Hoplite: Coordinated LIMOBot Swarm Control via DS5 and OptiTrack MoCap with ROS Integration

Team member: Dakota Winslow, Kanghyun Lee

Github repo: [dakotawinslow/hoplite: System for moving AgileX Limo robots in formation]

Video:

Abstract

This project presents a coordinated multi-robot system using three AgileX LIMO robots controlled via a wireless controller and localized with an OptiTrack motion capture system. To address the mismatch between the ROS1-based LIMOBots and the ROS2-based MoCAP localization data, we implemented a ROS1–ROS2 bridge that converts real-time position data into ROS1-compatible topics. A leader-follower control architecture enables the leader LIMOBot to receive joystick input, while the MoCAP computer publishes target position and orientation topics for both the leader and follower bots. This allows the entire swarm to move in a fixed polygonal formation, maintaining their relative positions and heading while traveling from point A to point B. The swarm is not capable of rotation but can perform coordinated, Ackermann-style translation. We successfully demonstrated real-time leader control, verified accurate MoCAP localization streaming, and created a simulation environment to test swarm behavior. This system lays the groundwork for a fully autonomous formation control framework for small mobile robot teams.

Introduction

As multi-robot systems become increasingly common in autonomous logistics, warehouse automation, and swarm robotics, efficient coordination and communication among robots is essential. However, maintaining formation control across multiple robots in real-time poses significant challenges, especially when the localization system and control logic operate across different versions of the Robot Operating System (ROS).

This project explores the implementation of a small-scale mobile robot swarm using three AgileX LIMO robots. These robots rely on external localization via an OptiTrack motion capture system, which streams robot positions through the Motive software as ROS2 topics. Since the LIMOBots

operate on ROS1, a ROS1–ROS2 bridge is required to translate positional data, allowing robots to use MoCAP-based feedback for control.

Our approach uses a leader–follower (master–slave) control architecture, where the leader LIMOBot receives DS5 joystick commands, determines the formation center, and computes target positions for follower LIMOBots. The system is designed to maintain fixed formation spacing and simulate coordinated motion, even in the presence of localization delays and communication overhead.

This project builds on concepts from embedded systems, real-time control, and distributed robotics, applying ROS frameworks and motion capture integration in a resource-constrained environment. Our implementation provides a foundation for future extensions such as dynamic formations, autonomous leader behavior, and sensor-based localization fallback systems.

System Architecture and Control Logic

1. High-Level System Overview

The user controls the swarm through a wireless gamepad connected by USB to any one of the robots in the swarm. Joystick events are communicated to a control node that integrates the joystick movements into a pose, which is processed by the hoplite leader node to create target poses for each bot in the swarm. Each individual robot then navigates itself to its target pose while receiving real-time feedback about every robot in the swarm from the motion capture system.

2. Joystick Interface for Leader Control

The leader node is controlled by a generic USB gamepad, which can be connected to any USB and ROS1 capable computer on the network (we connect it to a robot for simplicity). Joystick and button events are captured by the linux xpad driver and exposed as a /dev/input/js* device. The device is read by ROS's off-the-shelf joy node, which emits /joy messages detailing the status of each button and joystick every time the driver reports a change.

```
if __name__ == '__main__':
  try:
    controller = control_point()
    # rospy.spin()
    while not rospy.is_shutdown():
      controller.update_control_point()
      controller.publish_marker()
      controller.publish_twist()
      rospy.sleep(1.0 / FREQUENCY)
    # direct_control = rospy.get_param("~direct_control")
  except rospy.ROSInterruptException:
    pass
```

These /joy messages are subscribed to by our joystick_control node, which contains an internal posed model. Each time a /joy message is received, the model's pose is updated and published as a Marker() topic, /joystick_controller/marker. We also publish a Twist() message that is a direct pass-through of the joystick input for debugging (/cmd_vel).

While we read /joy messages as fast as they can come in, we limit traffic on the network by only publishing makers every 0.1 seconds. This is plenty responsive enough for humans, but is quite infrequent for the network, leaving plenty of room for the more important real-time position updates.

3. Squad Leader Command

The squad leader node can run on any computer in the ROS network (for our demo, we used a VM running Ubuntu 18.04 and ROS Melodic). The node reads in the markers from the joystick controller, extracts the pose from within, and calculates the target poses for each individual member of the swarm. Our only currently functional positioning mode, "regular_polygon_ formation()", defines a rigid spacing between robots and places n of them around the central control point such that the spacing between neighbors is fixed, creating regular polygons. Each robot's facing is set to match the input pose.

From these calculations, the leader publishes the poses under the topics /hoplite*/_pose (one for each member of the swarm).

4. Individual Hoplite Software

Each robot runs its own stack of control nodes, consisting of a hoplite_soldier, a limo_driver, and a mocap_cleaner.

hoplite_soldier

The hoplite_soldier node is responsible for reading target _pose messages from the leader and calculating a path to match that pose. It does so using a PD controller to set velocities for the limo_driver node, which controls the underlying mecanum motion platform. A PD controller was used as the actual kinematic system used in the LIMO platform is somewhat obscured, making model-based control difficult and unnecessary. No integral term is required as the motion capture system provides incredibly stable positional feedback with no appreciable error or drift (<1mm).

Within the hoplite_soldier node is a repulsion-force-based collision avoidance system. After the PD controller calculates a velocity toward the target (the 'desired velocity'), the collision avoidance subroutine is called to take into account the positions of other robots. Any robot nearby will apply a deflection force to the desired velocity, changing its direction. The deflection force decays with distance, so nearer robots deflect by greater magnitudes. This results in a sort of virtual magnetism, where robots 'bounce' off each other without touching.

limo_driver

The final corrected velocities are published as Twist messages under the /hoplite*/cmd_vel topic for the limo_driver to read. The limo_driver node is a bridge between ROS and the LIMO's motion platform, which is connected to the linux system via a serial link. This node was provided as part of the LIMO's default software package and only modified very slightly to read messages from our custom topic.

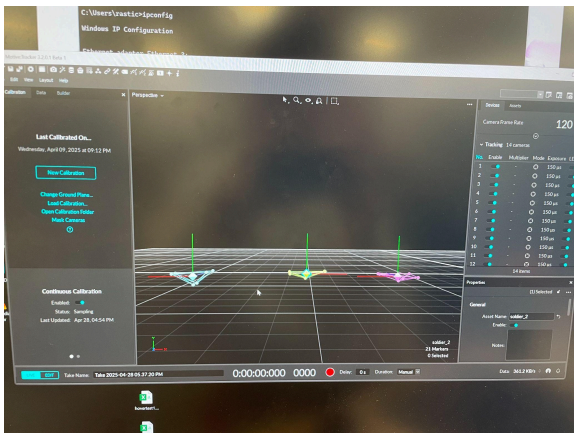
mocap_cleaner

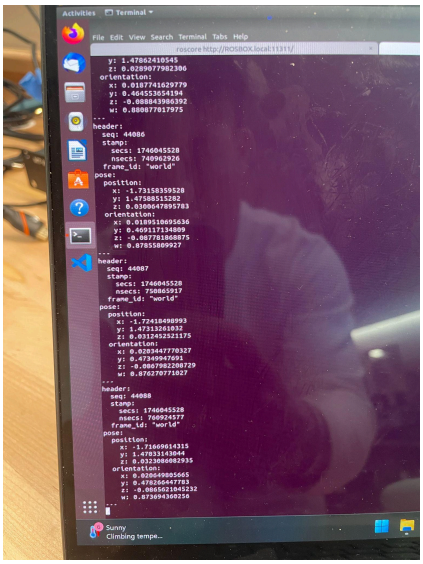
The mocap_cleaner node is a simple translator to transform the XZY coordinate messages from the motion capture system into XY poses for the robots, which generally only can think in 2D. Messages from the motion capture system may include movements or rotations in the 'up' axis, but we discard those for this system since the robots cannot actuate in those axes.

5. Localization via OptiTrack Motion Capture

We use an OptiTrack Motion Capture system to provide each robot with localization data about itself and its peers.Optitrack uses infrared cameras to track individual retroreflective markers. Markers in known configurations are tracked as rigid bodies. Poses for each of these detected rigid bodies are published to a VRPN server, which interfaces with a ROS2 node that converts the server updates into ROS2 pose messages. The ROS2 messages are then bridged to our ROS1 network using a using a special bridge node that can connect to both networks. This bridge node (as well as the ROS2 VRPN node) runs on a Raspberry Pi located in RASTIC as a part of the motion capture infrastructure there.

The motion capture system captures and reports position data at 120hz with positional precision less than 1 mm, making it more than capable for our system. While we explored data cleaning via Kalman filter or rolling average, neither was found to be necessary for this system, and we safely use the motion capture localization as ground truth for all positional calculations.





Results

The experimental setup consisted of three AgileX LIMOBots operating in a triangle formation within a motion capture (MoCap) space. Each robot was equipped with reflective markers tracked by OptiTrack cameras, with their pose data streamed via Motive software and published to ROS2 topics. These were bridged into ROS1 and consumed by each robot's control node. The leader LIMOBot was manually driven using a wired Xbox joystick connected to the control laptop. The system used real-time pose updates to maintain formation, with each follower computing its target position relative to the leader and adjusting its velocity accordingly using a 2D kinematic controller.

A PD control strategy was incorporated to reduce oscillations in the followers' motion, resulting in smoother convergence toward target positions. While no formal quantitative metrics (e.g., formation error or latency benchmarks) were recorded, qualitative results showed that the robots maintained a stable triangle formation during movement. The velocity commands and pose updates were visualized in RViz for real-time monitoring. Although occasional minor drift was observed during sharp turns, the system successfully demonstrated real-time decentralized coordination with minimal delay. Future experiments could include logging ground truth vs. estimated poses to quantitatively evaluate formation stability and control precision.