Swarm Robots: Circle Formation Around Dynamic Target Using ROS 2 — ROB-GY 6333 Networked Robotics Systems, Cooperative Control and Swarming — Final Term Project Report — Spring 2024

Vineela Reddy Pippera Badguna[1]

[1]New York University, Tandon School of Engineering, Department of Mechanical and Aerospace Engineering (MAE)

GitHub Repository of the Project: click here

Abstract

Collective behaviors observed in nature have sparked interest in the development of aerial swarm robotics, drawing inspiration from the coordinated movements observed in honeybee colonies, fish schools, and bird flocks. These robotic swarms, modeled after natural principles, have found diverse applications ranging from surveillance and agriculture to military operations. By emulating natural rules, they provide cost-effective solutions for tasks such as targeting, surveillance in disaster zones, and the establishment of emergency communication networks. This project presents a multi-robot system equipped with a formation control algorithm similar to Reynold's[1] designed to encircle a target point, particularly suitable for search and rescue operations. Through simulation in ROS 2[2] & Python, the effectiveness of the multi-robot system in forming a circular arrangement around the target point is demonstrated, showcasing its potential utility in real-world scenarios.

Introduction & Project Motivation

In recent years, robotics has increasingly turned to nature for inspiration, observing the coordinated behaviors found in honeybee colonies, fish schools, and bird flocks. Spanning across domains such as surveillance, agriculture, and military operations, swarm robotics presents promising solutions through its utilization of collective intelligence and decentralized control mechanisms.

This project is centered around the application of swarm robotics in the context of search and rescue operations, a critical area demanding prompt and efficient responses. Utilizing a multi-robot system equipped with a sophisticated formation control algorithm, our objective is to adeptly encircle target points. Leveraging ROS 2, we developed a UAV simulator node that subscribes to updated UAV and target positions, integrating sensor information for enhanced situational awareness. Through Python simulations, we assess the system's performance, highlighting its potential in real-world scenarios and contributing to the ongoing advancements in swarm robotics for mission-critical applications.

Algorithm

Algorithm 1: Circle Formation & Simulation of UAV Swarm

Initialize num_steps, num_uavs, $threshold$, and max_speed
Initialize $target\_pos$ to $(5.0, 5.0)$ and randomly generate $initial\_uav\_positions$ within a 10×10 grid
procedure UPDATE_TARGET_POSITION($target\_pos$, max_speed)
    $target\_pos \leftarrow target\_pos +$ random_uniform$(-max_speed, max_speed, size = 2)$
    return $target\_pos$
end procedure
procedure ALGOREYNOLDS($uav\_positions$, $target\_pos$, $threshold$, max_speed)
    for i in $range(len(uav\_positions))$ **do**
        $direction \leftarrow target\_pos - uav\_positions[i]$
        $distance \leftarrow \|direction\|$
        if $distance < threshold$ **then**
            $uav\_positions[i] \leftarrow uav\_positions[i] - max\_speed \cdot \frac{direction}{distance}$
        else
            $uav\_positions[i] \leftarrow uav\_positions[i] + max\_speed \cdot \frac{direction}{distance}$
        end if
    end for
    return $uav\_positions$
end procedure
procedure UPDATE($frame$)
    $target\_pos \leftarrow$ update_target_position$(target_pos, max_speed)$
    $initial\_uav\_positions \leftarrow$
    $initial\_uav\_positions \leftarrow$ algoReynolds$(initial_uav_positions, target_pos, threshold, max_speed)$
    print "Initial UAV Positions:", $initial\_uav\_positions$
    print "Target Position:", $target\_pos$
    Update the plot with the new target and UAV positions
end procedure

This algorithm is for circle formation of UAVs around dynamic target and for simulation.

System Design

As workflow shown in Figure 1, incorporating infrared (IR) sensors, each robot in the swarm is equipped to gauge its proximity to the target position. Upon detection, if the distance falls below the pre-defined radius, the robot initiates movement away from the target, ensuring safe distance maintenance. Conversely, if the distance surpasses the designated radius, the robot adjusts its trajectory towards the target, facilitating the formation of a cohesive circle.

Moreover, the updated positions of both the target and the UAVs are seamlessly relayed to the UAV simulator node. This integration of real-time data enables comprehensive simulation, allowing for the evaluation of swarm behavior and performance under various scenarios. By leveraging sensor-derived information and simulation capabilities, our approach ensures robustness and adaptability, key factors in the success of swarm robotics applications.

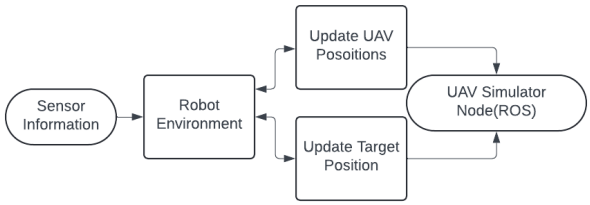

Figure 1: System Diagram

Software Implementation

Operating system: Ubuntu 20.04 LTS Focal [VM]
Software: ROS 2 Foxy for Ubuntu (Debian) amd64, arm64

Set locale

```
locale  # check for UTF-8
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale  # verify settings
```

Setup sources

```
sudo apt install software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Install ROS 2 packages

```
sudo apt update
sudo apt upgrade
sudo apt install ros-foxy-desktop python3-argcomplete
sudo apt install ros-foxy-ros-base python3-argcomplete
sudo apt install ros-dev-tools
```

Environment setup

```
source /opt/ros/foxy/setup.bash
```

Git clone project workspace

```
git clone https://github.com/vineelarpb/Swarm-Robots-Circle-Formation-Around-Dynamic-Target-Using-ROS-2.git
cd ~/ros2_ws/
```

Resolve dependencies

```
rosdep install -i --from-path src --rosdistro foxy -y
#Your console should return - "All required rosdeps installed successfully"
```

Build the workspace with colcon

```
colcon build
# If you want to build a specific package use,
# colcon build --packages-select [package name]
source install/setup.bash
```

Project package directory structure

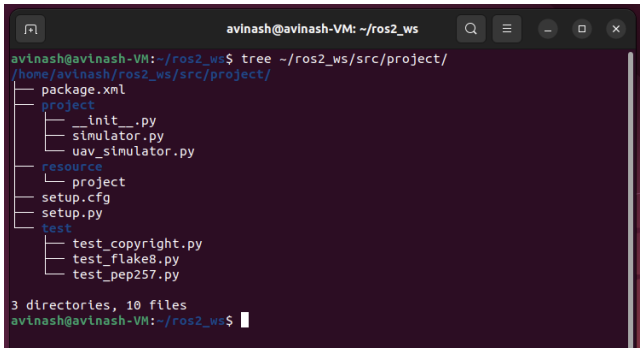

Figure 2: Package directory structure

ROS 2 Runtime & Simulation Outputs

Project — uav_simulator

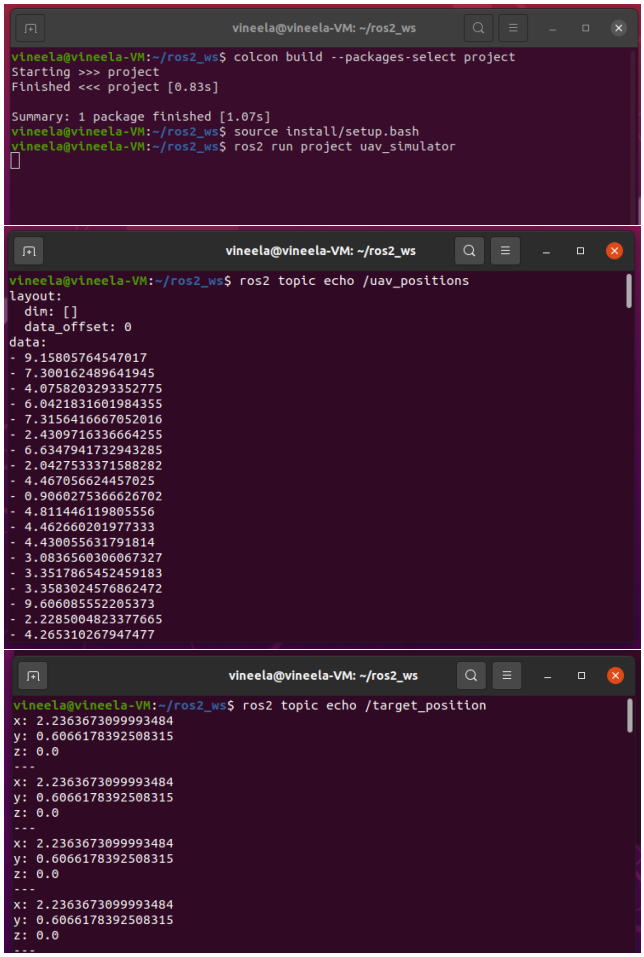

Figure 3: 'uav_simulator' in ROS 2

Figure 3 illustrates that one terminal publishes the UAV simulator node, while the other two terminals display the updated target positions and UAV positions messages. To run the uav_simulator and getting messages:

```
# This in terminal window 1
source /opt/ros/foxy/setup.bash
cd ~/ros2_ws/ && source install/
    setup.bash
ros2 run project uav_simulator
# Open new terminal window and run
    these below (This is terminal
    window 2)
source /opt/ros/foxy/setup.bash
cd ~/ros2_ws/ && source install/
    setup.bash
ros2 topic echo /uav_positions
# Open new terminal window and run
    these below (This is terminal
    window 3)
source /opt/ros/foxy/setup.bash
cd ~/ros2_ws/ && source install/
    setup.bash
ros2 topic echo /target_position
```

Project — simulator

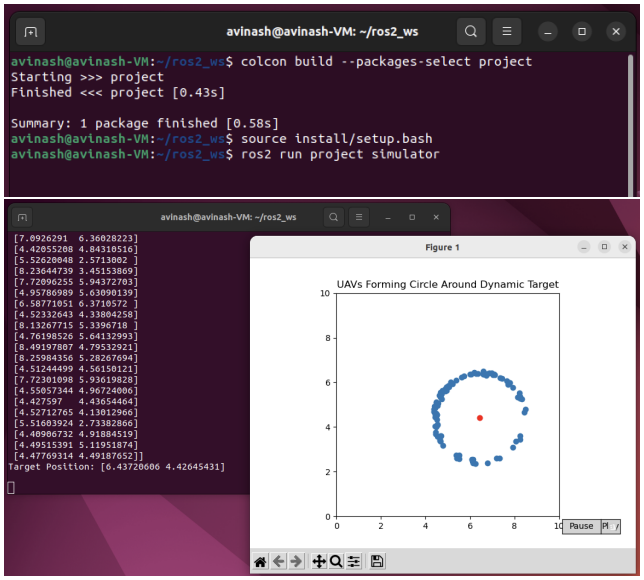

Figure 4: 'simulator' in ROS 2 using Python

Figure 4 demonstrates that the simulation is conducted in ROS2 using Python.

To run the simulator using Python:

```
# Open a new terminal window and run
    these below
source /opt/ros/foxy/setup.bash
cd ~/ros2_ws/ && source install/
    setup.bash
ros2 run project simulator
```

Challenges Faced

Implementing simulation in Gazebo posed challenges due to compatibility issues with the system. As a result, simulation was instead carried out using Python. However, errors arose due to directory path mismatches, particularly when attempting to create separate directories such as the message directory.

Additionally, ROS2 installation on a Ubuntu dual boot configuration resulted in missing packages, further complicating the setup process. Consequently, the decision was made to proceed with Ubuntu Desktop, ensuring smoother operation and access to the necessary resources for simulation.

Real Life Applications

1. **Disaster Site Exploration:** Swarm robots excel in exploring disaster sites, including collapsed buildings or areas affected by natural calamities. By forming a circle around key points of interest, such as trapped survivors or hazardous zones, these robots conduct efficient surveys, providing essential data for rescue teams.

2. **Resource Delivery and Communication:** Swarm robots are essential for delivering vital supplies, medical aid, or communication devices to those in need during disasters. By forming circles around designated drop-off points or communication stations, they facilitate the swift distribution of resources and establish reliable communication channels between survivors and rescue teams.

Hardware Implementation

Each UAV is outfitted with a suite of essential hardware components, including infrared (IR) proximity sensors, wireless communication capabilities, and four motors. These motors are crucial for maneuvering the drone and are controlled by signals transmitted via wireless communication. The code necessary to operate these components is uploaded onto each drone, ensuring seamless integration and functionality.

In addition to the hardware setup, the initial target positions are predefined to initiate the formation process. Upon receiving signals, the UAVs autonomously commence the formation, circling around the target positions. This coordinated movement is facilitated by the integration of sensor data from the IR proximity sensors, allowing the drones to maintain a safe distance from the target while moving in synchronization with it.

Conclusion

An algorithm able to accomplish key tasks.The project reveals that as the number of robots increases, achieving complete circle formation becomes feasible. Notably, utilizing 100 robots facilitated the attainment of full circle formation. Furthermore, there is potential to optimize resource utilization by refining the algorithm, suggesting the possibility of reducing the number of UAVs in future implementations. This observation underscores the scalability and adaptability of the proposed approach, opening avenues for further enhancement and efficiency in multi-robot systems.

Future Work

In the future, the algorithm could undergo refinement to automatically generate sub-swarms upon the successful detection of more than one target. This entails robots detaching themselves from the primary swarm to form sub-swarms, while the primary swarm continues to explore the remaining area. Additionally, a robot from the primary swarm can detach and hover between sub-swarms or the primary swarm to maintain connectivity. Ensuring connectivity between sub-swarms facilitates efficient search and rescue missions by aerial swarms, enabling comprehensive area coverage and exploration.

Citations

[1] Shahzad, M. M., Asad, M. H., Haris, M., Munawar, H., Yousaf, M. H. (2023). Formation Control and Sub-Swarm Generation of Multirotor UAVs. In 2023 International Conference on Robotics and Automation in Industry (ICRAI) (pp. 1-7). doi:10.1109/ICRAI57502.2023.10089546

[2] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7(66), eabm6074. doi:10.1126/scirobotics.abm6074