Butler Robot ROS Node Documentation

# **Introduction**

This document provides an in-depth description of the butler\_robot node that is implemented in Python. In this document, a butler robot is described with its intended environment being a cafe where it can move through home position, kitchen, or to customer tables to give orders. Many scenarios are accepted by the robot, including but not limited to waiting for confirmations, handling timeouts as well as cancellations; the robot can process the order in batches.

# **Code Overview**

The ButlerRobot class inherits from rclpy.node.Node and contains all the functionalities required for the robot's operation within the ROS framework. The node communicates with other components through publishers and subscribers, which handle messages on tasks, positions, confirmations, and cancellations.

# **Explanation of Key Components**

## **Publishers and Subscribers**

* **Publishers**
  + /goal\_pose (PoseStamped): Publishes goal positions for navigation.
* **Subscribers**
  + /task\_topic (String): Receives new tasks (e.g., table1,table2,table3).
  + /odom (Odometry): Receives the current position of robot .
  + /confirmation\_topic (String): It listens for confirmations in the kitchen or tables(ex. kitchen/table1/table2/table3).
  + /cancellation\_topic (String): It gets canceling a task requests(e.g. table1,table2,table3).

## **State Variables**

* **Timeout Settings**
  + timeout\_duration: time in seconds it must wait to get a confirmation.
  + remaining\_timeout: Time left in the current timeout countdown.
  + timeout\_timer: Timer object for the timeout mechanism.
* **Task Management**
  + task\_queue: Queue holding the pending tasks.
  + current\_task: The task currently being executed.
* **Flags**
  + task\_in\_progress: Indicates if a task is in progress.
  + waiting\_for\_confirmation: Indicates if the robot is waiting for a confirmation.
  + waiting\_started: Prevents multiple initializations of the waiting state.
  + confirmation\_received: Indicates if a confirmation has been received.
  + returning\_home: Indicates if the robot is returning home.
  + returning\_kitchen: Indicates if the robot is returning to the kitchen.
  + timeout\_reached: Indicates if the timeout has been reached.
  + unconfirmed\_deliveries: Tracks if there are any unconfirmed deliveries.

## **Methods**

* **Task Handling**
  + task\_callback: Processes incoming tasks and updates the queue.
  + execute\_task: Executes tasks based on the robot's state and transitions.
  + confirmation\_callback: Handles confirmations from the kitchen or tables.
  + cancellation\_callback: Manages task cancellations.
* **Navigation**
  + move\_to\_kitchen: Commands the robot to move to the kitchen.
  + move\_to\_table: Commands the robot to move to a specified table.
  + return\_home: Commands the robot to return to the home position.
  + return\_to\_kitchen: Commands the robot to return to the kitchen before going home.
  + publish\_goal: Publishes navigation goals to the /goal\_pose topic.
* **Position Monitoring**
  + position\_callback: Updates the robot's position and checks goal attainment.
  + has\_reached\_goal: Determines if the robot has reached its goal.
* **Confirmation and Timeout Management**
  + start\_waiting\_for\_confirmation: Initiates the waiting period for confirmations.
  + handle\_timeout: Handles actions when the timeout is reached without confirmation.
  + stop\_timeout: Stops the timeout timer and resets related flags.
  + proceed\_to\_next\_phase: Moves to the next phase after receiving confirmation.
* **Task Completion**
  + end\_task: Resets state variables after all tasks are complete.

## **Flow of Execution**

1. **Receiving Tasks**
   * Tasks are received via the /task\_topic and added to the task\_queue.
   * If the robot is idle, it starts executing tasks.
2. **Moving to Kitchen**
   * The robot moves from the home position to the kitchen.
   * Upon arrival, it may wait for a confirmation from the kitchen.
3. **Handling Confirmations and Timeouts**
   * If a confirmation is received, the robot proceeds to the next phase.
   * If a timeout occurs, the robot handles it based on the current state.
4. **Moving to Tables**
   * The robot moves from the kitchen to the customer tables as per the tasks.
   * It may wait for confirmations at each table.
5. **Handling Unconfirmed Deliveries**
   * If a delivery at a table is unconfirmed, the robot marks it and continues.
   * Unconfirmed deliveries may cause the robot to return to the kitchen before going home.
6. **Handling Cancellations**
   * The robot listens for cancellation messages.
   * If a task is cancelled, it adjusts the task queue and state accordingly.
7. **Returning Home**
   * After completing all tasks, the robot returns to the home position.
   * It resets its state and is ready for new tasks.

## **Coordinate Mapping**

* **Home Position**: (5.496074199676514, 2.938681125640869, 0.11181640625)
* **Kitchen**: (-3.001096725463867, 1.293923020362854, 0.002471923828125)
* **Table1**: (0.6848546266555786, -0.13853590190410614, 0.002471923828125)
* **Table2**: (3.0809290409088135, 0.06948712468147278, 0.002471923828125)
* **Table3**: (4.854504108428955, 0.06061259284615517, 0.002471923828125)

## **Testing and Usage**

To test the node:

1. **Initialize the ROS Environment**: Ensure that ROS is properly installed and sourced.
2. **Run the Node**: Execute the script to start the butler\_robot node.

ros2 run turtlebot3\_custom butler\_robot.py

1. **Send Tasks**: Publish messages to the /task\_topic to assign tasks.

ros2 topic pub /task\_topic std\_msgs/String "data: 'table1, table2'"

1. **Send Confirmations**: Publish messages to the /confirmation\_topic when the robot reaches the kitchen or tables.

ros2 topic pub /confirmation\_topic std\_msgs/String "data: 'kitchen'"

ros2 topic pub /confirmation\_topic std\_msgs/String "data: 'table1'"

1. **Send Cancellations**: Publish messages to the /cancellation\_topic to cancel tasks.

ros2 topic pub /cancellation\_topic std\_msgs/String "data: 'table2'"

# **Conclusion**

The ButlerRobot node is designed to fulfill the requirements outlined in the problem statement. It handles various complex scenarios, including waiting for confirmations with timeouts, managing cancellations, and processing multiple orders. The code is structured to be generic, avoiding hardcoded algorithms, and follows ROS best practices.