Design Document

Human-autonomous teamwork of ground and air vehicles

Team Members
Yavanni Ensley, yensley2022@my.fit.edu
Younghoon Cho, ycho2021@my.fit.edu
Jaylin Ollivierre, jollivierre2022@my.fit.edu

Faculty Advisor
Thomas Eskridge, teskridge@fit.edu

1. Introduction

1.1. Purpose

This design document will create both a high-level overview of our project – Human-autonomous teamwork of ground and air vehicles – architecture. The goal of the project is to allow for continuous compositional control between human operators and autonomous vehicles. This includes allowing one operator to take control over multiple different vehicles/robots.

1.2. Scope

- Support direct robot control
- Support switching between robots quickly
- Allow for operators to get a “birds eye” view of the situation

2. System Overview

2.1. Users

- **Mission operators:** Responsible for monitoring all autonomous vehicles and assisting them to complete their tasks.
- **Field operatives:** Check for updates on mission status and directives.

2.2. Key Features

- **Different control modes:** Support different control schemes: tanks, cars, drones, etc.
- **Top level view:** Allow operatives to see all active robots and their status.
- **Robot map:** Utilizing GPS data or SLAM tracking, display a map of robot location.
- **Robot-robot communication:** Autonomous agents should be able to communicate with each other, requesting help when they are not suited for specific tasks.
- **Focus view**: Allow operators to pin multiple robots to their screen.
- **Non-local access:** The control panel should be accessible from external networks to account for users on cellular devices.

3. System Architecture

3.1. High-Level Architecture

- **Frontend**: HTMX for HTML management and TailwindCSS for styling
- **Backend:** Python server running as a ROS2 node.
- **Database:** PostgresSQL or MongoDB

3.2. System flow

3.2.1. Setup

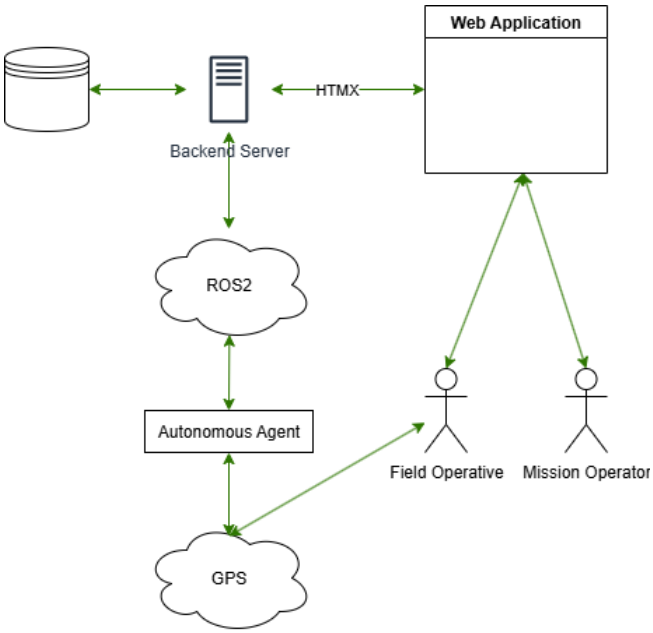
- Server initialized
- Establish connection with database
- Autonomous agents discovered through ROS2

3.2.2. User

- User logins in to the frontend
- Presented with array of all agents and map
- Select an agent to view, control scheme displayed
- Optionally, pin additional agent views

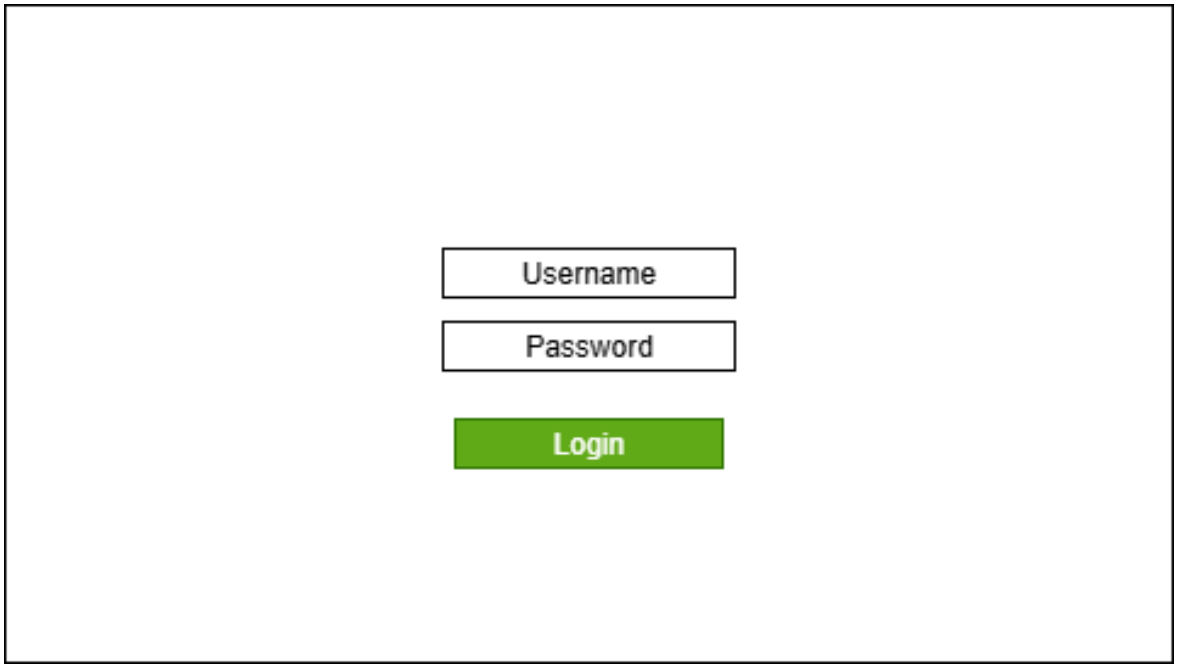
4. Diagrams

4.1. App Structure

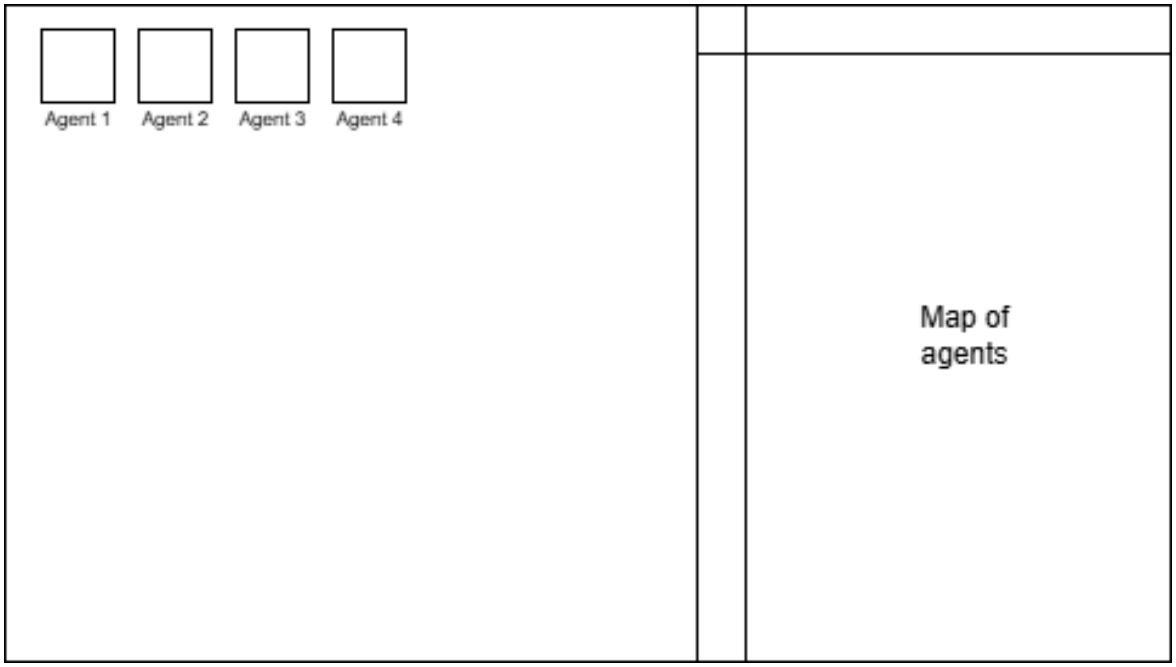


4.2. Frontend Design

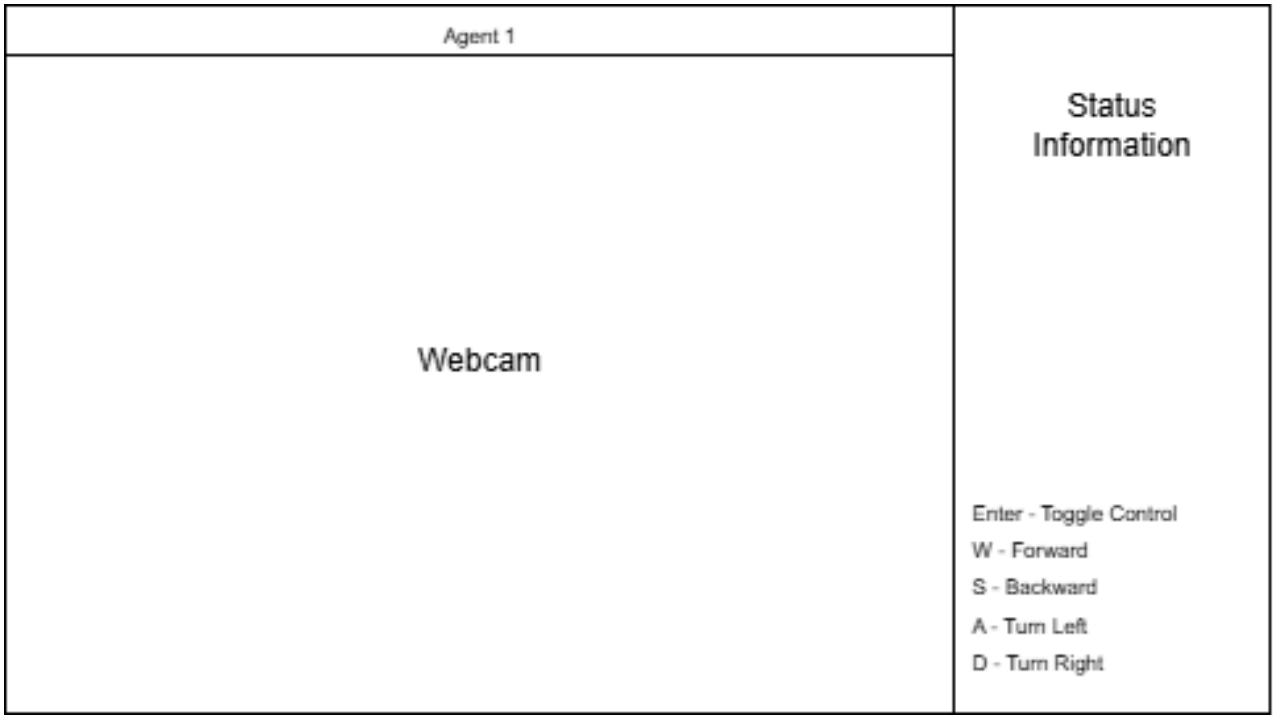
4.2.1. Auth View



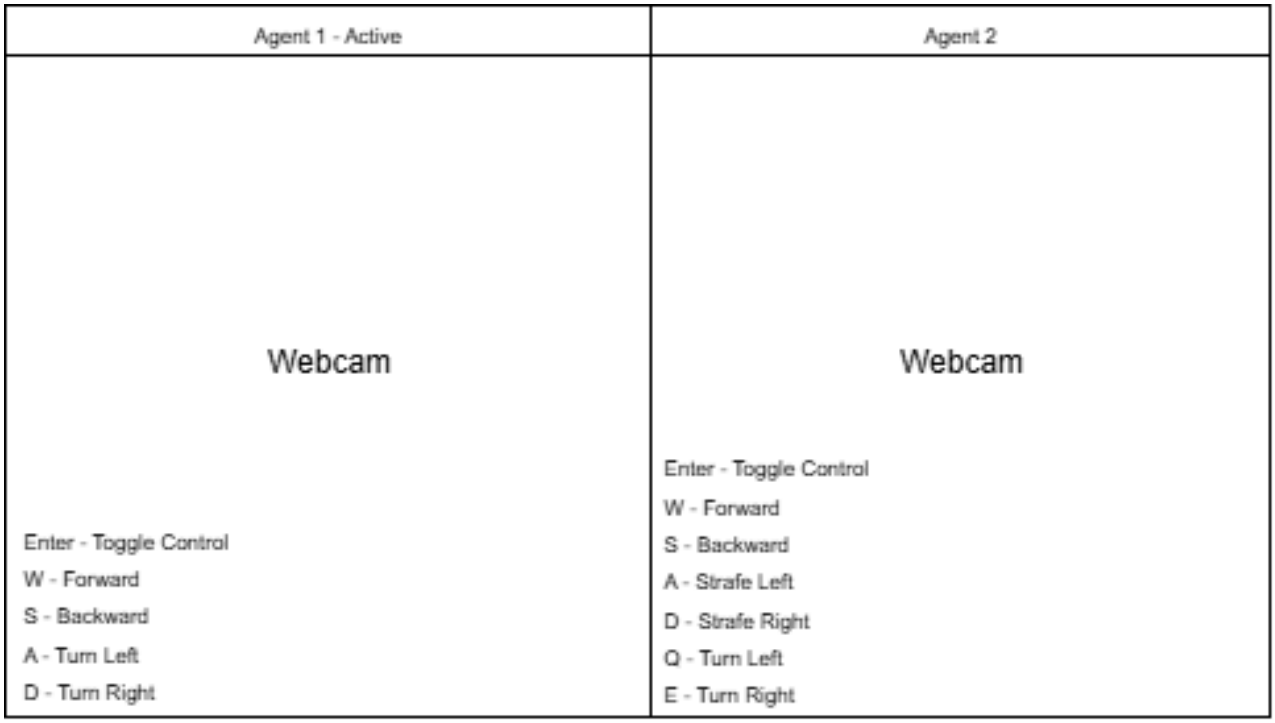
4.2.2. Agent List View



4.2.3. Agent View



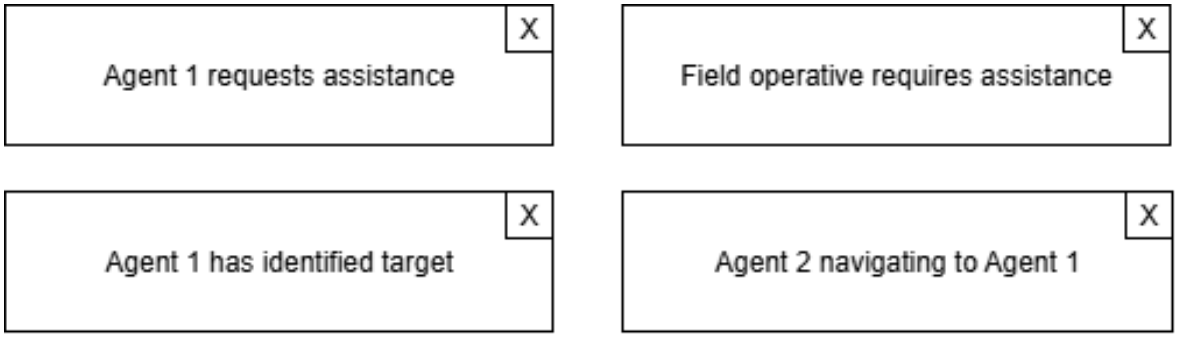
4.2.4. Multi Agent View



4.2.5. Multi Agent View cont.

Agent 1 - Active	Agent 2
Enter - Toggle Control W - Forward S - Backward A - Turn Left D - Turn Right Webcam	Enter - Toggle Control W - Forward S - Backward A - Strafe Left D - Strafe Right Q - Turn Left E - Turn Right Webcam
Agent 3	**Agent 4**
Enter - Toggle Control W - Forward S - Backward A - Turn Left D - Turn Right Webcam	Enter - Toggle Control W - Forward S - Backward A - Turn Left D - Turn Right Webcam

4.2.6. Agent Notifications



5. Classes

5.1. Agent

Contains the capabilities of a given agent, as well as its ROS2 information.

- node: str
- namespace: str
- capabilities: set[Capabilities]

5.2. Capability

Represents behaviours/functions the robot has access too

- kind: str

5.3. Example Capability: Move

Represents the robot's ability to move.

- kind: str = "move"
- move(linear: tuple[float, float, float], angular: [float, float, float])
    - "linear" represents the linear velocity
    - "angular" represents the angular velocity

5.4. Example Capability: Location

Represents the robot's ability to have a global location

- kind: str = "location"
- get_location()

5.5. Example Capability: Vision

Represents the robot's ability to see

- kind: str = "vision"
- get_image()
    - Get the current frame from the robot.

5.6. User

Represents a user of the application.

- role: "operator" | "operative"
    - Important for preventing general operatives from interfering with operators.
- username: str
- static login(username: str, password: str)