ROBOADS

Roboads Coding Interview

Submission Deadline: 16/02/2024

Task Description:

Objective: Write custom ROS nodes for Autonomous Docking/Charging of Turtlebot3 Robot, within a simulated Gazebo world environment. The system should enable the robot to autonomously navigate back to the docking station, when the battery level is below 30%. Utilize ArUco Marker/AR Tag Transformation for localization of the docking station within the map. Utilize depth camera and laser scanner for precise positioning of the robot with the center of the docking station.

Key Requirements:

1.Simulation Environment Setup:

- Utilize the Gazebo simulation environment to model the TurtleBot3 and the docking station.
- Incorporate a custom simulation world that includes obstacles and a docking station marked with an ArUco marker/AR Tag for the robot to interact with.

2. Robot Localization and Navigation

- Use SLAM to build and save a map, use localization algorithm to localize on the map and keep navigating to 4 set goals in a continuous loop. Implement ArUco marker/AR Tag detection to localize the docking station within the map.
- Develop a method for the robot to monitor its battery status and initiate the docking procedure when the battery is below a predefined threshold which is 30%.

3. Docking Logic:

- Use depth camera and laser scanner data to navigate the robot accurately to the docking station.
- Implement algorithms to align and position the robot precisely at the center of the docking station, ensuring successful docking. Start a timer as soon as the robot reaches near the dock and starts reversing and aligning to track the time your algorithm takes to successfully dock.
- Extend the functionality by adding a feature where the robot can also autonomously undock and resume its goals after charging.

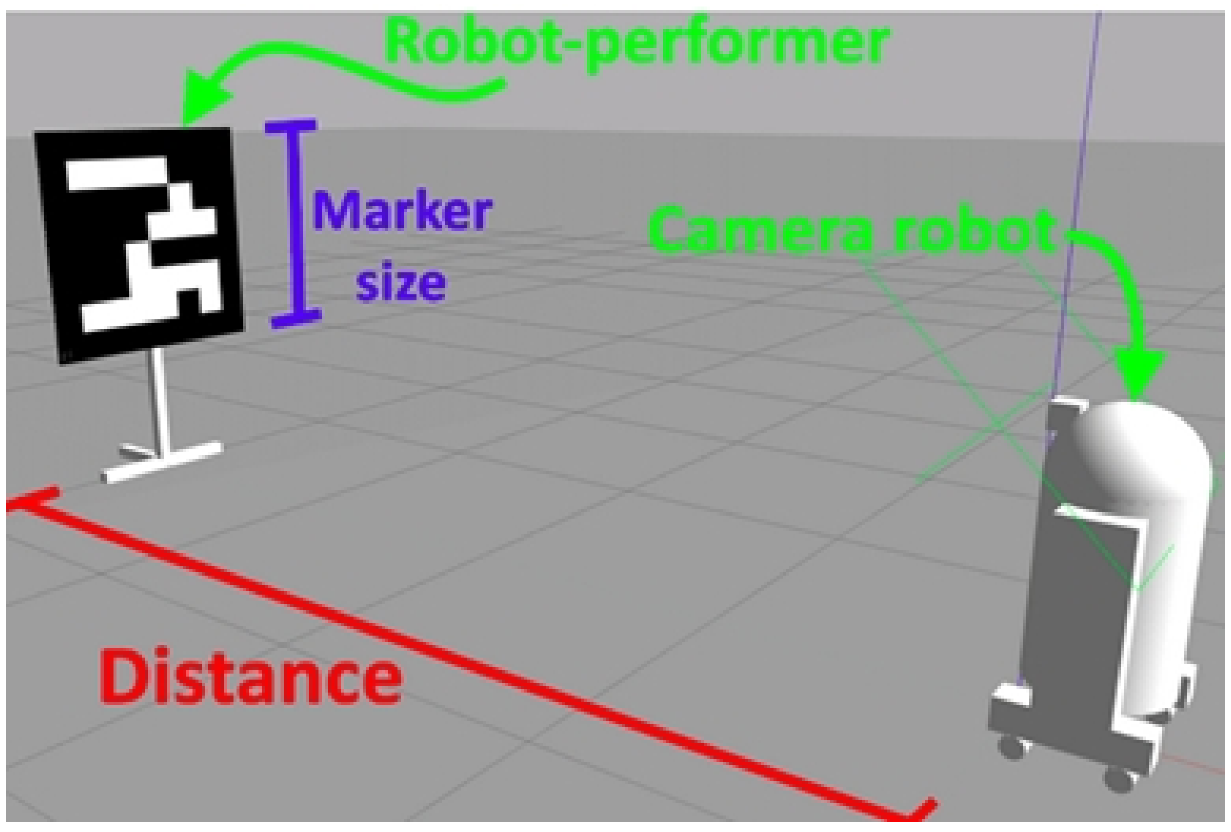
Resources and Tools :

ROS Packages:

- turtlebot3_simulations for TurtleBot3 simulation models.
- aruco_ros/ar_track_alvar for ArUco/AR tag marker detection.
- move_base for autonomous navigation and path planning.
- gmapping or slam_gmapping for SLAM to create a map of the

Simulation World and Docking Station Object:

You can use any gazebo simulation world, please define a particular area or a dock with a ArUco tag/AR tag mounted on it representing a docking station.



Testing and Evaluation:

- Autonomy and Precision: The system must autonomously detect the low battery condition, locate the docking station using the ArUco marker, and accurately dock without human intervention.
- Obstacle Avoidance: Demonstrate the robot's ability to navigate around obstacles while moving towards the docking station.
- Efficiency: The docking time and accuracy of alignment with the docking station will be also evaluated.

Presentation and Code Sharing :

- Documentation: Candidates must provide comprehensive documentation detailing their design choices, algorithms, and a user guide for setting up and running the simulation.
- Code Sharing: Submit the complete ROS package, including source code, simulation environments, and necessary configurations, via a public Git repository (e.g., GitHub).
- Demonstration: Prepare a video demonstration showcasing the successful implementation of the docking system, including scenes of the robot autonomously docking.

Additional Tasks for Bonus Points:

- Use the latest State of the art algorithms for SLAM and Localization instead of the traditional Gmapping and AMCL respectively. Evaluate their pros and cons with traditional algorithms mentioned above.
- Use Dynamic Gazebo Simulation environment preferably with Humans walking.
- If you have successfully imported dynamic objects, then add an extra layer of obstacle detection where the robot would stop at a safe distance away from the dynamic obstacle instead of avoiding and going around it. Only continue navigation when the obstacle has moved out of the way. Create a separate node for this task which subscribes to the depth camera and laser scanner for detection.

Deliverables:

This is a general idea of the expected nodes from the task, you combine certain nodes or create nodes for modularity and better design purposes.

1. Battery Monitor Node:

- **Function**: Monitors the robot's battery status and publishes a notification or a service call to initiate docking when the battery level falls below 30%.
- **Deliverables**: Source code files (e.g., battery_monitor_node.py or .cpp), launch file to start the node.

2. ArUco Marker Detection Node:

- **Function**: Detects ArUco markers in the environment, extracting their positions and orientations to use for localizing the docking station within the robot's map.
- **Deliverables**: Source code integrating aruco_ros for marker detection, launch file for the node, configurations for marker detection parameters.

3. Docking Control Node:

- **Function**: Manages the logic and actions for navigating to the docking station and performing the precise docking maneuver, utilizing inputs from the depth camera and laser scanner.
- **Deliverables**: Source code (e.g., docking_control_node.py or .cpp), launch file for the node, and any configuration files for tuning navigation and docking behaviors.

4. Advanced Obstacle Avoidance Node (Bonus):

- **Function**: Utilizes move_base and sensor data to navigate towards the docking station while detecting and stopping for dynamic obstacles.
- **Deliverables**: Source code (e.g., advanced_obstacle_detection.py or .cpp), launch file for the node, and any configuration files for changing the parameters

5. Continuous Goal to Goal Navigation:

- Function: Utilizes move_base action client interface for continuous looped navigation, to 4-5 goals until the docking sequence is called and can also autonomously undock and resume its goals after charging.
- Deliverables: Source code (e.g., continuous_waypoint_navigation.py or .cpp), launch file for the node, and any configuration files for changing the parameters

Simulation Environment Files

6. Gazebo World File (*.world):

- **Function**: Defines the simulation environment, including obstacles, floor plan, and the docking station with an ArUco marker.
- **Deliverables**: The world file that can be loaded into Gazebo, along with any associated models and textures.

7. URDF/Xacro Files for the Docking Station:

- **Function**: Describes the 3D model of the docking station, including the physical properties and the ArUco marker placement.
- **Deliverables**: URDF/Xacro files defining the docking station model, and associated mesh files for visualization in Gazebo.

Documentation and Demonstrations

- **README.md**: A comprehensive guide detailing how to set up and run the simulation, descriptions of each node, and the overall architecture of the docking system.
- **Video Demonstration**: A recorded demonstration showing the successful execution of the docking process in the Gazebo simulation environment.

Additional Files

- **Launch Files**: For launching the entire simulation setup, including the TurtleBot3 robot in the Gazebo environment, all necessary ROS nodes, and configurations for the docking process.
- **Configuration Files**: YAML or other configuration files for tuning the parameters of the navigation stack, ArUco marker detection, and any other aspect of the system that requires fine-tuning.
- **Scripts/Utilities**: Any additional scripts or utilities developed to facilitate testing, debugging, or data analysis related to the docking process.

Note:

- Consider handling error conditions and optimizing the code for real-world applications.
- Writing the code in OOPs (Object-oriented programming) style is a big bonus.
- You can use Python/C++ nodes, C++ nodes will be given higher weightage, though a combination of both will also be highly marked.
- Please focus on the primary task of docking, bonus tasks will only be marked if the primary task functions well enough.
- Use your creativity as much as possible to solve the problem, any extra efforts to represent your work will be given extra attention.

EOD