Robot network swarm order sorting in a simulated warehouse

Mario Tilocca & Zaida Brito Triana
Distributed Systems for Measurement and Automation
Professor Daniele Fontanelli
Department of Industrial Engineering - University of Trento
mario.tilocca@studenti.unitn.it
zaida.britotriana@studenti.unitn.it
Project Repository: Github page

***Abstract*—The unstoppable growth of eCommerce has come hand in hand with a growing worldwide demand for better, more efficient and quicker warehousing of orders.**
The aim of this project is to investigate within the context of a distributed system (robot network) the different advantages offered by the collaboration between AGVs to improve order delivery and general performance in the context of a warehouse with static and dynamic obstacles receiving different orders simultaneously or at different times. Two *swarm* algorithms, one with intra agents collaboration and one without, are proposed and their performances are analyzed in a simulation scenario developed in *Python*.

***Keywords*—Distributed Systems, Robot Network, Warehousing, Multi-agent, Navigation, Autonomous Robots**

I. INTRODUCTION

The continued growth of e-commerce underscores the need for research and development in this area to achieve more efficient automation in warehouses. Data backs support it, on the third quarter of 2020 retail e-commerce sales are estimated at 209.5 billion meaning an increase of 36.7% from the third quarter of 2019 [1]. This rapid increase highlights the importance to start developing networks of reactive agents to improve efficiency in logistic operations in automated warehouses.
Together with this, it needs to be taken into account the fact that it has come into play the improvement of collaboration between the different systems that constitute a warehouse. This field is where technologies such as automatic guided vehicles (AGVs) can be widely employed to enable safer, faster, and more efficient connections in distribution centers. In this sense, horizons are opening up for research and development on the challenges involved. Warehouses efficiency plays a crucial role in the global industries supply chains. Therefore with a steadily increasing number of e-commerce related warehouses around the globe, efficient sorting of orders from one area of the warehouse to the other is essential. The main goal of warehouse automation is to eliminate human related labor-intense duties while maximizing efficiency in terms of robots improving the speed, reliability and accuracy of the tasks related to order sorting [2].

A. Problem Formulation

Therefore with above discussed premises it is worth noting that while hardware wise there exists a large number of AGVS suited for warehouse sorting; navigation and efficient algorithms for fast order sorting offer an interesting area of research. In this paper two different *Distributed System* solutions are elaborated and analyzed in a simulated warehouse setting. Thus a research question is formulated as following:

What multi-agent approach can offer a reliable and optimized solution for fast order sorting minimizing the delivery time within intra warehouse orders ?

The two proposed solutions will present two similar but yet different approaches how a *Robot Network* can be built in a simulated environment where the main focus will be on the algorithmic development of a reliable software for order managing and autonomous agents capabilities.

II. ADOPTED MODELS

Before jumping in the explanation of the proposed algorithms, the implementation and results of the developed simulator it is important to note that the simulator is a Distributed System, whose definition can be explained as following:

Distributed system: a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user [3].

A. Robot Network

In this paper the research is focused on a particular type of *Distributed System*, the **Robot Network System** (RNS). The core principles of a RNS are based upon the assumptions that the agents are able to freely move within an environment, either UAVS or mobile robots for instance. Furthermore in RNS a collaborative approach of each single agent is adopted in coordination with the other agents in order to appear a single system. Moreover an RNS in order to be considered as

such it must include the following [4]:

- ***Physical embodiment***:
  Any RNS has to have at least a physical robot which incorporates hardware and software capabilities
- ***Autonomous capabilities***:
  A physical robot must have autonomous capabilities to be considered as a basic element of a RNS
- ***Network based cooperation***:
  The robots, environment sensors and humans must communicate and cooperate through a network
- ***Environment sensors and actuators***:
  Besides the sensors of the robots, the environment must include other sensors, such as vision cameras and laser range finders etc.
- ***Human-robot interaction***:
  In order to consider a system as NRS, the system must have a human- robot related activity.

B. System Model

In the RNS model analyzed several assumptions had to be made.
For what regards the ***Physical embodiment*** the system model focuses solely on the development of software capabilities. Hardware constraints such as environment mapping through techniques such as SLAM or position monitoring through GNSS and IMU sensors are outside the scope of this paper as due to scalability issues it would have been difficult to test a system an increasing number of agents without taking into account possible sensors malfunctions in the single agents. For this reason it is assumed that the environment is already mapped and known to each agent and that each agent is an *holonomic* robot when modeling the motion in the warehouse. Similarly the part related to the ***Environment sensors and actuators*** will not be analyzed as well.

Conversely in this paper the ***Autonomous capabilities*** strongly emphasised and the proposed solutions mainly focus in this domain, such as the ability to independently or cooperatively with another agent to execute single tasks. Similarly a strong emphasis in one of the two proposed algorithms is put on the ***Network based cooperation***, while in the first algorithm discussed the Network based cooperation will mainly focus on the agent to the main control layer, in the second algorithm the agents are also able to have intra-cooperation and networking capabilities. Lastly the human related robot activity analyzed is represented by the sorting of orders within a warehouse, which in turn if not partially or fully automated is a labor-intensive task for the humans.

With the above mentioned assumptions the *system model* idea is represented in Fig.[1].

The system model aims to simulate a warehouse environment, where the warehouse dimensions are represented by an nXn square. At the two horizontal extrema of the warehouse a variable number of *Pickup Stations* and *Delivery Stations* as well as a variable number of *Agents*; their precise quantities can be specified by the user, however for conceptual reasons in Fig.[1] it was decided to portray only two of each. It was decided to place them in this particular configuration in order to resemble loading and unloading areas connected to external terminals where orders arrive and depart via trucks.
As it is shown in Fig.[1] and the corresponding explanation of its symbols in Tab.[I] the distribution of the orders taken into consideration from each pick up station can be directed towards all delivery stations and each agent can be assigned all types of orders to first pick up from a specific pick up station with the end goal of delivering it to the assigned delivery station in the order details.

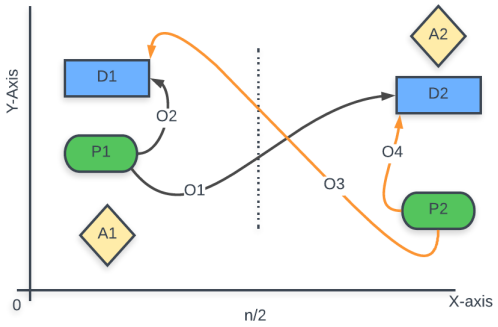

Fig. 1: Warehouse setting and order possibilities

TABLE I: Components of the *System Model*

Components	Symbol
Pick up station	P
Delivery Station	D
Agent	A
Order	O

Therefore, ***Order Distribution*** is the first **layer** of the system, where each order is assigned to a specific agent. The first two secondary layers of the system model are represented by the ***Agents*** and ***Orders***, once an agent receives an order it autonomously takes care of its task until the order has been delivered, once this happens this is communicated to the Order Distribution layer which will acknowledge that the order has been successfully sorted.
Finally it is important to mention that in order for the agent to reach its goal destinations a ***Path Finder*** layer is needed in order to calculate the optimal trajectory from its current location to its goal location.

III. Solution

In this section firstly the two proposed multi-agent scenarios are presented. Secondly the well-known A$*$ navigation algorithm is presented, as it plays a crucial role for both RNS scenarios. The last subsection concerns the metrics and evaluation methods which constitute the base for the tests and results discussed in Section [V].

A. RNS Modeling Approaches

The two proposed solutions ideas are shown in Fig.[4]. As it is shown there, several common factors between the two are present. It is worth noting that *Environment* will act as the master node in a **Master-Worker architecture**.Given an input file where the specific number of agents and orders is specified, the environment will sort them and assign them to the agents.

In scenario 1 as depicted in Fig.[4] once the environment has distributed the orders according to a specific pattern, randomic or distance optimized. The agents will complete the assignments in time-steps autonomously. Firstly from their current location they will reach the pick-up point of the order, once this goal is reached their goal position will be updated into the delivery station and the navigation algorithm will compute the shortest path to reach it. Secondly, the goal position will be updated to the delivery station. Once the agent has reached the delivery station of the assigned order, it will communicate to the master node that it has completed its assignment. In case other orders need to be processed the agent will be assigned a new one, otherwise it will reach its initial position and will stay there until all other agents have finished to deliver their orders.

It is worth noting that in both scenarios fixed obstacles will be present, as well as *dynamic obstacles*, which are represented by the agents and their current position. Thus it is explained the need of executing the tasks in time-steps as path recalculation based on the new position of the other agents is needed in order to reach the goal position.

In scenario 2 as shown in Fig.[4] once an agent has received an order it is checked if the order delivery station is situated in the opposite side of the warehouse compared to the robot. In that case the master node will search for an available agent stationed on the side of the delivery station and a **Collaborative Pair** with the two agents is created. The first agent will pickup the order from the pickup station, in similar fashion as presented for scenario 1 and will meet the second agent in a pre-determined meeting point area. The meeting point coordinates are contained in the order details and while they will be the second goal of the agent which picks up the order, the agent positioned on the side of the delivery station will reach the meeting area and will wait there for the agent which picked up the order to arrive. The meeting points for all orders are assumed to be positioned along the $n/2$ x-axis as shown in Fig.[1]. Thus for this scenario the system model, previously discussed in Section.[II-B] had to be modified with the addition of the designated meeting areas in the middle of the warehouse.
Once both agents of the pair have reached the meeting area, the 'Deliverer' agent is assigned the order and its goal will be the delivery station, the 'Picker' agent will be assigned a new order or will reach its initial position. It is worth noting that in case the 'Picker', once it has met with the 'Deliverer' could itself become a 'Deliverer' agent in a new order, as the assignment of the 'Picker' or 'Deliverer' role is purely based on the specific order pickup and delivery coordinates and it is not pre-defined in the agent itself.

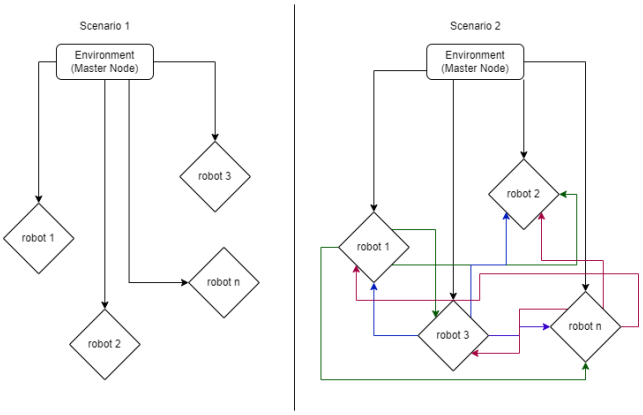

Fig. 2: RNS proposed approaches

B. Agent Navigation - A∗ algorithm

A∗ algorithm : A * algorithm or Astar algorithm is a widely used algorithm not only in robotics but in artificial intelligence solutions and is a searching algorithm that searches for the shortest path between the initial and the final state. In spite of the fact that there are multiple shortest path finding algorithms, A* algorithm was chosen for this project over others algorithms. The heuristics approach from the A* algorithm is one of the reasons why it is widely used in environments such as the one developed in this project, meaning, path-finding on geographical environments(maps in general), making possible to calculate the cost between nodes and the target. Although A* algorithm might not always result on the most optimal result is considerably fast and not as computational expensive as other broadly used algorithms like Dijkstra that explores all possible paths. It is worth mentioning that in the implementation of this algorithm as well as the randomStep function we have used the help of pre-existing libraries and repositories [6] [7].

C. Evaluation Metrics

It is worth noting that although in the python simulator the distance is calculated in units of distance per time-step (every one time-step is one unit of distance), it has been decided to consider and analyse the results obtained in metric system, in units of metres, in order to be able to analyse the data from a more practical point of view.
Following this line of research it has also been decided to consider the size of the experimental maps, considering that the average American warehousing size is around [50x50]m [8].
It is also important to note that one of the parameters used as an indicator of the efficiency of the system is the λ parameter,

a variable that reflects the difference between the Euclidean distance between the Pick-up-station and Delivery-station selected in each order and the distance actually travelled by the agent or agents involved since the order was placed, which is increased by factors such as the need to deviate from static and mobile obstacles (other agents), time lag in the collaboration between robots or simply inefficiencies in the optimisation of the path calculated for the robot. It can be seen in Eq.[1] where *End[n]* and *Inital[n]* are the time-steps or the initialization and end of a particular order and where *Distance[n]* is the Euclidean distance between the stations.

Its calculation are shown in Eq. [2], where x_2 and y_2 represents the coordinates of the Delivery-Station and x_1 and y_1 represent the coordinates of the Pick-up-Station. Likewise, the standard deviation and relative standard deviation ratio in the total distance travelled will be studied, as it gives a reliable approximation of how the system remains consistent in the time it takes to complete an order, since the Euclidean distance does not differ considerably from one order to another, as the stations have fixed positions around the map and the distance between all of them due to their specific placement at the borders of the warehouse yields to all of them being similar.

$$\lambda = [End[n] - Initial[n]] - Distance[n] \tag{1}$$

$$d = \sqrt{[(x2 - x1)^2 + (y2 - y1)^2]} \tag{2}$$

IV. Implementation Details

The system developed consists of 4 main layers, which are the master node the *environment*, the robots logic's as *agent*, a *path finder* which is common to the two scenarios presented, a peer-to-peer interaction layer for the agents as *pair* and an *input* layer to define the external characteristics the warehouse environment, the number of agents and orders. A secondary layer is represented by the *order* where details about the order characteristics are retrieved by the *environment*, *agent* and *pair* layers. This system was built using **Python** based on pure logic for operation as well as pre-existing libraries. It is worth noting that for the visualization of the simulation and path finding layer, the simulator was developed using as example the framework proposed in [6]. On the input layer the map grid, the obstacles and the different stations are defined, the environment code is the responsible then for the integration and renderization of the warehouse simulation and the creation of an *output* file which the *visualization* layer uses together with the *input* layer to create the animation. As it is shown in Tab.[II] the *agent*, *order* and *pair* layers are modeled according to the values of the states. It was chosen to implement it following this logic in order to better subdivide the navigation goals of the agents in order to achieve a more efficient order sorting following a fixed scheme.

Moreover, by defining the grid, obstacles, agents and stations in the input layer, the environment is able to integrate and render a whole warehouse. After which, the agent layer defines the logic behavior of all agents, and in the second scenario the agent and pair layers. Consequently, the path finding layer receives a map from the environment layer which it utilizes to generate a trajectory for each agent through traversing the map as a graph using the A* algorithm. Furthermore in order to keep track efficiently an ID is assigned to each order and agent. For a more in depth explanation of the logic of the interaction between the layers

TABLE II: States used by the three layers

State	Agent	Order	Pair
0	Done	Not Assigned	Available
1	Picking	Assigned	Busy
2	Delivering	Picked	
3	Meeting	Delivered	
4	Waiting		
5	Del. Collab		

Algorithm 1 Environment Layer

```
Read Input file containing: Map_size, Pick_up_points,
Delivery_points, Orders, Agents
from input file create:
Initialise variables
    orders list
    agents list
    Create Pairs list                          ▷ Scenario 2 Only
end Initialise variables
if orders list is empty then
    break
else
    Assign orders
    for order in order list do
        for agent in agent list do
            if agent State == Done then
                order assigned to agent.
            end if
            if order is collaborative then     ▷ Scenario 2
                Find closest free agent on other side
                if all other agents are busy then
                    Complete the other alone
                else
                    create Collaborative Pair
```

A. System 1: multi-agent order sorting

Specifically as it can be observed in Alg.[1] in the first scenario after the initialization of the orders and agents by reading the input file, the environment layer will run in a while loop until the maximum number of time-steps for the simulation has been reached or the order list is empty. It is worth noting that once the agent has reached the delivery station, the order state is updated to *Delivered* as shown in Alg.[2]. Moreover this is communicated to the environment layer which removes the order from the orders list. The agent once it has accomplished its delivery assignment in case no other orders need to be sorted, it is sent to its initial position. Furthermore in this configuration the agents are able to take care of only order at a time.

Algorithm 2 Agent Layer

Set order
if order is $Collaborative$ and agent is $Deliverer$ **then**
  Set $agent\ state$ to $Meeting$
  Meeting point coordinates from order.py
  Call A star path planner
  if $agent\ position$ == $Meeting\ point$ **then**
    Set $agent\ state$ to $Waiting$
  end if
  if agents have met **then**
    Set $agent\ state$ to $Deliverying\ Collab$
    Delivery coordinates from order layer
    if $agent\ position$ == $Delivery\ station$ **then**
      Set order state to Delivered
      Set $agent\ state$ to $Done$
    end if
  end if

else
  Set $agent\ state$ to $Picking$
  Get pick up station coordinates from order layer
  Call A star path planner
  if $agent\ position$ == $pick\ up\ station$ and order is non collaborative **then**
    Get Delivery station coordinates from order layer
    Set $agent\ state$ to $Delivering$
    Call A star path planner
  end if
  if $agent\ position$ == $Delivery\ station$ **then**
    Set order state to Delivered
    Set $agent\ state$ to $Done$
  end if
  if $agent\ position$ == $pick\ up\ station$ and order is collaborative **then**
    Get meeting point coordinates from order layer
    Set $agent\ state$ to $Meeting$
    if $agent\ position$ == $meeting\ point$ and collab agent is there **then**
      Set $agent\ state$ to $Done$
    end if
  end if

B. System 2: multi-agent intra collaborative sorting

As it can be noted in both Alg.[1] and Alg.[2], system 1 and system 2 share a considerable part of their logic. However it is worth noting that the *pair* layer represents the core difference between the two systems. In particular as it allows a peer-to-peer communication and collaboration between the agents. Conversely from the agents and the orders, the number of pairs is decided and they are created in the environment layer after the number of agents it is known. The pairs are all initialized as 'empty', once an agent is assigned a collaborative order and a suitable available agent is found the pair is assigned the agents and order. Consequently the pair state is updated to *Busy*. The purpose of the pair is to constantly check the agents state at each time-step of the simulation. When an agent has reached the meeting point in a collaborative order as shown in Alg.[2] its state is updated to *Waiting*. When both agents of the pair are *Waiting*, there is an immediate exchange where the order is de-assigned from the agent which picked it and immediately reassigned to the second agent of the pair without the intervention of the environment layer; this task is achieved in a collaboration between the pair, the agent and order layer. It is worth noting that with the new assignment in this particular case the order state is maintained equal to 2. Immediately after the agents have met as there is no need for collaboration any longer, the pair is de initialized and the 'picker' agent is available for other orders, and itself could become a 'deliverer' agent.

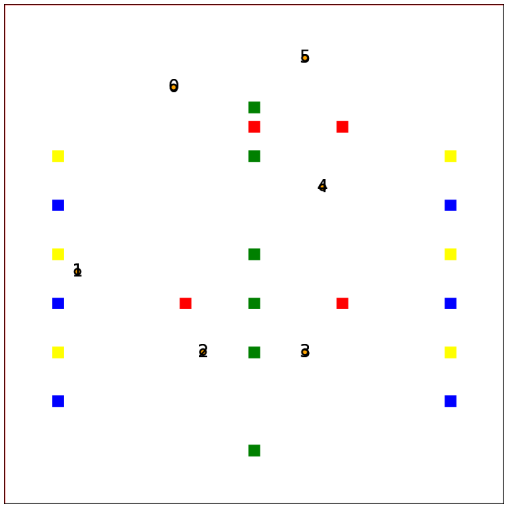

Fig. 3: Warehouse rendering map set up scenario 2

V. Results

Before discussing the evaluation of the test results and comparison of the two scenarios, in Fig.[3]it is presented the visual rendering of the warehouse simulator. It is worth noting that the sole difference between the two scenarios would be the absence of the meeting points in the rendering. Further example of the videos of simulation results can be found on the linked *GitHub* repository of the project.

In order to understand the behavior of the agents and how the two implemented approaches differ, it was decided to conduct 10 different possible experiments. In Tab.[III] the specifications of each test can be found. Furthermore a summarizing excel sheet with the characteristics and results of the different experiments performed can be found in the GitHub repository. It is worth noting that due to the fact that scenario 2 has the randomic factor in the assignment of the orders, it was firstly needed to understand how this could have affected the overall quality of the results.

In Tab.[IV] it is shown how the random assignment of orders to the agents does not constitute a major drawback to the whole scenario logic. As it is shown the avg. distance performed has

TABLE III: Characteristics of the experiments conducted

Test Nº	File name	Nº of agents	Nº of orders	Map size[m]
1	inputs1map1	6	20	[51,51]
2	inputs1map2	6	10	[152,152]
3	inputs1map3	6	6	[51,51]
4	inputs1map4	8	20	[51,51]
5	inputs1map5	12	20	[51,51]
6	inputs2map1*	6	20	[51,51]
7	inputs2map2	6	10	[152,152]
8	inputs2map3	6	6	[51,51]
9	inputs2map4	8	20	[51,51]
10	inputs3map5	12	20	[51,51]
11	inputs3map6	12	6	[51,51]

*Five extra random experiments were conducted with the input data from *inputs2map1* to study the consistency of the results in a setup where the orders are assign randomly. The results are displayed on IV

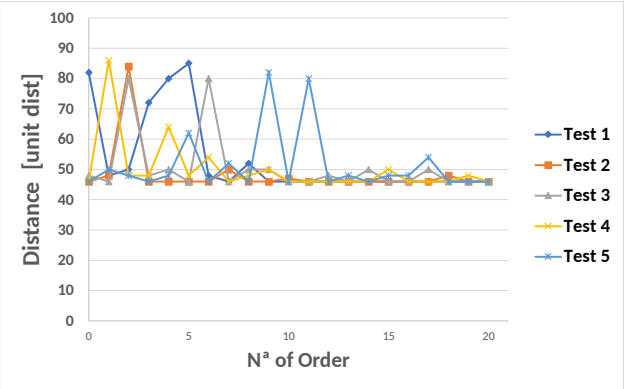

Fig. 4: Average distance performed by the agents in tests 1-5

a maximum difference of roughly 10% between the highest and lowest test run results, namely the first and the second test run.

In Fig.[5] it is shown how the map size has an impact on the average distance covered by each agent in each different map. It can be said that an increase of the number of orders in a map does not influence the average distance as greatly as the dimensions of the map. This is a pattern which can found on both scenarios. However despite the [152x152]n map being 9 times bigger than the [52x52] the average distance is always the close to being 2 times the value of the smaller map on average. This is due to the fact that when an agent delivers an order is already potentially closer to the next order pick up point, thus it could have to cover a considerably shorter distance to pick up to second or third order assigned to it.

TABLE IV: Scenario 2 Test 6 random order distribution effects

Test Nª	avg. Distance performed	RSD
1	48.24	4.76%
2	53.10	4.83%
3	50.48	0.34%
4	50.00	1.28%
5	51.43	1.54%

Secondly in order to analyze and compare the performances of the two scenarios, four equal tests have been conducted.

TABLE V: Results of experiments from scenario 1 - non collaborative setup

Agents	Orders	AvgLoss	Sim time[s]	Avg MaxDist	STD.dev	Map Size
6	20	80.58	276	119.86	40.93	[51,51]
6	10	128.58	507	266.70	128.58	[152,152]
6	6	19.03	63	59.33	2.22	[51,51]
8	20	57.96	237	97.24	34.44	[51,51]
12	20	43.58	219	82.86	21.37	[51,51]

TABLE VI: Results of experiments from scenario 2 - collaborative setup

Agents	Orders	AvgLoss	Sim time[s]	Avg MaxDist	STD.dev	Map Size
6	20	155.32	374	195.62	80.99	[51,51]
6	10	205	501	350	87.81	[152,152]
6	6	100.87	199.7	141.17	24.89	[51,51]
8	20	119.70	301	160	48.48	[51,51]
12	20	85.99	276	126.29	28.90	[51,51]
12	6	66.70	159	107	29.67	[51,51]

The results of the tests can be found in Tab.[V] for the non-collaborative solution and in Tab.[VI] for the collaborative solution. The results indicate that the distance based assignment bid of scenario 1 overall yielded better results than the collaborative approach but non distance optimized proposed in scenario 2. In Fig.[6] the advantage of scenario 1 solution over scenario 2 can be clearly observed. In several cases however the collaborative approach does help to reduce the distance as shown in the aforementioned figured. However it is important to note that the total simulation time as well is more efficient in scenario 1.

As might initially be expected, longer maps require more steps to complete. We can also see from this graph how increasing the number of agents improves the overall performance of the system even if a higher number of robots also means a higher number of moving obstacles.

As discussed in previous sections, the $lambda$ parameter allows studying the distance deviation between the Euclidean distance between stations and the path actually travelled by the agents to complete the order. The results of the data extrapolation related to the λ factor of the two scenarios are

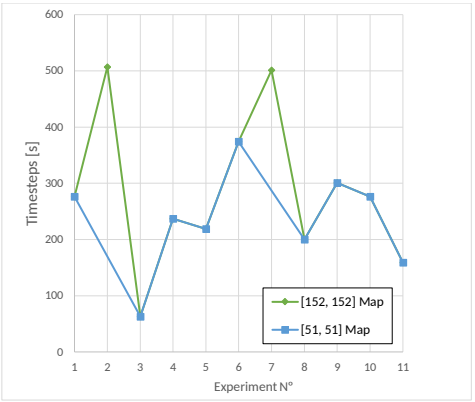

Fig. 5: Time-steps needed to complete each test

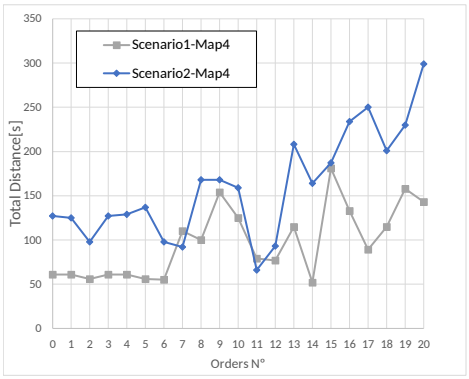

Fig. 6: Total distance operated in map4 (20 orders and 8 robots) in both scenarios

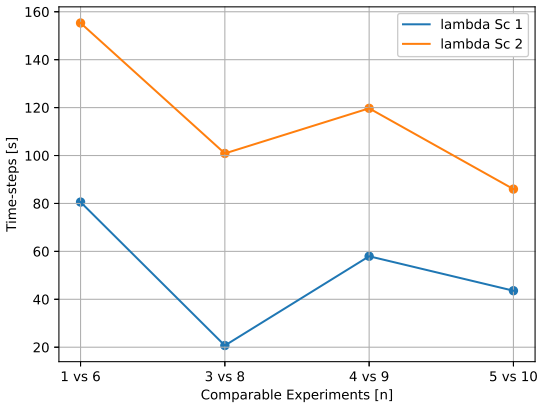

Fig. 7: Deviation parameter in each test

presented in Fig.[7]. As it can be observed the performance of scenario 1 solution yields better results compared to scenario 2. It can be highlighted how the proportion between the graphics might reference the shared loss in common obstacles and setups (the obstacles themselves, the initial position of the robots).

TABLE VII: Scenarios comparison Tests used for Fig.[7]

Sc1 Test n	Sc2 Test n	Agents	Orders	Map Size
1	6	6	20	[51x51]
3	8	6	6	[51x51]
4	9	8	20	[51x51]
5	10	12	20	[51x51]

Apart from confirming that the scenario 1 implementation is more optimal, in this environment it can be seen how even with remarkably longitudinal differences (high values of RSD) on the third experiment the first implementation still achieves better results as shown in Fig.[8] and Fig.[9.

Moreover it was important to compare the overall simulation time in a similar setting as well. Therefore based on the comparable experiments shown in Tab.[VII], in Fig.[10] the overall simualtion time performance of the two scenarios

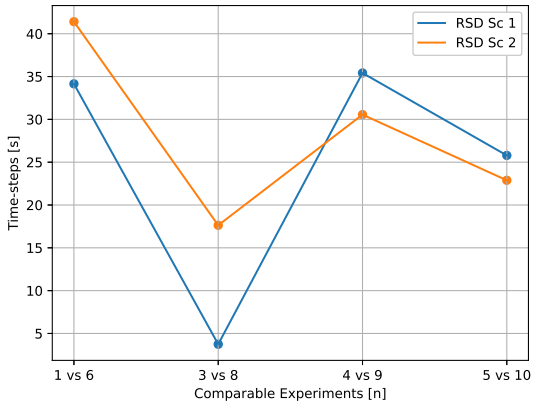

Fig. 8: Relative deviation scenario 1 vs scenario 2

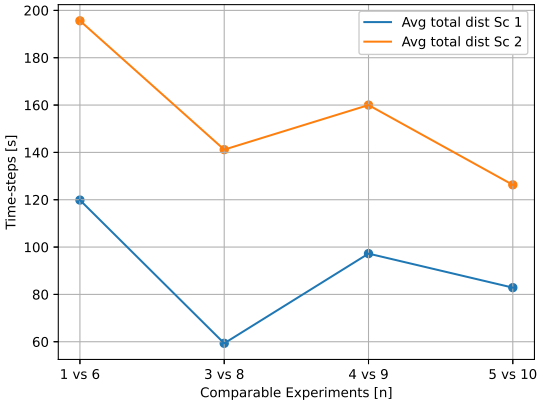

Fig. 9: Average operated distance by each agent

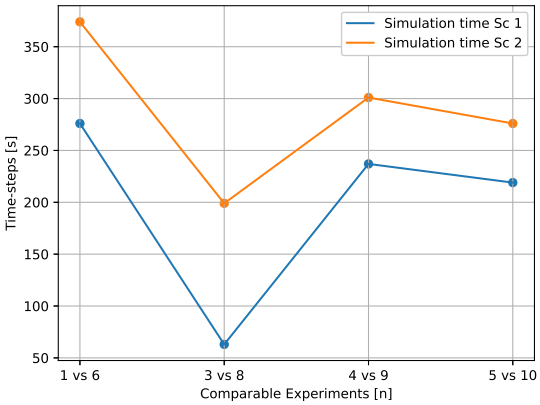

Fig. 10: Simulation time

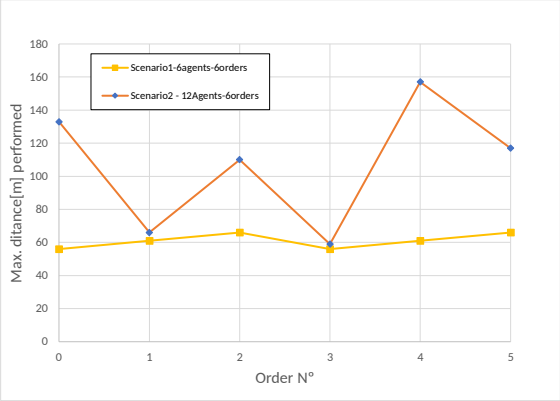

Fig. 11: Sc1 6 agents 6 orders vs Sc2 12 agents 6 orders

measured as a function of total time steps to complete the sorting of all the orders is presented. Similarly to what it was previously discussed the behavior of the simulation time confirms that scenario 1 represents a more time efficient approach.
Nevertheless it is important to note that in case of an high number of orders, scenario 1 is able to sort an higher number of orders at the same time compared to scenario 2. Therefore in order to further analyse the behavior of the two scenarios, a final comparison test was conducted. The details of it are presented in Fig.[11].

As it can be observed in the aforementioned figure, despite a map and setting which would simulate a more advantageous condition for scenario 2, the overall performance compared to scenario 1 remains inferior.

VI. Conclusion

In this paper two different robot network modeling approaches were for sorting orders within a warehouse were presented. The developed python simulator offers a reliable and scalable tool able to test maps with increasing size, agents and orders quantity.
From the conducted tests it was shown that the most efficient and reliable algorithm was the one proposed in scenario 1, where the multi-agent order sorting is non collaborative. Therefore it can be said that scenario 1 represents a more reliable solution, thus answering the initial *research question*. However it is worth noting that the second scenario offers vast improvements possibilities. A similar distance based order optimization approach could be implemented as well as more flexible meeting scheme between the collaborative agents.
Moreover in a real life scenario the optimization of scenario 2 would be more beneficial, as hardware constraints could not be neglected. For instance battery life constraints linked to the autonomy range of the robot, in that case a complete collaborative approach for every order could improve the overall performance of the agents during one charge discharge cycle as a smaller distance would be on average covered.

References

[1] Quarterly Retail E-COMMERCE Sales 2022, US Department of Commerce
[2] Warehouse Automation Explained: Types, Benefits Best Practices, Oracle, consulted on 06/2022
[3] Distributed Systems
[4] Network Robot Systems. Alberto Sanfeliu, Norihiro Hagita, Alessandro Saffiotti, Universitat Politècnica de Catalunya, ATR Intelligent Robotics and Communication Labs & ATR Media Information Science Labs. 2008
[5] Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics. Sharma, Harshita Alekseychuk, Alexander Leskovsky, Peter Hellwich, Olaf Anand, R.s Zerbe, Norman Hufnagl, Peter. (2012)
[6] Multi-Agent path planning in Python
[7] Tcod library repository
[8] A Look at the U.S. Commercial Building Stock: Results from EIA's 2012 Commercial Buildings Energy Consumption Survey, 2015. US Energy Information Administration. article