RoboSAR

Multi-Robot Search and Rescue

Conceptual Design Report

Team F

Charvi Gupta | Indraneel Patil | Jaekyung Song
Narendar Sriram | Rachel Zheng

Sponsors: Prof. Katia Sycara | Jaskaran Grover

Date: December 13th, 2021

Master of Science
Robotic Systems Development

Table of Contents:

1. Project description

Traditional search and rescue is cost, time, and labor intensive, and is often fraught with dangers. Rescuers frequently expose themselves to hazardous situations with little information to go on, and time is rarely on their side. To reduce cost and risk while improving search time, we propose integrating robots into the process.

Rather than sending in a team of humans to search a building from the beginning, our project will send in a team of robots, called agents. These agents will then search the rooms and hallways of the building, reporting people and obstacles to the end user. Once every room has been searched, the system will report that search has been completed, and the agents will return to the end user.

By sending in robots first, we can locate obstacles and injured personnel without putting the rescuers at risk. Armed with this information, the human rescue team can then rapidly plan and rescue people while minimizing risk to themselves.

The primary focus of the project is the multi-agent search algorithm: how to most efficiently search a known environment with multiple agents, how to best allocate tasks to each agent, and how to implement the system to minimize search time.

2. Use Case

There is a dangerous gas leak in a school. The alarms go off, and the students are evacuated. However, after doing a headcount, the teachers realize there are some students missing. Firefighters are sent to rescue the remaining students in the school. However, it is very dangerous for the firefighters to go into the toxic fumes for an extended period of time, and there is a persistent risk of an explosion. Instead of risking more lives, a swarm of robots is deployed to search the school for trapped victims (figure 1).

Based on a prior floor plan in the form of an occupancy grid like one shown in the image below, the robots search the entire school for students. The swarm of robots all start in one location outside of the main hall doors. The central server, controlled by the firefighters, assigns each robot an area in the school to search. When the robots are navigating the school hallways towards their assigned area, they move in swarm formation so that they do not collide with one another. If a robot detects a movable obstacle, it pushes it out of the way so that future robots will be able to pass easily.

The robots branch off from the swarm to search each individual room. When they detect a victim, they send the victim's location to a firefighter for rescue. A firefighter can then head to the victim and evacuate them to safety. Once a robot is done searching a room, it receives the next room to search, and begins to navigate there. If a robot is lost due to connection error,

navigation hazards, or other causes, the central server reallocates the search tasks to the remaining robots. The robots finish searching after visiting every room in the school.

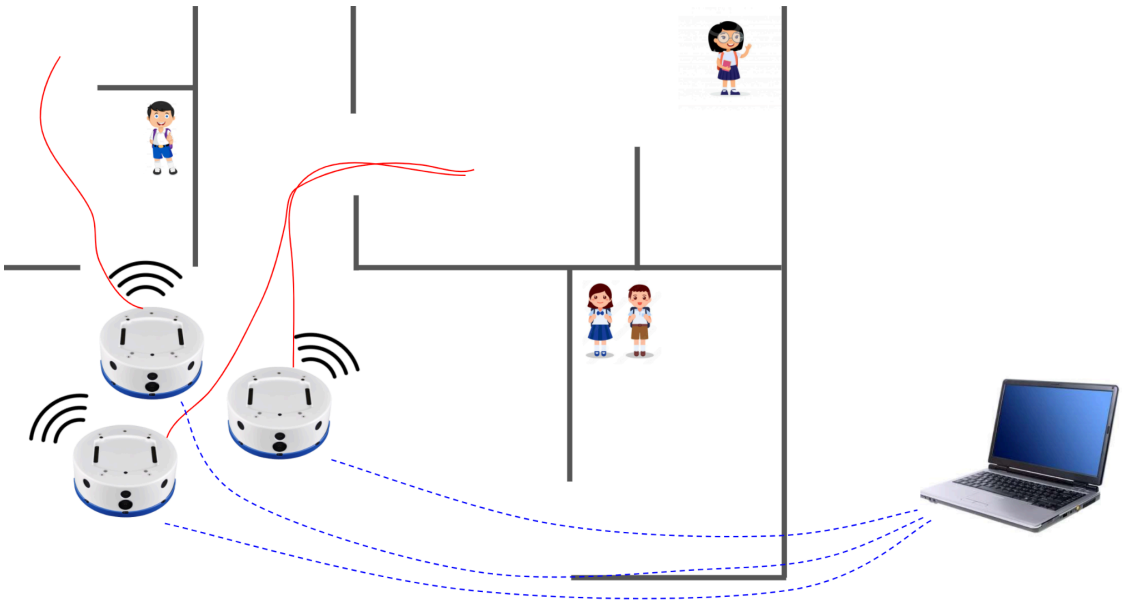

Fig 1. Use Case Illustration

3. System-level requirements

System requirements were derived from the use case, discussions with sponsors, and what was realistically achievable in the given time and budget. All requirements are shown in tables 1 through 6.

3.1. Mandatory Requirements

Table 1. Mandatory Requirements

	Requirement
M.F.0.	System shall identify and localize victims in environment using fiducial markers within desired rescue time
M.F.1.	System shall localize robots with a known map of the environment
M.F.2.	System shall dynamically allocate tasks to agents in the fleet for optimizing time required for the search
M.F.3.	System shall ensure simultaneous navigation of agents which have been allocated a task
M.F.4.	System shall adapt to changes in the fleet size like addition of agents by the users or loss of robots during operation.

3.1.1. Performance Requirements

Table 2. Performance Requirements

	Requirement
M.P.0.	System will detect 95% of victims in the environment.
M.P.1.	System will localize victim position within 20cm.
M.P.2.	System will finish searching the environment 100% faster than a single robot system.
M.P.3.	System will have downtime of less than 5s for busy agents.
M.P.4.	System will update agent status for user at 1Hz frequency.
M.P.5	System will handle at least 4 agents in the fleet.

3.1.2. Non-Functional Requirements

Table 3. Non-Functional Requirements

	Requirement
M.N.0.	System shall visualize the environment, locations of victims, and robot agents during operation
M.N.1.	System shall home agents on emergency stops, and after completion of search
M.N.2.	System shall have user interface to add/subtract robots during operation
M.N.3.	System shall be well documented and have reusable software framework for swarm robotics

3.2. Desirable Requirements

Table 4. Desirable Requirements

	Requirement
D.F.0.	System shall map and search in unknown environment simultaneously during operation
D.F.1.	System shall detect and move obstacles during search (non-prehensile manipulation)
D.F.2.	System shall support flocking/formation control of multiple robots executing similar trajectories
D.F.3.	System shall account for victims needing simultaneous detection by multiple robots
D.F.4.	System shall account for victims needing detection by specific types of robots

3.2.1. Performance Requirements

Table 5. Performance Requirements

	Requirement
D.P.0.	System will detect movable obstacles with 95% accuracy.
D.P.1.	System will successfully move obstacles of less than 15g, and 12.5x6.6x6.6cm.

3.2.1. Non-Functional Requirements

Table 6. Non-Functional Requirements

	Requirement
D.N.0.	System shall visualize the locations of obstacles during operation
D.N.1.	System shall visualize victims needing multiple or special robots
D.N.2.	System shall support heterogenous robot search and control

4. Functional architecture

The system is divided into two parts: the server and the agents.

4.1. Server Functional Architecture

Shown in figure 2, the server sends and receives information with the agents, and also provides an interface to the user. To start **mapping** the environment, the user provides an initial map of the building, showing the hallways, rooms, and walls. As the agents explore the environment and discover victims, obstacles, and landmarks, they will report this information along with their own positions to the server. The server will then update its own map with this information to generate the most up-to-date map. This map is then visualized for the user through a GUI and provided to the agents for their localization algorithms.

The map is also used for **task allocation**. Using the map, the server will identify the best areas to explore, as well as obstacles that need to be moved aside. At the same time, the server will track the number and status of agents with an **agent manager**, and create a list of available agents. Task allocation will then use this list and the identified hotspots and obstacles to assign tasks to the agents.

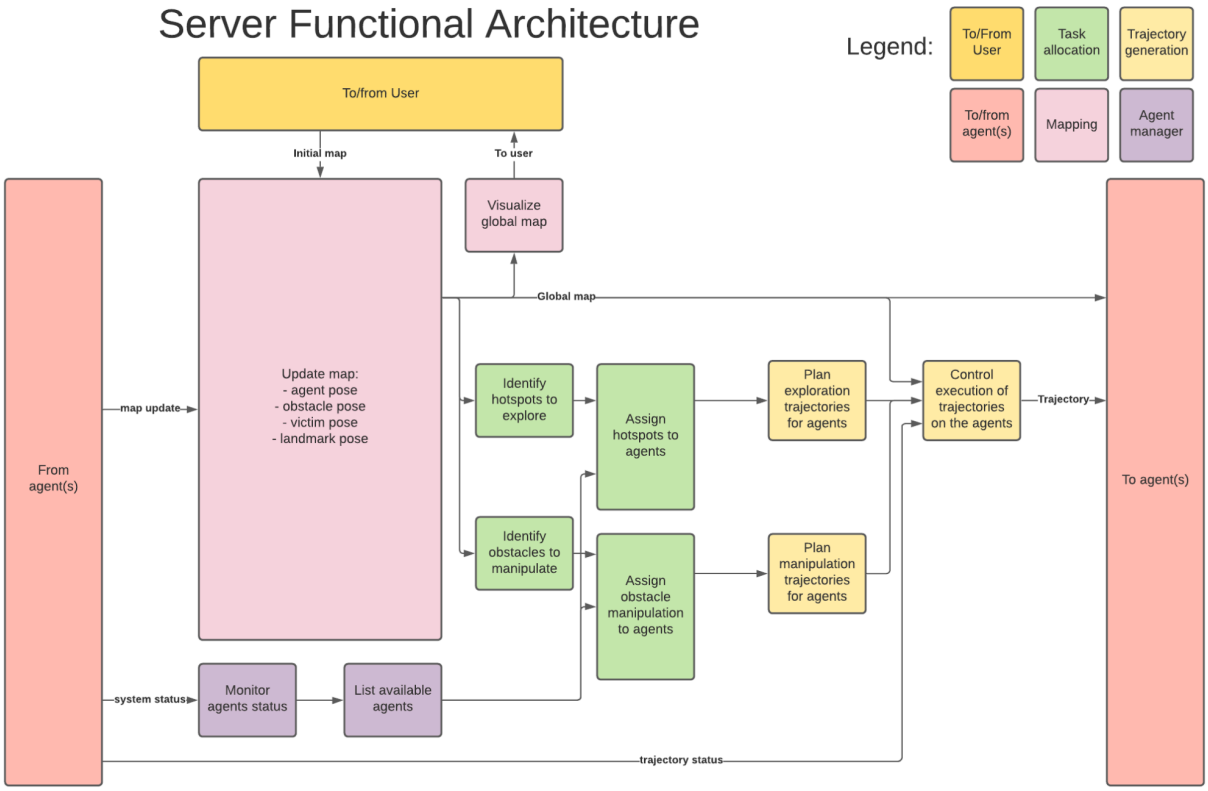

Fig 2. Server Functional Architecture

Last is **trajectory generation**: trajectories are computed so agents can carry out their assigned tasks. Additionally, the server must control the execution of these trajectories to ensure the agents are moving as needed, which is done by comparing the desired trajectory with the agent poses from the global map, as well as monitoring the agents' trajectory status flag. If an issue occurs, the trajectory is modified, and then transmitted to the agents.

4.2. Agent Functional Architecture

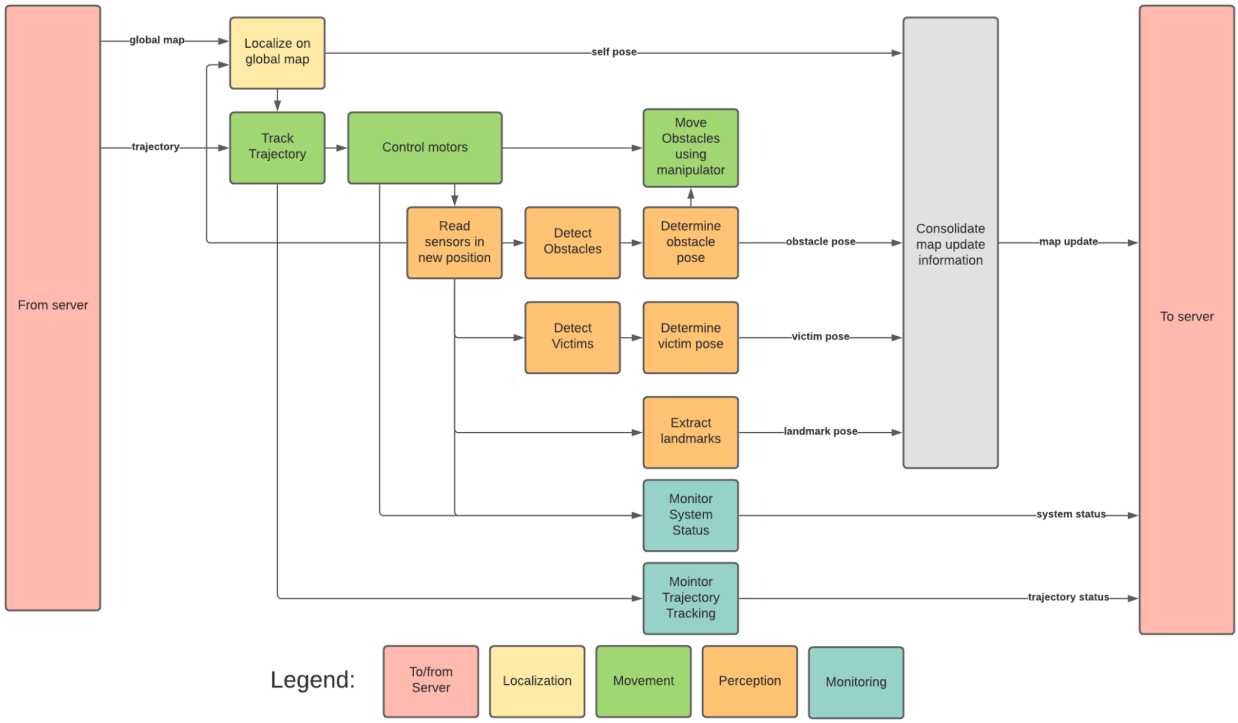

Fig 3. Agent Functional Architecture

The agent explores the environment, manipulates obstacles, and reports to the server, as shown in figure 3. First, the agent performs **localization** to determine self pose by using the latest map from the server and the outputs of the sensor readings. This is used to compute **movement** by comparing its current position with the server-calculated trajectory. The agent's motors are actuated to minimize this difference, moving the robot, thus achieving exploration or obstacle manipulation.

As the agent moves, it reads its sensors to perform **perception**. The data from the sensors are analyzed to locate obstacles, victims, and landmarks. This information, along with self pose, is transmitted to the server to update its global map.

The agent is always **monitoring** its own status, both in terms of health and trajectory tracking. The agent performs self testing as well as checks the outputs of its sensors to detect failures. If any occur, the issue is reported to the server so the server can decide what to do with the malfunctioning agent. Likewise, if an issue occurs with tracking the trajectory for any reason, the server is informed so it may adjust the agent's trajectory accordingly.

5. System and subsystem-level trade studies

5.1. Centralized vs Decentralized Multi-Robot control

This trade study (table 7) highlights the main design choice that we had to make in the pre functional architecture phase. The choice of whether the multi-robot team will have a centralized control (Multi agent system) or a decentralized control (Swarm intelligence). Both these systems have their own benefits but we have used our system functional requirements as a lighthouse to decide what's truly important to our search and rescue scenario in the evaluation criteria. We have also added a column which indicates the mapping of the criteria to our functional requirements. Amongst all the criteria, optimal task allocation, robot localisation complexity and navigation complexity are the most important to us because the efficiency of search and rescue and the feasibility of our system depends on it. Followed by other criteria which should be considered but not as important, such as formation control, scalability of fleet and search area size, robustness to failures.

Table 7. Multi-Robot Control Trade Study

Criteria	Source Functional requirement	Weight Factor	Value (1 to 5)	
			Decentralized	Centralized
Optimal Task Allocation	M.F.2	30	1	5
Flocking/Formation control	D.F.2	5	5	3
Robot localisation complexity	M.F.1	25	1	5
Size of robot fleet/search area	M.F.4	5	5	1
Kin Recognition/Navigation complexity	M.F.3	25	1	5
Robustness to Robot Failures	M.F.4	5	5	1
Operator Graphical User Interface	N.F.0	5	3	5
Weighted Average		100	1.7	4.5

5.2. Initial environment mapping algorithm

In the functional architecture the server assumes that a map of the environment is available from the user. In this trade study (table 8) we have explored prospective SLAM algorithms which can be used to build this initial map of the environment [1], [2]. There are no requirements which map directly to this initial map building activity, but it is an essential part of the localisation of our overall system. General evaluation criterias are chosen here to ensure ease of usage and quality of the produced map.

Table 8. Mapping Trade Study

		RTAB-Map	Gmapping	ORB-SLAM
Criteria	Weight Factor	Value (1 to 5)		
Robustness to various environments	20	4.5	4	4
Map accuracy	25	4.5	4	5
Localization accuracy	20	5	4	4.5
Handling of scale ambiguity	5	5	5	1
Ease of use and integration	10	5	5	4
Sensor attainability	15	3	3	5
CPU usage	5	5	5	5
Weighted Average	100	**4.55**	**4.125**	**4.45**

5.3. Agent localisation in the environment

Our M.F.2 says that the agents must localize in the environment with a known map of the environment. This trade study (table 9) tries to figure out the best technique to localize the team of agents in our indoor environment scenario. Out of all the methods described here only the indoor positioning system using beacons needs additional sensors to be installed on the khepera robot. All the methods are the same in terms of accuracy so the evaluation criteria is largely based on non functional parameters. The criteria which can directly jeopardize the mission have the highest weight are CPU load and multi robot interference. Other criteria which directly impact the effectiveness of the system are next, namely "requires changes to the environment", and "introduces new limitations on size of environment".

Table 9. Localisation Trade Study

		ArUco Tags	IPS using Vicons	IPS using beacons	Centralized SLAM	Decentralized SLAM
Criteria	Weight factor	Value (1 to 5)				
Needs changes to environment	15	1	3	3	5	5
Ease of	10	3	5	5	1	3

implementation						
Introduces limitation of size of environment	15	3	1	2	5	5
Computational cost on khepera CPU	25	3	5	5	5	1
Potential multi robot interference	25	3	5	5	5	2
Requires map	10	5	5	5	3	3
Weighted average	**100**	**2.9**	**4.1**	**4.25**	**4.4**	**2.85**

5.4. Choice of multi robot simulator

There are a lot of robotics simulators but all of them are not scalable to multi robot applications since some of them are super detailed and hence become computationally inefficient while simulating multiple robots. On the other hand there are multi robot simulators which are so primitive that they dont model simple obstacle interaction. Through this trade study (table 10) we are trying to find a good trade off between simplicity for the scalability required for our multi robot scenario but also good built in collision checking using a physics engine.

Table 10. Simulator Trade Study

		Gazebo	Morse	Webots	Stage	V-Rep	ARGoS
Criteria	Weight Factor	Value(1 to 5)					
CPU Load	25	1	5	3	5	3	5
Multirobot collision checking	25	5	5	5	3	5	5
Ease of developing new environments	10	3	5	2	5	3	3
Modelling obstacle interaction	10	5	5	3	1	5	1
Documentation	30	5	3	3	5	3	3
Weighted Average	**100**	**3.8**	**4.4**	**3.4**	**4.1**	**3.7**	**3.8**

6. Cyberphysical architecture

Following the functional architecture, the cyberphysical architecture is also divided into two parts. The server runs on a stationary central computer and the agents are out in the field. There are multiple agents and they have the same cyberphysical architecture as depicted in this section.

6.1. Server Cyberphysical Architecture

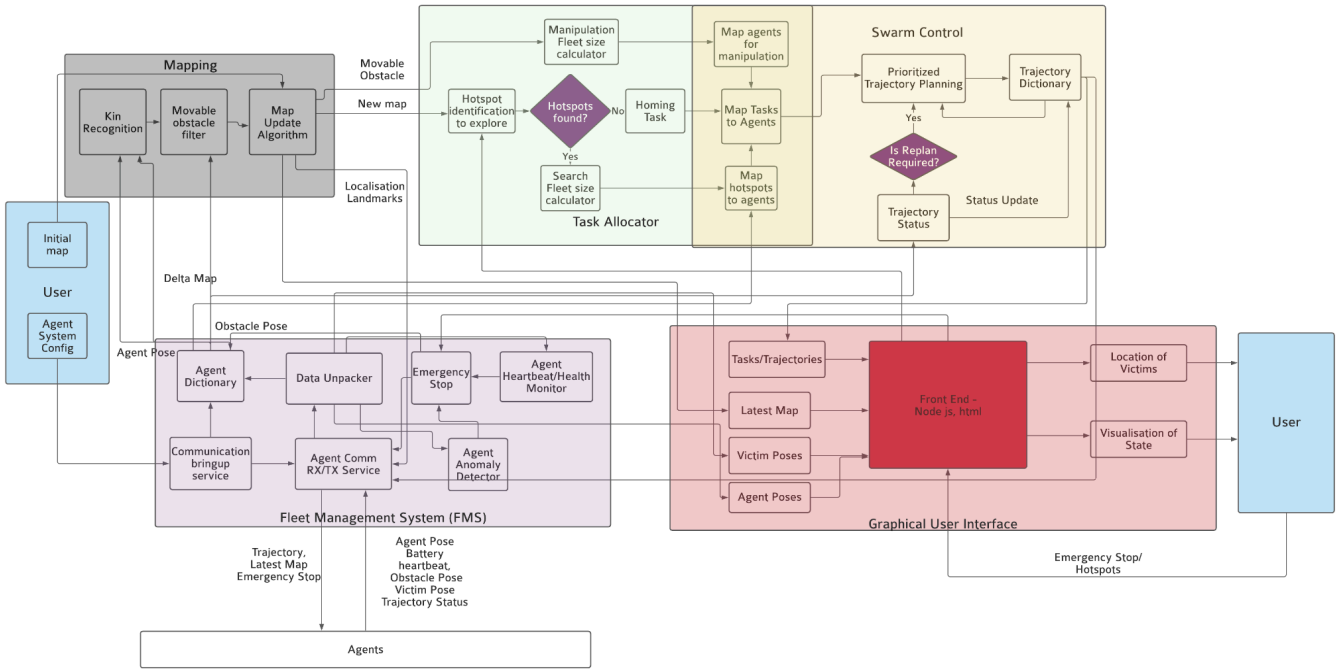

Fig 4. Server Cyberphysical Architecture

The server acts like the brain of the system, interacts with the end user and orchestrates behavior of the agents, as shown in figure 4. It takes the initial map of the environment and a system config file which contains the information of all agents in the system as an input from the user. Then the **Fleet Management system** (FMS) uses the system config file to set up the communication interface with each agent in the fleet. It then passes the communication handler to the RX/TX service and also updates the agent dictionary based on the successful connections. Emergency stop inside the FMS can be triggered from the Anomaly Detector, Health monitor as well as the user. Up to date information about all agents is maintained inside the agent dictionary.

The **Mapping** subsystem is incharge of collecting new sensor data from the FMS and building the map, returning localisation cues to the agent and passing this map to the Task allocator.

Task Allocator subsystem identifies new tasks to undertake, calculates optimal number of agents needed for these tasks and then also maps agents to these tasks. Task allocator performs its function in very close conjunction with the Swarm control subsystem.

Swarm Control subsystem as the name suggests ensures that the swarm of agents complete their tasks simultaneously and efficiently. After planning trajectories for each agent ,it also monitors the execution of these trajectories and replans paths if needed.

All these operations are visualized by the **Graphical User Interface** (GUI) for the user in real time so that the operator is always informed about the search and rescue mission. The GUI also allows the user to trigger emergency stop and insert search hotspot requests into the system.

6.2. Agent Cyberphysical Architecture

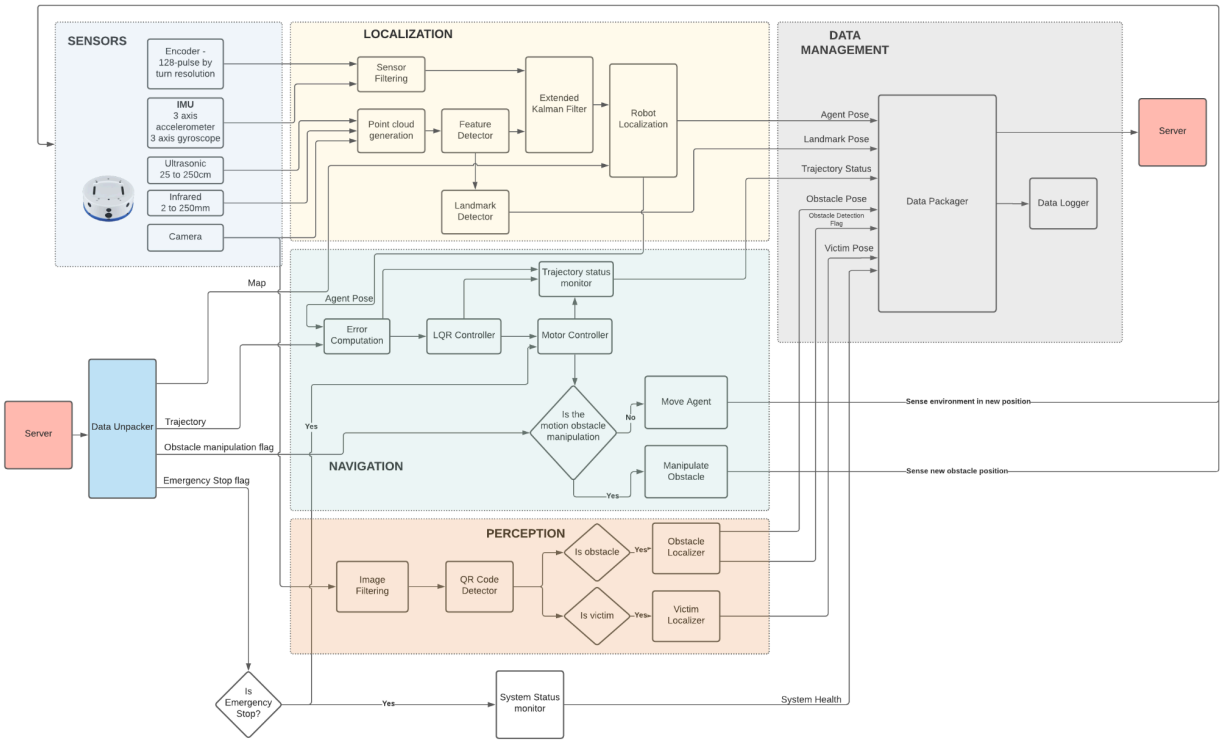

Fig 5. Agent Cyberphysical Architecture

We are using Khepera IV robots for the agents. This platform was primarily designed for testing artificial intelligence algorithms, especially multi-agent systems, so they are well suited for our purpose [3]. The Kheperas come with several **sensors** (as shown in figure 5), which can be used for proprioception (odometry, 3-axis gyroscope and 3-axis accelerometer) as well as measuring the environment (infrared and ultrasonic as range finders, RGB camera).

The sensors are used in the **localization** subsystem to determine agent pose. Proprioception is used for dead-reckoning navigation, which is augmented by environment

measurements to form an Extended Kalman Filter (EKF). The output of the EKF is combined with the server map to determine self pose. This subsystem will also detect and localize landmarks, allowing itself as well as other agents to use the found landmark for loop closure.

The **navigation** subsystem uses self pose to track trajectories. The difference between the desired and actual pose will generate an error, which is fed into an LQR controller , which controls the differential steering motor speeds to achieve the desired trajectory. By following the trajectory, the Khepera will explore the environment and manipulate obstacles using a non-prehensile manipulator. A trajectory status monitor will monitor the error, LQR controller, and the motor currents to detect failures, and reports to the server as appropriate.

The Khepera's camera is used to identify and locate both obstacles and victims in the **perception** subsystem. Because obstacles and victims will use QR tags, the Khepera will look for QR tags in its camera stream and read it. Once found, the contents of the tag will identify the object as a victim or obstacle. The camera feed is then analyzed to determine the victim or obstacle's pose, which is then reported to the server.

The **Data Management** subsystem packages all the information going to the server before logging and transmitting it. Packaging makes the data transfer more efficient, as well as easier to work with on the server side.

7. Subsystem descriptions

7.1. Localization

The localization subsystem localizes each of the agents within the environment and is run on the agents themselves. Its inputs include odometry data and sensor data from the RGB camera, ultrasound, and infrared sensors. It uses feature or landmark matching to update the Extended Kalman Filter and predict its current pose. The subsystem outputs the agent pose and also landmark poses. We will test out several localization methods as shown in the previous SLAM trade study.

7.2. Navigation

The navigation subsystem takes in a trajectory from the server, and executes control inputs to track the given trajectory. This subsystem ensures that the agent is able to navigate towards its assigned task. It constantly computes a pose error, and uses a LQR controller to compute the optimal control inputs. It outputs the trajectory status, whether it is being successfully followed or not. The navigation subsystem also handles obstacle prehensile manipulation. It takes in an obstacle manipulation, which tells the subsystem when and where a movable obstacle is detected. The subsystem then generates a new trajectory to manipulate the obstacle, and when finished, resumes the original trajectory tracking.

7.3. Perception

The perception subsystem takes in sensor data from the camera, ultrasonic and infrared sensor to detect movable obstacles and victims. It filters the images and constantly runs an obstacle and victim QR code detector. It outputs the obstacle or victim location.

7.4. Data Management

The data management subsystem takes all the information within the agents, packages them, and sends them to the server. This information includes: agent pose, landmark pose, obstacle pose, obstacle detection flag, and victim poses. The relay of information happens through a data handler which makes decisions on whether to persist a data or simply relay it. In the former case, it is pushed on to a database and in the latter, it is transmitted to the concerned module via API calls. Apart from these functions, Data Management subsystem also logs errors, unusual behaviors and major task completion updates on the DB: this can be exported into a text / csv file when required.

7.5. Fleet Management System

The fleet management system (FMS) communicates with the agents and monitors their status. It is responsible for establishing socket connections between the server and agents. It is located on the central server. FMS takes in agent configuration and current members of the agent class (such as agent pose, battery, heartbeat etc) as input and is given the intelligence to make various decisions on behalf of the agents such as when to avoid obstacles, when to remove/add an agent etc based on these inputs. If an agent is lost or added, this information will be relayed to the rest of the system through the FMS.

7.6. Map Update Module

The map update module updates the global map with new information from the agents. This could include previously unmapped debris or moved obstacles. It takes in map changes and agent poses, and uses this information to update the map.

7.7. Task Allocator

The task allocator assigns unexplored rooms in the map to each agent while minimizing search time. It takes in the map as an input, and uses an optimized search algorithm to find a mapping between rooms and agents. It then identifies the next search location, or hotspot, for each agent and sends this assignment to the swarm control subsystem. It also generates a collision-free global path to the hotspot.

7.8. Swarm Control

The swarm control subsystem takes in the assigned hotspots, and generates collision-free trajectories for the agents to collectively navigate to their goals. The subsystem will utilize both spatial and temporal planning strategies such as Velocity Obstacle Planning and Safe Interval

Path Planning to ensure collision avoidance [4], [5]. It outputs the generated local trajectories to each agent.

7.9. Graphical User Interface

The graphical user interface (GUI) visualizes what is going on for the user. It visualizes the global map, agent poses, victim poses, and obstacle positions.

8. Project Management

8.1. Work Plan & Tasks

The tasks to be completed were derived from the work breakdown structure in figure 6:

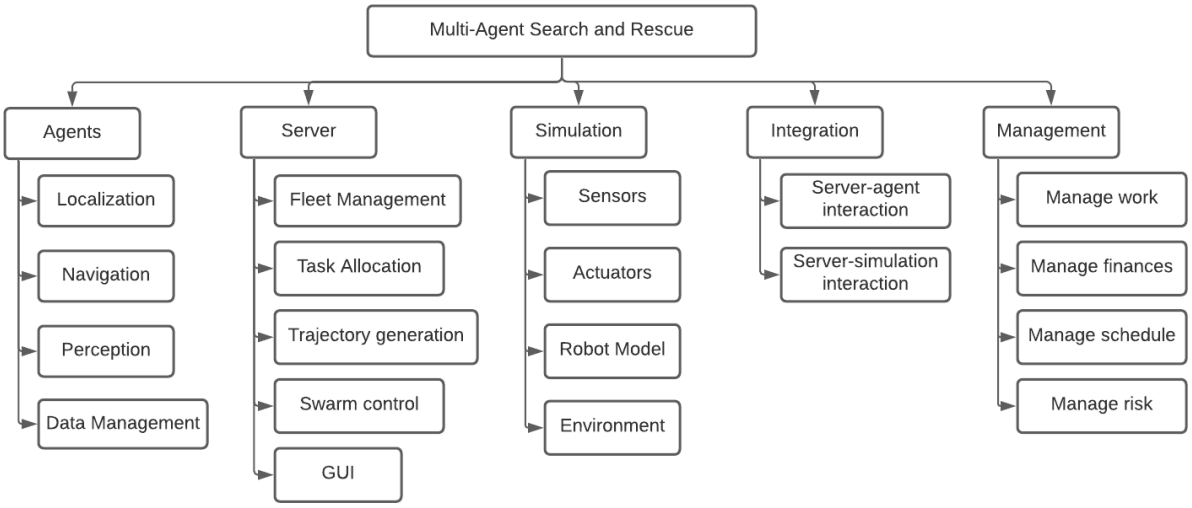

Fig 6. Work Breakdown Structure

The project is broken into 5 parts: agents, server, simulation, integration and management. Each part was elaborated upon by analyzing the cyberphysical architectures.

Our milestones are to design, integrate, and test each of the tasks in the WBS, as shown in the following schedules.

8.2. Schedule

Detailed schedules are available in references [6]; below we present a high level overview of our schedule for the Spring and Fall semesters in figures 7 through 11. Note simulation is expected to be completed in the Spring semester. Project management software of choice is Jira, and scheduling was done using Gantt charts.

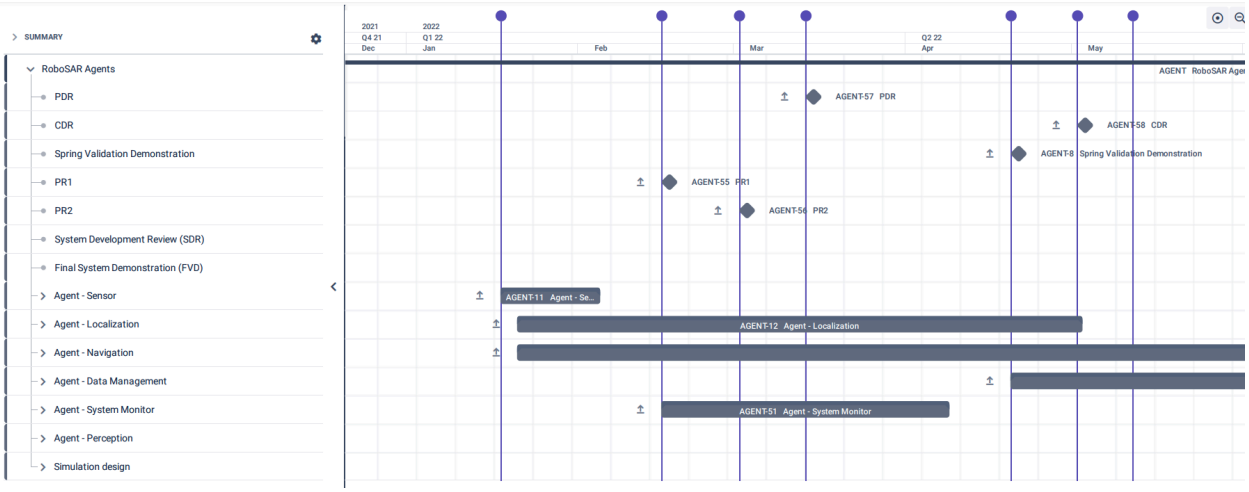

Fig 7. Agents Spring 2022 Schedule

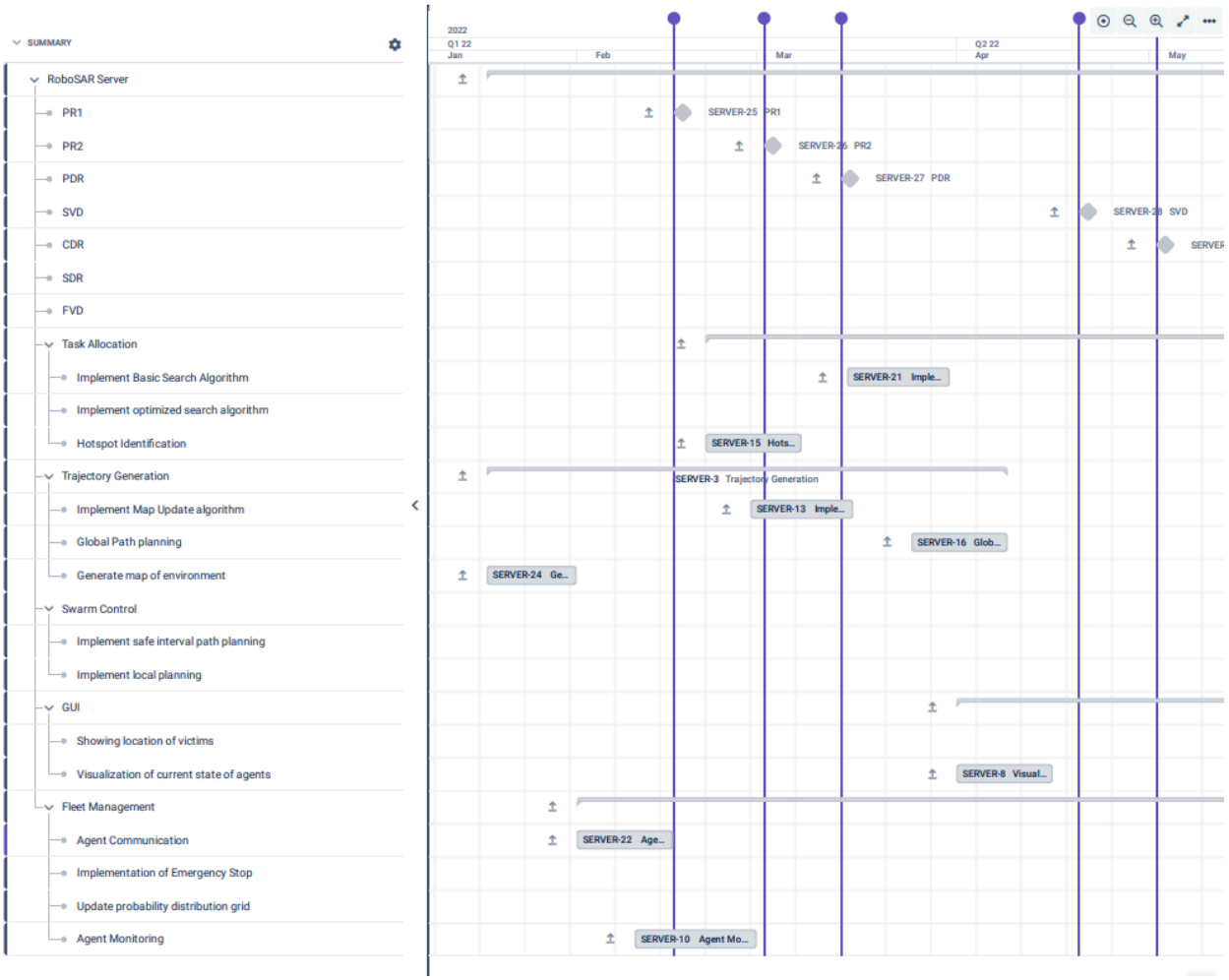

Fig 8. Server Spring 2022 Schedule

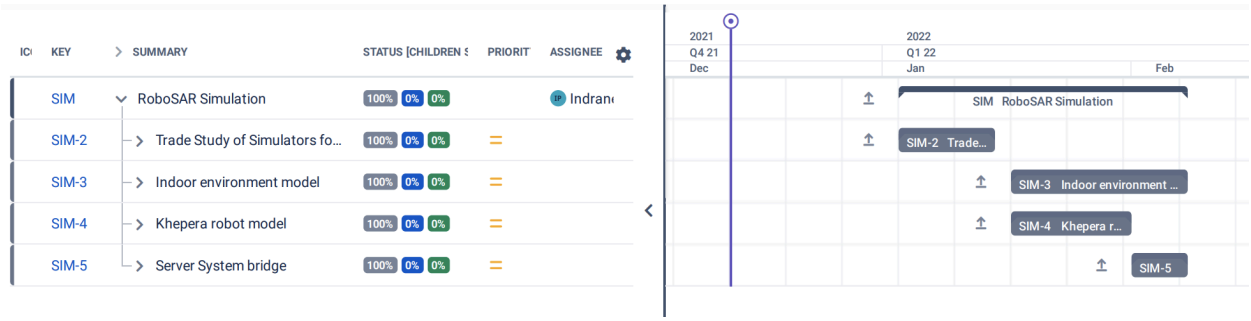

Fig 9. Simulation Spring 2022 Schedule

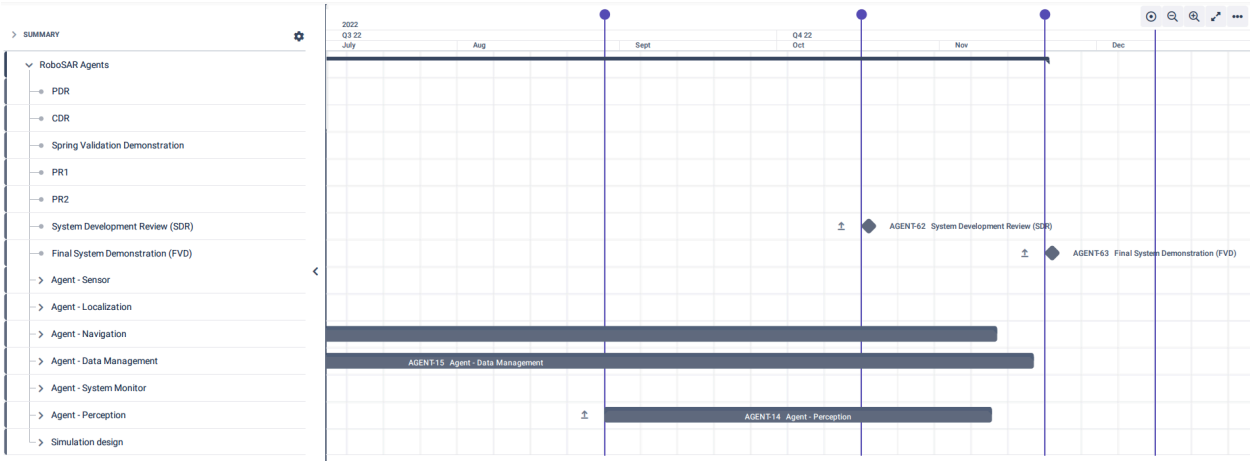

Fig 10. Agents Fall 2022 Schedule

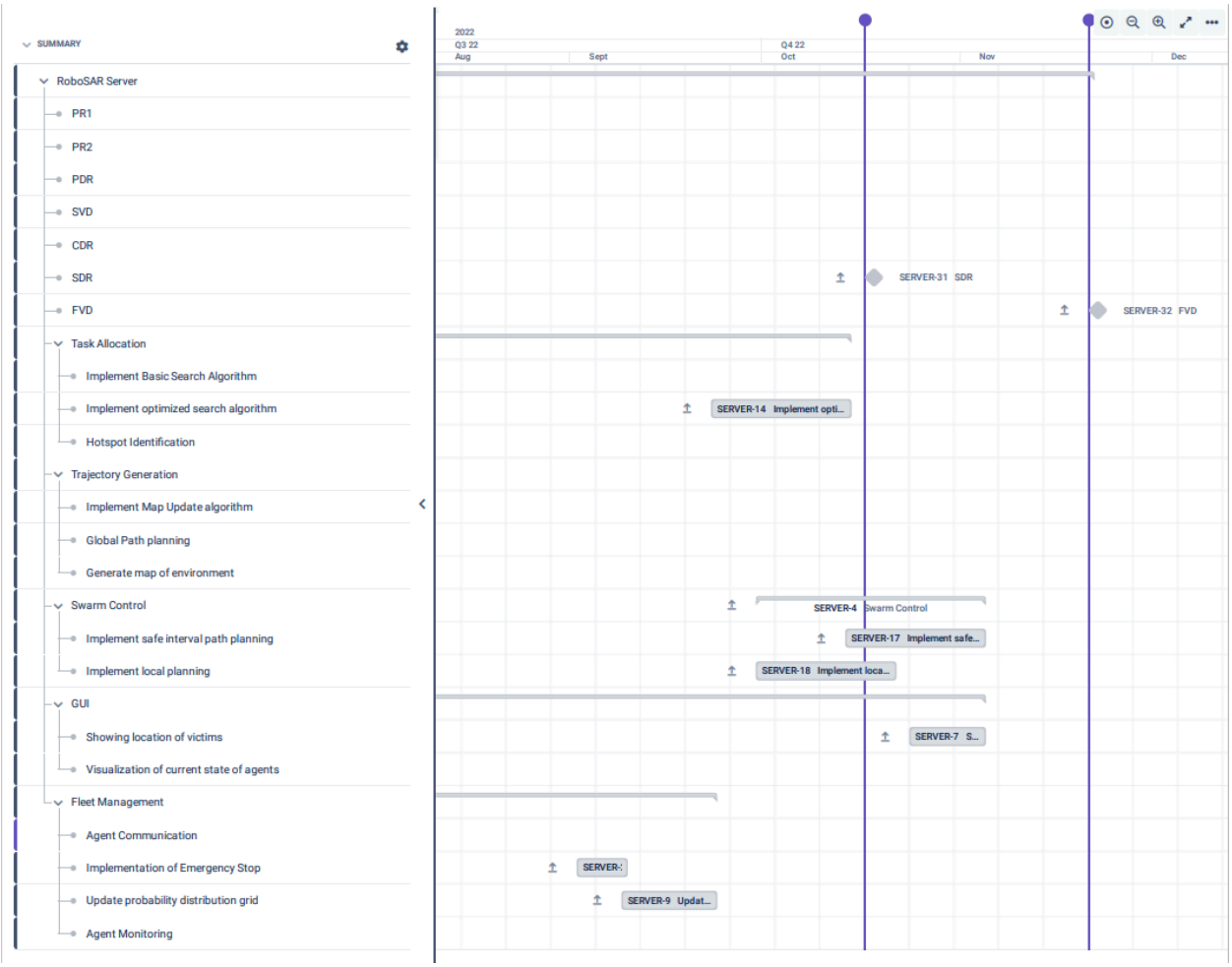

Fig 11. Server Fall 2022 Schedule

8.3. Milestones

First we will cover the milestones and dates, and elaborate on select milestones:

8.3.1. Spring 2022 Milestones

- Progress Review 1 (PR) - February 16
- Progress Review 2 - March 2
- Preliminary Design Review (PDR) - March 14 & 16
- Spring Validation Demonstration (SVD) - April 20 & Encore April 27
- Critical Design Review (CDR) - May 2 & 3

8.3.2. Fall 2022 Milestones

- System Development Review (SDR) - October 17 & 19
- Final System Demonstration (FVD) - November 21 & Encore November 30

8.3.3. Progress Review 1

- Acquire map of test environment
- Demonstrate agent's ability to execute control inputs.
- Demonstrate ability of server to generate basic trajectories.
- Demonstrate agent's ability to track trajectories.
- Demonstrate agent - server communication.

8.3.4. Progress Review 2

- Demonstrate agent's ability to localize within a map of the environment.
- Demonstrate simultaneous execution of control inputs for multiple agents.
- Demonstrate agent monitoring.

8.3.5. Spring Validation

We shall demonstrate:

- task allocation on physical or simulated agents.
- collision free trajectory generation for physical or simulated agents.
- integrated task allocation and trajectory generation.

Demonstration conditions:

- Location: Vicon test area in Advanced Agent-Robotics Technology Lab
- Operating area: 5 m^2
- Equipment: laptop, Khepera IV robots, WiFi, Vicon system

Procedure:

- Agents will be placed at an initial starting position within the operating area.
- The user will give the agents an initial position estimate, along with a map of the environment.
- The agents will be assigned a trajectory.
- The agents will localize in the environment, and track the trajectory.
- The Vicon system will provide the agent's ground truth pose.
- Once an agent completes a given trajectory, steps 3-5 will be repeated for this agent in such a way that other agents' trajectories are not affected.

Fulfilled system requirements:

a. Functional Requirements:

- M.F.1. System shall localize robots with a known map of the environment.
- M.F.2. (Partial fulfillment) System shall dynamically allocate tasks to agents in the fleet for optimizing time required for the search.
- M.F.3. System shall dynamically allocate tasks to agents in the fleet for optimizing time required for the search.

b. Performance Requirements:

- M.P.2. System will finish searching the environment 100% faster than a single robot system
- M.P.3. System will have downtime of less than 5s for busy agents.
- M.P.4. System will update agent status for users at 1Hz frequency.

8.3.6. Fall Validation

We shall demonstrate:

- Demonstrate autonomous search and exploration of the environment by the agents.
- Demonstrate simultaneous navigation of the entire fleet of robots integrated with task allocation and trajectory tracking.
- Demonstrate swarm control of robots.
- Demonstrate GUI for visualization.
- Demonstrate agents identifying fiducial markers as victims and relaying info to the server.
- Demonstrate homing of agents after search is complete.
- Demonstrate addition and removal of agents from UI.
- Demonstrate emergency stop of agents from UI.

Demonstration conditions:

- Location: NSH 4th floor
- Operating area: 50 m^2
- Equipment: laptop, Khepera robots, wifi, Fiducial markers

Procedure:

- Agents will be placed at an initial starting position within the operating area.
- The user will give the agents an initial position estimate, along with a map of the environment.
- Agents will be given a trajectory based on the search algorithm.
- The agents will localize in the environment, follow the trajectory and identify hotspots to explore.
- Agents will report every identified victim (fiducial marker).
- Tasks allocated to agents will get updated with every reported agent along with a new collision free trajectory.
- Steps 3-6 will be repeated until <5% of fiducial markers are left to report.

Fulfilled system requirements:

a. Functional Requirements:

- M.F.0.System shall identify and localize victims in environment using fiducial markers within desired rescue time
- M.F.2. System shall dynamically allocate tasks to agents in the fleet for optimizing time required for the search.
- M.F.4. System shall adapt to changes in the fleet size like addition of agents by the users or loss of robots during operation.

b. Performance Requirements:

- M.P.0. System will detect 95% of victims in the environment.
- M.P.1. System will localize victim position within 20cm
- M.P.4. System will handle up to 8 agents in the fleet.

c. Non Functional Requirements:

- M.N.0. System shall visualize the environment, locations of victims, and robot agents during operation
- M.N.1. System shall home agents on emergency stops, and after completion of search
- M.N.2. System shall have user interface to add/subtract robots during operation
- M.N.3. System shall be well documented and have reusable software framework for swarm robotics

8.4. Team Member Responsibilities





Charvi Gupta | Rachel Zheng | Narendar Sriram | Jaekyung Song | Indraneel Patil

Charvi Gupta

- Primary: Agent subsystem
- Secondary: Task allocation

Indraneel Patil

- Primary: Simulation subsystem
- Secondary: Motion planning

Jaekyung Song

- Primary: Agent subsystem
- Secondary: System integration

Narendar Sriram

- Primary: Server subsystem
- Secondary: Motion planning

Rachel Zheng

- Primary: Server subsystem
- Secondary: Agent localization

8.5. Budget

The cost for this project is low. The project is largely software based, so very few physical components are needed. The Khepera robots are provided by our sponsor, so they do not add to the cost. Additionally, some of the sensors we are interested in are available in the MRSD inventory, so they will not add to the cost either. However, because we have many agents, most items must be ordered in large quantities. Note that table 11 shows Rough Order of Magnitude (ROM) costs.

Table 11. ROM Budget

Name	Description	Source	ROM Cost ($)	Qty	Subtotal ($)
Khepera IV	Robot platform for agents	Sponsor	0	10	0
YDLIDAR X4	LIDAR	COTS	100	10	1000
D435i	Intel Stereo camera	Inventory	0	10	0
PCB, assembled	PCB to add more sensors to agents	Custom	50	10	500
				Total ($)	1500

8.6. Risk Management

Table 12 shows our risk management and risk mitigation plans.

Table 12. Risk Management

	Risk	Likeli-hood	Conse-quence	Mitigation Plan
1	Failed kin recognition on server due to mismatch of robot pose on server and actual location	2	4	● Turn off robot once the confidence covariance matrix goes below a certain threshold ● Use a robot model size in proportion to the confidence covariance matrix of robot location for kin recognition
2	Agents sensors interfering with each other	4	4	● Test how sensors interact as soon as possible ● Investigate sensor alternatives without interference ● Design algorithms to rely on sensor without interference

3	Unable to procure the required sensors for localization on Kheperas	2	4	● Procuring lower quality sensors which might be available ● Localize using camera imaging ● Shift the localization task to Jackals (alternate robots)
4	Swarm algorithm failure due to bad data from faulty agents	1	3	● Robust swarm control algorithm ● Rugged testing for edge case and failure ● Simulate faulty agents in centralized control and discard
5	CPU usage overloading on the robots	2	4	● Offload computation to server as much as possible ● 'CPU compute requirement' included as a performance criteria in all algorithm trade studies
6	Hardware development is limiting software development	3	3	● Create simulators for software testing ● Modularize system and software development
7	Khepera robots are damaged or broken during testing	2	3	● Acquire extra robots ● Use simulator to continue work

9. References

1. N. Ragot, R.Khemmar, *Benchmark of Visual SLAM Algorithms: ORB-SLAM2 vs RTAB-Map*, Eighth International Conference on Emerging Security Technologies (EST). https://ieeexplore.ieee.org/document/8806213
2. M. Filipenko, I. Afanasyev, *Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment,* International Conference on Intelligent Systems (IS), 2018. https://ieeexplore.ieee.org/document/8710464
3. KHEPERA IV user manual: http://ftp.k-team.com/KheperaIV/software/Gumstix%20COM%20Y/UserManual/Khepera%20IV%20User%20Manual%204.x.pdf
4. P. Fiorini, Z. Shiller, *Motion Planning in Dynamic Environments using Velocity Obstacles*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. https://ieeexplore.ieee.org/document/1545110
5. M. Phillips, M. Likhachev, *SIPP: Safe Interval Path Planning for Dynamic Environments*, IEEE International Conference on Robotics and Automation, 2011. https://ieeexplore.ieee.org/document/5980306
6. Full Spring and Fall schedules: https://drive.google.com/drive/folders/11hsdIut4dU1Eb-_ACeGkfMsXPGktRfjt?usp=sharing