Final Project - Lab Report
AI and Robotics (236609)

Elashkin Andrew
eandrey@technion.ac.il
CS Department
Technion - Israel Institute of Technology

March 27, 2023

1 Introduction

In this project we are tasked with planning the motion of a robot in a fully mapped room containing workstations, where each workstation has a set of activities that can be performed from specific configurations. The robot must complete an unordered set of activity sequences, each associated with a reward. The goal is to find a plan that maximizes the net reward (reward minus cost), taking into consideration the execution time and the cost associated with each motion and activity. The cost of a path is determined by the time it takes to follow the path and the cost map, which accounts for the robot's proximity to walls and obstacles.
The algorithm developed in this assignment has numerous real-life applications across various domains. In industrial settings, it can be employed to optimize the efficiency of robots in assembly lines and warehouses, where they must navigate complex environments to perform tasks at specific workstations. In agriculture, the algorithm can be applied to autonomous vehicles that need to perform various tasks such as planting, harvesting, or monitoring crop health while navigating around obstacles. Additionally, the algorithm can benefit search and rescue operations, where robots need to efficiently traverse unknown environments while minimizing risk and time spent in dangerous areas.

2 Background

Below is the required background for our setup:

- Path planning: the process of determining the best path for the robot to follow to achieve a desired goal. In this task, path planning can be used to determine the most efficient path for the robot to follow to examine the objects.
- Branch and bound: is a method for solving optimization problems by breaking them down into smaller sub-problems and using a bounding function to eliminate sub-problems that cannot contain the optimal solution.

- Itertools: a Python module that provides a collection of fast, memory-efficient tools for working with iterators.

3 Setup

In this section we will give a brief explanation of our code.
Upon the turtlebot client creation, we subscribe to several channels, similar to the code provided in the tutorials: '/initialpose' to set the initial position, '/move_base/global_costmap/costmap_updates' to update the costmap and '/amcl_pose' to update our current position relative to the map. We also create the 'move_base' client to be able to set a goal command for the robot.

Our algorithm can be split in five main parts:

1. **Preparation:** The method generates the distances and distances_cost dictionaries, so we can access them in $O(1)$. This preparation phase can be seen on Figure 1.

2. **Optimization:** The method optimizes the action sequence for the given workstations and tasks by minimizing the total time and cost. The optimization process takes into account the speed of different actions, distances between actions, and the cost of traversing different paths. It uses the calculate_time(), calculate_cost(), and calculate_path_cost() functions to evaluate different possible sequences of actions and selects the best one based on the optimization criteria.

3. **Planning:** Once the optimal action sequence is determined, the method plans the robot's path by identifying the sequence of waypoints that the robot needs to visit. This is achieved by finding the affordance centers of the workstations involved in the optimized action sequence. The affordance centers represent the points in the environment where the robot can perform the required actions.

4. **Execution:** With the optimal action sequence and the corresponding waypoints determined, the method executes the action sequence by navigating the robot to each workstation's affordance center and performing the specified actions. This is done using the execute_action_sequence() function, which communicates with the "/do_action" service to execute the actions.

5. **Navigation:** The method navigates the robot to each point in the optimized path using the movebase_goal() function. This function sends a goal to the MoveBase action server, which is responsible for controlling the robot's movement.

Optimisation can be split into subparts:

1. Generate all possible action sequences: Enumerate all possible permutations of actions to be executed at the workstations. This is done using a branch-and-bound approach.

2. Evaluate action sequences: For each generated action sequence, calculate the total time and cost using the provided calculate_time() and calculate_cost() functions. We use calculate_path_cost() to find the cost of the path for the sequence.

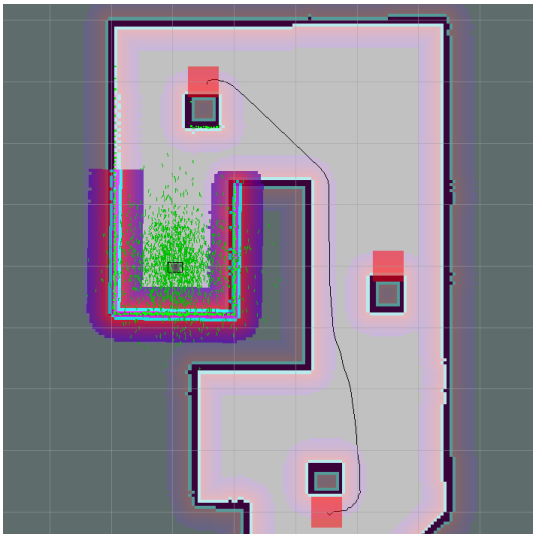


Figure 1: *calculate_distance_dict*() calculates the distance and cost dictionaries with GetPlan

3. Compare sequences: Compare the calculated total time and cost of each action sequence to determine which one is optimal based on the given optimization criteria. We choose the sequence that maximizes the reward and then minimizes the total cost.

4. Select the optimal sequence: Select the action sequence that best meets the optimization criteria and use it for further planning and execution.

4 Experiments

Once again, since the lab was closed for the renovation we could not test our code on the physical robot, so we the results below are for simulation only.
We used the "map3", which was the one provided to us. The map is representing the room with 3 workstations, each workstation has an affordance center that is spawned in randow spot near the workstation. All actions can be performed only in those affordance areas.
The goal of the experiments was to validate the plans proposed by our planning algorithm. In order to generate that plans we run "assignment3_manager" that simulated the affordance function, broadcasted the locations of affordance centers and validated the robot actions. We also provided the initial position of the robot through RVIZ in order to produce the paths using GetPlan.
After the best plan was chosen by the planning algorithm we used it to run the robot through the tasks and collect the reward.

5 Results

Only one map was provided for the experiments, so the results are for the "map_3" only. For a time limit of 5 minutes our algorithm succesfuly executed the plan compleeting all possible tasks in 3 minutes and 43 seconds, achieving total reward of 110 and with the cost of 3560. Our estimated time was 2 minutes and 25 seconds. This difference can be explained by the proximity

of the affordance zone centers to the workstations and inaccuracy of the initial position, that made GetPlan return the shorter times for paths than it was in simulation. This can be easily adjusted by a proper multiplier on the initial estimation, but we deliberately decided to keep it this way since the multiplier can be different for different maps.

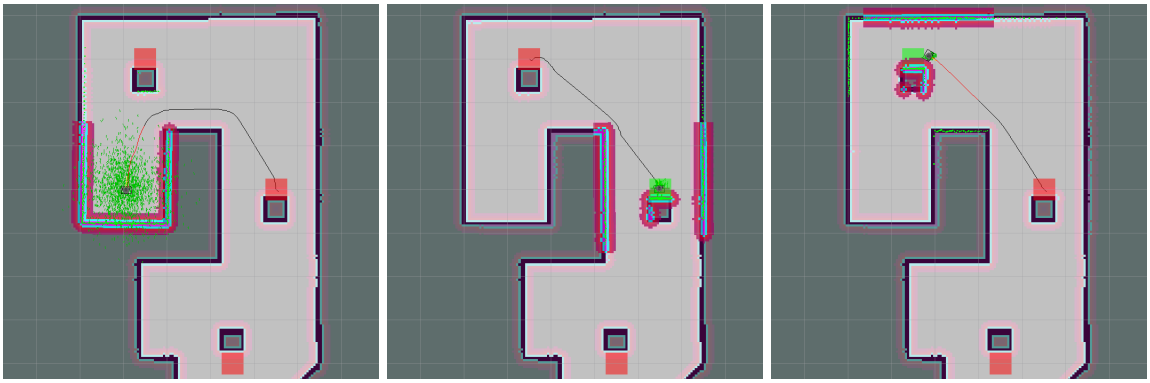
Navigation process can be seen on Figure 2, 3 and 4. The robot executes the action sequence only after the optimal action sequence if found. For a static environment like a warehouse where dynamic replanning of the tasks is not required and all workstations are fixed that gives us an ability to precalculate all the paths and use very simple and cheap robots to execute them. This also guarantees us path optimality by the nature of our algorithm.

Upsides of suggested approach:

- **Modularity:** The code is structured in a modular way, with different classes and functions handling specific tasks. This makes it easier to understand and maintain.
- **Functionality:** The code implements basic functionality for navigating the robot through the workstations and executing action sequences and runs really fast for given number of workstations.

Downsides of suggested approach:

- **Heuristics:** The current solution does not include any heuristics that could reduce the time required for finding the best action sequence if the number of workstations is exponentialy bigger.
- **Limited scalability:** Generating all possible action sequences can be computationally expensive when the problem size increases.

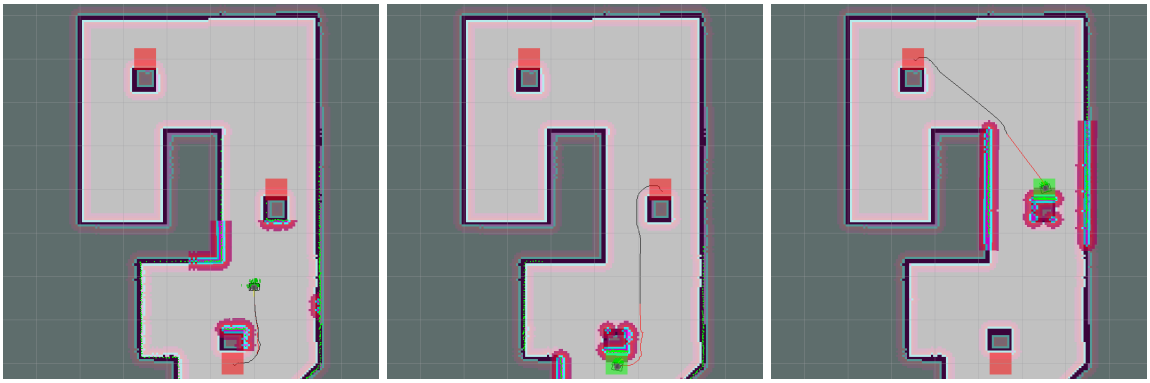


(a) Initial postiton (b) WS0 executing ACT1 (c) WS1 executing ACT3

Figure 2: Navigation process

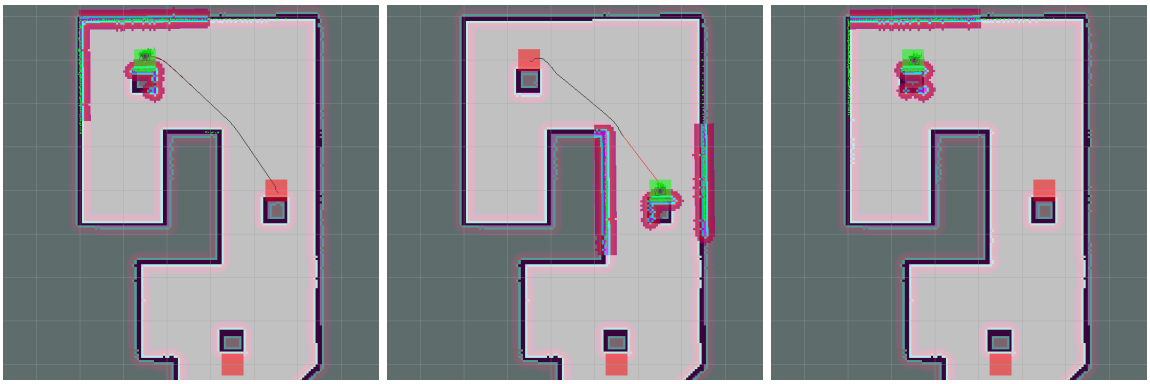
6 Discussion

The experiment have taught us several important tasks from different fields. We learned the importance of choosing the efficient planning algorithm that is relatively simple and can plan an efficient path to scan all the objects. However, we see a lot of directions to improve our algorithm:



(a) WS0 executing ACT2 (b) WS2 executing ACT4 (c) WS0 executing ACT1

Figure 3: Navigation process



(a) WS1 executing ACT2 (b) WS0 executing ACT2 (c) WS1 executing ACT3

Figure 4: Navigation process

- Optimization algorithm: Implement an optimization algorithm, such as a genetic algorithm, simulated annealing, or a dynamic programming approach, to find the optimal action sequence based on the criteria of time, cost, or both.
- Heuristics: Use heuristics or approximate algorithms to handle large problem sizes more efficiently and reduce computational complexity.
- Multi-robot coordination: Extend the solution to handle multiple robots working together, which would require additional coordination and synchronization mechanisms.
- Dynamic task allocation: Implement a dynamic task allocation mechanism that allows the robot to handle changes in tasks or workstations during execution, such as tasks being added or removed, or workstations becoming unavailable.