Sepehr Goshayeshi
MTRE 6300, Assignment 7
Due Dec 2, 2024

Youtube presentation: https://youtu.be/Pvn3qxD9gGo

To launch Gazebo with the robot arm and navigation bot seamlessly integrated together:
$ roslaunch panda_moveit_config demo_gazebo.launch

To run script for pick and place task:
$ rosrun franka_example_controllers pick_operation.py

Also, **nav.rviz** contains rviz configuration for navigation bot.

3.1 Custom Robot Design

The navigation bot's URDF is located in slam_pkg. This custom URDF model defines a wheeled navigation robot with a box-shaped base (base_link) and motorized left and right wheels, while the front caster wheel provides stability. The robot's dimensions emphasize a compact 0.6m x 0.3m base with a 0.1m height, balancing maneuverability and platform stability. Features include a laser scanner for environment sensing and a plugin for differential drive control. The robot design incorporates a container-like structure with fixed walls for load-bearing, potentially accommodating objects for manipulation. Load capacity is managed through lightweight components and balanced inertia properties to maintain stability during movement. The left and right wheels' configuration, combined with a 0.25m separation, ensures sufficient torque and smooth navigation. Key considerations include robust inertial calculations and fixed joints for structural integrity.

Also, I used a custom URDF models for Franka robot arm, and it involves defining its seven-axis arm configuration, precise joint limits, and dynamic properties for safe and accurate motion. The arm's compact design and high dexterity allow it to operate within a standard workspace, accommodating tasks requiring precision, such as object manipulation. Features include an end-effector interface for grippers or tools, integrated collision detection, and real-time torque control for adaptability. The design prioritizes lightweight construction, high payload capacity, and accurate modeling to ensure seamless integration with ROS for advanced manipulation tasks.

Source for the Franka arm: https://github.com/frankaemika/franka_ros

3.2 Gazebo Environment

The Gazebo simulation is set up by initializing the environment with essential elements, including a ground plane, multiple tables as obstacles, and objects like a stone for manipulation. The simulation launches two robots: the Franka Panda arm for object manipulation and an Explorer Bot for navigation, each configured with URDF models and their respective controllers. The Panda arm is spawned with a gripper, MoveIt integration, and precise placement on a table for manipulation tasks, while the Explorer Bot uses gmapping for SLAM and Move Base for navigation. Verification of interactions involves running the simulation in Gazebo with both robots active, observing the Panda's ability to manipulate objects, and monitoring the Explorer Bot's navigation using RViz. Navigation

tasks are enhanced by loading navigation parameters from YAML files, ensuring collision-free movement around obstacles and objects.

The `demo_gazebo.launch` file in the `panda_moveit_config` package was significantly edited to include namespacing and remapping, allowing seamless integration of both the navigation bot and the robot arm into the simulation. Namespacing isolates each robot's ROS topics, avoiding conflicts, while remapping ensures compatibility with shared resources, such as `/scan` for laser data and `/move_base` for navigation commands. These adjustments enable coordinated operation of the Panda arm for manipulation tasks and the Explorer Bot for navigation within a unified environment, ensuring smooth interactions and accurate simulation behavior.

3.3 Robotic Arm Integration

The robotic arm was configured to perform a pick-and-place operation using the `pick_operation.py` script located in the `scripts` folder of the `franka_example_controller` package within the `franka_ros` metapackage. The script uses MoveIt and ROS to define joint positions for precise arm movement and control the gripper for object manipulation. The arm executes a sequence of movements to approach the object (the rock), grasp it using the gripper, and move it to the designated location on the navigation bot.

The script initializes a ROS node, sets up `MoveGroupCommanders` for the arm and gripper, and uses predefined joint configurations for planning and execution. It includes gripper control via the `/franka_gripper/grasp/goal` topic to securely hold the rock. After grasping, the arm moves through intermediate positions to avoid collisions before placing the object on the navigation bot. The planning scene is reset at the start to ensure a clean setup for the operation, and the script handles resource cleanup upon completion.

4.1 Task 1: Pick and Place

The robotic arm was programmed to perform a pick-and-place task using the following approach:

1. **ROS MoveIt Pipeline**:
   The MoveIt framework was utilized for motion planning. MoveGroupCommander interfaces were set up for the Panda arm (panda_arm) and gripper (panda_hand), enabling precise joint trajectory planning. The script defined a series of joint positions to execute the arm's motion, ensuring safe and collision-free paths.
2. **End-Effector Calibration**:
   The end-effector's position was carefully calibrated to align with the object (the rock) for grasping. This involved defining specific joint positions and validating them in simulation to ensure the gripper's alignment with the object's location. Parameters such as grasp width, speed, force, and tolerances were adjusted to securely grasp the object and release it accurately onto the second robot's platform.

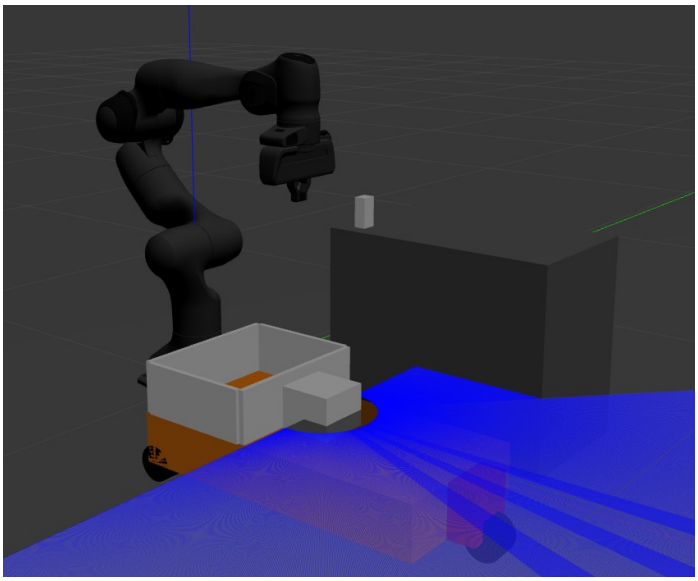
This process was implemented in the pick_operation.py script, with steps including gripper control, motion execution, and integration with the MoveIt planning scene for spatial awareness and precision.

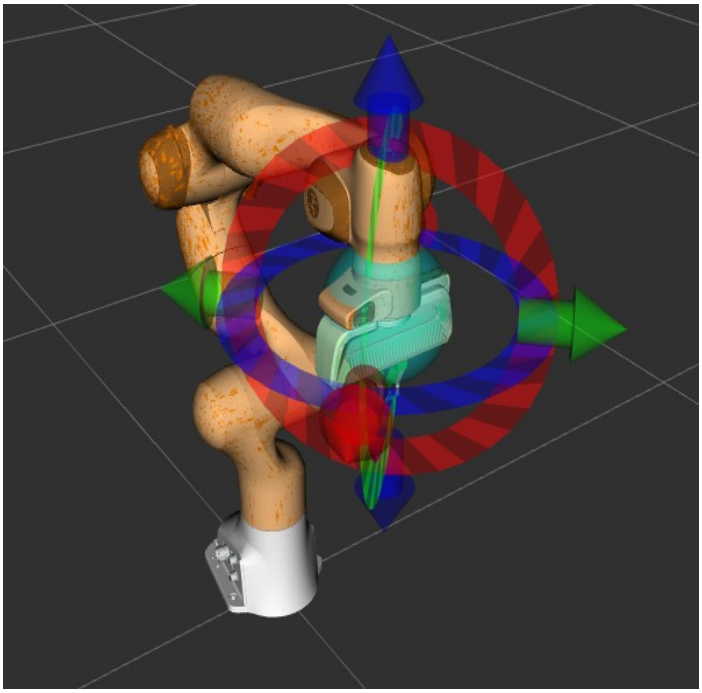
4.2 Task 2: Object Transportation

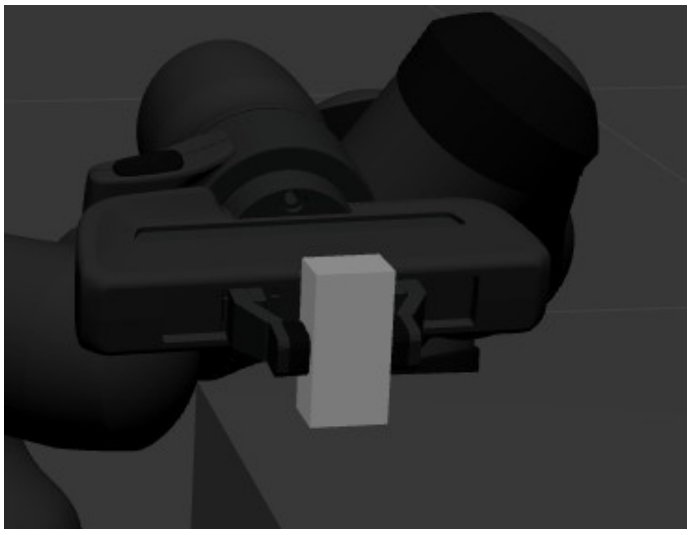
The second robot was programmed for path navigation leveraging global and local costmap yaml files. The global costmap handles long-term path planning with a 15x15m rolling window and a 2.5m inflation radius for obstacle safety. The local costmap provides short-term navigation with a 3x3m area and VoxelLayer for 3D obstacle representation. The trajectory_planner.yaml configures speed, acceleration, and goal tolerances for smooth and precise movement. The robot's driver controller translates these plans into wheel movements, enabling safe transport of the rock to the target location.

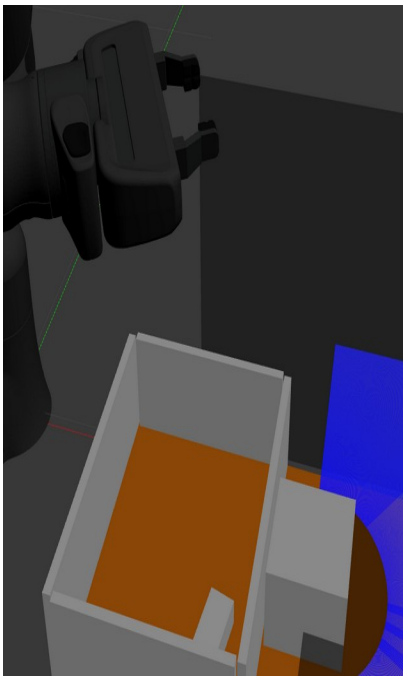
5.1 Simulation Results

- Present screenshots or diagrams of:
  - Robots in the Gazebo environment.
  - Robotic arm grasping and placing objects.

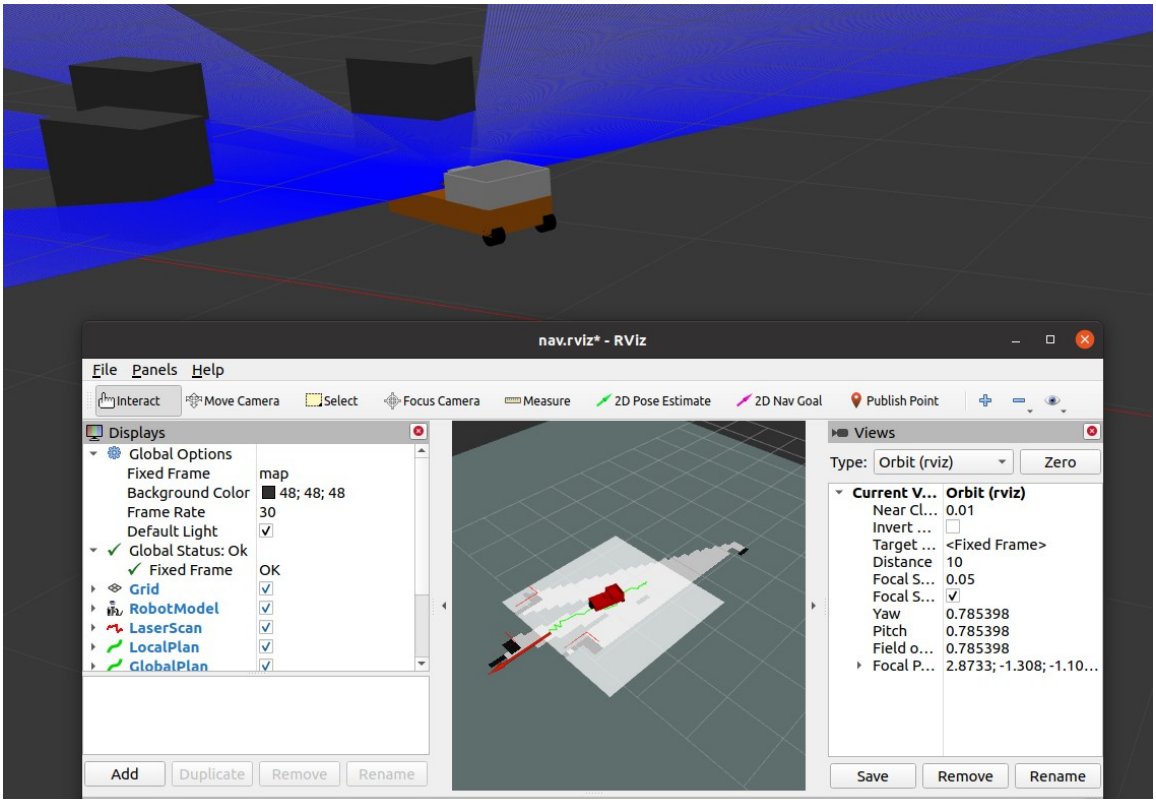








- The second robot transporting the object to the target location.



Performance Metrics

1. **Success Rate of Pick-and-Place Operations**:
   All steps of the pick-and-place operations completed successfully with the status SUCCEEDED in each execution. This indicates a **100% success rate** for the operation.

2. **Accuracy of Object Placement**:
   The success of each trajectory and execution implies the Panda arm consistently met its internal accuracy thresholds, as defined by the MoveIt framework. Precise placement of the object was achieved without errors, supported by collision-free planning and execution logs.

3. **Time Taken for Task Completion**:
   The operation took approximately **23.5 seconds**, from the first planning request (74.435s) to the final completion (97.934s). This includes:

   - **Planning times** of ~0.011–0.013 seconds for each trajectory.
   - **Execution times** ranging from ~8.4 seconds to ~12 seconds for each major trajectory.

These metrics demonstrate the Panda arm's efficiency and precision in performing the pick-and-place task.