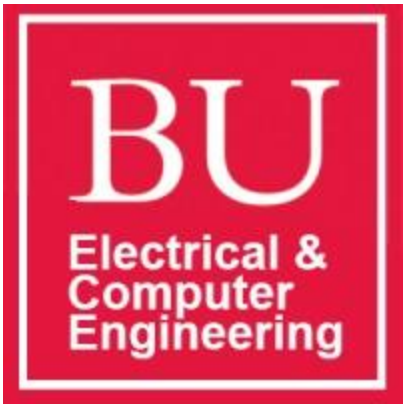

Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

Semi-Autonomous Ground Convoy

User's Manual

Submitted to
Caitrin Eaton
Charles Stark Draper Laboratories
555 Technology Square, Cambridge, MA 02139
(617) 258-1000
ceaton@draper.com
by
Team 10

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: 4/18/202

Table of Contents

Executive Summary

The Convoy addresses critical safety and autonomy bottlenecks in dynamic environments through an integrated vision-aided perception and control system for TurtleBot 4 robots. Our semi-autonomous solution enables robust target recognition, tracking, obstacle avoidance, and adaptive following behaviors. The system incorporates transfer learning-based recognition, multi-modal sensor fusion, and advanced control algorithms within a comprehensive ROS 2 framework. The Convoy delivers real-time follow-the-leader behavior for designated targets, elegant and simple swarming, as well as predictive control systems for seamless navigation.

1.0 Introduction

1.1 Problem Definition and Background

The Charles Stark Draper Laboratory has recognized the need to enhance semi-autonomous ground vehicle convoy autonomy to improve field operations safety and productivity. Our TurtleBot convoy system enables robots to autonomously track and follow convoy members or human leaders using video feeds, with adaptive speed control and obstacle avoidance. This system bridges current limitations in robotic coordination across warehouse logistics, hospital transport, disaster response, and military applications, navigating reliably through dynamic environments.

1.2 System Overview

The convoy uses TurtleBot 4 Lite robots in a leader-follower architecture where the lead robot tracks a high-visibility-vested human while followers track tennis ball markers on preceding robots. The system integrates YOLO-trained models with OAK-D-LITE cameras, SLAM and Nav2 for navigation, RPLIDAR-A1 sensors for obstacle detection, all operating on the ROS 2 Jazzy framework for modular communication.

Figure 1.1: Image of a TurtleBot 4 Lite robot with the OAK-D-LITE camera and RPLIDAR-A1 LiDAR visible

1.3 Capabilities and Limitations

The system features >90% confidence target recognition, 1.5-meter obstacle detection, 300ms collision avoidance, and 0.5-second control responsiveness. Its sophisticated path planning allows robots to navigate around obstacles and maintain formation when the leader turns corners, intelligently adapting routes through complex environments. With sub-50ms data exchange and multi-robot scalability, the convoy operates cohesively. Limitations include 2.5-hour charging requirements, moderate speed constraints, occasional manual SLAM initialization, and reduced performance in crowded environments.

1.4 Manual Structure

This manual provides setup and operation information, with Section 2 covering installation, Section 3 detailing procedures, Section 4 explaining the technical architecture, Section 5 addressing costs, and appendices containing specifications. Users should follow all installation instructions and safety warnings carefully.

2.0 System Overview and Installation

2.1 Overview block diagram

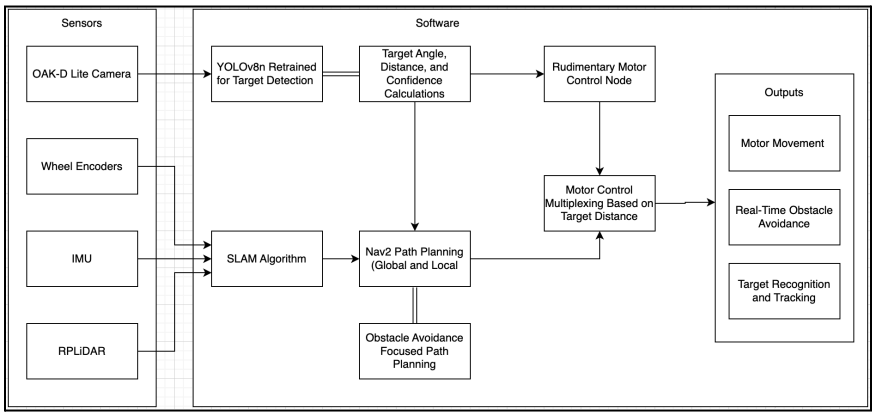

Figure 2.1 Overall Flow of Our System

Figure 2.1 describes the flow of information through our system, with the inputs coming from the onboard sensors. The information gathered through our sensors is then fed through our software pipeline, whose output is then given back to the robot's motor system.

2.2 *User interface*

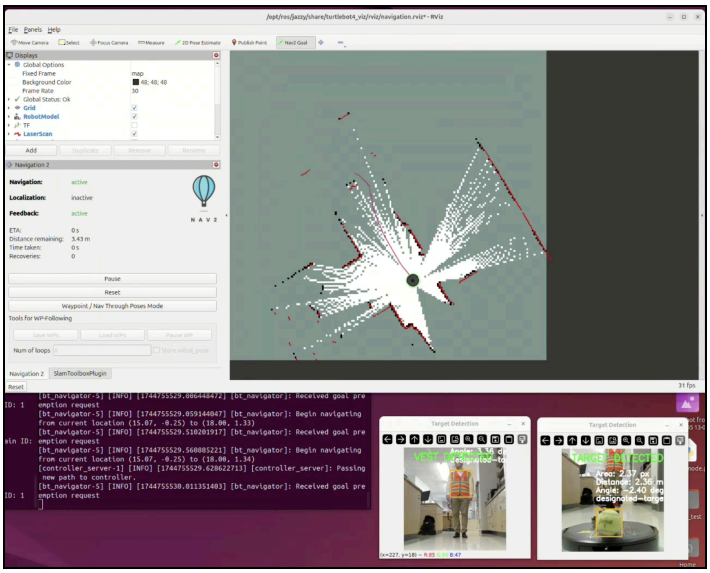

Figure 2.2 GUI and Visualization for User

There is no setup required for our GUI and user interface. Upon launching the bash scripts, RViz will launch with a visualization of the SLAM mapping and Nav2 path planning, and the camera feeds from each bot will open, allowing the user to see what the TurtleBots are doing.

2.3 Physical description

Figure 2.3 Photograph of TurtleBot 4 Lite

2.4 Setup & Installation

2.4.1 Software Setup

Software Required:

- Ubuntu 22.04
- ROS 2 Jazzy Jalisco
- ROS 2 Packages from GitHub

To install and set up ROS 2 Jazzy Jalisco:

On a machine running Ubuntu 22.04, follow the instructions below, or at:
https://docs.ros.org/en/jazzy/Installation/Ubuntu-Install-Debs.html

Run the following commands in a terminal window:
Set Local:

```
locale  # check for UTF-8
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale  # verify settings
```

Enable Required Repositories:

```
sudo apt install software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install curl -y
sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
-o /usr/share/keyrings/ros-archive-keyring.gpg
```

```
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/ros-archive-keyring.gpg]
http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee
/etc/apt/sources.list.d/ros2.list > /dev/null
```

Install ROS 2:

```
sudo apt update
sudo apt upgrade
sudo apt install ros-jazzy-desktop
```

Setup Environment:

```
# Replace ".bash" with your shell if you're not using bash
# Possible values are: setup.bash, setup.sh, setup.zsh
source /opt/ros/jazzy/setup.bash
```

Install Dependencies:

```
sudo apt install python3-colcon-common-extensions
python3-rosdep python3-vcstool git
```

Create a ROS 2 Workspace:

```
mkdir -p ~/ros2_ws/src
```

To obtain our ROS 2 packages, clone our repository and set up ROS 2 Workspace:

```
git clone https://github.com/corndog-overflow/the_convoy.git
# Replace src/ directory in ~/ros2_ws/src with src/ from our
repository
# Navigate to the workspace root and build the workspace:
cd ~/ros2_ws
colcon build
```

To run the nodes, obtain bash scripts from the cloned repository and run:

```
./launch_follower_bot.sh #(follower bot - domain defaults to 0)
./launch_leader_bot.sh 1 #(leader bot - 1 to specify domain of
the second bot)
```

2.4.2 Hardware Setup

Hardware Required:

- Computer
- Wi-Fi Router
- 2 (Minimum) TurtleBot 4 Lites with docking stations

1. Power on the Wi-Fi router
    - No internet connection is required, simply creating a host for the TurtleBots to interface with

2. After unboxing the TurtleBots, plug the docking stations into an outlet and set the TurtleBots on the stations with the camera facing the elevated section of the station. Wait 2.5 hours for the TurtleBots to charge fully.

3. Once charged, remove the TurtleBots from the charging dock and allow them to power on

4. To connect the TurtleBots to your Wi-Fi network, follow the instructions at:
    a. https://turtlebot.github.io/turtlebot4-user-manual/setup/basic.html

5. To connect the TurtleBots with the router and computer during runtime, we used ROS 2's simple discovery protocol for multicasting. The instructions for this can be found here:
    a. https://turtlebot.github.io/turtlebot4-user-manual/setup/networking.html

6. It is recommended to label the Turtebots, the leader being bot 2.

3.0 Operation of the Project

3.1 Operating Mode 1: Normal Operation

1. To set up for operation, take each TurtleBot off the docking station and place it in a line on the floor, the leader bot in front, facing the direction of desired travel. You should notice the LiDAR begins spinning, and the lights around the power button begin to light up. Wait for each TurtleBot to chime, this will let you know they have fully booted up (takes around 5-10 minutes)

2. After each TurtleBot has chimed, turn on your desktop and open two terminals (can be done by typing cmd into search if there is not a saved shortcut). In each terminal, type in the following:

Leader	Follower

`cd ~/ros2_ws`
`source /opt/ros/jazzy/setup.bash`
`source install/setup.bash`
`export RMW_IMPLEMENTATION=rmw_fastrtps_cpp`
`export ROS_DOMAIN_ID=0`
	`cd ~/ros2_ws`
	`source /opt/ros/jazzy/setup.bash`
	`source install/setup.bash`
	`export RMW_IMPLEMENTATION=rmw_fastrtps_cpp`
	`export ROS_DOMAIN_ID=1`

3. Then type in:

```
ros2 topic list
```

In each terminal, this will open topics, look for /oakd topics, if you see them, then the startup has initialized properly.

4. In bot 1s terminal type:

```
./launch_leader_bot.sh 1
```

In bot 2s terminal type:

```
./launch_follower_bot.sh
```

This will run the bash files that boot up all the nodes for each bot. It is recommended that you start and initialize one bot at a time, starting with the follower bots, for ease of navigating the vast number of terminal windows.

5. Once step 4 has been run for the follower bot, you will see a camera feed open for this TurtleBot and several terminal windows. To set the follower as ready to go, click the command prompt in the foreground and press “p”; this takes the bot out of manual mode and into automatic mode (manual mode instructions are always displayed at the top of this terminal for easy use if desired). You will notice it will move closer to the leader bot but not crash into it; this is good.

6. After step 5 is complete, return to step 4 to run the leader bash file you will see RViz open and more terminal windows. RViz will slowly boot up, and you will be able to see the initialized LiDAR data. Now, make sure you can find the original terminal for the commands you ran for the leader, which should display:

```
"Press any key to continue with the remaining nodes..."
```

   now click on this terminal window and press any button.

7. Now find the terminal window that is second foremost of the terminals that appeared, and when ready to begin tracking a human leader, press p. Enjoy your new playmates! Make sure to walk slowly so they can keep up with you and have some fun seeing them follow you around corners and map your environment.

8. NOTE: before step 7 it is advised you go to the RViz window, locate the "Nav2 Goal" button near the top of the screen (on the left of the + - buttons) and press it, then click a point slightly in front of the TurtleBot (if you can't see the map see 3.2). You will know which way is the front because the back will have a large hole in the data where the follower is. You will see the leader and follower move slightly forward, and the RViz SLAM map improves in quality. Step 8 is for increased quality of initialization of the localization for Nav2; skipping this step may degrade the performance of the navigation and mapping behaviour. This method can also be used to manually move the whole convoy from the desktop.

9. To end the run of our programs, return to the desktop and close ALL terminal windows. Then return to pick up the bots and place them back in their resting positions on the docking stations.

3.2 Operating Mode 2: Abnormal Operations

Occasionally, the SLAM map will not initialize properly or will start to give incorrect data if the leader bot is hit by something, someone picks up and moves the leader to a new position, or the battery gets low. If the battery is low, the lights around the power button will begin to flash red. When this happens, simply return the TurtleBots to the power station. You will be able to tell when the SLAM data is incorrect by the red lines no longer being synchronized with the black lines on the RViz display. To address this, you will need to restart the TurtleBot.

To restart the Turtlebot:

1. Hold down the power button on the TurtleBot until it chimes, the lights should turn off, and LiDAR will stop spinning. This should take 4-6 seconds.
2. Next, place the TurtleBot on the docking station, the lights should now turn back on, and the LiDAR will start spinning again. The restart is now complete, and you may return the leader bot to the position at the head of the convoy.

In addition to this, in the RViz window, occasionally the map will appear far outside of the grid system. If nothing appears after RViz has been open for a couple of minutes, zoom out until you see the map (it tends to be North of the grid when this happens).

3.3 Safety Issues

When picking up the TurtleBots to place them back on the docking stations, remember to check that all terminal windows are closed on the desktop. If not, they may react by backing up or spinning to avoid your hands, which could result in running over your fingers, so please return to your desktop first to check that all terminal windows are closed.

4.0 Technical Background

4.1 System Architecture and Data Flow

Our final product consists of a group of TurtleBot4 robots configured to autonomously navigate environments in a coordinated convoy. The goal was to enable multi-robot collaboration using real-time sensor data, object detection, and adaptive navigation planning. The system works by taking in raw sensor data from the TurtleBot attachments, such as the camera or the lidar, and parsing the data through a network of nodes to send specific instructions to the motors. The information will depend on the TurtleBot's current orientation, the obstacle activity in the map around it, the relative position of the tracking target to the TurtleBot, and the proximity of the other TurtleBots (if not the leader bot).

4.2 Software and Detection Implementation

The underlying software architecture is built on the ROS 2 framework, which enables modular communication between nodes on each TurtleBot using a publisher-subscriber model. Each robot runs its own set of ROS 2 nodes for perception, localization, and control, operating autonomously based on its own sensor data. A leader-follower paradigm governs the convoy behavior: the lead robot plans its path using the Nav2 stack, while follower bots use a simpler PID-based motor controller to track the leader's position and adjust their motion accordingly. This allows each robot to respond dynamically to local conditions and maintain coordinated movement without requiring inter-robot communication.

For detection, we are using a YOLO-trained model for the leader bot camera to detect a Hi-VIS vest on the human leader, as well as a tennis ball on the leader bot for any follower bots' cameras to detect and follow. YOLO (You-Only-Look-Once) was chosen for its real-time processing capabilities, processing the entire image in a single forward pass of the neural network. This architecture's speed, accuracy, and robustness make it ideal for our real-time object detection needs. Bounding boxes generated on camera feeds are used for path planning algorithms.

4.3 Navigation and Control Systems

For motor control, we have implemented both a native motor control algorithm and an integration with Nav2, an industry standard for robot navigation. The Nav2 implementation uses a dynamically generated map created by the LiDAR as the robot explores its environment. Using SLAM with our algorithm inputs, Nav2 enables real-time path planning and obstacle avoidance in changing environments.

Our testing revealed that the simple PID motor node performs better at closer ranges with faster, smoother turning capabilities ,ideal for close-range tracking where small target movements require quick camera adjustments. For longer distances where camera detection reaches physical limitations and bounding boxes may flicker, Nav2 proves more effective by following the last waypoint rather than requiring constant output. Nav2 also handles relocation around corners based on the target's last horizontal coordinate, allowing the TurtleBot to follow targets out of view until reacquisition.

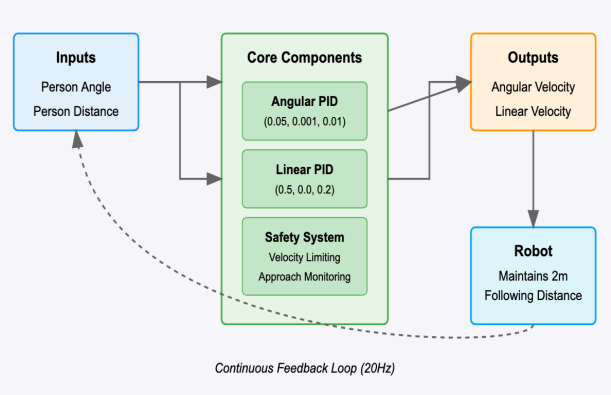

Figure 4.1: Closed-loop diagram of PID motor node

The system switches between these modes using a multiplexer that filters command velocity messages based on the active mode. By customizing Nav2's output topic, we enable smooth transitions without starting or stopping control algorithms during switching, as both continuously publish motor commands with only the appropriate ones reaching the motors.

4.4 Practical Applications

The applications of the TurtleBot convoy are vast. In a warehouse setting, a group of TurtleBots could strategically transport cargo around, following a human leader as required. In a hospital setting, the TurtleBot convoy could effectively transport medical supplies from room to room as the doctor wishes. The robust system with built-in safety features ensures safe operation by any user without compromising personal or team safety.

5.0 Relevant Engineering Standards

In developing our TurtleBot4 convoy, some of the standards we adhered to included

- IEEE 802.11
    - Defines wireless LAN communication, relying on this to connect TurtleBots to Wi-Fi
- IEEE 802.3
    - Ethernet standard, used to connect TurtleBots to a general router that allowed us to handle computation on a computer as opposed to overloading the TurtleBot's Raspberry Pi
- IETF RFC 4251
    - SSH Protocol, utilized to SSH into our TurtleBot's Raspberry Pis
- PEP 8
    - Standard of Python code formatting, we made sure to keep our Python files organized, used for our YOLO nodes, and more
- ISO/IEC 14882
    - Standard for implementations of C++, used for our teleop motor control node
- IEEE Std 1003.1
    - POSIX, shell coding, in our case, creating bash scripts to run our ROS 2 files
- ISO 8373
    - Terminology for robotics, some examples we used were "Pose-to-pose control" and "Autonomy"
- IEEE 12207
    - Framework for software life cycle processes, most notably splitting up into subgroups and initially working on sub-components of a product, and later working together to combine different components into the final product

And perhaps most relevant, some ROS 2 Standards we assimilated to a great extent during this project:

- REP 2001 (ROS 2 file layout and packages)
- REP 2011 (ROS 2 namespacing rules and remapping)
- REP 2002 (Real-time components, e.g, obstacle avoidance)
- REP 2004 (Expected functionality of ROS 2, in our case, node behavior with publisher/subscriber model)

6.0 Cost Breakdown

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	TurtleBot 4 Lite	Made by Clearpath Robotics and comes with the LiDAR and Oak-D-Lite camera that our programs use.	$1,337.56	$0
2	TurtleBot 4 Lite	Made by Clearpath Robotics and comes with the LiDAR and Oak-D-Lite camera that our programs use.	$1,337.56	$0
3	Tennis Ball	Any tennis ball, the newer the better	$1.50	$0
4	High-Vis Safety Vest	Any orange or yellow safety vest typically found on construction sites	$15	$0
			Beta Version-Total Cost	$2,691.62

Our project requires a minimum of two Clearpath Robotics TurtleBot 4 Lite Mobile Robots, with the cost detailed on the RobotShop product page [7.3.6]. This project works with more than two Turtlebot 4 Lites, and for users attempting to create a larger convoy, they should consider the unit cost of each TurtleBot. The tennis ball count will also increase by one with each additional TurtleBot to the minimum two-bot system. It should also be noted that this project requires the user to have a computer or a virtual machine running Ubuntu Linux and a Wi-Fi router that the user has direct access to.

7.0 Appendices

7.1 Appendix A - Specifications

Requirement	Value, range, tolerance, units
Target Recognition	Detected targets and convoy members have > 90% confidence ratings. Objects below this threshold will not be considered per safety and correctness considerations.
Tracking Precision	The PID controller keeps targets within focus, deviations do not exceed 10%. In the event of target loss, intelligent path planning will relocate via the next position prediction.
Obstacle Detection Range	The system detects obstacles within a range of 1.5 meters with 95% reliability
Collision Avoidance	Objects obstructing the path to the leader will be avoided through a generated plan. 300ms or more before anticipated collision.
Control Responsiveness	Respond to speed and distance changes within 0.5 seconds of detected variation.
System Integration	Communication within the ROS 2 framework achieves real-time data exchange in under 50 milliseconds
Scalability	The system supports scalability to as many convoy members as within the budget.

7.2 Appendix B – Team Information

Team Member	Biography	Email	Phone Number
Hugo Sillmberg	Graduating senior in Electrical Engineering at Boston University. Looking to learn more about systems engineering and cloud computing, he specialized in backend software engineering and cloud computing security during his time at Boston University.	hsilmber@bu.edu	(914) 310-6862
Nikhil Krishna	3rd-year computer engineering student 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	Graduating senior in Computer Engineering 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. Ben will be returning to BU as a grad student studying Robotics.	bhsu25@bu.edu	(617) 860-9997
Sadman Kabir	Graduating senior in Computer Engineering at Boston University interested in IC design and fabrication, machine learning, power electronics, and photonics chips.	kabirs@bu.edu	(718) 593 2041