Kalyani Bharat Forge | Cobots Swarm

Final Report

Inter-IIT Tech Meet 13.0, IIT Bombay

Team_83

Contents

Chapter-1 | Introduction to the Problem Statement

1.1 Motivation & Overview

With the advent of the Industry 4.0, there is an increased requirement in the firms looking for a more robotic based approach towards solving their problems. One of the most common solution is a swarm of cobots. Cobots are defined as robots that co-work with humans in a given workspace. Through the given problem statement, our motive was to develop a centralised brain (`manager` of the swarm of robots).
The primary focus of ours was to make the system scalable and in line with that. we worked towards designing *all* our solutions feasible for an n number of robots instead of say, 4 robots. Since, GPS location may or may not be found accurately within the boundaries of a factory, there is a need for localised mapping approach.

1.2 System Architecture

Following is the description of the system architecture and an overview of the approach that we have adopted towards solving this problem statement - (reference image) A.1.

1.2.1 Hardware Overview

To ensure the feasibility of the swarm in real life, we have designed the system to be as less computationally expensive as possible. Each robot is equipped with a bare minimum of the following sensors, additionally a few optional sensors maybe added:

- **Inertial Measurement Unit (IMU)**: An IMU helps to estimate the orientation of the robot and is necessary to give feedback and maintain balance in the robots.
- **Light Detection and Ranging (2D-LiDAR)**: We utilise a 2D-LiDAR for mapping the workspace and aid in localisation while the cobots are performing their tasks.
- **Camera**: A simple 2D-camera will help to identify the obstacles and the objects whose details need to be stored in the database (Usage of a depth camera is optional). We implement a basic computer vision model (to know more about it).

1.2.2 Software Architecture

Our robotic-based solution is based on ROS2 (Humble Hawksbill), being run on an Ubuntu-22.04.5 system, and the simulator environment of Gazebo (version-11) is being used. More details to this can be found in the upcoming chapters. In addition to that, an user-friendly software has been designed that has a back-end integrated with NLP (API key of open-source *Llama3*) to process the user's prompt and return the `tuple` of maximum importance.
We have implemented a `json` based database to store the live locations of the obstacles, the objects as well as the robots for future reference and use.

Chapter-2 | Technological Framework

In this chapter, we discuss about how we plan to map the unchartered territory effectively, using Reinforcement Learning (RL), and SLAM Toolbox.

2.1 Utilising LiDAR

The Robot uses a 2D-LiDAR with 360-laser sampling in 360°. Thus an effective resolution of 1, the LiDAR data is used to create a high resolution map of the obstacles and the mapping system rejects moving objects dynamically, thus only recording static obstacles, it utilises the power of pre-trained computer vision models based on a high resolution RGB camera to identify points of interest around the map and continuously updates it in its database.

This LiDAR data is compiled in form of ROS2 Datatype `sensor_msgs/LaserScan` and passed to `SLAM_Toolbox` node which complies it to a `nav_msgs/Map` datatype. This map, along with robot's odometry and position data is sent to the RL Model via a ROS2-ML Bridge which translates the ROS2 data into ML processable format.

2.2 SLAM Implementation

We have utilised the `SLAM_Toolbox` for running SLAM on the cobots. It is a powerful, open-source library that supports a variety of functionalities and various SLAM methodologies, the ones that we have used in our implementation are as follow:

1. **Scan Matching:**
   - **Iterative Closest Point (ICP):** The ICP, as the name suggests is an iterative algorithm, which is designed to minimise the distance between the corresponding points in two point clouds. The algorithm iteratively minimizes the following cost function in order to align the point clouds A.1:
   $$E = \sum ||p_i - R_{p_j} - t||^2$$
   - **Normal Distributions Transform (NDT):** It is a description of the probability distribution of the points in the considered region of space. The algorithm aligns two point clouds under the assumption that likelihood of the second point cloud conditioned by the first one gets maximized.
2. **Pose-Graph Optimisation**: Optimising a pose graph is the optimisation of the estimated robot trajectory such that the error in the pose graph in general gets minimised. It solves a nonlinear least-squares problem:
   $$\text{Minimise:} \sum ||z_{ij} - T_{ij}^{-1} x_i + x_j||^2_{\sum_{ij}}$$
   references to the notation can be referred from here: A.1

2.3 ROS-ML Bridge

Our implementation includes a bridge system that facilitates seamless communication between ROS2 and our machine learning model. This bridge establishes a connection between the robot's core systems and our Reinforcement Learning (RL) model.

This bridge node independently translates data between ROS2 and the RL model. It subscribes to relevant ROS2 topics (Map, LiDAR, Odometry, and Position) to collect robot sensor data and forwards the extracted information to the RL model for processing in the form of Translational and Angular velocity vectors, and map in the form of Occupancy Grid Array. The RL model processes the data and generates velocity commands (translational and angular velocity vectors), which are then published to the appropriate robot's command velocity topic by the ROS2-ML bridge.

2.4 Reinforcement Learning

We have used an off-policy, model-free RL algorithm specialised for continuous action spaces. **Twin Delayed Deep Deterministic Policy** enhances efficiency and performance in environments with continuous actions by addressing overestimation bias and high-dimensional dynamics.

- **Policy Type**: Learns a deterministic policy $(\pi_\theta(s))$ while operating off-policy.
- **Critic Networks**: Utilizes two Q-networks to minimize overestimation bias.
- **Target Smoothening**: Reduces overfitting by introducing noise during target updates.

2.4.1 Algorithm Workflow

2.4.1.1 Twin Delayed Deep Deterministic Policy (TD3)

1. **Initialisation**: Actor: $\pi_\theta(s)$; the two critics are Q_{ϕ_1}, Q_{ϕ_2}, and their target networks. Replay buffer to store transitions.
2. **Interaction**: Collect transitions: (s, a, r, s') by interacting with the environment.
3. **Critic Update**: Compute target Q-value:
$$y = r + \gamma \cdot \min(Q_{\phi'_1}(s', a'), Q_{\phi'_2}(s', a'))$$
Minimize the TD error for: Q_{ϕ_1}, Q_{ϕ_2}.
4. **Actor Update**: Update the actor using: $\nabla_\theta J(\theta) \approx \nabla_a Q_{\phi_1}(s, a) \nabla_\theta \pi_\theta(s)$
5. **Target Network Update**: Smoothly update targets via Polyak averaging:
$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

2.4.2 Training Parameters

State Dimensions: 6	**Batch Size**: 64
Action Dimensions: 2	**Discount Factor** (γ): 0.99
Replay Buffer Size: $10,100$ transitions	**Target Smoothing** (τ): 0.005

Reward Function: The **RewardLiDAR** class encourages safe, efficient navigation:

1. **Collision Avoidance**: Penalties for proximity to obstacles or collisions.
2. **Safe Paths**: Rewards for maintaining safe distances from obstacles, prioritising forward paths.
3. **Action Efficiency**: Penalizes unnecessary or aggressive actions.
4. **LiDAR Denoising**: Smooths sensor data to avoid misjudgments.

2.4.3 LaserScan Preprocessing and Integration

To preprocess and normalise LiDAR data:

- Replace values exceeding the maximum range with a constant.
- Normalize distances by subtracting a collision threshold.

- Map closer objects to higher values for better sensitivity.
- Cap values to prevent distortion in high-proximity scenarios.

Replay buffer, reward function, and LiDAR preprocessing are modular and imported into `operatorNode.py`. Actor and critic networks are fine-tuned through iterative episodes. The `operatorNode.py` has an integrated ROS2-ML Bridge for communicating with the robots.

2.4.4 Working Mechanism

1. **State Observation:** Robots sense their environment to determine the current state (s_t) (position, orientation, sensor readings).
2. **Action Selection:** Generates a continuous action (linear and angular velocities).
3. **Environment Interaction:** The selected action updates the robot's state (s_{t+1}), producing a reward (r_t).
4. **Replay Buffer Update:** Transitions ($s_t, a_t, r_t, s_{t+1},$ done) are stored for training.
5. **Agent Update:**
    - **Critic:** Updates Q-values using sampled transitions.
    - **Actor:** Refines the policy to maximize returns.
    - **Target Networks:** Smoothly adjusted via Polyak averaging.
6. **Episode Completion:** The process continues until the robot:
    - Reaches the goal.
    - Collides with an obstacle.
    - Completes the maximum steps.

2.5 Simulation Setup in Gazebo

The simulation is done using the Gazebo Simulator (Version-11), in a custom created factory world featuring two rooms, each being $15m \times 15m$ and $10m \times 14m$ respectively, each room is filled with typical furniture and objects like shelves and tables and cardboard boxes, it is designed in such a way to showcase foolproof-ness of the algorithm. Further, there is the presence of humans (dynamic obstacles) who constantly walk around, motion of whose has been randomised.

2.6 Visualisation in RViZ2

RViZ2 is ROS2's 3D visualisation software, that allows us to visualise the live map that gets made using the LiDAR and the camera vision of the one that is mounted on the bot. Following is an example of the same, along with the nodes and the communication channels that exist in the running of the simulation.

2.7 Software

We have created a user-friendly interface to communicate with robotic architecture which can found here: Team-83 Kalyani Bharat Forge Software. The user can enter a prompt in the provided input bar, post which the input gets fetched to the backend, and NLP helps to identify the item of importance and returns the tuple, which is then passed on to the json database, which communicates with the `manager` robot and relays the required commands to the suitable robots, which provides timely feedback through which, the details are updated on the software for the user to know.

Chapter-3 | Experimental Setup

3.1 Environment Setup

A total of 20+ unique objects are present in the custom world that we have developed. The environment that we have developed (namely `factory.world`) consists of:

- **Absolutely Static Objects**: Like staircase, shelves, tables, these objects are assumed to be static in nature, and would be mapped unchanged since the process of exploration has been performed.
- **Moveable Static Objects**: These objects are static in nature but can change their locations over time, they may be something like a box, a fire extinguisher, etc.
- **Dynamic Objects**: These objects are assumed to be constantly moving around in the world, and their location shall not be mentioned in the database, like humans, forklifts, other moveable machinery or other robots, etc.

Following is the list of the test objects against their colors in the world that we are considering:

- Traffic Cones - orange and white
- Fire Extinguisher - red
- Cardboard boxes - brown
- Paller mover - dark yellow
- Chairs - dark grey
- Humans

3.2 Testing Protocols

1. **Task: exploration** - We load a pre-trained reinforcement learning based model onto the `manager` robot, which is further coded to spawn the cobots according to the specified conditions (when the first robot reaches a dead end on either side, the next robot shall be spawned which moves in a different direction and so on). The path of the cobots for exploration is decided using the RL model's output.
2. On successfully mapping the complete territory, the map of the factory shall be saved for localisation. And, now the cobots are ready to be assigned with tasks.
3. **Task: assigning tasks to cobots** - As the user enters a task that needs to be performed, and post the computation (for finding the minimum path, Euclidean minimum distance is calculated between each robot and the target object's presumed location), one of the cobot is assigned a task, which enters it's task *queue* (first-in-first-out requirement).

   While the *queue* of any robot is empty, it is performing random motion in the factory, and reporting the location of the target objects. As, any task is assigned to a robot, it switches to the same.
4. When the cobot finishes the task, the same is dequeued from the *queue list* and a message is published by the `manager` robot node, and the same reflects on the frontend.

Chapter-4 | Results and Evaluation

4.1 Effectiveness

- **Obstacle avoidance**: An accuracy of 100% can be obtained, because of the definite nature of the code and the level of abstraction we have assumed for the sake of simplicity.
- **Task Completion**: Refer to the equation written below for knowing about the mathematics that runs behind deciding the suitable robot. Now, that the task has been decided, it gets assigned to that particular robot by relaying a `message (msg) - target`, which then enqueues the task in the *queue* of that particular cobot. In case any other task gets relayed to this currently working robot, it gets queued up.

  $$\text{Robot selected} = \min d(r_i, o_j)$$

  where r_i represents the coordinates of the i^{th} robot and o_j represents the coordinates of the j^{th} object of the required kind.

4.2 Scalability

- The solution we present is highly scalable owing to the minimal number of hard-coded conditions. One can simply spawn as many bots as they wish to.
- For exploration, the RL model can work on as many cobots as required. And, the map merge algorithm that we have used is supported to work in RViZ for an *n* number of bots as well.
- Based on the current functioning of the project, given that you define what all could the cobot identify in the industry, it shall be able to map the same and store it in the database post that. There is no limitation to the database size or complexity.
- Since, the complete `manager` robot and the other cobots architecture has been deployed on ROS2, there is no limitation in the computational capabilities or communication.
- Owing to the decreased computational complexity of the solution, the solution can be used by a multi-variated range of companies.

4.3 Technical Innovations

- Smart utilisation of machine learning algorithms. For example, the path planning is done using the A^* algorithm, which, from our test ensures better results as compared to a delicate ML-based approach.
- While, for exploration the idea of reinforcement learning seems over the top, but as economy of scale is applicable here as well, owing to which as the project scales and the number of robots increases, the better would be the output.
- The model of abstraction that we have assumed are close to real life and fairly cover a lot of edge cases, without having to specially code focused to that. This further advocates the fact that the solution is industry-deployable in nature.

Chapter-5 | Conclusion and Future Work

5.1 Technical Difficulties

- The primary issue that we faced was the lack of sufficient resources related to the components of the robot.
- The project involved a lot of different aspects, coding for each of them took a considerable amount of time. The primary issue lies in the ability to make all of the things co-operate with each other, creation of separate `msg` and `topics`.
- Incorporation of a centralised machine learning algorithm instead of on-edge has a limited amount of resources available to it.
- ROS2 support for this kind of a swarm is not available, as much as it is for ROS1, in which there exists an autonomous SLAM swarm of robots itself - RRT Exploration.
- The lack of clear communication from the organising team led to some confusion on our part and ended up working in the wrong direction.

5.2 Future Plan of Action

- We plan to robustly integrate all the developed resources, this includes ongoing work on components currently under development, which will be finalized and incorporated into the framework.
- We plan to develop a Deep Learning based Computer Vision model to detect objects with significantly higher accuracy and precision, making it suitable for complex and dynamic environments.
- We plan to utilize depth cameras to improve spatial awareness and situational understanding. These sensors will enable to system to accurately perceive, interpret and respond to the three dimensional structure of the environment.

5.3 Learnings

- **Understanding Autonomous Navigation Systems :** Gain knowledge of algorithms used for autonomous mobile robot (AMR) navigation in GPS-denied environments.
- **Advanced Path Planning Techniques:** Understanding optimization techniques for task allocation and travel time minimization in robot swarms.
- **Multi-Agent Systems and Coordination:** Exploring concepts like swarm intelligence and co-operative task planning and learning to design and train multi-agent systems capable of collaboration and shared memory use.

Chapter-A | Appendix

A.1 Images

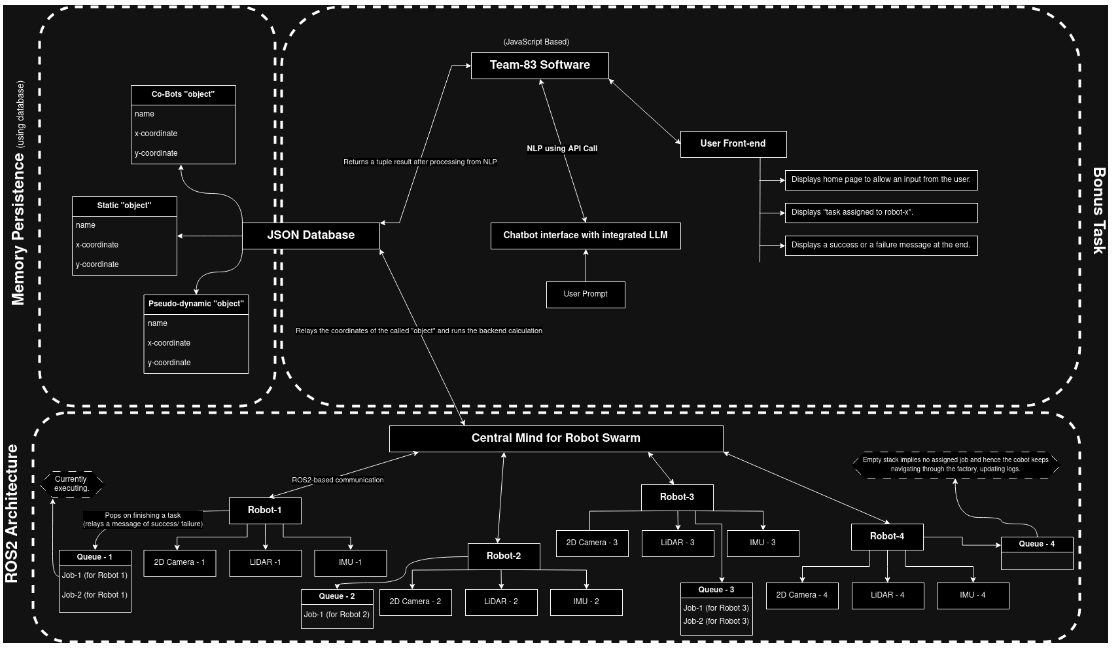


Figure A.1: System Overview for the Cobots Swarm

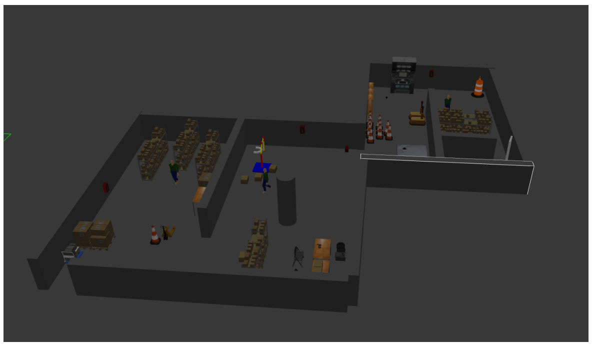


Figure A.2: Gazebo World Image - 1

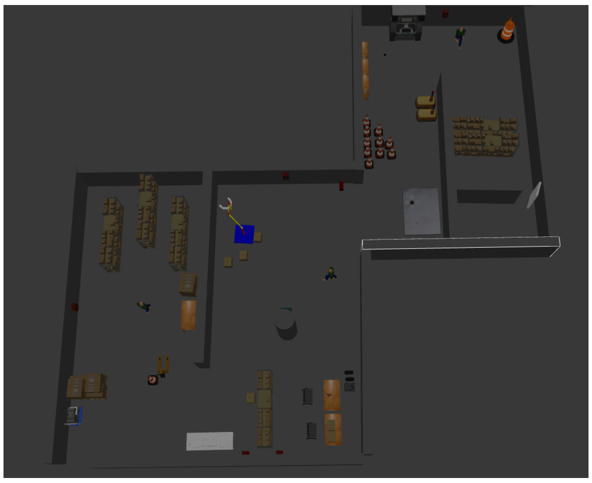
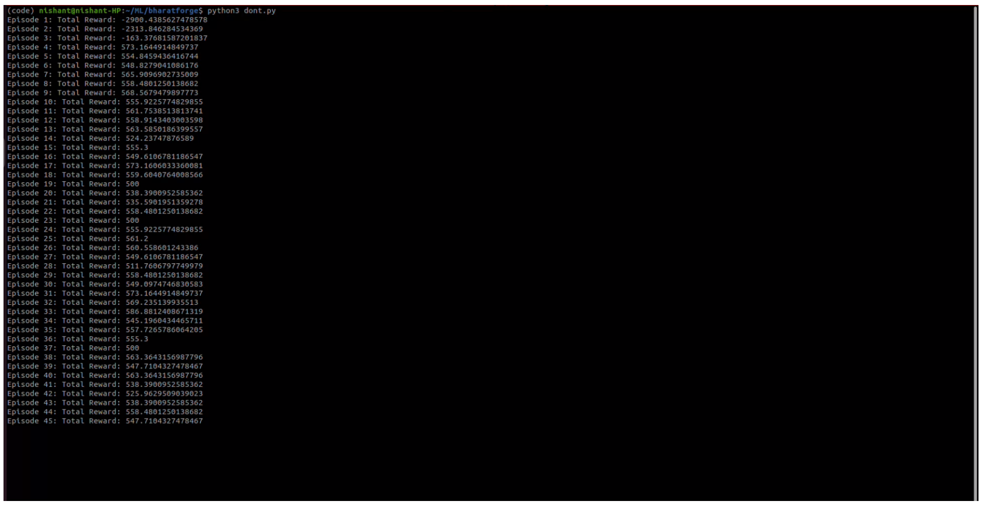


Figure A.3: Gazebo World Image - 2

A.2 Assumptions

Following is the list of assumptions that we have made while presenting our final submission:

- The world features a single entrance for all robots to walk into from, thereafter splitting into multiple paths. The simple reason for this is to ensure the best results in a generalised case,



```
(code) nishant@nishant-HP:~/ML/bharatforge$ python3 dont.py
Episode 1: Total Reward: -2900.4385627478578
Episode 2: Total Reward: -2313.846284534369
Episode 3: Total Reward: -163.37681587201837
Episode 4: Total Reward: 573.1644914849737
Episode 5: Total Reward: 554.8459436416744
Episode 6: Total Reward: 548.8279041086176
Episode 7: Total Reward: 565.9096902735009
Episode 8: Total Reward: 558.4801250138682
Episode 9: Total Reward: 568.5679479897773
Episode 10: Total Reward: 555.9225774829855
Episode 11: Total Reward: 561.7538513813741
Episode 12: Total Reward: 558.9143403003598
Episode 13: Total Reward: 563.5850186399557
Episode 14: Total Reward: 524.23747876589
Episode 15: Total Reward: 555.3
Episode 16: Total Reward: 549.6106781186547
Episode 17: Total Reward: 573.1606033360081
Episode 18: Total Reward: 559.6040764008566
Episode 19: Total Reward: 500
Episode 20: Total Reward: 538.3900952585362
Episode 21: Total Reward: 535.5901951359278
Episode 22: Total Reward: 558.4801250138682
Episode 23: Total Reward: 500
Episode 24: Total Reward: 555.9225774829855
Episode 25: Total Reward: 561.2
Episode 26: Total Reward: 560.558601243386
Episode 27: Total Reward: 549.6106781186547
Episode 28: Total Reward: 511.7606797749979
Episode 29: Total Reward: 558.4801250138682
Episode 30: Total Reward: 549.0974746830583
Episode 31: Total Reward: 573.1644914849737
Episode 32: Total Reward: 569.235139935513
Episode 33: Total Reward: 586.8812408671319
Episode 34: Total Reward: 545.1960434465711
Episode 35: Total Reward: 557.7265786064205
Episode 36: Total Reward: 555.3
Episode 37: Total Reward: 500
Episode 38: Total Reward: 563.3643156987796
Episode 39: Total Reward: 547.7104327478467
Episode 40: Total Reward: 563.3643156987796
Episode 41: Total Reward: 538.3900952585362
Episode 42: Total Reward: 525.9629509039023
Episode 43: Total Reward: 538.3900952585362
Episode 44: Total Reward: 558.4801250138682
Episode 45: Total Reward: 547.7104327478467
```

Figure A.4: Reinforcement Learning - Episode v/s Reward

under which we assume the factory initially to be completely unknown (as a `black-box`), except that one entrance.

- We have assumed that one is already aware of the ”kind” of objects they may find inside the factory like stair, fire extinguisher, etc. But, the labelling of the location of these objects shall be done automatically using computer vision.
- To simplify the mapping process without compromising generality, we have assigned specific colors to be explicitly and exclusively mapped to the following objects:
  - **Traffic Cones:**
    - Orange : $\#5f1e09$
    - White : $\#ffffff$
  - **Fire Extinguisher Boxes:**
    - Red : $\#3d0403$
  - **Cardboard Boxes, Pallets, and Shelves:**
    - Brown : $\#463825$
  - **Pallet Movers:**
    - Dark Yellow : $\#4f3506$
  - **Chairs:**
    - Dark Grey : $\#111111$
  - **Humans :**
    - Shirt :
      - Green : $\#0a1a08$
    - Pants :
      - Blue : $\#060924$

These Color-Object mappings ensure unambiguous identification under all scenarios.

A.3 Notations

Symbols	Meaning
p_i & p_j	Two points in the cloud of points
R	Rotation Matrix
t	Translation vector
z_{ij}	Relative assessment between poses i & j
T_{ij}	Relative transformation between poses i and j
x_i & x_j	Poses of nodes i and j within the 3D space
$\sum_{ij}$	Covariance matrix of z_{ij}

Table A.1: Notations Table

A.4 Pseudocode:

Algorithm 1 Task Allocation System

```
Require: User specifies a task with a target location $T$
Ensure: Task $T$ is assigned to the appropriate robot
 1: Input Topics:
 2:    /robot_positions: Contains robot names, positions, and timestamps
 3:    /target: Accepts the target location $T$
 4: Output Topic:
 5:    /target_return: Publishes the name of the assigned robot
 6: Initialization:
 7: Parse the input target $T$ from the /target topic
 8: Retrieve robot data from the /robot_positions topic
 9: Extract robot positions RobotPositions $= \{(r_1, p_1), (r_2, p_2), \ldots, (r_m, p_m)\}$
10: Initialize $ShortestDistance \leftarrow \infty$, $AssignedRobot \leftarrow$ None
11: for each $(r, p) \in$ RobotPositions do
12:     if Robot $r$ is not busy then
13:         Compute distance $d =$ Distance$(p, T)$
14:         if $d < ShortestDistance$ then
15:             $ShortestDistance \leftarrow d$
16:             $AssignedRobot \leftarrow r$
17:         end if
18:     end if
19: end for
20: if $AssignedRobot \neq$ None then
21:     Assign target $T$ to $AssignedRobot$
22:     Publish $AssignedRobot$ on /target_return
23: else
24:     Find the robot $r'$ whose task queue's next destination is closest to $T$
25:     Add $T$ to $r'$'s task queue
26:     Publish $r'$ on /target_return
27: end if
28: Relay /target_return message to the backend
```