METR4202 Report

Project 1

Overall system

The overall system is a python script designed to be used by a Turtlebot3 to perform SLAM using LIDAR in an environment with a known size, no moving obstacles and unknown starting point, with a focus on frontier detection and iterating through all these frontiers repeatedly to explore all parts of the environment.

System Workflow – The See-Think-Act cycle

1. Initialisation
    a. To initialise the script, the main function is run. This main function firsts creates an ExplorationNode node, then uses a MultiThreadedExecutor executor to be able to run this node while also spinning the node. This allows for processing multiple messages and events in parallel.
2. Initialization of subscriptions and Publishers – "SEE"
   There are 1 publisher being made:
    1. PoseStamped – this was used to give the robot a goal pose to reach once it has determined an appropriate frontier to target exploring.

   There are 2 subscriptions being made:
    1. OccupancyGrid – this was used to create a global costmap that was utilised to determine frontiers that the robot could then consider going to, then this was done until the map was fully explored.
    2. BehaviorTreeLog – using a node named NavigateRecovery, this subscription was used to determine whether the robot was in an idle state before publishing the target goal that it then must achieve.
3. Analysis of map data to determine frontiers - "THINK"
    a. The OccupancyGrid costmap msg was used to retrieve the size of the map and its origin, then through a callback function, the costmap coordinates were updated. The costmap was then converted into a normalised grid with the size of the map.
    b. To determine which coordinates on the map at its current stage could be used as frontiers for the robot to consider exploring, a function was made to iterate through the contents of the normalised grid representing the costmap. This was used to check if a cell that is considered "free space" is surrounded by "unknown space". If so, a frontier point would be added to that cell. Finally, returning a 2D array frontier map. The frontier map is then passed through another function to help the robot decide which frontier to prioritise exploring.
4. Choosing the most appropriate frontier to set as a goal
    a. It was found that when there is no wall for LIDAR rays to bounce off, the robot records everything past a short distance as unknown, thus marking what might not be frontiers as frontiers. This allows for inefficient exploration as the robot will move the short distance, which likely has no walls or obstacles, iterate over its records of the map, process the data and move again. This was circumvented by applying a radius of 1.5m around the robot, and only the determined frontiers past this radius were considered for the algorithm described in 4b.
    b. A function was constructed which iterates over the 2D frontier map using a 3x3 kernel, calculating the sum of frontiers within the kernel at each position in the frontier map. The coordinates with the highest concentration of frontiers were then returned and these were sorted according to their distance to the current position of the robot, with the coordinate with the smallest distance being prioritised by the robot to create a waypoint. The logic behind this implementation is to ensure that the robot

covers as much unknown ground as efficiently as possible, meaning that areas on the current iteration of the map that are effectively “most unknown” and easiest to access are explored first.

5. Detecting Aruco Tags
    a. The camera on the turtlebot is constantly on the lookout for Aruco tags, where it only scans for 6x6 100 mm tags and will identify the IDs of those tags should they be detected and print them to the terminal. Additionally, the algorithm also calculates the position of the tag relative to the robot by using the robot’s camera calibration information. It will print the position of the tag relative to the robot should a tag be detected while it is exploring the map.
6. Moving the robot to the target frontier - “ACT”
    a. To move the robot to the prioritised area of frontier cells, these cells were converted from normalised grid cells to pixel/meter units with a resolution of 0.05. Once done, this was then transformed to the frame of the map using its size and origin, then a PoseStamped publisher was used to publish a pose for the robot to achieve.
    b. Nav2 libraries were referenced and used to implement obstacle avoidance logic into the robot.
7. Stopping the process once map has been fully explored
    a. To determine if a map has been fully explored, the process of removing frontiers from the list of frontiers generated as these frontiers were explored was implemented, and then used to check if the map is completed by checking that the final iteration of the list is empty.

System Behaviour

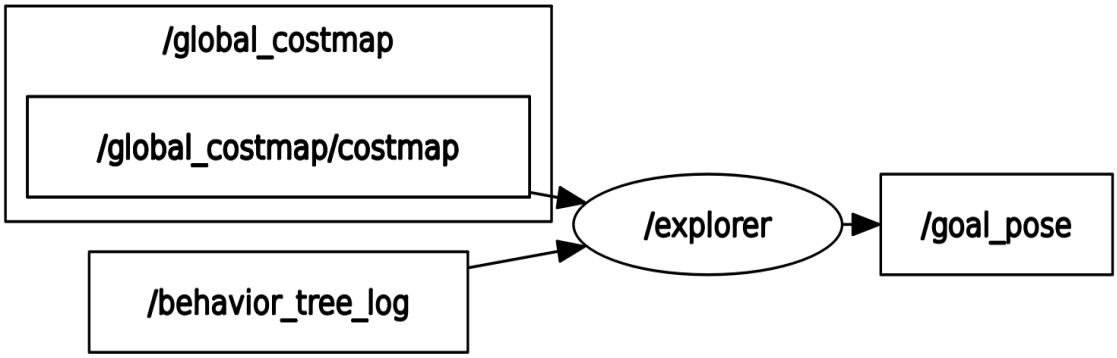

Fig. 1: System Diagram

The explorer node subscribes to /global_costmap/costmap topic to find the safe frontiers within the map. The explorer node then creates a waypoint to a point of the high concentration of frontiers and publishes that waypoint to the /goal_pose topic to send the robot to the frontier coordinate. Explorer also subscribes to the /behaviour_tree_log topic for waypoint handling. The waypoint handling involves waiting for the robot to be in a recovery/idle state before finding and moving to a new frontier. This process is repeated until the costmap contains no more frontiers.