BUTLER_BOT

1st Milestone: Task Navigation and Dynamic Return to Home

Objective

The goal of this milestone is to enable the robot to:

1. Navigate to the **kitchen** to pick up tasks.
2. Deliver orders to specified **tables** sequentially.
3. **Return to home** when:
    - All tasks in the `task_queue` are completed.
    - The robot is at the **kitchen** after task completion.

Ideology Behind the Implementation

Modular Design

The script is organized into independent modules to manage specific functionalities:

- **Task Initiation:** Adds tasks to a queue when new orders are received.
- **Goal Navigation:** Publishes navigation goals based on predefined coordinates.
- **Task Execution:** Processes tasks sequentially using the task queue.
- **Post-Task Behavior:** Ensures the robot returns to "home" intelligently.

Event-Driven Logic

Robot behavior is triggered by events:

1. **New Orders:** Start task execution when the robot receives orders via the `/new_order` topic.
2. **Navigation Results:** Use navigation status updates to decide the robot's next action.

State Management

The robot's current state (`current_task`) dictates its behavior:

- `"kitchen"`: Picking up orders.
- `"tableX"`: Delivering to a table.
- `None`: Idle and waiting for new tasks or returning to "home."

State transitions ensure predictable and efficient navigation:

- **After reaching the kitchen:** Deliver the next table task.
- **After completing all tasks:** Return to "home."

Goal-Based Navigation

The robot uses the `publish_goal()` function to send goals to the `/move_base_simple/goal` topic. Goals are defined in a YAML file with x, y, and yaw coordinates for locations like "kitchen" and "home." This approach ensures precise navigation.

Implementation Details

1. **Robot Initialization:**

The robot starts at "home" by default:
```
self.publish_goal("home")
```

2. **Receiving New Orders:**

When new orders are received, they are added to the `task_queue`, and task execution begins:
```
if self.current_task is None:
    self.handle_kitchen_task()
```

3. **Kitchen Navigation:**

The robot navigates to the kitchen to pick up orders:
```
def handle_kitchen_task(self):
    self.current_task = "kitchen"
    self.publish_goal("kitchen")
```

4. **Task Execution:**

After reaching the kitchen, the robot delivers orders sequentially:
```
if self.current_task == "kitchen":
    table = self.task_queue.pop(0)
    self.current_task = table
    self.publish_goal(table)
```

5. **Return to Kitchen:**

After delivering to a table, the robot navigates back to the kitchen to process the next task:

```
if self.current_task.startswith("table"):
    if self.task_queue:
        self.current_task = "kitchen"
        self.publish_goal("kitchen")
```

6. **Dynamic Return to Home:**

If the task queue is empty when the robot is at the kitchen, it navigates to "home."

```
if not self.task_queue and self.current_task == "kitchen":
    self.current_task = None
    self.publish_goal("home")
```

Key Features Achieving the Milestone

1. **Sequential Task Processing:**
   - Tasks are handled in order, ensuring smooth execution.
2. **Dynamic Return to Home:**
   - The robot intelligently returns to "home" only after completing all tasks.
3. **Event-Driven State Transitions:**
   - The robot transitions between goals based on navigation success and task queue status.
4. **Configurable Goals via YAML:**
   - The system is easy to configure and extend with new locations.

Advantages of This Approach

1. **Efficiency:**
   - Minimizes unnecessary movement by only returning home when tasks are completed.
2. **Scalability:**
   - Can easily handle more locations or complex tasks by extending the YAML and state management logic.
3. **Reliability:**
   - Predictable behavior ensures smooth task execution and goal transitions.

This milestone establishes a solid foundation for more advanced functionalities, such as handling task cancellations, confirmations, or multi-robot coordination.

2nd and 3rd Milestone Enhancements

The 2nd and 3rd milestones build upon the foundation of the 1st milestone by incorporating **confirmation handling** and **dynamic response** to ensure efficient order processing and delivery. Here's how the script has evolved to address these milestones:

2nd Milestone: Confirmation Handling

Objective

To ensure that the robot confirms task completion at key locations:

- **Kitchen Confirmation:** Verify that the robot has picked up the order at the kitchen.
- **Table Confirmation:** Verify that the robot has delivered the order to the table.

Changes Implemented

1. **New Subscribers for Confirmation:**

Added `/kitchen_confirm` and `/table_confirm` subscribers to handle confirmation signals from external systems:

```
rospy.Subscriber("/kitchen_confirm", Bool,
self.kitchen_confirm_callback)
rospy.Subscriber("/table_confirm", Bool, self.table_confirm_callback)
```

Callback functions update `kitchen_confirmation` and `table_confirmation` flags:

```
def kitchen_confirm_callback(self, msg):
    self.kitchen_confirmation = msg.data
    rospy.loginfo(f"Kitchen confirmation received:
{self.kitchen_confirmation}")

def table_confirm_callback(self, msg):
    self.table_confirmation = msg.data
    rospy.loginfo(f"Table confirmation received:
{self.table_confirmation}")
```

2. **Wait for Confirmation Logic:**

Introduced a `wait_for_confirmation()` method that waits for a specified confirmation (kitchen or table) with a timeout:

```
def wait_for_confirmation(self, location, timeout=30):
    start_time = rospy.Time.now()
    rate = rospy.Rate(1)  # 1 Hz

    while (rospy.Time.now() - start_time).to_sec() < timeout:
        if location == "kitchen" and self.kitchen_confirmation:
            self.kitchen_confirmation = False
            return True
        elif location.startswith("table") and self.table_confirmation:
            self.table_confirmation = False
            return True
        rate.sleep()

    return False
```

3. **Integration into Navigation Workflow:**

The robot now waits for confirmation at the kitchen and table locations:

At the kitchen:

```
if self.wait_for_confirmation("kitchen"):
    rospy.loginfo("Kitchen confirmation received. Proceeding to
table.")
    if self.task_queue:
        table = self.task_queue.pop(0)
        self.current_task = table
        self.publish_goal(table)
else:
    rospy.logwarn("No kitchen confirmation received. Returning to
home.")
    self.current_task = "home"
    self.publish_goal("home")
```

At the table:

```
rospy.loginfo(f"Reached {self.current_task}. Waiting for 30 seconds
confirmation...")
self.wait_for_confirmation(self.current_task)
rospy.loginfo("Returning to kitchen to check for new orders.")
self.current_task = "kitchen"
self.publish_goal("kitchen")
```

Key Benefits

- Ensures task accuracy by verifying order pickup and delivery.
- Handles scenarios where confirmation signals are delayed or not received (using timeouts).

3rd Milestone: Dynamic Task and Goal Management

Objective

To enhance robot decision-making:

- Dynamically handle **task queue emptiness** at key locations.
- Efficiently return to **home** when no tasks are pending after reaching the kitchen.

Changes Implemented

1. **Dynamic Task Queue Check:**
    - The robot checks the `task_queue` at critical points to determine the next action:

After completing all tasks:

```
elif self.current_task == "home":
    self.current_task = None
    if self.task_queue:
        self.handle_kitchen_task()
```

Upon returning to the kitchen with an empty task queue:

```
if not self.task_queue and self.current_task == "kitchen":
    rospy.loginfo("No pending tasks. Returning to home.")
    self.current_task = "home"
    self.publish_goal("home")
```

2. **Task Queue and Home Integration:**

The robot only returns to "home" after confirming that no further tasks are pending:

```
else:
    rospy.loginfo("Returning to home after completing all tasks.")
    self.current_task = "home"
    self.publish_goal("home")
```

3. **Improved Navigation Flow:**
    - The state transitions for goal navigation now include:
        - **Reconfirming tasks at the kitchen.**
        - **Returning to home when idle.**

Key Benefits

- Prevents unnecessary navigation to "home" unless all tasks are processed.
- Smoothly transitions between tasks, reducing idle time and inefficiency.
- Readily adapts to real-time task updates.

Summary of Milestone Enhancements

2nd Milestone: Confirmation Handling

- Introduced confirmation mechanisms (`kitchen_confirmation` and `table_confirmation`).
- Added a `wait_for_confirmation()` method to validate task completion at specific locations.

3rd Milestone: Dynamic Task and Goal Management

- Implemented checks for task queue emptiness at the kitchen and dynamically transitioned to "home."
- Refined state transitions for efficient navigation and task execution.

These milestones significantly improve the robot’s ability to handle real-world tasks with accuracy and adaptability.

Milestones 4, 5, 6, and 7: Final Implementation

The final script enhances the robot's functionality significantly, addressing all milestones and ensuring robust, real-world operation. Here's a detailed explanation of the improvements:

Milestone 4: Task Cancellation

Objective

Allow tasks to be canceled dynamically, whether they are currently being executed or queued.

Changes Implemented

1. **New Subscriber for Cancellation:**

Added a `/cancel_task` subscriber to listen for task cancellation requests:

```
rospy.Subscriber("/cancel_task", String, self.cancel_callback)
```

2. **Task Cancellation Logic:**
    - Canceled tasks are handled in two cases:

Currently Executing Task: If the robot is performing the task to be canceled:

```
if self.current_task == cancel_order:
    rospy.logwarn(f"Current task {cancel_order} canceled!")
    self.current_task = None
    self.publish_goal("home")
```

Task in Queue: If the task is in the `task_queue`:

```
elif cancel_order in self.task_queue:
    self.task_queue.remove(cancel_order)
    rospy.loginfo(f"Task {cancel_order} removed from the queue.")
```

Key Benefits

- Offers dynamic control over the robot's task execution.
- Prevents unnecessary navigation for canceled tasks.

Milestone 5: Improved Confirmation System

Objective

Enhance the confirmation mechanism to ensure reliability and reset states post-confirmation.

Changes Implemented

1. **Resetting Confirmation Flags:**

After a confirmation signal is received, flags are reset to ensure accurate future checks:

```
if location == "kitchen" and self.kitchen_confirmation:
    self.kitchen_confirmation = False  # Reset for next use
    return True
elif location.startswith("table") and self.table_confirmation:
    self.table_confirmation = False  # Reset for next use
    return True
```

2. **Timeout Handling:**

Ensures the robot doesn't get stuck waiting for a confirmation indefinitely:

```
def wait_for_confirmation(self, location, timeout=30):
    ...
    return False  # Timeout occurred
```

Key Benefits

- Prevents errors due to stale confirmation signals.
- Maintains smooth operation with timeouts for unresponsive systems.

Milestone 6: Refined Task Flow

Objective

Ensure seamless transitions between tasks and states, with better handling of idle states and the task queue.

Changes Implemented

1. **Transition to Home When Idle:**

When no tasks remain, the robot automatically returns to the home location:

```
elif self.current_task == "home":
    self.current_task = None
    if self.task_queue:
```

```
        self.handle_kitchen_task()
```

2. **Queue-Driven Task Execution:**

Automatically navigates to the kitchen when tasks are available:

```
if self.current_task is None and self.task_queue:
    self.handle_kitchen_task()
```

Key Benefits

- Reduces unnecessary navigation.
- Keeps the robot ready for new tasks by returning to "home" when idle.

Milestone 7: Final Execution Loop

Objective

Implement a centralized task execution loop to manage the robot's operations in real-time.

Changes Implemented

1. **Central Execution Loop:**

Added an `execute_tasks()` method to continuously manage the robot's state and tasks:

```
def execute_tasks(self):
    rate = rospy.Rate(1)  # 1 Hz
    while not rospy.is_shutdown():
        if self.current_task is None and self.task_queue:
            self.handle_kitchen_task()
        rate.sleep()
```

2. **Main Entry Point:**

Ensures continuous task management and navigation:

```
if __name__ == "__main__":
    robot = ButlerRobot()
    robot.execute_tasks()
```

Key Benefits

- Centralized control of task execution.
- Keeps the robot in an operational loop, ready for new commands.

Summary of Milestone Enhancements

Milestone 4: Task Cancellation

- Added `/cancel_task` subscriber.
- Allowed cancellation of both currently executing and queued tasks.

Milestone 5: Improved Confirmation System

- Implemented reliable confirmation mechanisms with flag resets and timeouts.

Milestone 6: Refined Task Flow

- Seamless transitions between kitchen, table, and home locations.
- Improved handling of idle states and task queue.

Milestone 7: Final Execution Loop

- Centralized task execution with `execute_tasks()` method.
- Ensures continuous, real-time operation.

Final Features Overview

1. **Dynamic Task Management:**
    - Add, execute, or cancel tasks in real-time.
2. **Confirmation-Based Execution:**
    - Ensures reliability at critical points (kitchen and table).
3. **Intelligent State Transitions:**
    - Efficient navigation between tasks and home.
4. **Robust Error Handling:**
    - Handles unknown locations, timeouts, and cancellations gracefully.

This script represents a fully functional, adaptable solution for the robot's operational requirements.

Final Script:

```python
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import PoseStamped
from tf.transformations import quaternion_from_euler
import yaml
from std_msgs.msg import String, Bool
from move_base_msgs.msg import MoveBaseActionResult

class ButlerRobot():
    def __init__(self):
        rospy.init_node("goal_publisher")
        self.goal_pub = rospy.Publisher("/move_base_simple/goal", PoseStamped, queue_size=10)
        rospy.Subscriber("/new_order", String, self.order_callback)
        rospy.Subscriber("/cancel_task", String, self.cancel_callback)
        rospy.Subscriber("/kitchen_confirm", Bool, self.kitchen_confirm_callback)
        rospy.Subscriber("/table_confirm", Bool, self.table_confirm_callback)
        rospy.Subscriber("/move_base/result", MoveBaseActionResult, self.goal_result_callback)

        # Load goals from YAML file
        self.goals = self.load_goals("/home/sk/ws/goat_robotics/src/butler/src/goals.yaml")

        self.task_queue = []
        self.current_task = None
        self.kitchen_confirmation = False
        self.table_confirmation = False
        self.status = None

        # Initialize robot at home
        self.publish_goal("home")

    def load_goals(self, file_path):
        try:
            with open(file_path, "r") as file:
                data = yaml.safe_load(file)
            rospy.loginfo("Goals loaded successfully.")
            return data["goals"]
        except Exception as e:
            rospy.logerr(f"Failed to load goals: {e}")
            return {}

    def order_callback(self, msg):
        new_orders = msg.data.split()
        self.task_queue.extend(new_orders)
        rospy.loginfo(f"New orders received: {new_orders}")
        if self.current_task is None:
            self.handle_kitchen_task()

    def kitchen_confirm_callback(self, msg):
        self.kitchen_confirmation = msg.data
        rospy.loginfo(f"Kitchen confirmation received: {self.kitchen_confirmation}")

    def table_confirm_callback(self, msg):
        self.table_confirmation = msg.data
        rospy.loginfo(f"Table confirmation received: {self.table_confirmation}")
```

```
def wait_for_confirmation(self, location, timeout=30):
    start_time = rospy.Time.now()
    rate = rospy.Rate(1)  # 1 Hz

    while (rospy.Time.now() - start_time).to_sec() < timeout:
        if location == "kitchen" and self.kitchen_confirmation:
            self.kitchen_confirmation = False  # Reset for next use
            return True
        elif location.startswith("table") and self.table_confirmation:
            self.table_confirmation = False  # Reset for next use
            return True
        rate.sleep()

    return False  # Timeout occurred

def goal_result_callback(self, msg):
    self.status = msg.status.status
    if self.status == 3:  # Goal reached successfully
        rospy.loginfo("Navigation completed successfully.")

        if self.current_task == "kitchen":
            rospy.loginfo("Reached kitchen. Waiting for 30 seconds confirmation...")
            if self.wait_for_confirmation("kitchen"):
                rospy.loginfo("Kitchen confirmation received. Proceeding to table.")
                if self.task_queue:
                    table = self.task_queue.pop(0)
                    self.current_task = table
                    self.publish_goal(table)
                else:
                    rospy.loginfo("No tasks in queue. Returning to home.")
                    self.current_task = "home"
                    self.publish_goal("home")
            else:
                rospy.logwarn("No kitchen confirmation received. Returning to home.")
                self.current_task = "home"
                self.publish_goal("home")

        elif self.current_task.startswith("table"):
            rospy.loginfo(f"Reached {self.current_task}. Waiting for 30 seconds confirmation...")
            if self.wait_for_confirmation(self.current_task):  # Wait regardless of confirmation
                rospy.loginfo("Returning to kitchen to check for new orders.")
                if self.task_queue:
                    table = self.task_queue.pop(0)
                    self.current_task = table
                    self.publish_goal(table)
                else:
                    rospy.loginfo("No tasks in queue. Returning to home.")
                    self.current_task = "kitchen"
                    self.publish_goal("kitchen")
            else:
                rospy.logwarn("No table confirmation received.")
                if self.task_queue:
                    table = self.task_queue.pop(0)
                    self.current_task = table
                    self.publish_goal(table)
```

```
                    else:
                        rospy.loginfo("No tasks in queue. Returning to kitchen.")
                        self.current_task = "kitchen"
                        self.publish_goal("kitchen")

            elif self.current_task == "home":
                self.current_task = None
                if self.task_queue:
                    self.handle_kitchen_task()

    def cancel_callback(self, msg):
        cancel_order = msg.data.strip()
        if self.current_task == cancel_order:
            rospy.logwarn(f"Current task {cancel_order} canceled!")
            self.current_task = None
            self.publish_goal("home")
        elif cancel_order in self.task_queue:
            self.task_queue.remove(cancel_order)
            rospy.loginfo(f"Task {cancel_order} removed from the queue.")

    def publish_goal(self, location):
        if location not in self.goals:
            rospy.logerr(f"Unknown location: {location}")
            return False

        goal = PoseStamped()
        goal.header.frame_id = "map"
        goal.header.stamp = rospy.Time.now()
        goal.pose.position.x = self.goals[location]["x"]
        goal.pose.position.y = self.goals[location]["y"]
        quaternion = quaternion_from_euler(0, 0, self.goals[location]["yaw"])
        goal.pose.orientation.z = quaternion[2]
        goal.pose.orientation.w = quaternion[3]

        self.goal_pub.publish(goal)
        rospy.loginfo(f"Goal published to {location}")
        return True

    def handle_kitchen_task(self):
        if not self.current_task:
            rospy.loginfo("Navigating to kitchen for order pickup.")
            self.current_task = "kitchen"
            self.publish_goal("kitchen")

    def execute_tasks(self):
        rate = rospy.Rate(1)  # 1 Hz
        while not rospy.is_shutdown():
            if self.current_task is None and self.task_queue:
                self.handle_kitchen_task()
            rate.sleep()

if __name__ == "__main__":
    robot = ButlerRobot()
    robot.execute_tasks()
```