Autonomous Trash Collection System with Mobile Manipulation

Group 13: Suryoun Lee, Ruoyu Xu, Swetha Vinay Argekar

Introduction

Problem

- Current cleaning robots lack efficient, robust, and scalable navigation.
- Limited adaptability in dynamic environments.

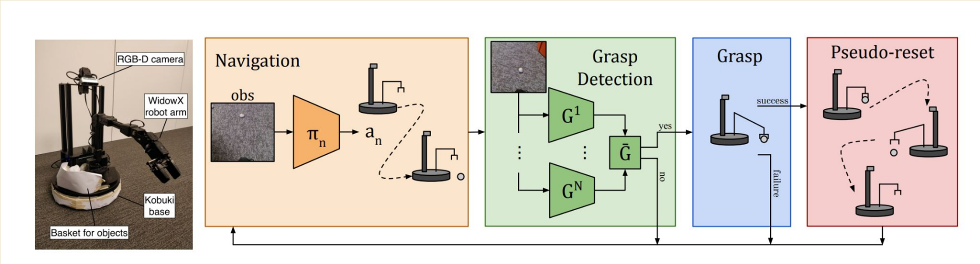
Our Approach

- Vision-guided navigation and manipulation.
- Reinforcement learning (RL) for training navigation and manipulation policies.

Goal

- Develop a scalable, efficient, and adaptable cleaning robot.
- Test state-of-the-art RL-based methods in real-world scenarios.

Related works



Fully Autonomous Real-World Reinforcement Learning with Applications to Mobile Manipulation.

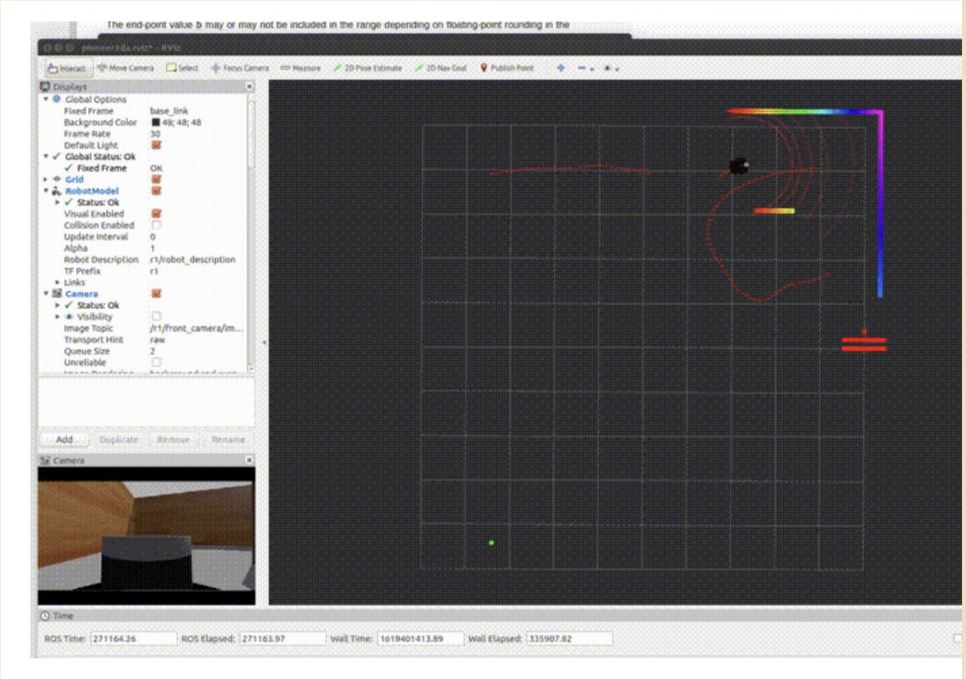
Deep reinforcement learning training separate navigation and grasping policy
Reward given to navigation policy when the robot grasps an object or initiates a grasp action

[Charles Sun, Jedrzej Orbik, Coline Devin, Brian Yang, Abhishek Gupta, Glen Berseth, and Sergey Levine. Conference on Robot Learning, 2021]

Related works

Goal-Driven Autonomous Exploration Through Deep Reinforcement Learning

Reinis Cimurs, Il Hong Suh, Jin Han Lee



Methods

Navigation Policy:

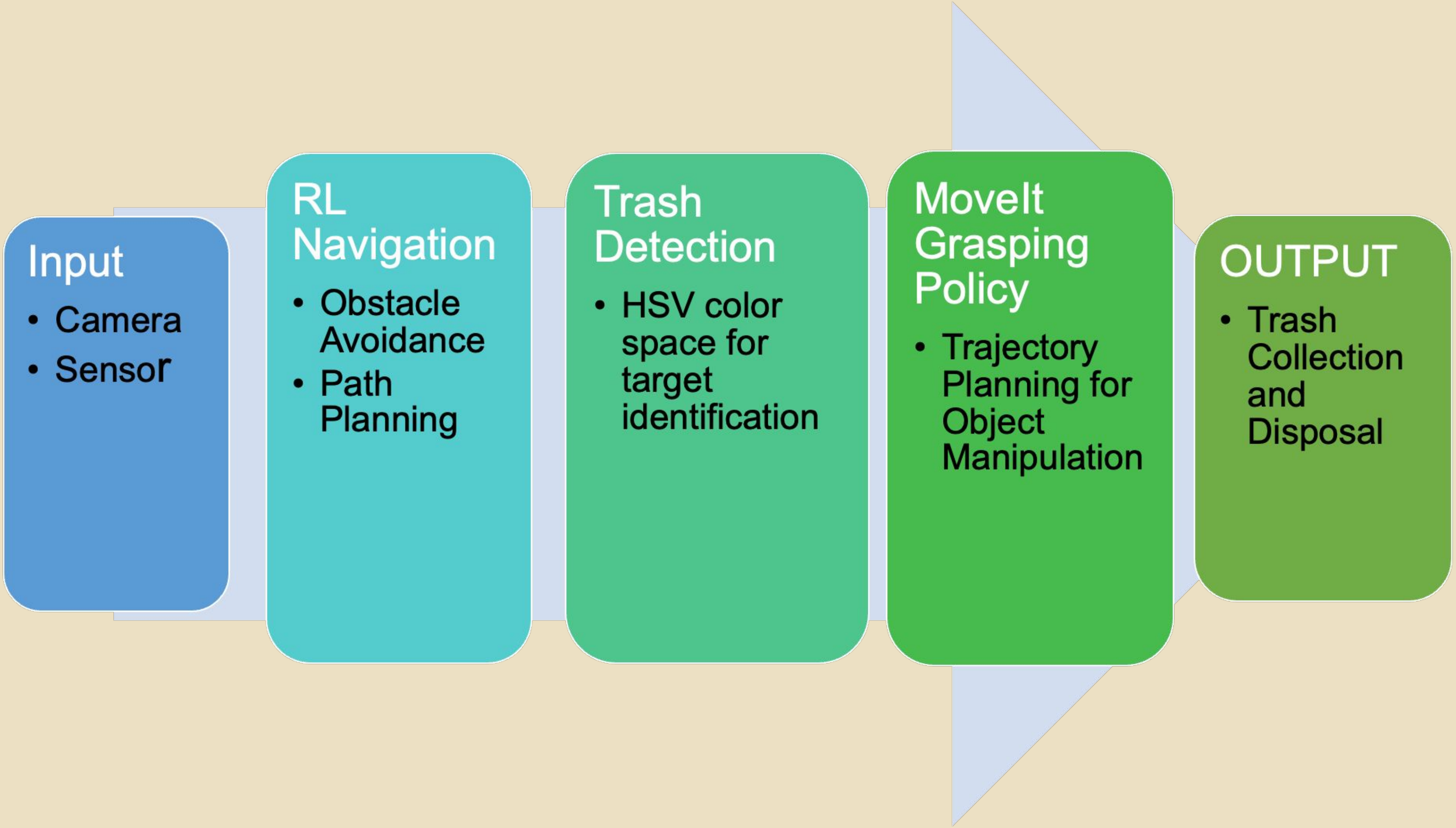
- RL-based policy trained for dynamic environments.
- Inputs: Simulated camera images for real-time decisions.
- Pre-trained policies used for robust navigation.

Grasping Policy:

- Traditional ROS and MoveIt-based pipeline for object manipulation.
- Reliable gripper control for successful grasping tasks.

Hybrid Approach:

- Combines RL for adaptability with traditional methods for reliability.
- Ensures flexibility and performance stability in varying scenarios.


Input
• Camera
• Sensor
RL Navigation
• Obstacle Avoidance
• Path Planning
Trash Detection
• HSV color space for target identification
MoveIt Grasping Policy
• Trajectory Planning for Object Manipulation
OUTPUT
• Trash Collection and Disposal

Environment

Gazebo Simulation Setup

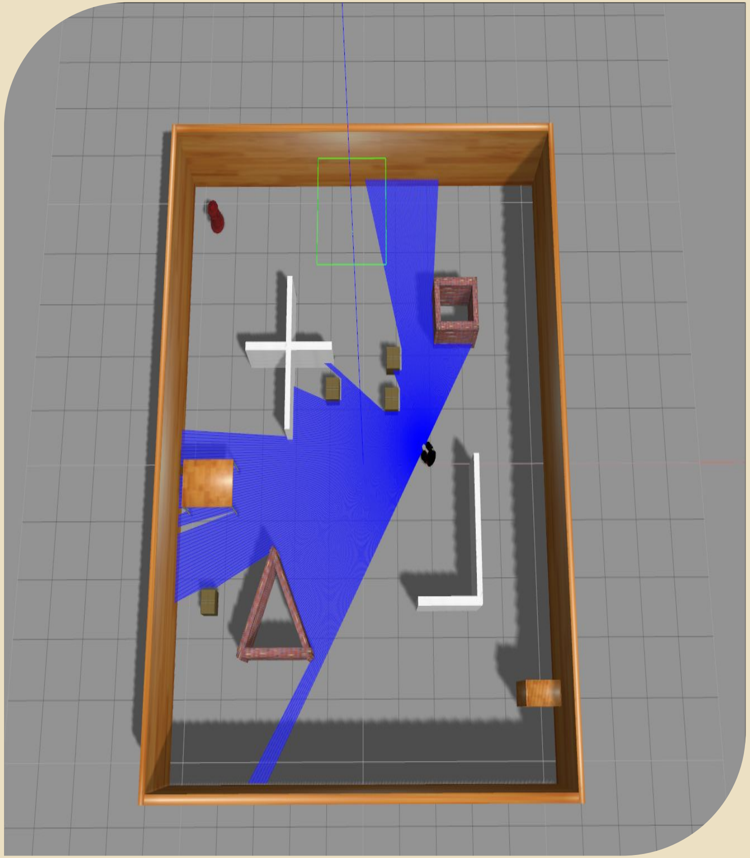
- **Platform:** Gazebo integrated with ROS for realistic indoor scenarios.
- **Features:**
  - Dynamic obstacles (tables, walls, narrow pathways).
  - Randomly placed trash items of various shapes and colors.

Environment Components

- **Camera Integration:**
  - Onboard camera captures real-time images for trash detection and navigation.
- **Gripper:**
  - Simulated gripper for object manipulation and trash collection.

Dynamic Scenarios

- Moving obstacles and repositioned trash to test adaptability and robustness.



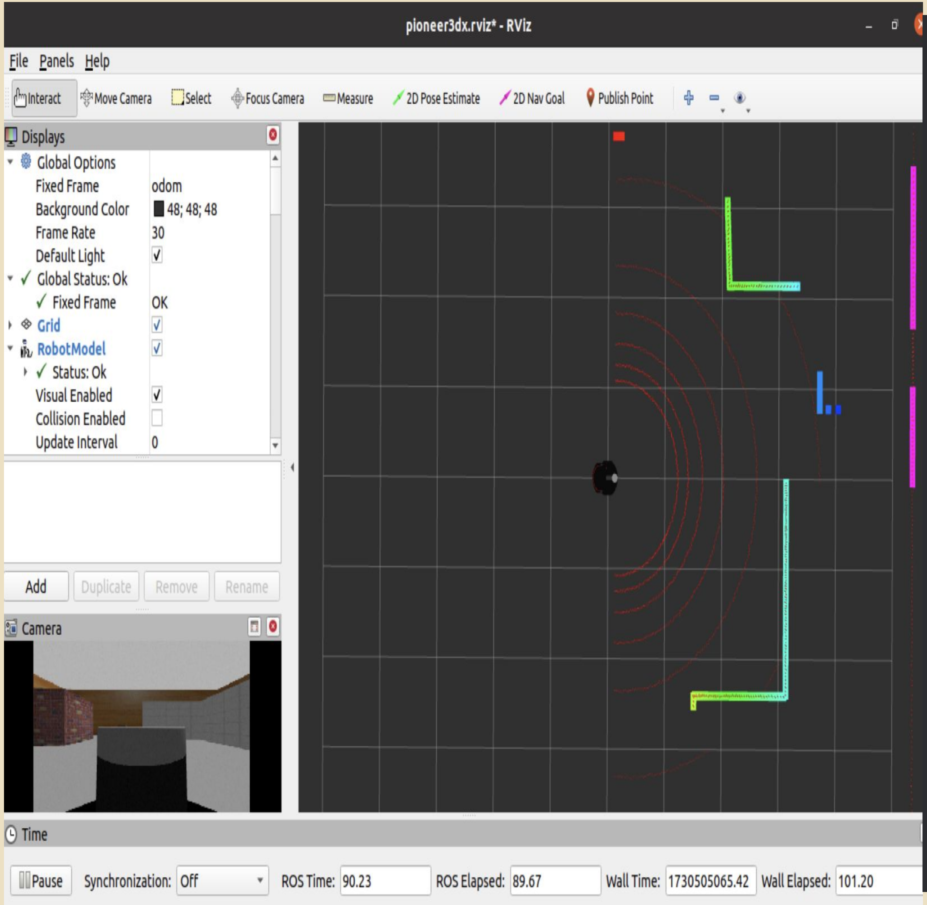
Progress

Achievements

- Successfully set up **Gazebo simulation environment** integrated with ROS.
- Implemented a **vision-guided navigation policy** with reinforcement learning.
- Developed a **MoveIt-based grasping pipeline** for object manipulation.

Progress

- **Navigation Policy**
  - RL policy demonstrates consistent performance in dynamic environments.
  - Initial success in avoiding obstacles and reaching target locations.
- **MoveIt Pipeline**
  - **Current Status:**
    The MoveIt-based grasping pipeline has been designed to handle predefined object grasping tasks reliably. However, the code is still in the early stages of development and not fully functional. Current implementation struggles with adapting to objects of varying sizes and positions.
  - **Challenges and Errors:**
    a. **Grasp Accuracy:** Difficulty in detecting the exact position of objects or inaccuracies in calculating position coordinates.
    b. **Error Messages:** Unexpected failures in path planning or gripper control, reported by MoveIt or ROS.
    c. **Robot Control:** The gripper fails to fine-tune its approach at the target, either missing the object or approaching at an incorrect angle.




```
def camera_callback(self, msg):
    # Convert ROS image message to OpenCV format
    cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
    # Detect trash in the image and update trash observation
    self.trash_observation = self.detect_trash(cv_image)

def detect_trash(self, image):
    # Convert to HSV for color-based trash detection
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_color = (30, 150, 50)  # Adjust these values based on trash color
    upper_color = (50, 255, 255)
    mask = cv2.inRange(hsv_image, lower_color, upper_color)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours:
        # Get the largest contour as trash item
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        center_x, center_y = x + w // 2, y + h // 2

        # Normalize coordinates
        rel_x = (center_x - image.shape[1] / 2) / (image.shape[1] / 2)
        rel_y = (center_y - image.shape[0] / 2) / (image.shape[0] / 2)
        return [rel_x, rel_y]
    else:
        return [0.0, 0.0]  # No trash detected
```

```
# Detect if the goal has been reached and give a large positive reward
if distance < GOAL_REACHED_DIST:
    target = True
    done = True
# Add Trash Manipulation logic here
if done:
    rospy.loginfo("Goal reached. Initiating trash manipulation...")
    manipulator = TrashManipulator()  # Instantiate the TrashManipulator
    manipulator.run()  # Perform trash collection task
    rospy.loginfo("Trash manipulation complete.")
robot_state = [distance, theta, action[0], action[1]]
state = np.append(laser_state, self.trash_observation)
reward = self.get_reward(target, collision, action, min_laser)
return state, reward, done, target
```

Code added to previous source code -env setting

Code we added to gripper - grasp trash manipulation

```
import rospy
from moveit_commander import MoveGroupCommander, roscpp_initialize, roscpp_shutdown
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

class TrashManipulator:
    def __init__(self):
        # Initialize ROS node
        rospy.init_node("trash_manipulator", anonymous=True)

        # Initialize MoveIt
        roscpp_initialize(sys.argv)
        self.arm_group = MoveGroupCommander("arm")
        self.gripper_group = MoveGroupCommander("gripper")

        # Camera setup
        self.bridge = CvBridge()
        rospy.Subscriber("/camera/rgb/image_raw", Image, self.camera_callback)
        self.trash_position = None

    def camera_callback(self, msg):
        """Detect trash using the camera feed."""
        try:
            cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            hsv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
            lower_color = (30, 150, 50)  # Adjust based on trash color
            upper_color = (50, 255, 255)
            mask = cv2.inRange(hsv_image, lower_color, upper_color)
            contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

            if contours:
                largest_contour = max(contours, key=cv2.contourArea)
                x, y, w, h = cv2.boundingRect(largest_contour)
                center_x, center_y = x + w // 2, y + h // 2
                self.trash_position = (center_x, center_y)
        except Exception as e:
            rospy.logerr(f"Error in camera_callback: {e}")

    def move_to_trash(self):
        """Move the arm to the detected trash position."""
        if self.trash_position:
            rospy.loginfo("Moving to trash position...")
            target_pose = self.arm_group.get_current_pose().pose
            target_pose.position.x = self.trash_position[0] / 100.0
            target_pose.position.y = self.trash_position[1] / 100.0
            target_pose.position.z = 0.1  # Adjust height
            self.arm_group.set_pose_target(target_pose)
            success = self.arm_group.go(wait=True)

            if success:
                rospy.loginfo("Arm reached the target. Grasping...")
                self.grasp_trash()

    def grasp_trash(self):
        """Grasp the detected trash."""
        self.gripper_group.set_named_target("closed")
        self.gripper_group.go(wait=True)

    def release_trash(self):
        """Release the trash into the bin."""
        self.gripper_group.set_named_target("open")
        self.gripper_group.go(wait=True)

    def run(self):
        """Main loop."""
        rospy.loginfo("Starting trash manipulation...")
        while not rospy.is_shutdown():
            if self.trash_position:
                self.move_to_trash()
                self.grasp_trash()
                rospy.sleep(2)  # Simulate moving trash
                self.release_trash()
                break
            rospy.sleep(1)  # Wait for trash to be detected


if __name__ == "__main__":
    try:
        manipulator = TrashManipulator()
        manipulator.run()
    except rospy.ROSInterruptException:
        pass
    finally:
        roscpp_shutdown()
```

Planning

Refine Navigation Policy:

- Continue to improve the RL-based navigation policy for more efficient pathfinding and robust collision avoidance in dynamic environments.
- Address current challenges in handling complex obstacle arrangements.

Enhance Grasping Mechanism:

- Debug and optimize the MoveIt pipeline to improve grasping success rates, focusing on varied object shapes, sizes, and positions.
- Integrate feedback from testing to fine-tune grasping parameters and improve adaptability.

Integrate System Components:

- Establish seamless interaction between the navigation policy and grasping mechanism to ensure coordinated operation.
- Address synchronization challenges between RL-based navigation and MoveIt-based manipulation.

Evaluate Performance:

- Conduct final trials in the simulation environment to measure key performance metrics, including:
    1. Trash detection accuracy.
    2. Grasp success rate.
    3. Navigation efficiency and overall task completion time.
- Compare performance across different object types and environmental conditions to validate robustness.

REFERENCE

[1] Charles Sun, Jedrzej Orbik, Coline Devin, Brian Yang, Abhishek Gupta, Glen Berseth, and Sergey Levine. Fully Autonomous Real-World Reinforcement Learning with Applications to Mobile Manipulation. Conference on Robot Learning, 2021.

[2] Max Bajracharya, James Borders, Richard Cheng, Dan Helmick, Lukas Kaul, Dan Kruse, John Leichty, Jeremy Ma, Carolyn Matl, Frank Michel, Chavdar Papazov, Josh Petersen, Krishna Shankar, and Mark Tjersland. Demonstrating Mobile Manipulation in the Wild: A Metrics- Driven Approach. Robotics: Science and Systems XIX, Robotics: Science and Systems Foundation, July 2023.

[3] Daniel Honerkamp, Tim Welschehold, and Abhinav Valada. Learning Kinematic Feasibility for Mobile Manipulation through Deep Reinforce- ment Learning. IEEE Robotics and Automation Letters, 6: 6289–6296, 2021.

[4] Reinis Cimurs. DRL-robot-navigation, GitHub. Available: https://github.com/reiniscimurs/DRL-robot-navigation.

Thank you