SRCIS

Search and Rescue

Coordinated Intelligence Systems

User / Developer Manual

Table of Contents

1. Introduction

1.1. Project Summary

SRCIS (Search and Rescue Coordinated Intelligence Systems) is a ROS2-based system designed to improve the effectiveness of search and rescue operations through human-robot collaboration. A key feature of the system is its continuous compositional control (CCC), which allows operators to provide guidance to the agents by remotely controlling them and then transitioning the control back to the agents. With humans and agents capable of changing their behaviors based on the environment, we can significantly reduce target search and tracking time beyond fully autonomous performance. In addition, SRCIS provides a scalable architecture that enables seamless integration of new robots with various roles. The system integrates heterogeneous platforms, including UGVs (Unmanned Ground Vehicles), Quadrupeds, and UAVs (Unmanned Aerial Vehicles), and can be further expanded by incorporating additional robotic platforms. Overall, SRCIS combines autonomy and human decision-making to improve efficiency in search-and-rescue operations.

1.2. Project Objective

A human-agent teamwork system designed to improve the effectiveness of search and rescue operations by merging human decision-making with robotic platforms through autonomous agents.

1.3. Intended Users

1.3.1 General Users

a. Public Safety and Event Security Teams

Security personnel at crowded locations such as amusement parks, airports, shopping centers, stadiums, and public events can use SRCIS to help locate missing children.

b. Search and Rescue Teams

Emergency response teams can activate SRCIS in disaster zones, collapsed buildings, forests, floods, or other hazardous environments which are limited for rescue workers to search for survivors while reducing risk to human responders.

c. First Responders

Police, firefighters, and emergency management personnel can use SRCIS during urgent incidents where fast target identification and area coverage are critical.

d. Defense and Security Organizations

Military or security teams can use SRCIS to locate designated targets, perform reconnaissance missions, and monitor dynamic environments using coordinated robotic platforms.

2. System Requirements

2.1. Hardware Requirements

2.1.1. Control Station/ Ground computer

- Processor : Multi-core 64 bit CPU (Intel Core i7/ Ryzen 7 or equivalent or higher)
- Memory: Minimum 32 GB RAM (64 GB strongly recommended for multi-agent operations + Docker setup)
- Storage: Minimum 64gb required
- Graphics: Dedicated GPU recommended for computer vision or AI workloads

2.1.2. Agent Requirements

- Camera Sensors
- Depth Camera
- Wireless Communications

2.1.3. Prefered but not Required

- IMU Sensor
- Embedded Computer

2.2. Software Requirements

2.2.1. Operating System

- Ubuntu 22.04 LTS
- Linux environment (preferred for ROS2 compatibility)

2.2.2. Core Robotics Framework

- ROS2 Humble
- ROS1_bridge (needed for robots that solely support ROS1)
- RViz2 (important for mapping checkings)

2.2.3. Computer Vision Libraries

- OpenCV (Arucos Detection, SLAM)
- cv_bridge (for encoding management)
- NumPy
- SLAM toolbox

2.2.4. Networking & Distributed Communication

- HazelCast
- ZeroTier

2.2.5. Drone Control

- Olympe
    - Basic control functions are available for the UAV(Parrot).
    - IMU data is available for SLAM.
    - AIR-sdk is an option for advanced control

2.2.6. Backend-Frontend

- Python
- Flask, Flask-Sock

2.2.7. Recommended dev tools

- Github
- Docker
    - Docker could be optional if using multiple control stations. However, when using a single control center, OpenCV requires different NumPy versions for OpenCV/cv_bridge.
- Threading Lib
    - The Threading Library is used to ensure smooth robot control while the system simultaneously receives and processes image data. Separate threads are assigned to robot control and image publisher/subscriber tasks so they can run concurrently.
    - This prevents one process from blocking the other, reduces delays, and avoids situations where publishers must wait for turns or temporarily stop. As a result, the robot can respond to control commands smoothly while maintaining continuous image streaming and processing.

2.3. Dependencies

- Flask (≥ 3.1.2) https://pypi.org/project/Flask/
- Flask-Sock (≥ 0.7.0) https://pypi.org/project/flask-sock/
- Google GenAI Python SDK (≥ 1.73.1) https://pypi.org/project/google-genai/
- Hazelcast Python Client (≥ 5.5.0) https://pypi.org/project/hazelcast-python-client/
- NumPy (≥ 1.26.0) https://pypi.org/project/numpy/
- Ollama (≥ 0.4.0) https://ollama.com/
- OpenCV Python Headless (≥ 4.9.0.80) https://pypi.org/project/opencv-python-headless/

- Pydantic (≥ 2.0.0) https://pypi.org/project/pydantic/

3. Installation & Setup

This part is for future developers who continue the SRCIS project

3.1. Prerequisites / Installation

1. Install the required software first
    - Ubuntu 22.04 LTS
    - ROS2 Humble
    - ZeroTier
    - Hazelcast
2. Clone Repository
    - Github Link
    - Slam Toolbox SLAM: github

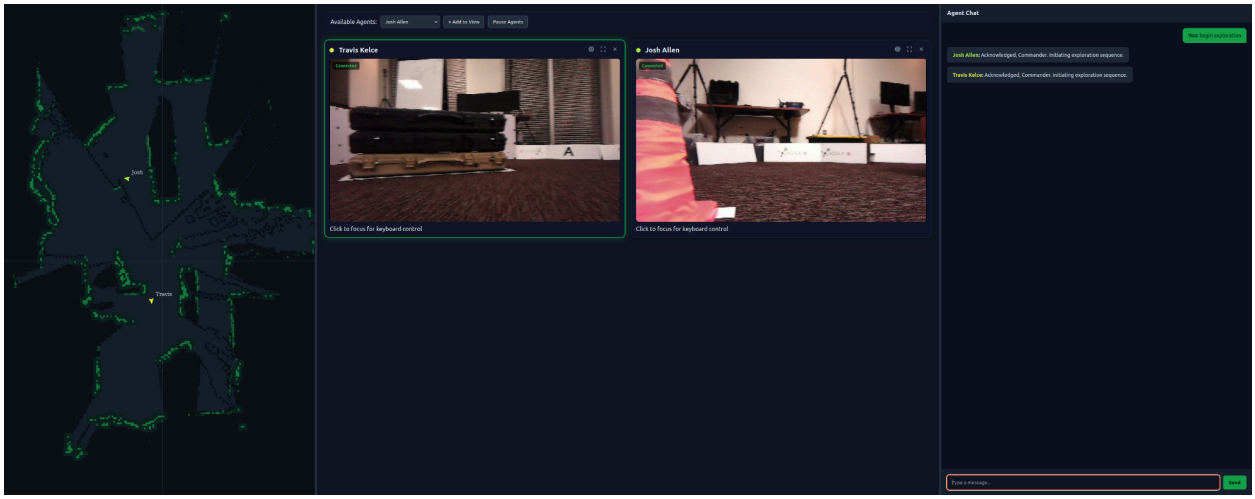
Follow more specific instructions in the repository

3.2. Running the System

1. Check all the running programs(nodes) with RViz2
2. Terminal: `RViz2`

4. User Guide

4.1. Main Features



4.1.1. Total Map

On the left side of the view, we have the map for the world. Initially when the robots start moving, they build the map and merge them. Each of the robots have their own pointers and names

4.1.2. Agent List with Camera View

In the middle of the screen, there are all robots viewing the world. The user can select additional features

1. Target Robot follow Button (Map View) 
2. Full screen Button 

4.1.3. Agent Chat

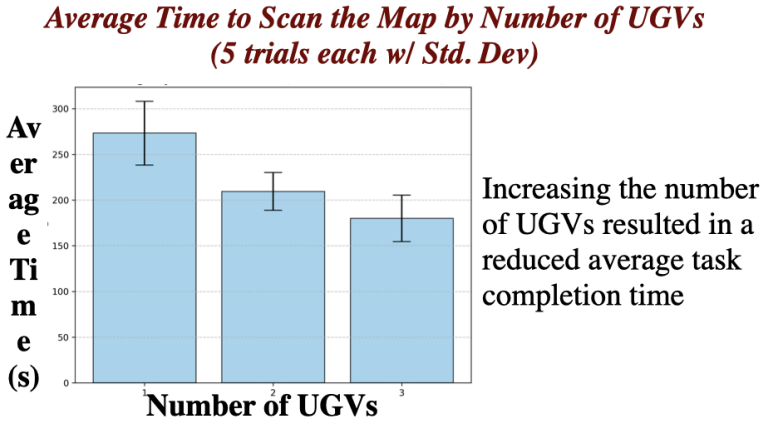
On the right side of the screen, there is a chat box currently for Agents current state. Each agents reports their current state, and there are mainly three functions from the robot including ‘Idle State’.

1. Following state : follow target

2. Exploration state

4.2. Correlation Between Mapping Time and Number of Robots

The number of robots has a crucial impact on the initial mapping process. As seen in the graph with horizontal lines as the number of robots and vertical lines with time taken, the efficiency goes up when they increase.



4.3. Example Workflow

1. Activate all programs using ‘limo_startup.py’.
2. Run the robots needed. (limo, UAV, Unitree.. etc)
3. Initially, the robots will perform a 360 (except UAVs) to map the area and determine the area to search.

5. Developer Guide

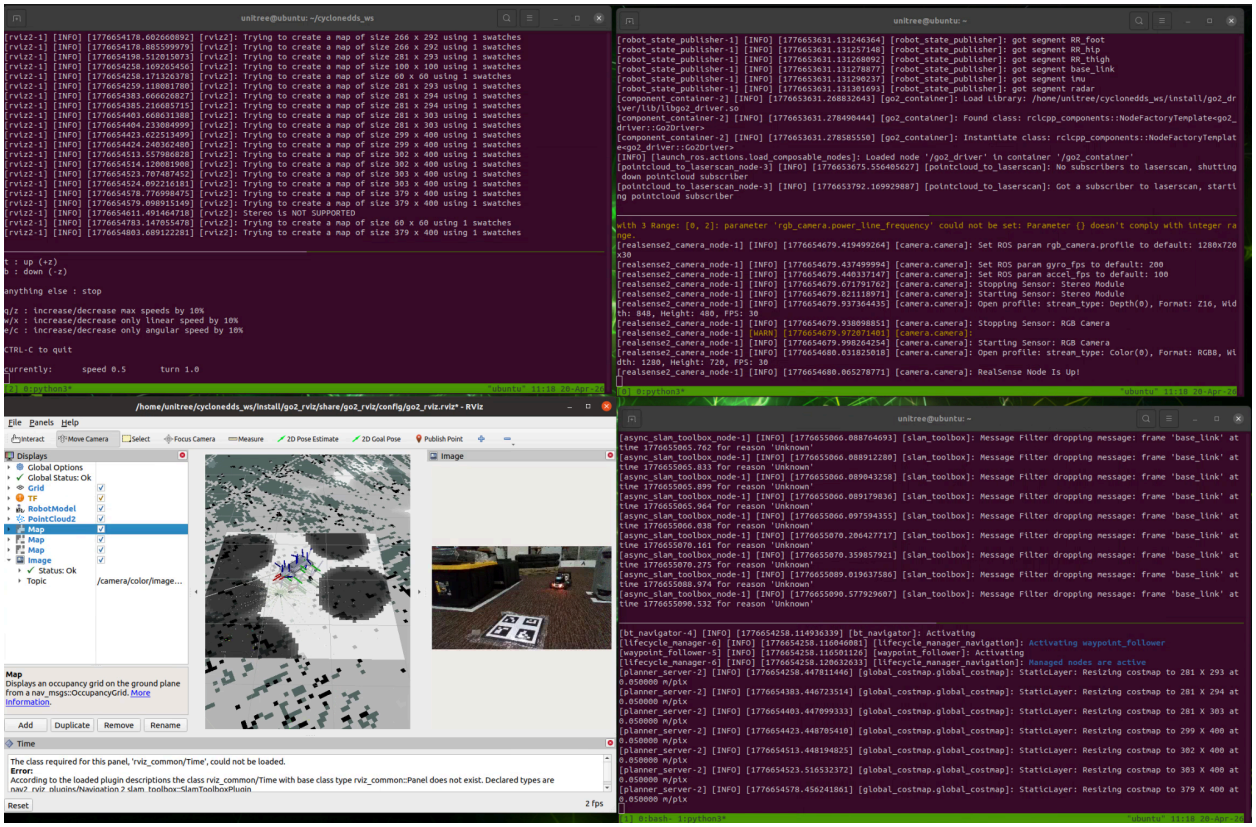
The purpose of the developer guide is to provide a walkthrough of the following:

- Basic server setup for each robotic platform
- Command structure
- Implementation of the ROS framework.

5.1. Drivers (Limo, Parrot)

- The drivers are responsible for managing the operational state of the UAV (Unmanned Aerial Vehicle), Quadruped, and UGV (Unmanned Ground Vehicle). It controls state transitions between behaviors, including idle, exploration, retargeting, following, and patrolling, ensuring the platform operates in accordance with mission requirements.
- In addition to state management, the driver initializes the SRCIS integration nodes required for system interoperability. This includes the command node, which issues velocity commands to the UGV and registers the robotic platform with Hazelcast.
- The driver is important because without it, you have to run:
    - ROS Teleop – Provides manual control of the robot via keyboard or joystick inputs.
    - RViz2 – Visualization tool used to monitor sensor data, maps, and robot state in real time.
    - Nav2 Bringup – Initializes the ROS 2 Navigation stack for autonomous path planning and execution.
    - SLAM Toolbox – Performs simultaneous localization and mapping, allowing the robot to build and update maps of its environment.
    - RealSense – Handles input from Intel RealSense cameras for depth sensing and perception.

- Go2 Bringup – Launches the core drivers and interfaces required to operate the quadruped robot hardware.



5.2. Limo_startup

The purpose of the program is solely to control the whole interface. Unlike other Python programs, the Ros2 program needs to be built before running. This usually requires multiple terminals open. However, with limo_startup.py, we can control them all in one single interface and turn them on in the required order when needed.

5.3. Mapping and Environment Representation

5.3.1. Overview

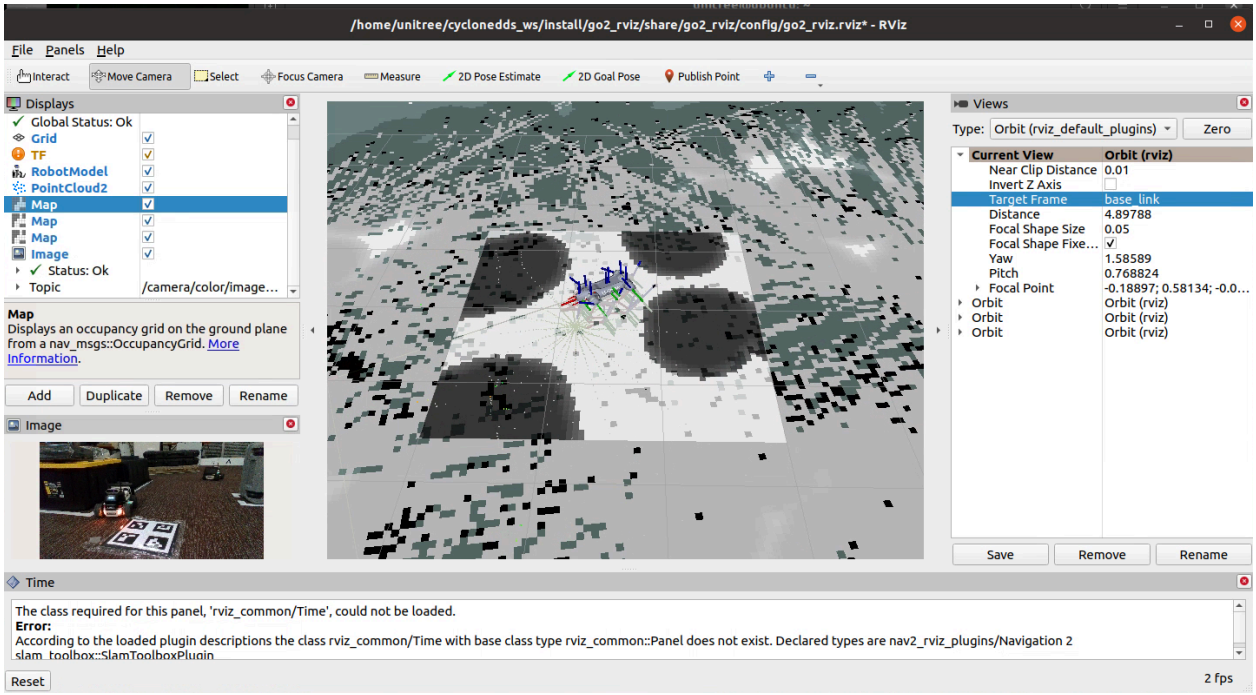
All visualization is performed in RViz2 with slamtoolbox and the nav2 library. The system builds and maintains a spatial understanding of the

environment using a combination of LiDAR and RGB-D camera data.

This information is used for:

- SLAM (Simultaneous Localization and Mapping)
- Obstacle detection and collision avoidance
- Path planning using costmaps

5.3.2. RViz2 layout for quadraped w/ Map, Quadruped, and Camera visible



5.4. Target Engagement

5.4.1. Overview

The target engagement subsystem is responsible for coordinating agents once a target has been detected. It integrates perception, shared mapping, and path planning to enable agents to:

- Track a target's position
- Predict its movement

- Navigate efficiently to intercept or contain it

5.4.2. Target Detection and Registration

When a target is identified by any agent (via ArUco detectors):

- The detecting agent computes the target's position in the global frame
- The position is transformed using the system's TF tree (Transform tree that tracks how all frames are related to each other)
- The target is published to a shared topic and inserted into the global map

The shared map acts as the single source of truth for:

- Target location
- Target motion

5.4.3. Target Tracking and Prediction

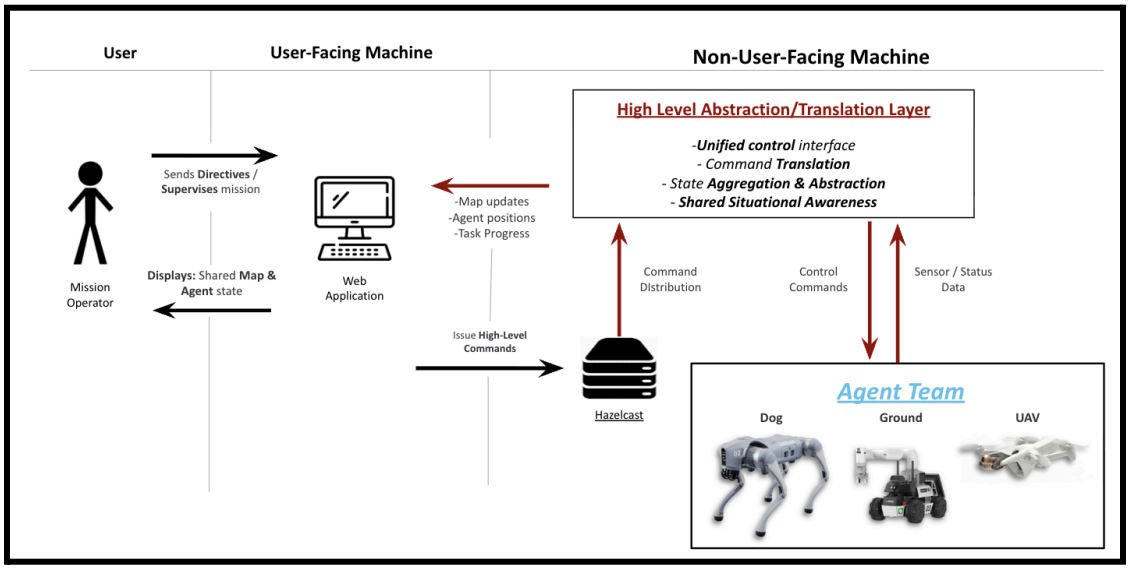
- If continuous observations are available, the system estimates target motion using simple kinematics or filtering (e.g., velocity estimation).
    - Short-term trajectory prediction
    - Reduced latency in response
    - More effective interception planning

5.4.4. Interception and Containment Strategies

- Direct Pursuit
- Interception
- Cornering

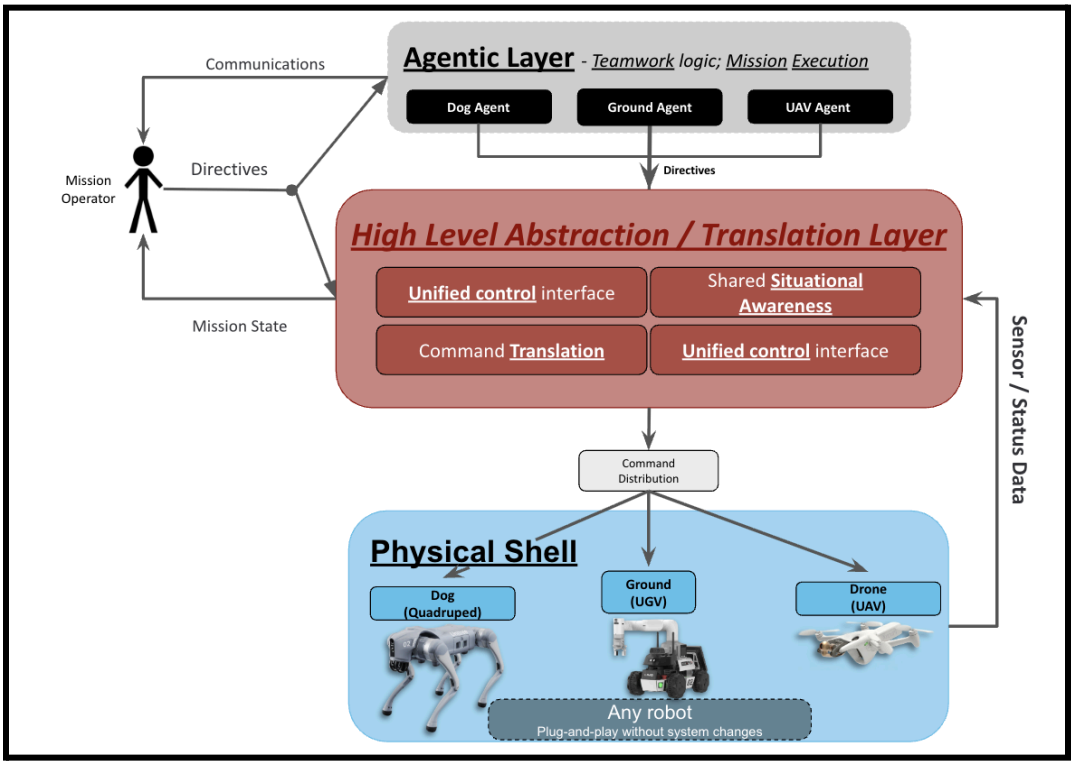
6. System Architecture Diagrams

High-Level Diagram



6.1. System Interaction Diagram

6.1.1. This diagram highlights the data flow and the system backend work.



- Mission Operator:
    - The mission operator defines high-level objectives and supervises overall mission progress. The operator interacts with the system by issuing high-level directives and receives aggregated system state information to maintain situational awareness.
- Web Application:
    - The web application serves as the primary control interface for the mission operator. It displays the shared map, agent positions, and task progress in real time, while enabling the operator to send commands to the system. The application does not interact directly with hardware and communicates exclusively with backend services.
- Central Intelligence Backend (CI):
    - Uses Hazelcast, manages command routing, distributed state management, and system scalability across nodes. Within the backend, the high-level abstraction layer is responsible for ingesting commands, aggregating system state, and distributing commands to the appropriate agents.
- Agentic Layer:
    - The execution layer consists of the robotic agents, including Go2 (quadruped, dog), LIMO (ground vehicle, UGV), and Parrot (drone, UAV). These agents are responsible for executing the mission assigned commands and continuously reporting telemetry and sensor data back to the system.
- Data Flow Summary:

- In the forward control flow, the mission operator sends directives through the web application, which forwards them to the backend. The backend processes these commands via the abstraction layer, distributes them appropriately, and delivers them to the agents for execution. Otherwise, the agents continue their exploration for the target as a team, as they initially planned.
- In the reverse feedback loop, agents send sensor and status data back to the abstraction layer, where it is aggregated and processed. This aggregated state is then transmitted to the web application and presented to the operator for monitoring and decision-making when necessary..

7. Maintenance and Future Work

7.1. Target Detection improvement

Currently the target is using Aruco detectors for recognition, which is not available in real world situations. We can solve this problem by implementing CNN.

7.2. AI agent voice enable

In the frontend ‘agent chat’, the user can send commands but are limited like ‘current status’. If this project is future selected, we can have voice libraries connected for non-experts to use it without training.

7.3. Large space investigation

Due to wifi connection, the project was only able to be used on the lab which has Internet connection. We could expand the project using LTE searching large areas with more robots.