Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

First Semester Report

Semi-Autonomous Convoy

Submitted to

Caitrin Eaton
Charles Stark Draper Laboratories
555 Technology Square, Cambridge, MA 02139
(617) 258-1000
ceaton@draper.com

by

Team 10
Draper Convoy

Team Members

Benjamin Hsu bhsu25@bu.edu
Sadman Kabir kabirs@bu.edu
Hugo Silmberg hsilmber@bu.edu
Nikhil Krishna nikhilk@bu.edu
Michael Moran mmoran3@bu.edu

Submitted: 12/08/2024

Table of Contents

Executive Summary - Michael

Semi-Autonomous Convoy
Team 10 - Draper Convoy

This project addresses the growing need for increased autonomy in semi-autonomous ground vehicle convoys, specifically for TurtleBot 4 robots. During collaboration with robotic teammates in the field, users often identify the need for greater autonomy as a bottleneck to safety and productivity. The final deliverable will be a fully functional and integrated vision-aided perception and control system for a convoy of TurtleBot 4 robots. This system will be capable of autonomous target recognition and tracking (humans, vehicles, convoy members), obstacle detection and avoidance, adaptive autonomous driving to follow a designated leader within the convoy, and comprehensive system integration and testing within a ROS2 framework to ensure that all subsystems work seamlessly together. The proposed technical approach begins with sensor characterization, calibrating cameras, IMUs, and LiDAR to optimize data collection for tasks such as target recognition, tracking, and obstacle avoidance. Automated target recognition (ATR) will leverage transfer learning from networks such as ResNet18 to identify convoy members, providing bounding boxes and confidence levels while minimizing false positives. Tracking systems will estimate the position and velocity of detected objects using computer vision, optionally fused with sensor data, while obstacle detection will rely on vision, LiDAR, or both for safe navigation. Control systems will integrate visual servoing and path planning to enable adaptive speed regulation and obstacle avoidance. All components will be integrated within a ROS2 framework, with rigorous testing to ensure performance and fault tolerance. Extensions, such as SLAM or swarm behaviors, may be incorporated as time allows. This approach includes innovative features such as transfer learning-based recognition for accurate target identification, vision-based adaptive tracking for real-time convoy dynamics, and energy-efficient motion primitives for optimized path planning.

1.0 Introduction - Benjamin

1.1 Problem Definition and Background

The Charles Stark Draper Laboratory has recognized the need to enhance the autonomy of semi-autonomous ground vehicle convoys to improve safety and productivity during field operations. Currently, limited autonomy creates challenges in coordinating robotic teammates, hindering effective collaboration. The desired solution will enable each TurtleBot 4 robot to autonomously track and follow convoy members, a human leader, or a lead vehicle using video feeds. Additionally, the robots will demonstrate robust driving behaviors, such as adjusting their speed to match the leader and avoiding obstacles, ensuring cohesive convoy movement.

Semi-autonomous convoys combine autonomous navigation with human oversight, offering flexibility across multiple applications such as automated warehouse logistics, hospital supply transport, disaster response, and military missions. For these applications to succeed, the convoy system must be able to navigate reliably and avoid obstacles in dynamic environments.

1.2 Goal

To address the aforementioned factors, our goal is to develop vision-aided perception and control algorithms for the TurtleBot 4 robots. By leveraging onboard sensors like the OAK-D-LITE camera [7.3.4] and RPLIDAR-A1 LiDAR [7.3.3], we aim to create an integrated, modular system that autonomously detects and tracks leaders, avoids obstacles, and dynamically adjusts to environmental conditions. This approach is designed to overcome the current limitations of autonomy, allowing the robots to collaborate effectively in convoy formation without continuous human intervention.

Fig 1. Image of a Turtlebot 4 Lite robot with the OAK-D-LITE camera and RPLIDAR-A1 LiDAR visible

1.3 General Approach

Our system will focus on three core components: leader identification, obstacle detection, and adaptive motion control. We will utilize vision-based perception techniques, initially using pre-trained YOLO models and later progressing to custom recognition algorithms, to enable the robots to identify and track convoy leaders, whether human or robot [7.3.9]. Obstacle detection will rely on LiDAR data to identify and navigate around obstacles, ensuring safety and smooth convoy movement. The motion control system will integrate these inputs, allowing the robots to plan their paths and adjust their speed to match the leader’s while maintaining optimal safety.

The system will be built using the ROS2 framework, ensuring modular, scalable communication between the different nodes and components [7.3.10]. Key features of our solution include machine learning for leader recognition, efficient path planning for obstacle avoidance, and energy-efficient control algorithms to extend the robots' operational range. By addressing these core areas, our solution will enhance the autonomy of semi-autonomous convoys, improving collaboration, safety, and efficiency.

1.4 End Goal

This project will provide a flexible, reliable, and scalable solution for semi-autonomous convoys, adaptable to a wide range of applications. Time permitting, we also plan to explore additional functionalities, such as integrating compatibility with remotes for manual control, open-source SLAM for improved localization, state estimation for better decision-making, and swarm behaviors for more coordinated actions among multiple robots.

2.0 Concept Development - Michael

2.1 Problem Reiterated

Our customer's problem centers around the need to enhance the autonomy of semi-autonomous convoys composed of TurtleBot 4 Lite Robots. These robots must be able to collaborate with human teammates in the field to improve safety and productivity in dynamic environments. The core requirements of the solution to this problem include the ability to autonomously recognize and track other convoy members, a designated lead vehicle or person, and detect and navigate obstacles safely while maintaining the cohesion of the convoy. Our customer's problem translates into specific engineering challenges such as developing robust vision-based perception algorithms, safe and efficient tracking and control systems, and thorough integration for real-time coordination between the convoy members. These solutions must be scalable, reliable, and adaptable to different field environments and varying convoy sizes.

2.2 Approach

Our conceptual approach addresses our customer's problem through a modular framework. We propose a vision-aided system that combines sensor calibration, transfer learning for target recognition, a pre-trained YOLO model for human leader detection, real-time object recognition and tracking, obstacle detection and avoidance, and control algorithms [7.3.9]. This framework integrates all components using ROS2 nodes to ensure scalability and seamless communication across systems. Thorough testing and iterative design will ensure the system's robustness and safety.

We decided to pursue this approach for its balance of innovation, feasibility, and alignment with our customer's requirements. The use of transfer learning and our pre-trained YOLO model enables us to leverage pre-trained models to identify other convoy members and leaders, reducing our development time while ensuring an accurate and reliable system. ROS2 allows us to interface with the TurtleBot 4 Lite's built-in operating system, and facilitates our modular system design. This simplifies our integration and testing processes, while also enhancing our scalability.

2.3 Abandoned Solution

One of the solutions that we had considered, yet have abandoned at this point, was the implementation of our own custom neural network for perception and detection purposes. By building our own detection model from scratch, the architecture could have been optimized for specific tasks such as recognizing other convoy members, tracking their positions, and identifying obstacles. This approach would have given us full control over the design and training process, potentially giving us a model that is extremely efficient and highly specialized for the tasks at hand. We decided not to take this path for a few reasons. First off, the development of our own custom neural network would require a substantial amount of training data, which we would not have on our TurtleBot 4 Lite. Attempting to collect enough of this data and label it correctly would have been extremely time-consuming and require much hands-on work from our team. Secondly, the training process for a newly built neural network would have required us to obtain high-performance GPUs and train the model for extended periods of time to find what works and optimize the model to our needs.

2.4 Chosen Solution

With all of this in mind, it made much more sense to go with our current approach of using transfer learning with ResNet18. This offers us a much more practical alternative that balances the performance we need with the time that we have to complete this project. With ResNet18 being a pre-trained model that has been extensively validated for image processing, it provides us with a very reliable foundation for adaptation to our own needs. By fine-tuning ResNet18 for our specific project requirements, we can leverage its robust image recognition capabilities without the need for extensive training and large datasets. Our current approach significantly reduces our development time while still achieving high accuracy and reliability within the system, making it a more suitable solution for our project's necessary constraints.

3.0 System Description - Hugo

3.1 Sensors

To provide a solution to our clients problem we will break down different behaviors that would be desired for enhanced autonomy of robots. These behaviors include obstacle avoidance, follow the leader behavior (human, vehicle, or another TurtleBot), and possibly convoy behavior. We won't need to add any additional sensors to our design as the TurtleBot 4 Lite comes with several onboard sensors [7.3.5]. Below is a description of each sensor and what they are used for:

> Camera [7.3.4]:
> The TurtleBot 4 Lite comes with an OAK-D-LITE camera which includes a colored camera lens along with two monochrome stereo camera lenses that can be used for detecting distance. The OAK-D-LITE also includes the IMU for the device which measures the device's linear acceleration, angular velocity, and orientation.
>
> LiDAR [7.3.3]:
> The TurtleBot 4 Lite comes with a RPLIDAR-A1, this is a 2D LiDAR so it will return scans across a 2D plane. It returns up to 8000 points per scan depending on the frequency we choose. The maximum distance this LiDAR can return data back is 12 meters. It will return the angle, distance, and quality of each point in a scan.

3.2 System Architecture

Our solutions are based on the system block diagram in Figure 2. The TurtleBot 4 Lite uses ROS2 which makes it easier for us to compartmentalize our subtasks for the project. The behaviors we are aiming for can be broken into two main parts: leader detection and obstacle detection. Once these two tasks are done in the ATR (leader identification node), obstacle detection node, and distance calculation node.

3.3 Perception and Tracking Nodes

The distance calculation node will use the LiDAR data (specifically the angle and distance of each point) to determine objects which will then be sent to the obstacle detection and ATR nodes. The LiDAR node will first create one-dimensional linear objects and then merge these into two-dimensional objects if there are overlapping endpoints. This creates an array of two-dimensional objects which represent both the leader, possible other convoy members, and obstacles.

The ATR node is where we identify the leader and other convoy members. For this node we will use the colored camera for this identification, we will start with a pre-trained YOLO model for human detection for the leader. This YOLO model is an off-the-shelf, pre-trained object detection model that we imported from an Ultralytics library [7.3.9]. We use OpenCV to feed live camera data into the model, detecting objects in view. The detected objects are then enclosed in bounding boxes, with their labels displayed above each box. In addition, a confidence rating is provided, indicating how certain the model is about the accuracy of its classification. A centroid is also generated onto the bounding box and its details are available for use within the codebase. This centroid is the working reference point to determine the relative angle between the camera

and objects in frame. Later, we plan to use our own built recognition models that could identify specific individuals using some sort of identity like an easily identifiable shape from distances of up to 12 meters or perhaps a barcode. We will also aim to use transfer learning on a pretrained ResNet18 neural network to identify TurtleBot 4 Lite's in order to identify other convoy members and to distinguish between the leader and convoy members. However, because the TurtleBot's overall complexion is common (black and circular) this could be easily confused with other objects so we may still add easily identifiable shapes on the convoy members to keep the accuracy high. The ATR node will then match the identified leader's angle with an angle within the range of the LiDAR counterpart's object along with possible convoy members. This allows us to treat the remaining LiDAR-calculated objects as obstacles to avoid which will be confirmed with a binary value of 0 in the obstacle detection node.

3.4 Motion Control Nodes

The acceleration node and rotation node will use the onboard IMU to store the current state of the robot including linear acceleration, angular velocity, and orientation. The acceleration node will take the dataset of objects from the perception nodes to slow the motors to a stop if it gets too close to an obstacle or if the leader stops moving, the max speed of the turtlebot is 0.31 m/s so this will be a small difference in speed overall [7.3.5]. The rotational node will also take in the perception data and use it to turn towards the leader at all times unless it is about to hit an obstacle. This node will read "red zones" areas determined by the distance and relative angle of nearby obstacles, the presence of these "red zones" overrides follow the leader behavior and begins obstacle avoidance behavior, it will maintain a heading within 180 degrees of where the leader was last spotted by the LiDAR so it will be able to continue following the leader once it moves around the obstacle with the smallest chance of losing the leader. Avoiding the obstacles will mean finding the nearest gap in obstacles and moving through it, after this if it can't find the leader again it will begin spinning 360 degrees until the camera can see the leader again. The TurtleBot 4 also has a pre-built model called NAV2 for path planning which we can use with this current plan to make it more efficient as it will only have to search through half the LiDAR data [7.3.2].

3.5 ROS2 and Node Communication

ROS2 allows us to use different languages for each node which makes it easy to use the best language for each functionality. For the perception nodes, we decided on using Python since it was much easier to use machine learning models. Although Python is not efficient in terms of time and electricity so for the other nodes which primarily are for pure calculations and processing data we will use C++ as it is very efficient for both time and electricity, both important as the robot will be actively moving and has a battery life of 2.5-4 hours [7.3.5]. For communication between nodes, ROS2 uses a publisher-subscriber system where each node is built to know which subscriber to listen to and each subscriber will output the data desired. Figure 2 shows this between our nodes and what is being communicated between these nodes. ROS2 has unique message types and for our communications we settled on using a polygon message [7.3.1]. This message type is an array of Point32 messages. Point32 messages hold 3 float32 data points, usually used for cartesian coordinates however we will use it to hold the distance, angle, and the binary value representing whether or not an object is the leader or not (1 being the leader, 0 being an obstacle, and possibly 2 being a convoy member although it would no longer be binary this shouldn't affect the design as it's being stored as a float32). Using a

polar coordinate system instead allows us to store the leader value as an extra parameter which cuts down on the amount of messages between nodes for efficiency. Each polygon message will be an object which will also make it easier to then input each message into the desired class of object for use in every node.

3.6 Pseudo code:

```
        create 1D shapes        // in distance calculation
        create 2D shapes        // in distance calculation
send to ATR and obstacle detection
        identify leader/convoy members        // in ATR
        identify obstacles        // in obstacle detection
send to acceleration and rotation calculations
        calculate path to the leader avoiding obstacles // in rotation calculation
        turn motors towards leader or to free path // in rotation calculation
        change speed depending on obstacles and the leader // in acceleration calculation
send output to the motor
```

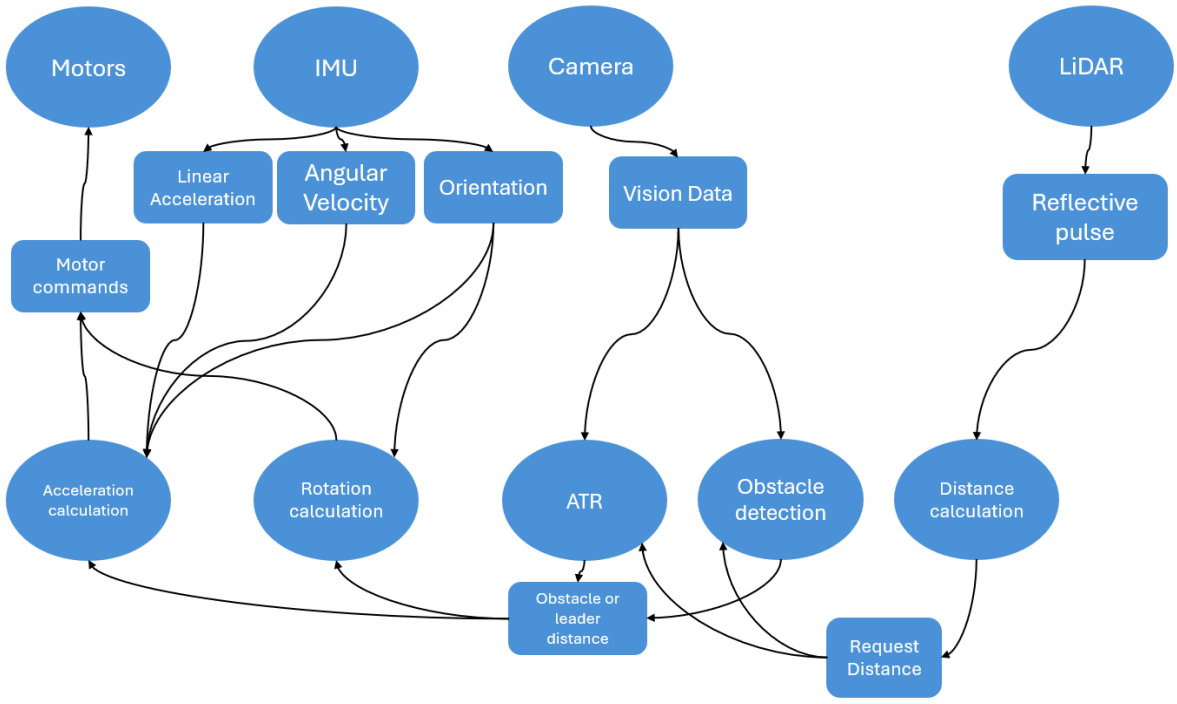


Figure 2. System block diagram of ROS2 node system for TurtleBot 4 Lite

4.0 First Semester Progress - Nikhil

4.1 Simulation Dependence

Our progress for this semester included getting a virtual Gazebo simulation of a TurtleBot to run with a majority of the expected behavior of a single TurtleBot, including human detection and relocation. We were limited this semester in obtaining the physical TurtleBot robots, so most of our progress took the shape of virtual simulation, nonetheless, we intend to transfer our sensor functionality to the physical TurtleBots when we receive them in the spring.

4.2 Node Development

The early stages of our progress included developing the ROS nodes to support the sensor functionality of the TurtleBots. These included motor control, object detection, and path planning nodes, along with some other input/output and data nodes. We developed an object hierarchy initialized at our LiDAR node: groups of point objects being combined to form a 2D object, a vector of these 2D objects representing the group of unidentified objects the TurtleBot views at a specific timestamp. Additionally, object detection was established using a YOLO algorithm to identify humans and place bounding boxes around them [7.3.9]. The rest of our progress involved figuring out how to obtain the Gazebo software for our use, and then translating our node logic to Gazebo.

4.3 Simulating with Gazebo

For the gazebo simulation, we were able to transfer the human detection node and motor control node to the virtual space. Of course, we plan to incorporate the rest of the nodes, as well as other sensor data inputs that conflict with the Gazebo software (ex. LiDAR) once we have access to the physical TurtleBot robots. That being said, the virtual aspect of Gazebo did offer some extra testing options that we were able to configure, one being including a secondary option of remote control driving of the TurtleBot with a keyboard, in addition to the default autonomous driving mode.

4.4 Results

In terms of some key results from our First Deliverable Testing, we were able to:

- Identify a human model in the Gazebo simulation (our test showed identification even when the human wasn't fully in view, as long as the waist was visible)
- Place a bounding box around the human model with a confidence value displayed and a centroid
- Engage motors and move the robot toward the human model, maintaining a confidence level of >=80 throughout the seeking phase
- Stop a distance away from the human once close enough (still in range of identification)
- Demonstrate auto relocate behavior: if a human model is moved out of the robot's vision it will perform a 360-degree rotation to relocate the human model, and once spotted will repeat the seeking behavior

This would summarize the progress we made in the first semester.

5.0 Technical Plan - Sadman

The following technical plan, organized by numbered tasks, is what we aim to accomplish between the end of this semester and the functional testing of our project next semester in April.

1. Implementation of a Functional LiDAR subsystem.

   An integral component in the functionality of the entire system, is the ability to calculate distance between the robot hardware and entities in the physical world. This distance is required to calculate the speed of the motors, whether the robot should speed up or slow down, when motors should start turning, whether the robot moves at all and so on. We aim to create a functional LiDAR subsystem that is able to make use of our hardware's 2D LiDAR sensor, and retrieve distance data between the robot and physical entities [7.3.3]. This sensor data is then parsed as distance and angle, that is, the distance the robot is from some physical mass and the angle at which the mass exists relative to the direction the robot is facing. The parsed information must then be published onto appropriate ROS2 topics that the other nodes, motor control, and path planning, can subscribe to and utilize the information from. This subsystem should provide accurate distance and angle information regarding objects in view, and should be effective in both simulation and real-world environments on physical hardware. *Lead: Hugo, Assisting: Benjamin.*

2. Finalization of the dataset for object detection models.

   As the convoy system requires the automated target recognition of designated humans, vehicles as well as fellow convoy members, we will need to train an object detection system capable of this. And to train these models, we will need a dataset of curated, preprocessed data points. Remaining work to be done on this includes generating more data points of the TurtleBot4 Lite, which the system must be trained to recognize and follow, to minimize underfitting. The other label to be recognized is likely our "emitter" or "recognizable symbol" that will designate humans/vehicles as leaders. Both images of the TurtleBot4 Lite and this recognizable symbol must be annotated and preprocessed in order to use in training our model. *Lead: Sadman, Assisting: Michael.*

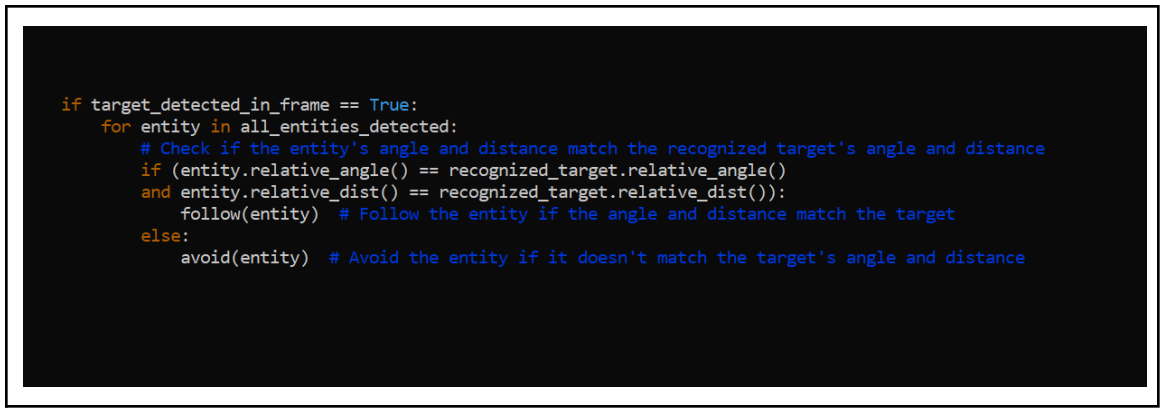
3. Training/Fine-tuning of object detection models.

   With a comprehensive dataset, we must train and fine-tune our object detection models to minimize testing error. Ideally, we wish to minimize error rates to be below 5% and have a confidence rating of greater than 95% above each bounding box. This will be an iterative process and fine tuning will continue so long as we continue to see improvements. A successful model will be able to correctly classify both convoy members as well as entities marked with our "recognizable symbol" for designated targets. *Lead(s): Sadman and Michael.*

4. Implementation of Obstacle discernment.

   The system must be able to distinguish between targets and obstacles, and subsequently avoid these obstacles to continue following targets. The perception team must then use LiDAR data, to distinguish between entities. As our LiDAR subsystem records and

publishes the angle and distance of objects in the robot's environment, we can cross reference this with the data from our object detection system. This means that if the recognized target's relative angle and distance (done by the perception nodes) do not match the same attributes for the entities detected by the LiDAR system, then those entities that have differing values *are not the target,* and thus obstacles to be avoided. For example, see the following high-level pseudo-code:



```
if target_detected_in_frame == True:
    for entity in all_entities_detected:
        # Check if the entity's angle and distance match the recognized target's angle and distance
        if (entity.relative_angle() == recognized_target.relative_angle()
        and entity.relative_dist() == recognized_target.relative_dist()):
            follow(entity)  # Follow the entity if the angle and distance match the target
        else:
            avoid(entity)  # Avoid the entity if it doesn't match the target's angle and distance
```

Lead(s): Sadman and Michael.

5. Path-planning implementations/Obstacle Avoidance.

   Once obstacle discernment is working, path planning will be our next challenge. Our current implementation design works by mapping areas that are considered obstacles/obstacle-filled. These areas are to be avoided and the leader followed provided the leader remains in frame. In the event that the leader is no longer in frame, the robot must retain a heading of 180 degrees from where the leader was last spotted. In other words, the robot should not face backward to search for the leader but keep facing the general direction that the leader was last spotted in. Given these obstacle-filled areas, the robot must find "openings", to move towards and continue searching for the leader. NAV2, a pre-built navigation model is expected to be used to optimize this path planning [7.3.2]. Once the leader is seen once again, follow-the-leader behavior re-engages. *Lead: Nikhil, Assisting: Benjamin.*

6. Integration and Testing.

   While each individual subsystem will be thoroughly tested throughout all stages of development, the integration of all components into a fully cohesive, reliable physical and simulated system will present our greatest challenge. Every subsystem, from motor control and path planning to perception, must be rigorously tested once integrated to identify edge cases, performance or accuracy losses, bugs, and potential failures. Any subsystem issues found must be corrected for use within the complete system. In addition, the synergy between the subsystems will be evaluated by how well the complete system completes its functions. Latency, accuracy, and precision will be optimized for as long as we have time. *Lead: Hugo, Assisting: Everybody.*

7. Implement Stretch Goals, time permitting.

   If time allows and the rest of the system is completed to a satisfactory degree, we plan to implement SLAM (Simultaneous Localization and Mapping) algorithms, swarming behavior, remote control via a game controller, and/or a GUI that visualizes the robot's camera feed, what it sees and the actions it takes. *Lead: Everybody.*

6.0 Budget Estimate - Benjamin

Item	Description	Cost
1	TurtleBot 4 Lite	$1,336.99
2	TurtleBot 4 Lite	$1,336.99
	Total Cost	$2,673.98

Our project requires two Clearpath Robotics TurtleBot 4 Lite Mobile Robots, with the cost detailed on the RobotShop product page [7.3.6]. Despite having no information on where Draper will source the hardware, it can be reasonably assumed that the price for a Turtlebot 4 Lite will be approximately $1,350. This budget is notably higher than the $750 allocated by Boston University, the remainder of the cost will be covered by our client, Draper. Additionally, we have not included sales tax in the estimate, as the purchase may qualify for tax exemption under Section 501(c)(3) of the Internal Revenue Code [7.3.7].

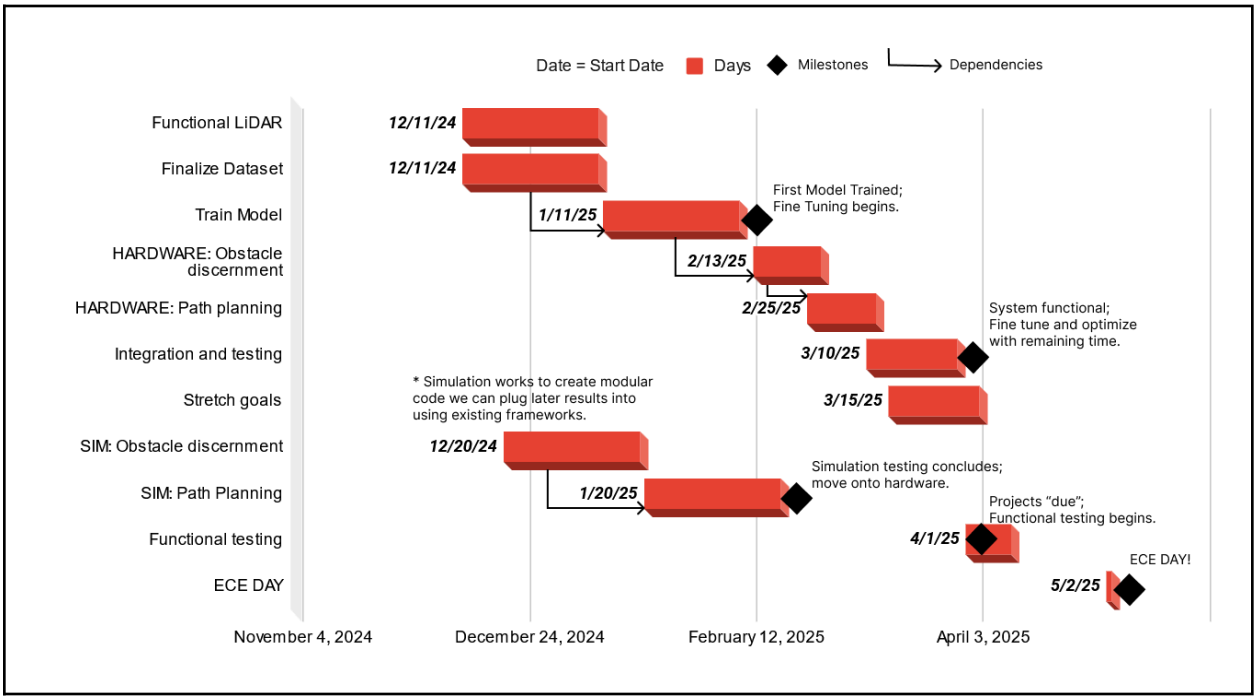
7.0 Attachments - Sadman, Hugo, & Michael

7.1 Appendix 1- Engineering Requirements

Team 10 Team Name: Draper Convoy
Project Name: Semi-Autonomous Convoy

Requirement	Value, range, tolerance, units
Target Recognition	Detect convoy members and the leader with a confidence level of at least 95%
Tracking Precision	Position and velocity estimates for detected objects should maintain an error margin of less than 5%
Obstacle Detection Range	The system must detect obstacles within a range of 1.5 meters with 95% reliability
Collision Avoidance	The system must safely avoid obstacles with a response time of less than 300 milliseconds when detected in specified range
Control Responsiveness	Respond to speed and distance changes within 0.5 seconds of detected variation
System Integration	Communication within ROS2 framework must achieve real-time data exchange in under 50 milliseconds
Scalability	The system should support scalability to a convoy of at least 5 robots (in simulation)

7.2 Appendix 2 - Gantt Chart



* Simulation (SIM) of the system works in parallel to create modular code we can plug later developments into.

7.3 Appendix 3 - Other Appendices

1. Ros. (n.d.). *geometry_msgs/Polygon Message*. Retrieved from https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Polygon.html

2. TurtleBot. (n.d.). *TurtleBot4 User Manual: Tutorials - TurtleBot4 Navigator*. Retrieved from https://turtlebot.github.io/turtlebot4-user-manual/tutorials/turtlebot4_navigator.html

3. SLAMTEC. (n.d.). *SLAMTEC RPLIDAR A1*. Retrieved from https://www.slamtec.com/en/lidar/a1

4. Luxonis. (n.d.). *OAK-D Lite Documentation*. Retrieved from https://docs.luxonis.com/hardware/products/OAK-D%20Lite

5. Clearpath Robotics. (n.d.). *TurtleBot 4 Overview*. Retrieved from https://clearpathrobotics.com/turtlebot-4/

6. RobotShop. (n.d.). Clearpath Robotics TurtleBot 4 Lite Mobile Robot. Retrieved from https://www.robotshop.com/products/clearpath-robotics-turtlebot-4-lite-mobile-robot

7. Massachusetts Department of Revenue. (n.d.). *AP 101: Organizations exempt from sales tax*. Massachusetts Government. Retrieved from https://www.mass.gov/administrative-procedure/ap-101-organizations-exempt-from-sales-tax#:~:text=Sales%20to%20%C2%A7%20501

8. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788. https://doi.org/10.1109/CVPR.2016.91

9. Ultralytics. (n.d.). *How can I get started with YOLO? Installation and setup*. Ultralytics. https://docs.ultralytics.com/#how-can-i-get-started-with-yolo-installation-and-setup

10. Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics, 7*(66), eabm6074. https://doi.org/10.1126/scirobotics.abm6074

11. Team information sheet

Team Member	Biography	Email	Phone Number
Hugo Sillmberg	4th-year electrical engineering major at Boston University. Looking to learn more about systems engineering and cloud computing, he specialized in backend software engineering and cloud computing security during time at Boston University.	hsilmber@bu.edu	(914) 310-6862
Nikhil Krishna	3rd-year computer engineering major at Boston University interested in cybersecurity and systems engineering. He also enjoys exploring machine learning algorithms to improve practical applications.	nikhilk@bu.edu	(609) 937-2295
Michael Moran	4th-year computer engineering student at Boston University looking to learn more about the realms of machine learning and cybersecurity. Interested in low-level systems including operating systems and hardware-software integration.	mmoran3@bu.edu	(508) 740-6025
Benjamin Hsu	4th-year computer engineering student at Boston University interested in all things embedded, robotics, and video game design.	bhsu25@bu.edu	(617) 860-9997
Sadman Kabir	4th-year computer engineering student interested in IC design and fabrication, Machine Learning as complement to human abilities, and computer architecture.	kabirs@bu.edu	(718) 593 2041
The Charles Stark Draper Laboratory Inc.	A not-for-profit research and development company based in Cambridge, Massachusetts. Founded in 1932 by Dr. Charles Stark Draper. Has played a key role in the Apollo program and currently works in defense, space exploration, biomedical systems, and energy innovations. Draper is the sponsor for this project at Boston University.	ceaton@draper.com	(617) 258-1000