An Experiment on Robotic Navigation and Object Manipulation Tasks in Simulated Room Environment

Wang Wenzhao
National University of Singapore
A0276544W
e1132277@u.nus.edu

Wei Xiaochen
National University of Singapore
A0279556J
e1143369@u.nus.edu

Zhou Xiangyu
National University of Singapore
A0291289N
e1331097@u.nus.edu

Abstract

This report documents the default coursework project for CS5478, carried out by a group of three students. The project focuses on implementing navigation and object manipulation tasks for a mobile robot, in a simulation environment based on the Pybullet framework. The primary goal of navigation is to reach a pre-defined target location while avoiding obstacles. [Method for navigation] algorithems are explored for path planning. The other task performed is to pick and place a mug with a robotic arm attached on the mobile robot. A [vision model] is implemented to get mug position from visual and depth cameras attached on the robot. The procedure for grasping and placing the mug is set deterministically based on the environment and robot arm configurations, aided by utilities provided. The code is placed in a Github repository: `https://github.com/GeorgeZXY/CS5478Project`.

1 Introduction

Robotics has become a promising area of study in recent years. Robots could offer solutions to a wide range of applications, from autonomous driving to automation of household chores. Two fundamental aspects in robotics are navigation, where robots move efficiently through a space, and manipulation, which involves interacting with objects to perform specific tasks. These capabilities are essential for robots to be functional and useful in various scenarios and environments.

As part of the coursework for CS5478 Intelligent Robots: Algorithms and Systems in academic year 24/25, this project explores the implementation of navigation and manipulation tasks using a mobile robot in a simulated environment. The navigation task involves moving the robot from a start position to a goal while avoiding obstacles, requiring efficient path-planning techniques. After reaching the goal area, it needs to perform a manipulation task using the multi-linkage arm and gripper it to pick up a mug and place it in a drawer. The drawer is initialized with a fixed position while the mug is placed randomly within an small area that is on top of the drawer.

The project adopts a structured approach by leveraging predefined locations and utility functions for object localization and robot actions, simplifying the challenges within the controlled simulation environment. The navigation is achieved by path planning using A* algorithm from obstacle locations based on the environment. The pick and place of the mug is performed using a fixed procedure with parameters locating the mug object.

This report presents methods applied to implement the specified tasks, followed by analysis on the results, with discussion of challenges encountered and possible improvements for future works.

2 Related Works

Recent advancements in robotics have enabled significant progress in the domains of manipulation, navigation, and sensor integration. Several studies have explored path planning techniques for autonomous robots in dynamic environments, where real-time obstacle detection and avoidance are critical. In [1], the authors propose a framework for real-time path planning based on LiDAR data, enabling robots to detect and avoid obstacles dynamically while maintaining a feasible trajectory. Similarly, [2] integrates vision and LiDAR data for simultaneous localization and mapping (SLAM), enhancing the robot's ability to navigate complex, unstructured environments.

In the area of manipulation, much work has been done to improve the grasping and manipulation of objects in both simulated and real environments. [3] describes an advanced grasping technique using force and vision feedback to adjust the gripper's motion and apply appropriate forces to maintain an object's stability. Another method focuses on inverse kinematics and trajectory planning to improve robotic arm control for pick-and-place tasks [4]. In their approach, [5] use machine learning techniques to predict the optimal trajectories for robotic arms based on sensor feedback and task-specific goals. However, as seen in the current implementation, limitations arise from simplified simulation models, especially with regard to object grasping and physical interaction modeling. For instance, the reliance on constraint-based methods, such as those implemented in [6], results in non-realistic behaviors that fail to capture nuanced aspects of contact forces and friction in robotic interactions.

Moreover, several studies have investigated the integration of vision-based systems for object detection and task planning in robots. In [7], a top-down camera view is employed to assist a robotic arm in picking objects from a cluttered environment. This approach allows the robot to estimate the position of objects and plan its motion accordingly. Vision-based systems have also been shown to be effective in dynamic environments, with [8] demonstrating the use of vision-based feedback to adapt to moving obstacles in real-time path planning. However, limitations exist when incorporating vision with inverse kinematics and trajectory planning, as noted in [9], where the lack of accurate 3D understanding can hinder the robot's ability to complete tasks such as pick-and-place with high precision.

These works highlight key challenges in robotics, such as ensuring reliable object manipulation, improving dynamic path planning, and leveraging vision systems effectively. Our approach, similar to existing frameworks, aims to enhance robot autonomy by integrating these capabilities, with a focus on improving the accuracy and realism of grasping techniques, as well as refining path planning methods for dynamic and unstructured environments.

3 Methodology

3.1 Setup

This project is conducted in a Pybullet simulation environment. The environment setup is provided from course material. This environment mimics an simplified household setting with multiple rooms, such as bedroom, living room and kitchen. Various objects like bed, drawer, fridge etc. are placed in the rooms. The robot used is a mobile robot with an robotic arm attached to it. The robot's base provides mobility on the floor with forward, backward and turning functionalities. The robotic arm's first link is a long vertical link with translational joint to adjust height. Second link is a horizontal link with translational joint to perform stretch. This stretch direction is perpendicular to the base's forward motion direction. The end effector is a 2-finger gripper and is mounted on 2 rotational links that serve for roll and pitch. The task involves 2 parts. First, the robot is initialized in the kitchen and it should navigate to the bedroom and reach goal position beside bed and in front of drawer. Second, the robot would pick up a mug that is placed on the drawer and put it in an open cabinet of the drawer.

3.2 Navigation

By integrating real-time obstacle detection (through simulated data), efficient path planning (using the A* algorithm on a grid-based map), and precise robot control (via a PID controller for motion execution), the system ensures that the robot can reach its goal safely and efficiently. The system takes

into account the physical dimensions of the robot, avoiding collisions by adhering to inflated obstacle boundaries, and dynamically adjusts the path based on the changing environmental conditions.

3.3 Real-time Obstacle Detection Using Sensor Data

3.3.1 Setting Up the Simulation Environment with PyBullet

The simulation environment is initialized using PyBullet, a physics engine that enables realistic interaction between the robot and objects in the environment. The `init_scene` function in `stretch.py` sets up the environment by loading various objects, such as tables, walls, cabinets, and bottles. Each object is assigned a unique identifier (`object_id`), which is crucial for tracking and managing these objects within the simulation.

3.3.2 Obtaining Object Bounding Boxes (AABBs)

Obstacle detection is primarily handled by retrieving the axis-aligned bounding boxes (AABBs) of each object using PyBullet's `getAABB` function. The `map.py` file contains functions for extracting these bounding boxes:

- `get_bbox(id)`: This function retrieves the AABB for each link of a given object, identified by its id. It iterates over all link IDs (including the base link) and collects the corresponding AABBs.
- `get_navigation_map(robot, plot=True)`: This function aggregates the AABBs of all obstacles in the environment, excluding the robot and the floor. It iterates over all object IDs, using `get_bbox` to obtain their bounding boxes, which are stored in the list `ob_bboxs`. If plotting is enabled, it visualizes these obstacles on a 2D map using Matplotlib, aiding in debugging and visualization.

3.3.3 Constructing a Static Occupancy Map

A static occupancy map represents the navigable space and obstacles in the environment in a grid format. The construction process is as follows:

- **Grid Resolution and Size**: The environment is discretized into a 2D grid based on a specified resolution (e.g., 0.05 meters per grid cell). The grid covers the dimensions of the environment (e.g., from -`x_max` to `x_max` and -`y_max` to `y_max`).
- **Inflating Obstacles**: To account for the robot's size and ensure safe navigation, the bounding boxes of obstacles are inflated by a certain number of grid cells (`inflate_cells`). This creates a buffer zone around each obstacle, preventing the robot from planning a path too close to them.
- **Marking Occupied Cells**: For each inflated obstacle bounding box, the corresponding grid cells in the `static_map` array are marked as occupied (value = 1). The map effectively distinguishes between free space (0) and obstacles (1).
- **Validation**: The start and goal positions are converted to grid coordinates, ensuring they lie within the environment's boundaries and are not located on obstacles.

3.4 Path Planning Using the A* Algorithm

3.4.1 Grid-Based Graph Representation

The A* planner operates on a graphical representation of the environment. The graph construction process is as follows:

- **Creating a 2D Grid Graph**: A 2D grid graph is created using NetworkX's `grid_2d_graph`, where each node represents a grid cell in the occupancy map.
- **Removing Obstacle Nodes**: Nodes corresponding to occupied grid cells (obstacles) are removed from the graph, effectively marking them as non-traversable areas.

3.4.2 Implementing the A* Algorithm

In environments with narrow gaps, such as that between a bed and cabinet, traditional path planning algorithms like PID control, which assume the robot moves in a straight line along its current heading, may not be sufficient. This is because the robot may become obstructed if it cannot change its orientation to navigate through such confined spaces. To address this, we propose a two-phase A* path planning approach, specifically designed to guide the robot through narrow gaps by adjusting its orientation during navigation.

- **Path Planning in Two Phases**: The path planning process is divided into two distinct phases:

  - **Phase 1: Start to Narrow Gap**: In this phase, the robot follows the A* algorithm to navigate from the start position to a point just before the narrow gap. At this stage, the robot continues in its forward direction, but upon reaching the gap, it must transition to a sideward orientation (i.e., perpendicular to its previous heading) in order to pass through the gap.

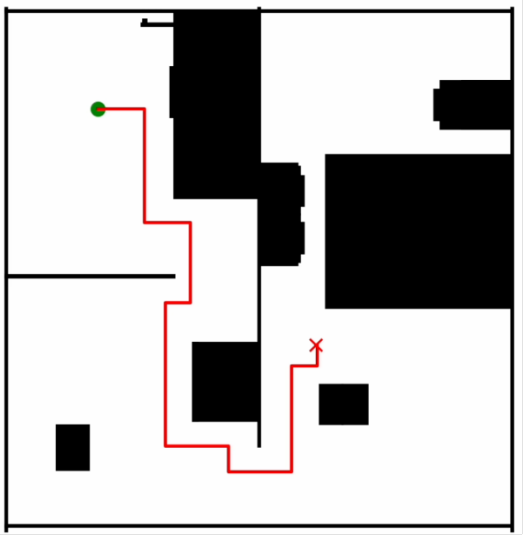

Figure 1: Phase 1

  - **Phase 2: Sideways Navigation through Gap to Goal**: Once the robot reaches the intermediate position (the green-marked point) just before the gap, its orientation is adjusted to a sideward direction. The robot then continues to move through the gap in this new orientation, following the A* path until it reaches the goal.

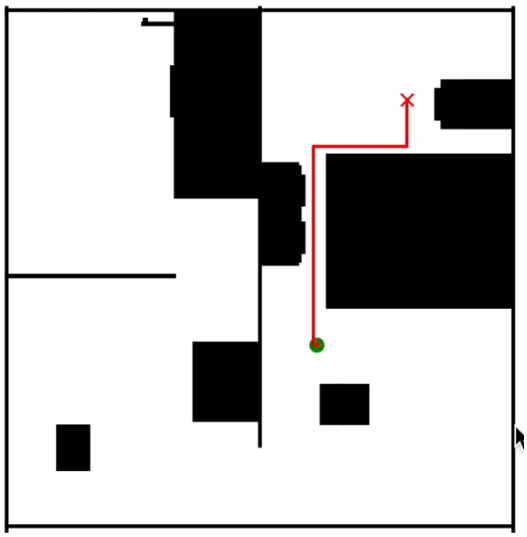

Figure 2: Phase 2

- **Algorithm Implementation**:
  - **Initialization**: The `AStarPlanner` class is initialized with parameters such as robot size, obstacle boundaries, grid resolution, and visualization preferences. The robot's path is planned in two stages as described above.
  - **Path Finding**: In the first phase, the A* algorithm is used to find the shortest path from the start position to the point just before the narrow gap. Upon reaching this point, the robot is instructed to switch to a sideward orientation. In the second phase, the algorithm computes a new path, now assuming that the robot is side-facing, from the narrow gap to the goal position.
  - **Orientation Adjustment**: At the intermediate point (the green marker), when the robot transitions from the first phase to the second, it adjusts its orientation from forward-facing to side-facing. This transition is critical for allowing the robot to traverse through the narrow gap. The robot continues along the new path in this adjusted orientation.
  - **Visualization**: If enabled, the planner visualizes the environment, including the static map, start and goal positions, narrow gap, and the planned path. The intermediate point (where the robot switches orientation) is marked clearly in the visualization, which assists in debugging and verifying the path planning process.

This two-phase path planning approach ensures that the robot can effectively navigate through tight spaces by dynamically adjusting its orientation, overcoming limitations posed by traditional PID-based control methods.

3.4.3 Ensuring Efficient Navigation

The A* algorithm ensures the robot finds the most efficient path by simultaneously considering the cost to reach a node and the estimated cost to the goal. By inflating obstacles and using a fine grid resolution, the planner takes into account the robot's size and navigation constraints, preventing collisions and optimizing path length.

3.5 Obstacle-Avoiding Navigation

The `RobotController` class in `move_robot.py` manages the robot's movement along the planned path:

- **Initialization**: The controller is initialized with the robot's initial pose, the planned path to follow, and parameters such as the PID controller for smooth motion.
- **Navigation to Waypoints**:

- **Waypoint Iteration**: The robot navigates sequentially to each waypoint along the planned path.
- **Yaw Adjustment**: Before moving to a waypoint, the robot adjusts its yaw to face the target direction, ensuring the movement aligns with the path.
- **Motion Execution**: The robot uses a simplified PID controller to compute necessary adjustments based on the distance to the target, with the `sim_action` function incrementally updating the robot's position to ensure smooth and controlled motion.
- **Collision Avoidance**: By following the path generated by A*, the robot inherently avoids obstacles. Additionally, the path planning process accounts for the robot's size, maintaining a safe distance from obstacles.

3.6 Manipulation

The initialization of the mug position is within a 0.15 by 0.2 area at the top surface of the drawer. The drawer is initialized with the first cabinet open. Hence, the robot does not have to open the drawer before picking the mug.

A deterministic procedure could be derived from the environment setup and the task requirements. First, the robotic arm should raise above the mug's height to get ready for the picking task and also avoid collisions with the mug or drawer. From observing the setup, the drawer's back side is against wall and right side is next to the bed. Therefore, the robot may only approach the mug from the front or left side of drawer. If the robot move towards the mug in the front side of the drawer, it may collide with the open cabinet and close it, so robot needs to keep a distance from the drawer at the front side. Hence, it is more feasible and stable to approach the drawer from its left. Even if the robot collide with the drawer, there is minimal impact on the pick and place operation. Since the drawer's left side is along X direction, the robot could move along Y direction until its X position is beside the drawer's left side, assuming that the drawer location and dimensions are known. Then, the robot should turn and move along X direction to get to the left side of drawer and move to Y position of the mug. The robot should be facing X direction at this moment and is to the left, or Y direction of the mug. Since the stretch direction of the robot is perpendicular to its motion direction, the gripper could reach the mug from the left side of the drawer by adjusting the stretch. The gripper position is adjusted to be above the mug's position and the opening space of the gripper is towards the edge of the mug, aided by the `get_mug_pose()` method from the provided tools. Then the gripper is lowered such that the end effector link is almost touching the top edge of the cup to grasp the mug. The point of contact for grasping is fine tuned to be the bottom side edge of the mug when viewing from top, with one gripper finger inside the mug and the other outside. The gripping action is performed using the `attach()` function from tools. Although this method is using a constraint to simulate gripping instead of applying physical friction on the object to really grasp it. This simplification is sufficient in the fixed simulation environment and also makes it more practical to implement. After picking up the mug, the robot could simply move in -X direction until the mug is above the open drawer. Lower the gripper and releasing it to place the mug in the drawer. The task is completed.

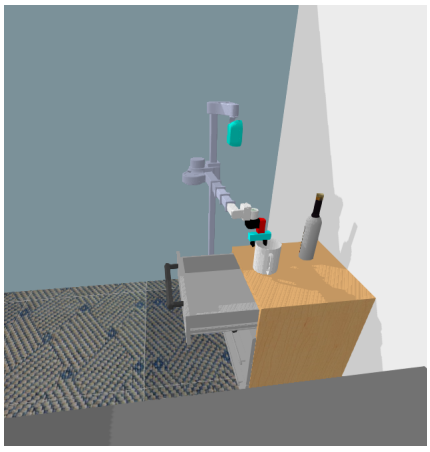

Figure 3: Picking the mug

4 Discussion

4.1 Navigation

The navigation task was implemented successfully, but several iterations were required to address various challenges encountered. The first challenge was selecting an appropriate path-planning method. Initially, we explored with Dijkstra's algorithm but ultimately adopted the A* algorithm for its efficiency and ease of implementation. The second challenge was obstacle avoidance during path planning. Early attempts saw the robot colliding with obstacles. Also the narrow space formed between the bed and the cabinet on the wall opposite the bed proved particularly problematic. These issues were resolved through fine-tuning the A* algorithm and adjusting the robot's orientation when navigating tight spaces. The robot's direction is reversed before passing the bed to avoid the robotic arm colliding with the cabinet. Lastly, the robot's motion was initially very slow. This was addressed by implementing customized motion control functions and incorporating a simplified PID controller to improve speed and accuracy in motion controls. These adjustments ensured successful task completion with enhanced performance.

4.2 Manipulation

As for the manipulation task. The current solution using fixed procedure works well with the fixed environment. However, the limitation on this method is that this analysis of the trajectory and action planning is done manually. Hence, it is not generalizable to random or real-world environments. For instance, it will not work if the drawer is turned to face sideways or if the drawer is moved to the other side of the bed. Another difficulty is the simulation of gripping effect. Current implementation relies on the `attach()` function from the tools. As this function only sets a constraint to fix the relative position of the end-effector link and the cup, it demonstrated random behavior on the orientation of the mug. In addition, as the default constraint is applied at the center of each object, the mug would crash into the gripper. The reason is the origin point of the mug is inside its body at its center of mass and the gripper opening dimension is not wide enough to cover the outer diameter of the mug. Also, it seems the axis definition of the mug's urdf file is not aligned with its orientation placed in the simulation, as it is rotated during the initialization process. As a result of above factors, the mug rotates randomly when attached to the end effector. After the `detach()` method is called, the mug gets stuck by the end effector link and gripper fingers and the place operation could not be completed. The `attach()` method is improved to include constraints on the mug's orientation and set the correct relative positions between the mug and the end effector, such that the mug's top edge is slightly below the end effector link and is in between the two gripper fingers. With the improved `attach()` function, the simulation animation looks smooth and the mug falls naturally due to gravity after calling `detach()` function.

5 Potential Improvements

The current implementation of navigation uses a static map obtained from the simulation. This approach assumes known environment but in a real-world dynamic environment, real-time obstacle detection is crucial. To address this challenge, the path planning system needs to continuously update the occupancy map based on sensor data, such as LiDAR and cameras. This means the robot must be able to detect changes in the environment and re-plan its path based on newly detected or moved obstacles. For example, when sensors detect a new obstacle, the path planner will adjust the original path in real time to avoid collisions and ensure safe navigation. To achieve this, the algorithm must handle sensor data noise, make reasonable inferences about dynamic changes in obstacles, and generate new feasible paths. As technology advances, robots will be able to adapt to real-time changes in more complex and uncertain environments, enhancing their flexibility and efficiency in dynamic settings.

There are two aspects in the manipulation task that could be enhanced. First, inverse kinematics and trajectory planning should be implemented for the robot to autonomously wield its arm and gripper to reach target locations. For this simple project, the current deterministic procedure practical, due to its easy implementation within short time schedule and relatively stable performance. However, as it is not transferrable to apply on any other tasks, proper trajectory planning and inverse kinematics should be deployed. It could be conducted analytically or using machine learning models [10]. Second,

more realistic simulation of gripping mechanism could be introduced by implementing gripper finger control with contact and friction forces. Current method of grasping relies heavily on the constraint functionality from pybullet, which does not reflect real interaction between objects. This solution also makes too much simplification on the gripper action control. There is no control on the motions of the gripper fingers. A simulation that captures the physical mechanism of closing the gripper fingers and applying contact forces to create friction with the object would better represent the physical dynamics of grasping. This simulation may also improve the robot performance. For example, the robot could detect distance between the fingers or the torque on the gripper motor to examine if any object is grasped, if an object is too heavy, the simulation could also simulate the fall of the object.

Lastly, more advanced features could be incorporated from the vision capabilities. Current robot control algorithms does not fully utilize the vision abilities provided by the cameras mounted on the robot. The visual processing could allow the robot to gain more information about the environment. The robot may use camera input to detect and avoid obstacles in navigation tasks. Also, the robot could make automatic action planning if the image of the environment could be perceived. For example, if the robot is able to see the drawer is closed, it could plan actions to open the drawer first and pick mug afterwards.

Acknowledgments

We would like to express our sincere gratitude to Professor Shao Lin and all teaching staff of the course CS5478 Intelligent Robots: Algorithms and Systems, for conducting the course and the coursework project. This coursework project has been a rewarding learning experience, offering valuable hands-on experience and practical knowledge beyond the lecture content. Though the team have little prior experience in the robotic field and the solution proposed is still preliminary, we really enjoyed this learning opportunity. The process allowed us to gain deeper insights on robotics. We are truly grateful for the opportunity to undertake this project.

References

[1] Sabiha, A. D., Kamel, M. A., Said, E., & Hussein, W. M. (2022). Real-time path planning for autonomous vehicle based on teaching–learning-based optimization. *Intelligent Service Robotics*, 15(3), 381-398.

[2] Debeunne, C., & Vivet, D. (2020). A review of visual-LiDAR fusion based simultaneous localization and mapping. *Sensors*, 20(7), 2068.

[3] Kouskouridas, R., Amanatiadis, A., & Gasteratos, A. (2011, April). Guiding a robotic gripper by visual feedback for object manipulation tasks. In 2011 IEEE International Conference on Mechatronics (pp. 433-438). IEEE.

[4] Kaur, M., Yanumula, V. K., & Sondhi, S. (2024). Trajectory planning and inverse kinematics solution of Kuka robot using COA along with pick and place application. *Intelligent Service Robotics*, 17(2), 289-302.

[5] Ying, K. C., Pourhejazy, P., Cheng, C. Y., & Cai, Z. Y. (2021). Deep learning-based optimization for motion planning of dual-arm assembly robots. *Computers & Industrial Engineering*, 160, 107603.

[6] Leeper, A., Chan, S., Hsiao, K., Ciocarlie, M., & Salisbury, K. (2012, March). Constraint-based haptic rendering of point data for teleoperated robot grasping. In 2012 IEEE Haptics Symposium (HAPTICS) (pp. 377-383). IEEE.

[7] Muralikumar, S. (2022). Vision-Based Control Using Object Detection and Depth Estimation for Robotic Pick and Place Tasks in Construction Applications (Master's thesis, Arizona State University).

[8] Yao, P., Wang, H., & Su, Z. (2015). Real-time path planning of unmanned aerial vehicle for target tracking and obstacle avoidance in complex dynamic environment. *Aerospace Science and Technology*, 47, 269-279.

[9] Ahuactzin, J. M., & Gupta, K. K. (1999). The kinematic roadmap: A motion planning based global approach for inverse kinematics of redundant robots. *IEEE Transactions on Robotics and Automation*, 15(4), 653-669.

[10] Zhao, D., Ding, Z., Li, W., Zhao, S., & Du, Y. (2023). Robotic Arm Trajectory Planning Method Using Deep Deterministic Policy Gradient With Hierarchical Memory Structure. *IEEE Access*, 11, 140801-140814. doi: 10.1109/ACCESS.2023.3340684.