Scalable and Fault-Tolerant Autonomous Hexapod Swarm for Search and Rescue Missions

Kobe Zhang, Akash Arun, and Casper Wong

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—** Search and Rescue teams dealing with natural disasters face many challenges. For example: a lack of communication infrastructure, understaffing, harsh conditions, and compromised structural integrity. With modern technological developments, certain groups have been exploring the integration of sophisticated autonomous systems with the Search and Rescue process. This project furthers this exploration by creating an autonomous swarm of hexapod robots that collaborate to complete search and rescue (SAR) tasks. The hexapod swarm utilizes LAN communication, YOLOv8 object detection, VSLAM and a distributed search algorithm to get around the challenges that human SAR teams face and coordinate their search.

***Index Terms*—Autonomous Robot, SLAM, YOLO, Isaac ROS, NVIDIA Jetson Orin Nano, Search and Rescue**

I. Introduction

Our end product is a fully autonomous search and rescue system with a scalable number of hexapod robots. Each robot is responsible for mapping its terrain as well as identifying potential survivors.

The motivation for our solution is to reduce the need for human intervention in search and rescue as much as possible. Understaffing is a huge problem in search and rescue missions as workers are often not paid sufficiently to compensate them for the risk they undertake. A small rescue force may not be sufficiently large enough to cover a large search surface with adequate efficiency. In search and rescue missions every minute counts and can lead to loss of life. An additional problem is that these missions often involve workers being in precarious situations that risk injury or even death. Casualties and injury from such missions cause the workers and their families to have psychological stress from the vocation itself. Our product is scalable so the number of robots could be scaled up or down in response to the mission requirements and also fault tolerant so it can work even if some robots fail. Hence human intervention might only be needed to interface with the robots.

An existing solution that we saw was Inuktun’s small robots with tank-like treads that were used after 9/11 at the Twin Towers site and after Hurricane Katrina. These robots were very useful but a key difference between our solution and these was that robots were remote-controlled and needed a human to operate them. In comparison, our solution improves upon it by having our robots completely autonomous. Another difference is the establishment of a local network which helps each robot communicate its information with the other and optimize their collaborative search effort as much as possible.

II. Use-Case Requirements

We define the following Use-Case-Requirements for this project:
1) The hexapod swarm shouldn’t need constant signal access to communicate with each other. This comes from the use case where our hexapods enter areas to perform search and rescue tasks without a strong network infrastructure.

2)The hexapod should have an active battery life of at least 1 hour. This is a requirement to ensure that the hexapods can conduct a thorough search of a house or enclosure.

3) The hexapods should have a high accuracy of detecting a possible survivor in the frame with a low false negative rate. This requirement is to ensure that our hexapods can correctly identify survivors in a search and rescue environment. Additionally, we want to have our hexapods lean towards more false positives for survivor detection than false negatives, since we don’t want to accidentally miss any real survivor.

4) The hexapod swarm should be scalable; it should be able to seamlessly incorporate additional hexapods to make the swarm more efficient and it should also be able to adapt to failures of single hexapods. Our solution would need to be flexible in swarm size for human SAR teams to be able to effectively deploy our swarm. In cases where the search location is larger, the swarm should be able to scale up in numbers to compensate for the increase in search area. Additionally, the swarm should be able to detect and account for failures in case hexapods die in the process of the SAR mission.

III. Architecture and/or Principle of Operation

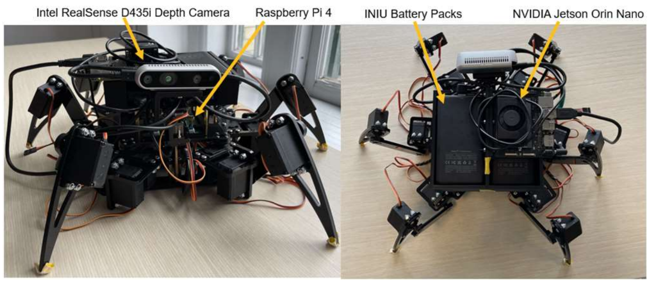


Fig. 1. Annotated image of 1 hexapod robot

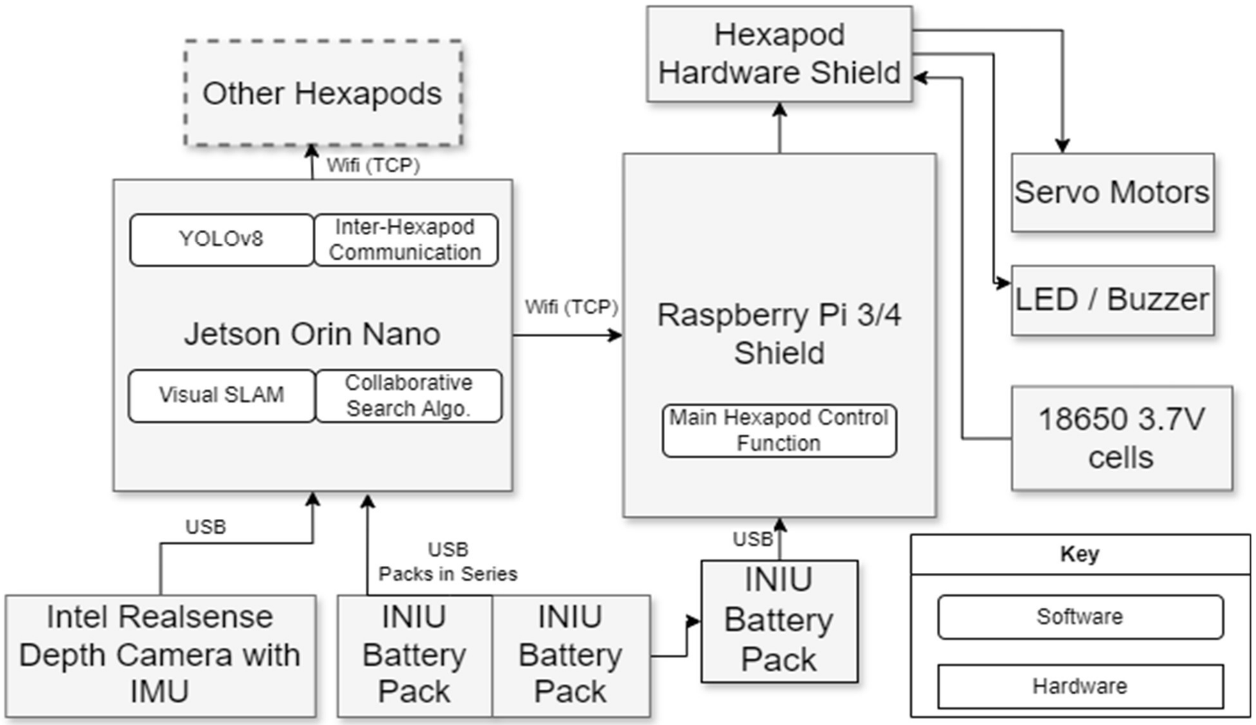


Fig. 2. Hardware block diagram

Each hexapod consists of a few essential subsystems that work together to make our search and rescue possible. First, for controlling the hexapod we use the off-the-shelf hexapod robot from FreeNove. This hexapod is controlled by our RPi and interfaces with a hardware shield

Figure 3. Software block diagram of our design

that moves

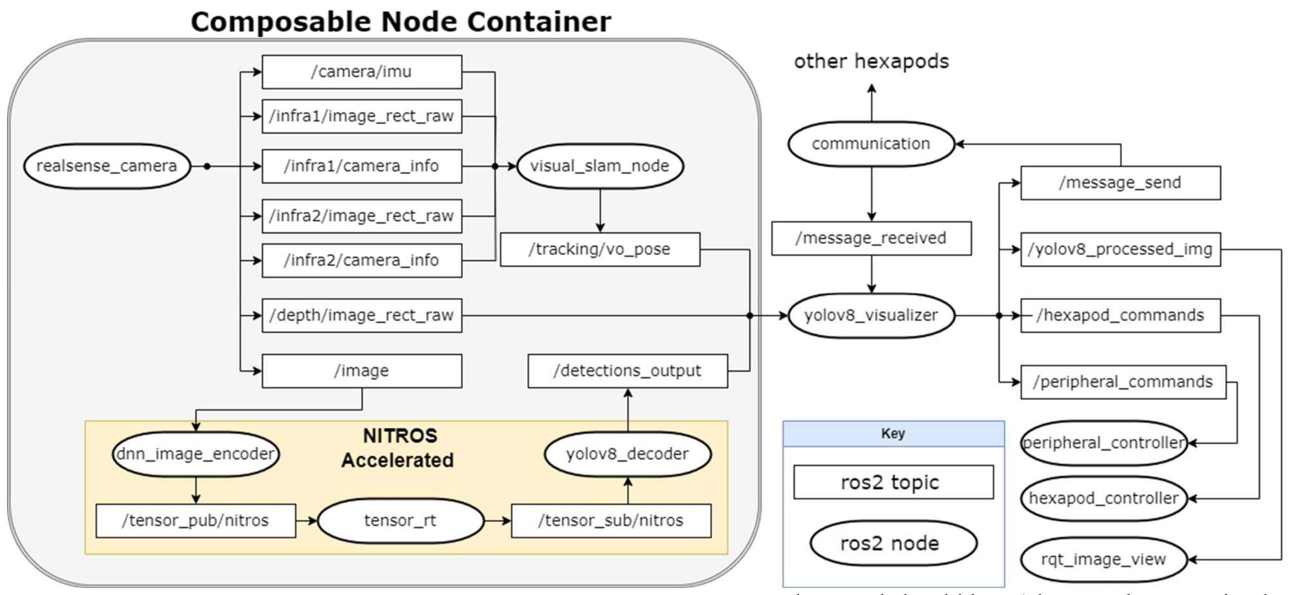
around the hexapod using its 6 servo-powered legs. We hijacked their existing controls framework and changed it so that we send commands from our central compute unit (Jetson Orin Nano) over a LAN using TCP to change our hexapod's motion. Using this pre-existing hardware gave our team more time to implement our desired swarm behavior, communication, object detection, and hexapod localization.

The swarm behavior, inter-hexapod communication, object detection, and localization were implemented on the Jetson Orin Nano, which we chose because of its relatively small size and weight in combination with its computational capacity and software support. The Orin Nano

will be interfaced with an Intel Realsense D435i stereo vision camera that also has depth sensing and an IMU (Inertial Mass Unit).

The Jetson also has an attached Wi-Fi module which allows it to host a local network for communication with other hexapods. This local network is one of the cornerstones of the inter-hexapod communication subsystem. The object detection subsystem on the Orin Nano runs the YOLOv8 object detection algorithm to search for survivors, getting stereo images from the Realsense camera. The Orin Nano utilizes the camera's depth sensing, infrared cameras, and IMU to do Visual SLAM which allows it to localize itself and map its surroundings to remember paths and survivor

locations. With this information, the hexapods coordinate with fellow hexapods and route their search path based on a distributed search algorithm to optimize search area coverage in the swarm behavior subsystem.



IV. Design Requirements

For the project's design requirements, the focus is on 4 main aspects of our hexapod swarm function that were described in the use-case requirements: 1) Inter-hexapod communication, 2) Hexapod survivor identification, 3) Swarm scalability, and 4) Hexapod battery life.

1. For inter-hexapod communication, hexapods should be able to send packets to each other at varying distances up to 20m with <5% packet loss. The average global house is 20mx20mx20m hence we came to a max distance figure of 20m. This should be achievable as we plan on using 2.4GHz LAN which has a range of approximately 50 feet indoors.
2. For Hexapod survivor identification, our team preferred to err on the side of caution and create a model that results in false positives rather than make one that neglects potential survivors. As a result, we want to be strict about having <5% rate of false negatives. This thought process led us to arrive at a figure of >80% mAP (mean average precision) for the detection accuracy of different kinds of human and non-human objects. 80% also represents a good balance between detection accuracy as well as the limitations of our hardware and real-time detection needs. When we ran experiments with YOLOv8 with images that had humans clearly in them it still had an accuracy of around 85-90%. This also influenced the selection of our figure since we can't predict if our accuracy will be worse or better when we train it with a custom dataset.
3. For swarm scalability, the swarm should have a 1.5x search completion time speedup with 3 hexapods compared to a single hexapod. This requirement is so that the additional cost of having more hexapods is justified with a corresponding improvement in search efficiency.
4. For hexapod battery life, the battery duration for a hexapod should be >1 hour under an active load (i.e. constant movement, running a Jetson…etc.). The average search and rescue mission lasts for 31 hours, in a real-life use case where our solution is used the hexapod wouldn't be useful if it wasn't able to search for at least an hour before getting substituted with another hexapod.

V. Design Trade Studies

To get to our current solution, our team examined a lot of possible approaches to create our hexapod swarm. We first explored the various options for the hexapod itself. FreeNove offered a variety of options for hexapod robots with varying costs and functionalities. The first hexapod we looked at was a smaller model that was controlled using an Arduino. While the low cost of this model was desirable, we were concerned about the ability of the smaller hexapod to carry larger loads including the weight of the Jetson Orin Nano and various needed peripherals. Additionally, we were unsure of the ability of the Arduino to handle communications from the Jetson Orin Nano while simultaneously running all of the necessary controls for the hexapod movement. Due to these concerns, we decided to look at the largest hexapod available that is also controlled through an RPi. This hexapod, which is our current one, offers the additional advantage of having a head module that could swivel 360^o, a desirable trait for our object detection tasks. While this hexapod came in at a much larger price than the original model, it also came with an ultrasonic sensor, camera module, and a higher weight capacity. Ultimately, we decided to move forward with this hexapod model for the benefits of computational power, head mobility, and weight carry capacity. The trade-off was an increase in cost per hexapod and an increase in power consumption.

Another critical design choice we made was for the main computational unit of our hexapod. We originally chose the NVIDIA Jetson Nano due to its low cost and GPU support. After a few rounds of initial trials with the software that we wanted to run, it was evident that the JetPack versions that the NVIDIA Jetson Nano was able to support were not enough for our project's needs, specifically for our object detection tasks. This was because the Jetson Nano could not support JetPack 4.7 and higher so it could not run Python 3.7 and above which was critical to our object detection subsystem. After using a virtual machine to get around software dependencies, we were able to run the object detection algorithm we wanted but the detection process was too slow (~30 seconds) for our purpose. Thus, we decided to switch to using a Jetson Orin Nano. The original Jetson Nano was released in 2019 whereas the Orin Nano came out in 2023. The difference in computation power in comparison to their size difference is representative of these last 4 years of hardware innovation. Once again, the tradeoff is an increase in price ($150 vs $500) and an increase in power consumption (max 10W vs max 15W, 5V input vs 7-20V input), for better support, more modern software, and higher JetPack version support, faster computation (approx. 80x), and a lot more benefits. We realized that the Orin Nano supports the usage of Issac-ROS which helps us better utilize the GPUs of the Nano to perform object detection and SLAM more efficiently. The Orin Nano is also approximately the same size as the Jetson Nano, making it feasible to use with our hexapod without needing a substantial change in the design of our harness.

Figure 4. Comparison of Jetson Nano and Jetson Orin Nano

For our object detection subsystem, we compared various versions of the YOLO object detection algorithm for speed, accuracy, and ease of use. Since we upgraded from the Jetson Nano to the Jetson Orin Nano, we decided to continue with YOLOv8 which is one of the newest versions of YOLO that offers one of the highest accuracy ratings. While we were considering using YOLO-Nas since it utilizes quantization-aware training and post-training quantization to reduce the size of the model and increase performance, we valued the increased accuracy of YOLOv8 more. This decision was also motivated by the upgrade of our central computing unit to the Orin Nano which could run YOLOv8 with fast speeds. Additionally, we found that Isaac-ROS supported YOLOv8 and allowed for the hardware acceleration of our object detection through their NITROS optimization.

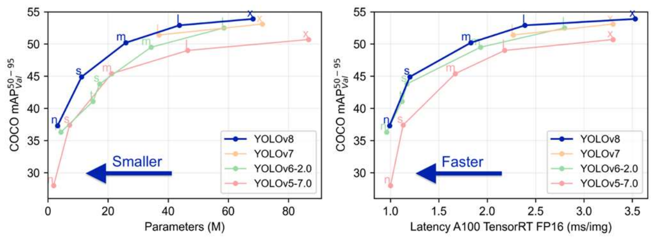


Figure 5 Comparison of various YOLO versions and varying sizes for each version

More specifically, our project uses YOLOv8s with FP32. We tested various combinations of reducing size and increasing the quantization level from FP32 to FP16 to INT8. After some evaluation, we chose Yolov8s FP32 due to the consistency of the predictions and the moderately low size of the model. This model would be pretrained on the COCO dataset, and we would further train the model with data that would be more specific for our use case. This would be in the form of images of human survivors in low-light settings.

For the inter-hexapod communication subsystem, we chose to go with a local area network (LAN) with Wi-Fi over nRF and UWB. This is because upon conducting some deep research on different forums that discuss the applications of these protocols in various robotics projects, we discovered a couple of key challenges that we would have to face if we used UWB or nRF over Wi-Fi. One challenge stem from our usage of an operating system (Jetson Linux 36.2) with a scheduler rather than having a microcontroller that runs bare metal code. UWB or nRF have very tight timing/latency requirements that need to be met for it to function properly. This wouldn't be an issue in the case of Wi-Fi as the Wi-Fi card handles this requirement, but UWB Modules expect the user of the module to deal with the requirement, which the Linux scheduler would be unable to meet. A workaround could be to connect our Jetson to an Arduino or another microcontroller and write a UWB driver that helps us meet our timing needs. They also have lower communication bandwidth in comparison to Wi-Fi which might cause us issues down the road. Wi-Fi modules also have a lot more built-in support with drivers, etc. in comparison. The big advantage of these protocols over Wi-Fi, however, is that they consume very little power. After considering these facts, we decided to choose Wi-Fi because of how easy it is to work with and create a LAN. A convenient plus with this decision is that Jetson Orin Nano's come with Wi-Fi cards built in which saves us additional expenditure. We chose to communicate information between the hexapods using TCP to reduce packet loss. TCP is something that we're more familiar with and has been proven to be very consistent.

A key design decision in our project is to build our solution by leveraging ROS (Robot Operating System) which is an open-source middleware framework. This decision gives a handful of advantages. For one, ROS is very modular so it's easy to plug and play different software packages. It has an extensive network of researchers worldwide who contribute to its software packages which will further speed up our development time. Isaac ROS is a recently released version of ROS2 that contains a lot of packages that allows for the development of computer vision robotic applications. This gives us access to pre-existing implementations of various object detection, SLAM, and control algorithms so we won't have to reinvent the wheel. ROS also supports simulations through tools like Gazebo and RViz so we can test our integrated system before we deploy it on the hexapod hardware.

Lastly, to allow our hexapods to get orientation data, we originally planned to get a magnetometer but after more consideration we realized that the amount of servos on our hexapod would cause the magnetometer to be very inconsistent and we would not be able to get solid readings. We got to this conclusion after consulting with friends that have used a magnetometer before. To get around this, we decided that we would continue with getting SLAM to work on our system. In the beginning we considered finding a way to purchase a LIDAR for SLAM but due to budget constraints we were not able to. Thus, we chose to go forward with VSLAM which is Visual Simultaneous Localization and Mapping. VSLAM would provide orientation data for us by tracking landmarks in the camera data. The tradeoff here is that VSLAM is less consistent than normal SLAM with some kind of lidar since VSLAM relies on the camera being consistent and can be affected by the camera's FPS.

VI. System Implementation

Isaac ROS

The majority of our software was run using the Jetson Orin Nano. We developed our software in an Isaac ROS environment that we set up via a docker container on our Jetson's solid state drive.

ROS (Robot Operating System) is an open source robotics middleware that allows for easier and more effective robotics development. Nvidia's Isaac ROS is a recently released collection of hardware accelerated, high performance, and low latency ROS2 packages that are made for autonomous robot perception tasks. Using Isaac ROS, we can leverage the power of GPU acceleration on NVIDIA platforms like the Jetson Orin Nano.

Object Detection and Classification

We used an object detection model to locate and track search and rescue survivors. This predominantly focuses on people, who may be partially obscured under rubble. Other objects of interest also include pieces of clothing and domestic animals. We trained our model such that it commits virtually no type II errors, so that the Hexapods do not accidentally ignore any survivors. To avoid these false negatives, we trained our model on a diverse dataset that includes people from different ethnicities, genders, and other demographic variables.

Our hexapods run YOLOv8 which is a start of the art deep learning model designed for real time object detection in computer vision applications. YOLOv8 is the latest version of the *You Only Looked Once* object detection algorithm that was developed by Joseph Redmon and Ali Farhadi. We looked into various options for object detection, testing different algorithms like YOLO-Nas, YOLOv7, and FOMO (*Faster Objects More Objects*). We originally wanted to use YOLOv7 for our project, since we were also running our entire computation on just a Jetson Nano. However, after testing we found that the performance of the Jetson Nano is inadequate and upgraded to the Jetson Orin Nano; this allowed us to also use YOLOv8 for the best accuracy and the best precision. Specifically, we used YOLOv8s since we decided it would be a good middle ground between size of model (which would increase our performance) and accuracy. Isaac ROS also has a YOLOv8 package, which means we gain the additional performance benefit of hardware acceleration. Our model was pretrained on the COCO dataset, and we further trained it for our use case of only detecting humans in disaster environments, such as in low light conditions.

With the object detections output, our hexapods are able to locate humans and follow them based on their location in the images. When hexapods find a human and get to a sufficiently close location, they will stop, sound a buzzer, and alert human search and rescue workers.

Visual Simultaneous Localization and Mapping

Hexapods need to map their surroundings to remember paths and survivor locations – this is important for Hexapods to effectively search through a space and convene with one another. To achieve this, we used the Isaac ROS Visual SLAM library developed by NVIDIA, which again utilizes

GPU acceleration to provide low-latency results in robotics applications. Visual Simultaneous Localization and Mapping refers to the process of determining the position and orientation of a sensor with respect to its surroundings while also mapping the environment around that sensor. We chose to use VSLAM because usual SLAM sensors, such as LIDAR, are unreasonably expensive for the scope of our project. Using an IMU and stereo camera, VSLAM combines visual-inertial odometry, which estimates the position of a robot relative to its start position from successive camera frames, with SLAM, which creates a map of key points to determine if an area is previously seen. Using the Isaac ROS VSLAM library, the Hexapods can quickly map out obstacles such as walls and insurmountable rubble, as well as retain the path they took to get to their current locations.

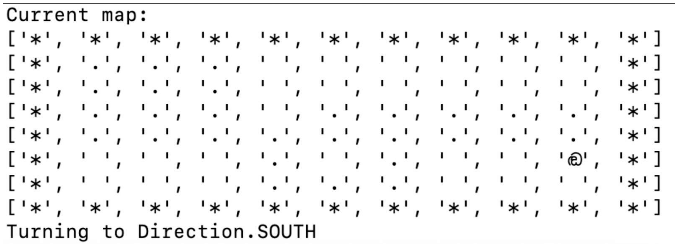
Our VSLAM node takes in data from our camera feed and outputs point clouds and pose data that we collect via a ROSbag. This ROSbag allows human search and rescue workers to visualize the path that our hexapod robot took and its surroundings in the process. A demonstration of the VSLAM running on the Jetson Orin Nano can be found here.

Search Algorithm
We created our own custom search algorithm, which was loosely adapted from a previously
implemented cooperative search algorithm for distributed autonomous robots (Cheng, 2004). This simple method almost fully eliminates communication between robots to reduce overhead when scaling up. Each robot follows 5 behavioral rules, prioritized with 1 being the highest.

1. Avoid obstacles and fellow robots ultrasonic sensors.
2. Find targets and alert neighboring robots
3. Response to neighboring robots' messages
4. Follow external commands
5. Wander in the environment.

As we can see in the original algorithm, robots do not need to know either their position or environmental layout. Although this prioritizes simplicity, it also leads to a lot of search redundancies, where multiple robots might search the same area. To make the search more efficient, we have Hexapods retain their position using SLAM, keeping track of previously visited locations and communicating this across bots. In doing so, the Hexapods can follow a new rule of avoiding previously searched areas.

The map is kept track of in the form of an unbounded 2-dimensional grid. The grid initially starts as a 1x1 array. Whenever the Hexapod moves to a new square, which we defined to be 2 feet apart, then the grid is updated to reflect the new Hexapod position and mark the previous position as visited. If the Hexapod walks beyond the border of the grid, then we expand the grid to have another column or row.


```
Current map:
['*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*']
['*', '.', '.', '.', ' ', ' ', ' ', ' ', ' ', ' ', '*']
['*', '.', '.', '.', ' ', ' ', ' ', ' ', ' ', ' ', '*']
['*', '.', '.', '.', ' ', '.', '.', '.', '.', '.', '*']
['*', '.', '.', '.', '.', '.', '.', '.', '.', '.', '*']
['*', ' ', ' ', ' ', '.', ' ', '.', ' ', ' ', 'ඞ', '*']
['*', ' ', ' ', ' ', '.', '.', '.', ' ', ' ', ' ', '*']
['*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*']
Turning to Direction.SOUTH
```
Figure X. Grid created from Hexapod while being enclosed in a rectangular box. 'ඞ' signifies Hexapod. '.' signifies visited square. ' ' signifies an empty square. '*' signifies blocked square.

General Algorithm – State Diagram

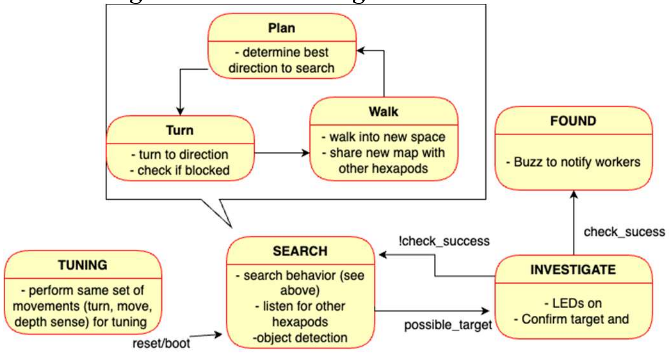


Figure X. state diagram for hexapod behavior.

Upon start-up, the Jetson Nano Orin will run the algorithm shown in Figure X, beginning in the search state.

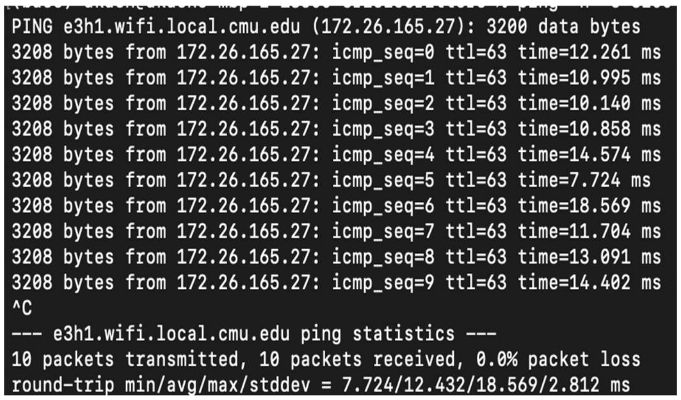
In the search state, Hexapods will wander around and actively run object detection to search for survivors as well as VSLAM to map out the environment and their path. Upon finding an object of interest with YOLOv8, the robot will enter the investigate state, which involves walking within a foot of the object of interest. Finally, the Hexapod will buzz to notify nearby rescue workers to retrieve the object of interest, before resetting back into search state again.

VII. Test, Verification and Validation

Building upon the design decisions outlined in the previous section, a crucial aspect of ensuring the effectiveness of our hexapod swarm solution lies in testing, verification, and validation. This stage involves a series of controlled and real-world evaluations to assess the functionality, performance, and suitability of the developed system. As we transitioned from the design phase you can see a contrast between our initially planned tests and the actual tests we ended up doing because of pivots in our implementation
We have 4 main categories of design specifications: Communication, Identification, Scalability, and Battery Life.

A. Tests for Communication

We tested communication by sending packets of around 3200 bytes from different distances ranging from 0.5-20m. The original purpose of this test was to profile our robot communication's packet loss and the 20m mimics the average global households dimension to mimic the real-world use case. We compared the percentage packet loss at each of these distances and wanted to ensure that our mean packet loss was less than 5%. Because we chose to use a LAN and TCP to send our messages our packet loss after conducting our tests was essentially 0 as seen in the below figure. Since we were previously experimenting with other communication protocols this test could have garnered different results if we used UWB, nRF, etc. One of the main reasons we chose LAN was because it helped us get our desired packet loss percentage without having to meet the strict timing requirements that using UWB entails. The test results fulfill our first use case requirement which requires that the hexapods shouldn't be dependent on a consistent internet connection to communicate with the outside world. The packet loss result is a testament to the reliability of our communication system.



```
PING e3h1.wifi.local.cmu.edu (172.26.165.27): 3200 data bytes
3208 bytes from 172.26.165.27: icmp_seq=0 ttl=63 time=12.261 ms
3208 bytes from 172.26.165.27: icmp_seq=1 ttl=63 time=10.995 ms
3208 bytes from 172.26.165.27: icmp_seq=2 ttl=63 time=10.140 ms
3208 bytes from 172.26.165.27: icmp_seq=3 ttl=63 time=10.858 ms
3208 bytes from 172.26.165.27: icmp_seq=4 ttl=63 time=14.574 ms
3208 bytes from 172.26.165.27: icmp_seq=5 ttl=63 time=7.724 ms
3208 bytes from 172.26.165.27: icmp_seq=6 ttl=63 time=18.569 ms
3208 bytes from 172.26.165.27: icmp_seq=7 ttl=63 time=11.704 ms
3208 bytes from 172.26.165.27: icmp_seq=8 ttl=63 time=13.091 ms
3208 bytes from 172.26.165.27: icmp_seq=9 ttl=63 time=14.402 ms
^C
--- e3h1.wifi.local.cmu.edu ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 7.724/12.432/18.569/2.812 ms
```

B. Tests for Identification

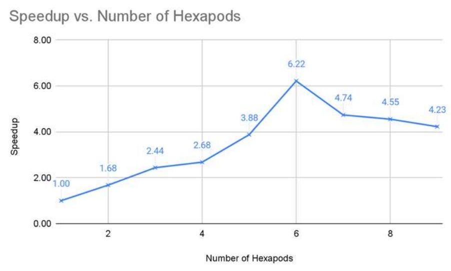
To test our identification we created a testing dataset consisting of images of different nonhuman and human objects and also had 15 of our friends as test humans to see their confidence scores as well as object classification accuracy when 20 meters away from our camera. We initially wanted to test various model frameworks such as many versions of YOLO, FOMO, detect-net, etc. However, as we worked on these object detectors we found out that they weren't all made to work on our hardware well. For example, YOLOv7 can only output at an average rate of 7 FPS meanwhile since YOLOv8 implementations can be hardware accelerated leveraging Nvidia's NITROS acceleration we can get 30 FPS from it.
We required that our object detection model has an Average Confidence of greater than 70% and a Mean Accuracy of greater than 90%. Our model hits this requirement because our average confidence is 83.1% and our mean accuracy is 98.2%.

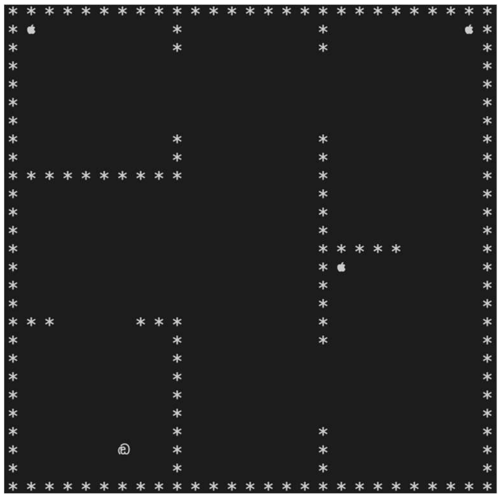
Architecture	**Average FPS**
YOLOv7	~7 FPS
Isaac ROS YOLOv8	~30 FPS

Test	**Requirements**	**Results**
Avg Confidence at 20m	>70%	83.1%
Mean Accuracy at 20m	>90%	98.2%

C. Tests for Scalability

For our scalability tests, we ran our search algorithm in a sample test environment that was 10 feet by 16 feet and had a target human randomly positioned in the space our test is set up to see how long it takes to find the target and see if it speeds up as we increase the number of robots.
We initially tested this with a simulated test of our robots and observed a roughly linear speedup of up to 5 robots. The goal that we set was a 1.5 times speedup with an additional robot which we achieved. We also tested this with our actual robots and also observed this same 1.5 times speedup.





C. Tests for Battery

We ran our hexapods under maximum stress (max speed, SLAM, and Object Detection running) and evaluated how long it takes for the hexapods to be unable to function from a full charge. We have 18 MG995 servo motors. From the manufacturer datasheet, each servo draws 10mA when idling and 1200mA when under max load. Although the servo is not running at max load given that the Hexapod is relatively light, we still assumed stall current to obtain a conservative estimate for our battery requirement. Since only half the legs move at a time, and the robot will be stopping frequently during path planning, we estimate that the servos have an active duty of 40%. From this data, we calculated that the servos would draw 5.674A from a 7.4V source.

Most 18650 batteries are 3.7V and have a capacity of around 3Ah, such as the Samsung 30Q 18650 cells. To meet our 1-hour battery life requirement, we decided to power the servos using 4 Samsung cells, with 2 wired in series and in parallel. This provides us with a 7.4V source and a capacity of 6Ah, which is enough to account for our worst-case power demand.

We did similar calculations for powering the Raspberry Pi and Jetson Orin Nano, and we found that the 10Ah battery packs we provide is able to support the embedded computers for 6 hours..

Number of Servos	18
Voltage (V)	4.8
Idle Current (A)	0.01
% Idling	60%
Stall Current (A)	1.2
% Stall	40%
Total Power (W)	41.990
Total Power (A at 7.4V)	**5.674**

Project Management

A. *Schedule*

As seen from the schedule above, we planned to front load most of our project to be done before Spring break. This entails the ordering and assembly of the Hexapods, as well as setting up the Jetsons with object detection models and ROS 2. After Spring break, we will set up VSLAM and the search algorithm, then begin to test and integrate all the various components of our project to ensure that it functions as expected.

However, as the semester continued, we quickly realized that this schedule was way too ambitious. Thankfully we had some built-in slack time to account for the many setbacks we hit along the way. For example, one of the setbacks was when our SD card with all of our code got corrupted in the middle of the semester, this was due to us running all of our docker code on the SD card and not purchasing a SSD for Jetson. Because of this we lost out on a lot of progress and had to redo our entire setup, this time with a SSD. On the bright side, this did allow us to learn a lot more about docker containers and set us up for more success down the road since we now understood more about how the environment was built and what pitfalls to be wary of.

A lot of other setbacks were related to the software integration of VSLAM and YOLOv8. This was mainly caused by the camera not being able to support the amount of data that it was required to collect to support both of these heavy processes.

B. *Team Member Responsibilities*

In our team, Casper was mainly in charge of the hardware and hexapod setup. Casper also developed and simulated the search algorithm that we later implemented in our robots. Casper did a lot of the 3D printing design aspect of the project. Casper and Akash worked on optimizing the battery usage.

Kobe was mainly in charge of the Isaac ROS pipeline and getting VSLAM and YOLOv8 to run correctly with eachother. He and Akash went through a lot of research for Isaac ROS, YOLOv8 and other versions of YOLO object detection, and VSLAM and how that compares with normal slam. They were also mainly in charge of the creation of the ISAAC ROS environment and setting up the docker containers…etc. and using them. Akash also implemented the communication node that hexapods used to communicate with each other. A lot of the other engineering responsibilities were shared among all team members and most of the time paired programming/engineering was the preferred method.

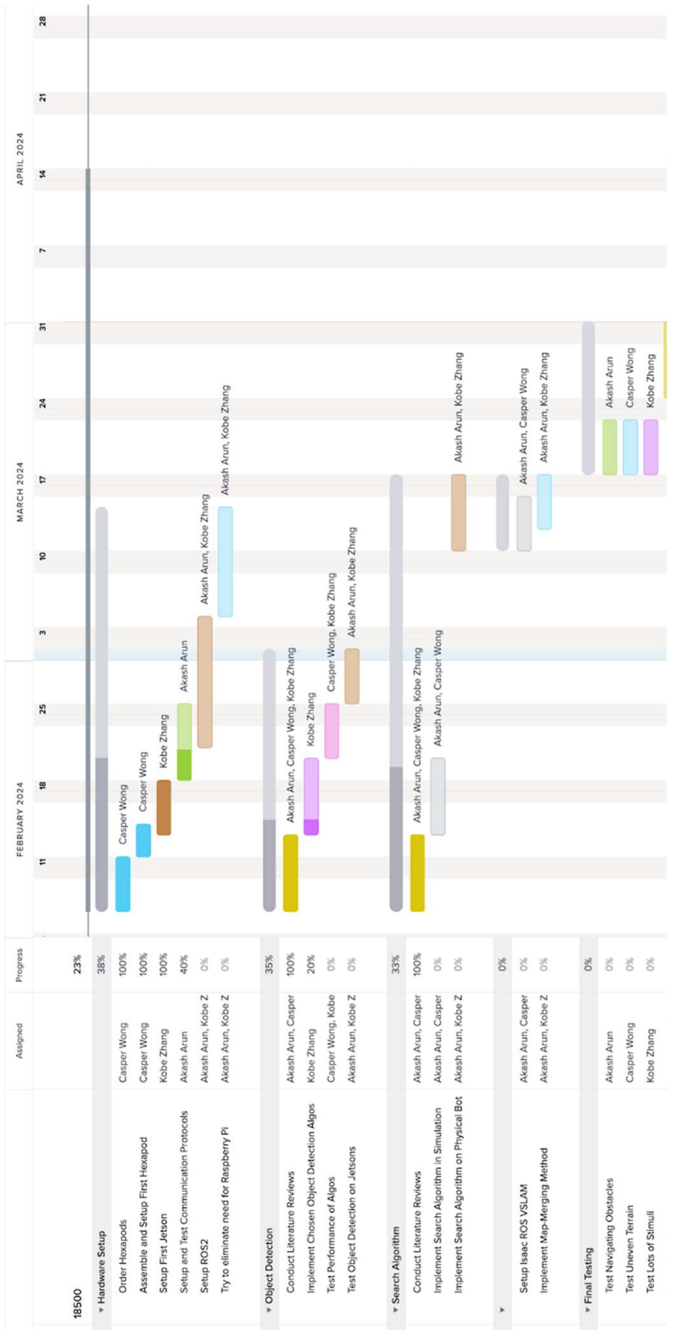


Figure 1. Schedule of work

C. Bill of Materials and Budget

The majority of the items that we purchased were used in our project, however there was a big change relating to the camera that we were going to use. More specifically, we no longer wanted to use the eYs3D camera since we wanted to get depth data for our obstacle avoidance and VSLAM processes. Because of this, we pivoted to using an Intel Realsense D435i Depth Camera. This allowed us to run YOLOv8 easier too since there were more supporting packages for the Intel Realsense.

Part	Supplier	Unit Price	Quantity	Total Price
FreeNove Big Hexapod	Amazon	$169.95	2	$339.90
FreeNove Big Hexapod	Marios Savvides	$0.00	1	$0.00
Raspberry Pi 3 Model B	ECE Inventory	$0.00	3	$0.00
Jetson Orin Nano	ECE Inventory	$0.00	3	$0.00
18650 Batteries + Charger	Amazon	$32.99	3	$98.97
64GB Class10 SD Card (2 pack)	Amazon	$14.99	3	$44.97
eYs3D Stereo Camera - EX8036	ECE Inventory	$0	3	$0
Intel Realsense D435i	ECE Inventory	$0	1	$0
			Total	$483.84

D. AWS Usage [if credits requested/used]

None Used

E. Risk Management

Some of the main risks we faced were related to us not being able to complete the integration of our full system in time. Fortunately, we were able to finish but our main risk management strategy was having alternatives that we knew worked that we could descale to if our reach goals were not possible. An example of this is when our VSLAM vo_pose was not being consistent due to the lack of visual tracking across a low fps camera feed, we had the option to pivot to hardcoded turning. This meant that hexapods would have a hardcoded time duration that they would turn in order to turn a full 90 degrees instead of just checking the vo_pose data to figure out their orientation.

VIII. Ethical Issues

Some ethical considerations for our project include the following:

1. Privacy Concerns – since we our hexapods use computer vision to identify humans, they are also capturing a lot of data of these humans during the process of search and rescue. We need to ensure that we build in privacy into future design iterations of our project to protect the privacy of the survivors that we rescue.
2. Fairness – a big consideration is that computer vision algorithms could be biased which could lead to hexapods treating people of different demographic groups differently. To prevent this we had our model pretrained on the COCO dataset which has a large amount of data of people from all demographics. In addition, we tested our model on our friends of different races, genders…etc. and found that our model was consistently recognizing all of them correctly.
3. Lastly, all autonomous robots will always have the ethical consideration of robot autonomy. Depending on the degree of autonomy we grant the robot, there can be a lot of ethical questions about who is liable or responsible if something happens that could harm a human's life. To address this, we made our hexapods not have any kind of method to directly interact with a human. All they do is search and alert the human search and rescue workers about the location of a potential human.

IX. Related Work

- RoboBees - autonomous flying robots with bee-like behavior developed at Harvard
- Swarmanoid - heterogeneous swarm of robots that work together to perform tasks like exploration and mapping
- SAFFiR - The Shipboard Autonomous Firefighting Robot - developed by US Navy, focusing on creating autonomous robots to assist firefighting on ships
- Swarm-SLAM - Sparse Decentralized Collaborative SLAM Framework for Multi-Robot Systems developed by MISTLab
- Inuktun - Rescue robots used in 9/11 and Hurricane Rescue Operations with tank-like treads.

X. Summary

Our system was able to meet most of the design specifications. The limit of the system comes from the camera not being able to process a large amount of camera data in time to serve both the VSLAM and YOLO processes. Some of our servos were very inconsistent and would break quickly. Additionally, cooling was an issues since the Jetson Orin Nano could overheat at certain times and die.

To improve our system's performance, we could have used LIDAR sensors instead of only relying on VSLAM. This would give us a really nice occupancy grid as well as rid us of a dependency on the visual frames for SLAM data. That would make our pose estimation a lot more consistent and would also reduce the load on the camera which was our main bottleneck. LIDAR would also help us better combat robot drift.

A stronger computer system would also allow us to avoid overheating, this could come in the form of a Jetson Xavier TX.

A. Lessons Learned

Don't be too ambitious with project scope, making something very polished is just as cool. Leave more time for integration and don't overestimate your abilities.

Glossary of Acronyms

SAR – Search and Rescue
SLAM –Simultaneous Localization and Mapping
RPi – Raspberry Pi
YOLO – You only look once
FOMO – Faster objects more objects
ROS - Robot operating system

References

1. "The 5 Common Challenges Facing Any Search and Rescue Team." Rigging Lab Academy, rigginglabacademy.com/the-5-common-challenges-facing-any-search-and-rescue-team/. Accessed 3 May 2024.
2. "Average Home Sizes." Thunder Said Energy, thundersaidenergy.com/downloads/average-home-sizes/. Accessed 3 May 2024.
3. "Versatrax Inspection Crawlers." Eddyfi, www.eddyfi.com/en/product/versatrax-inspection-crawlers. Accessed 3 May 2024.
4. "Take AI Learning to the Edge with Jetson." NVIDIA Developer Blog, developer.nvidia.com/blog/take-ai-learning-to-the-edge-with-jetson. Accessed 3 May 2024.
5. "Isaac ROS YOLOv8." NVIDIA Isaac ROS, nvidia-isaac-ros.github.io/repositories_and_packages/isaac_ros_object_detection/isaac_ros_yolov8/index.html. Accessed 3 May 2024.
6. "Isaac ROS Visual SLAM." GitHub - NVIDIA ISAAC ROS, github.com/NVIDIA-ISAAC-ROS/isaac_ros_visual_slam. Accessed 3 May 2024.
7. "A Brief Overview of Search and Rescue Robotics." Springer Link, link.springer.com/chapter/10.1007/978-3-642-21434-9_1. Accessed 3 May 2024.