REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

FLEET MANAGEMENT TASK OPTIMIZATION

16011701 — Atakan TEKOĞLU
16011012 — Süleyman Ali Burak ÇINAR

SENIOR PROJECT

Advisor
Assoc. Prof. Dr. Sırma Yavuz

May, 2021

ACKNOWLEDGEMENTS

In the realization of this work, I will never forget the importance of every word he has used in my life, who shared his valuable knowledge with me for 4 months. We would like to thank Assoc. Prof. Dr. Sırma YAVUZ for her support.

Atakan TEKOĞLU
Süleyman Ali Burak ÇINAR

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AGV	Automated Guided Vehicle
PSP	Polynomially Bounded Shortest Path Algorithm
SFT	Segmented Flow Topology
A*	A Star Algorithm

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Fleet Management Task Optimization

Atakan TEKOĞLU
Süleyman Ali Burak ÇINAR

Department of Computer Engineering
Senior Project

Advisor: Assoc. Prof. Dr. Sırma Yavuz

Automatic guided vehicles (AGV) located all over the world and used by the largest companies today are widely used in factories, production centers and ports. Researches in this field are carried out in order to reach the target in the shortest way for technological devices called robots.

It is hoped that the robots represented as symbols in our project will reach the target point with the tasks given in the shortest way with two specific algorithms. Algorithms to be designed with Python language will be presented with a desktop application.

Keywords: : Automated guided vehicles,AGV, Python, robots

ÖZET

Filo Yönetimi Görev Optimizasyonu

Atakan TEKOĞLU
Süleyman Ali Burak ÇINAR

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Doç. Dr. Sırma Yavuz

Dünyanın her yerinde bulunan ve günümüzde en büyük şirketler tarafından kullanılan otomatik yönlendirmeli araçlar fabrika, üretim merkezleri ve limanlarda yaygın olarak kullanılmaktadır. Bu alandaki araştırmalar robot olarak tabir edilen teknolojik cihazların en kısa yoldan hedefe ulaşması için yapılmaktadır.

Projemizde robot olarak temsil edilen simgelerin belirli olan iki algoritma ile en kısa yoldan verilen görevlerle birlikte hedef noktaya ulaşması hedeflenmektedir. Python dili ile tasarlanacak olan algoritmalar bir masaüstü uygulaması ile sunulacaktır.

Anahtar Kelimeler: Otomatik yönlendirmeli araçlar, AGV

1
Introduction

1.1 Goal

In this project, it is aimed to reach the representative robots to the target point with the shortest route together with the task assigned to them.

The main motivation is that such applications are used by companies around the world. It is very motivating to work on the algorithms that small or large companies use as their main reference when using these applications.

1.2 Objective

Points represented as robots are expected to perform their tasks with two specific algorithms on a specific route. The aims of our research consist of the following items:

- To examine the algorithms used in this field
- Choosing two of the analyzed algorithms
- Performing selected algorithms

1.3 What is the meaning of AGV application?

Agv tools are commonly used devices in production centers. They are used to carry various loads. The loads in the workstations are taken from certain points and unloaded.

The devices move along a certain line and take and carry these loads defined for them. The usage in the automotive sector can be given as an example to these applications.

2
Relevant Literature Review

The research we have done for the algorithms that will be used in our project will be briefly explained in this section. It will be given along with references from which sources we have researched.

It means choosing a route for devices with vehicle route planning. There are certain problems in this process. These are problems such as collision and intersection.

The researches made for our project can be examined under 3 main headings.

2.1 General Routing Algorithms

Broadbent introduced his early research on how to steer robots without collision. The method he uses in his studies is based on the Dijkstra algorithm. One of the algorithms chosen in our project is this Dijkstra algorithm. [1]

Glover developed the PSP algorithm that finds the distance between two points in an area. PSP stands for segmentation shortest path. [2]

Egbelu and Tanchoco showed that an algorithm that can be designed bidirectionally can be more efficient.But in the method they developed, intersections were a problem. [3]

Daniels developed a PSP algorithm that can be used bidirectionally. According to his method, a non-conflict routing could be performed for a newly added device without changing the routes used by other devices. [4]

Huang developed a polynomial time algorithm. According to him, the shortest path was found by using time values on the nodes in the area where the devices move.[5]

Kim and Tanchoco used Dijkstra's shortest path algorithm, which could optimize the route and avoid two-way collisions.[6]

Krishnamurthy introduced a method of creating a column-forming for bidirectional routing of devices in an area.[7]

Kim and Tanchoco introduced a method called the conservative myopic strategy, in which vehicles can be managed bidirectionally. According to this method, tasks were assigned to devices only when they were empty.[8]

Kim introduced a new algorithm. According to the algorithm developed by Kim and Tanchocon, this algorithm called Q Learning, which shortens the vehicle distance, shortened the vehicle distance more.[9]

Taghaboni and Tanchoco presented a new algorithm for route planning. In this algorithm, which is described on a node basis, the potential nodes to which the vehicle can go are selected by looking at the neighboring node status. This situation repeats until there is no conflict. [10]

2.2 Routing Algorithms for Specific Domains

Tanchoco and Sinriech presented an algorithm in which all vehicles move at equal speed. According to this algorithm, there would be no collision since the vehicles were going in the same direction.[11]

Lin and Dgen developed an algorithm that has its own loops in it. The stations where the vehicles left their cargo were grouped within themselves. Vehicles were finding the shortest route for the station in their area. [12]

Sinriech and Tanchoco presented an algorithm called Segmented Flow Topology. Vehicles were divided into sections, each served by only one device. These vehicles could move in two directions in their own sections.[13]

2.3 Scheduling Algorithms

Taghboni and Tanchoco have developed an algorithm that will prevent devices from colliding along a path.[14]

The working and travel times of unloaded vehicles are presented by Selim Aktürk with an algorithm that is considered at the same time.[15]

Zhiheng Yuan worked on a project that would prevent collisions and enable devices in an area for the sake of finding out the shortest path. For this, chosen algorithm was A* algorithm. We chose this algorithm in our project. [16]

2.4 Classification of Dynamic Route Planning Algorithms

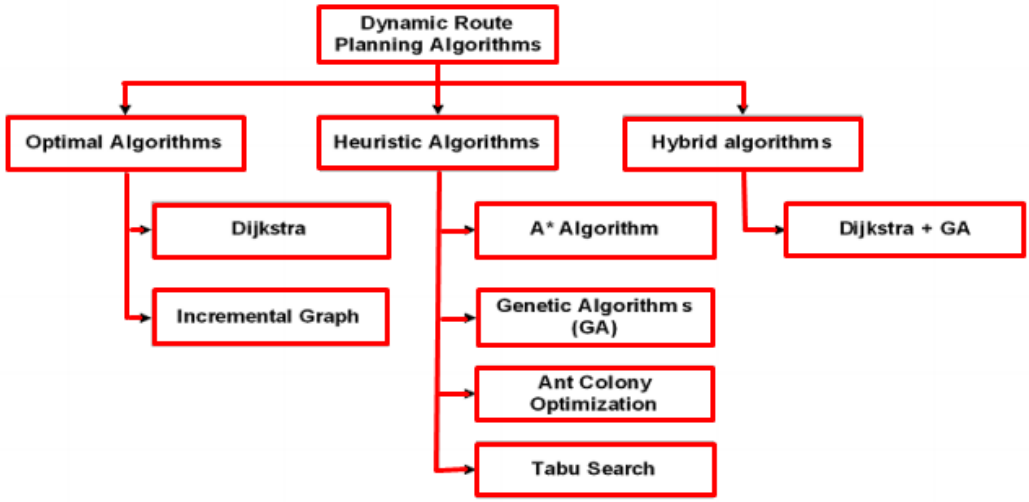

Figure 2.1 Dynamic Route Planning

For the discovery of algorithms and for a better overview of the methods used, the algorithms are listed categorically in the table above.

3
Feasibility

Necessary researches have been made to design the project in a suitable structure. The technical structure has started to be prepared. Design decisions are determined to facilitate use.

3.1 Technical Feasibility

Software and hardware feasibility are listed in part 3.1.1 and part 3.1.2

3.1.1 Software Feasibility

The elements that will be used to implement the routing algorithm and web application are listed below.

- Windows 10
- Python
- Pygame
- PyCharm

3.1.2 Hardware Feasibility

The system that software developers will use to perform the application mentioned in software feasibility is listed below.

- İlk Bilgisayar
  - Monster ABRA A5 V15.2
  - Intel Coffee Lake Core i7-9750H

  - 8GB(1x8GB) DDR4 2666Mhz
  - 4GB GDDR5 Nvidia GTX1650 128-Bit
  - 1TB 7200RPM HDD + 240GB M.2 SSD Sata 3
- İkinci Bilgisayar
  - 2 X CORSAIR 8GB Vengeance LPX Siyah 3000MHz Ram
  - ASUS PRIME B450M-K Motherboard
  - AMD Ryzen 5 CPU MPK
  - PNY GeForce GTX1650 4GB Graphic Card
  - TOSHIBA 500GB SSD

3.2 Workforce and Time Planning

After the project is determined, a literature review will be conducted, and companies engaged in fleet management in the market will be searched. Routing algorithms will be examined. The infrastructure will be determined and the skills required to implement the web application will be learned. The goal is to gain the ability to decide on the appropriate algorithm and infrastructure.

	Task Name	Duration	Start	ETA
1	Determination of Project	7 Days	15.04.2021	22.04.21
3	Literature Review	6 Days	22.04.21	28.04.21
4	Examing Market Example	6 Days	22.04.21	28.04.21
6	Searching Routing Algorithm	8 Days	28.04.21	05.05.21
5	Determination of Infrastucture	8 Days	28.04.21	05.05.21
9	Meeting	1 Day	05.05.21	06.05.21
10	General Path Topology	9 Days	06.05.21	15.05.21
11	Routing Algorithms for Specific Path	9 Days	15.05.21	24.05.21
18	Experimental Result	9 Days	24.05.21	02.06.21
17	Interface Design	14 Days	19.05.21	02.06.21
19	Software Process	28 Days	06.05.21	03.06.21
20	Report	28 Days	06.05.21	03.06.21
2	Learn Wep Development	32 Gün	22.04.21	17.05.21

Figure 3.1 Gantt Chart

3.3 Legal Fasibility

The elements to be used while implementing the application are free of charge. Free community versions of PyCharm and VSCode applications will be used. The legal compliance of the web application we will do has been investigated and no violations have been found.

3.4 Economic Feasibility

Used software, hardware and all other fees are given in Figure 3.2 and Figure 3.3

Development's Computers	Monster Abra A5 V15.2	6.500 TL
	System	7.900 TL
Operating System	Windows 10	Free
IDE	PyCharm	Free
	VScode	Free
Total		**14.400 TL**

Figure 3.2 Employee Expenses

Title	Quantity	Aylık Maaş
Programmer	2	5.500 TL
System Analyst	1	6.500 TL
Project Supervisor	1	8.000 TL
Total		**20.000 TL**

Figure 3.3 Hardware and Software Expenses

4
System Analysis

After the literature review, the algorithms to be used were started to be examined.

4.1 Dijkstra Algorithm

The Dijkstra algorithm is used to find the shortest path and as the final step is used to obtain the shortest path tree. The logic on which the algorithm is based is defined as the greedy strategy.

4.1.1 Dijkstra Algorithm Pseudo

Below are the basic steps about the algorithm.

- The value of the nodes is considered infinite at the starting point. The starting node is given a value of zero.
- The starting node is permanent, the other nodes are temporary.
- The starting point is activated.
- The temporal distances of all neighboring nodes of this active node are calculated and the weights and distances of its edges are calculated.
- According to the calculated values, if the current value for a node is smaller than the calculated value, it is necessary to update the distance.
- The temp node with the lowest value is set to active.
- Repeat items 4 and 7 until there is no permanent distance node.

4.1.2 Dijkstra Algorithm Step by Step

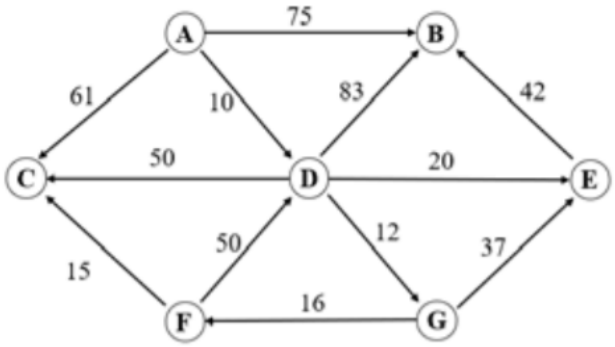

Figure 4.1 Dijkstra Example Figure

Algorithm Steps	S Collection	U Set
1	Select node A, S = < A > A→A=0 Take node A as the middle point, start searching from node A.	U=< B , C , D , E , F , G > A → B = 75,A → C = 61,A → D = 10, A → Other vertices in U = ∞ It is found that A → D = 10 has the shortest weight.

Figure 4.2 Dijkstra Step 1

2	Select node D,S = < A , D > Using node D as the middle point, start searching from the shortest path A → D = 10	U=< B , C , E , F , G > A → D → B = 93 A → D → C = 60 (less than the previous A → C = 61) A → D → E = 30,A → D → G = 22 A → D → Other vertices in U = ∞ It is found that A → D → G = 22 has the shortest weight

Figure 4.3 Dijkstra Step 2

3	Select the G node, S = < A , D , G > A → D → G = 22 Take G node as the middle point, start searching from the shortest path A → D → G = 22	U = < B , C , E , F > A → D → G → E = 59 A → D → G → F = 38 A → D → G → Other vertices in U = ∞ It is found that A → D → G → F = 38 has the shortest weight

Figure 4.4 Dijkstra Step 3

4	Select the E node, S = < A , D , E > A → A = 0 , A → D = 10, A → D → E = 30 Using node E as the intermediate node, start searching from the shortest path A → D → E = 30	U = < B > A → D → E → B = 72 (less than A → B = 75) Found that A → D → E → B = 72 (less than the previously searched A → B = 75)

Figure 4.5 Dijkstra Step 4

5	Select node B, S = < A , D , E , B > A → D → E = 30,A→D→E→B=72	The U collection is empty and the search is complete.

Figure 4.6 Dijkstra Step 5

6	Select node F, S = < A , D , G , F > A → D → E = 30, A → D → G → F = 38 With F as the middle point, start searching from the shortest path A → D → G → F = 38	U = < C > A → D → G → F → E → B = 101 A → D → G → F → C = 53 (less than A → C, less than A → D → C = 60)

Figure 4.7 Dijkstra Step 6

7	Select node C, S = < A , D , G , F , C > A → D → E = 30, A → D → G → F = 38, A → D → G → F → C = 53	The U collection is empty and the search is complete.

Figure 4.8 Dijkstra Step 7

4.2 A* Search Algorithm

The A* search algorithm is one of the most popular path finding algorithms.

The most basic feature that distinguishes the A* search algorithm from other algorithms is that it has a mind in itself. It finds the shortest path in many games and navigation very wonderfully.

4.2.1 A* Explanation

Let's consider a map. Suppose we break this map into very small pieces. Mark one of these small pieces as the starting point. Another piece is marked as the target point. Our goal is to reach the target point using the shortest path.

This algorithm selects a node based on a (Z) value. This z value is equal to the sum of the two different parameters, (X) and (Y).

- X: The cost of movement required to move to the target part starting from the part specified as the starting point.
- Y: The estimated cost of movement required to move from the part specified as the starting point to the target part. (Heuristic)

4.2.2 A* Algorithm Step by Step

```
// Initialize both open and closed list
let the openList equal empty list of nodes
let the closedList equal empty list of nodes

// Add the start node
put the startNode on the openList (leave it's f at zero)

// Loop until you find the end
while the openList is not empty

    // Get the current node
    let the currentNode equal the node with the least f value
    remove the currentNode from the openList
    add the currentNode to the closedList

    // Found the goal
    if currentNode is the goal
        Congratz! You've found the end! Backtrack to get path

    // Generate children
    let the children of the currentNode equal the adjacent nodes

    for each child in the children

        // Child is on the closedList
        if child is in the closedList
            continue to beginning of for loop

        // Create the f, g, and h values
        child.g = currentNode.g + distance between child and current
        child.h = distance from child to end
        child.f = child.g + child.h

        // Child is already in openList
        if child.position is in the openList's nodes positions
            if the child.g is higher than the openList node's g
                continue to beginning of for loop

        // Add the child to the openList
        add the child to the openList
```

Figure 4.9 A* Algorithm Step by Step

The pseudo-code above was taken from the specified address.[17]

Let's start the explanations according to the pseudo-code above.

$(F = G + H)$

- F: It is the name given to the total cost of a node.
- G: Distance from the starting point to the current point
- H: Estimated distance between current point and last point (Heuristic)

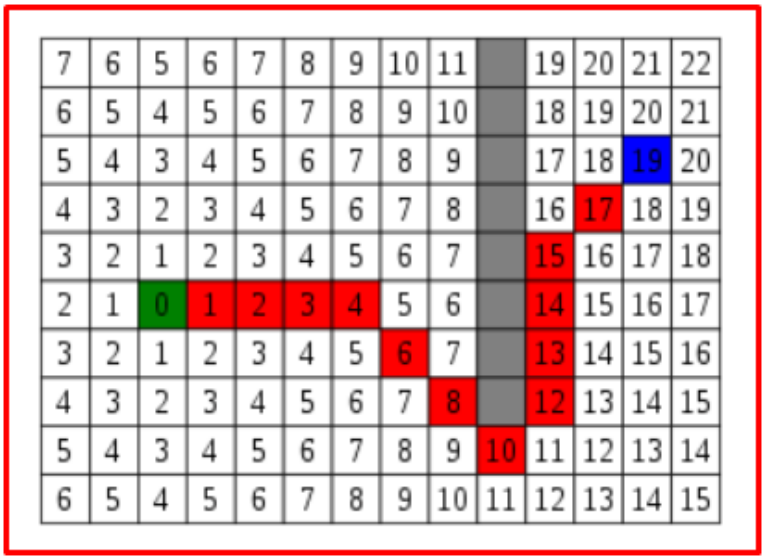

Figure 4.10 A* Example Figure

The cell indicated in green is our starting point. The cell indicated in blue is our target point. Let the cell we are in be the 4th red cell.

The distance between the current red cell (4) and the green starting cell is set to 4. Accordingly, my g value for my red current cell would be 4.

In the heuristic calculation part, we will calculate as a bird's eye view. If we go 7 steps to the right and 3 steps to the up and draw a line between the start and the end, we can apply the Pythagorean theorem.

If we go 7 steps to the right and 3 steps to the up and draw a line between the start and the end, we can apply the Pythagorean theorem. Accordingly, my heuristic makes 58. If we add the g and h values we found, our f value will be 62. A value of 62 here is the sum of our best odds.

According to this algorithm, priority is given to the cell with the best probability to reach the target cell using the f value.

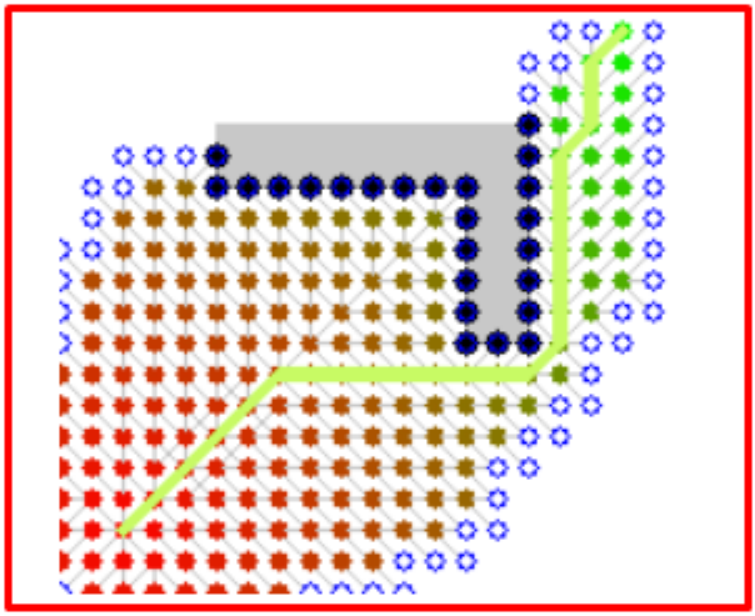

Figure 4.11 A* Algorithm Viki

5
System Design

5.1 Problem Definition

Certain elements should be included in the prepared application environment. Icons representing robots should be able to perform tasks that represent load elements in the environment. Robots must find the shortest path. While going the shortest way, it should not hit the obstacles in the environment.

According to the information above, algorithms should work and robots should perform their tasks.

Our application is coded in the development environment called PyCharm. It consists of 8 different Python files in total.

5.1.1 Spot File

This class represents the regions in our grid. It contains features such as coordinate, distance, neighbors list, color.

```
class Spot:
    def __init__(self, row, col, width, total_rows, flag):
        self.row = row
        self.col = col
        self.x = row * width
        self.y = col * width
        self.color = WHITE
        self.neighbors = []
        self.visited = False
        self.distance = 1
        self.weight = 9999999
        self.width = width
        self.total_rows = total_rows
        self.flag = flag
        self.prev = None
```

Figure 5.1 Spot Function Init

5.1.2 AGV File

This class includes variables such as robots' coordinates, route lists, occupancy status, availability, and goods transportation. This class is derived as an object in the simulation class.

```
class Agv:  # Robotlar Classımız
    def __init__(self, x, y, idNo):
        self.idNo = idNo
        self.x = x
        self.y = y
        self.color = ORANGE
        self.flag = 0
        self.target = None
        self.firstPath = []
        self.secondPath = []
```

Figure 5.2 AGV Function Init

5.1.3 Target File

This class represents the targets on our map. It contains variables such as coordinate and availability.

5.1.4 Button and InputBox File

These classes are used in our application to receive data from the user. It contains fixed structures suitable for the library we use.

5.1.5 Simulation File

This class is the class that provides the functionality and visuality in your application. It contains algorithms, framework, map and above 4 classes.

Below we will introduce a few functions included in this file.

- draw function
  - In this function, our map in our application is updated.
- handle event function

  - This function allows us to receive inputs from the user.

- check syntax function
  - In this function, the syntax of the data entered by the user is checked. Required error messages are returned for inappropriate syntax.

- do comment function
  - In this function, the suitability of the robots is checked and the commands are taken from the command list if it is not empty and executed.

- update function
  - In this function robots, targets, weighted nodes are updated.

- A* function
  - In this function, the path of the robot that is available for the given target is calculated. And the robot's path lists are updated.

6
Application

Our application is Multi Robot Optimization. In this application, the robots available take the targets and take them to the desired places. During this process, the shortest paths are determined and the robots are ensured to move optimally. From the interface, the user can add a robot and enter commands. Commands received from the user are stored for execution when appropriate. The suitability of the commands to be executed is checked. While determining the routes of the robots, obstacles and weighted nodes are taken into account and actions are taken accordingly.

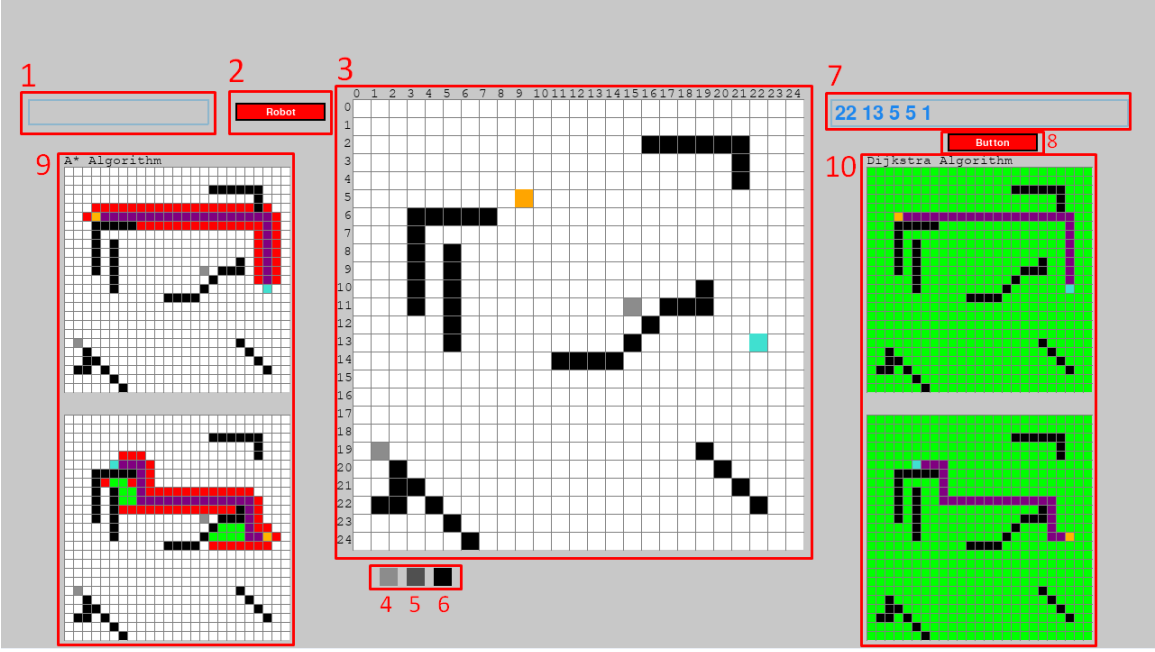

Figure 6.1 Target

The functions of the parts numbered above are as follows.

1. The part where we get the coordinates from the user to add a robot.
2. The part where the user sends the coordinates entered.

3. The map where robots, obstacles, roads, targets are displayed to the user up-to-date.
4. The part where we choose the level to add (Medium level).
5. The part where we choose the level to add (Hard level).
6. The part where we choose the obstacle to add.
7. The part that needs to be entered for single and multiple commands
8. The part where the user sends the commands entered.
9. When the system performs the incoming command, the section showing the area and path it scans for the A* algorithm
10. When the system performs the incoming command, the section showing the area and path it scans for the Dijkstra algorithm

User can update the map by choosing from section 4, 5 and 6. Choose 4 for intermediate level, 5 for hard level and 6 for obstacles. After the selection is made, it will be enough to click on the desired location to update it. It is possible to delete the places we want on the map with the right click of the mouse. The changes will be applied to subsequent commands.

Our application aims the existing robots to choose the shortest path while taking the targets to the desired location. When assigning tasks to robots, the proximity of the robots to the target is taken into account. While the robots are in motion, they detect the obstacles and robots that come in their way and change their path.Robots wait until access is provided in case of inaccessibility problem that may occur while picking up targets or taking them to the desired location.

Below are the icons we use in the application. The icons represent the robot, the target, the visited node, the node to be visited, the obstacle, the hard node, the medium node, and the easy node, respectively.

Figure 6.2 App Icons

7
Experimental Results

The applications we choose to use in our application are as follows.

- Dijkstra Algorithm
- A* Algorithm

These two algorithms have slight differences in terms of their structure.

Structural changes have been made to both algorithms that will affect performance and accuracy. In this way, by changing the two algorithms, new features have been added to these algorithms that will gain efficiency on the basis of performance.

Below we will examine these diversified algorithms in terms of performance and accuracy.

Both algorithms are handled in two ways.

7.1 Dijkstra

Dijkstra's algorithm traverses all nodes to find the shortest path and updates the distance matrix. In this way, it finds the shortest path for each node. In our second version, the algorithm is terminated and exited as soon as it reaches the target. In this way, performance losses are prevented, but accuracy is compromised.

7.2 A Star

The A* algorithm advances the target with a greedy approach using real distance and heuristic distances. Since the area where we present the application to the user is a grid, the f(n) values of the neighboring spots are equal. In our second version, the

heuristic distance (h(n)) is multiplied by 1.01 in order to increase the performance, and the orientation towards the target is provided.

7.3 Reviews

Below are explanations for the understanding of the symbols used while showing the analysis results.

Figure 7.1 Robot

Figure 7.2 Target

Figure 7.3 Path

Figure 7.4 Neighbor

Figure 7.5 Visited

Figure 7.6 Easy Spot

Figure 7.7 Medium Spot

Figure 7.8 Obstacle

Figure 7.9 Visited Medium Spot

Figure 7.10 Neighbor Medium Spot

Figure 7.11 Path Medium Spot

The icons symbolized by the gray and white colors above represent the different difficulty paths that the robot can move.

The paths symbolized by the gray icon are of 25 difficulty. The white icon represents only 1 difficulty value. The aim here is to test the response of algorithms for weighted graphs.

Now let's examine how algorithms respond in different scenarios.

7.4 Experimental Algorithm Comparison

7.4.1 Example 1

First, we will evaluate for different situations. These algorithms are Dijkstra and modified Dijkstra algorithms.

When the modified Dijkstra algorithm first reaches the destination, it terminates and the path is found. These situations create some problems. Finishing the scan when it first reaches the destination causes it to not find the shortest path for some cases (Figure 7.13). The following scenarios are designed to exemplify this situation (Figure 7.12).

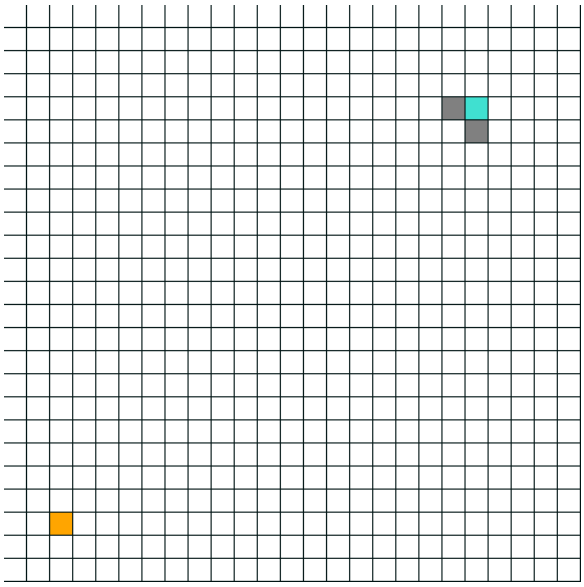

Figure 7.12 Scenario 1 Initial Grid

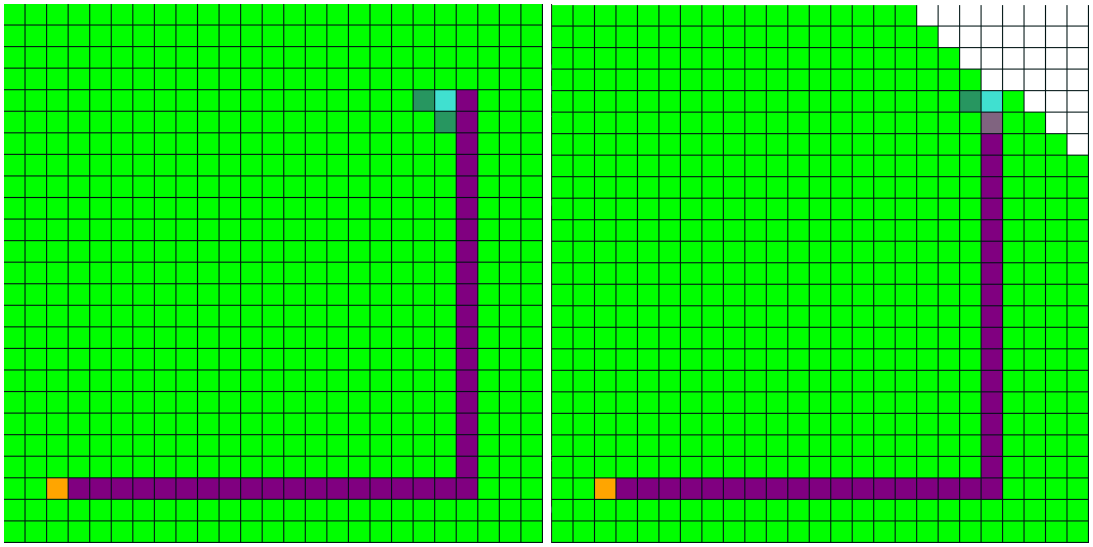

Figure 7.13 Scenario 1 Algorithms Results

	Path Cost	Steps	Time	Algorithm Name
Figure 7.13	38	629	4.001489	Dijkstra
Figure 7.13	60	585	3.510880	Tampered Dijkstra

Table 7.1 Scenario 1 Comparison Table

The running times of the algorithms, the number of steps and the length of the path they find are given in the table 7.1 above.

As seen in the examples above, Dijkstra's algorithm terminated early can find longer paths. This situation contradicts the principle of finding the shortest path, which is the main purpose of the application. Therefore, it is not suitable for use in our application.

7.4.2 Example 2

Second, we will examine the differences between A* and modified A* algorithms.

The A* algorithm finds the cost value of the destination by adding the actual distance and the estimated heuristic distance. It causes a wider area to be scanned on our map, which can only be moved to the right, left, up and down. For this reason, we made a slight change in the A* algorithm. We factored the heuristic into the calculation by multiplying it by 1.01 (Figure 7.15). This allowed our algorithm to prefer points close to the target when scanning.

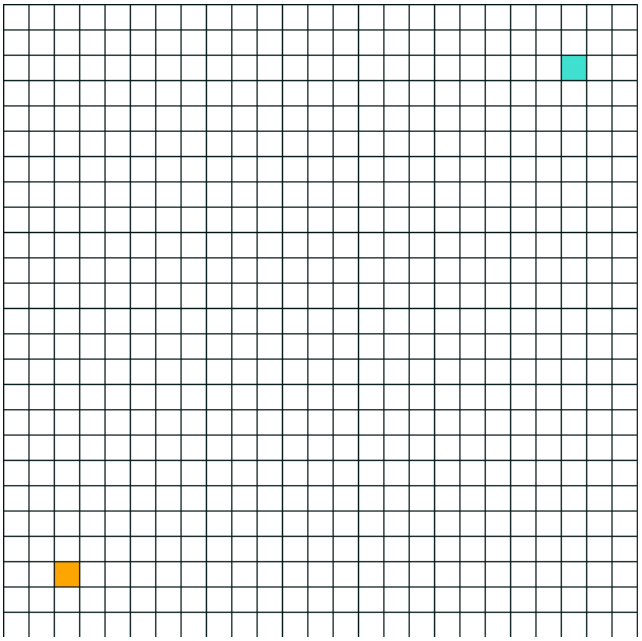

Figure 7.14 Scenario 2 Initial Grid

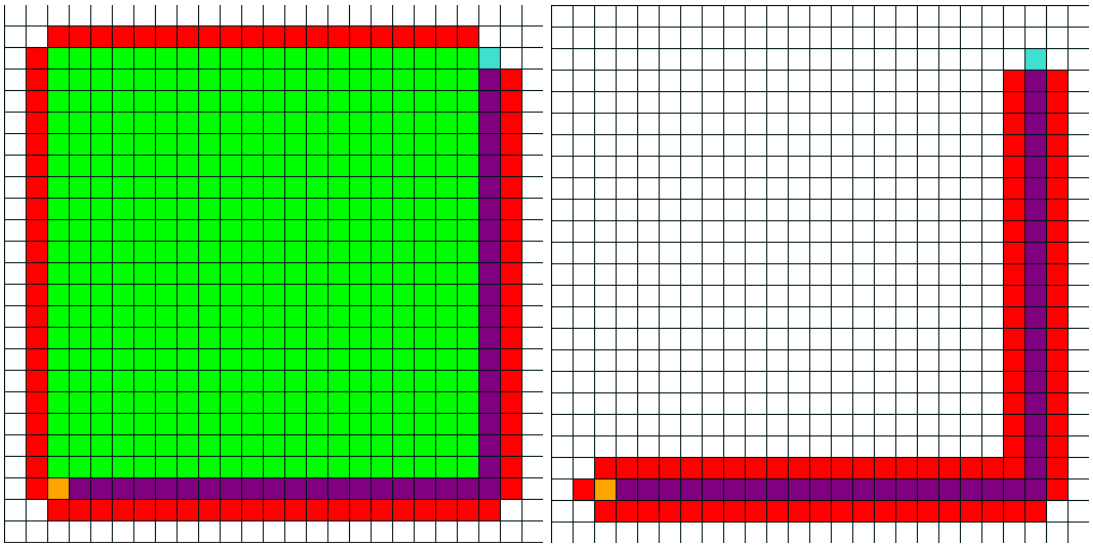

Figure 7.15 Scenario 2 Algorithms Results

	Path Cost	Steps	Time	Algorithm Name
Figure 7.15	40	441	0.2138600	A*
Figure 7.15	40	41	0.2180500	Tampered A*

Table 7.2 Scenario 2 Comparison Table

The running times of the algorithms, the number of steps and the length of the path they find are given in the table 7.2 above.

As seen above, the A* algorithm we modified scans much less open area than the other.

7.4.3 Example 3

In the first two examples, we compared and analyzed the A* and Dijkstra algorithms with their respective versions. Now let's examine the A* and Dijkstra algorithms through an example.

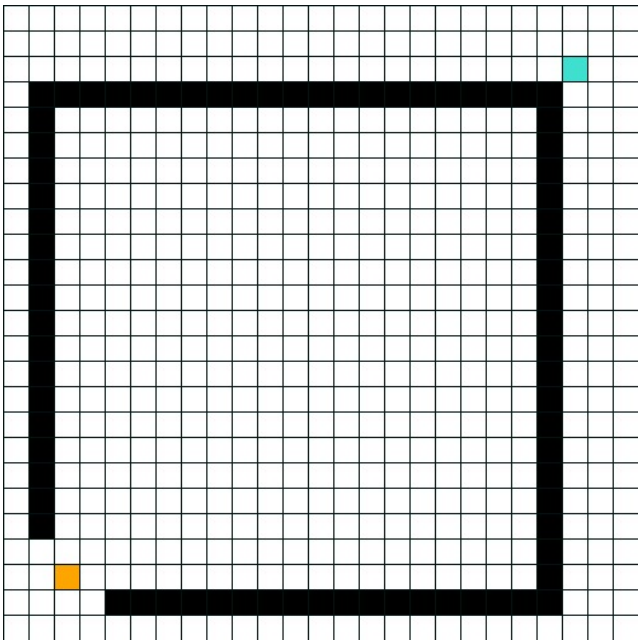

Figure 7.16 Scenario 3 Initial Grid

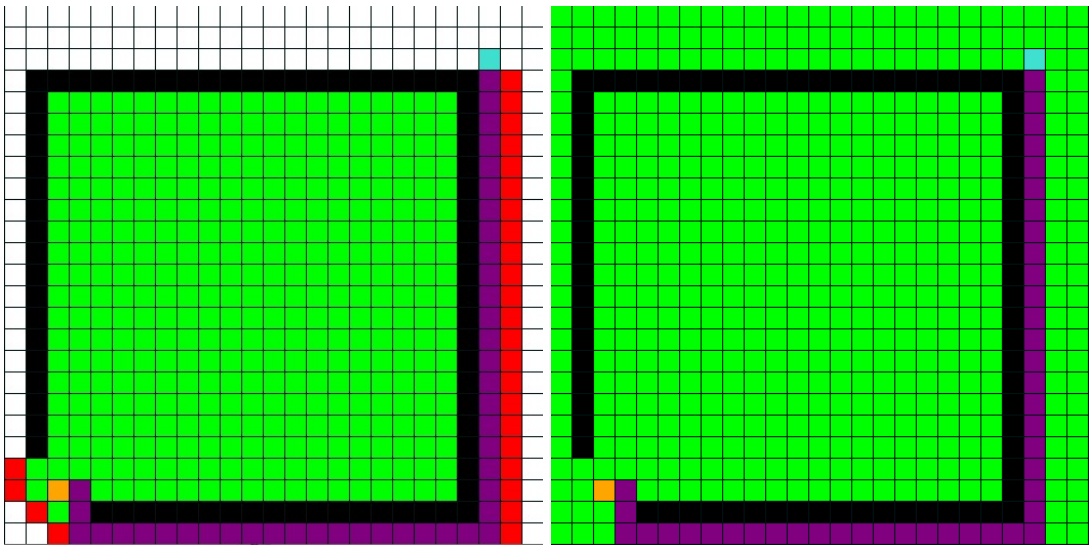

Figure 7.17 Scenario 3 Algorithms Results

	Path Cost	Steps	Algorithm Name
Figure 7.17	44	407	Tampered A*
Figure 7.17	44	551	Dijkstra

Table 7.3 Scenario 3 Comparison Table

Due to its structure, the A* algorithm scans towards the target. When he encounters an obstacle on his way to the target, he will not be able to gain efficiency from his greedy structure. As we saw in the example above, the A* algorithm converges to the Dijkstra algorithm, which scans everywhere when obstacles come between it and its target.

7.4.4 Example 4

Both of our algorithms detect weighted nodes and determine the path accordingly.In this example we will examine how these two algorithms respond to weighted nodes and obstacles.

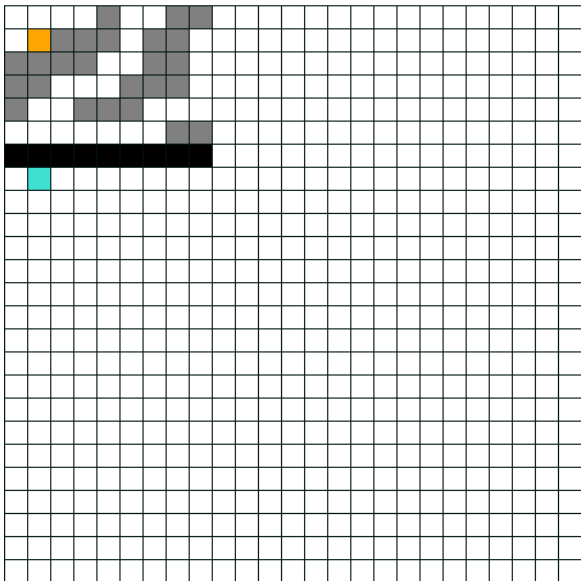

Figure 7.18 Scenario 4 Initial Grid

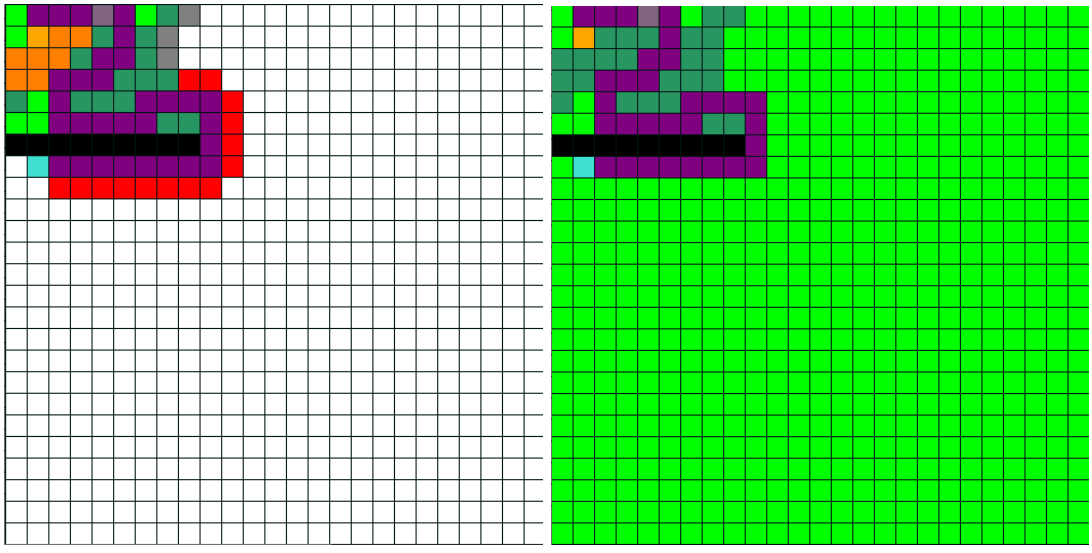

Figure 7.19 Scenario 4 Algorithms Results

	Path Cost	Steps	Algorithm Name
Figure 7.19	56	46	Tampered A*
Figure 7.19	56	620	Dijkstra

Table 7.4 Scenario 3 Comparison Table

As seen above, we see that the A* algorithm finds the way in much less steps when the obstacles are few and the target is close. At the same time, while the Dijkstra algorithm scans everywhere, the A* algorithm avoids scanning weighted nodes and performs scanning towards the target.

8
Performance Analysis

In this section, the performance of the algorithms will be examined using the data from the experimental results discussed above.

It has been observed that the Dijkstra algorithm, which came out early from the Dijkstra algorithms we selected, could not find the shortest path in some conditions. Since the purpose of our application is to deliver our robots to the target in the shortest way, this algorithm was not used in our application.

On the other hand, normal Dijkstra's algorithm scanning all over is tolerable given that it gives absolute results. In our application, this algorithm is included in terms of diversity.

When the A* algorithms are compared, we see that the modified A* algorithm has an absolute advantage over the normal A* algorithm in the number of steps, especially in the unobstructed map. It has been determined that both algorithms give correct results, so the modified A* algorithm is included in our application.

Both A* and Dijkstra algorithms that we choose to use in our application find the shortest path. When the examples are examined, we see that the A* algorithm uses much less number of steps in the open area. However, on maps with many obstacles, the A* algorithm converges to the Dijkstra algorithm in the number of steps. In the examples we examined above and the examples we examined independently, we see that the A* algorithm uses an average of 175.6 steps, while the Dijkstra algorithm uses 592.4 steps. While selecting these examples, care was taken to use maps with different conditions.

9
Result

Algorithms are examined for different scenarios in the experimental result and analysis part.

Large differences were observed in the open area in the number of steps of the modified A* algorithm compared to the normal A* algorithm. However, when large-scale obstacles come between the robot and the target, we see that these two algorithms converge.

It has been determined that the early exiting Dijkstra algorithm determines longer paths than normal Dijkstra. The whole purpose of algorithms is to draw the optimum path for the robots. This issue can never be compromised in order to gain the working time of the algorithms. Therefore, the early exiting Dijkstra application is not suitable for our system.

The modified A* algorithm was chosen by us as it works faster than normal in most scenarios. The grid structure of our map has put the A* algorithm ahead of the Dijkstra algorithm. That's why we use modified A* algorithm and normal Dijkstra algorithm in our code.

References

[1] A. Broadbent, "Free ranging agv systems: Promises, problems and pathways," in *Proceedings of the 2nd International Conference on Automated Materials Handling, 1985*, 1985.

[2] F. Glover, D. Klingman, and N. Phillips, "A new polynomially bounded shortest path algorithm," *Operations Research*, vol. 33, no. 1, pp. 65–73, 1985.

[3] P. J. Egbelu and J. Tanchoco, "Potentials for bi-directional guide-path for automated guided vehicle based systems," *International Journal of Production Research*, vol. 24, no. 5, pp. 1075–1097, 1986.

[4] S. C. Daniels, "Real time conflict resolution in automated guided vehicle scheduling," AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH, Tech. Rep., 1988.

[5] J. Huang, U. S. Palekar, and S. G. Kapoor, "A labeling algorithm for the navigation of automated guided vehicles," 1993.

[6] C. W. Kim and J. M. Tanchoco, "Conflict-free shortest-time bidirectional agv routeing," *The International Journal of Production Research*, vol. 29, no. 12, pp. 2377–2391, 1991.

[7] N. N. Krishnamurthy, R. Batta, and M. H. Karwan, "Developing conflict-free routes for automated guided vehicles," *Operations Research*, vol. 41, no. 6, pp. 1077–1090, 1993.

[8] C. W. Kim and J. Tanchocoj, "Operational control of a bidirectional automated guided vehicle system," *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, vol. 31, no. 9, pp. 2123–2138, 1993.

[9] J. K. Lim, J. M. Lim, K. Yoshimoto, K. H. Kim, and T. Takahashi, "A construction algorithm for designing guide paths of automated guided vehicle systems," *International Journal of Production Research*, vol. 40, no. 15, pp. 3981–3994, 2002.

[10] F. Taghaboni-Dutta and J. M. A. Tanchoco, "Comparison of dynamic routeing techniques for automated guided vehicle system," *International Journal of Production Research*, vol. 33, no. 10, pp. 2653–2669, 1995.

[11] J. TANCHOCOf and D. Sinriech, "Osl—optimal single-loop guide paths for agvs," *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, vol. 30, no. 3, pp. 665–681, 1992.

[12] J. Lin and P.-K. Dgen, "An algorithm for routeing control of a tandem automated guided vehicle system," *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, vol. 32, no. 12, pp. 2735–2750, 1994.

[13] D. Sinriech and J. Tanchoco, “Sft—segmented flow topology,” in *Material flow systems in manufacturing*, Springer, 1994, pp. 200–235.

[14] F. TAGHABONI and J. Tanchoco, “A lisp-based controller for free-ranging automated guided vehicle systems,” *International Journal of Production Research*, vol. 26, no. 2, pp. 173–188, 1988.

[15] M. Akturk and H. Yilmaz, “Scheduling of automated guided vehicles in a decision making hierarchy,” *International Journal of Production Research*, vol. 34, no. 2, pp. 577–591, 1996.

[16] Z. Yuan, Z. Yang, L. Lv, and Y. Shi, “A bi-level path planning algorithm for multi-agv routing problem,” *Electronics*, vol. 9, no. 9, p. 1351, 2020.

[17] N. Swift. (). “Easy a* (star) pathfinding,” [Online]. Available: `https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2`. (accessed: 02.28.2017).

Curriculum Vitae

FIRST MEMBER

Name-Surname: Atakan TEKOĞLU
Birthdate and Place of Birth: 01.08.1995, İstanbul
E-mail: atakantekoglu@gmail.com
Phone: 0537 295 82 82
Practical Training: RND Lab Technology

SECOND MEMBER

Name-Surname: Süleyman Ali Burak ÇINAR
Birthdate and Place of Birth: 21.02.1998, Kocaeli
E-mail: sabc4129@gmail.com
Phone: 0 506 262 48 62
Practical Training: -

Project System Informations

System and Software: Windows İşletim Sistemi, Python, PyCharm, PyGame
Required RAM: 2GB
Required Disk: 256MB