CyFlyBot

Design Document

Team Number: sddec26-11

Client: Dr. Diane Rover

Advisers: Dr. Diane Rover & Dr. Phillip Jones

Team Members/Roles

Daniel Bieber

John Brittain

Jacob Nickel

Ping Wu

Tu Reh

Mitch Fowler

Caleb Sanchez

Team Email

https://sddec26-11.sd.ece.iastate.edu/

Revised: 4/29/2026 v1.0

Executive Summary

CyFlyBot is a coordinated autonomous robotic system that pairs a mobile ground vehicle (UGV) with a quadcopter drone (UAV) to perform inventory management tasks in warehouse environments. The project addresses a fundamental gap in modern warehouse automation: ground robots are capable but limited by their onboard sensor horizon, while aerial drones provide broad situational awareness but are constrained by short battery life and cannot perform ground-level tasks independently. By combining both platforms, CyFlyBot enables missions that neither robot could complete alone, improving navigation efficiency, inventory accuracy, and safety.

The system operates as three coordinated nodes: the UGV (iRobot Create 2 CyBot platform with TM4C123 microcontroller), the UAV (Crazyflie 2.1 Brushless quadrotor equipped with the Flow Deck v2 for indoor positioning, Multi-Ranger Deck for obstacle avoidance, and a camera deck for visual perception), and an HQ computing device running the mission planner. All inter-node communication is managed through ROS 2 Humble using a shared interface package (cyfly_interfaces) that formally defines all message, service, and action types across the system. The UGV navigates autonomously using Nav2 and SLAM Toolbox for real-time occupancy-grid mapping, while the UAV scans shelf faces and identifies inventory items using ArUco marker detection. Key design requirements include: UAV flight time of at least 10 minutes per battery cycle, communication latency no greater than 200 ms, and operation under standard warehouse lighting without any infrastructure modifications.

A central element of the design is the UGV-ferry strategy: because the Crazyflie's 350 mAh LiPo battery provides approximately 10 minutes of flight, the UAV rides on top of the UGV between aisles and takes off only for short, precise scanning runs at each shelf. The UAV uses its downward-facing camera to detect an ArUco marker printed on the UGV landing pad, enabling precision landing and relative-position tracking without any external infrastructure.

Progress to date includes a complete ROS 2 workspace organized into seven packages, a functional Gazebo Classic 11 warehouse simulation environment running within a Docker container, a verified ArUco marker detection pipeline, and initial physical hardware testing of the Crazyflie with the Multi-Ranger and Flow Decks. The UGV firmware has been developed and validated in both remote-control and autonomous modes on the TM4C123 platform. Immediate next steps are SLAM Toolbox integration for full autonomous navigation in simulation, HQ mission state machine implementation, and the transition to physical hardware deployment.

Learning Summary

Development Standards & Practices Used

The following development standards and practices were applied: IEEE 802.11 (Wi-Fi) for all wireless communication between nodes; IEEE 1872 (Ontologies for Robotics and Automation) as a shared vocabulary for inter-node data definitions; IEEE 1873 (Robot Map Data Representation) for occupancy-grid map format compatibility; IEEE 12207 (Software Life Cycle Processes) as the framework for requirements documentation and structured development; ROS 2 DDS publish-subscribe architecture for distributed node communication; Docker containerization for reproducible cross-platform development environments; and Agile (sprint-based) project management.

Summary of Requirements

Key requirements (see Section 2 for full list): UGV autonomously navigates warehouse aisles and avoids dynamic obstacles; UAV identifies inventory via ArUco markers and relays locations to HQ; end-to-end communication latency ≤ 200 ms; UAV flight time ≥ 10 minutes per battery cycle; UAV maximum payload ≤ 40 g; both robots operate under standard warehouse lighting with no infrastructure changes; operators can visualize robot state and override behavior in real time; emergency stop and fail-safe protocols are implemented on both platforms.

Applicable Courses from Iowa State University Curriculum

CPRE 288/389 Embedded Systems I & II (TM4C123 microcontroller, CyBot platform firmware); CPRE 480/481 Computer Networks (Wi-Fi communication architecture and protocol selection); CPRE 488 Embedded Systems Design (hardware-software co-design and integration); CPRE 491/492 Senior Design Project (project management, documentation, and system integration); SE 319 Software Construction and User Interfaces (HQ operator interface design); SE 339/421 Software Architecture and Verification (ROS 2 node architecture, testing methodology); EE 324 Signals and Systems (sensor data processing and filtering).

New Skills/Knowledge acquired that was not taught in courses

ROS 2 Humble framework (nodes, topics, actions, services, launch files, colcon build); Gazebo Classic 11 simulation environment (world creation, robot URDF/SDF, sensor plugins); Docker containerization for cross-platform robotics development; Nav2 autonomous navigation stack (global planner, local controller, costmap configuration); SLAM Toolbox for real-time occupancy-grid mapping with map persistence; Crazyflie 2.1 Brushless operation (cfclient, Crazyradio, expansion deck integration); ArUco marker detection with OpenCV (camera calibration, 6-DOF pose estimation); Crazyswarm2 ROS 2 bridge for Crazyflie CRTP integration.

Table of Contents

List of figures/tables/symbols/definitions

1. Introduction

1.1. Problem Statement

In industrial and warehouse environments, operators must continuously verify what items are in stock and where they are located. Today, this often relies on manual aisle walks, handheld scanners, or fixed infrastructure, which can be slow, error-prone, and disruptive to normal operations. Warehouses are also dynamic spaces: aisles may be blocked by pallets or carts, lighting varies, and workers and equipment share the same floor space.

A single robot platform rarely satisfies the full needs of warehouse inventory capture. A ground robot (UGV) can operate for long periods and move safely through aisles, but its sensors are limited by line-of-sight and shelf occlusions, making it difficult to reliably scan inventory located at multiple heights. An aerial robot (UAV) can quickly scan shelf faces and access elevated inventory positions, but it is constrained by short indoor flight time and is inefficient to fly long distances between aisles.

This creates a coordination problem: the system must combine reliable ground mobility with efficient multi-height scanning, while maintaining safety around people and equipment and producing a unified, accurate inventory record. Solving this requires not just multiple robots, but a clear division of responsibilities and a central coordinator that can manage tasks, data fusion, and fail-safe behavior.

The CyFlyBot project addresses this need by designing a coordinated three-node system for warehouse inventory operations: (1) a UAV that performs close-range scanning of shelf faces and inventory identifiers (e.g., markers/barcodes), (2) a UGV that ferries the UAV between aisles and provides sustained ground mobility, and (3) a central HQ node (desktop PC) that plans missions, aggregates scan results into an inventory database, and supervises both robots. In the final design, the UAV is deployed for short, precise scan flights, then returns to the UGV for transport to the next scan location, enabling efficient coverage without requiring additional warehouse infrastructure.

Overall, the core problem is enabling accurate, repeatable, and scalable inventory scanning in industrial warehouse settings by coordinating an aerial scanner (UAV) with a ground transporter (UGV) under a centrally managed control architecture (HQ), while meeting constraints on safety, battery life, and communication reliability. Beyond this warehouse-focused use case, the system is also intended to advance practical coordination patterns for UAV–UGV teams (task allocation, communication, and supervision) that can transfer to other application scenarios.

1.2. Intended Users

CyFlyBot is designed as a collaborative robotic system for **industrial and warehouse environments**. Its primary users include warehouse operators, managers, and engineering teams focused on automation, logistics, and efficiency improvement.

- **Warehouse Operators & Staff:**
  Ground-level employees interact with the UGV for material handling and inventory tasks.

They rely on the UAV to provide real-time situational awareness of aisle congestion, stock locations, and potential hazards. Their needs include safe, predictable robot behavior, easy task assignment, and minimal disruption to daily workflow.

- **Operations Managers & Supervisors:**
  Managers use the CyFlyBot system to monitor warehouse efficiency, track task completion, and identify bottlenecks. They need intuitive dashboards, real-time alerts, and metrics to make informed decisions on workflow optimization and resource allocation.
- **Engineering & Automation Teams:**
  Engineers and technical staff can integrate CyFlyBot into existing warehouse systems, refine navigation and task algorithms, and experiment with multi-robot coordination strategies. Their needs include reliable communication, compatibility with warehouse infrastructure, and flexibility for testing new operational procedures.
- **Stakeholders & Decision Makers:**
  Business stakeholders, investors, or company executives can observe tangible improvements in efficiency, safety, and automation capability. They need clear visualizations and demonstrable outcomes to evaluate the value of robotic integration.

Overall, CyFlyBot improves warehouse operations by **combining aerial and ground perspectives**, enhancing efficiency, safety, and task execution while providing actionable insights for operators, managers, and technical staff.

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

Functional Requirements

1. The UGV must autonomously navigate warehouse aisles and avoid dynamic obstacles.
2. The UAV must provide an aerial map of the warehouse, identifying obstacles and item locations.
3. The UAV must track inventory from above and relay item locations to the UGV.
4. The UGV and UAV must communicate in real-time with the HQ control station. All coordination between the UGV and UAV is mediated through the HQ; no direct peer-to-peer communication between the robots is required.
5. Both robots must implement emergency stop and fail-safe protocols.
6. The system must optimize task allocation and minimize travel time for missions.

User Requirements

1. Operators must be able to visualize real-time robot positions and progress.
2. Operators must be able to manually override UGV or UAV behavior if needed.
3. Staff should interact easily with the system without extensive robotics training.
4. Managers must be able to track efficiency metrics (task completion, travel time, bottlenecks).
5. System must be straightforward/cost-effective to install.

Physical Requirements

1. UGV width must fit within standard warehouse aisles (constraint).
2. UAV must operate safely under indoor lighting and ceiling heights typical of warehouses.
3. Robots must operate reliably at warehouse temperatures and humidity levels.

Operational Requirements

1. Communication must remain robust in indoor industrial Wi-Fi environments.
2. Robots must detect and avoid humans and moving equipment (forklifts, carts).
3. UGV battery life will support a full shift (several hours) or be swappable.
4. UAV flight time must be sufficient for continuous monitoring of assigned zones.

UI/UX Requirements

1. The interface must show robot locations, task status, and live aerial view.
2. Task reassignment must be intuitive and require minimal clicks.
3. Alerts and warnings (collisions, low battery, communication loss) must be clear and actionable.

Constraints

1. UAV flight time ≤ 10 minutes per battery cycle.
2. Maximum communication latency ≤ 200 ms for real-time coordination.
3. UAV maximum payload ≤ 40 g (Crazyflie 2.1 Brushless limit)
4. Both robots must operate under standard warehouse lighting without additional illumination.

2.2. Engineering Standards

The importance of engineering standards, particularly for a project like CyFlyBot, lies in their ability to ensure that systems like a ground-based CyBot and an aerial Bitcraze FlyBot can communicate. Not only to ensure they can communicate but operate safely in a shared environment as well.

Without standards like IEEE 802.11, communication between the UGV, the UAV, and the control station would be a mess. Standards provide a universal framework so that hardware from different manufacturers can exchange data packets without errors.

Standards provide a proven framework for how robots should identify objects or navigate in shared spaces. By using established standards like IEEE 1872, we can ensure the map or directions the drone creates is interpreted correctly by the ground robot, preventing collision and mission failure.

Using standards allows us to focus on the unique challenges of warehouse navigation rather than debugging basic communication layers. It also ensures that our project deliverables meet the

professional criteria expected by our clients and the engineering community. Essentially, standards turn a collection of parts into a predictable, professional system.

Standards:

1. IEEE 12207: The industry-standard guide for how teams should document requirements, design, and testing. It organizes the entire software development process, from initial requirements to final project delivery. It is used to bring professional rigor to your work by defining clear development stages and responsibilities.
2. IEEE 1872: This standard acts as a universal dictionary that defines the core concepts and relationships within a robotic system. The goal is to eliminate any gaps in communication between different platforms by providing shared vocabulary. It ensures that when the drone and ground robot exchange data about obstacles or target location, they are both using the exact same definitions.
3. IEEE 1873: This standard establishes a common format for how robots store and share map information. For example, a warehouse layout or a hazardproof area. Its primary purpose is to ensure that a map created by the drone sensors can be accurately read and utilized by the ground robot navigation software.

Relevance:

1. IEEE 1872: This is relevant to our mission because we are combining two distinct robotic platforms. We face the challenge of different software environments interpreting sensor data differently. By applying this, we establish a shared language for target location, obstacles, and shelf IDs. This prevents system errors where the drone might identify a hazard that the ground robot does not recognize, standardizing our decision-making logic.
2. IEEE 12207: This is essential for our two-semester timeline. Our project involves complex integration between the drone controller and the CyBot's motor controllers; we cannot rely on quick-fix coding. This standard is relevant because it forces us to document our system requirements and perform verification at each step. It changes our project from a loose collection of experiments into a structured engineering system.
3. IEEE 1873: This is the most practical standard for our warehouse application. Our system succeeds or fails based on the drone's ability to map the warehouse for the CyBot. By following this standard for our map data, we ensure that the spatial information is formatted in a way that is readable by any navigation algorithm we choose.

Review and Collaboration

While there were few differences between the standards set between all of us, we were able to agree on 12207, 1872, and 1873. When brought to each others attention, we agreed these will be the most crucial. However, standards like 1621-2004 proved to be useful for when it will be interacting with any humans at any point. Another option was 1936.1-2001, due to it's framework for describing the

drone and it's applications. Along with this, all of the standards within it's Smart City section all proved to be valuable resources in a roundabout way.

Implementation:

1. IEEE 12207: We will utilize the standard to formalize ou project management across both semesters. This will provide a clear audit trail for our advisors, demonstrating that the system was developed through a repeatable, professional engineering process rather than trial and error.
2. IEEE 1872: We will establish a common data dictionary for the entire team. This dictionary defines exactly what terms like obstacle, target, and coordinate mean, regardless of whether the data originates from the UAV's vision sensors or the UGV's LiDAR.
3. The map created by the FlyBot will be treated as a structured data asset that adheres to this standard's formatting requirements. This makes the system modular and scalable. It means that if we decide to upgrade our drone hardware in the future, we will not need to rewrite the CyBot's software.

3 Project Plan

3.1 Project Management/Tracking Procedures

For the CyFly projecting, we are adopting an agile approach to manage the complex integration between the ground unit and the drone. This process allows us to break down high level objectives such as drone-to-bot communication and path mapping into manageable sprints. By focusing on constant testing and incremental updates, we can pivot quickly if a hardware or software constraint changes rather than waiting for a single launch at the end.

To maintain transparency and ensure consistent progress across the two-semester timeline, our team utilizes multiple integrated digital tools. We use GitLab for the code repository. All software for the ground robot and quadcopter is posted on the GitLab. We utilize branching strategies to ensure the main codebase remains functional while individual members develop specific modules. We also can use milestones on the GitLab to set deadlines and make sure that we reach our goals during the building process. For file sharing, we use Microsoft SharePoint. All academic papers, reports, design documents, and brainstorming files are stored in SharePoint collaborative editing and organization purposes. The team also meets in person every Monday for an hour to talk about what has been done and what needs to be worked on. All of this together helps us stay on the same page and work better as a team.

3.2 Task Decomposition

The team will be split into groups to work on hardware, software, and UI. We aim to complete most tasks for hardware and software to get our main system running before implementing the UI. Having these subgroups allows our team to more efficiently meet milestones.

The hardware subgroup handles tasks related to the drone and ground unit. These tasks include attaching input and output pins for required cameras and sensors on the drone and ground unit and developing an API with commands to control both units that the software subgroup can use. This subgroup will also make any physical updates to either unit.

The software subgroup deals with creating modules and databases. This involves developing communication between the drone, ground unit and HQ, computer vision, mapping, and path planning modules, and a location database for items. This subgroup will also split into the UI subgroup to create an operator interface.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

To stay on track with the CyFlyBot project, the work is broken into milestones across hardware, software, and system integration. With an agile approach, these will be completed through sprints and refined as development continues.

One of the first major milestones is establishing basic communication between the drone (UAV) the ground robot (UGV), and HQ (A personal computer or PC). This will be considered successful when commands can be reliably transmitted between the systems with minimal delay and at least 95% successful message delivery during testing.

The next milestone focuses on hardware integration, where all required sensors such as cameras and positioning components are fully installed and operational on both the drone and ground unit. Progress will be measured by verifying that each component consistently collects and transmits usable data during test runs without failure.

For the software subsystem, a key milestone is the development of mapping and path planning functionality. This will be evaluated by testing whether the drone can successfully scan an environment and localize it's location, and whether the ground robot can navigate to a target location with at least 85% accuracy in reaching the correct destination.

Another important milestone is the implementation of computer vision and object detection. The system should be able to recognize and identify target objects such as items on shelves, bar-codes etc. with an initial accuracy of around 75–80%, with improvements over time as the model is refined.

The user interface will be developed after core functionality is stable. This milestone will be achieved when an operator can monitor the environment, receive drone data, and send commands to the ground robot through a clean and functional interface. Success will be measured by usability testing and the ability to complete basic tasks without errors.

Finally, the full system integration milestone will combine all subsystems into a working pipeline. The project will be considered successful when the drone can scan an area, identify a target, communicate that information to the ground robot, and the robot successfully navigates to and interacts with the target. This will be evaluated through end-to-end testing scenarios with a success rate of at least 80%.

Throughout the project, progress will be tracked using sprint goals, completed tasks, and performance metrics for each subsystem. These milestones will continue to be refined as development progresses to ensure the system meets both functional and performance expectations.

3.4 Project Timeline/Schedule

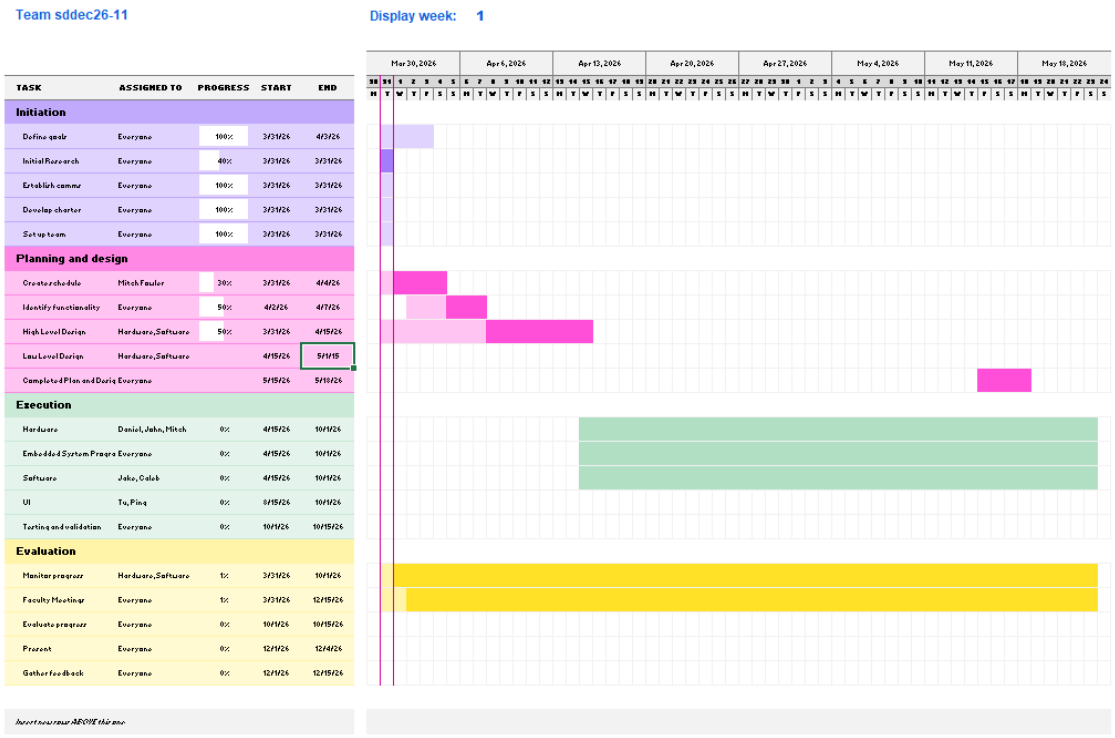


Figure 1 Project Timeline table

See [Team website](#) for more detailed view

3.5 Risks and Risk Management/Mitigation

For the CyFlyBot project, identifying and proactively managing risk is essential due to the technical complexity of integrating two distinct robotic platforms, the reliance on wireless communication, and the constraints of two semester timeline. The following section outlines the key risk the team

has identified, along with their estimated probability and the mitigation strategies that will be employed.

Risk	Probability	Severity	Mitigation
Communication Failure	0.7	High	Early communication testing; retry/timeouts; safe-hold behavior when links drop.
Object Detection Accuracy	0.5	High	Varied-lighting dataset; threshold/flag low-confidence detections for operator verification.
UGV Navigation Failure	0.4	High	Sensor fusion; obstacle-avoidance fallbacks; extensive simulation and controlled testing.
API Incompatibility	0.4	Medium	Contract-first interface; early cross-team reviews; examples/tests; integration checkpoints.
UI Usability Issue	0.3	Medium	Early user feedback; prototypes; UI/UX standards; refinement sprint.
Team Communication	0.3	Medium	Weekly meetings with minutes; PR reviews; shared documentation; integration lead.

Figure 2 Risk table

Risk 1: Communication Failure

The most critical risk facing the CyFlyBot project is communication failure. Wi-Fi interference in warehouse environments could cause latency or significant packet loss between the UAV, UGV, and HQ, hindering real-time coordination and severely impacting system functionality.

To mitigate this risk, the team will conduct early communication testing in environments with a simulated Wi-Fi interface during early testing to identify issues. The system will implement fallback protocols, including message queuing with timeouts and retry logic. If latency is high, both robots will enter a safe hold state until communication is restored.

Risk 2: Object Detection Accuracy

The computer vision pipeline responsible for identifying inventory items may fail to achieve the target 75–80% accuracy under real warehouse lighting conditions or with the specific demonstration items. This would impair the UAV's ability to reliably locate and identify targets.

To address this risk, the team will begin collecting a detailed dataset of target inventory items under various lighting conditions. The team could also implement an approach where detection below a confidence threshold will be flagged for human verification through the user interface,

allowing the operator to confirm or correct the identification. This will ensure mission success even when the model is uncertain.

Risk 3: UGV Navigation Failure

The ground robot may fail to achieve the 85% navigation accuracy requirement due to inaccurate maps provided by the UAV, sensor drift on the UGV itself, or dynamic obstacles such as humans that are not accounted for in path planning. The probability is estimated to be 0.4, with high severity.

Mitigation involves implementing sensors that combine the aerial map data from the UAV with onboard UGV sensors to improve accuracy. The navigation stack will include obstacle avoidance algorithms, such as potential fields or dynamic window approaches as fallback when unexpected obstacles appear. Extensive navigation tests will also be conducted in controlled environments before progressing to full warehouse simulation.

Risk 4: API Incompatibility

The API developed by the hardware subgroup for controlling the UAV and UGV may not be sufficiently abstracted, causing integration delays when the software subgroup implements mapping and path-planning functionality.

To mitigate this risk, the team will adopt a contract-first API development approach. The API interface will be defined in an early sprint with input from both hardware and software subgroups. Weekly integration checkpoints will verify compatibility as development progresses. Documentation and example usage code will be created for each API call to ensure clarity and ease of use.

Risk 5: UI Usability Issue

The operator dashboard may fail usability testing, making it difficult for non-technical warehouse staff to monitor robot status, assign tasks, or manually override behavior.

To mitigate this risk, the team will involve potential users early in the UI design process. Paper-prototype testing will be conducted before full implementation to gather feedback on layout and workflow. Established UI/UX design principles will be followed (clear visual hierarchy and minimizing clicks for common tasks). A UI refinement sprint will be planned based on feedback from initial testing.

Risk 6: Team communication

With subgroups working on hardware, software, and UI, integration points may be missed or delayed due to insufficient communication or documentation.

The team will enforce weekly in-person meetings with clear agendas and distributed meeting minutes. Subgroups will also meet regularly to discuss cross-cutting work, and each member will contribute a minimum of 6 hours per week (including meetings). GitLab will be used for code integration with mandatory pull-request reviews to ensure code quality and cross-subgroup awareness. SharePoint will contain up-to-date documentation for all subsystems. Discord will support communication during independent work hours. Finally, a designated integration lead will track cross-subgroup dependencies and ensure alignment.

3.6 Personnel Effort Requirements

Task	Hours	Explanation
High Level Design	15	Need to have clear goals for further design
Low Level Design	30	Appropriate documentation for design
CrazyFlie Orientation	20	Learn how the drone works
Cybot Orientation	20	Learn how the ground vehicle works
Crazy Fly and Cybot Coordination	50	Program coordination between the two
HQ Setup	20	Setup interface for user
Mapping Software	50	Needed for factory environment
Computer Vision	50	Needed for drone to navigate environment
UI Design and Implementation	50	Needed for user usage
Unified Coordination	50	All parts must fit together
Testing	100	Further stress testing needed

Figure 3 Effort requirements table

3.7 Other Resource Requirements

The CyFly project requires various hardware resources. A ground robot (iRoomba CyBot) and a Crazyflie quad-copter drone are key components of the project. The drone requires a base board, custom imaging-deck, and an optical-flow deck to meet our expected functionalities. 3D printed or custom-machined materials will also be needed for a drone landing pad on the ground unit and shelving for the project scenario demonstration.

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description
Public health, safety, and welfare	Slight safety risk to workers within the warehouse due to being in the same space as robots; no new dangers to safety of the public.
Global, cultural, and social	No violation of IEEE ethics. Has the same impact as any automation across the world. Could give smaller players a chance to rise.
Environmental	Minimal environmental impact since increased electricity usage would be the biggest potential impact. Potential to increase drone production if widespread adoption occurs.

Economic	With the goal of affordability, this gives smaller operators a chance to increase productivity within warehouse systems.

Figure 4 Broader context table

4.1.2 Prior Work/Solutions

Relevant Information:

Munasinghe, Isuru, Asanka Perera, and Ravinesh C. Deo. 2024. "A Comprehensive Review of UAV-UGV Collaboration: Advancements and Challenges" *Journal of Sensor and Actuator Networks* 13, no. 6: 81. https://doi.org/10.3390/jsan13060081

Kavas-Torris, Ozgenur, Sukru Yaren Gelbal, Mustafa Ridvan Cantas, Bilin Aksun Guvenc, and Levent Guvenc. 2022. "V2X Communication between Connected and Automated Vehicles (CAVs) and Unmanned Aerial Vehicles (UAVs)" *Sensors* 22, no. 22: 8941. https://doi.org/10.3390/s22228941

Kobori, Hiroaki, and Kosuke Sekiyama. 2025. "Mutual Cooperation System for Task Execution Between Ground Robots and Drones Using Behavior Tree-Based Action Planning and Dynamic Occupancy Grid Mapping" *Drones* 9, no. 2: 95. https://doi.org/10.3390/drones9020095

Chaffilla Rafaela, Alvito Paulo, and Basiri Meysam, C. 2025. " Collaborative Infrastructure-Free Aerial-Ground Robotic System for Warehouse Inventory Data Capture" Drones 2025,9,792. https://doi.org/10.3390/drones9110792

Similar Products:

Fortunately, most UGV and UAV coordination systems are reserved for the military. Using it in a factory setting appears to be fairly novel.

Previous Work:

We are not using a previous team's work, as our work is entirely new.

Target Solution/Competition/Pros and Cons:

Since our target solution has relatively little competition on the market, this brings about some unique issues and strengths. For pros, we are able to dictate standards with our decisions. We will be pioneering new grounds. For cons, this also adds a lot of pressure to be able to succeed. Choices will have impacts on whether this concept takes off.

4.1.3 Technical Complexity

From the software perspective, the CyFlyBot system presents many different aspects of technical complexity.

The core challenge of our project is coordinating three main parts over a shared wireless network in real time; the HQ, the UGV, and the UAV. Each agent has its own set of ROS 2 nodes, and the system depends on all of them exchanging messages with low latency and no message loss. This kind of coordination provides a systems engineering problem that requires careful node design, architecture, and interface contracts.

To manage this, the team defined a dedicated shared interface package (cyfly_interfaces) that will establish the message, service, and action types used across the entire system. This is the software

"contract" between components. Defining this upfront will prevent integration failures that would be present from incompatible data formats between the drone, ground robot, and HQ.

On the perception side, the UAV must perform real-time ArUco marker detection from a moving camera platform and return location data to the HQ. This involves camera calibration, pose estimation, and image processing running during flight, where there can be complications due to vibration and lighting variation.

The UGV navigation problem requires the system to build a map of its environment in real time using SLAM (Simultaneous Localization and Mapping) and plan paths through that map, and then replan when obstacles are encountered.

Finally, the entire development environment is containerized using Docker and ROS 2 Humble so that all team members develop and test in an identical environment. This reduces environment related bugs but requires every developer to understand Docker and ROS 2 building (colcon, ament).

4.2 Design Exploration

4.2.1 Design Decisions

Decision: Software Communication Framework – ROS 2

The team selected ROS 2 as the primary software framework for the CyFlyBot system, governing all inter-node communication, sensor integration, navigation, and mission coordination across the UGV and HQ computing device. We evaluated the requirements that the framework needed to satisfy. It needed to support real-time publishing and subscribe to communication between multiple nodes running on different machines. It allows integration with established robotics packages for mapping, navigation, and localization rather than requiring implementing these from scratch. It also needed to support the Crazyflie's integration pathway.

Adopting ROS 2 as the framework has several direct effects on the project. The framework provides packages like slam_toolbox, Nav2, and AMCL that expose well-defined topic interfaces for navigation and localization. It also provides a natural integration point for the HQ mission manager: because all agents on the network communicate over the same publish-subscribe layer, the HQ node can dispatch goals to the drone and the ground robot using the same mechanism. The ROS 2 decision significantly reduces the team's software development burden while still giving the team full control over mission logic and inter-robot coordination through custom nodes.

Decision: UGV Navigation and Path Planning Stack - Nav2 with SLAM Toolbox

The team needed to select a navigation and mapping strategy for the UGV that would allow it to autonomously traverse warehouse aisles, localize itself within the space, and plan collision-free paths to target shelf locations. This required choosing both a SLAM solution for map building and a navigation framework for path planning and obstacle avoidance.

Nav2 is the standard autonomous navigation framework for ROS 2. It provides a complete navigation pipeline: global path planning, local trajectory following, and recovery behaviors. SLAM Toolbox is a ROS 2 native SLAM implementation that supports both online mapping (building a

map in real time) and map persistence (saving and reloading a map across runs). Together they form a fully integrated navigation stack without requiring any component to be built from scratch.

The team considered a one-time static map approach, pre-mapping the demo space once and relying on AMCL for localization, as a simpler alternative. However, this would make the system brittle to any changes in the demo environment layout. SLAM Toolbox with map persistence offers nearly the same convenience: the space can be mapped once, the map saved, and subsequent runs load directly from that saved map while still supporting updates if the environment changes.

Pros:

- Native ROS 2 support, actively maintained alongside Nav2
- Online SLAM with map persistence: map once, reload for future runs
- Full Nav2 integration: global planner, local controller, costmaps, and recovery behaviors included
- Full Gazebo simulation support allows the team to develop and tune navigation entirely in simulation before hardware deployment
- Large active community and well-documented configuration parameters

Cons:

- Nav2 has a large number of configurable parameters across its planner, controller, costmap, and recovery behavior layers. Tuning these for the specific demo environment will require dedicated testing time
- SLAM accuracy depends on the quality of the UGV's odometry data, which has not yet been validated on the physical platform

The team selected Nav2 + SLAM Toolbox because no other option offered online SLAM, map persistence, and native ROS 2 Nav2 integration simultaneously. The configuration complexity is a known cost but is well-documented and is a one-time tuning effort for the demo environment.

Decision: Computer Vision Approach – ArUco Markers

The system requires a reliable method for the UAV (and later UGV) to identify inventory items and shelf locations from its camera. Our team needed to choose a computer vision strategy that was accurate, deterministic, and feasible to implement within the project's timeline and constraints. The team evaluated multiple computer vision approaches for inventory identification: ML-based detection (YOLO, SSD) requires significant training data and GPU compute; color/shape recognition is fragile under variable warehouse lighting; and QR codes require a reader held at a specific angle. ArUco markers were selected because they provide deterministic, reliable identification without training data, deliver a full 6-DOF pose estimate (position and orientation) from a single camera frame, and are lightweight enough to run in real time without dedicated GPU hardware. This directly satisfies two system needs: inventory item identification via the UAV's forward-facing camera scanning shelf faces, and precision landing via the downward-facing camera detecting the ArUco marker pad on the UGV roof. The primary trade-off is that ArUco markers must be physically placed on inventory items and shelf locations in advance. This is an acceptable constraint for the controlled warehouse demo environment.

Decision: Development Environment – ROS 2 + Docker Container

The team needed a development environment strategy that would allow all members to work in a consistent environment regardless of operating system, minimizing time lost to environment specific bugs.

The team selected Gazebo Classic 11 as the simulation environment based on its deep integration with ROS 2, full physics and sensor simulation (LiDAR, cameras, differential drive), and an extensive community plugin library. All team members develop and test within a Docker container based on the official ROS 2 Humble image, ensuring an identical build environment regardless of host operating system. This eliminates environment-specific bugs and allows the full simulation stack (Gazebo, Nav2, ArUco detection, SLAM Toolbox) to run identically on every machine. The current Gazebo world includes a warehouse environment with shelf structures, aisle geometry, and collision obstacles, allowing the software team to develop and validate navigation, object detection, and coordination behavior before physical hardware is available. On Windows hosts, an X server (VcXsrv) is required for Gazebo GUI rendering. The team also evaluated Webots as an alternative but found Gazebo's native ROS 2 support and existing team investment in the warehouse world to be the stronger choice.

Decision: UAV Aerial robot relies purely on relative/local localization rather than incorporate in-house infrastructure

The team needed a solution to solve the UAV Aerial robot's ability to localize itself within the warehouse environment that also requires minimal infrastructure to implement.

The team decided to incorporate on-board positioning sensors such as a multi-ranger deck with 5 toF lasers pointing in the left,right,forward,backward, and up directions. A optical-flow deck containing an optical-flow sensor and camera to detect the drones distance from the ground. As well, as incorporating a custom imaging-deck containing two cameras which face the UAV's forward and downward directions. Using this particular combination of hardware decks allows the drone to effectively 'sense' it's immediate surroundings (up to 4 meters) to avoid obstacles, and gather environment data to realize it's position while also requiring minimal-to-no extra infrastructure outside of the UAV itself. Such as incorporating external IR lighthouses, which requires setup that can be increasingly complex, expensive, and obstructive in a typical warehouse environment.

4.2.2 Ideation

Option 1 – ROS 2: ROS 2 is the second generation of the Robot Operating System. It provides a publish-subscribe communication model where nodes running on different machines can exchange typed messages over named topics across a shared network. It provides direct access to packages like slam_toolbox for occupancy grid mapping, Nav2 for autonomous navigation, and Crazyswarm2 for Crazyflie integration.

Option 2 – ROS 1: ROS 1 is the predecessor to ROS 2. It shares many of the same core concepts but uses an older architecture that lacks native support for real-time communication and multi-robot coordination. Many legacy robotics packages were originally written for ROS 1. ROS 1 reached end of life in May 2025 and no longer receives updates or patches.

Option 3 – Custom Middleware Using MQTT: MQTT is a lightweight publishing subscribe messaging protocol originally designed for IoT devices and low bandwidth networks. Rather than adopting a full robotics framework, the team would design a custom middleware layer where each node publishes sensor data and receives JSON commands. This approach would give the team complete control over the message format and communication architecture with no framework overhead. The team would also have to implement all the path planning, navigation, mapping, and localization functionality from scratch.

Option 4 – MAVLink with a Ground Control Station: This is a lightweight binary messaging protocol originally developed for unmanned aerial vehicles and widel used in the drone industry. The architecture would use the protocol for all inter-node communication between the HQ, UGV, and UAV, with a Ground Control Station application. MAVLINK is extremely well suited to UAV command and control. It is natively supported by many flight controllers. However, it was designed primarily for single vehicle aerial systems and does not provide much for ground robot navigation stacks.

Option 5 – Lightweight Custom Framework Using Python Sockets and ZeroMQ: ZeroMQ is a high – performance asynchronous messaging library that provides socket-based communication patterns. This includes publish-subscribe, request-reply, and push-pull. Instead of adopting an existing robotics framework, the team would build a lightweight custom coordination layer using ZeroMQ for all inter-node messaging. This option offers maximum simplicity and portability. ZeroMQ runs pretty much any platform without a complex installation process, and the team has full control over every aspect of the system. It would work well for the communication layer between the HQ and the bots.

4.2.3 Decision-Making and Trade-Off

Criterion	Weight	Rationale
Navigation stack availability	25%	Can not afford to implement SLAM, path planning, and localization from scratch.
Drone integration support	20%	The Crazyflie must be bridged into the system. The quality of existing integration tools directly affects UAV software complexity.
Multi-node communication	20%	Three nodes must share data reliably in real time. The framework must handle this natively without custom networking code.

Community support and documentation	15%	The timeline cannot afford extended debugging of poorly documented tools. Active communities reduce risk.
Simulation support	10%	Gazebo simulation is a huge part of software development before hardware is available.
Implementation complexity	10%	Frameworks that require extensive setup, bridging, or custom integration code reduce the time available for mission-critical features.

Figure 5 Decision-making criterion weight table

Option 1 – ROS 2:

Pros: Native support for slam_toolbox, Nav2, and AMCL covering all UGV navigation needs. Crazyswarm2 provides a well-maintained Crazyflie bridge that publishes pos data and accepts flight commands as standard ROS 2 topics. Publish-subscribe architecture handles multi-node communications across the UGV, HQ, and UAV. Provides Gazebo integration with full sensor simulation. Extremely large community with extensive documentation and active maintenance.

Cons: Initial setup and configuration of the full Nav2, and all package stacks have a learning curve. Requires Crazyswarm2 as a bride layer for the drone.

Option 2 – ROS 1:

Pros: Large body of legacy documentation and tutorials. Some older ground bot platforms have better ROS 1 support than ROS 2. GMapping and move_base provide functional navigation alternatives.

Cons: Reached end of life in May 2025 (no longer receives updates or patches). Bridging to ROS 2 via ros1_bridge adds significant integration complexity. Weaker support for real-time multi robot systems compared to ROS 2. Choosing a deprecated framework introduces log-term risk for a project expected to be demonstrated and potentially extended.

Option 3 – Custom MQTT Middleware

Pros: Lightweight and platform agnostic. Works on any operating system without complex installation. The drone's ESP32 already speaks on Wi-Fi and can publish MQTT messages natively. No framework overhead.

Con: No off the shelf, the shelf navigation or mapping libraries. We would need to implement or manually integrate SLAM, path planning, obstacle avoidance, and localization as standalone libraries and wire them together. This represents an increase in development scope that is not feasible within the timeline. No simulation tools.

Option 4 – MAVLink with Ground Control Station:

Pros: Industry standard UAV protocol with excellent Crazyflie-adjacent support through ecosystems. QGroundControl provides a polished operator interface out of the box. Well suited to drone command and control.

Cons: Designed for a single vehicle aerial system. Ground integration is unconventional and poorly documented. Would require running parallel middleware systems for the UGV and UAV and bridging them at the HQ. Significant integration complexity with no clear precedent for the hybrid UGV-UAV architecture.

Option 5 – Python Sockets and ZeroMQ:

Pros: Maximum portability and simplicity. Works on any platform. No installation complexity beyond a pip install. We would have full control over message formats and communication patterns.

Cons: No navigation, mapping, or localization libraries available out of the box. Same fundamental problem as Option 3 (all the robot's functionalities must be implemented from scratch). No simulation integration. Not good for the timeline.

Decision:

ROS 2 was selected as the software framework because it provides the most complete and immediately usable set of tools for the three hardest software problems in this project: mapping, navigation, and multi-agent coordination. The tools that address these. Slam_toolbox, Nav2, and Crazyswarm2 are all native to the ROS 2 ecosystem.

A major secondary reason for selecting ROS 2 is its Gazebo simulation integration. Gazebo allows the software team to develop and validate navigation, perception, and coordination behavior without requiring physical hardware to be available. This is critical for a two-semester timeline where hardware availability is inconsistent.

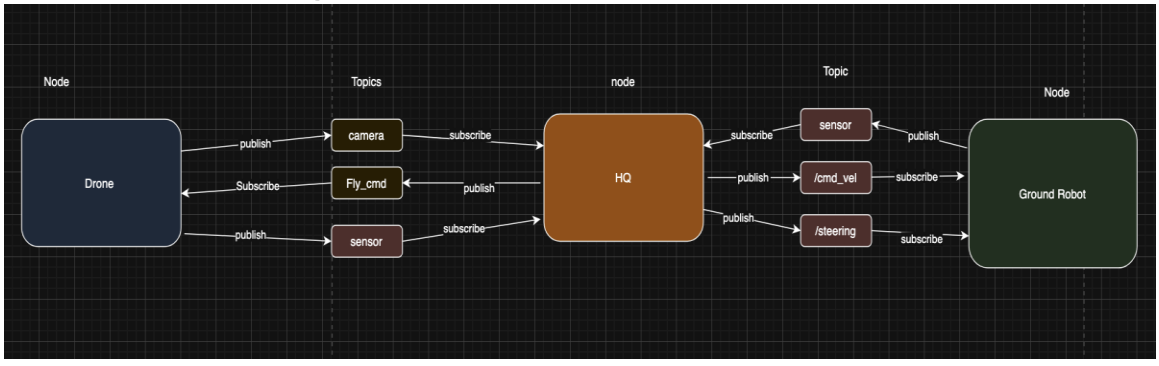
4.3 Proposed Design

4.3.1 Overview

CyFlyBot is a fully autonomous aerial-ground robotic system designed for inventory management in warehouse environments. The system uses a coordinated two-robot setup. A UGV(Unmanned Ground Vehicle) that drives through warehouse aisles, and a UAV(Unmanned Aerial Drone) that flies up alongside shelf faces to locate inventory items by scanning ArUco markers. Both robots are directed by a central HQ computing device running the mission planner.

Figure 6 System behavior overview diagram

The software system is a distributed ROS 2 application organized into seven packages, running across three nodes: the HQ desktop computer, the UGV, and the UAV. All communication routes through the ROS 2 publish-subscribe layer using custom message and action types defined in the cyfly_interfaces package. No direct communication occurs between the UGV and UAV. All coordination is mediated by HQ.



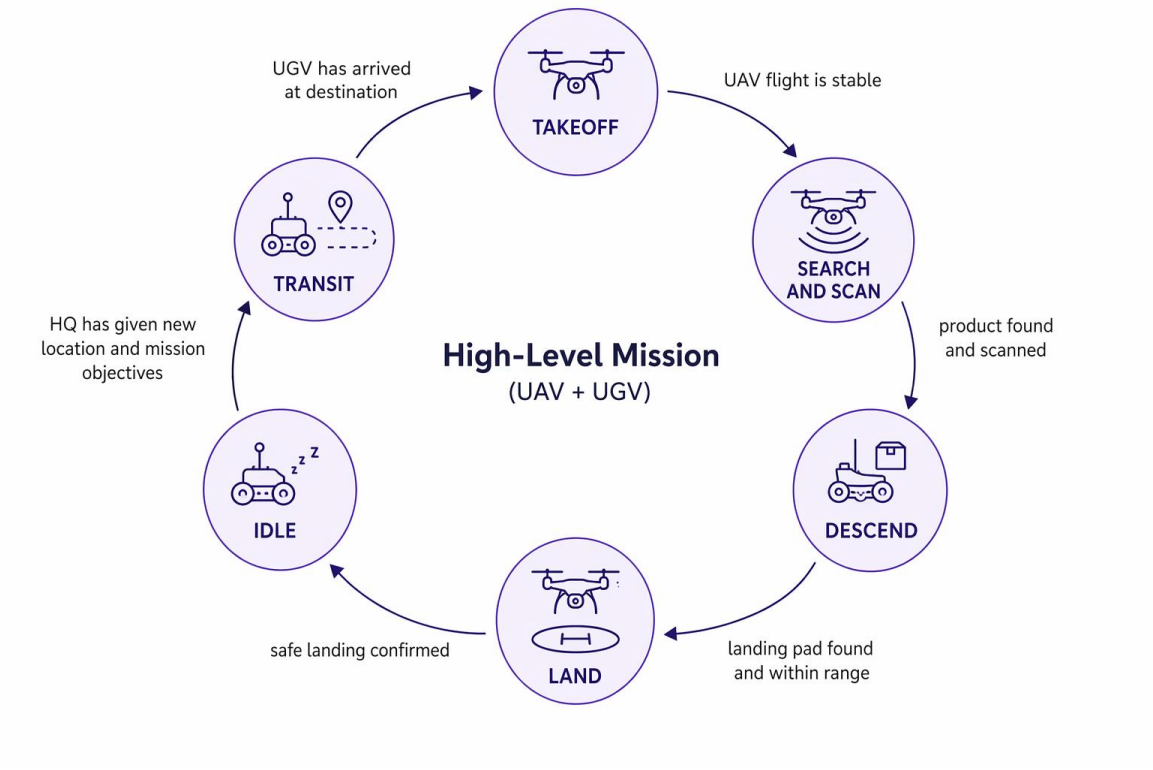
The workspace is structured as follows:

Package	Role

Cyfly_interfaces	Shared message, service, and action definitions. The system contract between components.
Cyfly_description	URDF and mesh files describing both robots (for simulation purposes only)
Cyfly_worlds	Gazebo world files for the warehouse simulation environment
Cyfly_bringup	Launch files for the full system and individual robots
Cyfly_ground	UGV nodes: driving, camera processing, retrieval execution
Cyfly_drone	UAV nodes: flight control ArUco object detection, reporting
Cyfly_hq	HQ coordinator: mission state machine, command dispatch

Figure 7 ROS 2 Package Table

4.3.2 Detailed Design and Visual(s)



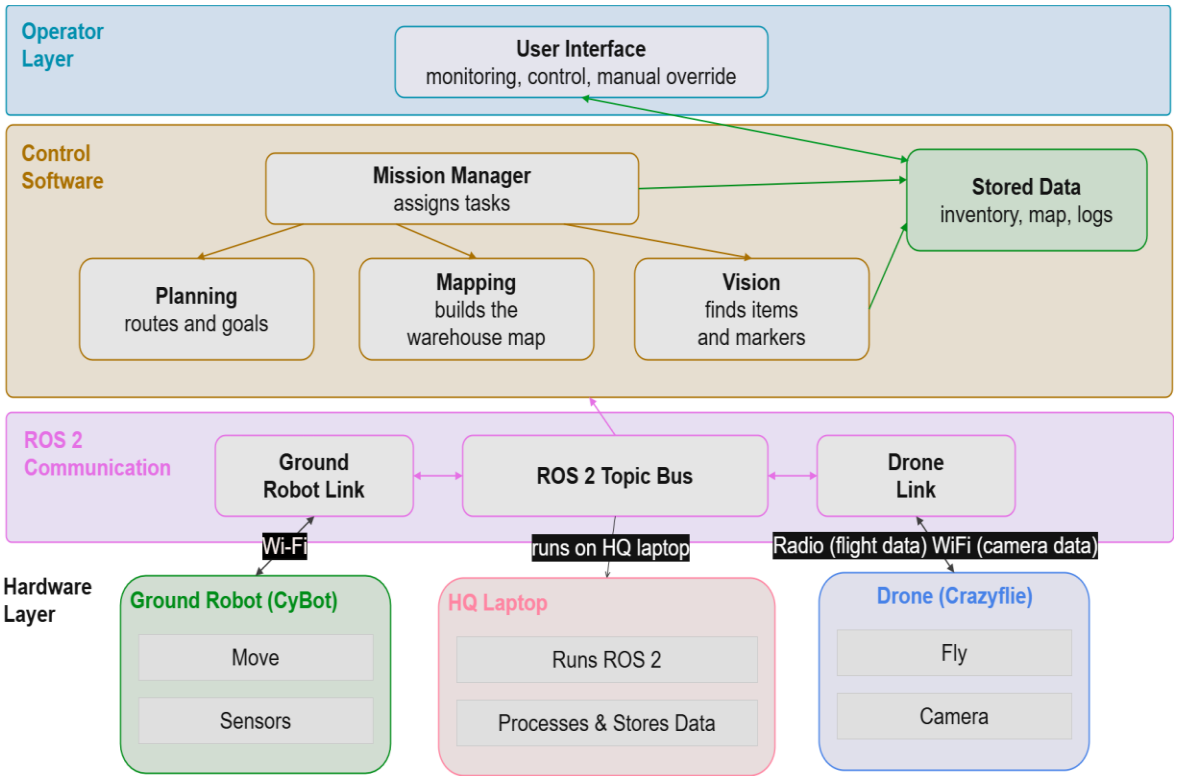
The HQ coordinator (cyfly_hq) is the central authority of the system. It runs a mission state machine that tracks the current phase of operation (IDLE, TRANSIT, TAKEOFF, SEARCH AND SCAN, DESCEND, LAND) and dispatches commands to the drone and ground robot via ROS 2

action servers. It also receives all sensor and detection data from both robots. The operator interacts with the system through HQ, meaning no direct operator-to-robot communication occurs.

The UGV software (cyfly_ground) manages differential drive control via Nav2. It subscribes to goal pose commands from HQ, uses SLAM Toolbox to maintain an occupancy grid of the warehouse, and uses Nav2's planner and controller servers to navigate to target positions autonomously. A camera node also runs ArUco detection for ground-level marker identification.

The drone software (cyfly_drone) manages flight control and aerial perception. It accepts takeoff, land, and waypoint commands from HQ. Its front-facing camera runs an ArUco detection pipeline that identifies inventory markers on shelf faces, estimates their 3D pose, and publishes findings back to HQ. The downward-facing camera detects the ArUco marker on the UGV landing pad, along with distance estimations from the ground.

The full architecture of the system is as follows:



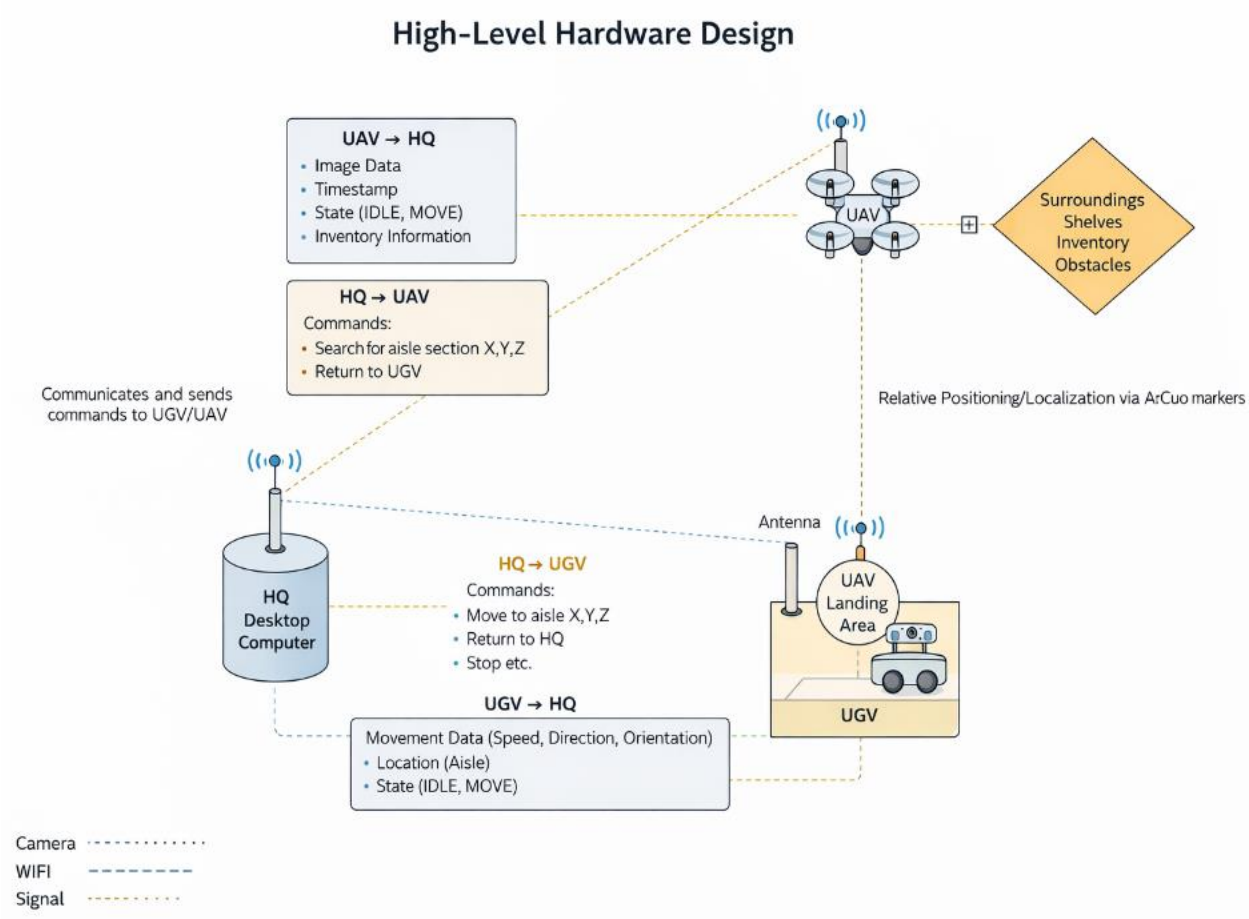


Figure 8 High-Level System block diagram

UAV:

The UAV subsystem of CYFLYBOT is built on the Crazyflie 2.1 Brushless development platform by Bitcraze [2]. It is a palm-sized, open-source quad-rotor drone that weighs about 34 g with landing legs and can carry a payload of up to 40 g - enough to support the full deck stack described in this section. The UAV handles autonomous flight inside the warehouse, including flying up to shelf faces, capturing close-range images of inventory and barcodes, and landing precisely back on the UGV between aisle traversals. Since the UAV operates indoors without GPS, all of its positioning, altitude control, and stabilization come from onboard sensors, add-on decks, and coordination with the HQ device.

The UAV is not meant to fly across entire warehouse aisles on its own - its stock 350 mAh LiPo battery only gives it about 10 minutes of flight time. To cover more ground, the system uses a UGV ferry strategy: the UAV rides on top of the UGV as it drives between aisles, then takes off only for short scanning runs at each shelf. This approach is directly based on the architecture described in Basiri et al. [1] and keeps the UAV focused on short, precise flights rather than long-range navigation.

The UAV subsystem consists of three primary expansion deck modules stacked onto the base flight platform. Each module is sourced from Bitcraze [2] with the exception of the imaging-deck which is a custom design.

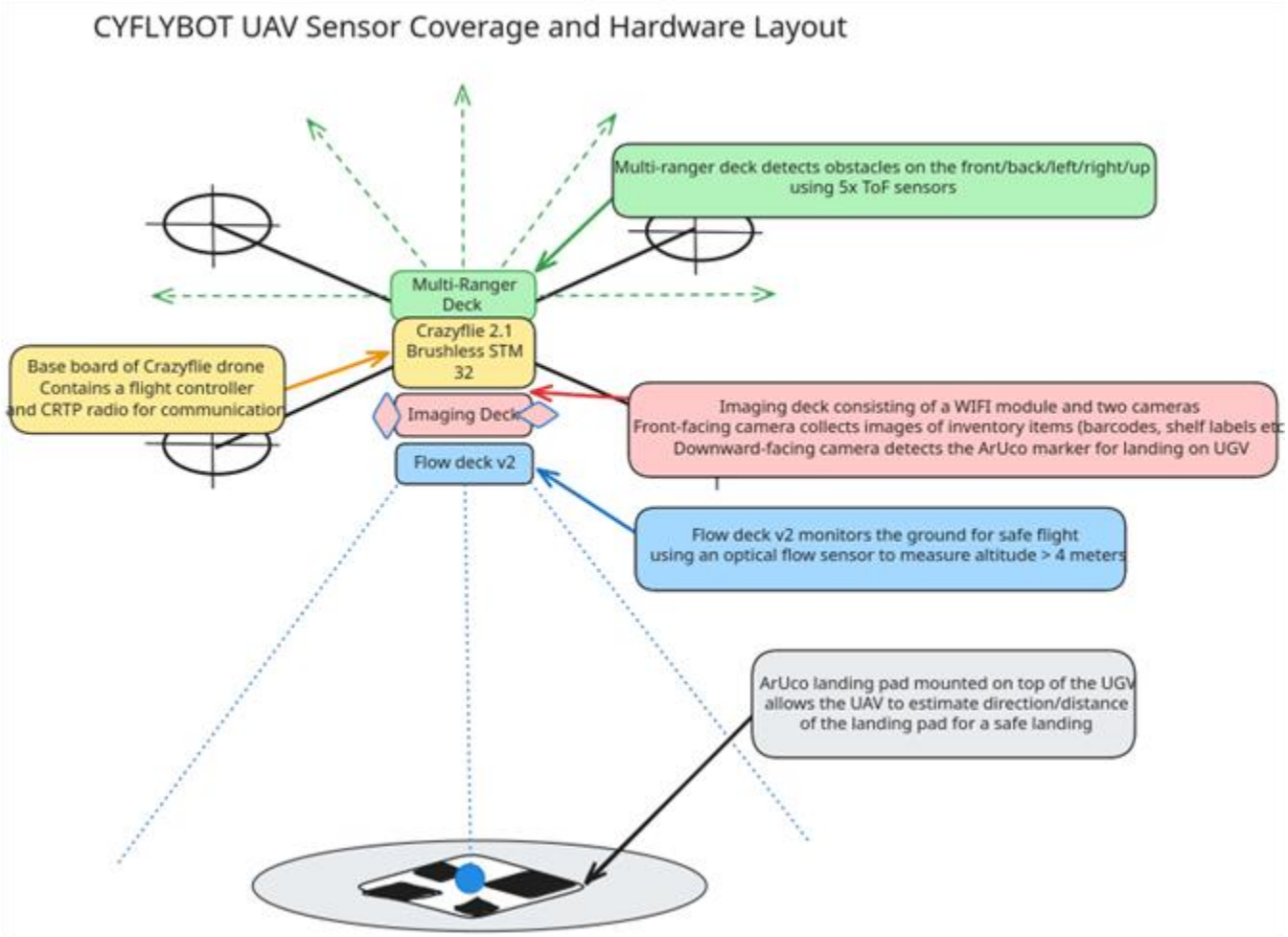


Figure 9 UAV Hardware layout and Sensor Coverage diagram

UAV navigation in CYFLYBOT is split into two layers: low-level state estimation that runs directly on the CrazyFlie STM32, and high-level mission logic that runs on the HQ device.

Low-level state estimation combines data from the IMU (BMI088), barometer (BMP388), optical flow (PMW3901), and altitude sensor (VL53L1x). This gives the drone a stable 3D position estimate - X, Y, and Z – as well as collision/obstacle avoidance without any GPS or external beacons.

Relative positioning and landing with respect to the UGV uses the downward-facing camera and ArUco detection. The UGV landing pad has a ArUco marker. The detected marker pose is passed to the STM32 over UART1, where it gets combined with the UGV's known global position (sent from HQ over Crazyradio) to give the UAV its global location when needed.

Custom Imaging-deck:

Sitting onboard the UAV are three expansion decks, *multi-ranger deck, flow deck v2,* and a custom imaging-deck. The custom imaging-deck is designed as a method of acquiring / transmitting image data back to the HQ. It will involve the use of two cameras, an OV2640 (Downward facing), and an OV5640(Forward Facing) which are operated by an onboard ESP-32-S3 mini microcontroller. The ESP-32 also contains WIFI functionality and will stream image data captured by both cameras back to the HQ. A custom PCB is designed on which the components will be soldered / connected, then assembled onto the UAV itself.

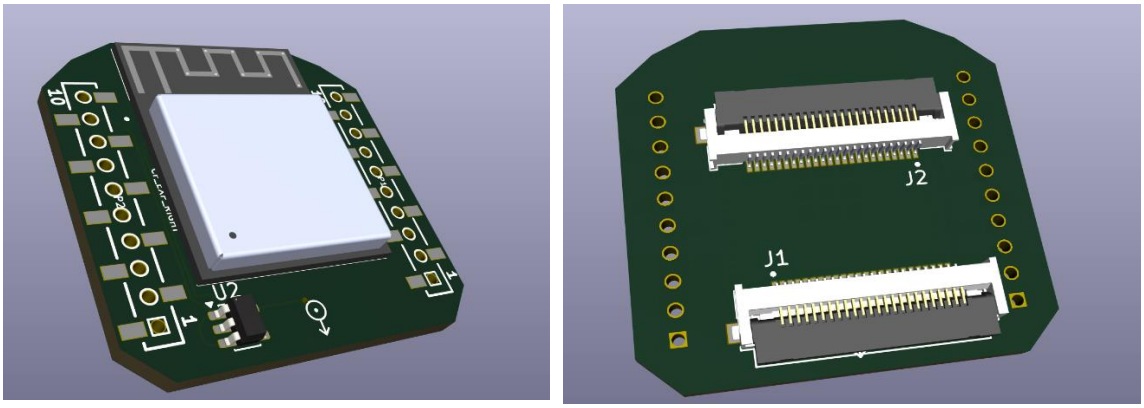


Figure 10 Imaging-deck PCB model (front and back)

The ESP-32-S3 mini microcontroller is mounted on the top, and two 24-pin connectors are mounted on the bottom for interfacing the cameras. (Plans include adding a 6-pin connector to allow programming the ESP-32 microcontroller). Currently, there are some design concerns involving the wiring and operation of two cameras on one microcontroller. In which case, the following behavior is explained.

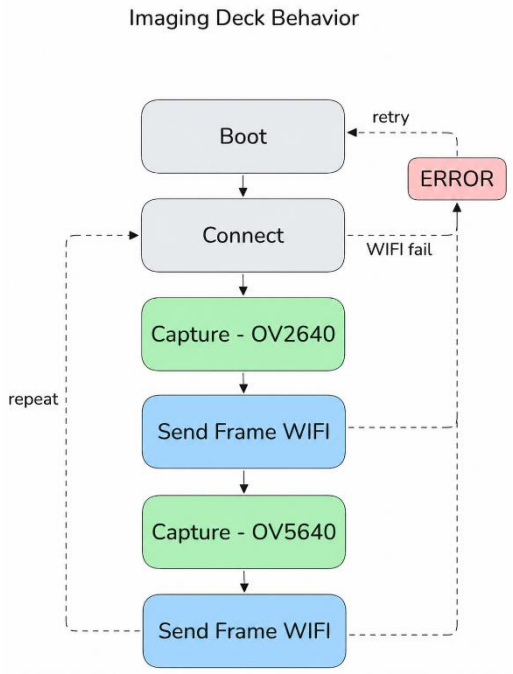


Figure 11 Imaging Deck behavior diagram

The imaging-deck will boot → connect to WIFI (retry if failed) → capture an image from the OV2640 → stream image data over WIFI → capture an image from the OV5640 → stream image data over WIFI → repeat.

4.3.3 Functionality

The operator submits an item retrieval request through the HQ interface, identifying the target item by its ArUco marker ID. The HQ mission coordinator dispatches the UAV with a search command. As the UAV scans warehouse aisles, it detects ArUco markers on shelf faces and reports each found item's ID and estimated location back to HQ. All detected items are logged along the way, building an inventory map as a side effect of the scan, not just tracking the target item. When the target is found, HQ dispatches a goal pose to the UGV. The UGV uses its SLAM-built occupancy grid and Nav2 to plan and execute a path to the target shelf. Upon arrival, the UGV retrieves the item and signals task completion to HQ, and the UAV returns and lands on the UGV pad using ArUco-guided landing. The operator is notified through the HQ interface.

4.3.4 Areas of Concern and Development

The software design as proposed is architecturally sound and grounded in well-established, widely-used tools. However, several areas carry meaningful risk going into the integration and hardware phases.

ArUco detection reliability under real conditions.

The ArUco detection pipeline has been implemented and tested in controlled conditions. Real warehouse environments may introduce challenges including variable lighting, shadows, motion blur from the moving UAV, and partial marker occlusion. The team plans to test detection under varied conditions.

Drone simulation to hardware transition.

The current UAV in Gazebo is a simplified placeholder model capable of basic flight. Transitioning to coordinated HQ-dispatched missions with the physical Crazyflie with the radio communication layer is the riskiest integration step in the software roadmap.

HQ operator interface.

The HQ UI design is currently TBD. the leading candidate is a web-based dashboard. This will be developed once the core HQ coordinator logic is stable and tested.

The team's immediate next steps are to integrate SLAM Toolbox and validate mapping in the Gazebo warehouse world, bring up Nav2 and test autonomous UGV navigation.

4.4 Technology Considerations

1. Communication Framework Technology: ROS 2 Humble

Strengths: Built-in support for distributed systems and real-time publish-subscribe communication across multiple machines. Native access to established robotics packages (Nav2, SLAM Toolbox) and full Gazebo simulation integration. The Crazyflie UAV can be bridged into the ROS 2 network through Crazyswarm2, allowing HQ to treat the drone as just another network agent.

Weaknesses: Steep initial learning curve, particularly for Nav2 configuration. Requires careful setup when running across multiple physical machines.

Trade-offs: The development overhead is justified by eliminating the need to implement any navigation, mapping, or coordination from scratch.

2. UGV Navigation and Mapping Technology: Nav2 + SLAM Toolbox

Strengths: Native ROS 2 packages, actively maintained. SLAM Toolbox provides real-time occupancy grid mapping with map persistence. The demo space can be mapped once and reloaded for subsequent runs. Nav2 provides a complete navigation pipeline including global planning, local control, and recovery behaviors. Full Gazebo support enables navigation development entirely in simulation.

Weaknesses: Large number of configuration parameters requiring tuning. SLAM accuracy depends on data quality, which has not yet been validated on the physical UGV.

Trade-offs: Not yet tested on hardware. The configuration burden is real but is a one-time tuning cost for the demo environment, and is outweighed by the elimination of implementing any navigation or mapping functionality from scratch.

3. Computer Vision and Object Detection Technology: OpenCV ArUco Detection

Strengths: Each detected marker returns a unique ID and a full 3D pose estimate (position and orientation relative to the camera). Fully supported by OpenCV with no training data required. Lightweight enough to run onboard in real time (if needed). Directly satisfies the system's need for the UAV to identify inventory items and estimate its position relative to targets and the UGV landing pad.

Weaknesses: Requires physical markers to be placed on all inventory items and shelf locations in advance. Detection can degrade under poor lighting, or motion blur.

Trade-offs: The dependency on physical markers is an acceptable constraint for the warehouse demo scenario. Reliability and pose accuracy make ArUco the clear choice over alternatives such as ML-based detection or color/shape methods within the project's timeline. ArUco detection has been implemented and tested. The node correctly identifies marker IDs and estimates camera-to-marker distance when a known marker is placed at a known distance.

4. Localization and Mapping Technology: SLAM Toolbox (planned, not yet tested on hardware)

Strengths: See Navigation and Mapping above. Designed to integrate directly with Nav2 and the ROS 2 ecosystem.

Weaknesses: Not yet validated on the physical UGV. Sensor quality on the final hardware platform is a variable we are unsure about thus far.

Trade-offs: Not yet tested. Results pending hardware integration phase.

5. Simulation Environment Technology: Gazebo Classic 11 + Docker

Strengths: Standard simulator for ROS 2 with the deepest plugin ecosystem and community support. Full sensor and physics simulation allows the software team to develop and validate behavior before hardware is available. Containerized with Docker, ensuring every team member runs the same simulation environment regardless of host OS.

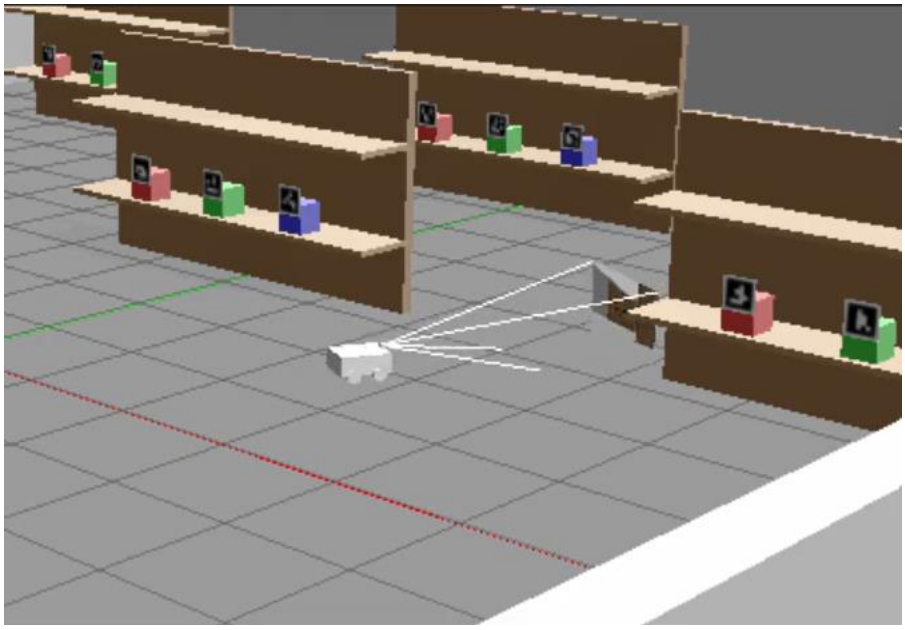
Weaknesses: Gazebo Classic 11 is in long-term maintenance mode, meaning it is not actively being developed. Can be resource-intensive on lower-spec machines. Requires an X server (VcXsrv) for GUI rendering on Windows.

Trade-offs: Despite being in maintenance mode, Gazebo Classic 11 is stable and fully sufficient for the scope of the demo environment. The team's existing investment in the warehouse world, robot URDFs, and sensor configurations made it the clear choice over Webots or other alternatives.

4.5 Design Analysis

On the software side, the team has established the full ROS 2 workspace structure and a functional Gazebo simulation environment. The workspace is organized into seven packages and builds cleanly within the Dockerized development environment, providing the architectural foundation that all future development will build on.

The ground robot simulation is the most mature component at this stage. The UGV spawns correctly in the warehouse Gazebo world, responds to velocity commands via the ROS 2 differential drive interface, and publishes image data through its camera. An ArUco marker detection node has been implemented and tested, correctly identifying marker IDs and estimating camera-to-marker distance when the size of the ArUco markers are known.



The drone model is present in simulation and capable of basic flight via ROS 2 topic commands, but currently serves as a simplified placeholder. Its ArUco detection pipeline has been implemented and tested in a standalone environment.

The system has not yet been run end-to-end. The HQ coordinator, drone, and UGV nodes have not been tested together in a full mission loop. However, each major piece of the proposed software stack has been thoroughly researched and is grounded in tools that are widely used in similar robotics applications. No fundamental feasibility concerns have been identified. The remaining risks are in integration and hardware tuning rather than in the technology choices themselves.

Immediate next steps:

- Integrate SLAM Toolbox with the UGV simulation and validate mapping in the warehouse world
- Bring up Nav2 and validate autonomous navigation to a target in simulation
- Implement the HQ mission state machine and connect it to drone and UGV nodes via the defined cyfly_interfaces actions
- Begin planning for the transition from simulation to physical hardware, starting with UGV ROS 2 integration

5 Testing

The testing plan should connect the requirements and the design to the adopted test strategy and instruments. In this overarching introduction, give an overview of the testing strategy and your team's overall testing philosophy. Emphasize any unique challenges to testing for your system/design.

- Is our testing plan unique to our project? (It should be)
- Are you testing related to all requirements? For requirements you're not testing (e.g., cost related requirements) can you justify their exclusion?
- Is your testing plan comprehensive?
- When should you be testing? (In most cases, it's early and often, not at the end of the project)

5.1 Unit Testing

Software:

- Testing the ground bot movement in isolation using teleop keyboard and verifying it responds correctly to cmd_vel commands on the gazebo simulation.
- Testing the LiDAR sensor by checking /ground_bot/scan topic is publishing valid data.
- Testing camera sensor publishes valid image data
- Testing the ArUco detection node correctly identifies marker IDs and returns accurate distances when a known marker is placed at a known distance.

5.2 Interface Testing

Software:

- We tested the interface between the camera and ArUco detection node. Does the image data coming from Gazebo get correctly processed by the detection algorithm.
- We tested the interface between the ground bot and ROS 2. Check to see if the different drive plugin correctly translates velocity commands to wheel movement.
- Tested that the ArUco node publishes the correct ROS topic with ID and distance information that other nodes can consume.

5.3 Integration Testing

Software:

The main critical integration paths in our design are the HQ to drone and ground bot communication. The HQ coordinator must be able to dispatch the drone and ground bot with a search command to assign the bots to a mission. The HQ must also receive back the location of a

detected object with information about the item. This will be tested by sending a simulated retrieval request from the HQ node and verifying the drone and ground bot node responds. All integration testing will be conducted within the Gazebo simulation environment using ROS 2 topic monitoring tools such as ros2 topic echo to verify data is being published and received correctly between nodes.

Hardware:

The UGV and UAV need to be able to handle commands sent from the HQ. This will be done using formatted String messages that will be parsed within the UGV and UAV's systems. To ensure correct message receive, various mock messages will be used to simulate a command from the HQ and be dissected by the UGV and UAV's systems to properly resolve the commands. Improperly formatted messages will also be tested to verify error detection.

5.4 System Testing

System level testing for the CyFlyBot project involves running the complete simulation with all three components, the ground bot, drone, and HQ coordinator. They will be active simultaneously and verifying the full end-to-end identification workflow functions correctly.

A successful system test is defined as follows: the HQ coordinator receives a mission request, dispatches the drone and ground bot to go find items to check location, stock, or inventory, the drone reports given information from ArUco tags back to the HQ to complete the mission. After the team completes the testing on the simulation for the software, testing will begin on the live demo with hardware implemented.

5.5 Regression Testing

Software:

We are working in a GitLab repository for gazebo simulation. This allows us to work off branches when implementing new code or testing big changes to one of the bots. Working off branches allows us to make these changes without having to worry about breaking the entire project. If a team member ends up breaking the project, they will simply not merge their broken branch with the main branch. Allow, every time a teammate pushes new code, we can launch the simulation and verify the ground bot is still working correctly.

Hardware:

The command parsing function will be retested before connection to the HQ to ensure that new implementations can be properly tested, and that test failures will not be a result of the already implemented system. The UAV's launch and landing functionality will also be retested to prevent UAV failures from occurring when connecting to the HQ.

5.6 Acceptance Testing

Software:
For the software team is currently working in a gazebo simulation to test out functionality. This allows us to show proof of functionality and non-functionality by trying ideas out in the simulation before testing the ground bot itself at the moment. The gazebo simulation is also run with ROS 2. This is helpful to us because a requirement we have for this project is to get the UAV and UGV communicating to complete a task. ROS 2 what we are using to get the UAV and UGV communicating and working together. In the future we will be creating a demo of a set up warehouse space and the UAV and UGV navigating through the environment. This will provide a full live demo to our clients which will show the final product of our project.

5.7 Security Testing (if applicable)

5.8 User Testing

Describe how the team will include real users to interact with the solution/product. Your user testing plan should evaluate your solution/product's usability and overall user experience by gathering user feedback through observations, surveys, etc. User feedback allows the team to identify areas for improvement and ensure the product meets user needs.

We plan to initially conduct user testing ourselves to refine our functionality and experience. When the design meets a more complete standard, our clients, Dr. Rover and Dr. Jones, will be given access to our design to provide user feedback.

5.9 Results

What are the results of your testing thus far? Include any numerical, graphical, or qualitative testing results here? How do they demonstrate compliance with the requirements or addressing user needs? Use a summary narrative to discuss what you've learned and what next steps need to be taken.

Software:

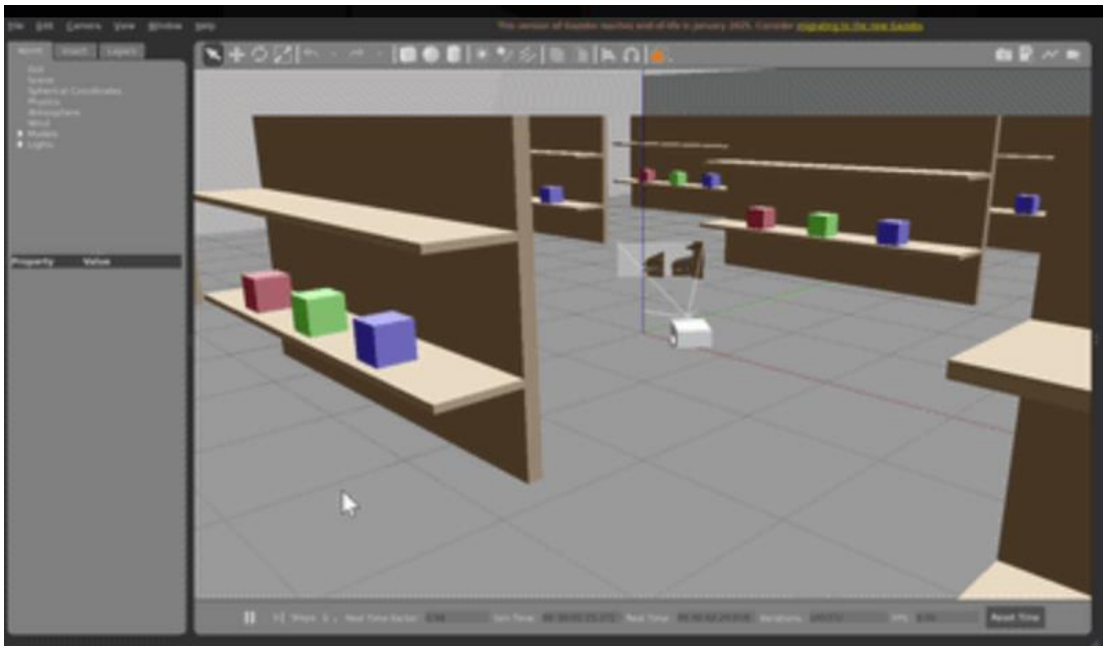


Figure 12 UGV within Gazebo simulation

This is the result of our gazebo simulation for the CyFlyBot warehouse system. The simulation demonstrates a fully function 3D warehouse world consisting of shelves and items. The groundbot is visible in the simulation and equipped with a front facing camera used for testing ArUco marker identification. The simulation environment has met several key testing milestones. The ground robot successfully spawns in the warehouse and responds correctly to velocity commands via the ROS 2 differential drive interface. This confirms that the unit level movement functionality is working as expected. The ground bot also has a working LiDAR sensor that is used for testing different mapping and localization techniques. Also, the camera sensor has been verified to publish image data, and ArUco marker detection has been implemented and tested. This will allow us to identify objects and how far away the camera is away from the tag.

The warehouse world itself provides a realistic testing environment with physical walls, shelf structures with collision geometry, and interactable objects. This allows the team to test navigation, object detection, and localization in a controlled setting before deploying physical hardware.

The next steps include testing SLAM toolbox integration to enable autonomous mapping and localization. Also, bringing up the drone and HQ coordinator nodes to test the full end to end workflow simulation.

6 Implementation

Our team is currently in the early implementation stage of the CyFlyBot project. We have not fully built or tested the complete UAV and UGV system together yet, but we have made progress on the main software setup, early simulation work, computer vision testing, a mostly-finished UAV design,

and possible ground robot hardware. At this point, our goal has been to create a foundation that we can keep building on as the physical system becomes more complete.

Figure 13 UGV within Gazebo simulation

So far, our team has created a Gazebo simulation using ROS 2. The simulation represents a simple warehouse-style environment with shelves, aisles, and colored objects that can act as inventory items. This gives us a safe way to test ideas before using real hardware. Instead of immediately testing the drone or ground robot, we can first use Gazebo to experiment with movement, object placement, scanning behavior, and basic navigation. ROS 2 is important to our implementation because it will be the main system used for communication between the UAV, UGV, and headquarters computer. In the final version, ROS 2 will allow different parts of the system to send information such as robot position, target locations, sensor data, and task status.

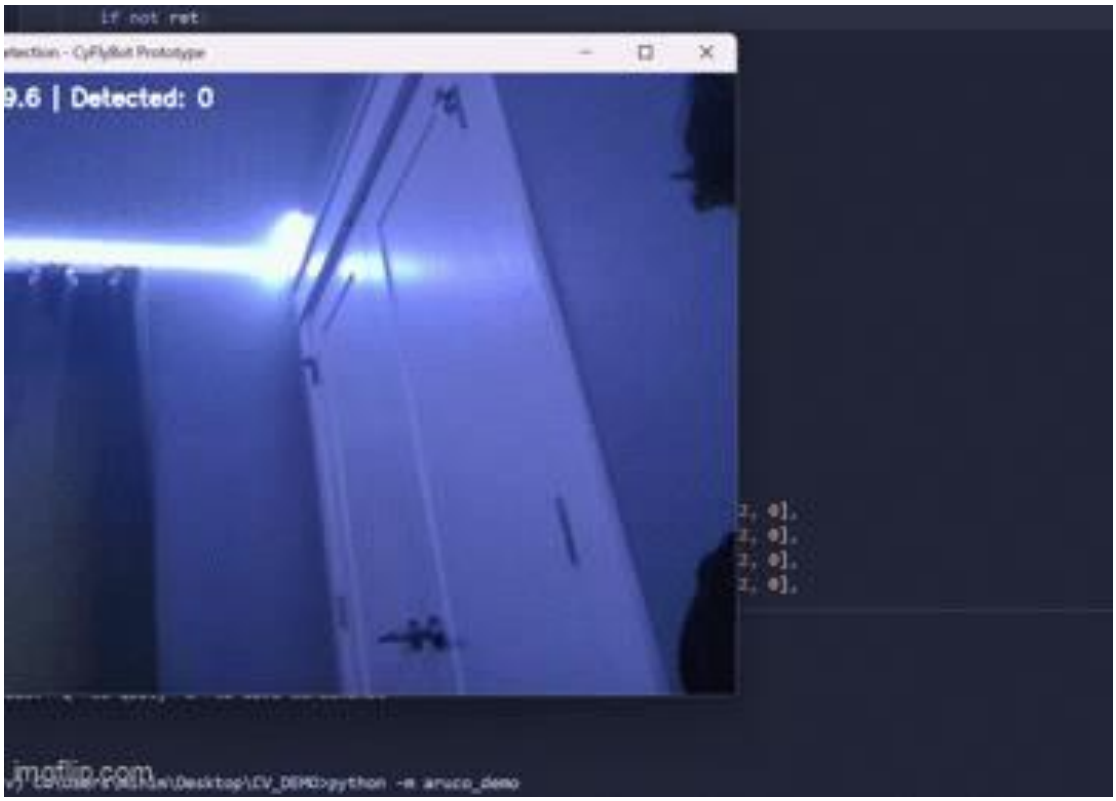


Figure 14 ArUco markers being detected

Our team has also started looking into ArUco markers as part of our computer vision approach. The GIF above shows marker detection in progress, which gives us a starting point for inventory scanning and item identification. In our project, the ArUco markers would potentially be used to assign IDs to inventory items or shelf locations so the drone can scan the environment and recognize what item it is looking at. Once the UAV detects the correct marker, it can send that item/location information back to the rest of the system, including the headquarters computer and the ground robot. This part is not fully connected to the complete system yet, but it shows that marker-based detection could be a practical way for the UAV to identify inventory and help guide the UGV toward the correct item.

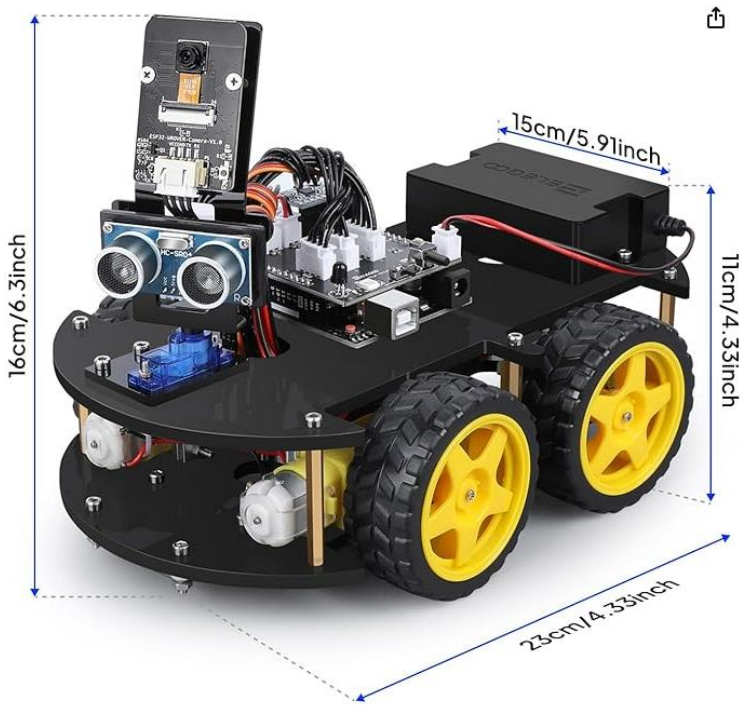


Figure 15 Elegoo Smart Robot Car v4.0 product image

For the ground robot, the primary UGV platform is the CyBot (iRobot Create 2 base with a TM4C123 microcontroller), which the team has instrumented with PING sonar, IR distance sensors, a servo-mounted scanner, and UART communication. Firmware has been developed and tested for both remote-control and autonomous modes. The team has also evaluated the ELEGOO Smart Robot Car V4.0 as a possible secondary platform due to its integrated Wi-Fi, video transmission, and Arduino-based control, which may offer a simpler path to ROS 2 integration during the upcoming semester. At this stage, the CyBot remains the primary hardware target, and the ELEGOO is being considered as a fallback or supplemental option if hardware-level ROS 2 integration proves more complex than anticipated.

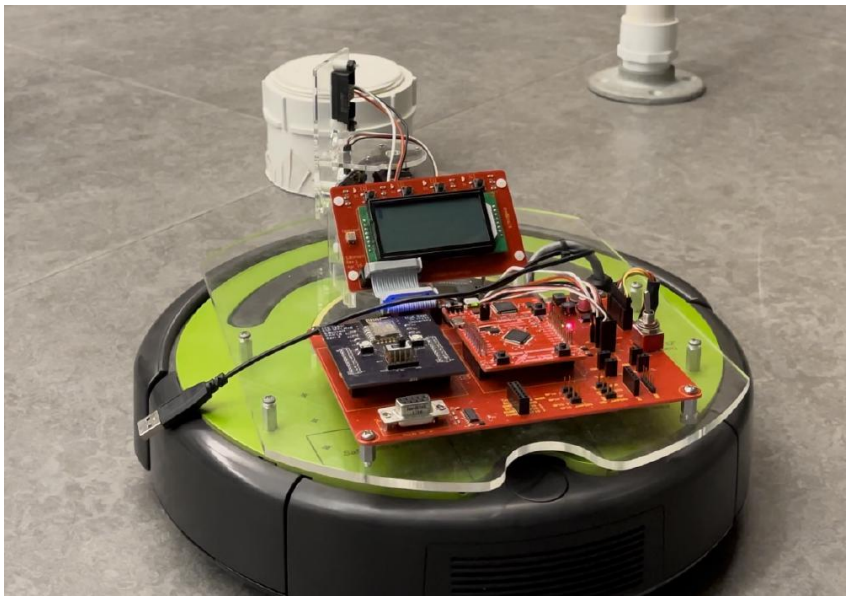


Figure 16 CyBot UGV image

For the aerial drone UAV utilizing the CrazyFlie platform, the hardware design is nearly completely finalized. We have selected and confirmed basic functionality of two of the three proposed hardware decks, including the Multi-Ranger Deck and the Flow Deck v2, which are currently available to the team for further development and experimentation. The custom Imaging Deck design is currently still in progress, with the basic requirements and functionality decided.

Figure 17 CrazyFlie Brushless 2.1 Drone image

Overall, our implementation so far has focused on building and testing the foundation of the project. We have a ROS 2/Gazebo simulation started, a possible ArUco computer vision approach, and a physical ground robot platform that may be easier to modify. Our next steps are to improve the simulation, connect more ROS 2 topics, test basic movement commands, continue ArUco detection work, and begin integrating the physical CyBot UGV into the ROS 2 software environment.

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	Definition	Reference	Example from Project
Work Competence	Produce high quality work and contribute within our domain	ACM Code of Ethics 2.6	Team members are designated to sub teams in relation to their skillset.
Financial Responsibility	Operate within the client's budget and only make necessary expenses.	ACM Code of Ethics 2.1	Bill of materials collected before purchase to ensure our budget is maintained.
Communication and Honesty	Report work with transparency in an understandable manner.	ACM Code of Ethics 1.3	Progress is reported to clients weekly and highlights individual contributions.
Health, Safety, and Well-being	Minimize safety risks for stakeholders and operators.	ACM Code of Ethics 2.9	Design includes methods of obstacle evasion to limit collisions.
Property Ownership	Respect equipment not developed by the team.	ACM Code of Ethics 1.5	Additional components are expected to be removable from systems not owned by the team.
Sustainability	Efficient use of resources to reduce impact on the environment.	ACM Code of Ethics 3.1	Design uses minimal resources to implement an efficient system.
Social Responsibility	Enhance the reputation of engineers by acting responsibly and lawfully.	ACM Code of Ethics 3.4	Team members follow our core virtues to uphold an ethical standard.

Figure 18 Code of Ethics table

The team has excelled in maintaining the responsibility of health, safety, and well-being. A main consideration for our design has been obstacle avoidance and that has overlapped with creating a safe design that avoids causing harm to users and bystanders.

The team has considered sustainability very minimally throughout our design process. To improve upon this, the team can evaluate our resources to more efficiently manage additional effects of implementing our design.

7.2 Four Principles

	Beneficence	Non-maleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	Design targets tasks with safety risks	Safety risks are prevented through our system design	Gives users the ability to limit area of operation	Improves safety for low level workers
Global, cultural, and social	Performs less desirable tasks for workers	Minimal expertise required for operating	Provides an automated option to complete tasks	Allows smaller companies to automate warehouses
Environmental	Base design relies on minimal resources	Design avoids the necessity of unsustainable materials	Design does not include an eco-friendly alternative	Design does not disturb the environment
Economic	Design elevates job responsibility	Affordable design for smaller operators	Design can be scalable for larger use	Design does not aim to replace workers

Figure 19 Four ethical principles table

The team places importance in maintaining nonmaleficence in public health, safety, and welfare. Our design considers the risks of unattended UAV use and has made decisions on components such as the multi-ranger deck that help avoid obstacles, including people.

Our design does not adhere to an environmental respect for autonomy. The CyFlyBot does not aim to provide an eco-friendly alternative, but the overall design is planned to rely on minimal resources and avoid unsustainable materials to trivialize the need for an eco-friendly alternative.

7.3.1 Virtues

Safety – Do not cause harm

The team has reviewed the dilemma of automation: robots must improve efficiency of flow without causing hazards. Our design is intended to be used for high-risk tasks to handle height risks and repetitive strain. The UAV will have real-time obstacle detection using sensor decks and camera scans to avoid workers. There is implementation for controlled behavior and no fly zones to avoid the UAV navigating into unwanted areas. All UAV behaviors are confirmed in a Gazebo simulation before physical deployment.

Accessibility – Usable by as many people as possible

Our team plans to make our design affordable for small operators. Due to automation systems being expensive, smaller companies are excluded from access to these types of systems. We will be

making our design fully open source; all the code and documentation will be publicly available. Our system is also run on commodity tools such as ROS 2 to be vendor-independent and thus more affordable. Our user interface will not require any robot piloting experience to operate the CyFlyBot and will be accessible to non-technical users.

Usefulness – Serve a meaningful purpose

The team has reviewed another aspect of the automation dilemma: automation could eliminate jobs at scale. Our goal isn't to slow progress, but to be intentional about what automation can be used for. Our design targets repetitive, ergonomically harmful tasks such as taking inventory. The system will be designed to assist rather than replace, and a worker will be involved in decision making. We aim to be transparent with the deployment of our design and inform operators of what the system can and cannot do.

7.3.2 Individual Virtues

Daniel:

I think creating a safe design is very important in an autonomous design and has been demonstrated well in the project. The hardware team put a lot of effort into finding methods to improve object detect and avoidance to develop a safe design.

I think it is important for a design to be accessible to a wide range of clients. Accessibility creates opportunities for smaller operators to have equivalent resources that will allow for expansion. I have not evaluated more affordable options for hardware components. A way I can change this going forward would be to review alternate options that still meet our needs while being more affordable.

John:

Intellectual Humility is important to me because it allows me to recognize when my understanding is incomplete, in fact I am often grateful getting "called out" in that way. Since it's an obvious signal that I haven't fully understood something. This happened several times during the semester, many times during faculty or team meetings where I would propose a design I had been working on and, in that moment felt like I knew everything about it, to just be left speechless when somebody else says, "so what about this?". It often can feel stressful in the moment but ultimately motivated me to learn more about the subject and improve.

Consistency is important because it ensures steady progress and reliability not just for your own work but probably more importantly for the team. It's untaintedly not fair putting stress on others because you simply weren't available that day or had other things going on. Parts of this semester I definitely felt inconsistent for my team, though I believe I improved as time went on by quicker replies, and setting aside dedicated time each week. Ultimately, I'd like to keep improving throughout the course of the project.

Jacob:

I believe that accessibility is an important virtue that is represented well in our project. This is important to me because I think innovation happens best when there are many minds involved.

Allowing people to have easy access to your project or code gives them the opportunity to improve it. We currently have open-source code and are attempting to create an affordable product.

Thoroughness was something that I struggled with at the beginning of this project. There were times when I kept an idea to myself or did not communicate with the team fully, which led to internal confusion. As time went on, I got a lot better and fully transparent about everything. I also got better at speaking up whenever I had an idea even if I thought it might make me stupid.

Ping:

One virtue I have shown in this senior design work so far is communication. This matters to me because working on a large group project requires everyone to stay updated, organized, and aware of each person's progress. Without strong communication, it is easy for confusion to happen, tasks to overlap, or parts of the project to fall behind. I have shown this virtue by staying connected with my team through weekly meetings, Discord messages, and emails. Using these communication methods has helped me keep track of the project's progress and stay involved as our team continues working on CyFlyBot.

One virtue that I value but have not fully shown yet is task management. Task management is important because it helps each team member make steady contributions and keeps the project moving forward. This semester, I sometimes had trouble finding clear tasks to complete because the project was still in the beginning stages and many parts were still being figured out. Next semester, I want to improve by creating clearer personal goals and keeping a list of tasks I plan to finish. This will help me stay organized, contribute more confidence, and feel like I am supporting the team more effectively.

Tu:

One virtue I have demonstrated in my senior design work so far is responsibility. This is important to me because our project involves autonomous robots operating around people, so every design decision needs to be made carefully. If one part of the system fails or is not tested properly, it could affect the safety and reliability of the entire project. I have demonstrated responsibility by staying involved with the software side of the project, helping organize how the HQ, UAV, and UGV communicate, and making sure our design choices support the overall system goals.

One virtue that is important to me, but I have not fully demonstrated yet is confidence. Confidence is important because senior design requires sharing ideas, asking questions, and speaking up when something may not work. At times, I have been hesitant to voice my ideas because I was unsure if they were correct or useful. Going forward, I can demonstrate more confidence by contributing more during team discussions, explaining my reasoning clearly, and being more willing to take ownership of parts of the software design.

Mitch:

A virtue I have shown has been balance. There has been so many different things that needed to get done, that my focus has been making sure nothing falls through the cracks. Whether through administrative work like emails or report writing, I was able to balance it with my work as part of the hardware team and the team as a whole. It has been important to me because it has been needed for the success of the team. In a sense, someone needed to do it.

A virtue I would like to work on is excellence. While I believe my work has met an appropriate standard, I have not "knocked it out of the park" every time. I will be working over the summer to make sure my knowledge and effort level is up to par with that if each individual part could stand on its own at a high level. In a sense, I have covered breadth up until now, now I need to cover depth for each section equally. This part is important to me due to it improving our teams success. Things can only improve if I improve.

Caleb:

Initiative is important to me because a project like CyFlyBot doesn't come with a set plan. Being able to identify what the project needs and act on it is what keeps the project moving. I feel like I have demonstrated this in a few concrete ways throughout the semester. Early in the project, I researched and demoed ArUco marker detection as a localization approach and made the case for adopting it, and it has become a core part of our system so far. I also took ownership of setting up the Docker environment, the Gazebo simulation, and the ROS 2 packages so that my teammates had a working foundation to build on. Later, I implemented the ArUco marker detection within the simulation itself. None of these were assigned to me specifically; I just saw what was needed and did it.

A virtue that I want to work on in this upcoming semester is deliberateness or taking the time to carefully evaluate options before committing. I believe this is important in a project like this, because the cost of a wrong decision becomes worse over time. There have been several moments in our project where I moved fast rather than carefully. For example, when setting up the simulation, I didn't evaluate our options as much as I should have. While it turned out fine, that's not the same as making a well-reasoned decision. Going forward, I want to slow down and evaluate my options more in depth when making key decisions, especially as we move into the integration phase where stakes are higher.

8 Closing Material

8.1 Conclusion

The CyFlyBot project aims to solve the critical limitations in modern warehouse automation by integrating a UGV with a UAV to create a coordinated robotic system. The primary goal of this system is to combine the strengths of both platforms, with the UAV providing aerial mapping and object detection, while the UGV performs ground-level navigation and task execution. Together, they improve efficiency, accuracy, and safety in warehouse operations.

So far, the team has made strong foundational progress, particularly in system architecture, simulation development, and technology selection. The adoption of ROS 2 as the core framework enables scalable communication between all system components, while the Gazebo simulation environment allows early testing of navigation, sensing, and coordination without relying on physical hardware. Additionally, the implementation of ArUco marker detection demonstrates a viable approach for object identification and localization.

The best plan of action to move forward is to continue transitioning from simulation to real-world integration. This includes establishing stable communication between the UAV and UGV, and HQ system, refining SLAM-based mapping and navigation, and improving computer vision accuracy. Testing will ensure that each subsystem works reliably before full system deployment.

Several constraints have impacted progress, including limited access to hardware, the complexity of multi-agent communication, and the short UAV flight time. Additionally, integration challenges between hardware and software components remain a key risk. In future iterations, improvements could include optimizing onboard processing to reduce communication latency, enhancing battery efficiency for longer UAV operation, and refining the user interface for better usability.

Overall, the CyFlyBot system demonstrates strong potential as a scalable and efficient warehouse automation solution. With continued development, testing, and refinement, the system is expected to meet its design requirements and provide meaningful improvement to real-world logistic operations.

8.2 References

https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

[1] M. Basiri, F. Schill, P. Lima, and D. Floreano, “Onboard relative positioning for autonomous UAV-on-UGV landing,” in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), 2014, pp. 1765–1771. https://doi.org/10.1109/IROS.2014.6942793

[2] Bitcraze, “Crazyflie 2.1 Documentation,” {Online}, Available: https://www.bitcraze.io/

8.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but help support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.

Appendix A. System Architecture Diagrams

Appendix B. Hardware Schematics

Figure 20 CrazyFlie Brushless 2.1 schematic

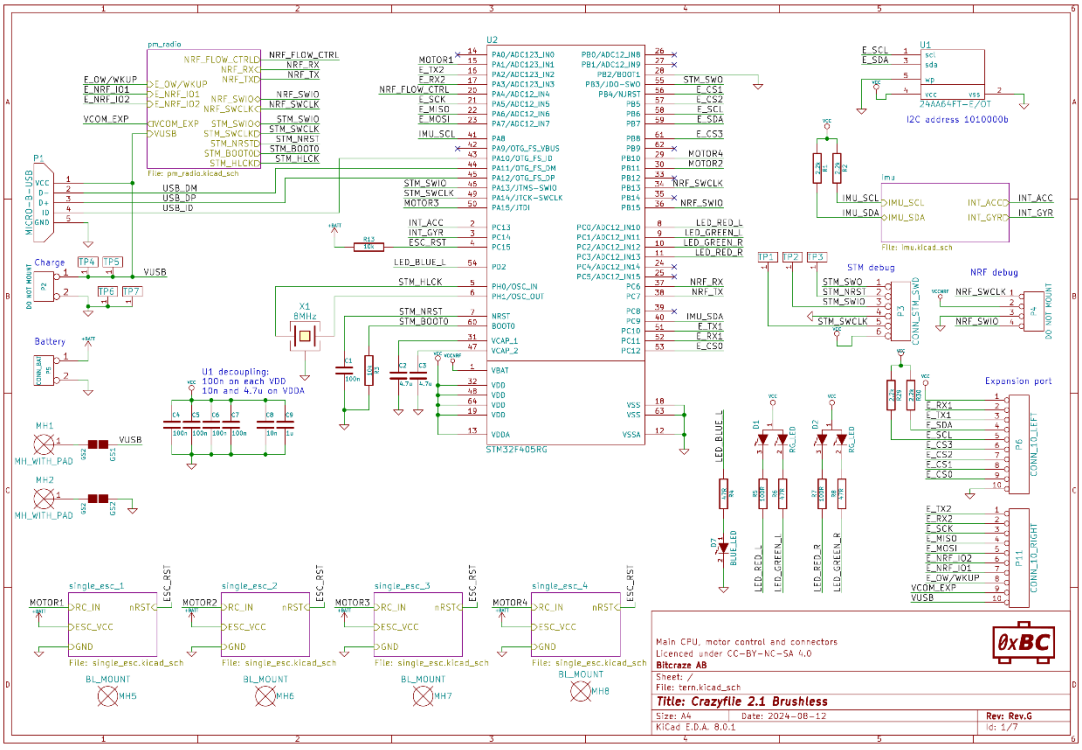


Figure 21 Multi-Ranger Deck schematic

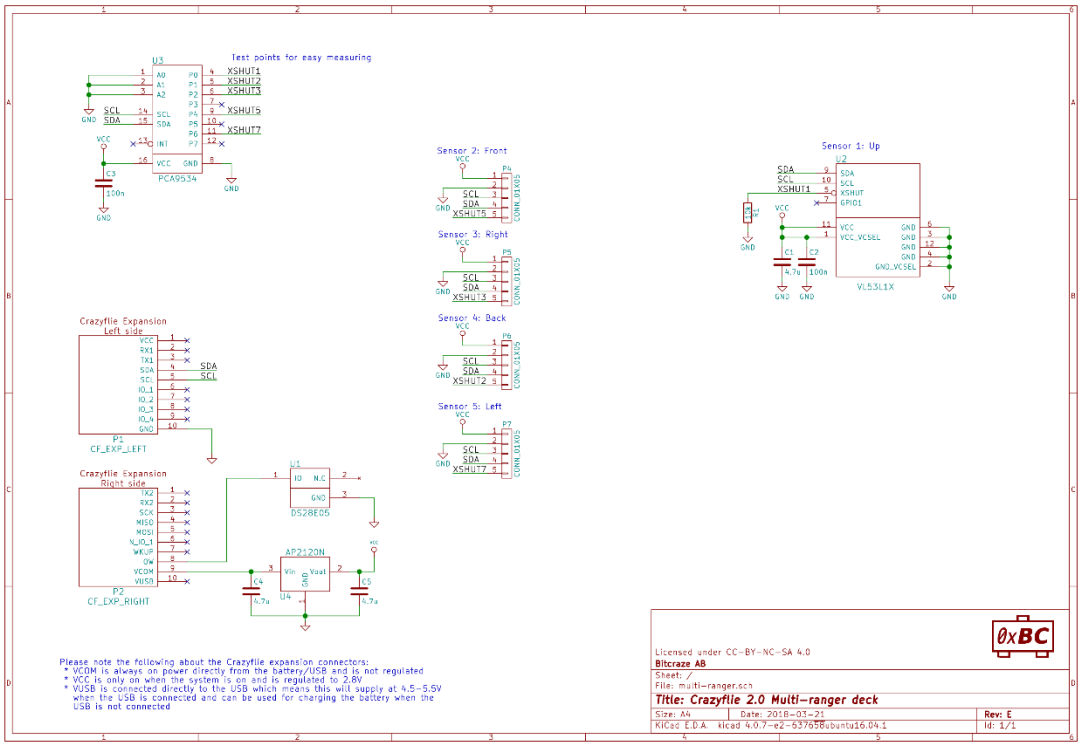


Figure 22 Flow v2 deck schematic

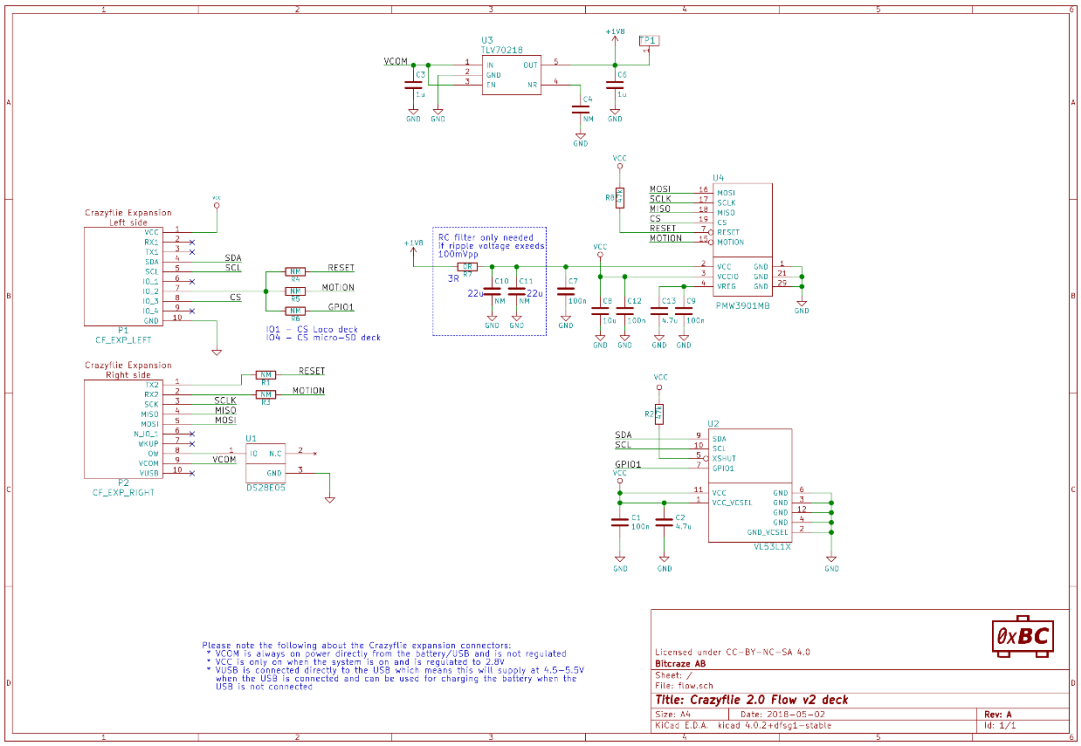
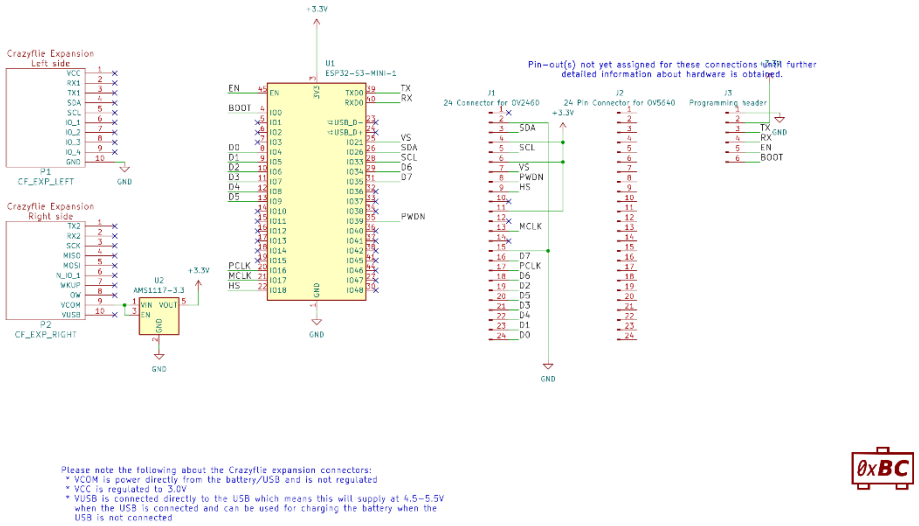


Figure 23 Imaging deck schematic



Appendix C. Software & ROS Graphs

Appendix D. Simulation & Testing Visuals

Appendix E. Additional Data / Tables

9 Team

Complete each section as completely and concisely as possible. We strongly recommend using tables or bulleted lists when applicable.

9.1 Team Members

9.2 Required Skill Sets for Your Project

Embedded systems programming (TM4C123 / CyBot firmware);

ROS 2 node development (Python/C++);

Autonomous navigation and SLAM; Computer vision and ArUco detection (OpenCV);

Drone operation and Crazyflie firmware; PCB/hardware design (expansion deck, imaging solution); Docker and Linux systems administration; UI/web development (HQ operator interface); Wireless communication (Wi-Fi, CRTP Crazyradio).

9.3 Skill Sets covered by the Team

Embedded systems / hardware: John Brittain, Mitch Fowler, Daniel Bieber (Hardware subgroup).

ROS 2, SLAM, navigation, computer vision: Tu Reh, Ping Wu, Caleb Sanchez, Jacob Nickel (Software subgroup).

Drone operation and Crazyflie integration: John Brittain, Mitch Fowler, Daniel Bieber.

UI/HQ operator interface: Tu Reh, Ping Wu, Caleb Sanchez, Jacob Nickel. All team members contribute to testing and integration.

9.4 Project Management Style Adopted by the team

The team follows an Agile project management style. Work is organized into biweekly sprints with clearly defined goals per subgroup. Weekly all-team meetings are held on Mondays at 4:00 PM in SIC 3321. Weekly faculty advisor meetings are held on Fridays at 3:15 PM in Durham 353 with Dr. Diane T. Rover and Dr. Phillip H. Jones. Progress is tracked on GitLab with milestone-based issue tracking and mandatory branch-based pull request reviews.

9.5 Initial Project Management Roles

Project Manager / Report Lead: Mitch Fowler. Hardware Subgroup Lead: John Brittain. Software Subgroup Lead: Caleb Sanchez. Hardware Members: John Brittain, Mitch Fowler, Daniel Bieber. Software Members: Tu Reh, Ping Wu, Caleb Sanchez, Jacob Nickel. Client / Advisor Liaison: Mitch Fowler (primary contact with Dr. Rover and Dr. Jones).

9.6 Team Contract

Team Members:

1) Tu Reh_____ 2) Ping Wu_____

3) Caleb Sanchez_____ 4) Daniel Bieber_____

5) John Brittain_____ 6) Jake Nickel_____

7) Mitch Fowler_____ 8) _____

Team Procedures

Day, time, and location (face-to-face or virtual) for regular team meetings:

Mondays at 4pm in SIC 3131 face-to-face

Fridays at 3:15pm in Durham 353 face-to-face w/faculty advisors

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-

mail, phone, app, face-to-face): Communication on Discord. Urgent contact through text

3. Decision-making policy (e.g., consensus, majority vote): General consensus

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be

shared/archived): Minutes shared and archived on Discord

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

Team members are expected to have regular attendance to meetings and give prior notice to the team's Discord if they cannot attend.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

Team members are expected to match the effort required for all work, communication, to meet deadlines comfortabley, IE. No last-minute submissions.

3. Expected level of communication with other team members:

Team members are expected to send requests immediately, teammates get in contact within 24 hours. Daily checking of the group server.

4. Expected level of commitment to team decisions and tasks:

Team members are all expected to have equal commitment required across the team, but if somebody wants to go above-and-beyond they can with the expectation that not everybody can do that.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction,

individual component design, testing, etc.):

Each team member will oversee their own components.

2. Strategies for supporting and guiding the work of all team members:

Regular group meetings supplement self-directed work.

3. Strategies for recognizing the contributions of all team members:

High fives, pizza during meetings

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

Daniel Bieber	Embedded systems experience, backend development, and testing experience
John Brittain	Embedded systems focus, versatile and practical ideas guy
Jacob Nickel	Good amount of experience with ground bots. Good amount of experience with different software.
Ping Wu	Software user interface
Tu Reh	Software web application experience
Mitch Fowler	Electrical engineer, jack of all trades
Caleb Sanchez	Lots of computer vision experience (mainly open-cv)

2. Strategies for encouraging and support contributions and ideas from all team members:

Open dialogue during all meetings

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

Communicate immediately, solve issues with zero ego

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

Regular achievement of goals set by team during meetings, aiming for being a week ahead of schedule time

2. Strategies for planning and assigning individual and team work:

Each member be responsible for their section, members finishing early assist others needing help

3. Strategies for keeping on task:

Regular check in with members during weekly meetings

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

Regular infractions of obligations in the contract will be communicated to the offending team member by the whole group.

2. What will your team do if the infractions continue?

If infraction continues, the matter will be brought up to the appropriate authorities.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the

consequences as stated in this contract.

1) ___Daniel Bieber_____ DATE_____2/5/26_____

2) ___John Brittain_____ DATE _____2/6/26_____

3) _____Jacob Nickel_____ DATE_____2/7/26_____

4) _____Ping Wu_____ DATE _____2/10/26_____

5) _____Tu Reh_____ DATE _____2/10/26____

6) _____Caleb Sanchez_____DATE _____2/10/26_____

7) ___Mitch Fowler_____DATE _____2/5/26_____