I. World Simulation

In this part, we present the simulation of a robot world using Python and some results. The simulation involves creating a virtual environment populated with robots and landmarks, simulating their movements, and analyzing their interactions. Through this simulation, we simulate the behavior of robots in an environment. Further enhancements could involve more complex movement algorithms and collaborative tasks.

A. World Initialization

A Python class named **world** is utilized to create and manage the simulated world. This class allows for the creation of robots and landmarks and whatever we need to simulate WAVN within specified boundaries. The code snippet creates an instance of a world simulation. Instantiate a world with 5 robots and 20 landmarks, denoted by the model name "P".

```
w = world(modelname="P", nRobots=5,
    nLandmarks=20)
w.drawWorld(filename="initial_world.pdf")
```

and the result is:

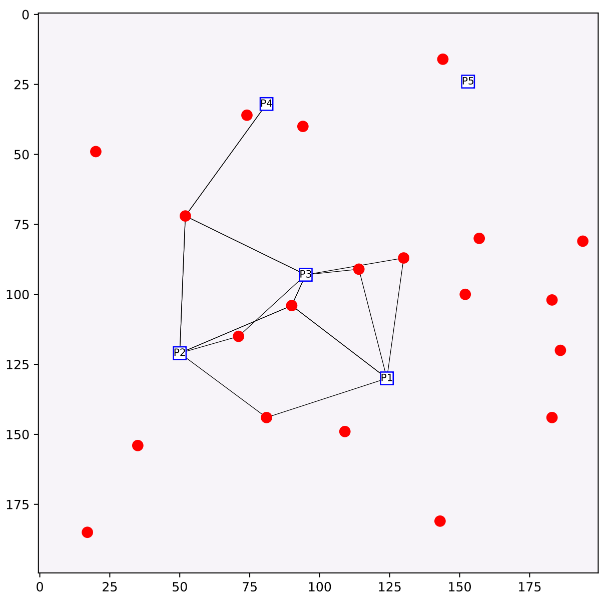

Fig. 1. Initial world setup

In the visualization, landmarks are represented by red circles, while robots are denoted as squares based on their names. Additionally, the visualization includes lines connecting common landmarks between two specific robots.

The simulation identifies common landmarks between pairs of robots based on their proximity to each landmark. To maintain consistency between the simulation and real-world cases, the structure of the common landmarks data is designed to mirror that of WAVN. This ensures that the data organization remains consistent across different environments and scenarios, facilitating more straightforward interpretation, comparison, and migration. for example

```
matches, reverse_matches = w.
    CommonLandmarkPanos("P1", "P2")
print("Matches␣between␣'P1'␣and␣'P2':")
print("Matches:", matches)
print("Reverse␣Matches:", reverse_matches)
```

and the output is:

```
Matches between 'P1' and 'P2':
Matches: [[102, 81, 144, (1, 5)], [146, 90,
    104, (0, 2)]]
Reverse Matches: [[102, 81, 144, (5, 1)],
    [146, 90, 104, (2, 0)]]
```

For our experiment, each robot is tasked with searching for home by examining transactions. We randomly assign one landmark to serve as the home for each robot, and this assignment remains constant throughout the duration of the experiments.

```
print("Home␣positions:", w.homelandmark)
>> home:  {'P1': (109, 149), 'P2': (114, 91)
    , 'P3': (35, 154), 'P4': (186, 120), 'P5'
    : (81, 144)}
```

1) Movement Simulation: Robots are programmed to move within the world boundaries randomly. The movement is limited to a maximum range to maintain realism. The movement of robots is constrained by limitations, and ultimately, we can generate a PDF file that depicts the robots' movements. It's important to note that the positions of landmarks remain fixed throughout, and we operate within a static environment rather than a dynamic one.

```
w.move()
w.drawWorld(filename="world_after_movement.
    pdf")
>> Robots are moved!
Robots that moved: ['P1', 'P2', 'P3', 'P4',
    'P5']
```

In our scenario, all robots are in motion. We've intentionally structured the code to increase the likelihood of robot movement by setting

```
should_move = random.choice([True, True,
    False])
```

, where two 'True' values and one 'False' value are included, thereby favoring movement. This setup allows for easy adaptation to an environment where all robots either move or remain stationary.

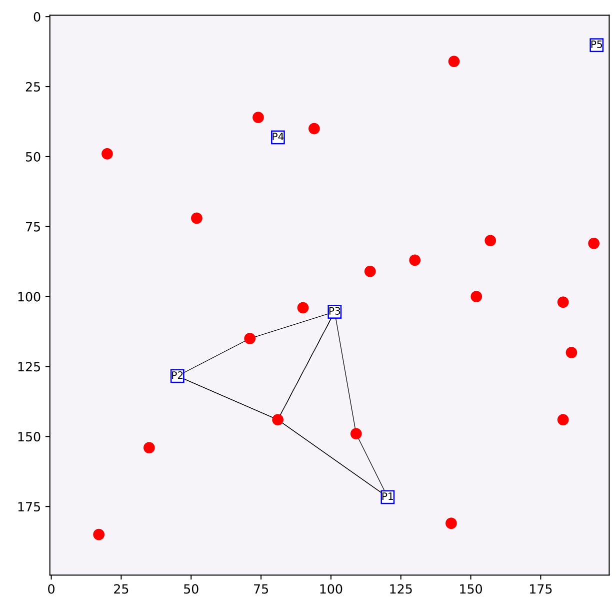

Fig. 2. Movements!

And for tracking replacements of robots:

```
% w.movements(filename="robot_movements.pdf"
    )
```

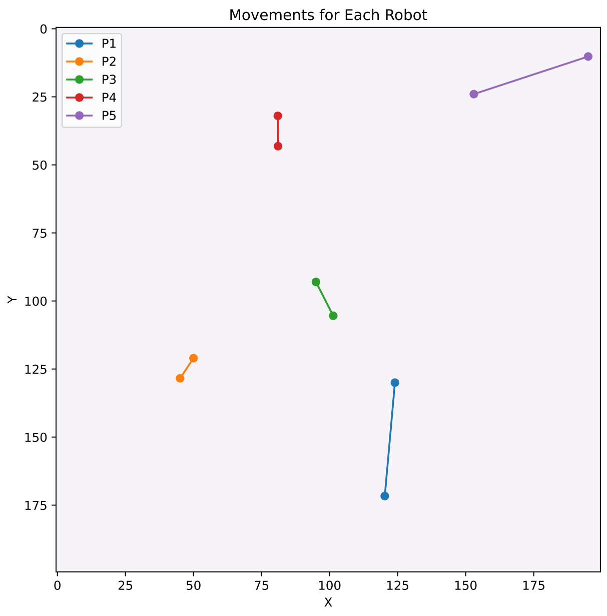

Fig. 3. Tracking of robots positions!

In the final step, we identify the robots capable of observing the home of a specific robot in order to determine the shortest path. To achieve this, we iterate through all robots to identify those with the searcher robot's home within their field of view. For example, which robots can see the home of robot "P1":

```
print("Robots␣seeing␣the␣home␣of␣'P1':",
    robots_seeing_home_of_P1)
>> Robots seeing the home of 'P1': ['P1', '
    P3']
```

B. Discussion

The simulation demonstrates the dynamics of a robot-populated environment, including their movements and interactions with landmarks. By identifying common landmarks, robots can potentially collaborate or communicate based on shared environmental knowledge.