Decentralized Multi-Agent Controllers for Formation Control and Leader-Follower Coordination: A ROS2 Implementation

Varun Rayamajhi, *Student Member, IEEE,* and Dr. Patrick Martin, *Member, IEEE*

***Abstract*—This paper presents a practical implementation of decentralized multi-agent control systems for formation control and leader-follower coordination using the Robot Operating System 2 (ROS2) framework. We integrate well-established control laws from the multi-agent systems literature, including position-based and distance-based formation controllers, and leader-follower coordination. The implementation uses graph-theoretic methods to enable decentralized control, where each agent requires only local neighbor information to compute its control inputs. We apply the non-linear transformation to map single-integrator control inputs to unicycle dynamics, which enables deployment on differential-drive mobile robots. The complete software architecture is developed as a modular ROS2 package suite and is available at link if possible to share publicly. We provide a self-contained mathematical exposition of the underlying theory and validate the implementation through simulation experiments.**

***Index Terms*—Multi-agent systems, formation control, leader-follower, decentralized control, graph theory, ROS2, networked control systems**

I. Introduction

Add Introduction

A. Related Work

Add Related Work

B. Contributions

The contributions of this paper are:

1) Integration of formation control and leader-follower algorithms into a modular ROS2 software architecture.
2) A complete implementation (Can we say that?) that bridges the gap between theoretical control laws and practical deployment on mobile robot platforms .

C. Organization

The remainder of this paper is organized as follows. Section II introduces mathematical preliminaries and notation. Section III provides formal problem statements. Section IV develops the formation control laws. Section V extends the framework to leader-follower coordination. Section VI presents the nonlinear transformation to unicycle dynamics. Section VII describes the ROS2 implementation. Section VIII discusses the results. Section IX concludes the paper.

V. Rayamajhi and P. Martin are associated with the Department of Computer Science, University of Richmond, Richmond, VA 23173 USA (e-mail: varun.rayamajhi@richmond.edu; patrick.martin@richmond.edu).

II. Mathematical Preliminaries

A. Notation

We denote the set of real numbers by $\mathbb{R}$, the n-dimensional Euclidean space by $\mathbb{R}^n$, and the set of $m \times n$ real matrices by $\mathbb{R}^{m\times n}$. The Euclidean norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is denoted $\|\mathbf{x}\|$. The identity matrix of dimension n is $\mathbf{I}_n$, and $\mathbf{1}_n$ denotes the n-dimensional vector of ones.

B. Graph Theory

Let us consider a team of n agents indexed by the set $\mathcal{V} = \{1, 2, \ldots, n\}$. The communication network among agents is represented by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. Specifically, we represent the agents by an undirected complete graph $\mathcal{G} = K_n$.

Definition 1 (Adjacency Matrix)**.** *The adjacency matrix $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n\times n}$ of graph $\mathcal{G}$ is defined as:*

$$a_{ij} = \begin{cases} w_{ij} > 0 & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where w_{ij} represents the edge weight. For undirected graphs, $a_{ij} = a_{ji}$.

Definition 2 (Neighbor Set)**.** *The neighbor set of agent i is defined as:*

$$\mathcal{N}_i = \{j \in \mathcal{V} : (i,j) \in \mathcal{E}\} \tag{2}$$

In our implementation, agents only have access to the information from their neighbors $\mathcal{N}_i$. Moreover, in our implementation, $\mathcal{G} = K_n$, so we have $\mathcal{N}_i = \mathcal{V} \setminus \{i\}$. The adjacency matrix is predefined and determines which agents can communicate with each other.

C. Coordinate Transformations

We have two coordinate frames: a global map frame $\mathcal{F}_M$ and a local body frame $\mathcal{F}_B$ attached to each agent. A rigid body transformation from $\mathcal{F}_M$ to $\mathcal{F}_B$ is represented by a homogeneous transformation matrix:

$$\mathbf{T}_{MB} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3}$$

where $\mathbf{R}$ is the rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is the translation vector.

For planar motion, the rotation matrix is given by:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

The transformation of a point $\mathbf{p}_M$ from the map frame to the local frame is:

$$\begin{bmatrix} \mathbf{p}_B \\ 1 \end{bmatrix} = \mathbf{T}_{MB} \cdot \begin{bmatrix} \mathbf{p}_M \\ 1 \end{bmatrix} \tag{5}$$

The inverse transformation is given by:

$$\begin{bmatrix} \mathbf{p}_M \\ 1 \end{bmatrix} = \mathbf{T}_{MB}^{-1} \cdot \begin{bmatrix} \mathbf{p}_B \\ 1 \end{bmatrix} \tag{6}$$

where:

$$\mathbf{T}_{MB}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{7}$$

III. Problem Formulation

A. Agent Dynamics

Let us consider a team of n mobile agents operating in $\mathbb{R}^d$ (where $d = 2$). Each agent i has position $\mathbf{q}_i \in \mathbb{R}^d$ and evolves according to single-integrator dynamics:

$$\dot{\mathbf{q}}_i = \mathbf{u}_i, \quad i = 1, 2, \ldots, n \tag{8}$$

where $\mathbf{u}_i \in \mathbb{R}^d$ is the control input to be designed.

The collective position of the multi-agent system is:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_1^T & \mathbf{q}_2^T & \cdots & \mathbf{q}_n^T \end{bmatrix}^T \in \mathbb{R}^{nd} \tag{9}$$

B. Formation Specification

Definition 3 (Position-Based Formation)**.** *A position-based formation is specified by a set of desired relative position vectors $\{\mathbf{z}_i^*\}_{i=1}^n$, where $\mathbf{z}_i^* \in \mathbb{R}^d$ represents the desired position of agent i in the map frame $\mathcal{F}_M$. The formation is achieved when:*

$$(\mathbf{q}_i - \mathbf{q}_j) - (\mathbf{z}_i^* - \mathbf{z}_j^*) = \mathbf{0}, \quad \forall (i,j) \in \mathcal{E} \tag{10}$$

Definition 4 (Distance-Based Formation)**.** *A distance-based formation is specified by a set of desired inter-agent distances $\{d_{ij}^*\}_{(i,j)\in\mathcal{E}}$, where $d_{ij}^* > 0$ in the map frame $\mathcal{F}_M$ (not sure if this inter-agent distance is the same or different in map and base frame but in our implementation, we used the map frame). The formation is achieved when:*

$$\|\mathbf{q}_i - \mathbf{q}_j\| = d_{ij}^*, \quad \forall (i,j) \in \mathcal{E} \tag{11}$$

Assumption 1. *The communication graph $\mathcal{G}$ is undirected and connected.*

C. Problem Statements

Problem 1 (Formation Control): We implement the formation control laws $\mathbf{u}_i = f_i(\mathbf{q}_i, \{\mathbf{q}_j\}_{j\in\mathcal{N}_i})$ in a distributed manner such that all the agents asymptotically achieve the desired formation.

Problem 2 (Leader-Follower Coordination): Given the leader $\mathcal{L}$ and followers $\mathcal{V}_F = \mathcal{V} \setminus \{\mathcal{L}\}$, we implement leader-follower control laws such that:

1) Leader follows the predefined waypoint $\mathbf{r}_L$.
2) Followers maintain formation among each other while following the leader.

IV. Formation Control

In this section, we describe the control laws for multi-agent formation control used in our implementation.

A. Rendezvous Protocol

Before addressing formation control, we consider the fundamental rendezvous problem where all agents converge to a common point.

The basic rendezvous (consensus) control law is [1]:

$$\mathbf{u}_i = -\sum_{j\in\mathcal{N}_i} w_{ij}(\mathbf{q}_i - \mathbf{q}_j) \tag{12}$$

Under this control law, if the communication graph is connected, all agents asymptotically converge to the centroid of their initial positions.

B. Position-Based Formation Control

The position-based formation control law is add citation:

$$\mathbf{u}_i = -k_f \sum_{j\in\mathcal{N}_i} \left[(\mathbf{q}_i - \mathbf{q}_j) - (\mathbf{z}_i^* - \mathbf{z}_j^*)\right] \tag{13}$$

where $k_f > 0$ is the formation control gain. This control law drives agents to achieve the desired relative positions. Convergence is guaranteed when the communication graph is connected [3].

C. Formation Error

To analyze convergence in position-based formation control, we use the formation error coordinates defined as add citation:

$$\boldsymbol{\tau}_i = \mathbf{q}_i - \mathbf{z}_i^* \tag{14}$$

where $\boldsymbol{\tau}_i \in \mathbb{R}^d$ represents the difference in the achieved position of agent i in the formation from its desired position in the map frame.

In our planar implementation ($d = 2$), we decompose this into:

$$\tau_{x,i} = q_{x,i} - z_{x,i}^* \tag{15}$$

$$\tau_{y,i} = q_{y,i} - z_{y,i}^* \tag{16}$$

The importance of $\boldsymbol{\tau}_i$ is that when all agents converge to the desired formation, they share a common displaced coordinate, i.e., $\boldsymbol{\tau}_i = \boldsymbol{\tau}_j$ for all $i, j \in \mathcal{V}$. This follows because the position-based control law (13) can be rewritten in terms of $\boldsymbol{\tau}_i$ as

$$\begin{aligned} \mathbf{u}_i &= -k_f \sum_{j\in\mathcal{N}_i} \left[(\mathbf{q}_i - \mathbf{q}_j) - (\mathbf{z}_i^* - \mathbf{z}_j^*)\right] \\ &= -k_f \sum_{j\in\mathcal{N}_i} \left[(\mathbf{q}_i - \mathbf{z}_i^*) - (\mathbf{q}_j - \mathbf{z}_j^*)\right] \\ &= -k_f \sum_{j\in\mathcal{N}_i} (\boldsymbol{\tau}_i - \boldsymbol{\tau}_j) \end{aligned} \tag{17}$$

This is a consensus protocol, so all $\boldsymbol{\tau}_i$ to converge to a common value.

In our implementation, we record $\tau_{x,i}(t)$ and $\tau_{y,i}(t)$ for each agent over time. This provides us a clear visualization of formation convergence:

- **Before convergence**: We expect each agent to have distinct τ_i values, and the curves for τ_i are spread apart for all agents.
- **At convergence**: We expect all τ_i values to converge to the same value. This indicates that the desired relative positions have been achieved.

This visualization is particularly useful for debugging and validating the formation controller.

D. Distance-Based Formation Control

For distance-based formations, agents use only inter-agent distance information rather than absolute positions [3].

Let the distance error between agents i and j be defined as:

$$e_{ij} = \|\mathbf{q}_i - \mathbf{q}_j\|^2 - (d_{ij}^*)^2 \tag{18}$$

The distance-based formation control law is add citation:

$$\mathbf{u}_i = k_f \sum_{j \in \mathcal{N}_i} e_{ij}(\mathbf{q}_j - \mathbf{q}_i) \tag{19}$$

where $k_f > 0$ is the formation control gain. Each agent moves toward neighbors that are too far away and away from neighbors that are too close.

V. Leader-Follower Coordination

In this section, we describe our implementation of the leader-follower architecture, which combines distance-based formation maintenance with waypoint tracking for the leader agent.

A. Leader-Follower Architecture

Let us divide the agent set into the leader $\mathcal{V}_L = \{L\}$ and followers $\mathcal{V}_F = \{1, \ldots, n\}$. Leader agent has access to waypoints.

Assumption 2. *Each follower agent has a path to the leader in the communication graph $\mathcal{G}$.*

B. Leader Control Law

The leader agent follows the following law that combines formation maintenance with waypoint tracking:

$$\mathbf{u}_i^L = k_f \sum_{j \in \mathcal{N}_i} e_{ij}(\mathbf{q}_j - \mathbf{q}_i) + k_L(\mathbf{r}_i - \mathbf{q}_i) \tag{20}$$

where:

- $k_f > 0$ is the formation control gain
- $k_L > 0$ is the leader tracking gain
- $\mathbf{r}_i \in \mathbb{R}^d$ is the current target waypoint for leader i

The first term maintains formation with neighboring agents, while the second term drives the leader toward its waypoint. The waypoint is updated when the leader reaches a proximity threshold $\epsilon > 0$:

$$\|\mathbf{q}_i - \mathbf{r}_i\| < \epsilon \implies \mathbf{r}_i \leftarrow \mathbf{r}_i^{\text{next}} \tag{21}$$

C. Follower Control Law

Follower agents maintain formation using the distance-based control law:

$$\mathbf{u}_i^F = k_f \sum_{j \in \mathcal{N}_i} e_{ij}(\mathbf{q}_j - \mathbf{q}_i) \tag{22}$$

As the leader moves toward waypoints, follower agents continuously adjust their positions to maintain the desired inter-agent distances while following the leader agent.

VI. Nonlinear Transformation to Unicycle Dynamics

The control laws defined in the preceding sections assume single-integrator dynamics. To deploy these controllers on physical robots in our lab, we use the non-linear transformation technique [4] to transform the control inputs to unicycle dynamics.

A. Unicycle Model

Practical mobile robots, such as differential-drive robots, are subject to nonholonomic constraints and are better modeled as unicycles:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{23}$$

where (x, y) is the robot position, θ is the orientation, v is the linear velocity, and ω is the angular velocity.

B. Non-linear Transformation

To apply single-integrator control laws to unicycle robots, we use the non-linear transformation. [4].

Let us consider a reference point P displaced from the robot center by distance x_r along the heading direction:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} x + x_r\cos\theta \\ y + x_r\sin\theta \end{bmatrix} \tag{24}$$

The velocity of point P is:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -x_r\sin\theta \\ \sin\theta & x_r\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{25}$$

C. Inverse Transformation

Given a desired velocity (u_x, u_y) from the single-integrator control law, the required unicycle inputs are obtained by inverting the kinematic relationship in (25) [4]:

$$v = \cos\theta \cdot u_x + \sin\theta \cdot u_y \tag{26}$$

$$\omega = \frac{1}{x_r}(-\sin\theta \cdot u_x + \cos\theta \cdot u_y) \tag{27}$$

This transformation is valid for $x_r > 0$ and maps the Cartesian velocity commands to the linear and angular velocities of the unicycle.

Remark 1. *The parameter x_r should be chosen small but non-zero. In our implementation, $x_r = 0.1$ meters provides a good balance between responsiveness and stability.*

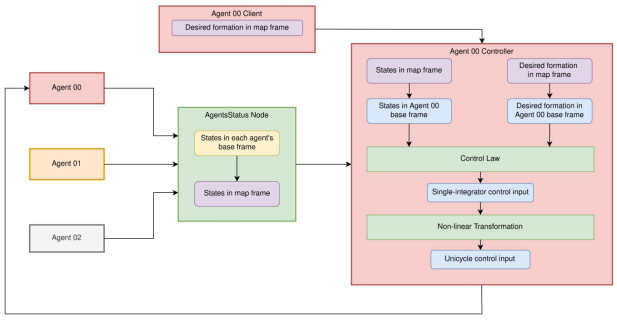

Fig. 1. Decentralized controller architecture. Each agent runs its own controller node that receives neighbor states from the AgentsStatus node and desired formation from a client. The controller transforms all information to the agent's local frame, applies the control law to compute single-integrator inputs, and then applies the nonlinear transformation to produce unicycle commands.

D. Velocity Saturation

Actuator limitations in real robots require us to have the following velocity bounds in our implementation:

$$|v| \leq v_{\max} \tag{28}$$
$$|\omega| \leq \omega_{\max} \tag{29}$$

The saturated control inputs are:

$$v_{\text{sat}} = \text{sat}(v, v_{\max}) = \max(-v_{\max}, \min(v, v_{\max})) \tag{30}$$
$$\omega_{\text{sat}} = \text{sat}(\omega, \omega_{\max}) = \max(-\omega_{\max}, \min(\omega, \omega_{\max})) \tag{31}$$

VII. ROS2 Implementation

This section describes the ROS2 implementation in detail, as this constitutes the primary contribution of our work. The implementation is designed to be modular, extensible, and suitable for both simulation and hardware deployment.

A. System Architecture Overview

The framework is implemented using ROS2 (Robot Operating System 2), which provides a distributed middleware architecture well-suited for multi-agent systems [5].

B. Decentralized Controller Architecture

The key contribution of this work is the implementation of truly decentralized controllers where each agent computes its own control input using only local information. Figure 1 illustrates the architecture for a system with three agents (Agents00-02), showing the information flow for Agent 00's controller.

The decentralized architecture consists of the following components:

Agent Nodes: Each physical robot runs an agent node that publishes its current state (position and orientation) to its own odometry topic. In the figure, Agent 00, Agent 01, and Agent 02 each publish their states in their respective base frames.

AgentsStatus Node: This node subscribes to the odometry topics of all agents in the network. It serves two purposes:

1) Collects states from each agent's base frame
2) Transforms all states to the map frame and publishes them on a shared `/agents_status` topic

Even though we define a node that collects information from all agents, the architecture remains decentralized because each agent's controller uses only the local information based on the communication graph $\mathcal{G}$ and independently computes the control input.

Client Node: For each agent, a client node sends the desired formation specification (e.g., desired positions $\mathbf{z}_i^*$ or desired distances d_{ij}^*) to the agent's controller. The formation parameters are specified in the map frame.

Controller Node: Each agent runs its own controller node that:

1) Receives states of all agents in the map frame from `/agents_status`
2) Receives desired formation specification from the respective client
3) Transforms neighbor states from the map frame to the agent's local base frame using TF2
4) Transforms desired formation positions to the agent's local base frame
5) Applies the appropriate control law ((13) or (19)) to compute single-integrator control input $\mathbf{u}_i = (u_x, u_y)$
6) Applies the nonlinear transformation ((26)-(27)) to convert to unicycle control input (v, ω)
7) Publishes the velocity command back to the agent

We emphasize that each controller node operates independently and only uses information about its neighbors $\mathcal{N}_i$. Even though the AgentsStatus node broadcasts all agent states, each controller uses only the information about its neighbors $\mathcal{N}_i$ according to the predefined communication graph. This is significant for our multi-agent system because it ensures the following:

- No single agent has decision-making authority over the entire system
- Each agent can join or leave the network without affecting the other controllers
- The computational load is distributed across all agents
- Failure of one agent's controller does not directly affect other agents' computations

C. ROS2 Package Structure

I don't know if the following makes any sense to write: I'm basically writing a brief description of each function. The complete system is organized into the following ROS2 packages:

1) networked_controllers: This package contains the core control algorithms implemented as Python functions:

- `rendezvous()`: Implements the consensus control law (12)
- `position_based_formation()`: Implements the position-based controller (13)
- `distance_based_formation()`: Implements the distance-based controller (19)
- `leader_controller()`: Implements the leader control law (20)

Each function takes the agent's current position, neighbor positions, and formation parameters as inputs, and returns the control input $\mathbf{u}_i \in \mathbb{R}^d$.

2) graph_theoretic_methods: This package provides utilities for graph construction and neighbor management:

- `compute_adj_mat()`: Computes the adjacency matrix from agent positions
- `get_nbr_dists()`: Extracts inter-agent distances from the adjacency matrix
- `get_nbrs()`: Returns the neighbor set $\mathcal{N}_i$ for each agent

The adjacency matrix is represented as an $n \times n$ NumPy array where entry (i, j) stores the distance $\|\mathbf{q}_i - \mathbf{q}_j\|$ if agents i and j are neighbors, and `None` otherwise.

3) transformations: This package handles coordinate frame transformations and the nonlinear mapping to unicycle dynamics:

- `get_transformation_matrix()`: Constructs the 4×4 homogeneous transformation matrix from a ROS2 `Transform` message
- `transform_to_local()`: Transforms coordinates from the global map frame to the robot's local frame
- `transform_to_map()`: Transforms coordinates from local to map frame
- `non_linear_transform()`: Implements equations (26)-(27) to convert Cartesian velocities to unicycle commands

4) agents: This package implements simulated agent nodes that model unicycle dynamics:

- Subscribes to velocity commands (`geometry_msgs/Twist`)
- Integrates unicycle kinematics (23) with configurable time step
- Publishes odometry (`nav_msgs/Odometry`)
- Supports process noise for realistic simulation

5) formation_controller: This package implements formation control as ROS2 action servers:

- `FormationController`: An action server that executes formation control tasks
- Supports both position-based and distance-based control laws (specified via parameters)
- Monitors formation achievement using a distance tolerance threshold
- Provides real-time feedback including current positions, inter-agent distances, and $\boldsymbol{\tau}$ values

6) leader_follower: This package implements leader-follower coordination:

- `LeaderFollower`: An action server for leader-follower tasks
- Leader nodes track sequences of waypoints
- Follower nodes maintain formation using distance-based control

7) multi_agent_launch: This package contains ROS2 launch files for starting multi-agent simulations:

- `agents_launch.py`: Launches n agent nodes with configurable initial positions
- `fc_tasks_launch.py`: Launches formation controller action servers
- `fc_clients_launch.py`: Launches action clients to send formation goals
- `lf_tasks_launch.py`: Launches leader-follower action servers
- `lf_clients_launch.py`: Launches leader-follower action clients

D. Communication Architecture

The inter-agent communication follows a publisher-subscriber pattern:

Agent State Broadcasting: Each agent publishes its odometry to a namespaced topic:

`/hivebot_<id>/odom`

An `AgentsStatus` node collects all agent states and broadcasts them on to a shared topic:

`/agents_status`

This architecture allows each agent to receive neighbor information by subscribing to the single `/agents_status` topic and filtering for its neighbors based on the communication graph.

Velocity Commands: Each agent subscribes to velocity commands on:

`/hivebot_<id>/commands/velocity`

E. Coordinate Frame Management

Our implementation uses the ROS2 TF2 library for coordinate frame management. We use two primary frames:

- `map`: The global inertial reference frame
- `hivebot_<id>/odom`: The local odometry frame for each agent

Each agent's controller computes the control inputs in the local frame to account for each agent's orientation. The transformation pipeline is:

1) Receive neighbor positions from `/agents_status` (in map frame)
2) Look up transform from map to local frame using TF2
3) Transform neighbor positions to local frame
4) Compute control law to obtain (u_x, u_y) in local frame
5) Apply nonlinear transformation to obtain (v, ω)
6) Publish velocity command

F. Action Server Architecture

Formation control and leader-follower tasks are implemented as ROS2 action servers, providing a robust interface for task execution:

Goal Specification: Clients send goals containing:

- Agent identifier and neighbor list
- Desired inter-agent distances
- For position-based control: desired positions $\mathbf{z}_i^*$ for each agent

- For leader-follower: waypoint sequence and proximity threshold

Goal Validation: The action server validates goals before acceptance:

- Checks that all required parameters are provided
- Verifies that desired distances are positive and above minimum threshold
- Ensures graph connectivity (all specified neighbors exist)

Execution Loop: Upon goal acceptance, the action server enters a control loop at 10 Hz where it does the following:

1) Read current agent and neighbor positions
2) Compute control input using the appropriate control law
3) Apply nonlinear transformation and velocity saturation
4) Publish velocity command
5) Check formation achievement condition
6) Publish feedback message

Feedback: During execution, the server publishes feedback containing:

- Current agent position in map frame
- Current distances to each neighbor
- Desired distances to each neighbor
- For position-based control: current $\boldsymbol{\tau}_i$ values

Result: Upon completion (or cancellation), the server returns:

- Success/failure status
- Final achieved configuration
- Achieved inter-agent distances

G. Custom Message Types

The implementation defines custom ROS2 message types in the `multi_agent_task_mgmt` package:

Agent.msg: Contains agent identifier and odometry message.

AgentUpdate.msg: Contains an array of `Agent` messages for broadcasting the state of all agents in the network.

VIII. Simulation Results

We validate our implementation through simulation experiments demonstrating both formation control and leader-follower coordination. All experiments were conducted in ROS2 with simulated unicycle agents.

A. Formation Control with Five Agents

In this experiment, five agents (agent00 through agent04) are tasked with achieving a regular pentagon formation using the position-based formation controller. The communication graph is fully connected. This means each agent can communicate with all other agents ($\mathcal{N}_i = \mathcal{V} \setminus \{i\}$ for all i). Figure 2 shows the results.

The top-left panel shows the K_5 graph, which represents the communication network. The top-right panel shows the paths taken by each agent from their initial positions to the final desired formation. Agents start at scattered positions and smoothly converge to their target positions. The trajectories demonstrate coordinated motion. The second row shows the linear velocity v and angular velocity ω commands sent to each agent over time. Initially, velocities are high as agents move toward the formation. As agents approach their target positions, velocities decrease and eventually settle near zero when the formation is achieved. The velocity saturation at $v_{max} = 0.4$ m/s and $\omega_{max} = 1.57$ rad/s is also respected by all agents. The third row shows the key validation metric for formation control success: the convergence of τ_x and τ_y for all agents. We recall that $\boldsymbol{\tau}_i = \mathbf{q}_i - \mathbf{z}_i^*$ represents the deviation from the desired formation position. When the formation is achieved, all $\boldsymbol{\tau}_i$ values should converge to the same point. The plots clearly show all five agents' τ_x values converging to approximately the same value. This also holds true for τ_y. It confirms that the position-based formation controller successfully achieves the desired formation. The bottom-left panel shows the evolution of inter-agent distances over time. Dashed lines represent current (measured) distances, while solid lines represent desired distances. We see that distance pairs converge from their initial values to the desired formation distances, with convergence achieved by approximately $t = 40$ seconds. The bottom-right panel shows the desired formation (blue) with the achieved formation (red). The close alignment demonstrates that the formation controller successfully drives the agents to the target configuration.

Maybe a sentence or two about real robots since I had run this experiment with the three TurtleBots.

B. Leader-Follower with Four Agents

In this experiment, four agents perform leader-follower coordination where agent00 (blue) acts as the leader tracking a sequence of waypoints, while agents 01, 02, and 03 act as followers maintaining a triangular formation. Figure 3 shows the results.

Communication Graph: The left panel shows the leader-follower graph topology. Unlike the fully connected formation control case, this graph has a specific structure:

- Agent00 (blue) is the leader with a connection to agent01
- Agent01 (orange) serves as a bridge, connected to the leader and to agents 02 and 03
- Agents 02 (green) and 03 (red) are connected to agent01 and to each other

This topology demonstrates that the decentralized controller works with sparse, non-fully-connected graphs.

Inter-Agent Distances: The top-right panel shows how inter-agent distances evolve over approximately 100 seconds of operation. Dashed lines show current distances while solid lines show desired distances. We observe that distances initially start far from desired values (e.g., curr01_02 starts above 6 meters). Within the first 20 seconds, all distances converge close to their desired values. We also observe that small deviations occur from the desired values. This happens when the leader continuously moves towards waypoints.

Agent Trajectories: The bottom-right panel shows the paths of all four agents as the leader navigates through a sequence of waypoints (marked as black triangles). The leader (agent00, blue) follows a path through the waypoints while the followers (agents 01, 02, 03) maintain their relative positions.

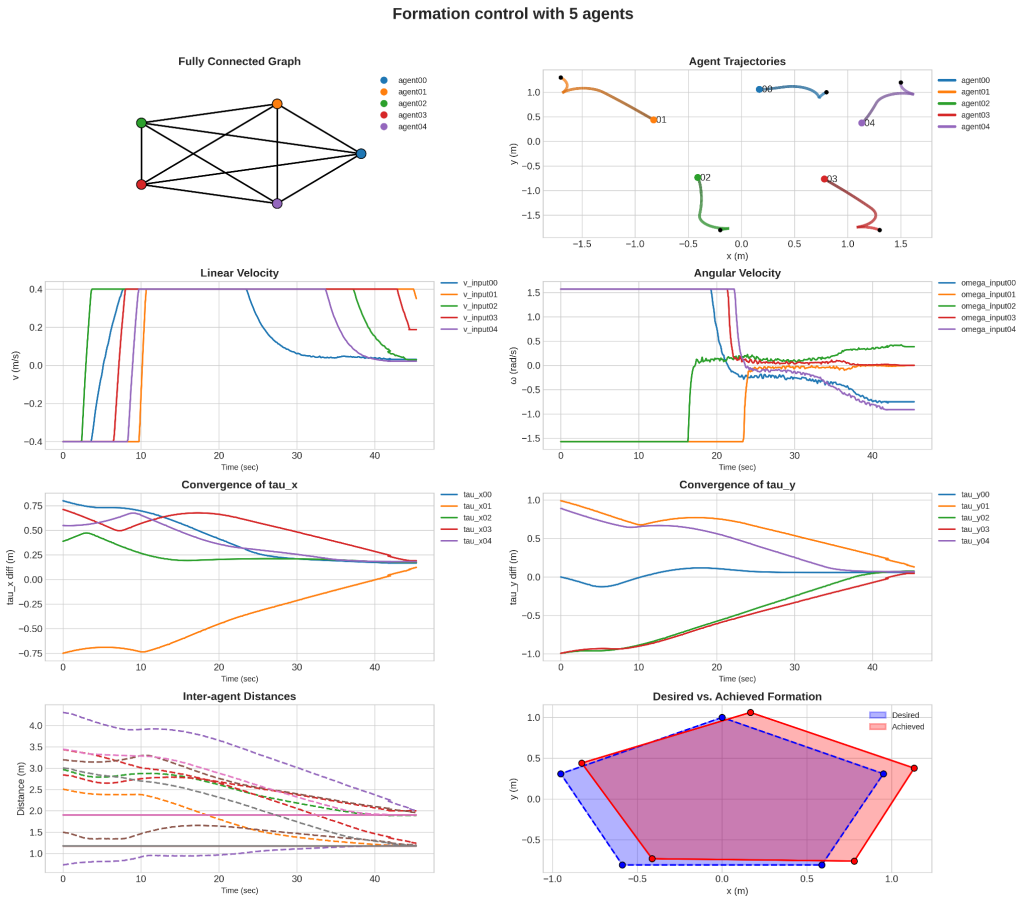

Fig. 2. Formation control results with 5 agents achieving a pentagon formation. Top row: communication graph (fully connected) and agent trajectories showing convergence from initial positions to final formation. Second row: linear and angular velocity profiles over time. Third row: convergence of displaced coordinates τ_x and τ_y, demonstrating that all agents converge to the same $\boldsymbol{\tau}$ value. Bottom row: inter-agent distances (dashed: current, solid: desired) converging to target values, and comparison of desired vs. achieved formation shape.

We observe a coordinated movement: all agents follow similar curved paths while preserving the formation. The waypoints guide the leader through approximately 20 meters of horizontal travel.

C. Discussion

The simulation results validate several key aspects of our implementation of decentralized control system:

1) **Convergence**: Both position-based formation control and leader-follower coordination successfully converge to the desired configurations. The $\boldsymbol{\tau}$ convergence plots (I should have plotted the τ convergence plots for the leader-follower case too) demonstrate that formation is achieved.
2) **Decentralized Control**: Each agent computes its control input independently based on local neighbor information. The leader-follower experiment demonstrates operation with a sparse communication graph.
3) **Formation Maintenance**: In the leader-follower case, followers maintain formation while the entire group moves through space. This demonstrates the controller's ability to handle moving formations.
4) **Velocity Saturation**: The velocity profiles show that saturation limits are respected by each agent throughout the operation.

IX. Conclusion

This paper presented a practical ROS2 implementation of decentralized multi-agent formation control and leader-follower coordination. We focused on integrating well-established control laws from the multi-agent systems literature into a working robotic software framework. ADD MORE Future work will address:

- Extension to dynamic communication topologies
- Integration of collision avoidance using control barrier functions
- Hardware deployment and experimental validation on physical robots

References

[1] 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.
[2] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton, NJ: Princeton University Press, 2010.

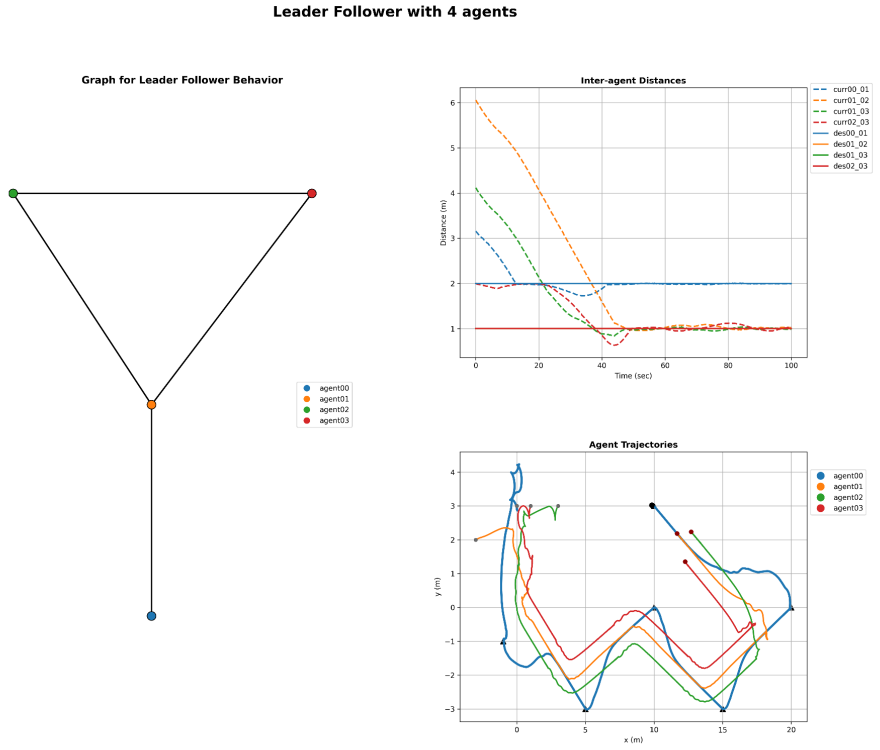

Fig. 3. Leader-follower results with 4 agents. Left: communication graph showing the leader-follower topology: agent00 is the leader connected to agent01, which is connected to followers agent02 and agent03. Top-right: inter-agent distances over time showing convergence to desired values while the formation moves. Bottom-right: agent trajectories as the leader navigates through waypoints (black triangles) while followers maintain formation.

[3] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[4] G. Oriolo, A. De Luca, and M. Vendittelli, "WMR control via dynamic feedback linearization: Design, implementation, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, 2002.

[5] 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, 2022.