Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Assignment 1

Mattia Cozza (mattia.cozza@studenti.unipd.it)
Lorenzo Tomai (lorenzo.tomai@studenti.unipd.it)
Nicola Berti (nicola.berti.6@studenti.unipd.it)

November 30, 2025

1 System Architecture

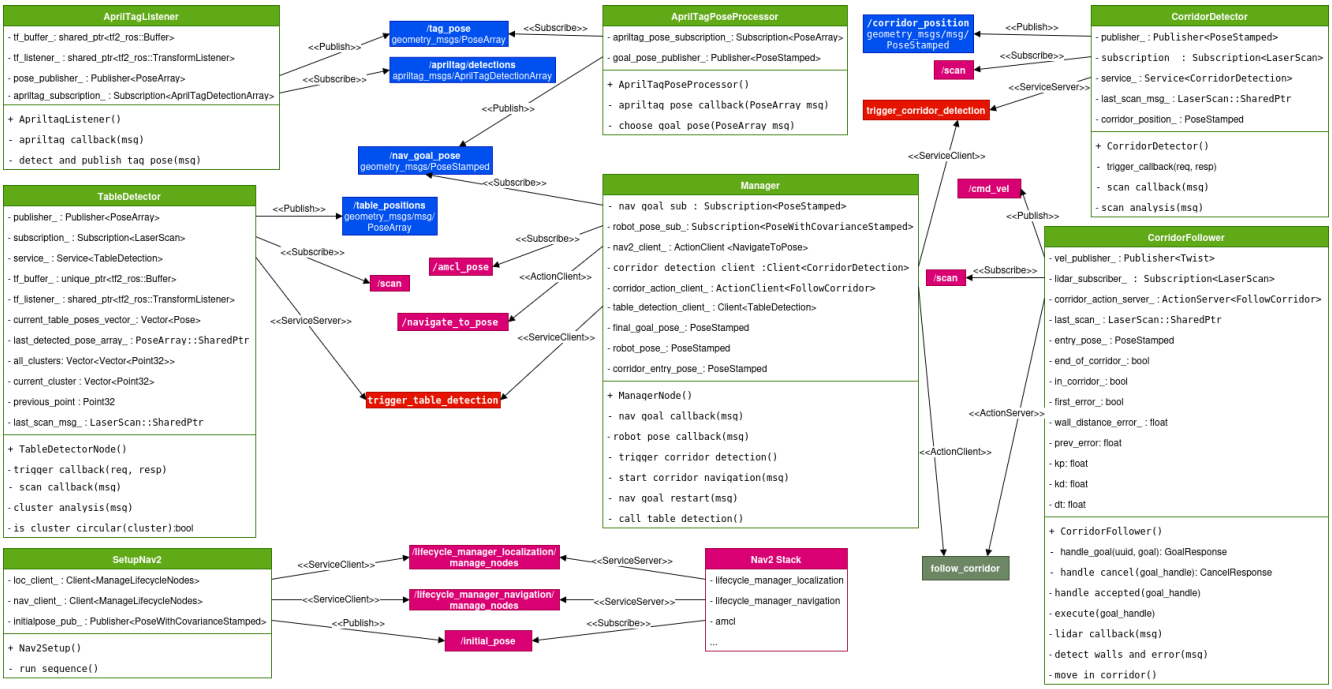

Figure 1: UML diagram

The system has been structured around modular ROS2 nodes designed for apriltag detections, autonomous navigation, corridor traversal, and table detection.

2 Nodes

2.1 Manager Node (`manager_node`)

The Manager node orchestrates all high-level behaviors and ensures seamless interaction between navigation, corridor traversal, and table detection. Its responsibilities include:

- Subscribing to `/nav_goal_pose` and forwarding goals to Nav2 via the `NavigateToPose` action.
- Monitoring the robot's current pose from `/amcl_pose` to determine proximity to corridor entry points.

- Executing corridor traversal using the `FollowCorridor` action. If a navigation goal is in progress when entering a corridor, it cancels the current Nav2 goal, performs corridor navigation, and then automatically resumes the previous goal.
- Triggering corridor detection via the `CorridorDetection` service before navigation. If the service is unavailable or fails, the system proceeds without blocking.
- Triggering table detection using the `TableDetection` service once a goal is successfully reached (either after regular navigation or post-corridor).
- Managing internal states (`goal_in_progress_`, `goal_reached`, `corridor_active_`, `canceling_for_corridor_`, `tables_detected_`) to ensure deterministic and safe operation, avoid conflicts, and provide detailed feedback.

2.2 Setup Node (`nav2_setup_node`)

The `nav2_setup_node` prepares and activates the Nav2 stack. It initializes localization, publication of the initial pose on `/initialpose` and navigation through lifecycle managers, and ensures Nav2 servers are fully active before task execution.

2.3 AprilTag Nodes (`apriltag_processor_node` and `apriltag_listener_node`)

The AprilTag system is implemented through two ROS2 nodes. The `ApriltagListener` subscribes to the `/apriltag/detections` topic, obtains the transforms of each detected tag using TF2, and publishes their poses, directly as a `PoseArray`, on the topic `/tag_pose`. The second node, `AprilTagPoseProcessor`, subscribes to it and computes a navigation goal pose based on the two detected tags, which it then publishes as a `PoseStamped` on `/nav_goal_pose`.

To setup the system for this project, the AprilTag configuration file in the *cfg* folder was modified to match the specific tags used. Moreover the `apriltag` launcher was first executed to identify the two tag IDs (1, 10) thanks to the `/apriltag/detections` topic, and the corresponding reference frames were verified.

A key aspect of the implementation was the selection of the final navigation position. Rather than placing the goal exactly at the midpoint between the two tags, an offset was applied to maintain a safe distance from them, reducing the risk of collisions with larger markers. This approach also ensures an almost central position within the room, providing an optimal viewpoint for observing all tables in the environment, as one can see from the simulation.

2.4 Corridor Nodes (`corridor_detector_node and corridor_follower_node`)

The `corridor_detector_node` is responsible for identifying the entrance of the corridor in order to initiate the "manual" navigation phase. Its detection algorithm combines corner identification with the detection of sudden distance jumps between consecutive `LaserScan` points. Instead of locating the exact boundary of the corridor, the algorithm intentionally selects a point slightly inside it to ensure that the robot fully enters the corridor before switching modes. This approach was chosen for its simplicity and computational efficiency, and it performs reliably as long as the inner corridor walls remain visible to the robot. However, it may fail in situations where the walls are outside the sensor's field of view or when multiple corridors create ambiguous geometric structures.

The `corridor_follower_node` makes use of an action server (`FollowCorridor`) that triggers the corridor-following behaviour upon receiving a goal. Using an action rather than a simple service allows the manager node to remain fully operational while the robot navigates the corridor, enabling it to continue executing other tasks in parallel. In more general scenarios, this design choice also provides the flexibility to interrupt the manual navigation if necessary, or to exploit action feedback for higher-level decision making or coordination with other nodes. In the current implementation, no complex feedback is required for the manager; so it just provides a feedback of the wall-distance error (other interesting feedback could be the distance from the closest wall/obstacle or the current position.

This error represents the lateral deviation of the robot from the corridor center and is used internally by the node to regulate its heading through a PD controller. A constant forward velocity ensures steady progression along the corridor, while the PD controller continuously corrects angular velocity to compensate for deviations. To detect the end of the corridor, the node monitors the variations in the wall-distance error. A big change indicates either

that the controller can no longer maintain a stable trajectory or that the corridor geometry has opened up. In both situations, the manual navigation phase is terminated and control is returned to the autonomous navigation stack with the goal position already derived. The PD gains are intentionally kept low to prevent excessive steering corrections in response to small deviations, as the task only requires straightforward corridor tracking along a wide and safe path (moreover the tests show the Lidar can provide noisy measurements).

2.5 TableDetector Node (`table_detector_node`)

The `table_detector_node` is responsible for detecting the three tables whenever it is triggered by the `manager_node`. Table detection is performed by applying a simple clustering algorithm to the incoming `LaserScan` data and evaluating whether each cluster is approximately circular, based on a variance threshold computed from the distances of all points to the cluster's centroid. This approach was chosen for its simplicity, efficiency, and suitability for the requirements of this assignment. Although it works well in most cases, occasional false positives may occur due to the limited resolution of the `LaserScan`, some tables may be represented by only a few scan points, and similarly, small sections of walls consisting of three close points may sometimes be mistakenly classified as circular shapes.

3 Interfaces

Custom Interfaces:

- `CorridorDetection.srv` provides corridor pose estimation.
- `FollowCorridor.action` manages corridor-following behaviour.
- `TableDetection.srv` triggers table perception routines.

Git Repository

`https://github.com/mattreturn1/ws_17_assignments.git`

Video

Videos of main pipeline and extra part: `https://drive.google.com/drive/folders/1vSADWVj_L5AOF5eMWP_o7lhuIRQXHkwO?usp=sharing`

Acknowledgments

This report includes content assisted by the AI system **ChatGPT**. AI was used for support in drafting the LaTeX structure and template. All technical descriptions and project content were authored and verified by the team.

References

[1] Open Robotics, *ROS 2 Documentation*, 2025. `https://docs.ros.org/en/jazzy/`.

[2] Open Robotics, *Navigation2 (Nav2) Documentation*, 2025. `https://api.nav2.org/`.

[3] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. MIT Press, 2011.