Multi-robot systems
Lab 3

Jorge Aranda NIP: 800839
Alberto Zafra NIP: 876628

MRGCV - 2023/2024

1 Exercise 1 - Multirobot formation for target enclosing

1.1 Introduction

This report details the development of a multirobot formation control algorithm for target enclosing. The primary goal of this practical exercise was to explore the application of multirobot formations in surrounding and securing a target. The project involved programming the formation control algorithm and implementing a simulation environment to test the performance of the robots.

The report outlines the methodology, formation control strategies, and the simulation environment used in this project. Finally, the report concludes with results changing different parameters.

1.2 Approach

The chosen programming language for this exercise was Python. We have created a class that handles the formation control logic and simulation. It has several key parameters.

Main Parameters and Variables of the Formation Class: The Formation class is the core component that handles the multi-robot formation control simulation. Here is a breakdown of its main parameters and variables:

Initialization (init):

- **q**: This NumPy array stores the initial positions of all robots (including the enclosed target).
- **n**: This integer stores the total number of robots in the simulation.
- **c**: This NumPy array stores the desired formation positions for all robots relative to each other.
- **rotation_cotrol**: This boolean flag determines whether rotation control is enabled for formation adjustment.
- **stop_flag**: This boolean flag controls the simulation loop. Setting it to True stops the simulation.
- **num_iters**: This integer specifies the total number of simulation iterations.
- **Kc**: This float value represents the control gain for robot movement towards desired positions.
- **Kg**: This float value represents the control gain for rotation control (only used if rotation_control is True).
- **dt**: This float value represents the timestep for robot movement updates.
- **enclosed_vel**: This NumPy array stores the random velocity of the enclosed target robot.
- **desired_vel**: This float value represents the desired maximum speed of the enclosed target robot.
- **vel_update_rate**: This integer specifies how often the enclosed target's velocity is updated (every vel_update_rate iterations).

1.3 Results

The following section presents the results obtained from the implementation and experimentation of the multi-robot enclosure and robot formation. In this case six different shape formations have been implemented, with the addition of the rotation control implementation. The implemented formations are presented bellow with a execution example. It is worth mentioning that for executing the program the command "`python enclosing.py --n_robots NUM_ROBOTS --shape SHAPE --rotate True/False`" has to be used.

- **Line**: the robots will perform a linear formation keeping the robot target in the median point of the line, as presented in Figure 1. It is worth noting that the command used to obtain the following result has been "`python enclosing.py --n_robots=7 --shape=line`".

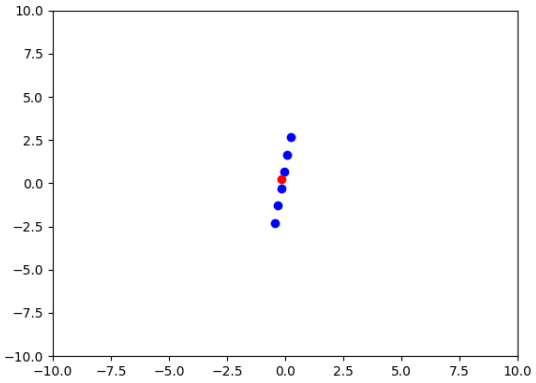


Figure 1: Linear formation

- **Circle**: the robots will perform a circular formation keeping the robot target in the centre point of the circle, as presented in Figure 2. The command used to obtain the following result has been "`python enclosing.py --n_robots=15 --shape=circle`".

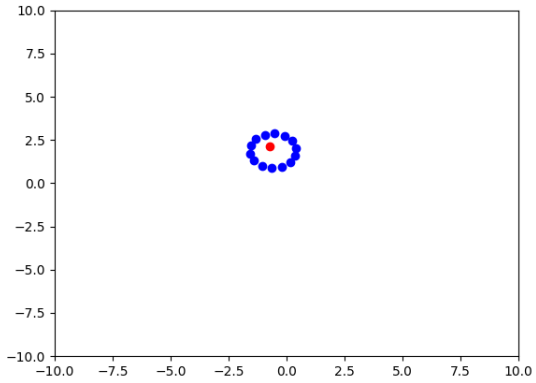


Figure 2: Linear formation

- **Grid**: the robots will perform a grid formation keeping the robot target in the mean point of the grid, as presented in Figure 3. The command used to obtain the following result has been "`python enclosing.py --n_robots=10 --shape=grid`".

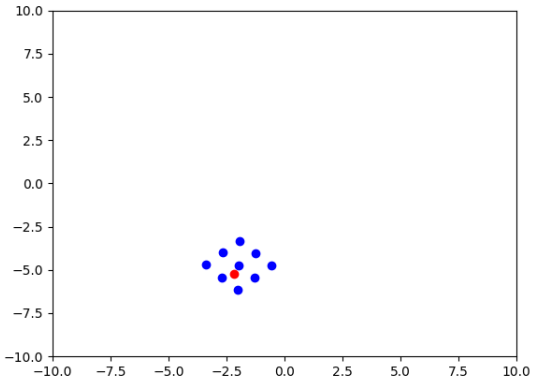


Figure 3: Grid formation

- **Square**: the robots will perform a square formation keeping the robot target in the mean point of the square, as presented in Figure 4. The command used to obtain the following result has been "`python enclosing.py --n_robots=13 --shape=square`".

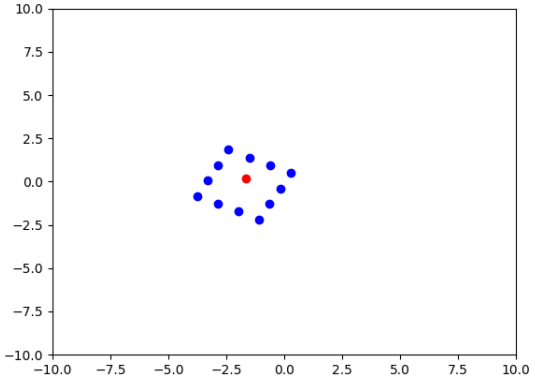


Figure 4: Square formation

- **Star**: the robots will perform a star formation keeping the robot target in the median point of the star, as presented in Figure 5. It is worth noting that the command used to obtain the following result has been "`python enclosing.py --n_robots=11 --shape=star`".

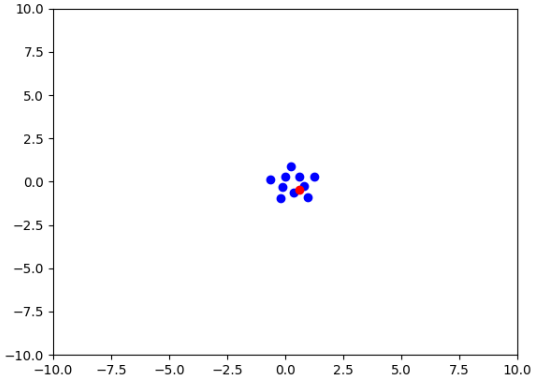


Figure 5: Star formation

- **Hexagon**: the robots will perform a hexagonal formation keeping the robot target in the median point

of the hexagon, as presented in Figure 6. It is worth noting that the command used to obtain the following result has been "`python enclosing.py --n_robots=20 --shape=hexagon`".

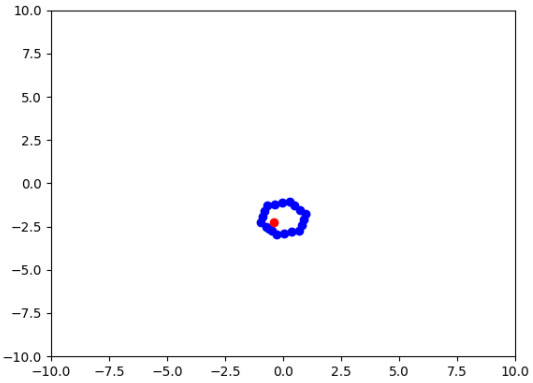


Figure 6: Hexagon formation

2 Exercise 2 - Persistent coverage

2.1 Introduction

This practical exercise aims to explore the various aspects involved in achieving persistent coverage using a group of mobile robots. The main goal is to develop a 2D simulator that models the persistent coverage behavior of these robots.

2.2 Approach

We have used Matlab for the programing and simulations. In order to model the simulation, we have several parameters that can be modfied before executing:

In the context of the persistent coverage simulator, each parameter plays a specific role in defining the behavior of the environment and the agents. Here is an explanation of each parameter:

- **A**: This parameter represents the rate at which the coverage level of the environment degrades over time. A negative value indicates that the coverage is continuously decreasing, simulating the natural loss of coverage quality when an area is not actively monitored by an agent.
- **B**: This parameter quantifies the influence of the agent's actions on the coverage level. A higher gain means that the agent has a more significant impact on improving or maintaining the coverage quality when it visits different points in the environment.
- **gridSize**: This parameter defines the size of the 2D grid representing the environment.
- **time_steps**: This parameter specifies the number of time steps for which the simulation will run. Setting this to a positive number limits the simulation to that many steps, while a value of -1 indicates that the simulation should run indefinitely until manually stopped.
- **num_agents**: This parameter determines the number of agents deployed in the environment. More agents can potentially cover the area more effectively, as each agent contributes to maintaining the coverage level.
- **method** (Update method): This parameter indicates the strategy used by the agents to update their positions and actions. Different methods (min, rand, quant, terc) represent different algorithms or heuristics for decision-making in the coverage process. For example, "rand" method will take a random destiny in the map from the points where the coverage is below the acceptable. "Qant" is a little bit more sophisticated. It takes a random value priorizing the points that are closer to the robot. On the other hand, "min" and "terc" are not random. They take the closest point below acceptance ratio and the $1/N$ closest respectively. Being N the number of points.

- **rad** (Radius of actuation): This parameter defines the effective range within which an agent can influence the coverage level. A larger radius means that each agent can cover a broader area around its current position.
- **power**: This parameter measures the effectiveness of the agent's coverage action. Higher power allows the agent to make a more substantial impact on the coverage quality within its actuation radius.
- **max_vel**: This parameter sets the maximum speed at which the agents can move within the environment. Higher velocity enables agents to cover more ground in a shorter amount of time, potentially improving overall coverage efficiency. Also, note that this is not the overall velocity but the max velocity. The speed in a given moment is calculated depending on how covered is the area where the robot is. The more covered, the faster the agent will go through that area to avoid overcovering.
- **minPerc** (Minimum acceptable percentage of coverage): This parameter defines the lower threshold for coverage quality. If the coverage level falls below this percentage, it indicates inadequate monitoring of the environment.
- **maxPerc** (Maximum acceptable percentage of coverage): This parameter sets the upper limit for coverage quality. Coverage levels above this threshold may indicate an excess of resources, suggesting that fewer agents or less frequent visits might be needed.

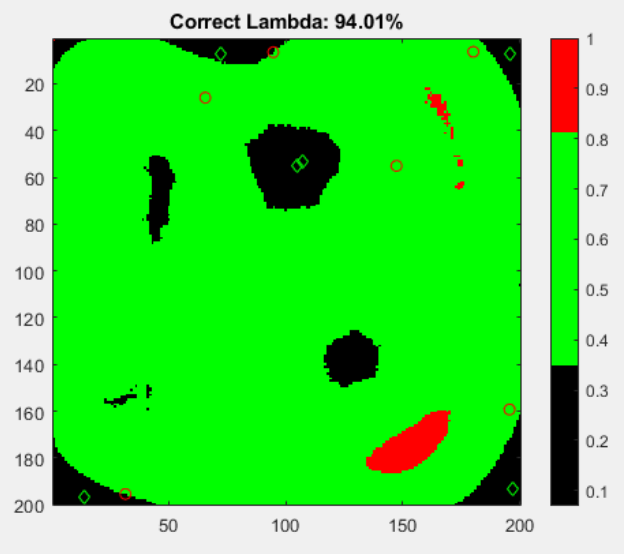


Figure 7: Simulator

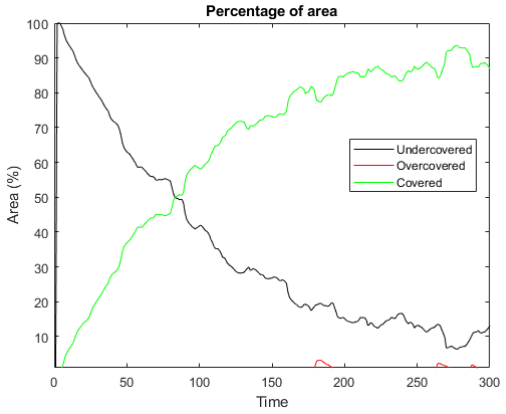
We can see in Figure 7 how the feed is distributed. In the center we have the world grid, where the red circles are the agents. The green region is correctly covered, the black region is below what is considered enough covered and the red regions are overcovered. The green diamonds represent goals for the agents. On top, we can see the percentage of correctly covered area.

For using it. You have to use the *main.m* script. In it, you can change the parameters mentioned above. Once you have your own parameters, or the ones from loading the presets, you can simulate it in the second section of the script. A live simulation will appear (it pauses for 2 seconds in order to give time to the user for seeing the scene). After simulating, the next section of the *main.m* script creates a plot that indicates the percentage of area that was okay or not. Finally, in the last section. You can replay Past simulations.

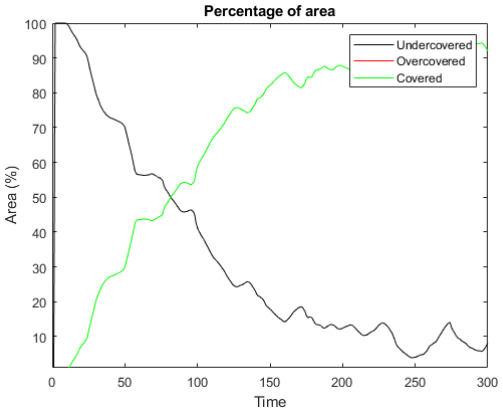
2.3 Results

We have stored three presets of the simulation parameters in a folder called *params*. In this presets we can see three interesting cases of use where 1, 4 and 6 agents try to clean areas with different properties.

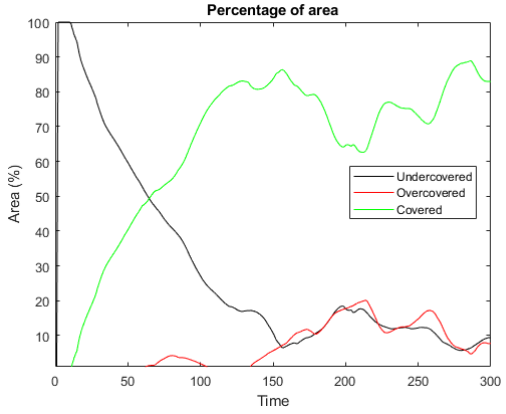
There are three videos of the simulations that we cannot put in the report for obvious reasons. However we have plotted the amount area that is under/well/overcovered of the three presets.



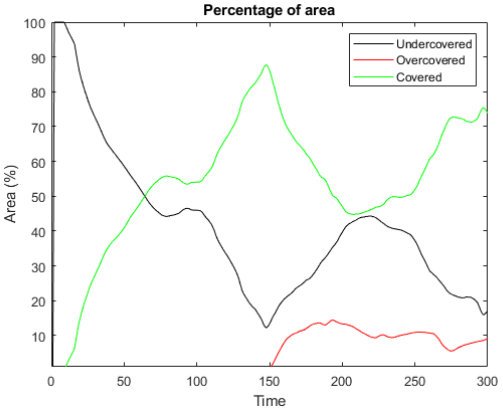
(a) One cleaner.



(b) Four cleaners.



(c) Six cleaners.



(d) Six cleaners using min method.

Figure 8: Results for the different presets

We can see in figures 8a, 8b and 8c how our solution uses to reach the 90% of covered land using the method *quant* for diverse amount of presets. Using other more naive methods such as *min* the efficiency is way worse and unstable as seen in Figure 8d.

3 Exercise 3 - Hybrid reciprocal velocity obstacle method (HRVO)

3.1 Introduction

The primary objective of this session is to implement and test a dynamic obstacle avoidance algorithm within a multirobot setup. Specifically using the Hybrid Reciprocal Velocity Obstacle (HRVO) method.

3.2 Approach

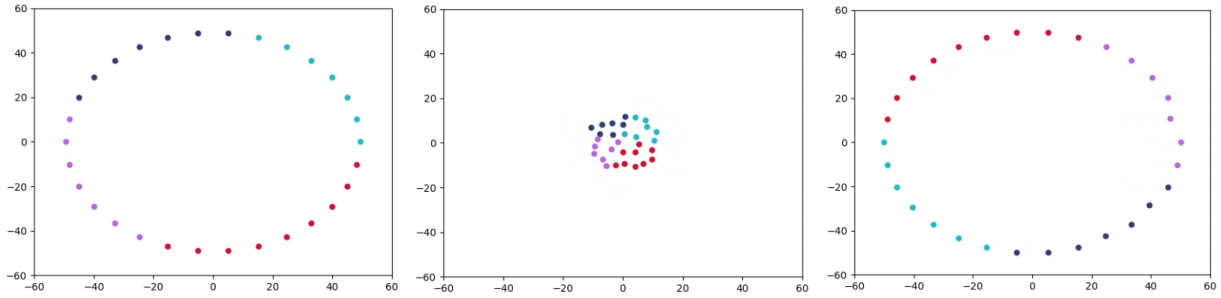
Within this exercise we have faced the visualization problem by, first storing the output generated by the *HRVO* system in a "*.txt*" file, for later reading the aforementioned output using a custom python script named "*visualize_sim.py*", which is capable of reading the previously generated output and plotting a visualization of the simulation. This whole process is performed sequentially by executing a single "*.sh*" file, which depending on the selected environment - *execute_circle.sh* or either *execute_A.sh* - would simulate a circular multi-robot behaviour or a letter shaped formation, while avoiding its neighbors.

For the circular implementation, found in *./examples/circle.cpp*. We have set the positions using basic trigonometry. For the letter implementation, found in *./examples/A_shape.cpp*. We have set different directions according to the movements you would have to do to draw an A letter. After that, assigned the positions to each robot respecting an inter-robot distance in between. It is worth mentioning, that as the A letter contains concurrent paths, where the robots may be really close between each other in their respective position of the letter formation, this previously mentioned inter-robot distance is incremented with respect to the circular and square inter-robot distances.

3.3 Results

We have included videos in the .zip. However, here are some examples of the different shapes:

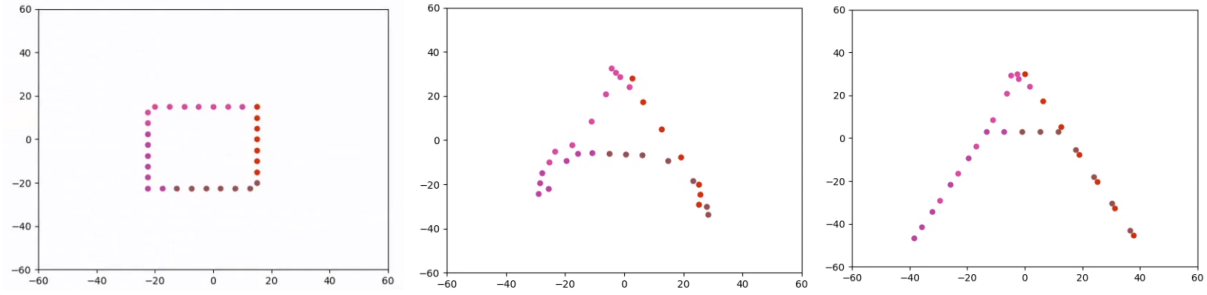
3.3.1 Circle



(a) Start of the simulation. (b) In the middle of the simulation. (c) End of the simulation.

In the preceding figure, we can see how in 9a the agents starting in a circular setting. They start going in 9b into the desired position (which is the opposite one). Untl eventually they reach the desired position in 9c.

3.3.2 A letter



(a) Start of the simulation. (b) In the middle of the simulation. (c) End of the simulation.

This case is different. The agents start in a square formation in 10a. They start moving until they reach an A formation as seen infigure 10c.

4 Conclusions

In this lab, we have explored various aspects of multirobot systems through three distinct exercises: multi-robot formation for target enclosing, persistent coverage, and dynamic obstacle avoidance using the Hybrid Reciprocal Velocity Obstacle (HRVO) method.

Exercise 1

The results demonstrated the effectiveness of the formation control algorithm in maintaining the desired formations around the target. The ability to control the formation's rotation further enhanced the flexibility and robustness of the system.

Exercise 2

We implemented a Matlab simulator to study persistent coverage by mobile robots. We tested different parameters and saw hoe they affect the behaviour of the system.

Exercise 3

We implemented code so the agents could navigate and form specific shapes, such as circle and the letter 'A'. The successful execution of these formations confirmed the HRVO method's efficacy in dynamic, multi-agent environments.