior of the feasible set $\mathcal{C}_R(\mathcal{G})$.

Task Specification. A single agent $\ell \in \mathcal{V}$ is designated as the leader and must reach a user-specified target location $x^* \in \text{int}(\Omega)$. Throughout this process, the multi-agent system

(MAS) must preserve the communication structure defined by the graph $\mathcal{G}$ at all times, ensuring $\mathbf{x}(t) \in \mathcal{C}_R(\mathcal{G})$ for every $t \geq 0$. The overall objective is to design a distributed controller satisfying:

(1) **Leader Navigation.** The designated leader $\ell \in \mathcal{V}$ reaches the user-specified target location $x^* \in \text{int}(\Omega)$;
(2) **Graph Maintenance.** The MAS maintains the prescribed communication structure for all time, i.e., $\mathbf{x}(t) \in \mathcal{C}_R(\mathcal{G})$ for all $t \geq 0$;
(3) **Obstacle Avoidance.** All agents remain within the workspace Ω, thus avoiding obstacles at all times

1.1. Edge Potential. To maintain the communication structure among agents while navigating through a workspace these potential functions are used to create virtual forces between agents that preserve the desired graph structure.

DEFINITION 13 (Edge Tension Function). For a non-negative function $r : [0, \infty) \to [0, \infty)$ and agents $p, q \in \mathcal{V}$, the edge tension is defined as:

$$w_{pq}(x) = r(\|x_q - x_p\|)$$

if $pq \in \mathcal{E}$ and $w_{pq} = 0$ otherwise.

DEFINITION 14 (Edge Potential). For each edge $pq \in \mathcal{E}$, the potential energy is derived from the tension function as:

$$V_{pq}(x) = P(\|x_q - x_p\|), \quad P(\sigma) = \int_0^\sigma r(s)s\, ds$$

Note that, when r is constant, V_{pq} is the classical potential energy of a mechanical spring.

DEFINITION 15 (Total Potential). The total potential of the system is the sum of all edge potentials:

$$V_G(x) = \sum_{pq \in \mathcal{E}} V_{pq}(x)$$

To clarify notation, for a function $V : (\mathbb{R}^d)^{\mathcal{V}} \to \mathbb{R}$, we define the partial gradient with respect to agent p as

$$\nabla_p V \triangleq \frac{\partial V}{\partial x_p} \in \mathbb{R}^d, \tag{27}$$

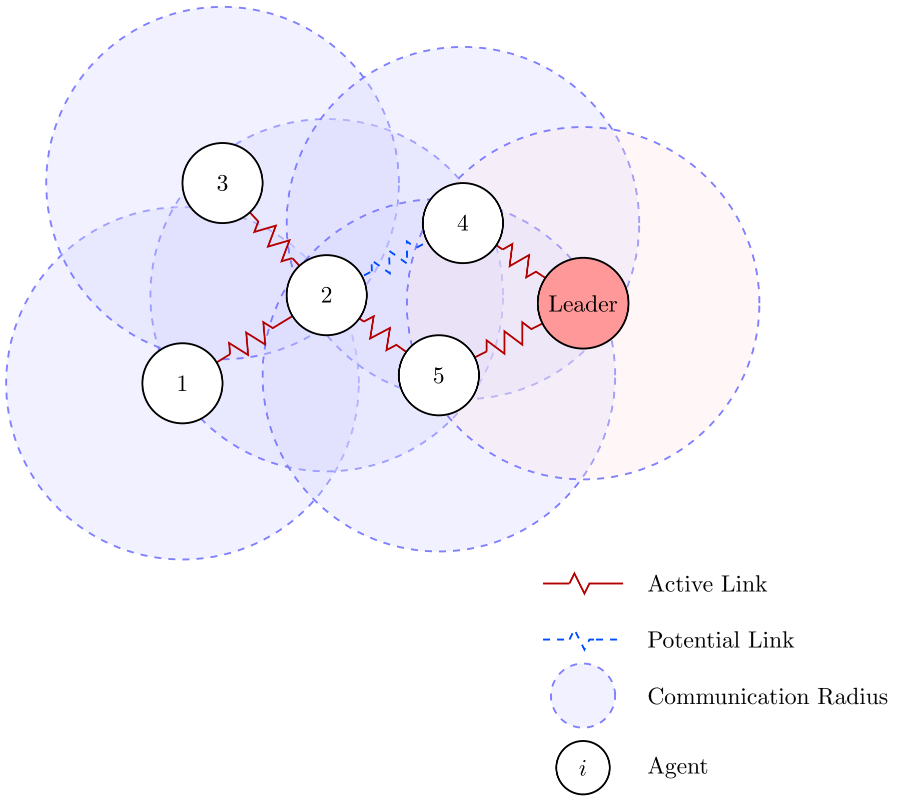

FIGURE 1. Visualization of edge potentials in a multi-agent system. Agents (numbered circles) maintain communication links while navigating through the workspace. The dashed blue circles represent each agent's communication radius R. Active links (solid red) indicate current communication edges in the graph, while potential links (dashed blue) show pairs of agents within communication range but not currently connected. The leader agent (highlighted in red) guides the formation through the environment while the edge potentials create virtual forces that maintain the desired graph structure.

i.e., the gradient of V with respect to the d-dimensional position vector $x_p \in \mathbb{R}^d$. For example:

$$\nabla_p V_{pq} \triangleq \frac{\partial V_{pq}}{\partial x_p} = w_{pq}(x_p - x_q) = -\nabla_q V_{pq} \tag{28}$$

For any other vertex $u \notin \{p, q\}$, we have

$$\nabla_u V_{pq} = 0. \tag{29}$$

2. Weak Invariance Principle (WIP) for Graph Maintenance

The WIP provides conditions under which a controller guarantees that if the system starts in a "safe configuration" (where all distances along graph edges are bounded away from R), it will remain in a valid configuration for all time.

THEOREM 16 (Weak Invariance for Graph Maintenance). *Let V_G be the total edge potential obtained via edge potentials with tension function $r : [0, \infty) \to [0, \infty)$ having only finitely many jump discontinuities. Further, let $\varrho \in (0, R]$ satisfy:*

$$|\mathcal{E}|P(\varrho) < P(R) \tag{30}$$

Suppose $\mathbf{x}(t) : \mathbb{R}_{\geq 0} \to \Omega^{\mathcal{V}}$ is continuous and piecewise C^1 with bounded $\dot{\mathbf{x}}(t)$ and $\mathbf{x}(0) \in \mathcal{C}_\varrho(G)$, and suppose $\dot{V}_G \leq 0$ whenever $\|\Delta\mathbf{x}(t)\|_\infty \in [\varrho, R]$. Then, $\mathbf{x}(t) \in \mathcal{C}_R(G)$ for all time.

The term "weak invariance" is used because the result provides a semi-global guarantee rather than a global one: For any $\varrho < R$, only trajectories starting in the "safe configurations" of $\mathscr{C}_\varrho(\mathcal{G})$ are guaranteed to remain in $\mathscr{C}_R(\mathcal{G})$ for all time, for appropriately selected values of the control parameters. Despite these limitations, the WIP is a powerful tool for controller design, as it provides clear, verifiable conditions that ensure communication links are maintained throughout the system's operation.

Proof Intuition. The proof works by contradiction:

(1) Assume a trajectory starts in a safe configuration but eventually leaves the set of R-admissible valid configurations;
(2) This means at some point in time, at least one edge reaches exactly length R;
(3) At this point, the total potential V_G would have increased from its initial value;
(4) The inequality (30) ensures $\dot{V}_G \leq 0$ whenever any edge is at risk (length between ϱ and R), a contradiction to the preceding observation.

The proof from [1] is provided in the appendix for reference.

2.1. Design of Tension Function. Two variants of the PnP controller arise from the following design of the tension function:

$$r(s) = \begin{cases} \mu, & \text{if } s \in [0, \varrho) \\ \mu + \omega(s-\varrho)^{1+\alpha}, & \text{if } s \in [\varrho, R] \\ 0, & \text{if } s \in (R, \infty) \end{cases} \tag{31}$$

where $\mu, \alpha \geq 0$, $\omega > 0$ are parameters determining the behavior of the PnP controller, which will be introduced in the next section.

(1) The contractive PnP controller arises when $\mu > 0$, characterized by a tendency of distances between neighboring agents to contract;
(2) The lazy PnP controller, obtained when $\mu = 0$, is characterized by the absence of attractive interactions between neighboring agents at distances below the safe distance ϱ

3. Controller Design

For a connected graph $G = (\mathcal{V}, \mathcal{E})$ and a navigation field $\mathfrak{n}$, the Plug-and-Play (PnP) distributed controller is defined as:

$$u_p(x) = \sum_{q \sim p} \xi_q^p \mathfrak{n}_q^p + v_p \tag{32}$$

where:

- $\mathfrak{n}_q^p(x) = \mathfrak{n}(x_q, x_p)$ is the navigation field evaluated from agent p to neighbor q;
- $\xi_q^p(x) = \xi(x_q, x_p)$ with

$$\xi(y, z) = \frac{r(\|y-z\|)\|y-z\|^2}{\langle \mathfrak{n}(y,z), y-z \rangle} \tag{33}$$

- $r : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a tension function;
- $v = (v_p)_{p \in \mathcal{V}}$ is the task component defined as:

$$v_\ell(x) = \gamma \mathfrak{n}(x^*, x_\ell) - \sum_{q \sim \ell} \xi_\ell^q \mathfrak{n}_\ell^q \tag{34}$$

with $v_p(x) = 0$ for $p \in \mathcal{V},\ p \neq \ell$, and $\gamma > 0$ a gain parameter. This leader design combines two effects: attraction to a target x^* and compensation for incident edge tensions. The subtraction of local edge terms ensures that v_ℓ is independent of the internal force structure, isolating the effect of the goal-driven behavior.

Importantly, this is not the only possible choice for the leader's task input. Any bounded control input $v_\ell \in \mathbb{R}^d$ would suffice, though different designs may require adjusting the minimum value of γ to maintain graph connectivity or preserve stability conditions. This form is chosen for clarity and analytical tractability, as it ensures that the leader's motion towards the target remains unperturbed by the edge-based graph tension structure.

3.1. (R,δ)**-Goodness Condition.** The design of the coefficients ξ_q^p in (33) is facilitated by the following critical tameness assumption about the navigation field.

DEFINITION 17. Let $\delta \in (0,1]$. A navigation field $\mathfrak{n}$ on Ω is (R,δ)-good if for all $y, z \in \Omega$ with $\|y-z\| \le R$:

$$\langle \mathfrak{n}(y,z), y-z \rangle \ge \delta \|\mathfrak{n}(y,z)\| \|y-z\| \tag{35}$$

This property ensures that the navigation field is sufficiently aligned with the radial field between agents for nearby targets. Geometrically, it means that the angle between $\mathfrak{n}(y,z)$ and the vector $y-z$ is bounded, with $\cos\angle(\mathfrak{n}(y,z), y-z) \ge \delta$. The (R,δ)-goodness property enables a decomposition of the navigation field that facilitates analysis of the controller's behavior. This decomposition separates the component of the field that acts directly along the line between agents from the orthogonal component, providing insight into how the navigation field influences inter-agent distances and graph maintenance.

3.2. Variants of Tension Functions. The edge tension function $r : \mathbb{R}_{\ge 0} \to \mathbb{R}_{\ge 0}$ plays a critical role in the design of the Plug-and-Play (PnP) controller. [1] examines two principal variants defined in (31) where $\mu, \alpha \ge 0$, $\omega > 0$ are parameters. Notably, $r(s) \ge \mu$ for all $s \in [0,R]$, consistent with μ's role in ensuring system stability.

Contractive PnP Controller. The contractive variant arises when $\mu > 0$. This configuration ensures that:

(1) The edge tension never vanishes within the communication range $[0,R]$;
(2) The negative-definite term in the potential derivative remains active for all edges with non-zero weight;
(3) The system maintains stronger cohesion properties as described in Theorem 1.

This formulation is particularly advantageous in applications requiring robust connectivity maintenance and disturbance rejection, as it provides stronger guarantees on the persistence of inter-agent connections.

Lazy PnP Controller. The lazy variant corresponds to $\mu = 0$, allowing:

(1) More flexible inter-agent configurations where edges may have zero tension;
(2) Reduced control effort in regions where strict connectivity is not required;
(3) Enhanced adaptability to changing environment conditions.

However, this flexibility comes at the cost of weaker connectivity guarantees, as edges with null weights no longer contribute to the negative-definite terms in stability analysis. The selection between these variants depends on the specific requirements of the multi-agent application, balancing the trade-off between strict connectivity maintenance and operational flexibility. For tasks prioritizing guaranteed graph topology maintenance, the contractive variant provides stronger theoretical guarantees, while the lazy variant offers greater adaptability for formation reconfiguration tasks.

3.2.1. *Key PnP Controller Result.* To summarize the main theoretical contribution of the PnP approach, we restate the primary result of [1, Theorem 2]:

THEOREM 18 (PnP Controller Graph Maintenance). *Suppose $\mathcal{G}$ is a N-path and the leader ℓ is an end of $\mathcal{G}$. Let $\mathfrak{n}$ be an (R, δ)-good navigation field and let $U \triangleq \sup_{z\in\Omega} \left\|\mathfrak{n}(x^*, z)\right\|$. Further, let $\delta^* \triangleq \frac{\sqrt{1-\delta^2}}{\delta}$, and suppose that the following holds:*

$$\delta^* \cdot M^2 m^2 \leq \frac{4}{9N} sin^4 \frac{\pi}{2N}. \tag{36}$$

Then, for any $\gamma \leq \gamma^*$, where

$$\gamma^* \triangleq \tfrac{\delta^*}{2U}\sqrt{N}R \cdot \begin{cases} \mu \cdot \frac{2+\alpha}{6}\left|\mathcal{E}\right|, & \mu > 0, \\ \omega \cdot R(R-\varrho)^{1+\alpha}, & \mu = 0, \end{cases} \tag{37}$$

any trajectory $\mathbf{x}(t)$ of $\mathbf{u}$ with initial condition $\mathbf{x}(0) \in \mathscr{C}_\varrho(\mathcal{G})$ satisfies the following statements:

(1) Graph maintenance: $\mathbf{x}(t) \in \mathcal{C}_R(G)$ for all $t \geq 0$
(2) Target acquisition: $\lim_{t\to\infty} \|x_\ell(t) - x^*\| = 0$
(3) Obstacle avoidance: $x_p(t) \in \Omega$ for all $p \in \mathcal{V}$ and $t \geq 0$

This result provides the foundational guarantee that enables distributed navigation with communication maintenance for multi-agent systems. The VGC framework presented in this thesis builds upon this theoretical foundation, providing a practical platform for empirical investigation of the gap between theoretical bounds and actual performance limits. The WIP (Theorem 16) requires $\dot{V}_G \leq 0$ whenever any edge length is between ϱ and R. For the PnP controller, this condition translates into the specific bounds on the controller parameters expressed in equations (36) and (37). At a high level, the PnP controller can be analyzed by considering how it influences the rate of change of the total potential energy. This analysis reveals that the controller must balance several competing factors:

- The goodness of the navigation field δ (how well aligned it is with direct paths);
- The leader's task gain γ (how aggressively it pursues its target);
- The graph structure parameters (connectivity, degree, size);
- The tension function design (how strongly agents are pulled together);
- The communication radius R and safety radius ϱ.

These bounds ensure that the attractive forces between agents dominate any forces that might pull them apart, thus satisfying the requirements of the WIP.

3.3. Practical Considerations and Simulation Insights. While the theoretical bounds derived for the PnP controller guarantee graph maintenance, they are often highly conservative in practice. Empirical evidence from simulations reveals that the controller can tolerate significantly higher leader gains than those predicted by theory. For instance, in a chain of 15 agents navigating through complex environments with non-convex obstacles, the theoretical bound on the leader gain γ may be conservative by a factor of 10^4 or more. This discrepancy arises from the worst-case assumptions made in the analytical derivations. The theoretical bounds must guarantee graph maintenance under all possible configurations and obstacle arrangements, whereas in typical scenarios, the system operates far from these worst-case conditions.

3.3.1. *Leader Dynamics and Task Flexibility.* The (R, δ)-goodness condition is particularly powerful because it essentially guarantees exponential convergence to any target within an R-ball. This property can be interpreted as establishing a Lyapunov function $V(y, z) = \|y - z\|$ with exponential convergence rate for the navigation field. Consequently, the leader task component $v_\ell(x) = \gamma \mathfrak{n}(x^*, x_\ell)$ can be replaced by any bounded signal, not just a navigation field directed toward a fixed target. This flexibility opens possibilities for more complex leader behaviors, such as:

- Time-varying targets for dynamic task execution;

- Reactive obstacle avoidance with changing navigation goals;
- Formation control with evolving geometric configurations

CHAPTER 4

VGC Software Framework

While [1] introduced rigorous theoretical conditions (e.g., (R, δ)-goodness of the input navigation field) required for maintaining graph connectivity under PnP controllers, simulations showed that these theoretical bounds are often overly conservative. The VGC framework is intended for empirical investigation of how significantly these constraints can be relaxed in realistic environments through systematic experimentation within a ROS2-based testbed.

The complete implementation of the VGC framework described in this thesis is publicly available as open-source software in the following GitHub repository: `https://github.com/kotmasha/variable_graph_MAS`.

1. ROS2 Communication Framework

Unlike its predecessor, ROS2 implements the Data Distribution Service (DDS) standard, providing robust peer-to-peer communication for distributed systems without requiring a central master node [30]. This design choice allows agents to operate independently, ensuring that failure of a single robot does not compromise the entire system.

1.1. DDS in Depth. The service underpinning ROS2 provides several critical capabilities that directly support the VGC framework:

- **Quality of Service (QoS) Profiles**: DDS allows fine-grained control over communication reliability, history, and durability. For the VGC framework, we leverage these profiles to:
  - Configure position updates with RELIABLE QoS to ensure graph maintenance guarantees
  - Use smaller history depths for high-frequency control messages to optimize bandwidth
  - Apply different durability settings for persistent vs. transient data

- **Domain Partitioning**: DDS domains provide network isolation, allowing multiple independent multi-agent teams to operate in the same physical space without message interference.
- **Content-Filtered Topics**: These enable selective message reception based on content, supporting advanced features like distance-based communication filtering without requiring complete message transmission.
- **Discovery Mechanism**: DDS's built-in discovery aligns with the dynamic graph management approach, automatically handling peer detection when communication edges are established or removed.

The VGC framework makes deliberate use of these DDS capabilities, mapping the mathematical graph maintenance constraints directly to corresponding DDS mechanisms, ensuring communication compliance with theoretical requirements.

Each agent runs its own instance of a compiled ROS2 package, communicating only with its defined neighbors in the current communication graph. Topics used by each agent are uniquely namespaced (e.g., `/agent_i/pose`, `/agent_i/cmd_vel`), ensuring clean isolation between agents and preventing topic collisions.

The VGC framework implements a genuinely distributed multi-agent system where:

(1) **No Central Coordinator**: Unlike many simulated MAS implementations that rely on centralized computation with distributed execution, VGC agents make decisions using only locally available information.
(2) **Independent Computation**: Each agent independently computes its control inputs based on its local view of the environment and communication with neighbors.
(3) **Asynchronous Operation**: Agents operate on their own processing cycles without global synchronization requirements, making the system more robust to varying computational capabilities and network delays.
(4) **Scalability**: The computational load per agent stays proportional to the number of neighbors, enabling linear scaling with the number of agents.

This distributed architecture portrays the real-world robotic deployments while providing a platform to investigate the practical bounds of theoretical guarantees under realistic communication and computational constraints.

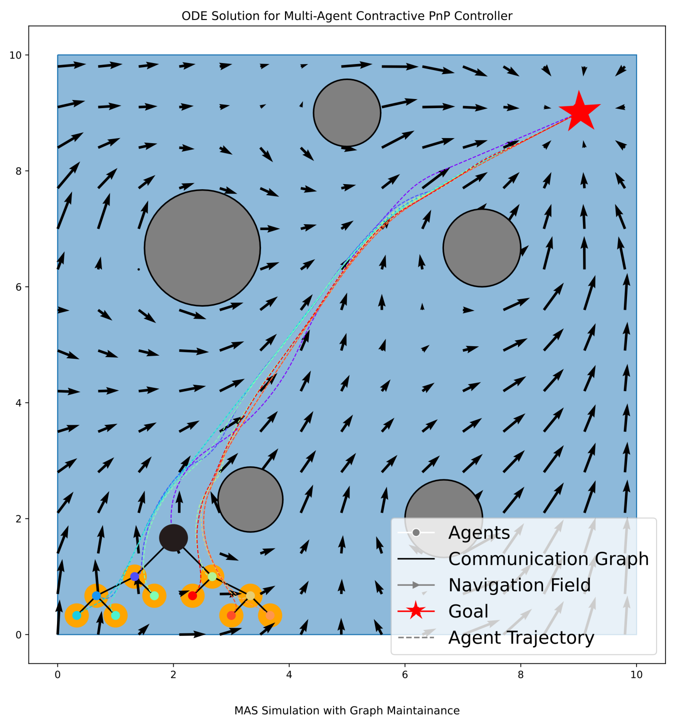

FIGURE 1. Multi-agent system navigation through an environment with obstacles. The figure shows the agent trajectories (colored paths), navigation field (blue arrows), communication graph (pink lines connecting agents), and goal position (red star). The solution demonstrates how agents maintain connectivity while navigating around circular obstacles toward the target.

2. Modular Software Architecture

A key strength of the VGC framework is its highly modular design, which enables flexible configuration and extension through a comprehensive class hierarchy. This modular architecture allows for rapid prototyping of different scenarios and controller configurations without modifying core functionality.

2.1. Environment Class Hierarchy. The framework implements environments through an abstract base class with specialized subclasses:

- **BaseEnvironment**: Defines the interface for all environment types
- **SphereEnvironment**: Implements sphere worlds with circular obstacles

- **StarEnvironment**: Handles star-shaped obstacles using diffeomorphisms, leveraging Vasilopoulos's state-of-the-art implementation from [2] — this integration is particularly significant as it allows VGC to benefit from cutting-edge reactive navigation techniques without reimplementing complex diffeomorphism constructions
- **PolygonEnvironment**: Supports arbitrary polygonal workspaces and obstacles, incorporating Arslan's sensor-based navigation field implementation from [17] — another crucial adoption that demonstrates VGC's ability to seamlessly integrate existing high-quality navigation solutions

This hierarchical design exemplifies the power of VGC's modular approach: by creating appropriate interfaces to state-of-the-art navigation field implementations, we can focus research efforts on the multi-agent coordination aspects rather than reimplementing established single-agent techniques. Each environment class encapsulates its specific geometric representations and implements navigation field computation methods appropriate for that environment type, allowing agents to navigate through different environment types using a common interface.

Importantly, other environment and agent types conforming to the BaseEnvironment class type and methods can be added to the system, which will automatically scan for them and add them when they are invoked. This plugin architecture is a significant contribution of the VGC framework, enabling seamless integration of new navigation field implementations without modifying the core system, representing a significant capability enhancement that would otherwise require months of specialized development.

2.2. Agent Class Hierarchy. Similarly, the agent architecture employs inheritance to support different robot kinematic models:

- **BaseAgent**: Defines common agent functionality
- **HolonomicAgent**: Implements fully-actuated agents with direct velocity control
- **UnicycleAgent**: Implements differential drive kinematics with near-identity diffeomorphism transformations

This hierarchy enables the PnP controller to be applied across different robot platforms while maintaining the underlying mathematical guarantees.

2.3. Controller Integration. Each agent integrates with the environment through the controller interface, which computes appropriate control inputs based on the environment's navigation field and the agent's state. The distributed nature of the system is reflected in this architecture, where:

(1) The environment provides navigation field computation but is accessed locally by each agent
(2) Each agent manages its own controller instance
(3) Control decisions are made without global knowledge of the entire system state

2.4. Configuration-Driven Instantiation. The VGC framework employs a configuration-driven approach where simulation/experimentation parameters are specified in YAML files:

For example:

```
environment:
  type: sphere_world
  obstacles:
    - center: [2.5, 2.5]
      radius: 1.0
    - center: [5.5, 4.5]
      radius: 0.8
  boundary: [0, 0, 10, 8]

agents:
  - id: 0
    type: unicycle
    position: [0.8, 1.2]
    is_leader: true
    target: [9.0, 7.0]
  - id: 1
    type: unicycle
    position: [1.5, 0.9]
```

During initialization, the framework parses this configuration and dynamically instantiates the appropriate environment and agent subclasses. This approach enables:

- Rapid experimentation with different environment configurations
- Testing of mixed agent types within the same simulation

- Easy adjustment of controller parameters without code changes
- Reproducible experiments through version-controlled configuration files

Additionally, the configuration-driven architecture is designed with extensibility as a primary goal. New agent types, controller algorithms, and environment representations can be seamlessly integrated into the framework by:

- Implementing the appropriate base class interfaces (BaseAgent, BaseController, or BaseEnvironment)
- Adding the new implementation to the registration system
- Updating configuration files to utilize the new components

This plugin architecture allows researchers to easily extend VGC's capabilities without modifying core functionality. For example, a researcher could implement a new reactive controller algorithm, register it with the system, and immediately compare its performance against the PnP controller across various environments and agent configurations.

2.5. Efficient QP Implementation. For the critical quadratic programming component that computes the navigation field $\mathfrak{n}_{sph}$, the framework utilizes the `PIQP` solver from the `qpsolvers` package. This high-performance solver is particularly well-suited for real-time control applications, providing:

- Low computational overhead suitable for onboard deployment
- Warm-starting capabilities for efficient sequential solutions
- Robust handling of constraints arising from complex obstacle arrangements

3. Advantages of the PnP Approach for Coordination

A significant advantage of the PnP controller approach is its ability to append existing navigation solutions as "black boxes" without requiring knowledge of their internal implementation details. This enables:

(1) **Complexity Encapsulation**: The underlying complexity of specialized navigation methods remains encapsulated, with only the navigation field interface exposed to the PnP controller. In fact, the PnP control scheme can be replaced with any distributed control scheme.
(2) **Accelerated Development**: By leveraging existing, well-tested navigation components, development focus can shift to multi-agent coordination aspects rather than reimplementing single-agent navigation.

This "plug-and-play" philosophy extends beyond just the controller to the entire framework, allowing researchers to rapidly test hypotheses about multi-agent coordination without being constrained by specific navigation implementations.

4. ROS2 Package Structure

The software is organized into a ROS2 package called `nav_sim_pkg`, containing the following directory structure:

- **launch/**: Contains ROS2 launch files to start simulations.
- **config/**: Contains YAML configuration files for workspace and graph parameters.
- **src/**: Contains the core node implementations for control and visualization.
- **rviz/**: Contains RViz configuration files for visualization.

5. Communication Graph Management

The system implements service-based graph management using ROS2 services. Each agent provides two key services:

- `addEdge`: Adds a new neighbor to the agent's communication list.
- `rmEdge`: Removes an existing neighbor from the agent's communication list.

These services enable dynamic reconfiguration of the communication graph at runtime. This design facilitates future extensions such as:

- Dynamic topology control based on workspace conditions.
- Formation changes or reorganization of agent roles.
- Adaptive networking strategies that react to communication loss or environment changes.

6. Agent Software Architecture

Each agent in the VGC framework operates as an independent computational entity, fully encapsulating the principles of distributed control. The node-based architecture enables agents to maintain autonomy, in the sense that each agent independently computes its control input based on local state information and messages received from neighboring agents, relying on ROS-2 based message passing rather than on a central coordinator.

6.1. Neighbor State Tracking. The neighbor subscriber component dynamically manages connections to other agents' state topics based on the current communication graph. Unlike traditional approaches that subscribe to all possible agents, this selective subscription mechanism only establishes connections with agents that are explicitly designated as neighbors, significantly reducing communication overhead in large-scale deployments.

6.2. Graph Topology Management. Two critical services, `addEdge` and `rmEdge`, are available to each agent, providing programmatic control over the communication graph structure. These services enable runtime modification of the neighbor relationships, supporting both static formations and dynamically evolving topologies. The graph service server component maintains internal adjacency information and coordinates the necessary subscription changes when graph modifications occur.

6.3. Distributed Control Computation. The core control algorithm processes local and neighbor state information to compute appropriate control inputs. It leverages the navigation field computation (particularly `navfSphere` for sphere worlds) to generate single-integrator control vectors that maintain both workspace constraints and graph connectivity requirements. This component implements the mathematical formulation of the PnP controller (32), including parameter scaling based on inter-agent distances.

6.4. Kinematic Adaptation Layer. For physical robots with nonholonomic constraints, the kinematic conversion component applies a near-identity diffeomorphism transformation. This critical step converts the idealized single-integrator control vectors into executable velocity commands (v, ω) appropriate for differential drive robots, ensuring that theoretical guarantees translate effectively to practical implementations.

This modular architecture enables independent operation while maintaining the coordination properties necessary for robust multi-agent behavior, with each component focusing on a specific aspect of the distributed control challenge.

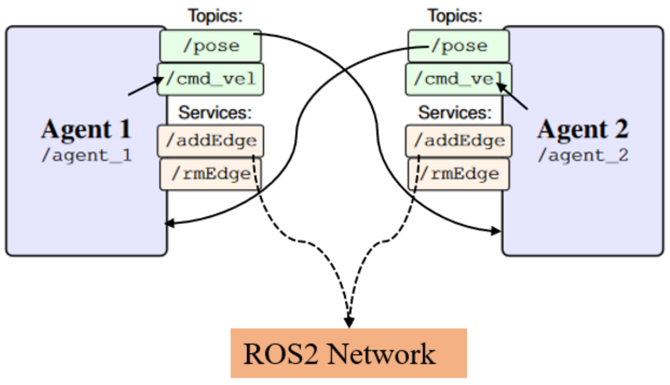

FIGURE 2. Architecture of a distributed multi-agent system over a ROS 2 network. Each agent node publishes its state via the `/pose` topic and receives velocity commands via `/cmd_vel`. Dynamic graph maintenance is supported through services `/addEdge` and `/rmEdge`, which enable agents to manage communication links at runtime. All interactions are coordinated over a shared ROS 2 communication layer.

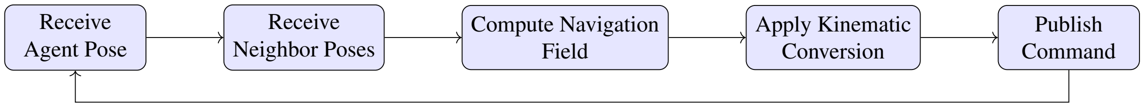

FIGURE 3. Control flow within each agent node in the VGC framework.

7. Control Flow Within Each Agent

The internal control loop within each agent operates as follows:

(1) Receive own pose estimate.
(2) Subscribe to poses of all neighbors (from current neighbor list).
(3) Compute control input using the navigation field $\mathrm{navfSphere}$.
(4) Convert control input to nonholonomic command using near-identity diffeomorphism.
(5) Publish velocity command (v, ω) to the topic `/agent_i/cmd_vel`.

8. Algorithm Implementation

The navigation field computation is a critical component of the VGC framework. Algorithm 4-1 presents the pseudocode for computing $\mathfrak{n}_{sph}(y, z)$ in the sphere world environment.

Algorithm 4-1 Computing Navigation Field $\mathfrak{n}_{sph}(y, z)$

```
1: procedure ComputeNavField($y, z, \{x_i^*, \rho_i\}_{i=1}^b, M, C$)
2:     Construct matrix $A$ with rows $(x_i^* - z)^\top$ for $i = 1, \dots, b$
3:     Construct vector $D$ with elements $D_i = \frac{\|x_i^* - z\|^2 - \rho_i \|x_i^* - z\|^2}{2}$
4:     $\tilde{C} \leftarrow C - My$
5:     $\tilde{D} \leftarrow D + A(z - y)$
6:     Solve QP: $\xi^* = \arg\min_{\xi} \|\xi\|^2$ s.t. $M\xi \leq \tilde{C},\ A\xi \leq \tilde{D}$
7:     return $\xi^*$                                  ▷ This equals $\mathfrak{n}_{sph}(y, z)$
```

The VGC framework implements this algorithm using the PIQP solver, optimized for real-time performance. For the lazy PnP controller, Algorithm 4-2 shows the computation of the control inputs.

Algorithm 4-2 Computing Lazy PnP Control Input

```
1: procedure ComputeLazyPnP(agent, neighbors, target, $\omega$, $\gamma$, $\varrho$, $R$, $\alpha$)
2:     $u \leftarrow \mathbf{0}$                                    ▷ Initialize control vector
3:     for each neighbor $q$ in neighbors do
4:         $y \leftarrow$ neighbor.position
5:         $z \leftarrow$ agent.position
6:         $d \leftarrow \|y - z\|$                                ▷ Inter-agent distance
7:         if $d \geq \varrho$ then          ▷ Only apply force when distance exceeds safe radius
8:             $r \leftarrow \omega(d - \varrho)^{1+\alpha}$                        ▷ Compute tension
9:             $\mathfrak{n} \leftarrow$ ComputeNavField($y$, $z$, obstacles)
10:            $\xi \leftarrow \frac{r \cdot d^2}{\langle \mathfrak{n}, y - z \rangle}$                        ▷ Compute coefficient
11:            $u \leftarrow u + \xi \cdot \mathfrak{n}$                   ▷ Accumulate control contributions
12:    if agent.isLeader() then
13:        $\mathfrak{n}_{\text{target}} \leftarrow$ ComputeNavField(target, agent.position, obstacles)
14:        $u \leftarrow \gamma \cdot \mathfrak{n}_{\text{target}}$                          ▷ Leader follows target
       return $u$
```

CHAPTER 5

Conclusion

This thesis introduced Variable Graph Control (VGC), a modular ROS2-based framework for implementing distributed multi-agent control algorithms with communication graph maintenance guarantees. Through the implementation of Plug-and-Play cooperative navigation controllers, VGC bridges the gap between theoretical models and practical robotic systems.

The primary research contribution lies in creating a platform that enables systematic experimentation, both numerical and physical, using a modular software system that is agnostic to the nature of its agential components. While mathematical analysis provides rigorous connectivity guarantees under specific conditions, our implementations consistently succeeded with parameter values orders of magnitude less restrictive than theory suggests.

VGC's architecture leverages ROS2's distributed communication capabilities to implement truly decentralized control, allowing each agent to compute its actions independently while maintaining global coordination constraints. The framework's modularity supports future extensions to different environment types, robot kinematics, and control laws.

Additionally, I'm particularly interested in extending the VGC framework to support heterogeneous teams with varying sensing and actuation capabilities. This would better reflect real-world deployment scenarios where robots may have different physical constraints and communication ranges.

Ultimately, my hope is that this work contributes to bridging the gap between theoretical elegance and practical implementation in multi-agent systems, bringing us closer to deploying reliable, distributed robotic teams in complex real-world environments.

Bibliography

[1] D. P. Guralnik, P. F. Stiller, F. M. Zegers, and W. E. Dixon, "Plug-and-Play Cooperative Navigation: From Single-Agent Navigation Fields to Graph-Maintaining Distributed MAS Controllers," *IEEE Transactions on Automatic Control*, vol. 69, no. 8, pp. 5262–5277, 2024.

[2] V. Vasilopoulos and D. E. Koditschek, "Reactive navigation in partially known non-convex environments," in *Proc. Intl. Workshop Algo. Found. Robot.*, pp. 406–421, 2018.

[3] D. P. Spanos and R. M. Murray, "Robust connectivity of networked vehicles," in *Proceedings of the Conference on Decision and Control*, vol. 3, pp. 2893–2898, IEEE, 2004.

[4] Z. Kan, L. Navaravong, J. Shea, E. Pasiliao, and W. E. Dixon, "Graph Matching Based Formation Reconfiguration of Networked Agents with Connectivity Maintenance," *IEEE Trans. Control Netw. Syst.*, vol. 2, pp. 24–35, Mar. 2015.

[5] W. Ren, "Consensus based formation control strategies for multi-vehicle systems," in *Proc. Am. Control Conf.*, pp. 4237–4242, IEEE, 2006.

[6] Y. Fan and G. Hu, "Connectivity-preserving rendezvous of multi-agent systems with event-triggered controllers," in *Proc. IEEE Conf. Decis. Control*, pp. 234–239, 2015.

[7] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. Robot. Autom.*, vol. 20, no. 2, pp. 243–255, 2004.

[8] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[9] D. V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos, and M. M. Zavlanos, "A feedback stabilization and collision avoidance scheme for multiple independent non-point agents," *Automatica*, vol. 42, pp. 229–243, 2006.

[10] P. Tabuada, G. J. Pappas, and P. Lima, "Motion feasibility of multi-agent formations," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 387–392, 2005.

[11] F. Zegers, *Lyapunov-Based Control of Distributed Multi-Agent Systems With Intermittent Communication*. PhD thesis, University of Florida, 2021.

[12] D. E. Koditschek and E. Rimon, "Robot navigation functions on manifolds with boundary," *Adv. Appl. Math.*, vol. 11, pp. 412–442, dec 1990.

[13] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.*, vol. 8, pp. 501–518, oct 1992.

[14] S. G. Loizou, "The navigation transformation," *IEEE Trans. Robot.*, vol. 33, pp. 1516–1523, December 2017.

[15] I. F. Filippidis and K. J. Kyriakopoulos, "Navigation functions for everywhere partially sufficiently curved worlds," in *Proc. IEEE Intl. Conf. Robot. Autom.*, pp. 2115–2120, 2012.

[16] S. Paternain, D. E. Koditschek, and A. Ribeiro, "Navigation Functions for Convex Potentials in a Space With Convex Obstacles," *IEEE Trans. Autom. Control*, vol. 63, no. 9, pp. 2944–2959, 2018.

[17] O. Arslan and D. E. Koditschek, "Sensor-based reactive navigation in unknown convex sphere worlds," *Intl. J. Robot. Res.*, vol. 38, no. 2-3, pp. 196–223, 2019.

[18] O. Arslan, D. P. Guralnik, and D. E. Koditschek, "Coordinated robot navigation via hierarchical clustering," *IEEE Trans. Robot.*, vol. 32, no. 2, pp. 352–371, 2016.

[19] H. G. Tanner, S. G. Loizou, and K. J. Kyriakopoulos, "Nonholonomic navigation and control of cooperating mobile manipulators," *IEEE Trans. Robot. Autom.*, feb 2003.

[20] X. Chu, R. Ng, H. Wang, and K. W. S. Au, "Feedback control for collision-free nonholonomic vehicle navigation on se(2) with null space circumvention," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 6, pp. 5594–5604, 2022.

[21] P. Reverdy, V. vasilopoulos, and D. E. Koditschek, "Motivation dynamics for autonomous composition of navigation tasks," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1239–1251, 2021.

[22] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the Performance of ROS2," in *Proceedings of the 13th International Conference on Embedded Software (EMSOFT '16)*, pp. 1–10, ACM, 2016.

[23] H. Kumar, S. Paternain, and A. Ribeiro, "Navigation of a Quadratic Potential with Ellipsoidal Obstacles," in *Proc. IEEE Conf. Decis. Control*, pp. 4777–4784, 2019.

[24] H. K. Khalil, *Nonlinear Systems*. Upper Saddle River, NJ: Prentice Hall, 3 ed., 2002.

[25] F. Blanchini and S. Miani, *Set-Theoretic Methods in Control*. Boston: Birkhäuser, 2008.

[26] J.-M. Bony, "Principe du maximum, inégalite de harnack et unicité du problème de cauchy pour les opérateurs elliptiques dégénérés," *Annales de L'Institut Fourier*, vol. 19, no. 1, pp. 277–304, 1969.

[27] H. Brezis, "On a characterization of flow-invariant sets," *Commun. Pure Appl. Math.*, vol. 23, pp. 261–263, 1970.

[28] O. Karabacak, R. Wisniewski, and J. Leth, "On the almost global stability of invariant sets," in *2018 European Control Conference (ECC)*, pp. 1648–1653, IEEE, 2018.

[29] R. Olfati-Saber, "Near-identity diffeomorphisms and exponential ϵ-tracking and ϵ-stabilization of first-order nonholonomic SE(2) vehicles," *Proceedings of the American Control Conference*, vol. 6, pp. 4690–4695, 2002.

[30] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

APPENDIX A

Appendix

1. Example of a conjugating diffeomorphism, following [2, 1]

To illustrate the diffeomorphism approach with a concrete example, consider a star-shaped obstacle O_j that can be parametrized in polar coordinates centered at x_j^* as:

$$\gamma_j(\theta) = x_j^* + r_j(\theta)[\cos\theta, \sin\theta]^T \tag{38}$$

where $r_j(\theta)$ is a smooth periodic function. A barrier function can be defined as:

$$\beta_j(x_j^* + \rho[\cos\theta, \sin\theta]^T) = \rho^2 - r_j(\theta)^2 \tag{39}$$

The diffeomorphism $h : \Omega \to \Omega_{sph}$ can then be constructed as:

$$h(z) = \sigma_d(z)z + \sum_{j=1}^{b} \sigma_j(z)\left(x_j^* + \rho_j \frac{z - x_j^*}{\|z - x_j^*\|}\right) \tag{40}$$

where σ_j are smooth partition functions designed to transition between the identity map and the star-shaped deformation. These σ_j functions form a partition of unity, meaning they sum to 1 at each point in the workspace, i.e., $\sigma_d(z) + \sum_{j=1}^{b} \sigma_j(z) = 1$. The functions σ_j are defined as:

$$\sigma_j(z) = \eta_j(\beta_j(z)), \quad \eta_j(s) = \frac{\zeta(\varepsilon_j - s)}{\zeta(\varepsilon_j)} \tag{41}$$

where ζ is a bump function that smoothly transitions from 0 to positive values.

This construction creates a "stitching" function that smoothly interpolates between the identity map far from obstacles (where $\sigma_d = 1$)and the spherical deformation near obstacles (where some $\sigma_i = 1$). Each σ_j is active only within a buffer region around obstacle O_j (where $\beta_j(z) \leq \varepsilon_j$). The non-overlapping buffer regions ensure that transformations applied to different obstacles do not interfere with each other. This approach is well-explained in the PnP paper [1], which provides a detailed implementation for converting complex star-convex obstacles into spherical ones while preserving topological properties.

2. Proof of the WIP (based on [1])

PROOF. Suppose $\mathbf{x}(t)$ is a trajectory with $\mathbf{x}(0) \in C_\varrho(G)$ exiting $C_R(G)$. Let

$$t_1 = \inf\{t \in [0,\infty) : \mathbf{x}(t) \notin C_R(G)\} \tag{42}$$

$$t_0 = \sup\{t \in [0,t_1) : \mathbf{x}(t) \in C_\varrho(G)\} \tag{43}$$

First, $\mathbf{x}(t_0) \in C_\varrho(G)$ implies $V_G(t_0) \le |E|P(\varrho) < P(R)$. Next, for at least one edge $pq \in E$, we have $\|x_q(t_1) - x_p(t_1)\| = R$, hence $P(R) \le V_G(t_1)$ and therefore $V_G(t_0) < V_G(t_1)$.

However, by assumption, we have

$$V_G(t_1) - V_G(t_0) = \int_{t_0}^{t_1} \dot{V}_G(t)dt \le 0$$

which contradicts $V_G(t_0) < V_G(t_1)$. □