Implementation Report

Assignment-I

Safe Exploration of Two Turtles in the `turtlesim` Environment

Research Track

MSc. in Robotics Engineering

University of Genoa

Submitted by:

Wolde, Mihret Kochito

Academic Year: 2025/2026

Submission Date: December 5, 2025

Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS)
University of Genoa – Italy

1 Introduction

A two turtlebot safe navigation system has been developed in this project using the ROS 2 framework and the `turtlesim` simulator. With two turtlesim robots operating inside a bounded environment, one node was designed to provide user–commanded velocity inputs, while a second node acting as a safety supervisor that prevents collisions and boundary violations.

2 System Architecture

Primarily, a User Interface Node (UiNode) is implemented through which user commands are received and applied to either robot for a specified duration. Second, a Safety Node is designed so that robot positions are continuously monitored, the inter-robot distance is computed, workspace limits are enforced, and the moving robot is stopped whenever unsafe conditions are detected. In this way, controlled user interaction and robust safety management are ensured throughout the system. To achieve a stable and robust solution, an architecture where only the Safety Node is allowed to publish the final velocity commands to the robots. All velocity commands instead pass through a safety layer that can deny or override unsafe motion.

The UiNode does not directly control the robots. Instead, it publishes "desired" velocity commands on dedicated topics:

- `/turtle1/ui_cmd_vel`
- `/turtle2/ui_cmd_vel`

The Safety Node consumes these desired commands together with real-time pose information from:

- `/turtle1/pose`
- `/turtle2/pose`

It computes the inter–robot distance and verifies whether the commanded motion is safe. If safe, the command is forwarded to the "real" velocity topics:

- `/turtle1/cmd_vel`
- `/turtle2/cmd_vel`

Otherwise, a zero velocity command is sent to stop the moving robot. Finally, the Safety Node publishes the instantaneous distance between robots on the inspection window.

3 User Interface Node

The UiNode provides a simple command-line interface that allows the user to manually control either robot. Its role is to collect user input, construct velocity commands, and publish these commands for a fixed and well-defined duration.

3.1 Functionality Overview

Upon execution, the UiNode repeatedly:

1. prompts the user to choose which robot to control (`turtle1` or `turtle2`);
2. prompts the user for a linear and angular velocity;
3. Perform a screening procedure to ensure validity of input for further processing.

4. publishes the requested velocity on the corresponding `/turtleX/ui_cmd_vel` topic for one second;
5. publishes a zero velocity to stop the robot;
6. returns to the main prompt for the next user command.

3.2 Command Duration

The node uses a simple timing loop based on `std::this_thread::sleep_for` to maintain a constant stream of velocity messages for approximately one second. The chosen publishing frequency is 5 Hz (every 200 ms), achieving enough samples for the Safety Node to interpret the intended motion. After one second, a zero velocity command is published to explicitly to halt the robot.

3.3 Published Topics

The UiNode publishes to:

- `/turtle1/ui_cmd_vel`
- `/turtle2/ui_cmd_vel`

In general, the structure of UiNode includes:

- two publishers of type `geometry_msgs::msg::Twist`;
- a blocking input loop for interactive user prompts;
- a method to apply command for 1 s and then issue a zero-velocity message.

4 Safety Node (Supervision and Arbitration)

The Safety Node continuously monitors the position and commanded motion of both robots and acts as a mediator that forwards only safe commands to the turtles. When unsafe conditions are detected, the node overrides the user's intention and stops the moving robot.

4.1 Inputs and Outputs

The Safety Node subscribes to:

- `/turtle1/pose`, `/turtle2/pose`
  (real-time robot states: x, y, and heading θ),
- `/turtle1/ui_cmd_vel`, `/turtle2/ui_cmd_vel`
  (desired velocity commands from the user).

It publishes:

- `/turtles_distance`
  (the instantaneous Euclidean separation between the robots: used for logging purpose),
- `/turtle1/cmd_vel`, `/turtle2/cmd_vel`
  (the final, safety-approved velocity commands).

4.2 Inter–Robot Distance Monitoring

At each 50 ms supervision cycle, the Safety Node computes the Euclidean distance:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2},$$

and publishes it on `/turtles_distance`. A configurable threshold (typically $d_{\text{safe}} = 1.5$) defines the minimum allowable separation. If the robots are closer than this threshold, further checks are applied to determine whether the commanded motion increases or decreases this separation distance.

4.3 Directional Safety via Heading and Relative Velocity

Instead of stopping the robots whenever $d < d_{\text{safe}}$, the node evaluates whether the *commanded motion* would attempt to move the robots towards each other or away. Given each turtle's heading θ and desired forward velocity v, the Safety Node computes their world-frame velocities:

$$v_x = v\cos\theta, \qquad v_y = v\sin\theta.$$

The relative motion is then characterised by:

$$\Delta v = v_1 - v_2, \qquad r = \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \end{bmatrix},$$

and in particular the dot product:

$$r^\top \Delta v.$$

This quantity is negative when the robots are *attempting to move closer* and positive when they are *attempting to move apart*. Thus:

$$r^\top \Delta v < 0 \Rightarrow \text{unsafe}, \qquad r^\top \Delta v > 0 \Rightarrow \text{safe}.$$

This refinement prevents deadlock, where by the robots may be close, but if the user commands motion that would increase separation, the Safety Node would allow the motion rather than freezing both robots.

4.4 Boundary Safety

Workspace boundaries at $x, y \in [1, 10]$ are enforced similarly. For each robot the Safety Node checks:

$$\text{out\_of\_bounds}(i) = (x_i - wall_min < 0.1) \vee (x_i - wall_max < 0.1) \vee (y_i - wall_min < 0.1 \vee (y_i - wall_max < 0.1$$

As with distance safety, the command is evaluated directionally. A motion that pushes the robot further outside is blocked, whereas a motion that guides the robot back inside the safe region is permitted, allowing natural recovery from boundary violations.

4.5 Command Limits

The Safety Node executes its supervision logic at 20 Hz (50 ms period), fast enough to react to boundary violations and changes in relative distance. Commanded velocity limits have to be keep intact to ensure that a robot cannot "jump past" the safety boundary between two supervisory checks. Thus these restriction has to be obeyed such that:

$$v_{\max} \leq \alpha\, d_{\text{margin}}\, f_{\text{safety}},$$

where d_{margin} is the gap between the safety boundary and the physical wall, f_{safety} is the supervision frequency, and $\alpha \in (0, 1)$ is a design factor. With a typical value of $d_{\text{margin}} \approx 1.2$ and $f_{\text{safety}} = 20$ Hz, a practical choice is $v_{\max} \approx 2.0$.

4.6 Event-Based Logging

Continuous warnings at 20 Hz are undesirable. Thus, the Safety Node employs *state-change logging*: warnings are printed only when the system transitions into an unsafe state, where a robot first begining to push outward at a boundary is one exemplary event triggering warning log.

4.7 Implementation Summary

The Safety Node consists of:

- four subscriptions (`/pose` and `/ui_cmd_vel` for each robot),
- three publishers (`/cmd_vel` for each turtle and `/turtles_distance`),
- a 20 Hz `rclcpp::Timer` driving the supervision loop,
- logic modules for:
  - distance computation,
  - directional safety,
  - moving-robot identification,
  - boundary checking,
  - state-change logging.

5 Expected System Behavior

This section explains what the user should observe when interacting with the system under different situations.

5.1 Normal Operation Inside the Workspace

When:

- Both turtles are well inside the workspace boundaries, and
- The inter-turtle distance is larger than the collision threshold,

then:

- The safety node behaves as an almost transparent layer: it forwards user commands to the turtles unchanged;
- No warning or error messages appear in the safety node logs;
- The turtles follow the commanded velocities in `turtlesim`.

5.2 Approaching and Touching the Boundary

When the user drives a turtle towards a wall:

- The turtle moves freely until its position is within `border_threshold` of the corresponding boundary.
- If the command continues to push the turtle outward (e.g., towards smaller x while near the left wall), the safety node:

  - Replaces the command with a safe alternative (typically zero velocity);
  - Logs a message indicating that the command was blocked due to a boundary violation.
- The turtle visually appears to "stick" to the wall despite the user attempting to push it further out.

If the user then reverses direction (commanding a velocity that moves back toward the interior), the safety node:

- Recognizes this as a *pushing outward* motion;
- Allows the command to pass through;
- Optionally logs a message indicating that the command is considered safe and helps recover from the boundary.

5.3 Collision Avoidance Between Turtles

If the user attempts to drive the turtles into each other:

- As long as the inter-turtle distance d is greater than the threshold $d_{\min}$, commands are forwarded normally.
- When $d \leq d_{\min}$ and the commanded velocities would *decrease* d further, the safety node:
  - Blocks the command;
  - Logs a message stating that a collision risk has been detected.
- If the user commands velocities that increase the distance, the safety node allows them and the turtles separate again.

5.4 Freeze State and Recovery

When any critical constraint is violated (or would be violated by the commanded motion):

- The safety node enters the freeze state for the affected turtle;
- Velocity commands for that turtle are overridden, usually with zero velocities;
- A log entry is printed once at the moment of entering the freeze state.

As soon as:

- The turtles move back inside the safe region, and
- The inter-turtle distance is again above the collision threshold,

the safety node:

- Returns to the `SAFE` state;
- Allows new user commands to pass through;
- Logs a recovery message, indicating normal operation has resumed.

6 Experimental Results and Analysis

The system was tested in `turtlesim` to evaluate teleoperation, distance supervision, boundary protection, and directional safety. The perceived behaviors are noted as follows.

Teleoperation Performance

- UiNode applied user commands for 1 s (linear, angular, curved motion).
- Robots consistently stopped after 1 s, ensuring deterministic behaviour.

Distance Monitoring

- Safety Node published inter–robot distance at 20 Hz.
- Reported values matched expected Euclidean distance in the simulator.

Collision Avoidance

- When $d < d_{\text{safe}}$, the Safety Node evaluated motion direction.
- Closing motions were immediately stopped; separating motions were allowed.
- Prevented deadlock by enabling safe escape when robots were already close.

Boundary Protection

- Near $x, y = 1$ or 10, outward-pushing commands were blocked.
- Inward or turning commands were allowed, enabling smooth recovery.
- Large overshoot only occurred with unrealistically high velocities; saturation removed this issue.

Directional Safety Validation

- Safety Node correctly classified escape commands (positive $r^\top \Delta v$) and allowed them.
- Unsafe closing commands (negative $r^\top \Delta v$) were suppressed.

Event-Based Logging

- Warnings were emitted once per unsafe event (state-change logging).
- Eliminated log flooding and produced clean diagnostic messages.

7 Conclusion

This project implemented two-robot safe-navigation control in ROS 2 architecture comprising of two nodes. The UiNode provides bounded teleoperation commands, while the Safety Node supervises motion at 20 Hz, enforcing inter–robot distance, workspace boundaries, and directional safety using relative velocity and heading. Experimental results show that unsafe motions are reliably blocked while escape manoeuvres are permitted, avoiding deadlock and enabling smooth recovery.