Høgskulen på Vestlandet

SEMESTERPROJECT

DAT160

Group 6

Henrik Hjellup Horne
Frikk Myhre Slåsletten

GitHub:
https://github.com/678147/Dat160_Semesterprosjekt/tree/main

18.11.2025

Table of Contents

1 Introduction

1.1 Task

The purpose of this project was to design and implement a Search and Rescue system using two TurtleBot3 mobile robots in the Robot Operation System 2 (ROS2) framework. The project builds upon those topics we've covered in the course, including mobile robot control, navigation, robot teams, and robot software architectures.

In the scenario of the task, the robots must navigate through an unknown environment, detect injured persons and fire sources, and communicate this information. Each robot is equipped with a forward-looking camera, LIDAR, IMU and wheel encoders. To detect inured persons or fires the robot had to detect ArUco markers spread across the map. We did not have any prior knowledge of what world the robots would perform their tasks in, except their starting positions in the possible worlds, which made it hard to preplan routes for the robots. This forced us to use more dynamic solutions.

The robots communicate over a network using ROS2 topics, services, or actions. They had a total of 10 minutes to complete their mission. There were five different variations of worlds, each containing unique layouts, ArUco marker placements and obstacles. For example, some maps include additional openings in walls or different target locations. During the competition, one map was chosen at random. Therefore, pre-programming specific paths with specific coordinates was not practical, and the robots had to dynamically explore, adapt and cooperate to complete the mission.

1.2 First approach

Our first approach was to implement a multi-robot navigation system that used bug2 for navigating the map and a coordinator node for giving goals to the robots. Originally, we intended to have at least three robots. Where two of them explored and found the points of interest, while the other robot would have been in reserve and react to the tasks that needed two robots.

The idea was that the exploring robots could handle the one-robot tasks fine on their own, while if it found a two-robot tasks it would post the coordinates for the reserve robot and wait for it to arrive to solve the two-robot problem. The intent behind this was to reduce the exploration delay that would be entailed with one robot having to stop the exploration to help solve a two-robot problem. The exploring robots were meant to have responsibility over half of the map each and use a weighted frontier exploration algorithm for the navigation goals. The weighting of these points was supposed to consider the distance from point of origin, how much of the map it reveals, and expected travel time/distance.

Implementing this however was not possible with the limited manhours we had for the project. We discovered this during development, after two of our project partners dropped the course early in October. This made us try to pivot to the simpler system of a frontier exploration algorithm for giving out navigational goals in our final approach.

1.3 Final approach

Our final approach is marred by our late realization of the actual workload needed to implement our fist idea for the program. It implements the same bug2 navigation algorithm that we implemented for our first draft and contains a simpler coordinator node were the targets for the robots intended to be given by a frontier exploration algorithm.

The goal-assignment for the robots was intended to use frontier exploration algorithm, although we had a great deal of problems with implementing this because of the already known map producing no

unknown frontiers. This ended up in that we implemented a simple random exploration algorithm that used the occupancyGrid form the /map topic to set the furthest extent of possible targets and removes previous targets as available to avoid getting the same ones again. The targets are delivered to the idle robots which activates the bug2 navigation to the given target. When the target is reached, the robot is set to idle and receives a new target. A problem with this approach is the possibility that the target gets assigned outside of the map. Targets can also end up being assigned close to the previous target ending up in little exploration of the map. Additionally, the robot can end up moving through mostly explored parts of the map on its way to a target on the other side of the map wasting a lot of time doing so.

The final approach for the project is simple and given enough time would explore and find all POI on the maps. However, to achieve this would be quite time-consuming and inefficient. The major reason for us ending up with this as a solution is our optimistic outlook on the time needed to implement our initial idea for the project and the reduction in our group membership during the semester.

2 Design process

2.1 Work division

When we started the project, the initial design and division of work were made with the group size in mind. Our strategy therefore changed from distributing the work into smaller tasks for four people, to the two of us remaining dividing the work that had to be done into bigger main categories:

- Robot navigation
- Robot coordination
- Robot Vision w/ArUco
- System setup and integration

Each of us took lead responsibility for some of these components, but we both contributed to testing, debugging and integration.

During the first weeks of implementation, we collaborated using Visual Studio Live Share for real-time coding sessions. As the system grew in complexity and nodes became more independent, we transitioned to using GitHub for better control and easier testing of the system.

2.2 Problems encountered and solutions

During the project we encountered several practical and technical problems:

- **Reduced group size**
  - *Problem*: Group members dropping the course.
  - *Solution*: We started motivated, ambitious and tried to make the initial plan work. Later, we realised that we had to simplify the architecture, reducing the amount of turtlebots and changing the coordination algorithm. The system still demonstrates multi-robot cooperation.
- **Integration between vision and coordination**
  - *Problem:* While the individual nodes worked well in tests, integrating them into one coherent system proved challenging. Small configuration issues in TF frames, topic names and timing caused the robots to do things that were hard to diagnose, especially close to deadline.

- *Solution:* There are still problems with this. Nevertheless, since we only were two people diagnosing, we found solutions to almost everything.

- **Interaction between navigation and wall following**
  - *Problem:* We tried mixing obstacle avoidance directly into the GoToPoint logic. This made the behaviour unstable, hard to tune and hard to debug.
  - *Solution:* We separated it, making GoToPoint responsible for only driving towards a goal, while the WallFollower is responsible for avoiding obstacles. Creating a controller to switch between these nodes based on LIDAR measurements and using states.
- **Uncertainty about map layout**
  - *Problem:* The competition map was chosen randomly from five variants, which made hard-coded paths or hand-crafted strategies insufficient.
  - *Solution:* Used a more general approach which was not map dependent, avoiding hard coding movement by implementing a bug2 algorithm for navigation around the map. And a random exploration algorithm for picking bug2 targets.

3 Final implementation

3.1 System overview

The final system consists of a set ROS2 nodes running on each TurtleBot3 and several shared nodes handling coordination and vision. Figure 3.1-1 illustrates the overall architecture.

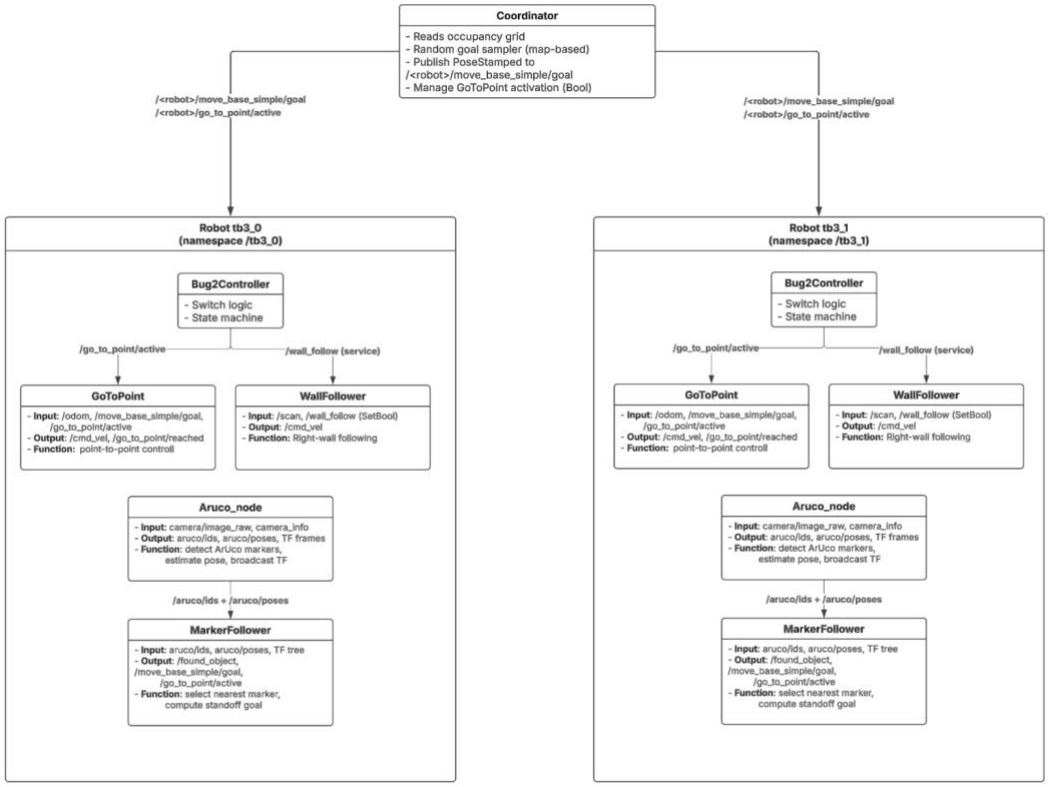


Figure 3.1-1: ROS2 System Architecture Diagram

The turtlebots are operating in parallel, each running a full navigation and vision stack inside its own ROS2 namespace (/tb3_0 and /tb31). A separate global Coordinator node handles goal assignment and overall exploration logic. Each robot contains the following main components:

Each turtlebot runs the **Bug2Controller** for decision-making in the bug2 navigation algorithm. It receives goal updates and laser scan data, and switches between the two local behaviours (GoToPoint and WallFollower). It does not compute velocities itself but determines which behaviour is currently active. This separation gives clearer logic and reusable components. A video of the launch of the bug2 can be seen here in this video.

The **GoToPoint** subscribes to the robots odometry, a goal pose and an activation flag. When enabled, it computes linear and angular velocity commands that drive the robot toward the goals assigned from the coordinator (PoseStamped). The turtlebots turns then drives and publishes a flag when the goal has been reached. It operates independently of obstacle avoidance.

For obstacle avoidance we're using a reactive node **WallFollower** that processes laser scan data and applies a simple rule to follow the right-hand wall. It is enabled and disabled via a SetBool service. When active, WallFollower continuously outputs velocity commands that keep the robot aligned with nearby walls and safely steer around obstacles.

We unfortunately missed that the **Aruco_node** was already given to us. This made us use unnecessary time on code that was already finished, even though it gave us a deep understanding of OpenCV's ArUco library. The theory for this was found on OpenCV's Detection of ArUco Markers site [1]. This node detects markers from the onboard camera, estimate their 3D pose and broadcasts individual TF frames (aruco_<id>). It publishes both marker IDs and pose arrays, making it able to detect the ArUcos as showed in the ArUco-video.

The idea for the **MarkerFollower** is that it should subscribe to the ArUco detections and use TF to compute each markers position relative to the map frame. The node should select the nearest detected marker and compute a standoff goal in front of it (0,5m away). This goal is then supposed to be sent to the navigation stack. Integrating this with the coordinator is our main issue.

The brain of the system is the **Coordinator**. It is the node responsible for exploration control. The node publishes a PoseStamped to /<robot>/move_base_simple/goal and activates local navigation via <robot>/go_to_point/activate.

When a robot reports that it has reached its goal through <robot>/go_to_point/reached, the coordinator publishes <robot>/go_to_point/activate as false. Now the robot is seen as inactive by the coordinator, so it publishes a new goal through /<robot>/move_base_simple/goal.

3.2 Navigation Stack (Bug2, GoToPoint, WallFollower)

The robot's navigation system is built around the Bug2 algorithm, implemented as a state machine inside the Bug2Controller. The state machine is shown in figure 3.2-1. The controller transitions between three states:

- **Idle**: no navigation goal assigned, both behaviours disabled.
- **GoToPoint**: point-to-point movement toward the assigned target.
- **WallFollowing**: reactive obstacle avoidance when a direct path is blocked.

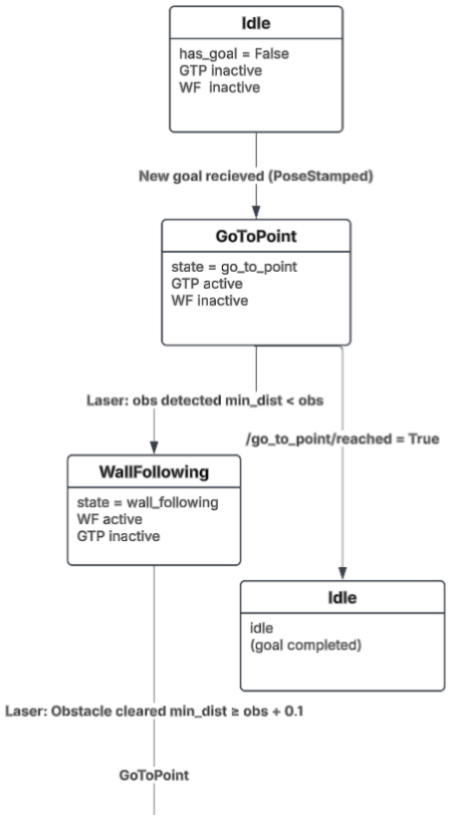


Figure 3.2-1: Bug2 Active State Diagram

3.2.1 State transitions

1. **Idle → GoToPoint**

The turtlebot gets triggered when a PoseStamped goal is received on /<robot>/move_base_simple/goal. The controller activates GoToPoint node by publishing True to /<robot>go_to_point/active

2. **GoToPoint → WallFollowing**

When an obstacle is detected by the robot's onboard camera via /scan, the controller inspects this raw data and switches states between min_front_distance < obstacle dist. GoToPoint is deactivated, WallFollower is enabled via the /wall_follow service.

3. **WallFollowing → GoToPoint**

When an obstacle is cleared and min_front_distance ≥ obstacle_dist + 0,1 control returns to GoToPoint.

4. **GoToPoint → idle**

When the robot is within a threshold distance dist_eps of the goal, GoToPoint publishes /go_to_point/reached = True, and the controller returns to idle.

4 Experiments

We went through three distinct iterations of ideas with some overlapping between each other.

1. Bug2 navigating and weighted frontier exploration
2. Constantly wall following robots
3. Bug2 navigation with random target allocation

For details about the first iteration, see **1.2**.

The second iteration used two wallfollowing robots going in separate directions, one following the left wall and one following the right wall while picking up tasks along the way. This worked well in scenarios where no walls where freestanding in the middle of the map, but when this was the case it usually never explored these walls, only the outer walls, or got stuck on them and never explored the outer walls. This often-left huge swaths of the map unexplored and the robots just doing loops around whatever wall they first encountered. Because of this glaring problem we soon realized that this implementation would not work for the project, so we changed to something more like our first implementation and worked on that instead.

For details about the third iteration, see **1.3** and **5.2**.

5 Conclusion

5.1 Experiences

Although the original project plan was designed for a four-person team, we ultimately completed the work as a team of two. While this inevitably reduced the amount of functionality we were able to implement, it also became one of the most valuable aspects of the project. The pressure of being two on a project like this, made us both having to take responsibility for every part of the system, navigation, vision, coordination and integration. This forced us to understand not only “our own” component, but how all components interacted. In retrospect, the reduced group size significantly accelerated our learning, particularly in areas such as debugging and reasoning about system behaviour under real-world constraints.

At the beginning of the project, most of our development happened through Visual Studio Codes Live Share extension. This allowed us to write and test code collaboratively in real time and was extremely useful during the initial setup phase. However, as the system grew, we quickly realised the limitations of this approach. Changes became harder to track, synchronisation issues appeared frequently, and debugging required individual experimentation. This experience made us appreciate version control far more than before. Switching to GitHub for structured commits, branches, and pull requests provided a cleaner workflow, and made it possible to test individual features without breaking the shared code base. Learning to use Git effectively became a key skill that we expect will benefit us in future projects.

Another major learning outcome was developing systematic approaches to debugging in ROS2. Because our system consisted of many nodes running at the same time, traditional print-based debugging was rarely sufficient. Instead, we relied on tools such as *rqt_graph, ros2 topic echo, ros2 topic hz*, and *ros2 node info* to inspect live data and isolate components.

Although the final system is not perfect, we are proud of what we accomplished as a two-man team. The experience has given us valuable lessons in planning, system architecture, communication and in ROS2.

5.2 Improvements

Improving our system would entail developing something more like our initial idea. Important changes could include implementing handling for 2 robot tasks, the addition of a third robot for two robot tasks, separation of responsibility over half of the map for exploring robots, implementing a functioning weighted frontier exploration algorithm and polish the bug2 algorithm for improved navigation.

The implementation of handling the 2 robot tasks to improve our system would entail the adding of a third robot. This robot's sole responsibility would be to navigate to POIs that need two robots to complete the tasks. For the assignment of a targets for the reserve robot we planned to pause exploration for the robot that encountered a 2 robot POI and for it then to send out a poseStamped message with the current robot coordinates and wait for the reserve robot to arrive at the coordinates and sign off on that the task is completed before the exploring robot resumes exploration again. This would have avoided the long pause in solving the 2 robot POIs if we were to run only two robots, as one robot would still actively explore while the task was completed. A possible more efficient alternative to this would be to have 2 robots in reserve to accomplish the 2 robot tasks instead, so that the exploration robot just sends out the coordinates of the task and then continues exploring.

Splitting the map into two separate parts where each exploring robot has responsibility for each part was intended to avoid overlap in exploration to abstain from wasting time. The planed implementation of this was splitting the occupancyGrid used for the weighted frontier exploration along the y-axis, ensuring no overlap in exploration and a faster discovery of the map. A potential problem with this approach is that if there were a room in the map where the entrance was on one robots side, while the interior was on the other robots side it could be time consuming for the robot to navigate to the interior of the wall, and back over to a new target point with little gain in exploration.

A weighted frontier explorer was the planed target assignment algorithm for our project. The idea was that the coordinator node would compare unexplored frontiers based on factors like weight of movement (a time to distance ratio?), exploration potential (how many frontiers are exposed) and expected coverage. The intention with a system like this was that it could work more efficiently and explore a larger proportion of the map faster than a regular frontier exploration algorithm.

Although we tried to implement a system like this, the limited time and manhours available to us made it not achievable for us to implement an algorithm like this. Though with further work into the proposed system above we would undoubtedly meet more problems and changes needing rectification and improvements. But we believe it is a feasible solution to the task.

6 Resources and references

We used resources from OpenCV to figure out the ArUco detection:

[1] OpenCV Team, ArUco Marker Detection, OpenCV Documentation, 2024. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. [Accessed: Oct. 15, 2025].

7 Figure list

Figure 3.2-1: Bug2Controller Active State Diagram

8 Link to videos

8.1 Link S&R

Link S&R: https://youtu.be/85LA-C9J6g8

8.2 Link bug2

Link bug2: https://youtu.be/tjVjF-X3vPI

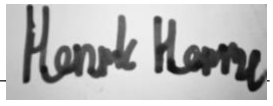
8.3 Link ArUco

Link aruco: https://youtu.be/uZioE-lxINM

9 Appendices

9.1 Work log

Week	Member	Task description	Category
33	Both	Setup VM and ROS2	Setup
38	Both	Discord meeting regarding planning and future work division.	Setup
39	Both	Discord meeting planning first iteration.	Setup
40	Both	Started iteration 1: Bug2 + weighted frontier concept.	Theory
40	Henrik	Iteration 1: Started the weighted frontier.	Code
40	Frikk	Iteration 1: Started bug2 algorithm.	Code
41	Henrik	Iteration 1: Implemented GoToPoint node.	Code
41	Frikk	Iteration 1: Implemented WallFollower, started on bug2controller.	Code
41	Both	PDR-Presentation + PowerPoint.	Presentation
42	Both	Created semester project report document.	Report
42	Both	Iteration 1: Attempted weighted frontier exploration.	Code
43	Both	Reorganizing to handle workload, discarded iteration 1.	Setup
43	Both	Iteration 2: Full wall-following exploration (Left/Right).	Code
43	Both	Iteration 1: Discarding iteration 2. Going back to iteration 1.	Setup
43	Both	Iteration 1: Continuing implementation of weighted frontier.	Code
43	Both	Iteration 1: Polishing bug2 algorithm.	Code
44	Both	Iteration 1: Discarding iteration 1. Workload.	Setup
44	Both	Iteration 3: Starting iteration 3.	Theory
44	Both	Iteration 3: Bug2 + random goal allocation.	Code
44	Henrik	Iteration 3: Implemented map-based random goal sampler.	Code
44	Frikk	Iteration 3: ArUco detection.	Code
45	Henrik	Iteration 3: Testing coordinator node.	Code
45	Frikk	Iteration 3: Testing navigation stack.	Code
45	Both	Iteration 3: Polishing and integration.	Code
46	Both	Iteration 3: Integration and debugging.	Testing
46	Both	Presentation and competition.	Comp.
46	Both	Writing report.	Report
47	Both	Writing report.	Report

Henrik Hjellup Horne:__  Frikk Myhre Slåsletten:__