Communication Papers of the of the 17th Conference on Computer Science and Intelligence Systems pp. 321–328 DOI: 10.15439/2022F208 ISSN 2300-5963 ACSIS, Vol. 32

Towards developing a cyber-physical warehouse system for automated order-picking for online shopping

Melike Oruc
Department of Computer Science
Dokuz Eylul University
melikeoruc97@gmail.com

Onur Berker Alhas
Department of Computer Science
Dokuz Eylul University
alhasonur@gmail.com

Goksun Beren Usta
Department of Computer Science
Dokuz Eylul University
goksunberen@gmail.com

Hussein Marah
University of Antwerp &
Flanders Make Strategic Research Center
hussein.marah@uantwerpen.be

Baris Tekin Tezel
Department of Computer Science
Dokuz Eylul University
baris.tezel@deu.edu.tr

Moharram Challenger
University of Antwerp &
Flanders Make Strategic Research Center
moharram.challenger@uantwerpen.be

***Abstract*—In recent years, Cyber-Physical Systems (CPS) have been studied in various application areas. Since there are many robots that need to implement both physical and cyber sides, interact with each other and the environment, and the system has to cope with many faced difficulties, a warehouse system for automated order-picking for online shopping seems to be very suitable for being modeled and implemented as CPS. In this study, the planning and control of mobile robots in a store warehouse is defined from the perspectives of obstacle avoidance, collision detection, and solving the shortest path problem. Accordingly, a simulation environment is designed and a layered CPS development model is proposed. In the simulation environment, both object avoidance and collision detection algorithms are introduced and implemented, different shortest path-finding algorithms are adapted and implemented and their performances are evaluated.**

***Index Terms*—Online shopping, Mobile robot, Simulation, Webots, Cyber-Physical systems**

I. Introduction

Cyber-Physical Systems (CPS) are complex systems that incorporate cyber and physical components that communicate and interact with each other [5, 3]. They collect environmental data through sensors and affect the environment via actuators. CPS has pervasive applications in various fields such as health, transportation, manufacturing, etc., making our lives easier by automating many aspects.

In today's world, online shopping is spreading everywhere in society. [17]. Especially during the COVID-19 pandemic, the increasing rate of online shopping [22] started to create an extra workload for warehouses' employees. In order to reduce this workload, it's necessary to automate the processes in warehouses by using robots and smart systems. Robots in warehouses are responsible for picking and delivering packages from/to the right destination. The robots must take the shortest possible path to collect/deliver the products to avoid time loss and save energy[26]. Therefore, path-finding algorithms such as Greedy [12] and A* [13] were used to find the shortest path. Since CPS interact with the environment, another problem that may be encountered is the possibility of robots hitting any obstacle that may come their way in the warehouse. During the movement, it is necessary to prevent the robots from colliding not only with an obstacle, but also with each other. For this, a master robot transmits the information it receives from the environment to other robots, enabling them to recognize the environment dynamically and direct their movements.

The presented study identifies the planning and control of mobile robots in a shop warehouse by deploying several shortest path algorithms and comparing the performance and efficiency of these algorithms regarding the number of destinations or nodes in the simulation environment. Also, the implementation focused on the autonomy of robots (i.e., obstacle decisions) and free mobility (i.e., navigation and collision avoidance). Therefore, we try to examine and provide answers to the following research questions:

- What does the simulation environment, specifically Webots, can provide for implementing CPS?
- What are the suitable shortest path algorithms for navigation and controlling robots?
- How can the simulation environment be converted to real CPS implementation and improve the design and implementation phases?

This paper is organised as follows: Section 2 gives the related works in this field. Section 3 describes the components and technologies we used in this study. Layered CPS development model is introduced in Section 4. Section 5 is about our preliminary findings by early experimental simulation studies. Section 6 gives information about future work and concludes the paper.

II. Related Work

In the era of massive online shopping, warehouse management has become an essential and crucial factor for ensuring safe and timed delivery. The operations in a warehouse are usually intensive and require significant effort, organization, management, accuracy, and speed to deliver packages in a reasonable time frame. Also, a warehouse requires a large space to allow vehicles to maneuver around the racks [4]. In a warehouse, autonomous robots can play a vital role in picking and delivering goods while ensuring time constraints and safety requirements (collision-free, obstacle detection, and energy efficiency). Thus, moving to partially-to-fully automated operations has been considered a priority for warehouses and big retailers in the last few years. However, implementing such intelligent systems requires using appropriate technologies. Several studies, approaches, and implementations for building autonomous robots have been introduced. Also, simulations for building such systems have been considered too. Simulation studies can be helpful to get deep insights and estimations of the requirements and performance before building and implementing the system.

In the literature, there are many implementations for robots in warehouses. For example, in the paper presented by [10], they implemented an autonomous robot equipped with RFID. The RFID reader is integrated with the PIC microcontroller as the main component. Also, a servo-motor that has an infrared sensor is used to follow the line. This project is aimed to build an autonomous robot with an RFID application. The implementation of the system used the Assembly languages. As a result of the implementation, the robots read the tag on the items to identify them, pick up those items, and then navigate to the desired location to store the item.

For most warehouses, there are many orders to be fulfilled in a given time constraint. As a result, task assignment for robots in such systems becomes one of the most crucial corners stone of the system. Thus, efficient algorithms for path-finding can boost the system's performance and achieve good results. The algorithm presented in [6] gives us a feasible adaptive task assignment algorithm for a real-time system to solve task assignment problems. With the given algorithms, such systems can be robust enough to meet the needs of huge order loads dynamically while being flexible at the same time.

The authors of the study [18] focused on investigating the robotic mobile fulfillment system (RMFS) in warehouses where there is high-density storage due to the limited space and high costs. The main focus was to assign tasks between workstations and the storage area, and this process is composed of three phases: task assignment, path planning, and traffic control. A simulation environment was implemented to evaluate the system requirements, and the results showed that 10% of storage space could be saved using the energy level. The study presented in [7] focused on the order processing problem in order to determine the time of picking and delivering the packages by the mobile robot. The topic of improving the throughput performance of Automated guided vehicles (AGV) was studied in the [24], and to achieve this, they introduced a mathematical model. The model was designed with genetic algorithm, and the main aim of this model is to achieve the shortest order completion.

Simulation of real warehouse or manufacturing environment systems has also been taken into account in the literature. For example, A simulation framework designed with ROS Gazebo simulator to simulate transport systems based on automated guided vehicles (AGVs) has been built in [19]. This framework is used for simulations by deploying algorithms and different policies in the control system. The adaptation to the Just-in-Time (JIT) concept was discussed in the [16]. In addition, a simulation model in a job shop environment was developed to improve transportation efficiency, and a dispatching algorithm was deployed in vehicles that move through stations. The paper [11], presents a strategic simulation model for comparing common fixed layouts that deploy the shortest path approach where collision avoidance is the central focus. In terms of transport capacity, the results of simulation experiments showed that dynamic free-ranging has high potential. Anyhow, the use of simulation environments have helped developers to build more dynamic and error-prone cyber-physical system (CPS) due to the favor of the valuable insights that have gained from simulation experiments.

Some of the open source simulators that have been seen in the previous works are like (Gazebo, Webots, Simbad, USARSim, RoboDK, MRDS and MisionLab) [23]. In this paper, the simulation environment has been designed and developed with Webots. In addition, an layered CPS development model is proposed to create a warehouse storage system automated order-picking for online shopping in the simulation environment. With the simulation environment and proposed architecture, both object avoidance and collision detection algorithms are introduced and implemented, as well as different shortest path-finding algorithms are applied and evaluated performances according to solving the traveling salesman problem for picking items within the optimum time.

III. Background

This section indicates the background information which relates to the components in the scope of this study. The section clarifies which algorithms such as Dijkstra, A-star, Travelling Salesman Problem and which tools such as Webots, e-Puck are used in the study.

A. Warehouse System and Order-picking for Online Shopping

Especially in the pandemic period, technical solutions to improve the manual order picking process have gained more and more importance. One of them is to do the order picking process with robots. While the ultimate goal is real robotics, it is often very beneficial to run simulations before researching with real robots. This is because simulations are easier to set up, cheaper, and more convenient to use than real robotics.

Webots: Webots is an open source robot simulator produced by Cyberbotics[20]. It is written by the developers of the open source program called Khepera Simulator, which is based on Khepera Simulator. Any type of mobile robot can be simulated using Webots. So it is really useful for the development of advanced robotics projects. The Webots library includes distance sensors, light sensors, cameras, accelerometers, touch/pressure sensors, and GPS, and the user can process data from these sensors.

e-Puck: The e-puck robot was designed by Francesco Mondada and Michael Bonani at the Swiss Federal Institute of Technology in 2006. It is an educational robot that has helped generations of students learn about embedded systems and robotics. It has a small differential wheel, 7.4 cm in diameter, 4.5 cm in length, and weighs 150 g. It has 8 infrared sensors that measure near the light at a range of 4 cm and the proximity of obstacles. It also contains a 3-axis accelerometer, three microphones, a speaker, a 640x480-pixel color camera, and a Bluetooth interface for communicating with a host computer. The current version of the e-puck model includes distance sensors, light sensors and many more features such as a camera [21].

B. Shortest Path Algorithm

Throughout history, it has always been a question asked how people can travel from one city to another in the shortest time. This is a very simple question in everyday life that almost everyone can easily understand and find a solution to. But when the input data is large, solving the problem may not be so easy. Machines can calculate this process more accurately and faster for us. As a result, the shortest path problem is considered a basic topic in the field of computer science. The shortest path problem is defined as the shortest distance movement from a given starting point to a given target point in a given environment with obstacles at different locations.

Travelling Salesman Problem (TSP): The Traveling Salesman, which is one of the well-known NP-hard optimization problems, was formally stated by Irish mathematician William Rowan Hamilton and British mathematician Thomas Kirkman in the nineteenth century. TSP is a problem in which the distance between "n" cities is known, it is aimed to find the optimal route in which the total distance traveled during the tour is the shortest, based on the principle of returning to the starting point, provided that each city is visited only once [14]. In short, the goal in TSP is to find the shortest path through each city in a given set of cities.

Greedy Algorithm: When computer scientist and mathematician Edsger Dijkstra tried to calculate the minimum spanning tree, he came up with the greedy approach. The greedy approach means selecting the option or solution that appears to be the greatest at the time. [12]. It merely means selecting the option or solution that appears to be the greatest at the time. To reach the optimum solution, the main purpose of this is to find the best among the locally optimal solutions for each sub-task. Thus, the optimal solutions found for each sub-task will provide the optimal solution for the main task.

A-star Search Algorithm: The A* algorithm is a heuristic search algorithm for finding the shortest path between two targets. It is considered an expansion of the Dijkstra method, however, it uses a heuristic to determine the best solution to save runtime. The algorithm was first published in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael. [13].

C. Cyber-Physical Systems (CPS):

Cyber-Physical Systems encompass any structures that facilitate communication and cooperation between the physical and cyber worlds. The fundamental goal of CPS is to meet the flexible and dynamic needs of production while also increasing industrial efficiency and productivity. "Cyber-physical systems" are defined as "the integration of physical processes and computation." This gives the production process a whole new level of control, monitoring, and efficiency [25].

IV. Layered CPS development model for a warehouse system

Cyber-physical systems (CPS) are enabling technologies that connect the virtual and physical worlds to create a genuinely networked world in which intelligent objects communicate and interact. They are also self-contained onboard systems having sensors to sense their surroundings and actuators to control physical processes. Because they can receive and process data, they can control certain tasks themselves, so CPS can enable communication between humans, robots or even products. In the traditional warehouse system, many processes such as product picking and delivery are done by people. However, under the pandemic conditions that affected the whole world, this has been even more costly in terms of both resources and time. That's why there is an understanding of making smarter with robots in the emergence of this project. This understanding also lies in the use of the CPS.

This section describes in sequence the steps of developing a cyber-physical warehouse system for automated order picking for online shopping. This study aims to collect the products ordered by robots through online shopping in a warehouse environment and to minimize the time and cost. For this, some shortest path algorithms were produced and Java was used as the programming language. Although this study is intended to be implemented in the physical environment in the future, it was carried out with the help of a simulator as it would be much easier and more convenient to implement in the simulation environment at the first stage. The architecture of the proposed approach is shown in Figure 1.

A. Creation of The Simulation Environment on Webots and Inclusion of e-Puck Robots

In this study, the Webots simulator is used to model the robots and the environment. The Webots allows the creation of obstacles of different shapes and sizes. It also consists of a collection of robots, sensors, actuators, and objects. The environment in Webots is called the "world" and robot or object are created inside this "world". For this study, a 3x3 world was created, then The e-puck was added to this "world".

The e-puck robot is equipped with a wide variety of sensors and actuators such as a camera, infrared sensors, GPS, and LED sensors. It also has two wheels, each controlled by a separate servo motor, enabling it to move. Each motor is equipped with an encoder that counts the pulses sent to it. Encoders are used to obtain the true motor rotation by counting the number of ticks in a wheel. It has the ability to travel

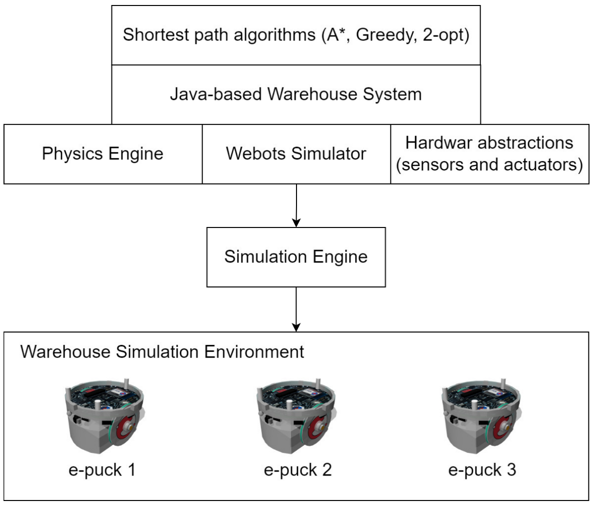
Shortest path algorithms (A*, Greedy, 2-opt)
Java-based Warehouse System
Physics Engine
Webots Simulator
Hardwar abstractions (sensors and actuators)
Simulation Engine
Warehouse Simulation Environment
e-puck 1
e-puck 2
e-puck 3

Fig. 1. The overview of the warehouse simulation architecture

forward, backward, and right or left. The wheels' maximum speed is 1000 steps per second, or one wheel revolution every second [2].

Robots drive forward and backward if the values give for the wheels are positive and negative respectively. The left wheel speed must be lower than the right wheel speed for the e-puck robot to turn left, and the right wheel speed must be lower than the left wheel speed for the robot to turn right. Functions of a derived class called $RotationalMotor$ from Webots' $Robot$ node are used to move the e-puck robot as mentioned above.

B. Implementing Obstacle Avoidance for e-puck

Obstacle avoidance can never be ignored when aiming to develop a cyber-physical warehouse system for automated order picking for online shopping. Because there are many obstacles such as shelves, baskets, and objects that the robot can hit in the warehouse environment. To create the obstacle avoidance algorithm, we need to read the values of the 8 infrared sensors of the e-puck robot and activate its two wheels. The implementations for the movements of the wheels were explain in the previous section. In this section, implementations for obstacle avoidance are described.

The e-Puck has 8 IR sensors that measure the proximity of obstacles or the intensity of infrared light at a distance of 4 cm from the environment. These are named from ps0 to ps7. Figure 2 shows how the IR sensors are positioned on the e-puck. It receives as input the IR sensor values. The algorithm is created according to these input values. To get these input values, import directives must be added for the $DistanceSensor$ to use the corresponding API to the Controller.

When the 2 IR sensors ("ps1", "ps0") located on the right front of the e-puck detect the obstacle, it turns their direction to the left and continues towards the target when the obstacle is out of the distance range. Likewise, when the 2 IR sensors

[1]e-puck: https://cyberbotics.com/doc/guide/epuck

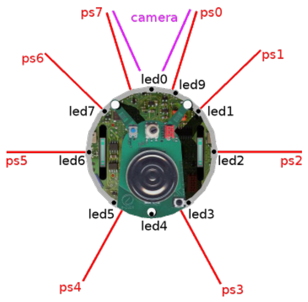

Fig. 2. Sensors, LEDs and Camera of e-puck[1]

("ps7", "ps6") on the left front detect an obstacle, it turns right and continues toward the target when the obstacle is out of the distance range.

In this case, when the e-puck robot detects an obstacle, it first stops, turns in the opposite direction, and takes action. When it is more than 4 cm away from the obstacle it detects, it adjusts its angle and continues to go towards the target again. Although this does not seem like a waste of time for a single e-puck robot and a single obstacle, it can cause a lot of wasted time when considering multiple obstacles and robots in the warehouse environment. For this reason, an obstacle avoidance algorithm was designed gradually.

Algorithm 1 An algorithm of wheel speeds for obstacle avoidance

```
1: $base \leftarrow 3$
2: $count \leftarrow 0$
3: $total \leftarrow 0$
4: for $iteration = 0, 1, 2, 3$ do
5:     if $distance\_sensors[iteration].getValue() \leq 1000$ then
6:         $total \leftarrow total + iteration$
7:         $count \leftarrow count + 1$
8:     end if
9: end for
10: if $count > 0$ then
11:     $leftSpeed \leftarrow base + (total/count - 2) * 3$
12:     $rightSpeed \leftarrow base - (total/count - 2) * 3$
13:     $setVelocity(leftSpeed)$
14:     $setVelocity(rightSpeed)$
15: else
16:     $setVelocity(base)$
17:     $setVelocity(base)$
18: end if
```

The "base" variable is used to store the base speed of the robot. The "total" variable is used to save the sum of the weights of the distance sensors that detect an object. The "count" variable saves the number of distance sensors that caught an object. The distance sensors of the e-puck robot can take values between 0 and 1000[2]. If the threshold sensor value

[2]https://cyberbotics.com/doc/reference/distancesensor

rises above 1000, an obstacle has been detected. Conversely, if the sensor's value is 0, there is no obstacle detected.

The weights of the distance sensors which are assigned to the indexes of a loop, when the total is divided by count give a value between 0 and 4. The left-most object that the robot can detect will give 0 and the right-most object that the robot can detect will give 4. Now if we subtract 2 from the leftmost will give -2 and the rightmost will give 2 giving us the possibility of detecting which side the object is on concerning the robot.

Then we multiply this value by another weight. This weight was found by trial and error with different values for this the robots turning speed and reaction time will change. To get the separate speeds for left speed and right speed the values we get are subtracted from the base speed. The resulting values are sent to the $SetVelocity$ function of Webots' $Motor$ library. $SetVelocity$ is used to control the speed of the robot. Thus, it provides a more effective working opportunity for obstacle avoidance of the robot without slowing down. The related demonstrating video can be found on https://youtu.be/siQvDc7AbG4

C. Implementing Collision Avoidance for e-Puck

Collision avoidance is a critical feature that provides accurate, fast, and dependable warnings before a collision occurs. In particular, collision avoidance in the warehouse environment for order-picking for online shopping robots is an area of work that should be considered. Because if more than one robot does not communicate with each other, possible conflicts may occur, resulting in loss of time and work. As a result, a collision avoidance algorithm has been devised in this study, allowing e-puck robots to reach their destinations without colliding with each other or obstacles.

Improvements have been expressed in this section. The improvement was initiated by adding two more e-pucks to the "world" and controllers for them. The collision avoidance algorithm introduced in the previous section was applied to these two e-puck controllers.

Each e-puck used has a priority over the other. This situation is used in this study with Webots' "$Supervisor$" library. The supervisor has omnipotent powers, including the ability to change the environment and deliver messages to robots. In Webots, it's linked to a controller program. The Supervisor controller, unlike a conventional robot controller, will have access to advantaged operations. Any robot can be turned into a supervisor when the "supervisor" field is set to TRUE in Webots. In this way, it can provide access to other e-puck robots. Considering this situation, the algorithm has been developed.

To explain the algorithm; in the beginning, three e-puck robots are moving toward their destination. The situation of collision avoidance occurs when the Euclidean distance between the e-puck robots is less than 5 cm. Any e-puck robot, when there is a possibility of collision with another e-puck robot, allows the passage of whichever has priority by messaging between them. When the distance between them eliminates the possibility of collision, the other e-puck robot continues to go to its target. The algorithm continues until all the e-puck robots reach their targets.

Algorithm 2 An algorithm of collision avoidance

```
1: function START
2:     if CheckDistance then
3:         MOVE(0, 0)
4:     end if
5: end function

6: function MOVE($leftSpeed, rightSpeed$)
7:     setVelocity(leftSpeed)
8:     setVelocity(rightSpeed)
9: end function

10: function BOOLEAN CHECKDISTANCE
11:     if $Distance(getPosition(Robot_2), getPosition(Robot_1)) < 5$ then
        return $TRUE$
12:     else
        return $FALSE$
13:     end if
14: end function
```

Within the scope of this study, 3 e-puck robots were used as mentioned in this section. However, in Webots, many robots can be added to the "world", and controllers for these robots can be added. Collision avoidance status can be implemented for many robots by providing priority conditions in the controller of each. The related demonstrating video can be found on https://youtu.be/h03AbXaXYV4

D. Path Planning with Shortest Path Algorithm for Travelling Salesman Problem

Finding the shortest path in the presence of obstacles, referred to as the shortest path problem, is one of the fundamental problems in path planning. As this problem occurs in many industrial applications, it also plays an important role in the warehouse ordering robot. Because in this way, order picking for online shopping at a warehouse can effectively reduce the cost and time. Therefore, improving the route planning algorithm of the online ordering robot has a significant impact on improving transportation efficiency.

The robot receives the coordinates of all ordered products, then automatically starts moving from the starting position to the first calculated product position to complete the loading task. After all the products it needs to collect are finished, it returns to its starting position and completes its process. Therefore, this scenario can be considered a "Traveling Salesman Problem". The Traveling Salesman Problem defines a salesman who must travel between a certain number of cities. The order of the cities to be visited does not matter, as long as it is visited in one go and ends in the starting city. Each of the links between cities has one or more weights that can represent distance, time or price cost. The real problem is to find the shortest path that starts at one point, travels through all needed destinations, and ends at the point where it started again. In the study, we test the robot's path planning through 2 algorithms for Travelling Salesman Problem. These are Greedy Algorithm and A* Algorithm. The related demonstrating videos can be found on https://youtu.be/hgCQA_o4QGs and https://youtu.be/w-z1-IhCY68, respectively.

1) Input Data Set: Within the scope of this study, the coordinates of the products that need to be collected in the warehouse environment for order picking for the online shopping robot are read from the file. The coordinates in this file are randomly generated in Python. It takes the starting position of the robot from Webots. This is also the final position.

2) Greedy Approach Algorithm for Order Picking Robot: The greedy algorithm is a strategy that contends that the best choice should be chosen under present conditions, without regard for the long-term consequences of solving a problem. The algorithm's main goal is to find the shortest path possible. Starting with the initial node, only the one with the smallest distance among the available nodes is chosen. After each selection, the selected node is marked not to be re-selected. As a result, the previously visited node is no longer visited. When there are no more nodes to visit, that is, all nodes have been visited, it returns to the starting point. This is the greedy method's idea for the Travelling Salesman Problem (TSP) solution.

The way the algorithm works for this study is as follows: The coordinates of the products that the order-picking for online shopping robot needs to collect are read from the file through the controls. Since there is a TSP problem, the start point and end point of the robot must be the same. This start and end position is automatically obtained from Webots. That is, the start point and end point of the robot can be changed dynamically. Since Greedy is an algorithm, it always chooses the closest one among the nodes it can go to. The distance was calculated with "Euclidean Distance". The weight of each node visited is changed to an "infinite" value since it will not be visited again. When all the coordinates to be visited are finished, that is, when all the ordered products are collected, the order-picking for the online shopping robot returns to its starting position and the algorithm ends.

3) A Algorithm for Order Picking Robot:* The A* algorithm is a search algorithm that searches for the shortest path between the starting and ending states. In the Greedy method, we only consider our estimated distance to the target and move in the direction where it is the least. The A* algorithm, unlike the Greedy method, takes into account heuristic cost and actual distance. This is accomplished by retaining a tree of paths that originate from the initial node and extending those paths one edge at a time until the termination requirement is fulfilled. The A* algorithm must decide which of its paths to extend at each iteration of its main loop. It does so based on the path's cost as well as an estimate of the cost of widening the way to the goal.

"Manhattan Distance" and "Euclidean Distance" can be used when calculating the cost of the road. In this study, both methods were tried on the algorithm and the results were examined. However, in [1], it has been seen that "Manhattan Distance" is recommended in case of high-dimensional data, and "Euclidean Distance" is recommended in case of lower data, as in our study. Therefore, "Euclidean Distance" was used while calculating the cost of the road [1, 9].

$f(n) = g(n) + h(n)$
$f(n)$ = total estimated cost of the path through node n
$g(n)$ = cost so far to reach node n
$h(n)$ = estimated cost from n to goal. This is the heuristic part of the cost function, so it is like a guess.

But the A* algorithm does not have a suitable scenario for the Travelling Salesman Problem (TSP) solution. In this study, the A* algorithm was adapted to TSP. For this, the robot went from the starting position to a point and then the A* algorithm was run. Thus, the scenario was made suitable for TSP. He made several attempts to decide where he should go first from his starting position. By making the robot go to the nearest node from the starting point, the A* algorithm was run. Then, on the contrary, this process made the robot go to the farthest node from the starting point, and the A* algorithm was run.

When we compared the results, it was observed that going to the farthest node from the starting point of the robot and then running the A* algorithm gave better results. For this reason, the warehouse ordering robot has been adapted to pick up the product at the furthest node from the starting position, then visit the nodes that need to be intuitively visited using the A* algorithm, collect all the ordered products, and finally return to the starting position.

V. Preliminary findings by early experimental simulation studies

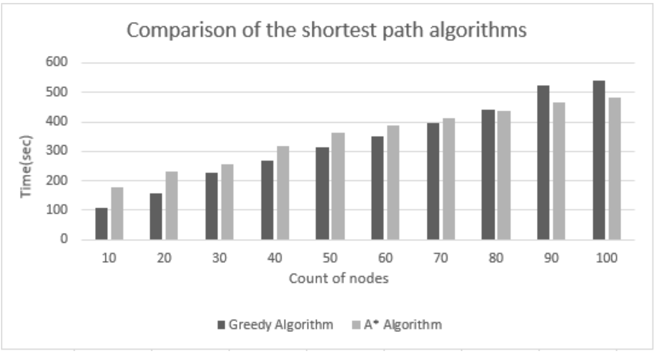

Fig. 3. Evulation of the algorithms with respect to time criteria

In this study, A* and Greedy Algorithm will be applied in the path planning of a cyber-physical warehouse system for automatic order selection for online shopping. We made some changes to adapt these algorithms to our scenario for the warehouse system. The results we obtained from the study of these two algorithms and the comparison of these results are shown in Figure 3.

The bar chart provides information about the running times of the simulations based on the number of nodes. The number of nodes here represents the number of items to be retrieved in sequence. The number of nodes was randomly generated from 10 to 100, and the values in the graph were created by running the simulation 20 times for both algorithms and taking the average. The main reason for testing the number of nodes between 10 and 100 here is to compare the two shortest path algorithms based on the complexity of the scenario. In both shortest path algorithms, the result has been reached, but it is aimed to decide which is better in which situations. In this section, it is explained which algorithm should be used in which situations for the scenario according to the simulation completion times of the two shortest path algorithms.

Overall, the Greedy Algorithm's time to complete the simulation initially performed better than the A* Algorithm. However, at the end of the period, the A* algorithm worked faster. The most significant finding is as the number of nodes increases, the running time of the A* algorithm started to give faster results compared to the Greedy algorithm. When the number of nodes is 10, the greedy algorithm completed the path in around 100 seconds, while the star algorithm's running time was just under 200 seconds. It can be clearly seen from the bar chart that the running times of algorithms began to approach each other with the increase in the number of nodes. When the count of nodes reaches 80, the running time of the A* algorithm was slightly above 400, giving a faster result than the Greedy algorithm.

According to the result of the early experimental simulation studies, when the robot receives the order, if the number of products to be collected by the robot is less than 80, it is more efficient to use the Greedy algorithm because it works faster. When the number of products is more than 80, the A* algorithm can be used since A* performs more successfully. However, these findings have not been confirmed by any statistical test. Therefore, in order to make these results more reliable, the number of trials should be increased and the results should be tested statistically. Although the findings obtained in this state are guiding, it will not be appropriate to make a definitive judgment.

VI. Discussion

The robot travels through the points most shortly, following a certain route. One of the first problems encountered while trying to solve this step was the selection of the method used to solve the shortest path problem. It was noticed that when using the Greedy algorithm to find the shortest path, it often misses the shortest distance on the overall route, as expected since the algorithm always chooses to travel the local minimum distance. So, trying the A* algorithm and the Greedy algorithm was decided to evaluate them. Another problem encountered at this stage was that the A* algorithm was not a completely suitable method to solve the Travelling Salesman Problem, so the algorithm was modified to made it suitable for the TSP problem. After the robot went to the node with the shortest distance from the starting point, the A* algorithm was ran. With this method, it was realized that the algorithm did not show the efficiency we expected.

When we ran the A* algorithm after going to the farthest node from the starting point of the robot, more successful results were got. When the results gathered from the Greedy and A* algorithms were compared, it was observed that as the number of nodes increases, the Greedy algorithm works faster up to about 80 nodes, while adding more nodes gives slower results. This may be because Greedy runs slower since its time complexity is n^2 as it travels through all the nodes. The data structures used while implementing algorithms can also be a factor that needs to be evaluated for speed. To find a better route in the A* algorithm, prioritizing some nodes using the cost function may have provided a faster solution to finding the optimal route.

VII. Conclusion and Future Work

In this paper, we have designed and developed a simulation implementation to simulate the real behaviors of Cyber Physical System (CPS) embodied as mobile robots that operate in a warehouse or a shop. In the simulation implementation, shortest path algorithms have been designed, deployed and tested. According to the results of the early experimental simulation studies, if the number of products to be collected by the robot is less than 80, it seems more efficient to use the Greedy algorithm. However, it has been observed that A* performs more successfully when the number of products is more than 80. Besides implementing and testing shortest path-finding algorithms, object avoidance and collision detection algorithms are introduced and implemented for appropriate to the scenario and the simulation environment.

We plan to integrate the current simulation environment with an agent-based paradigm to provide a simulation implementation with smart and autonomous behaviors. Agents can provide the simulation with advanced capabilities as well as make the system actors independent, interactive and proactive. Agents can behave competitively or cooperatively in the simulation environment to form Multi-Agent Systems (MASs). The MASs may have different perspectives like a plan, interaction, organization, role, environment etc. Thanks to these different perspectives, MASs can consider the structure, behavior, interaction and environment of complex systems such as CPSs. Therefore, both agents and MASs can be an ideal alternative in the modeling and development of CPSs [8]. This as future work will be discussed and explored with a clear and obvious case study that can show reliable results.

In addition, concerning providing digital twins with agent-based simulation, we intend to extend the current simulation and integrate it with the proposed agent-based Digital Twin architecture [15]. The plan is to have a digital twin framework where services such as simulation can be run by exploiting the real-time and historical data collected from the physical components to perform several tasks (prediction, maintenance, and improvement).

References

[1] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. “On the surprising behavior of distance metrics in high dimensional space”. In: *International conference on database theory*. Springer. 2001, pp. 420–434.

[2] Marwah Almasri, Khaled Elleithy, and Abrar Alajlan. “Sensor fusion based model for collision free mobile robot navigation”. In: *Sensors* 16.1 (2015), p. 24.

[3] Tansu Zafer Asici et al. “Applying model driven engineering techniques to the development of contiki-based iot systems”. In: *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*. IEEE. 2019, pp. 25–32.

[4] Kaveh Azadeh, René De Koster, and Debjit Roy. “Robotized and automated warehouse systems: Review and recent developments”. In: *Transportation Science* 53.4 (2019), pp. 917–945.

[5] Ankica Barišic et al. “Systematic Literature Review on Multi-Paradigm Modelling for Cyber-Physical Systems”. In: *Technical Report, Zenodo (Archive)* DOI: 10.5281/zenodo.2528953 (2019).

[6] Ali Bolu and Ömer Korçak. “Adaptive Task Planning for Multi-Robot Smart Warehouse”. In: *IEEE Access* 9 (2021), pp. 27346–27358.

[7] Nils Boysen, Dirk Briskorn, and Simon Emde. “Parts-to-picker based order processing in a rack-moving mobile robots environment”. In: *European Journal of Operational Research* 262.2 (2017), pp. 550–562.

[8] Moharram Challenger et al. “Agent-based cyber-physical system development with sea_ml++”. In: *Multi-Paradigm Modelling Approaches for Cyber-Physical Systems*. Elsevier, 2021, pp. 195–219.

[9] Ge Chen, Tao Wu, and Zheng Zhou. “Research on ship meteorological route based on A-star algorithm”. In: *Mathematical Problems in Engineering* 2021 (2021).

[10] Loh Poh Chuan et al. “An RFID warehouse robot”. In: *2007 International Conference on Intelligent and Advanced Systems*. IEEE. 2007, pp. 451–456. DOI: 10.1109/ICIAS.2007.4658428.

[11] Mark B. Duinkerken, Jaap A. Ottjes, and Gabriel Lodewijks. “Comparison of Routing Strategies for AGV Systems using Simulation”. In: *Proceedings of the 2006 Winter Simulation Conference*. 2006, pp. 1523–1530. DOI: 10.1109/WSC.2006.322922.

[12] A Fitriansyah et al. “Dijkstra’s algorithm to find shortest path of tourist destination in Bali”. In: *Journal of Physics: Conference Series*. Vol. 1338. 1. IOP Publishing. 2019, p. 012044.

[13] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[14] Holger H. Hoos and Thomas Stützle. “1 - INTRODUCTION”. In: *Stochastic Local Search*. Ed. by Holger H. Hoos and Thomas Stützle. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann, 2005, pp. 13–59. ISBN: 978-1-55860-872-6. DOI: https://doi.org/10.1016/B978-155860872-6/50018-4.

[15] Marah Hussein and Moharram Challenger. “Intelligent Agents and Multi Agent Systems for Modeling Smart Digital Twins”. In: *Engineering Multi-Agent Systems*. 2022 (in press).

[16] Saadettin Erhan Kesen and Ömer Faruk Baykoç. “Simulation of automated guided vehicle (AGV) systems based on just-in-time (JIT) philosophy in a job-shop environment”. In: *Simulation Modelling Practice and Theory* 15.3 (2007), pp. 272–284.

[17] Julia Koch, Britta Frommeyer, and Gerhard Schewe. “Online shopping motives during the COVID-19 pandemic—lessons from the crisis”. In: *Sustainability* 12.24 (2020), p. 10247.

[18] Xiaowen Li et al. “A simulation study on the robotic mobile fulfillment system in high-density storage warehouses”. In: *Simulation Modelling Practice and Theory* 112 (2021), p. 102366. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2021.102366.

[19] Joaquín López, Eduardo Zalama, and Jaime Gómez-García-Bermejo. “A simulation and control framework for AGV based transport systems”. In: *Simulation Modelling Practice and Theory* 116 (2022), p. 102430.

[20] Olivier Michel. “Cyberbotics Ltd. Webots™: professional mobile robot simulation”. In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5.

[21] Francesco Mondada et al. “The e-puck, a robot designed for education in engineering”. In: *Proceedings of the 9th conference on autonomous robot systems and competitions*. Vol. 1. CONF. IPCB: Instituto Politécnico de Castelo Branco. 2009, pp. 59–65.

[22] Hoang Viet Nguyen et al. “Online book shopping in Vietnam: The impact of the COVID-19 pandemic situation”. In: *Publishing Research Quarterly* 36.3 (2020), pp. 437–445.

[23] Aaron Staranowicz and Gian Luca Mariottini. “A survey and comparison of commercial and open-source robotic simulator software”. In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. 2011, pp. 1–8.

[24] Hongtao Tang et al. “Research on equipment configuration optimization of AGV unmanned warehouse”. In: *IEEE Access* 9 (2021), pp. 47946–47959.

[25] Michael Vierhauser et al. “Towards a model-integrated runtime monitoring infrastructure for cyber-physical systems”. In: *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE. 2021, pp. 96–100. DOI: 10.1109/ICSE-NIER52604.2021.00028.

[26] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. “Path planning for the mobile robot: A review”. In: *Symmetry* 10.10 (2018), p. 450.