TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

PROJECTWORK

Develop a BIM-supported robot path planning and energy reducing system for roof slabs on construction sites

eingereicht von
cand. ing. Mi Zhou
geb. am 26.11.1997
Matrikel-Nummer: 5054440

Betreuer/in:

- Prof. Dr.-Ing. habil. Karsten Menzel
- Dipl.-Ing. Shaowen Han

Dresden, 17.03.2023

(Insert here the original sheet with your task)

DECLARATION OF ORIGINALITY

I confirm that this assignment is my own work and that I have not sought or used the inadmissible help of third parties to produce this work. I have fully referenced and used inverted commas for all text directly quoted from a source. Any indirect quotations have been duly marked as such.

This work has not yet been submitted to another examination institution – neither in Germany nor outside Germany – neither in the same nor in a similar way and has not yet been published.

Name: Zhou

Vorname: Mi

Matrikelnummer: 5054440

Dresden, 0.03.2023

Signature B.Sc.ing. Mi Zhou

ABSTRACT

Based on BIM information provided by the construction field, this project work is expected to build the path-finding mechanism for construction robots on roof sites to achieve safety and energy consumption-reduced systems in construction tasks.

First extracts the needed data from the AutoCAD file. And a Fortran program was developed to calculate the middle point from previous data. Then, put the middle information into Matlab to visualize the composition of the entire roof. As for the pathfinding part, the raster map was created in Matlab. One program for delivering roof panels and one program for roof panel laying are created and combined. Finally, those programs can output coordinate data to be further used in the ROS system and Turtlebot to provide a simulated visualization of this entire project.

Keywords: BIM, civil engineering, Robot Operation System (ROS), Pathfinding, A-star algorism, Complete coverage path planning

TABLE OF CONTENT

1 INTRODUCTION

The construction industry in almost every developing country faces serious waste of resources(Rondinel-Oviedo 2021), low production efficiency(Filipe Barbosa), low level of information technology(michael), and lack of sufficient management and control from the building designer and engineer(Boadu et al. 2020).

The existence of those situations, along with the entire industry's outdated ideas, the complexities of management in construction(David Baccarini), unreliable building products(Ebrahim and Wayal 2020), building products in production alongside mass and disorganized data process(Adekunle et al. 2022), and impromptu formation of project teams are all characteristics triggered by it(Alhussein et al. 2022). This may result in an unpleasant living experience for the user and the risk of many safety problems, which are unwanted by every individual.

Building information modeling (BIM) is a digital representation of a physical model and its function, which can be used for planning, design, construction, and operation of the facility(Bernard et al. 2013). It helps architects, engineers, and even constructors visualize what is to be built in a simulated environment to identify any potential design, construction, or operational issues. In the meantime, the use of BIM technology can improve the efficiency of the project construction, while saving cost and making solving previous problems possible(Nuzul Azam Haron; Nuzul Azam Haron). When completed, the building information model contains precise geometry and relevant needed data to support the design, procurement, fabrication, and construction activities required to realize the building(Tang et al. 2017). After completion, this model can be used for operations and maintenance purposes.

And because of BIM's possibility to be intrinsically instrumental in eliminating dominant wastes in the construction field(jaywu), and the possibility for the designer and engineer to manage and control the project in time, it has gotten more and more popular among construction companies in recent years(Azhar and S).

In a nutshell, The use of BIM on the construction site is becoming more and more frequent(Thomas Vorbeck and Nadine Wills), and in this project, another application of BIM will be introduced.

Here in this project, we apply two robots in the roof construction field to perform tasks such as lifting and placing roofing materials and installing insulation. One robot's mission is to deliver roof panels, which refers to lifting panels, the process of determining the

optimal path for a robot to navigate from one stockpile point to another robot while avoiding obstacles and adhering to construction constraints and rules. And another robot's mission is to plan the panels according to the DIN standard, in the actual construction field, each roof condition has a different roof brick laying sequence, so the robot is expected to plan the roof along the set path and achieve some level's automatic. Also, considering one robot has a carry limit of roof panels, this second robot Is expected to send signals of missing panels to another robot so that the supply can be delivered by another robot.

And in order to accomplish this goal, this project uses Matlab to visualize the entire process and a Robot "Turtlebot" that is controlled by the ROS system to actually simulate this roof planning process.

2 THEORY DESCRIPTION-STANDARD TECHNIC

2.1. APPLICATION OF MODERN ENGINEERING IN CIVIL ENGINEERING

2.1.1. APPLICATION OF BIM IN CIVIL ENGINEERING

BIM was used in many engineering applications, such as in the field of ancient building protection(Li Yan), Juan Wang and other researchers from China have investigated a multi-dimensional BIM model for construction, integration combined with multi-source information, structural early warning mechanisms, and multifunctional management platform(Wang et al. 2022a). This model can real-time analyze the temperature, strain, and moisture of entire timber structures, and eventually, with the help of this BIM model, they have concluded that the overall wooden structural state is stable and safe.

Also, in the field of traffic and road design, Karen Castaneda and other researchers from Colombia have investigated a BIM-based methodological framework for traffic analysis at road intersections, which can help promote the reduction of design flaws both in traffic analysis and in other design disciplines. At the same time, in an application for existing road infrastructure, Valeria Vignali and his team analyzed in I-BIM (Infrastructure Building Information Modelling) the upgrade of a section of the SS 245 road, in order to show the benefits of I-BIM applied to existing road infrastructure(Castañeda et al. 2021).

And, as some new technology like Extrusion-based Construction 3D Printing developed in the construction industry and visual tracking using the 3D model of the building framework, the combination of BIM and those inventions can bring great potential to revolutionize the construction industry and be deployed on a massive scale shortly.

Ehsan Kamel and Ali Kazemian have focused on BIM-integrated thermal analysis and building energy modeling, a building energy modeling (BEM) workflow for C3DP – based on BIM – is developed and evaluated using a case study and considering the specifics of 3D-printed buildings. their result shows that common wall systems in C3DP cannot meet the minimum requirements of the energy codes, especially in cold climate regions, which necessitate additional insulation and using construction materials with lower thermal conductivity, such as lightweight concrete(Kamel and Kazemian 2023).

Debaditya Acharya and his colleague have demonstrated the ability of BIM-Tracker for drift-free localization using real images, which makes it suitable for navigation and augmented reality applications(Acharya et al. 2019).

2.1.2. PATHFINDING INTRODUCTION

Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points(Tarik Terzimehic 2011). Or, put in a nutshell, pathfinding is all about finding the best path between points A and B(RICHARD UGGELBERG). It is a more practical variant of solving mazes. This field of research is based on Dijkstra's algorithm for finding the shortest path on a weighted graph(Mora).

With the increased demand for efficient pathfinding systems(Lawande et al. 2022), more and more advanced and efficient theoretical models are proposed(Silvester Dian Handy Permana).

Fig. 1 – Pac-Man a classic game which requires pathfinding
source: https://www.bing.com/images/

Surprisingly, another almost irrelevant industry's development has contributed a lot to pathfinding(Xiao Cui). In the video game industry, there needed pathfinding in relevant huge maps with a limited amount of CPU usage, which leads to the shortest path search problem(Botea et al. 2013).

The concept of using abstraction and heuristics is older and was first mentioned under the name ABSTRIPS (Abstraction-Based STRIPS) which was used to efficiently search the state spaces of logic games. A similar technique is navigation meshes (navmesh), which are used for geometrical planning in games, and multimodal transportation planning which is utilized in traveling salesman problems with more than one transport vehicle.

Fig.1 shows a classic game that requires pathfinding, It is the old game's existence that pushed pathfinding to a new level.

2.1.3. ROOF LAYOUT

DIN, the German Institute for Standardization, is the independent platform for standardization in Germany and worldwide. As a partner for industry, research, and society as a whole, DIN plays a major role in helping innovations to reach the market in areas such as the digital economy or society, often within the framework of research projects. DIN is a standard document that specifies requirements for products, services, and/or processes. This helps ensure the free movement of goods and encourages exports. Standardization supports efficiency and quality assurance in the industry, technology, science, and the public sector. It serves to safeguard people and property and to improve quality in all areas of life. Studies show that standards generate economic benefits estimated at about € 17 billion a year(DIN).

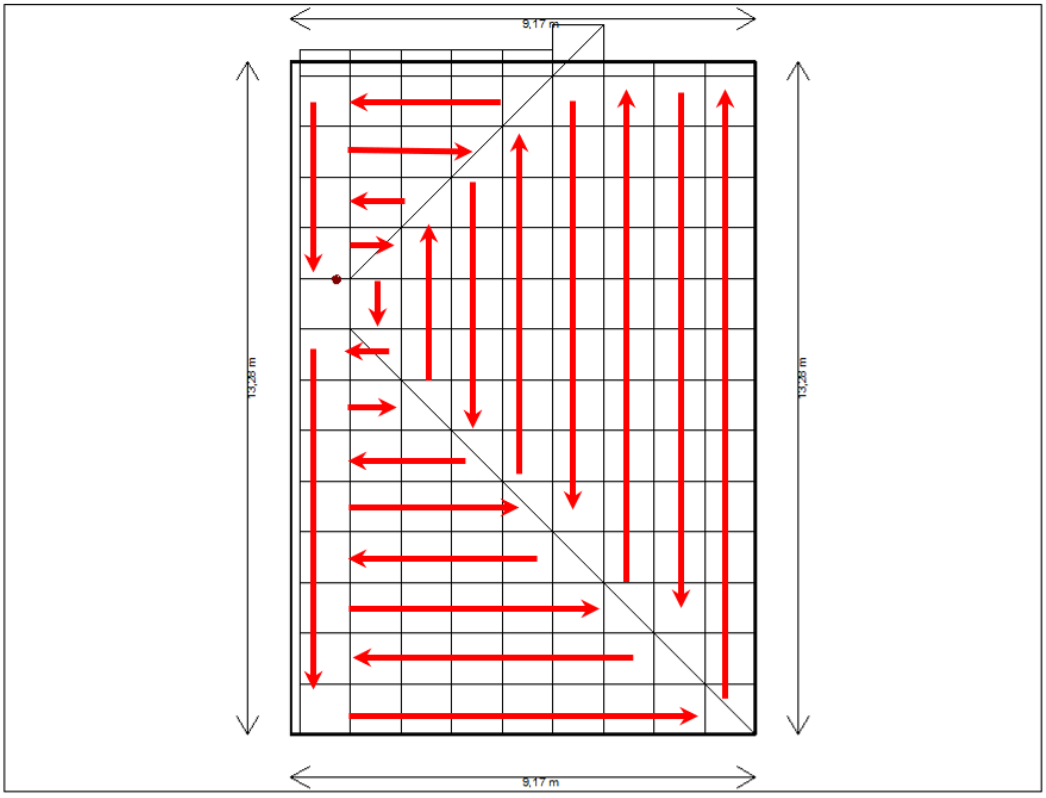

Fig. 2 – Optimized laying sequence

In this project, a roof path-finding robot program is needed. However, according to the regulations of the relevant building construction process, the layout plan must be set down before construction.

Thus, as shown in Fig. 2, this laying scheme is set according to the DIN18531 regulations in the order drawing in the red arrow. It's worth mentioning here that this laying scheme is only for the laying robot on this specific roof.

2.1.4. BUILT BY BUILDERS

The construction industry has played a critical role in the global economic system and it's closely bounded to every individual's daily life and also employs a substantial workforce. The global construction market size is expected to grow from $13.57 trillion in 2021 to $15.17 trillion in 2022 at a compound annual growth rate (CAGR) of 11.8%(The Business Research Company).

But still, there are plenty of factors in the construction industry that are unwanted, behind the prosperity, some chronic challenges such as lack of safety protection, significant employment shortages, constructing workers' lack of sufficient standard safety building technic, and low productivity impede further development of the entire industry(Huang et al. 2023).

The safety problem is severe. For example, the construction industry is also one of the most hazardous industries in the EU and many other countries worldwide(Lingard 2013). Major safety hazards for construction workers include working at heights, in excavations and tunnels, on highways, in confined spaces, and exposure to electricity, construction machinery, etc. (https://oshwiki.eu/wiki/Construction_safety_risks_and_prevention)

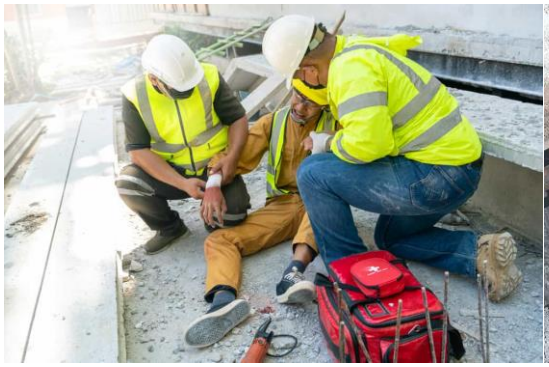

Fig. 3 – Injury in construction work source:https://www.cassisilawfirm.com/blog/2022/july/5-examples-of-injury-sustaining-construction-sit/

Fig. 4 – A group of workers lacks protection source:https://baike.baidu.com/item/

Although in many countries big efforts have been done to improve safety performance, the construction sector continues to lag behind most other industries(Luo 2020). Worldwide, construction workers are three times more likely to be killed and twice as likely to be injured than workers in other occupations(Gran Vía 2000). Altogether in Europe, every year more than 1,000 workers are killed, and over 800,000 workers are injured. The hard-to-guarantee safety conditions discourage workers from entering construction, contributing to labor shortages(Occupational Safety and Health

Administration). There is an injury in construction work shown in Fig. 3, and Fig. 4 shows that a group of workers facing safety problems because of lacking enough protection.

2.1.5. ROBOT USAGE IN THE FILED

Early research relevant to construction robots, lacking management commitment to technological change is often cited as primary obstacles to the rapid introduction of automation and robotics technology into the construction field(Guillermo Moral).

But, still, there is some pioneer who manages to use Inadequate technology to build some prototype, which can bring enlightenment to the design of modern construction robots.

The robot YAIR t is an autonomous intelligent robot that has a distributed architecture with several intelligent modules managing the sensors. These sensory modules give the robot perceptions about the surrounding environment. F. Blanes and his team developed an IR sensor that has been designed to estimate distances presented. This sensor can be used to construct maps under hard real-time restrictions(Blanes et al. 2000).

Dr. Hui-Ping Tserng and his team have established the framework for an affordable path-planning system for compaction operation using an onboard guidance facility. This facility would integrate an Asphalt Knowledge-Based System with an economical sensing system, such as a Global Positioning System (GPS), and a navigational compass. And their result, explored the potential for automation in the area of asphalt compaction in highway pavement operations using GPS. And an algorithmic strategy for path planning was developed and implemente(Hui-Ping Tserng).

Thanks a lot to those pioneers, whose persistent research spirit and innovative spirit build a solid foundation for the development of modern construction robots, And brought inspiration to the research of future generations.

Automation in construction, especially with robotics, has the potential to relieve some of the strain imposed by the labor shortage and increase productivity(ninakutzbach). The presence of robots in construction is becoming more interdisciplinary. Robots excel at performing repetitive tasks with precise motions in controlled environments such as those in offsite construction facilities. In addition, robots can save workers from dangerous, repetitive, and labor-intensive tasks, which in turn allows them to focus on more advanced and meaningful but less harmful tasks(Xiao et al. 2022).

Therefore, adopting robotic automation by the Architecture, Engineering, and Construction industries can bring numerous benefits in terms of productivity, quality, and most importantly safety(Davila Delgado et al. 2019), which is aligned with the goal of the economy, and restoring and improving the urban infrastructure also reduce construction's environmental impact. Eventually benefits other industries like

construction robots designing and manufacturing, which in return, can bring tremendous job opportunities to residents. Then, more demand for the construction industry arises(Wong Chong et al. 2022).

In the early stage of the construction industry, the usage of robots can be only seen in the developed region. Unimate as shown in Fig. 5 was developed in the United States by George Devol in 1954 as one of the first industrial robots. From that point, robotics became incorporated into production and manufacturing processes throughout the world(Automate). Carnegie Mellon University has been the first U.S. institution to become involved in the design and prototyping of remotely controlled and autonomous construction robotics. North Carolina State University is involved in three general areas of construction robotics application: excavation, bridge maintenance, and masonry construction. Those areas, even in the situations of the contemporary world, still need more robots to replace workers' tedious and repetitive tasks.

Fig. 5 –Unimate pouring coffee for a woman, 1967. source: https://en.wikipedia.org/wiki/Unimate#/media/

Fig. 6 –Imagining future construction robots source: https://www.bdcnetwork.com/robotics-new-way-demolish-buildings

The automation needs assessment methodology developed at the University of Texas at Austin has been applied to several construction projects, including the Taipei Rapid Transit System in Taiwan, the evaluation and ranking of automated road maintenance systems in the United States, and the application of advanced enabling technologies to construction tasks in environmental remediation. (Current Status and Key Issues for Construction Automation and Robotics in the USA)

In the contemporary stage of the construction industry. Benefiting from the progress of modern electronic integration processes and material science, more professional and powerful construction robots are invented. As shown in Fig. 6, a more advanced robot will be developed.

Developed in the 1980s, 3D printing is a process of sequential layering of construction materials, in order to make three-dimensional structures. The 3D printing process is also

known as Additive Manufacturing. 3D printing in construction is done using specialized 3D printers. The 3D printers for construction are controlled by computer programming and artificial intelligence. Advanced construction robotics technology reads the three-dimensional CAD models and 3D prints the structure accordingly by placing layers of the materials, with high accuracy(Reetie Multani).

An Australian company dealing in construction robotics in Fig. 7, while humans have been laying bricks in the same conventional way for the past 6000 years. To the manual bricklaying and wall-making procedure, Fastbricks Robotics Ltd has invented a brick-laying robot, called Hadrian X, the world's first mobile robotic block-laying machine and system, capable of safely working outdoors in uncontrolled environments with speed and accuracy. It uses modularly designed blocks with aligned cores, enabling easy installation of cabling and services through the cavities(Fbr).

Fig. 7 – Hadrian X block-laying machine system
source: https://www.fbr.com.au/view/hadrian-x

2.2. CONVENTIONAL PATHFINDING THEORY

2.2.1. DIJKSTRA

The Dijkstra algorithm (Dijkstra) was proposed by Dutch computer scientist Dijkstra in 1959, so it is also called the Dijkstra algorithm. It is the shortest path algorithm from one vertex to another vertex and solves the shortest path problem in the weighted graph. The main feature of Dijkstra's algorithm is to start from the starting point, adopt the strategy of the greedy algorithm, and traverse to the adjacent nodes of the vertex that is the

closest to the starting point and has not been visited each time until it extends to the endpoint(Krzysztof Apt).

For example, if the nodes of the graph represent cities and the costs of edge paths represent driving distances between pairs of cities connected by a direct road, then Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. A widely used application of shortest path algorithms in network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and OSPF (Open Shortest Path First)(J. Moy). It is also employed as a subroutine in other algorithms such as Johnson's algorithms, which is a way to find the shortest paths between all pairs of vertices in an edge-weighted directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist(Cantone and Faro 2014).

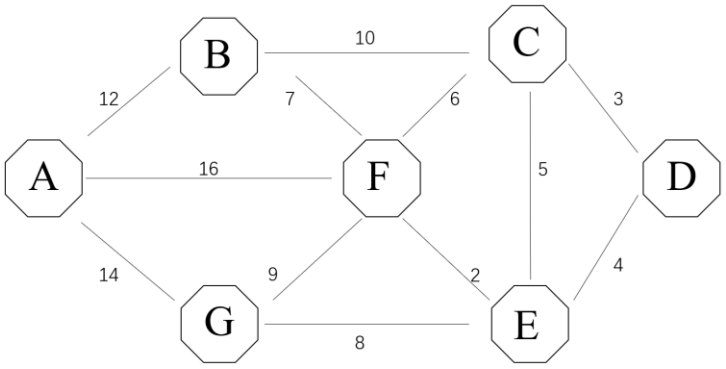

Fig. 8 – Dijkstra algorithm's map
source: https://blog.csdn.net/PRML_MAN/article/

Shown in Fig. 8, the Dijkstra algorithm can be explained as follows, Let G=(V, E) be a weighted directed graph, and divide the node set V in the graph into two groups, The first group is the set of nodes for which the shortest path has been calculated (denoted by S, initially in S there is only one source point, and each time the shortest path is obtained, the node is added to the In the set S until all nodes are added to S, the algorithm ends).

The second group is the rest of the undetermined shortest path node-set (denoted by U), according to the shortest path the increasing order of short path length adds the nodes of the second group to S in turn joining the process(Dorothea Wagner and Thomas Willhalm), the length of the shortest path from the source point v to each node in S is always kept small is the length of the shortest path from source v to any node in U.

In addition, each node corresponds to a distance, and the distance of nodes in S is from v to the shortest path length to this node, and the distance of nodes in U is from v to this node. Only the nodes in S are included as the current shortest path length of the

intermediate nodes. Initially, S only contains the starting point s; U contains all but other nodes other than s and the distance between nodes in U.

The distance is "the distance from the starting point s to the node" [for example, if the distance of node v in U is the length of (s, v), then s and v are not adjacent, then the distance of v is infinite]. Then select "the node k with the shortest distance" from U, and add node k to S; at the same time, remove node k from U.

Update the distance from each node in U to the starting points. The reason for updating the distance of nodes in U is since the determination of k in the previous step is to find the shortest nodes of the path so that k can be used to change new distances to other nodes; For example, (s,v)'s distance may be greater than the distance of (s,k)+(k,v).

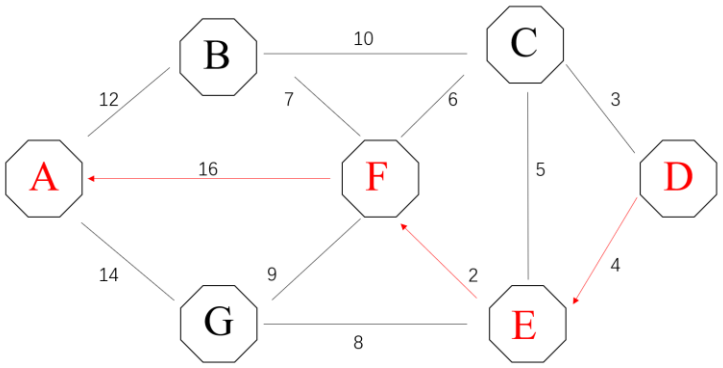

Fig. 9 – Dijkstra algorithm's map2
source: https://blog.csdn.net/PRML_MAN/article/

Eventually, repeat all those steps until all the points have been traveled. So that a locally optimal path can be found as shown in Fig. 9.

2.2.2. ANT COLONY ALGORITHM

Ant Colony Algorithm (ACA) was first proposed in 1992, The algorithm mimics the foraging behavior of ants in nature(Mullen et al. 2009): When an ant is looking for a food source, it will release a pheromone on its path and can perceive the pheromone released by other ants. Size-characterized pathways for pheromone concentrations. The higher the pheromone concentration, it means the shorter the corresponding path distance(Yaralidarani and Shahverdi 2016).

Usually, ants will preferentially choose the path with higher pheromone concentration with greater probability, and release a certain amount of pheromone to enhance the pheromone concentration on this path, so that there will be positive feedback. Eventually,

the ants are able to find a path from the nest to the food source the best path is the shortest distance(Xiong et al. 2021).

The feasible solution of the problem to be optimized is represented by the walking path of the ants, and for all the ant colonies there are paths that constitute the solution space of the problem to be optimized.

Ants with shorter paths released more pheromones, and as time went on, ants with shorter paths the concentration of pheromones accumulated on the path increases gradually, and the number of ants who choose this path also increases(Jayadeva et al. 2013).

Finally, the whole ants will concentrate on the best path under the action of positive feedback, which corresponds to the optimal solution of the problem to be optimized.

As an explanation of the selection part, each ant needs to construct a solution to move through the graph. To select the next edge in its tour, an ant will consider the length of each edge available from its current position and corresponding pheromone level. At each step of the algorithm, each ant moves from a state x to state y, corresponding to a more complete intermediate solution. Thus, each ant K computes a set A_k (x)of feasible expansions to its current state in each iteration, and moves to one of these in probability.

For ant K, the probability p_{xy}^k of moving from state x to state y depends on the combination of two values, the attractiveness η_{xy} of the move, as computed by some heuristic indicating the a priori desirability of that move and the trail level $\tau_{xy}$of the move, indicating how proficient it has been in the past to make that particular move. The trail level represents a posteriori indication of the desirability of that move.

So, the possibility of ant moving from x to y in k time can be written as:

$$P_{xy}^k = \begin{cases} \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\Sigma_{z \in allowed_x}(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}, & z \in allowed_x \\ 0, & z \notin allowed_x \end{cases} \qquad (1.1)$$

In the beginning, $allowed_x$ have in total (n-1) elements, which include all the city expect ant K's start city. Eventually, elements n $allowed_x$ would decrease over time, until the entire set is empty. Which means all the paths have been visited.

where τ_{xy} is the amount of pheromone deposited for the transition from state x to y, 0 ≤ α is a parameter to control the influence of τ_{xy} the bigger α is, the influence of τ_{xy} is higher. η_{xy} is the desirability of state transition x y (a priori knowledge, typically 1 /d_{xy} , where d is the distance) and β≥ 1 is a parameter to control the influence of η_{xy} , the bigger β is, the influence of η_{xy} is higher. τ_{xy} and η_{xz} represent the trail level and attractiveness for the other possible state transitions.

This process can be explained in Fig. 10.

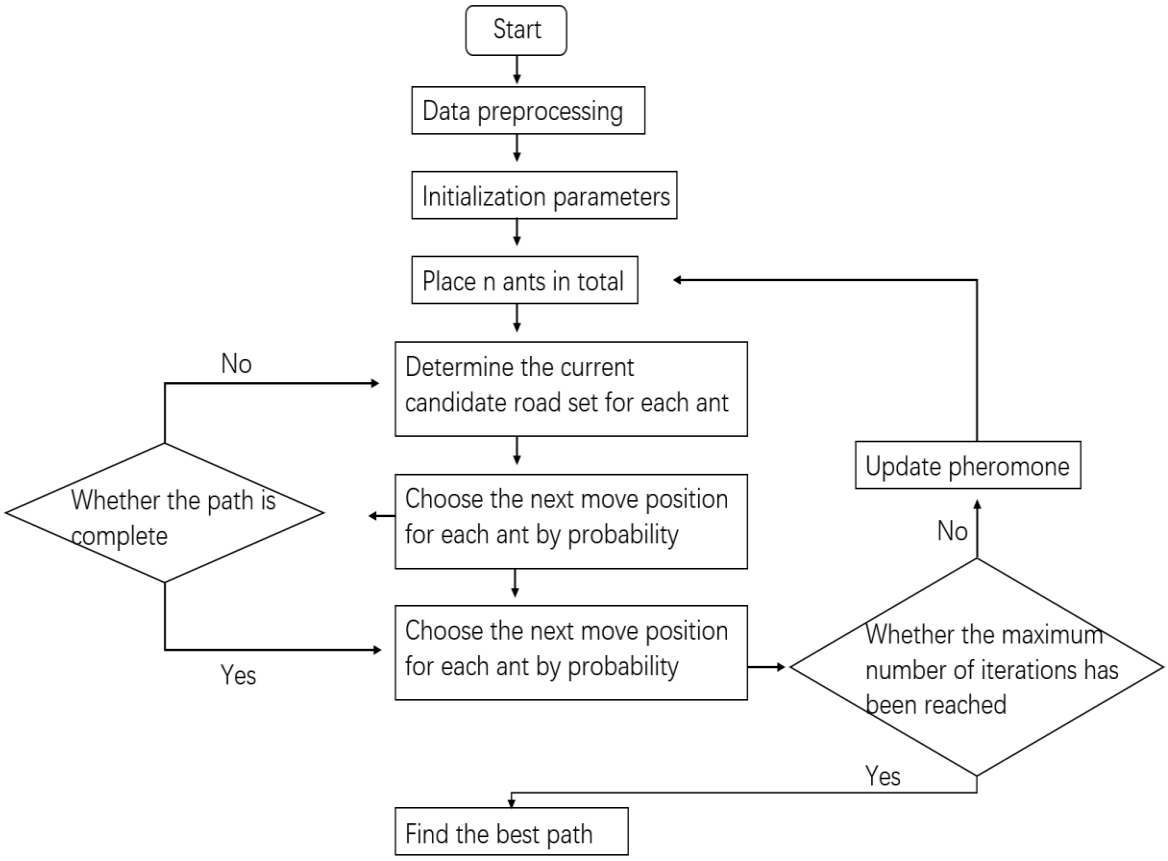

Fig. 10 – Ant Colony Algorithm

And Pheromones Concentration can be represented by:

$$\begin{cases} \tau_{xy}(t+1) = (1-\rho) * \tau_{xy}(t) + \Delta\tau_{xy} \\ \Delta\tau_{xy} = \Sigma_{k=1}^{n} \Delta\tau_{xy}^{k} \end{cases} ,0 < \rho < 1 \quad (1.2)$$

Where ρ (0<ρ<1) represents the volatility of pheromone, $\Delta\tau_{xy}^{k}$ represents k number's ant released pheromone amount between state x and state y. And $\Delta represents$ represent the sum of the pheromones released by all ants between state x and state y.

2.2.3. A STAR

The A* (A-Star) algorithm is the most effective way to solve the shortest path in a static road network. It is also an efficient algorithm for solving many search problems. Which is one of the algorithms being applied in this project work.(Wang et al. 2021)

The A* (A-Star) algorithm was widely used in indoor robot path search, game animation path search, etc. A* was created for the Shakey project, aiming to build a mobile robot

that could plan its actions(Vincent). Nils Nilsson originally proposed using the Graph Traverser algorithm for Shakey's path planning. Graph Traverser is guided by a heuristic function h(n), the estimated distance from node n to the goal node: it entirely ignores g(n), the distance from the start node to n. Bertram Raphael suggested using the sum, g(n) + h(n). Peter Hart invented the concepts we now call admissibility and consistency of heuristic functions(Church; Church). A* was originally designed for finding least-cost paths when the cost of a path is the sum of its costs, but it has been shown that A* can be used to find optimal paths for any problem satisfying the conditions of a cost algebra(Zeng and Church 2009).

About the A*(A-Star) algorithm, It combines the greedy algorithm (depth first) and the Dijkstra algorithm (wide degree priority), which is a heuristic search algorithm.

The evaluation formula of path pros and cons is:

$$f(n)=g(n)+h(n) \qquad (1.3)$$

where n is the next node on the path, f(n) is the cost estimate from the initial state to the target state via state n, and g(n) is the actual cost of going from the initial state to state n in the state space, and h(n) is the estimated cost of the best path from state n to the goal state. Defining and calculating g(n) and f(n) in our construction field is one of the problems this project work has to solve.

A* terminates when the path it chooses to extend is a path from start to goal or if no paths are eligible to be extended(Foead et al. 2021). The heuristic function is problem-specific. If the heuristic function is admissible – meaning that it never overestimates the actual cost to get to the goal, A* is guaranteed to return a least-cost path from start to goal.

The details about the A-Star program and how it’s being applied in this project work will be explained in chapter 3.

2.2.4. DYNAMIC PROGRAMMING

Dynamic programming is a branch of operations research, which is a mathematical method for solving multi-stage decision-making process optimization problems(MIT).

The choice of decision-making at each stage is not determined arbitrarily, it depends on the current state and affects future development. When the decisions at each stage are determined, decision-making, therefore, determines an activity route of the whole process. Such a multi-stage process with a chain structure is called a multi-stage decision-making problem.

Dynamic programming has a wide range of applications in the field of vehicle engineering technology(Wang et al. 2015), such as "optimal shift schedule for two-speed transmissions", "optimal energy management strategy for hybrid vehicles", "optimal grid map path search", etc.

In the early 1950s, the American mathematician Bellman and others proposed the famous principle of optimization that transforms multi-stage decision-making problems into a series of single-stage optimal problems(Hans Josef Pesch).

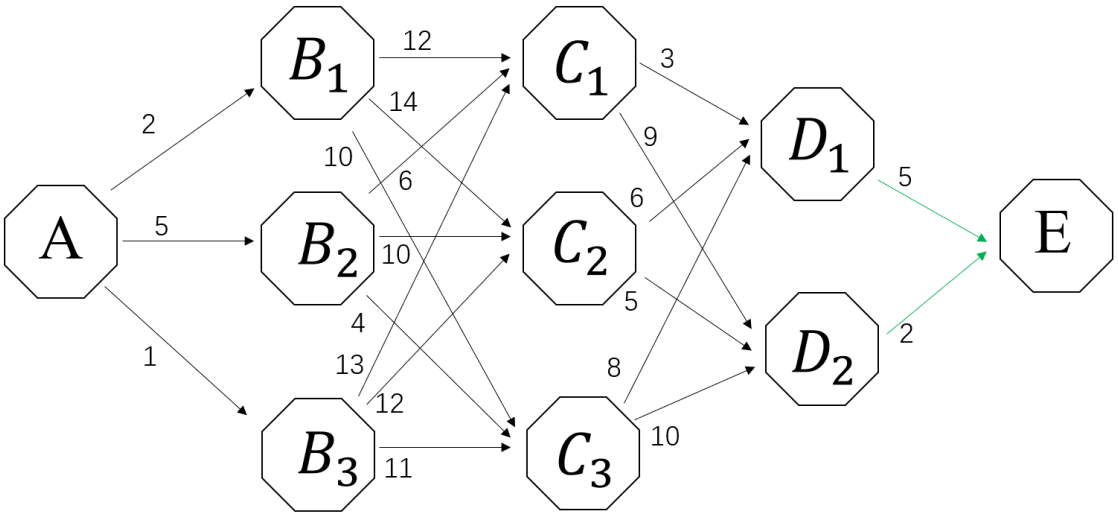

Fig. 11 – Dynamic programing step-1
source: https://blog.csdn.net/PRML_MAN/article/

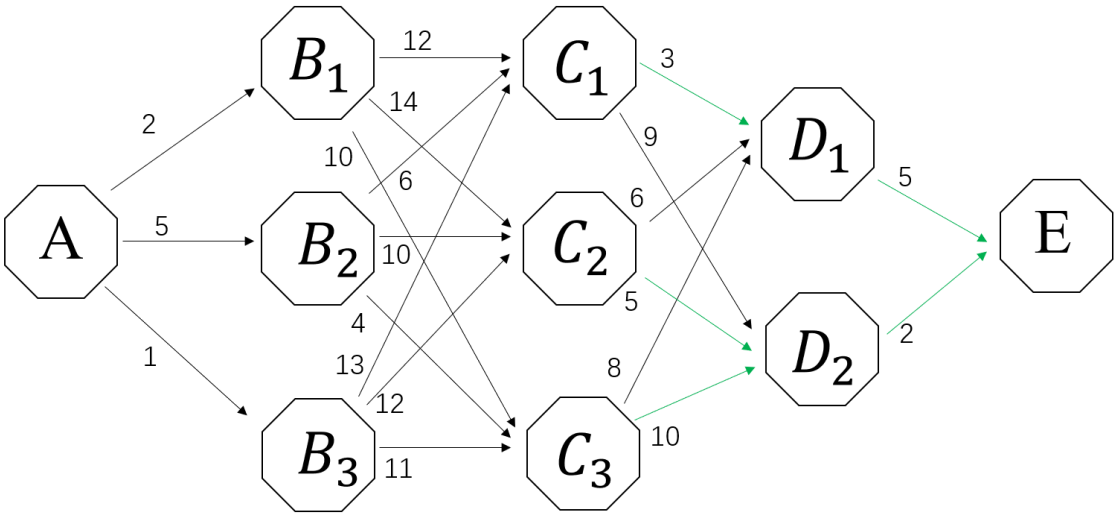

Fig. 12 – Dynamic programing step-2
source: https://blog.csdn.net/PRML_MAN/article/

The various stages that the optimal path (optimal decision-making process) passes through, each of which the path from the beginning point of a stage to the endpoint of the whole process must be the path from the beginning point of the stage to the endpoint of the whole process. The best path (optimal decision) among all possible paths to the end of the process is the famous optimization principle proposed by Bellman.

In short, a sub-policy of an optimal policy must also be optimal in the whole process.

Reverse optimization and the forward solution are the main characteristics of Dynamic programming. The DP algorithm essentially consists of three layers of loops; The first layer traverses each stage; The second layer traverses each state of the i-th stage; The third layer loops through each stage of the i+1th stage.

In detail, the DP algorithm can be explained as follows: Firstly, we start from end-point E, as shown in Fig. 11, and there are two paths from D towards E.

$$\begin{cases} f_4(D_1) = 5 \\ f_4(D_2) = 2 \end{cases} \qquad (1.4)$$

And in step-2, as shown in Fig. 12, from C to D there are in total 6 paths. And among these six routes, there are three optimal routes with distances of 3， 5 and 10 respectively.

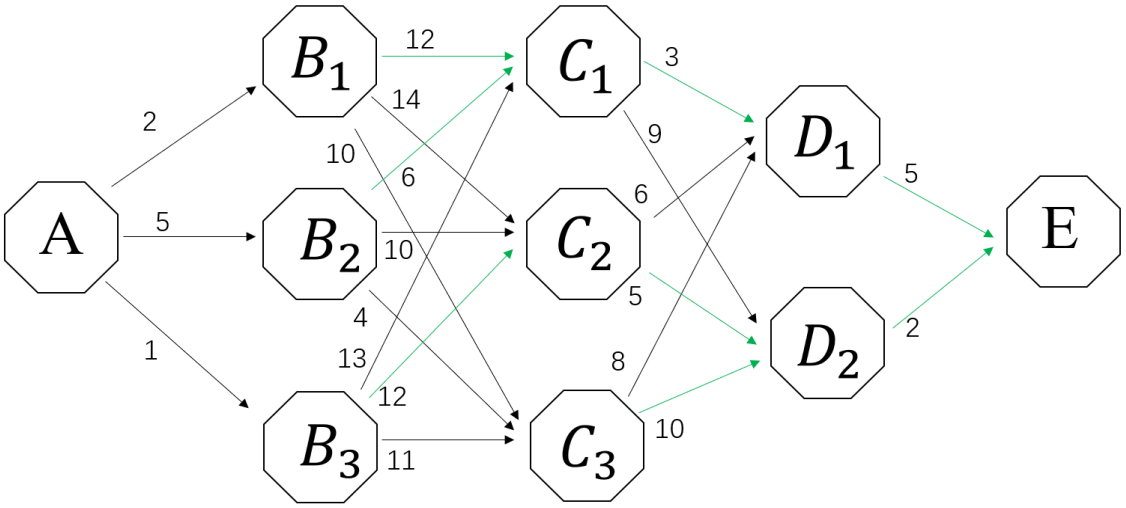

Fig. 13 – Dynamic programing step-3
source: https://blog.csdn.net/PRML_MAN/article/

$$f_3(C_1) = \min\begin{cases} d(C_1, D_1) + f_4(D_1) \\ d(C_1, D_2) + f_4(D_2) \end{cases} = min\begin{cases} 3+5 \\ 9+2 \end{cases} = 8$$

$$f_3(C_2) = \min\begin{cases} d(C_2, D_1) + f_4(D_1) \\ d(C_2, D_2) + f_4(D_2) \end{cases} = min\begin{cases} 6+5 \\ 5+2 \end{cases} = 7 \qquad (1.5)$$

$$f_3(C_3) = \min\begin{cases} d(C_3, D_1) + f_4(D_1) \\ d(C_3, D_2) + f_4(D_2) \end{cases} = min\begin{cases} 8+5 \\ 10+2 \end{cases} = 12$$

So, there is three local optimized path which is from C_1, D_1 to E, C_2, D_2 to E, and C_3, D_2 to E.

Then, in step-3, as shown in Fig. 13, there is a total of 9 paths from B to C. And among these nine routes, there are three optimal routes with a distance of 12, 6, and 12 respectively.

$$f_2(B_1) = \min\begin{cases} d(B_1, C_1) + f_3(C_1) \\ d(B_1, C_2) + f_3(C_2) \\ d(B_1, C_3) + f_3(C_3) \end{cases} = min\begin{cases} 12 + 8 \\ 14 + 7 \\ 10 + 12 \end{cases} = 20$$

$$f_2(B_2) = \min\begin{cases} d(B_2, C_1) + f_4(C_1) \\ d(B_2, C_2) + f_4(C_2) \\ d(B_2, C_3) + f_4(C_3) \end{cases} = min\begin{cases} 6 + 8 \\ 10 + 7 = 14 \\ 4 + 12 \end{cases} \quad (1.6)$$

$$f_2(C_3) = \min\begin{cases} d(B_3, C_1) + f_3(C_1) \\ d(B_3, C_2) + f_3(C_2) \\ d(B_3, C_3) + f_3(C_3) \end{cases} = min\begin{cases} 13 + 8 \\ 12 + 7 \\ 11 + 12 \end{cases} = 19$$

So, there is three local optimized path which is from B_1, C_1, D_1 to E, B_2, C_1, D_1 to E, and B_3, C_2, D_2 to E.

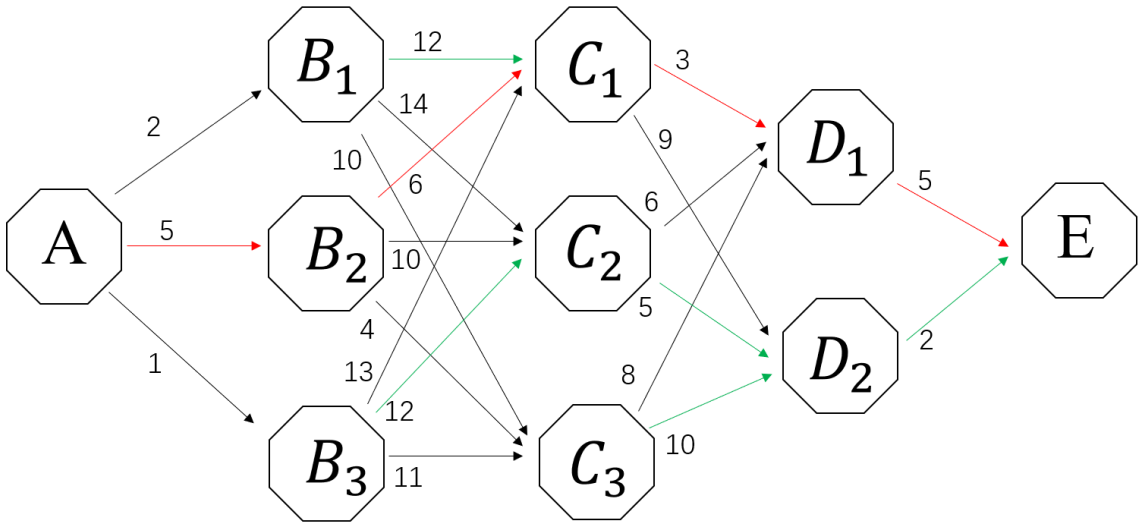

Fig. 14 – Dynamic programing step-4
source: https://blog.csdn.net/PRML_MAN/article/

Eventually, in the final step. as shown in Fig. 14. from A to B there is a total of 3 paths. And among these nine routes, there are three optimal routes with a distance of 2, 5, and 1 respectively.

$$f_1(A) = \min\begin{cases} d(A, B_1) + f_2(B_1) \\ d(A, B_2) + f_2(B_2) \\ d(A, B_3) + f_2(B_3) \end{cases} = min\begin{cases} 2 + 20 \\ 5 + 14 = 19 \\ 1 + 19 \end{cases} \quad (1.7)$$

So, the total optimized path which is from A, B_2, C_1, D_1 to E with the total cost of 19.

Additionally, there is also plenty of other methods for pathfinding. Like In 1986, Khatib first proposed the artificial potential field method and applied it to machine robot obstacle avoidance field, and modern cars can be seen as high-speed robots, so this method can also be applied to the field of obstacle avoidance path planning for automobiles area(Iswanto).

Or, sometimes multiple robots need to plan their path at the same time. Reynolds proposed the Boid model in 1987 to simulate the flocking of birds' behavior. In this model, each particle can perceive a certain range around the flight information of other individual particles in the particle, combined with its current flight state, make the next flight decision(Craig W. Reynolds).

In conclusion, with years of research by numerous scholars, the path-finding theory has been expanded and optimized significantly, In this project work, A star algorithm will be used and applied to the path planning of the material transportation robot.

2.2.5. COMPLETE COVERAGE PATH PLANNING

Different from all the previous theories, complete coverage path planning didn't just find the shortest path from two different specific positions, instead, It is a special path planning that requires robots to traverse all reachable areas in the environment.

In this project work, a software prototype to find and guide the shortest path for the robot and of assembly installed insulation panels on the roof, alongside an automatic building panel process arranging program is needed.

Thus, for the roof panels, we need completely cover the entire roof area with the required roof brick exactly in the required places. So, complete coverage path planning is needed.

The complete coverage path planning (CCPP) algorithm is usually used in special robot operation scenarios, such as robot cleaning, robot ground construction, etc. Such scenarios usually require the Footprint on the robot's running path to fully cover the ground of the area, to achieve full coverage of an area(Choi et al. 2009). The robot path planning methods we came into contact with in the early days are basically point-to-point (PTP) planning methods, such as A*, D*, etc., which are characterized by obstacle avoidance, the shortest running path, and no smoothness. I believe that many domestic Students who are new to mobile robot planning algorithms also start to get in touch with path planning them. The CCPP algorithm is a path-planning method to deal with the special operation of the robot. It should not be an algorithm in essence, but a collection of algorithms to achieve a certain function(Le et al. 2020).

The more thought used in full-coverage path planning is the artificial potential field method(Theresa Marie Driscoll). Adding some form of the artificial potential field to the planning area can guide the search direction of path planning, and at the same time, some point-to-point points around obstacles will be mixed in the middle. Full-coverage path planning can be divided into bow-shaped planning and back-shaped planning according to the coverage planning method. These are two planning methods that are currently used in the field of cleaning robots(Wang et al. 2022b).

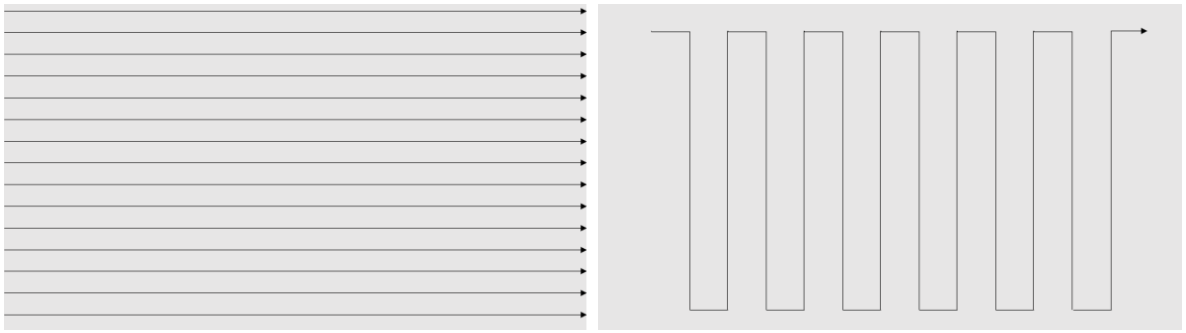

Fig. 15 – Artificial potential field

Add a one-sided potential field to the bow-shaped plan as shown in Fig .15 above, thus forming the bow-shaped path on the right.

In detail, the artificial potential field method is a virtual force method proposed by Khatib for robot motion planning. The basic idea is to concretize the influence of targets and obstacles on the robot's motion into an artificial potential field. The potential energy is low at the target and high at the block. This potential difference produces the attraction of the target to the robot and the repulsion of the obstacle to the robot, and their combined force controls the robot to move towards the target point along the negative gradient direction of the potential field. The artificial potential field method is easy to calculate, and the obtained path is safe and smooth. However, the complex potential field environment may produce local minimum points outside the target point, making the robot unable to reach the target(Yang et al. 2021).

But, those methods, In many cases, it is not sufficient to find any route that completely covers the field. Also, it is desired that the path also be optimal to minimize certain costs.

So, In recent years, neural network algorithms have been applied to full-coverage path planning. Utilize the self-learning, parallelism, and other characteristics of the neural network to enhance the "intelligence" of the robot and improve the coverage efficiency. Inspired by the similarity between the neural network structure and the grid map unit, Canadian scholar S. X. Yang proposed a full-coverage path planning algorithm for mobile robots based on biologically inspired neural networks, which will require full-coverage two-dimensional grid map units and biologically inspired neural networks. The neurons

of the network correspond one by one, and the real-time path planning of the robot to achieve full coverage is generated by the activity value of the neurons and the last position of the robot. The algorithm is completely based on the properties of the grid map unit (unsearched unit, searched unit, or obstacle), determines the input of the neuron, and directly calculates the activity value of the neuron. There is no neural network learning process, and the algorithm has good real-time performance. Can automatically avoid obstacles and escape from the dead zone.

The motion space of the mobile robot is expressed by the topological space composed of neural networks. The initial activity value of all neurons is 0, and the change of the activity value of each neuron is expressed as:

$$\frac{dx_i}{dt} = -A_i + (B - x_i)\left([I_i]^+ + \Sigma_{j=1}^{k}\omega_{ij}[x_i]^+\right) - (D + x_i)[I_i]^- \qquad (1.8)$$

Where k is the number of neural connections of an i-th neuron to these neighboring neurons within the receptive field. With the introduction of neural networks, complete coverage path planning became more complicated and functional(Yang and Luo 2004).

2.3. APPLICATION OF ROS IN CIVIL ENGINEERING

The Robot Operating System (ROS) is a comprehensive collection of software libraries and tools designed to assist in developing robot applications. These tools range from drivers to state-of-the-art algorithms and are supported by powerful developer tools. Moreover, ROS is an open-source platform, which means that the source code is freely available and can be customized and modified as needed(Morgan Quigley).

It is important to digitalize construction and bridge the gap between robotics and the building industry. Researchers at Fraunhofer center based in Bolzano, Italy, are developing a software interface that will enable mobile robots to find their way around construction sites(Mosler).

There is also the scientist who presents a flexible ROS-based software architecture for controlling unmanned vehicles. Experiments were conducted in various civil scenarios using a quadcopter, hexacopter, and a UAV simulator, and the results show that the proposed architecture is effective and can handle multiple vehicles and sensors. Therefore, the proposed architecture is a promising solution for controlling unmanned vehicles in civil applications(Al-Kaff et al. 2019).

Also, some researchers propose a security architecture for securing ROS-based applications, including an authorization server and cryptographic methods. A

demonstration with a collaborative robot shows the architecture is effective in ensuring data confidentiality and integrity(Bernhard Dieber et al. 2016).

Some researchers propose a method for integrating Timed Rebeca with ROS for modeling and verifying robot behaviors. Timed Rebeca is used to formally model robots' behaviors and verify properties using the Afra model checker. The method includes mapping a robot movement scenario to a Timed Rebeca model, mapping the Timed Rebeca model to ROS programming constructs, and generating ROS-based robotic programs automatically(Dehnavi et al. 2019)

Some researchers describe the design and implementation of BAICal, an autonomous surface vehicle developed at the University of Calabria for marine robotics research. The ASV includes a three-degree-of-freedom dynamical model and an extension to the GNC software architecture, with a manager module for autonomous mission coordination. The software is based on a module-based architecture using the ROS framework, making it suitable for different types of autonomous vehicles. The paper concludes that BAICal is a versatile and effective solution for marine robotics research(D'Angelo et al. 2022).

As shown before, ROS has many applications in the construction field, and in this project work will also be used as part of the simulation.

3 METHODS

3.1. PROJECT BACKGROUND

In this project, robots were required to be used in the roof construction field to perform tasks such as lifting and placing roofing materials and installing insulation which also can increase the speed and efficiency of the construction process, reduce the risk of injury to human workers, and improve the accuracy and quality of the work.

The use of two robots working in collaboration can further increase the efficiency and productivity of the project(Bloss 2016). And, implementing such a system requires careful planning and consideration of factors such as the programming, and the compatibility of the robots with the construction site and materials. And it's also essential to consider the actual field's construction conditions.

The first robot's mission to deliver roof panels involves using pathfinding algorithms to determine the best path to navigate from the stockpile point to the other robot while adhering to construction constraints and rules. The second robot's mission to plan the panels according to the DIN standard involves using algorithms to determine the optimal sequence for laying the panels and achieving some level of automation. Additionally, this robot is expected to monitor the supply of roof panels and send signals when missing panels to the other robot so that the supply can be replenished.

This project uses Fortran to extract BIM information from a .dwg file and uses Matlab to program and visualize the entire process and a robot such as the "Turtlebot" controlled by the ROS system can provide an effective means of simulating the roof planning process. The visualization aspect of Matlab allows for a more intuitive representation of the process and can aid in troubleshooting and refining the system. The robot platform, such as the "Turtlebot," controlled by the ROS system, can provide an effective means of testing and validating the algorithms and coordination between the two robots in a real-world environment.

However, the accuracy of the simulation may depend on how well the program is written and the "Turtlebot" robot's capabilities to reflect the actual construction site's delivery and roof planning process. It's worth mentioning here that different construction conditions require different but minor adaptions in the program. For example, in ROS, different roof layouts need the robot to move in differently. And that requires every time before we run the program, to input some basic information about the map of the construction fields. What ROS requires is the coordinate information point by point that robot go, this information includes the points x,y, and z in the map and the robot's

steering information, to get that information, the following is the process of using the BIM model in Fortran and Matlab and eventually in ROS.

But the downside is we also needed to change this program a little bit to adapt to different map conditions, which is where the program in this article is deficient and needs to be improved.

3.2. PROJECT METHODS DESCRIPTION

3.2.1. EXTRACT INFORMATION FROM AUTOCAD

The project work was given AutoCAD documents that include basic information about the roof, such as the roof layout shown in Figure. 16, from the designer on the construction side.

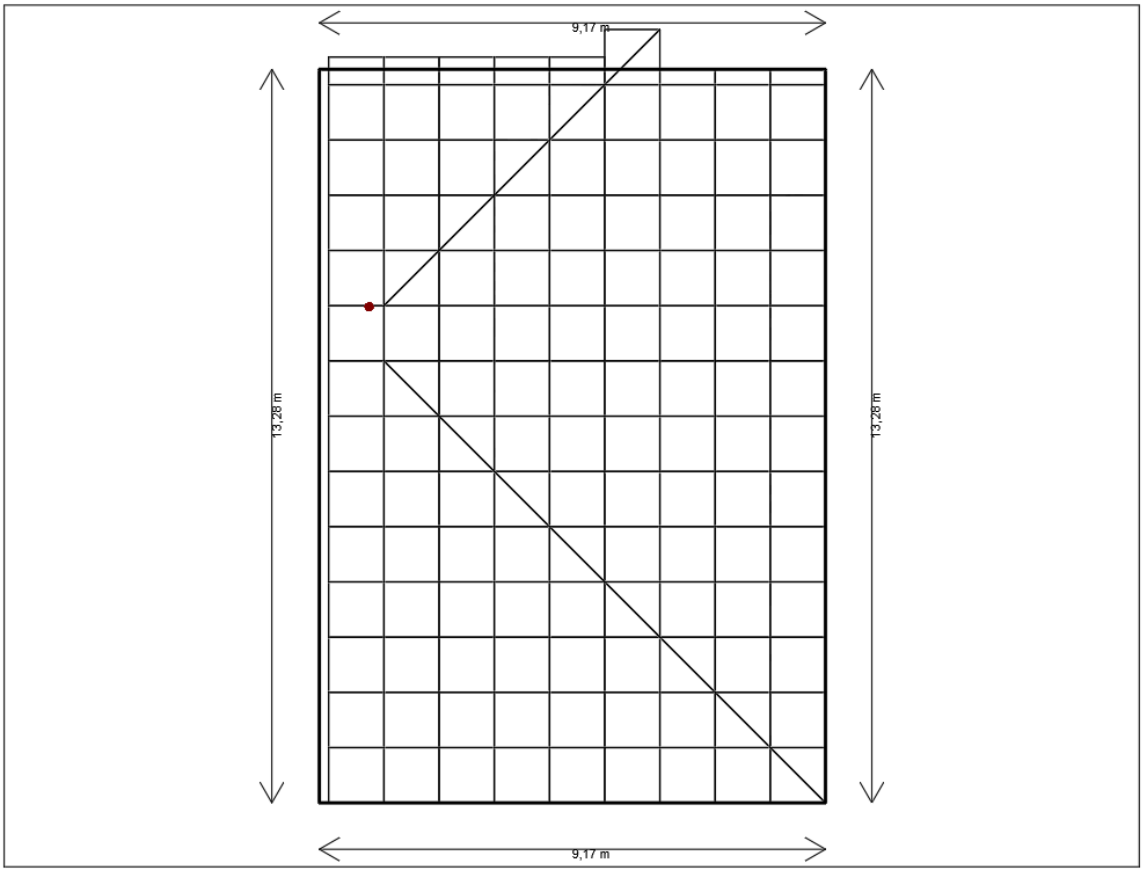

Fig. 16 – Layout of roof

To create a path-finding system, a detailed construction map is required, but the AutoCAD documents provided only show the layout, size, and thickness of the roof, making it difficult to obtain the necessary detailed information.

How to get every point coordinated and put them in the right places, eventually recreating such same roof layout in the path-finding program becomes the first problem that is needed to be solved. To obtain more detailed information for the path-finding system, it is necessary to extract the xlsx file that contains basic point information from AutoCAD.

After that, an .Xls table containing complete coordinates and other information will be provided by AutoCAD.

3.2.2. EXTRACT THE MIDDLE POINT USING THE FORTRAN PROGRAM

Although the .Xls table was generated from the previous steps, a detailed construction map is still required. However, it is possible to extract more information by obtaining the midpoint coordinates of each board from the previous table. A program designed specifically for this purpose was created to obtain the midpoint coordinates of each board.

The basic idea is to gather the discrete data from the .Xls file into a group of four points' coordinate data per cell. Then, the midpoint coordinates can be calculated as each cell's horizontal and vertical coordinates.

As shown in Fig. 17, a unit cell consisting of four points can return to the original diagram.

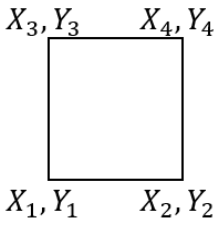

Fig. 17 – 4 units as a cell

The horizontal and vertical coordinates (X, Y) can be explained as:

$$\begin{cases} X = \frac{X_1+X_2}{2} \\ Y = \frac{Y_1+Y_2}{2} \end{cases} \quad (1.9)$$

Automatically calculating the midpoint coordinates in the program requires careful consideration, including recombining the data according to different cell locations and data order.

As an explanation of the Fortran midpoint program:

All the points' coordinates were input into the program and arranged automatically according to the row and column. The same row's points were assembled and those closed 4 points in Fig.25 were put together to get the middle points, then we arrange those points together.

Similarly, the same column's points were also put together, and use the program to get the middle point of those closed 4 points in the same column. By putting them together, a map made of the middle point can be created, which can be used in the following program and steps.

And using a txt file command, the X and Y value of middle points is extracted from Fortran. Using Gnuplot (a plot derivative system in Fortran), those middle points can be shown in one single diagram.

Thus, middle point extraction can be achieved by previous Fortran programs.

3.2.3. PATH PLANNING AND OBSTACLE AVOIDANCE SYSTEM FOR CARRIAGE ROBOT

a. Map

In this article, The method is to study the full-coverage path planning of mobile robots in the grid map, so it is necessary to construct a grid map for the working area.

And due to the reason that Matlab has better programmability and applicability, all the following codes will be written in Matlab.

First, we set the length of the row and column in order to create a raster map. Then, call the center of each square grid a node. Since we need to consider the obstacle on the map, some obstacle avoidance mechanisms are needed to be set here. So, we set each cell to have two different statuses: one is walkable while another is un-walkable.

Here the map's characteristics are set.

b. Preprocessing

Before running the program loop, several values should be set.

Firstly the location where the stockpile point will be set as a starting point for the robot. And this is where every time this carriage robot gets supply. Secondly, the location of the

first brick-laying robot will be set as the endpoint. That's because we need the carriage robot to deliver roof panels to the brick-laying robot when It's running out of roof panels.

Then, Define two list collections: openList and closeList. OpenList table by the un-inspected nodes and the closeList table is composed of the inspected nodes. Here un-inspected nodes mean the point at which the robot has already finished the roof panel laying and inspected nodes means the point still needs to be laid by the roof panel.

c. Program loop

There are 4 nodes around the parent node A, which are defined as child nodes and they represent 4 directions around the parent node. Put child nodes into the openList, it becomes the object to be investigated. Here, the parent node means the point where the robot laying right now, and the child node means the 4 points that surround the parent node and are about to be passed by a robot.

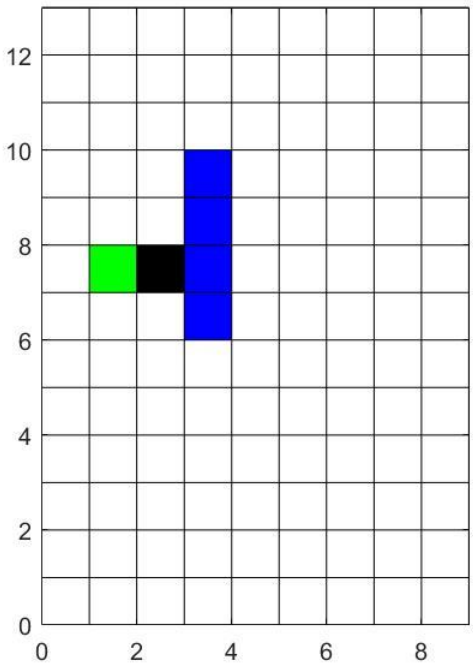

Fig. 18 – Moving cost calculation g(n)

If a node is neither in openList nor in closeList, it means the node was not passed by the robot before. The basis for judging whether the path is good or bad is the movement cost, and the single-step movement cost is taken Manhattan calculation method. The distance between two points is measured along axes at right angles. In a plane with point1 at (x1, y1) and point2 at (x2, y2), it is as follow functions:

$$D(point1\text{-}point2) = |x1-x2| + |y1-y2| \qquad (1.10)$$

And, the generation of moving a node horizontally and vertically valence is defined as 1. The moving cost evaluation function is:

$$f(n)=g(n)+h(n) \qquad (1.11)$$

f(n) is from the initial state the cost estimate of the state via state n to the target state, g(n) is in the state space the actual cost from the initial state to state n during the interval, h(n) is from state n to target state the estimated cost of the best path.

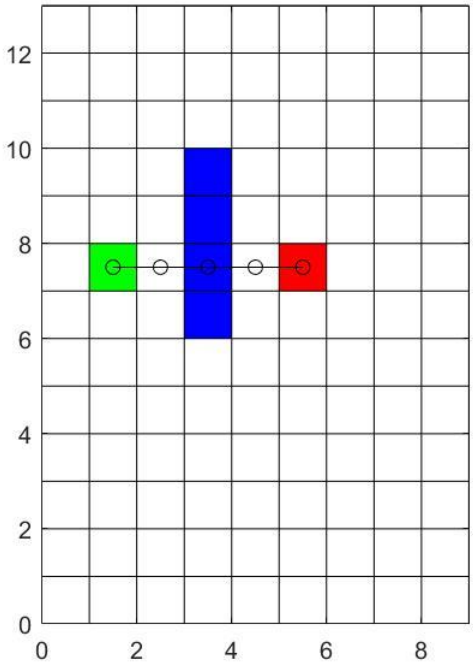

Fig. 19 – Moving cost calculation h(n)

Here we make an example in Fig. 18 and Fig. 19. Set the start point as green and the end point as red. To find the shortest path from those points, the moving cost evaluation started to form the 4 nodes around the parent node A (start point), the meaning of estimation is to ignore whether the remaining paths contain obstacles (non-walkable), completely calculated according to Manhattan formula to calculate the cumulative cost of only moving horizontally or vertically: Go right four cells. The distance is 4×1 = 8.

And for g(n), set the child node as black, from the start point to the child node is a vertical move, so the distance is 1×1 = 1 because there is 1 vertical move between the start point and the child point. Here g(n) = 1. Therefore, the total cost of moving from the start node to the end node is 8+1 = 9. (f(n)=g(n)+h(n)).

The previous step only calculates the moving cost from the start point to the one child node, and all the movement from node to node shall by analogy, calculate the remaining 3 child nodes in the current openList respectively. Move and calculate the cost, select the minimum cost node, and move it to the closeList.

Then, select the cell node I with the smallest f(n) value from the openList, and from the openList take it out, put it in closeList, and use it as the new parent node.

Check all its adjacent child nodes, ignoring obstacles and non-walkable nodes, Ignore nodes that already exist in the closeList; if the square is not in the openList, add them to the openList.

Check this path if some adjacent node is already in the openlist, when the path is better, that is, via the current node (the node we selected). Whether to reach that node has a smaller g value. If not, do nothing to that point.

And so on, repeating. Once the target node is searched, complete the path Search, and end algorithm. And the program loop ends here.

3.2.4. PATH PLANNING AND OBSTACLE AVOIDANCE SYSTEM FOR ROOF PANEL LAYING ROBOT

a. Map

As mentioned before, a grid map is required for the robot's path planning. A grid map is a representation of the working area that divides it into a regular grid of cells, with each cell representing a portion of the space.

In Fig. 20, there is some roof cell marked as red, which means those part have Irregular shape. In detail, those parts were not laid by construction robots but by human workers. Thus, those cells are unwanted parts of the roof-laying map. So In the program, those points were set with a different value than regular roof cells.

Also, to accomplish full coverage roof layer planning, some switch points need to be set. This part will be explained in the next chapter.

b. Preprocessing

To start the program, the start point where the robot start shall be set.

The logic behind this robot's movement is to install the roof panel according to the regulations of the relevant building construction process, the layout plan must be set down already before the construction. This laying scheme is set according to the DIN18531(Christian Herold) regulations in the order drawing in the red arrow as shown in Fig. 2. This sequence is because the roof panel has different thicknesses, and the thickness is decreasing from the outer part to the inner part, so the laying sequence is as Fig. 2. The thicker roof panel on the out layer are laid first (the roof panel has the thickness that robot can not walk on it, thus the outer part were laid first).

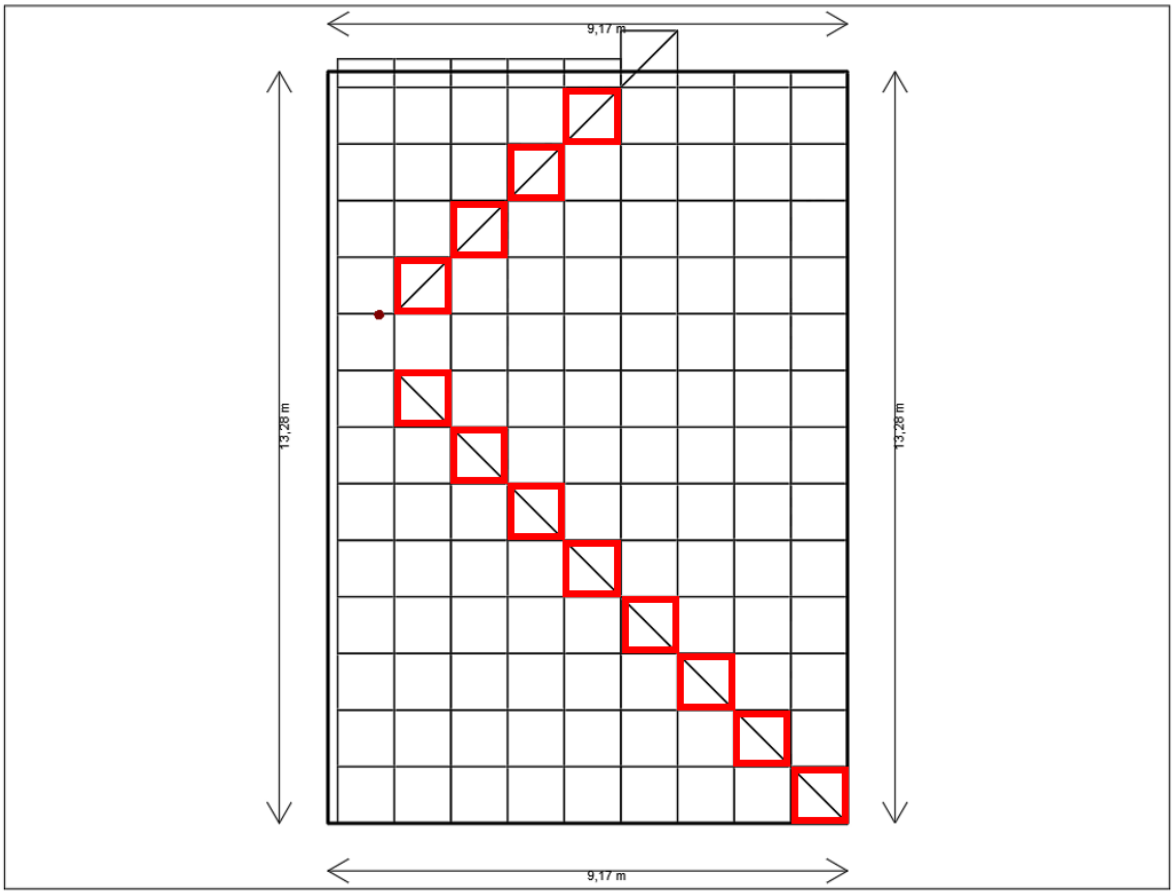

Fig. 20 – Layout of roof

c. Program loop

To set the logic behind the program, we wish this roof panel laying robot to move in two different ways: clockwise and anti-clockwise. So that the robot can move according to the install sequence shown in Fig. 2.

Firstly, the robot shall have four different moveable directions: left, right, up, and down. And the cell in the program shall have three different statuses: obstacle, uncleaned, and cleaned.

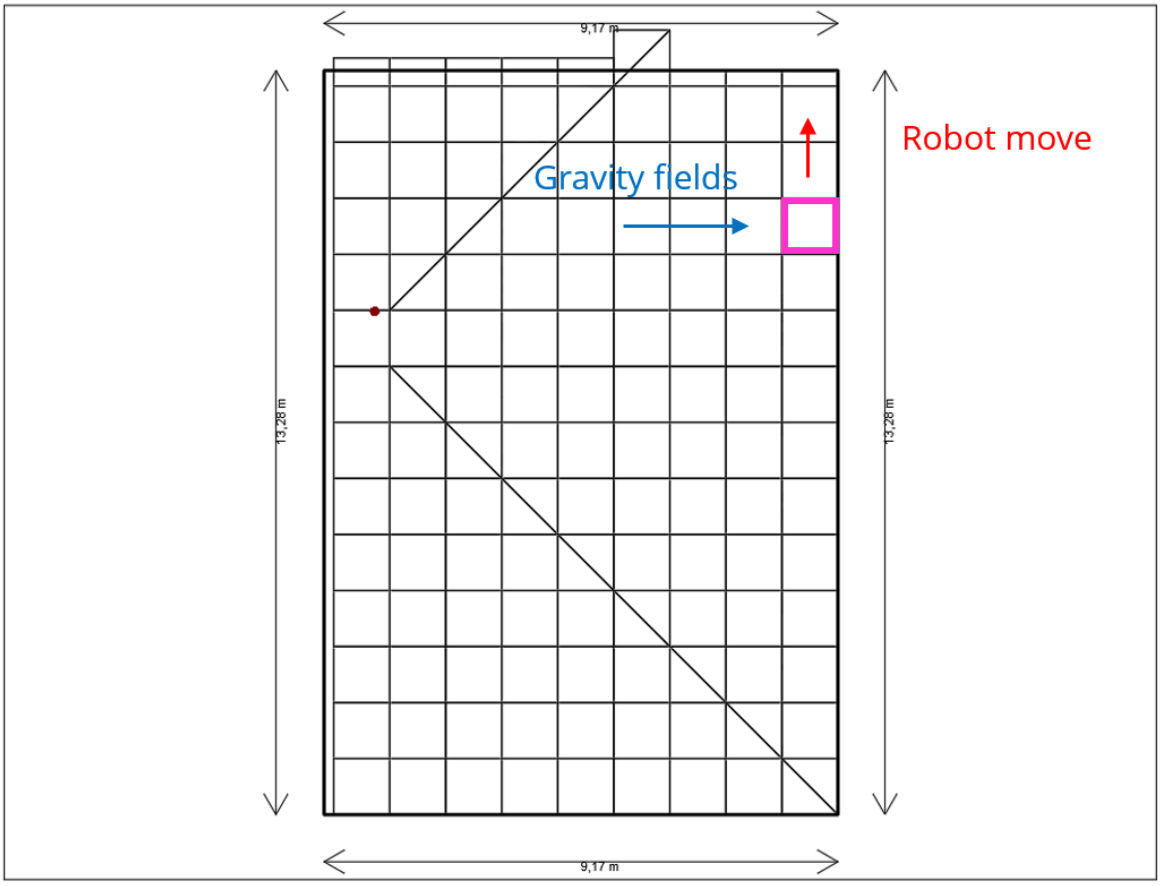

Fig. 21 – Influence of gravity fields

In order to accomplish so-called “full coverage” on the roof, the robot is expected to move tightly to the edge of the map and avoid the cleaned area and obstacles. So that every cell of the roof can be covered by the robot.

In the clockwise part, the robot’s four directions of movement can have some restrictions like gravity fields. Imagine that, as shown in Fig. 21, when the robot is moving along the edge, some artificial potential fields limited the robot’s movement and force it to stick to every outer edge and also move clockwise.

Likewise, In the anti-clockwise part, the robot’s movement is limited by an artificial potential field, it can move in four directions but must stick to the outer edges to secure the robot has covered all the possible cells along the roof layer planning process.

With the combination of the clockwise part and anti-clockwise part, this robot is expected to move according to the previous layout plan.

3.3. HOW TO USE THE TURTLEBOT(ROS)

In this project work, to visualize the entire robot moving process, the Turtlebot system was used.

For robots like Turtlebots, there exists a speedometer to get the robot's speed data and also use it to determine where the robot is. Thus the positioning control of the robot in the virtual environment and the accurate matching of the map are very important.

In this project work, the Turtlebot robot was used to simulate the roof construction process, which runs under the ROS system. ROS currently in this project runs on Unix-based platforms. And ROS software is mainly in running Ubuntu.

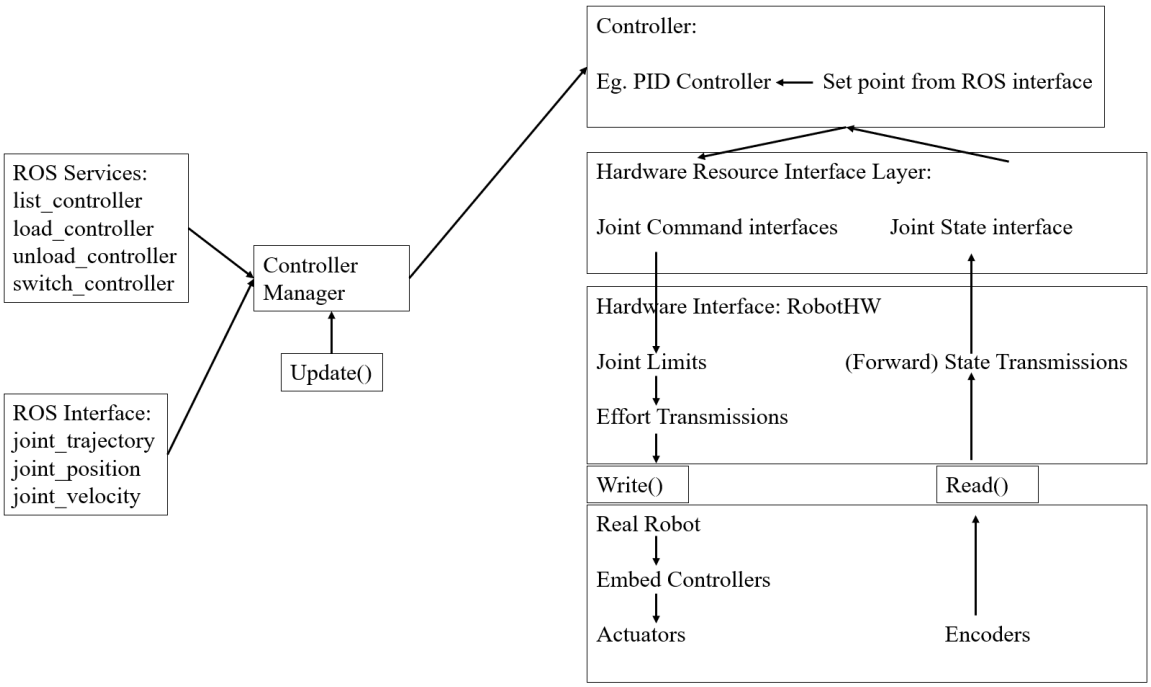

Fig. 22 – ROS control flow chart
source: https://joss.theoj.org/papers/10.21105/joss.00456

To use the calculated coordinate information from Matlab, a Fortran program was written to extract the txt file and write the coordinate file which can be accepted by Python. And from this, data transport and process from BIM to ROS were accomplished.

And to control and simulate Turtlebot's movement(Chitta et al. 2017) here in Fig. 22 is a flow chart that shows the process of ROS control.

Before inputting the coordinate information into the ROS system, several presets are needed. First, the Turtlebot simulation needs a map of the Gazebo, here the map of one

room in Bauinfomatic institute was used. Inside, one room was chosen to play the role of the platform of this simulation.

As shown in Fig.23, this building map was used as a platform in the simulation.

Initially, It is necessary to set a relative coordinate origin point for both the car and the map. And this point was set manually along the edge of one room.

Fig. 23 – Map of one floor in Bauinfomatic institute

To start the Turtlebot's system first is to set the map, as set previously, and then in the simulation environment, load the Gazebo environment.

After that, it is necessary to load the Turtlebot into the environment, set the initial location of the Turtlebot, and use it to laser scan the visual environment, so that the map can be matched.

Finally, after setting the map and inputting the Matlab-generated points into the Python control program for the Turtlebot, the Turtlebot will follow the input point coordinate from the start point to the endpoint.

In the simulation, eventually, the Turtlebot will follow the calculated point step by step, at the same time, automatic obstacle avoidance was also accomplished. By using the A-star program in the Turtlebot control.

Here, the detailed procedure will be explained in chapter 5.

4 SIMULATION

4.1. EXTRACT INFORMATION FROM AUTOCAD

From the construction side's designer, this project work was provided with some AutoCAD documents that conclude basic information about the roof. For example, the layout of the roof was provided as shown in Fig. 24.

And In order to create a path-finding system, first a detailed construction map is needed. But from the AutoCAD documents, we can only visualize the layout, the size, and the thickness of the roof.

How to get every point's coordinate and put them in the right places, eventually recreating such same roof layout in the path-finding program becomes the first problem that is needed to be solved.

N	O	P	Q	R	S	T	U	V
point X	point Y	point Z	width	angle1	start X	start Y	start Z	length
0.1214	0.0000	0.0000	0.0000	180	0.8357	0.0000	0.0000	0.7143
0.1214	0.7143	0.0000	0.0000	90	0.1214	0.0000	0.0000	0.7143
0.8357	0.7143	0.0000	0.0000	0	0.1214	0.7143	0.0000	0.7143
0.8357	0.0000	0.0000	0.0000	270	0.8357	0.7143	0.0000	0.7143
0.8357	0.0000	0.0000	0.0000	180	1.5500	0.0000	0.0000	0.7143
0.8357	0.7143	0.0000	0.0000	90	0.8357	0.0000	0.0000	0.7143
1.5500	0.7143	0.0000	0.0000	0	0.8357	0.7143	0.0000	0.7143
1.5500	0.0000	0.0000	0.0000	270	1.5500	0.7143	0.0000	0.7143
3.6929	0.7143	0.0000	0.0000	180	4.4071	0.7143	0.0000	0.7143
3.6929	1.4286	0.0000	0.0000	90	3.6929	0.7143	0.0000	0.7143
4.4071	1.4286	0.0000	0.0000	0	3.6929	1.4286	0.0000	0.7143
4.4071	0.7143	0.0000	0.0000	270	4.4071	1.4286	0.0000	0.7143
4.4071	0.7143	0.0000	0.0000	180	5.1214	0.7143	0.0000	0.7143
4.4071	1.4286	0.0000	0.0000	90	4.4071	0.7143	0.0000	0.7143
5.1214	1.4286	0.0000	0.0000	0	4.4071	1.4286	0.0000	0.7143
5.1214	0.7143	0.0000	0.0000	270	5.1214	1.4286	0.0000	0.7143
2.2643	0.7143	0.0000	0.0000	0	1.5500	0.7143	0.0000	0.7143
2.9786	0.7143	0.0000	0.0000	90	2.9786	0.0000	0.0000	0.7143
4.4071	0.7143	0.0000	0.0000	0	3.6929	0.7143	0.0000	0.7143
3.6929	0.0000	0.0000	0.0000	270	3.6929	0.7143	0.0000	0.7143
2.9786	0.0000	0.0000	0.0000	180	3.6929	0.0000	0.0000	0.7143
3.6929	0.7143	0.0000	0.0000	90	3.6929	0.0000	0.0000	0.7143
5.1214	0.7143	0.0000	0.0000	0	4.4071	0.7143	0.0000	0.7143

Fig. 24 – Xls information

So, we need Extract the xlsx file which concludes basic point information from AutoCAD. But the original diagram is locked as a whole we can't get every line and point's coordinate

from the program, and any attempt would result in one single piece of information about the entire entity.

Thus, the “Explode” command can be used to decompose a compound object into its component objects. For example, Polyline will discard any associated width or tangent information. For polylines with width, lines, and arcs are generated along the center of the polyline. After getting a line-by-line, point-by-point diagram. The “Data extraction” command can be used to directly extract not only coordinate but also all the relevant information. The “Data extraction” command can extract graph data from external sources and merge the data into data extraction tables or external files. Search and export object properties, block attributes, and drawing information to data extraction sheets or external files, and specify data links to Excel spreadsheets.

After the previous steps, an Xls table containing complete coordinates and other information will be provided by AutoCAD. Which include, for example, coordinates and start point, and length. Witch was shown in Fig. 24.

4.2. EXTRACT THE MIDDLE POINT USING THE FORTRAN PROGRAM

From the previous steps, the Xls table was calculated. But still, a detailed construction map is needed. Fortunately, we can extract the midpoint coordinates of each board from the previous table, so more information can be extracted.

A program specific to calculating middle points was created to solve that question.

The fundamental concept is to assemble the discrete data from the Xls file into a collection of cells as in Fig. 18, that is to say, using four points’ coordinate data as one group, and then, the midpoints can be expressed as the horizontal and vertical coordinates. As shown in Fig. 25, a unit cell consisting of four points can return to the original diagram. And in Fig. 17, an 4 unit cell is provided.

And to automatically calculate this in the program, there is a lot to be considered, like recombining the data according to different cell locations and data order, or how to handle those unwanted points. Sometimes there is a missing data point which might be a critical problem when the program is running, but here, the following program makes such problems not so critical:

Because this program uses each four-point cell to calculate one middle point, once there are two four-point cells next to each other, the boundary points will be repeatedly used In the program.

Thus, this kind of repeating can avoid missing data unless the missing data point is located at the edge, where can't be repeated.

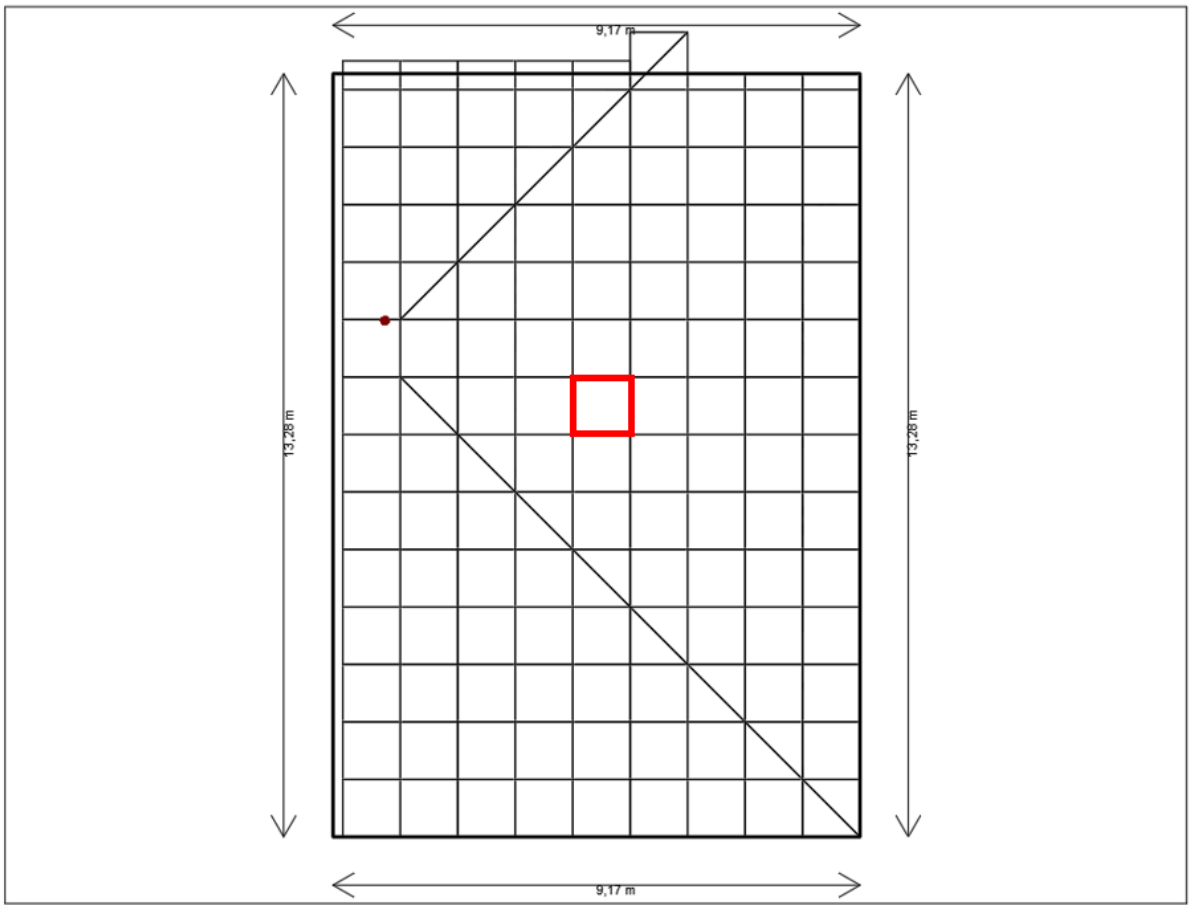

Fig. 25 – One cell

Here is a detailed explanation of the midpoint program:

Firstly, the unwanted points (those Points that are not within the range of the installation) were deleted. This is for the sake of reducing computation time, and sometimes there is some error point caused by AutoCAD, this progress can eliminate the adverse effects of these points.

Secondly, Import the horizontal and vertical coordinates of all points within the installation range into a txt file, which can be read by the Fortran program later.

Thirdly, inside the Fortran program, do a loop with the number of all points. In every loop, set the difference between the horizontal coordinates of the two consecutive data as q, and set the difference between the vertical coordinates of the two consecutive data as p.

Then divide into two kinds of hypotheses: one is if p equals zero, and this means that the difference between the vertical coordinates of the two points is the same. Thus, those points are in the same row. And the second one is if p equals 0.7143 (which is the difference between two rows per cell unit). this means that the difference between the vertical coordinates of the two points is next to each other. Thus, those points are in the neighboring row.

It is also worth mentioning there exists the third condition: the difference between the horizontal coordinates of the two consecutive data is bigger than 0.7143, which means those two points are located In a different row But that condition is unwanted, so It's not used.

And, when the result fulfills the first assumption (those points are in the same row). Divide again. If q equals zero, that means that the difference between the horizontal coordinates of the two points is the same. Thus, those points are in the same column. By adding 0.35715 (which is the half-length value of two neighboring points) to those points' both x and y values, a middle point's coordinates can be calculated.

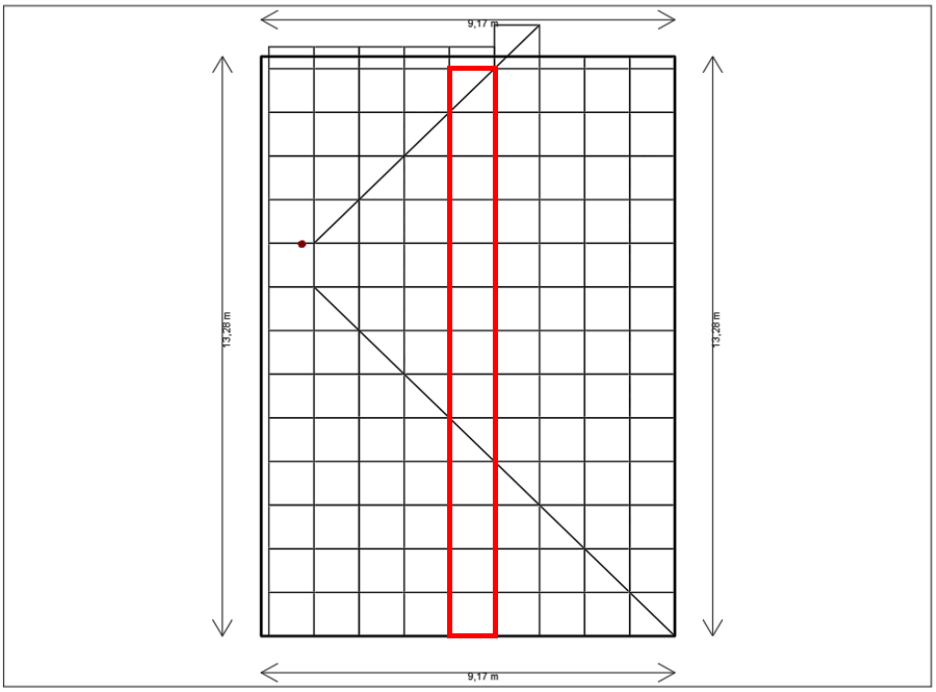

Fig. 26 – Same column

And the second one is if q equals 0.7143 (which is the difference between two columns per cell unit). this means that the difference between the vertical coordinates of the two points is next to each other. Thus, those points are in the neighboring column. By adding

0.35715 (which is the half-length value of two neighboring points) to those points' both x and y values, a middle point's coordinates can be calculated.

Similarly, there exists the third condition: the difference between the vertical coordinates of the two consecutive data is bigger than 0.7143, which means those two points are located In a different column. But that condition is unwanted, so It's not used.

When the result fulfills the second assumption (those points are in the neighboring row). Divide again. If q equals zero, that means that the difference between the horizontal coordinates of the two points is the same. Thus, those points are in the same column. By adding 0.35715 (which is the half-length value of two neighboring points) to those points' y values, and minerals 0.35715 to those points' x values. a middle point's coordinates can be calculated.

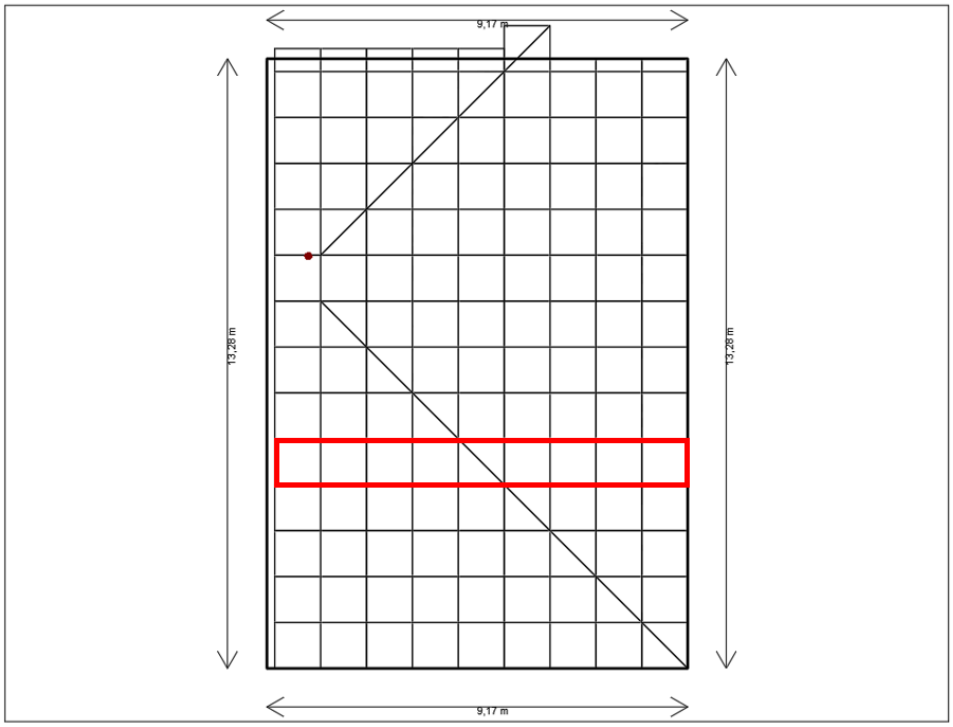

Fig. 27 – Same row

And the second one is if q equals 0.7143 (which is the difference between two columns per cell unit). this means that the difference between the vertical coordinates of the two points is next to each other. Thus, those points are in the neighboring column. By adding 0.35715 (which is the half-length value of two neighboring points) to those points' both x and y values, a middle point's coordinates can be calculated.

Similarly, there exists the third condition: the difference between the vertical coordinates of the two consecutive data is bigger than 0.7143, which means those two points are located In a different column. But that condition is unwanted, so It's not used. Fig. 26 and Fig. 27 show what the same column and row look like.

The robot is expected to move along those rows and columns.

```
1. do a=1,n
2.     do b=1,n
3.         q=x(a+1)-x(a)
4.                 p=y(b+1)-y(b)
5.         if(p.eq.0)then
6.                 if(q.eq.0.7143)then
7.                         c(a)=x(a)+0.35715
8.                             d(b)=y(b)+0.35715
9.                     elseif(q.eq.0)then
10.                        c(a)=x(a)+0.35715
11.                            d(b)=y(b)+0.35715
12.                    else
13.                        error=error+1
14.                    endif
15.                elseif(p.eq.0.7143)then
16.                    if(q.eq.0.7143)then
17.                        c(a)=x(a)+0.35715
18.                            d(b)=y(b)+0.35715
19.                    elseif(q.eq.0)then
20.                        c(a)=x(a)-0.35715
21.                            d(b)=y(b)+0.35715
22.                    else
23.                        gapcollum=gapcollum+1
24.                    endif
25.                else
26.                    gaprow=gaprow+1
27.                endif
28.
29.         enddo
30. enddo
```

After previous work, eventually, all the middle points coordinate were calculated. And using a txt file command, the X and Y value of middle points is extracted from Fortran. Using Gnuplot (a plot derivative system in Fortran), those middle points can be shown in one single diagram.

To extract X and Y values of middle points from a txt file in Fortran, you can use the "READ" statement. Then, to plot these points using Gnuplot, you can use the "plot" command in Gnuplot, specifying the file name, column numbers for X and Y data, and any additional formatting options. This will generate a scatter plot with the middle points labeled "Middle Points". Note that Gnuplot is not a derivative of Fortran, but rather a separate plotting program that can be used with various programming languages.

Thus, middle point extraction can be achieved by previous Fortran programs. As shown in Fig. 28.

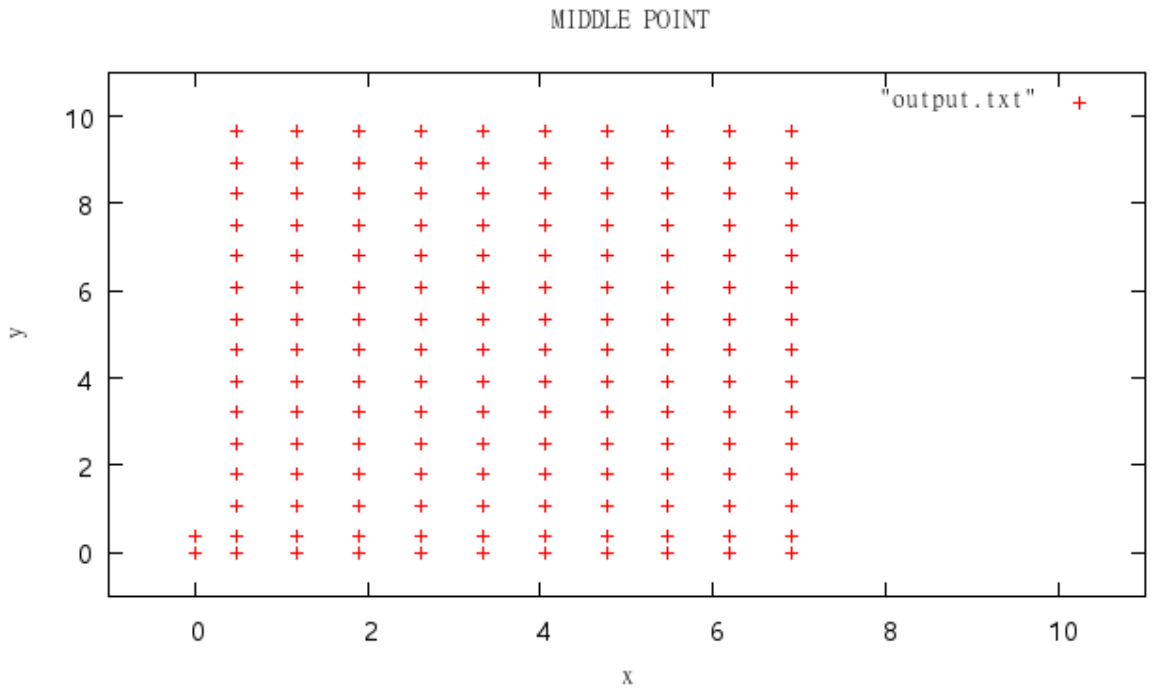

Fig. 28 – Extracted middle point in Gunplot

Those points can be further used in the following program to represent roof panels in a 2-dimension map. In detail, one middle point stands for one roof bricklayer, and those middle points can visualize the movement of the robot properly.

Here, this project work has accomplished the information transformation from the BIM model into detailed programming.

4.3. PATH PLANNING AND OBSTACLE AVOIDANCE SYSTEM FOR CARRIAGE ROBOT

a. Map

In this project work, The program takes advantage of the full-coverage path planning of mobile robots in the grid map, here a grid map for the working area is built.

To construct a grid map, data about the working area shall be collected. This might involve taking measurements or performing surveys to determine the size and shape of the area, as well as identifying any obstacles or other features that need to be represented on the map. As shown in the previous chapter, using the middle to represent each roof panel cell was applied, and the data was collected.

Grid map involves dividing the workspace into a regular grid of cells, with each cell representing a small portion of the workspace. The size and resolution of the grid will depend on the specific requirements of different projects. Once the data is prepared, the next is to create the grid for the map. This involves dividing the workspace into a regular grid of cells, with each cell representing a small portion of the workspace. Then it's necessary to populate the grid with data about the workspace. This might involve marking cells as worked or unworked or obstacles based on the presence of obstacles. Validate the grid Is also important to ensure that It accurately represents the workspace. This might involve performing simulations or testing with a robot to ensure that the map accurately represents the environment and that the robot can navigate it effectively.

Overall, constructing a grid map is an important step in full-coverage path planning for mobile robots. By accurately representing the environment in the grid map, plan paths that avoid obstacles and cover the entire workspace, enabling the robot to effectively and efficiently perform its tasks.

In this project work the fundamental concept is to set a grid map that can represent all the needed information. Firstly, the working area's width and length can be set by defining 10 and 14 as cell numbers according to the previous middle point calculation. Then we generate the map using the "ones" function to create a matrix consisting of all 1s, which has the length of roomlength+1 and the width of roomlength+1. This is due to the reason that we need one extra row and one extra column to set the outer boundary as a barrier, so the robot's movement can be limited in a set area.

Secondly, start_node shall be set, this is the location where the carriage robot starts to install the roof panel. target_node is also needed but differently, it's from the location of the roof panel laying robot. Here in this program, the obstacle is set as "obs", which can

be customized to any size's matrix to simulate the obstacles which the robot may occur in the construction field. And at mentioned before, The cells that human workers laid have an irregular shape and are marked in red to indicate that they are unwanted parts of the roof-laying map. As shown in Fig. 29.

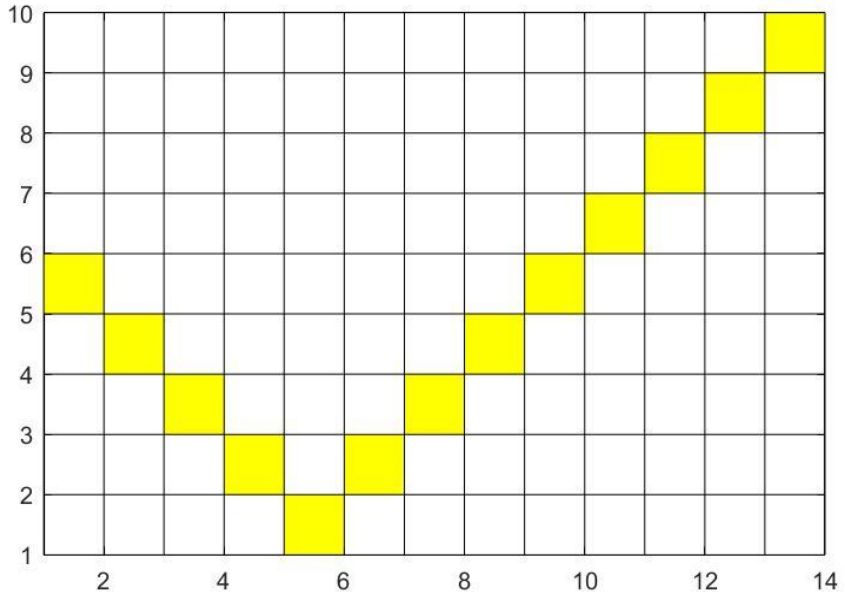

Fig. 29 – Human part marked in yellow in the Matlab program

To handle these irregular cells in the program, the points representing them were set with a different value than the regular roof cells. This could mean that they were assigned a different attribute or property that distinguishes them from the regular cells, or they were excluded from certain calculations or analyses that were performed on the regular cells. Overall, this approach can help to ensure that the roof-laying map accurately represents the construction process and enables efficient analysis and optimization of the construction workflow. This part was shown in Fig. 2.

```
1. for i = 1:m
2.     plot([0,n], [i, i], 'k');
3.     hold on
4. end
```

And to better visualize the map, grid lines shall be drawn in order to plot. Following the row and column settings, From 0 to 10 set each row's line using the function "plot" to set

line style, marker symbol, and color. Likewise, From 0 to 14 set each column's line using the function "plot" to set line style, marker symbol, and color.

Also, it's important to use the "axis equal" command to geometric axes such that each axis has evenly spaced ticks. And to set the limitation of upper and lower limits of the XY axis.

```
for i = 1:size(obs,1)
    temp = obs(i,:);
    fill([temp(1)-1, temp(1), temp(1), temp(1)-1],...
        [temp(2)-1, temp(2)-1 , temp(2), temp(2)], 'b');
end
for i = 1:size(humanpart,1)
    temp2 = humanpart(i,:);
    fill([temp2(1)-1, temp2(1), temp2(1), temp2(1)-1],...
        [temp2(2)-1, temp2(2)-1 , temp2(2), temp2(2)], 'y');
end
```

To draw obstacles, and start and end node color cell blocks, this experiment simulation uses "fill" to create a filled polygon (vertex colors specified by C) from the data in X and Y. C is a vector or matrix used as an index into the colormap. But differently, in this simulation, the area that needs to fill is not a single point but a cell block, so the X is defined by " start_node(1)", and the Y is defined by "start_node(2)". And set the color of the start point as green.

Considering there may be multiple obstacles on the map, use "size " to calculate the user-defined obstacle's number. Where A is the matrix whose dimensions are being queried, and sz is a row vector containing the length of each dimension of A. Then, create a loop using the "for" sentence, to fill each obstacle cell from number 1 till all the obstacle cells are filled. And set the color of the obstacle point as blue.

b. Preprocessing

```
for i = 1:size(openList,1)
    openList_path{i,1} = openList(i,:);
    openList_path{i,2} = [start_node;openList(i,:)];
end
for i = 1:size(openList, 1)
    g = norm(start_node - openList(i,1:2));
    h = abs(target_node(1) - openList(i,1)) + abs(target_node(2) - openList(i,2));
    f = g + h;
    openList_cost(i,:) = [g, h, f];
end
```

First of all, two list collections: openList and closeList shall be defined. Also, there should be a mechanism that calculates closeList_cost, but at the beginning, the closeList_cost is zero and closeList is defined as to start_node. Then, a subroutine special to find the child nodes set, this part will be explained in detail later.

About the openList's initialization, setting the openList as the child nodes, as mentioned before, there exist four child nodes in four different directions node. Using the "for" loop here for each child node to calculate each point's "openList_path" value, which is the distance between the parent node and the child node.

Then, the evaluation function for the A* algorithm is defined as:

$$f(n)=g(n)+h(n) \qquad (1.11)$$

where:

f(n) is the estimated total cost of the cheapest path from the initial state to the goal state that goes through node n. g(n) is the path's cost from the initial state to openlist.

$$g = ||(start\_node - openList)| | \qquad (1.12)$$

h(n) is a heuristic estimate of the cost from the openList to the target node.

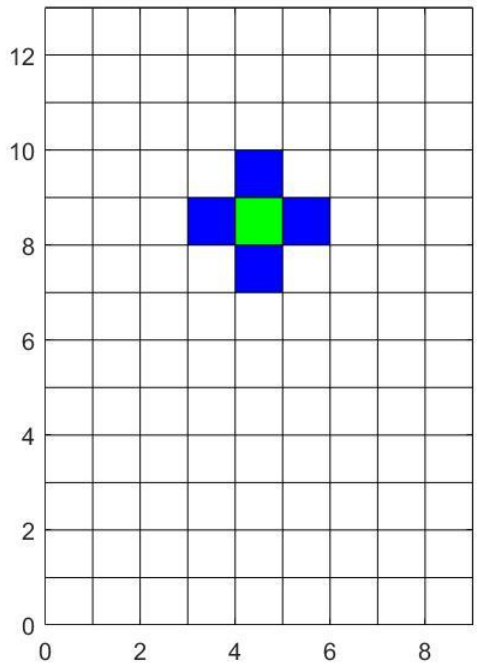

Fig. 30 – Child_nodes in the Matlab program

$$h = |(target\_node(x) - openList(x))| + |(target\_node(y) - openList(y))| \qquad (1.13)$$

So in the end, openList_cost can be written as:

$$f(n)=g(n)+h(n) \qquad (1.11)$$

Then start searching for the node with the smallest moving cost from openList, and use the "min" function to start searching for the node with the smallest moving cost from openList. After those steps, set parent_node as the minimum cost node from the openList, so that we can have a new start node.

c. Program loop

First, here explain the detailed structure of the subroutine which plays the role of searching child_nodes. To start with, set child_nodes as an empty matrix and set the working area inside the row and column limit. The first left child_node can be defined as follows:

[parent_node(1)-1, parent_node(2)], where 1 means the x-axis and 2 means the y-axis, so here is the coordinate representation of the relationship between parent_node and first left child_node. "parent_node(1)-1" means the child_node's x-axis equal to parent_node's x-axis minus 1, and "parent_node(2)" represents child_node's y-axis equal to parent_node's y-axis. Then, use "inpolygon" command to decide whether this point is inside or on the edge of the polygonal region, this step can avoid some error points. The child_nodes are shown in Fig. 30

```
1. if inpolygon(child_node(1), child_node(2), field(:,1), field(:,2))
2.       if ~ismember(child_node, obs, 'rows')
3.             child_nodes = [child_nodes; child_node];
4.       end
5. end
```

Then, use the command "ismember" to determine whether an array element is a set array member. This step is to judge whether the point is one of the obstacle points and if it is, this point would be expected from child nodes, this is how this program realizes automatic obstacle avoidance. In detail, use the "for" loop to compare whether this point is included in the obstacle array. And refresh those four child_nodes to avoid all the obstacle points.

Similarly, the upper child_node can be defined as follows:

[parent_node(1), parent_node(2)+1], is the coordinate representation of the relationship between parent_node and upper child_node. "parent_node(1)" means the child_node's x-axis equal to parent_node's x-axis, and "parent_node(2)+1" represent child_node's y-axis equal to parent_node's y-axis plus 1.

And the right and lower points can be defined as follows:

[parent_node(1)+1, parent_node(2)], is the coordinate representation of the relationship between parent_node and right child_node. “parent_node(1)+1” means the child_node’s x-axis equal to parent_node’s x-axis plus 1, and “parent_node(2)” represent child_node’s y-axis equal to parent_node’s y-axis. [parent_node(1), parent_node(2)-1], is the coordinate representation of the relationship between parent_node and lower child_node. “parent_node(1)” means the child_node’s x-axis equal to parent_node’s x-axis, and “parent_node(2)-1” represent child_node’s y-axis equal to parent_node’s y-axis minus1.

After defining and refreshing those four child_node points, Excluding nodes that already exist in closeList, is needed:

```
1. delete_idx = [];
2. for i = 1:size(child_nodes, 1)
3.       if ismember(child_nodes(i,:), closeList , 'rows')
4.             delete_idx(end+1,:) = i;
5.       end
6. end
7. child_nodes(delete_idx, :) = [];
```

To start with, define a matrix that is empty and give the name “delete_idx”. Then, use the “for” loop from 1 to the total number of child_nodes to search all the child_nodes, and again, use the combination of “if” and “ismember” to determine whether an array element is a set array member(whether this child_node is included in closeList). And when the result is “included”, use “delete_idx(end+1,:)” to add this child_node into delete_idx and then, delete it. Here this program successfully excludes the child_nodes that already exist in closeList. So, from this subroutine, all the child_nodes were found around the parent_node, and the obstacles were avoided. The output is refreshed child_nodes. Back to the main program loop, first set an index “flag” equal to 1, this will be used at the end of the program to finish the loop.

```
1.       for i = 1:size(child_nodes,1)
2.             child_node = child_nodes(i,:);
3.             [in_flag,openList_idx] = ismember(child_node, openList, 'rows');
4.             g = openList_cost(min_idx, 1) + norm(parent_node - child_node);
5.             h = abs(child_node(1) - target_node(1)) + abs(child_node(2) -target_node(2));
6.             f = g+h;
```

Then, find out the child nodes of the parent_node ignoring the closeList, this part can directly call the previously explained subroutine. And use the “for” loop from 1 to the total number of child_nodes to determine whether these child nodes are in the openList, if they are, they will be updated; if they are not, they will be appended to the openList. Details are as follows:

Using the “ismember” function indicates that the child node is in the open list and returns 1, judges the flag, and outputs the position of the child node in the openlist table. Then refresh the cost value of g(n):” g = openList_cost(min_idx, 1) + norm(parent_node - child_node)” according to the new parent node. Also refresh the cost value of h(n):” h = abs(child_node(1) - target_node(1)) + abs(child_node(2) - target_node(2))” also according to the new parent node. And refresh the f(n) value: f(n)=g(n)+h(n). Use the “if ” command to judge whether those child_nodes are included in the openList, If so, compare and update g and f, If not, append to openList. Thus, the child_node who has the least moving cost is found and decided already. Remove the node with the least moving cost from openList to closeList so this node can be chosen.

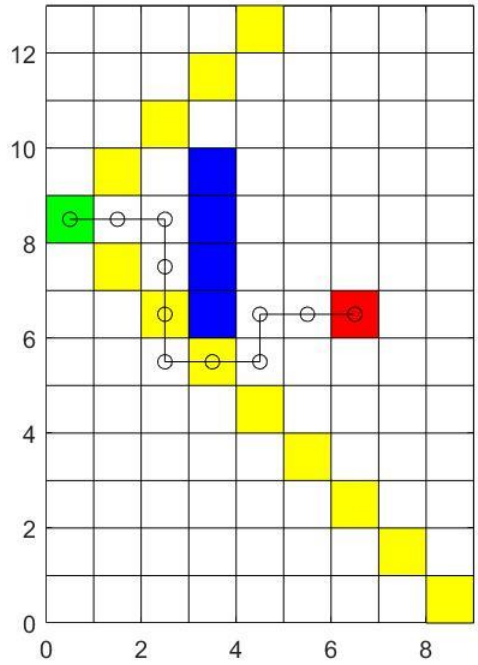

Fig. 31 – Roof panel delivering robot’s visualization

After this step, “openList(min_idx,:) = []” is to set the position of the jumped minimum value in the openlist table to empty. The openList is refreshed. And search again: search for the node with the least moving cost from openList (repeat steps), this repeating process is for the next loop, At this point, this loop ends and the next loop continues. Among those loops, it is also important to decide when to end this loop. And considering this robot is to deliver the roof panel to the roof panel laying robot, the target node shall be set as where the roof panel laying robot run outs of panels. So, use the “if” command

to judge whether, in one specific loop, the parent node equals the target node, when the result is true, end the loop and refresh the closeList and closeList_cost.

```
1. if in_flag
2.                 if g < openList_cost(openList_idx,1)
3.                     openList_cost(openList_idx, 1) = g;
4.                     openList_cost(openList_idx, 3) = f;
5.                     openList_path{openList_idx,2} = [openList_path{min_idx,2}; child_node];
6.                 end
7.             else
8.                 openList(end+1,:) = child_node;
9.                 openList_cost(end+1, :) = [g, h, f];
10.                openList_path{end+1, 1} = child_node;
11.                openList_path{end, 2} = [openList_path{min_idx,2}; child_node];
12.            end
```

To visualize this entire process, some sentences can be added. Define the “closeList_cost” as the movement of this delivers robot, and use the “scatter(x,y,sz,c)” command to draw the scatter point in the previous map, where x represents the x-axis and y represents the y-axis, sz represent the size of scatter and c can control the color of the scatter point.

Finally, use the “plot” command to draw a 2D line graph in the previous map. Here use green cell to represent the stockpile point where every time this carriage robot gets supplied. And red cell to represent the location where the roof-laying robot run outs of panels. Yellow cell represent the human part where robots don’t need to concern, and blue cell represent the obstacles. All are shown in Fig. 31, after the previous setting and programming, the robot avoids the obstacles and finds the shortest path from the stockpile point to another robot according to the A* (A-Star) algorithm.

4.4. PATH PLANNING AND OBSTACLE AVOIDANCE SYSTEM FOR ROOF PANEL LAYING ROBOT

a. Map

Similar to the previous part, a grid map for the working area is built.

But in this part, things are a little bit different, the roof panel laying robot is required to cover the entire map as much as possible, and that cell which is already laid by panels can not step on by the robot again because of its thickness. Here is an explanation picture (Fig. 32) of the roof panel in which the cells marked as magenta is the roof cell part that has already been laid roof panel by the panel-laying robot.

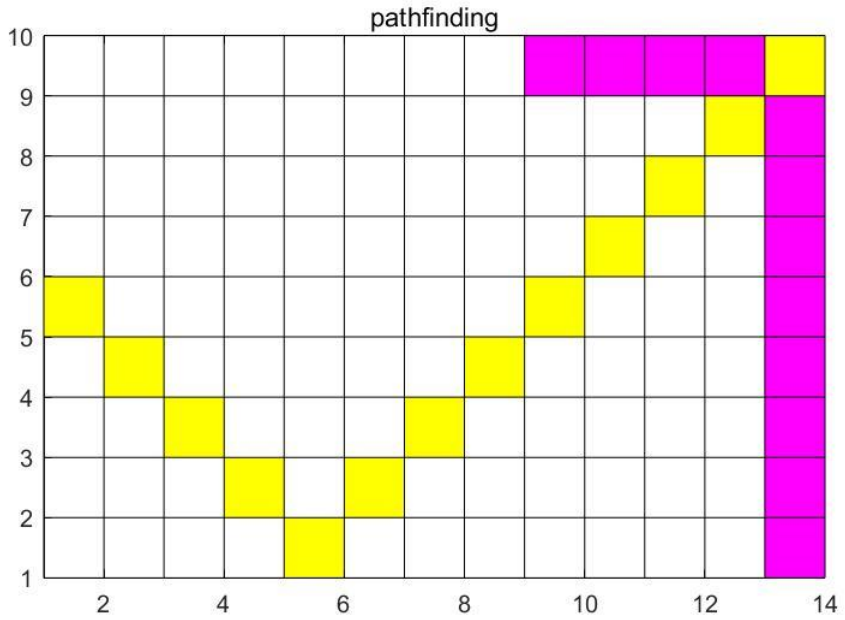

Fig. 32 – Roof panel which already laid

Considering this condition, it is necessary to set different parameters to different conditions in the map. Here 0 is an obstacle, 1 is an empty state, and 2 is the part that has already been laid roof panel by the panel-laying robot. And to restrain the movement of the robot, different from the previous robot, this map has an "Invisible" barrier around the entire map. That is, set the surrounding area as obstacles.

```
1. roomlength=14;
2. roomwidth=10;
3. map=ones(roomwidth+1,roomlength+1);
4. map(1,:)=0;
5. map(:,1)=0;
6. map(end,:)=0;
7. map(:,end)=0;
8. map(end+1,end+1)=0;
```

Here the boundary is set as 0.

And in this specific program, there exist two different patterns, one is to make the roof panel-laying robot move clock wisely and another is to make the robot anti-clock wisely.

Thus, according to the previously extracted BIM information and DIN stander, the optimal sequence for laying the panels is determined. And to follow the optimized laying

sequence, several switch points are needed to be set in the map as shown in Fig. 2.

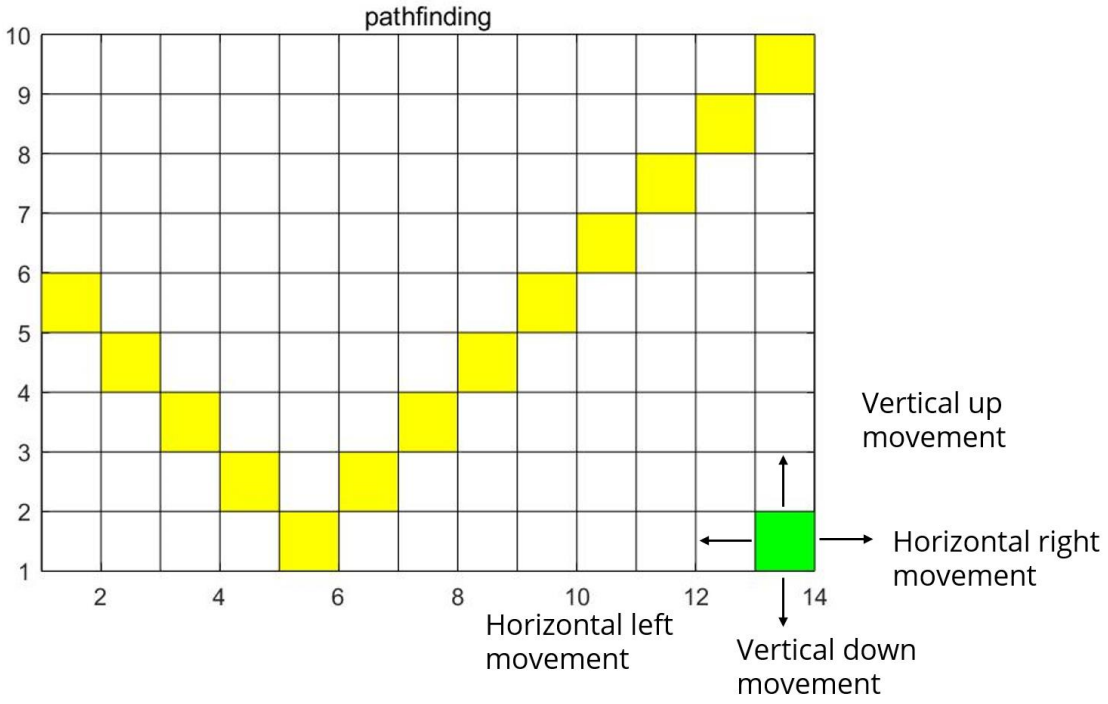

Fig. 33 – 4 robot move direction

```
1. switchpoint = [2,10];
2. switchpoint2 = [13,2];
3. switchpoint3 = [2,8];
4. switchpoint4 = [11,2];
5. switchpoint5 = [2,2];
6. switchpoint6 = [9,2];
7. switchpoint7 = [4,2];
8. switchpoint8 = [4,2];
```

here several switchpoint is set as a single matrix to avoid the error which may occur when putting them all together.

```
1. unwantedpoint1 = [2.5,3.5;4.5,1.5];
2. unwantedpoint2 = [4.5,1.5];
3. unwantedpoint3 = [4.5,1.5];
4. unwantedpoint4=[5.5,2.5;5.5,3.5;7.5,4.5;7.5,5.5;9.5,6.5;9.5,7.5;11.5,8.5;11.5,9.5;
5. unwantedpoint5 = [2.5,3.5;4.5,1.5];
6. unwantedpoint6 = [2.5,3.5;4.5,1.5];
7. unwantedpoint7 = [7.5,2.5;9.5,4.5;11.5,6.5;13.5,8.5];
8. unwantedpoint8 = [2.5,9.5;2.5,7.5;2.5,5.5;4.5,3.5];
```

And based on the reason for optimizing the display, some unwanted points need to be deleted, similarly here several switchpoint is set as a single matrix to avoid the error which may occur when putting them all together.

Then, like the previous program, generate a room grid map and geometric axes such that each axis has evenly spaced ticks and set the obstacle grid value to zero.

Here, the required map is set.

b. Preprocessing

In this part, due to the reason the program is more complicated, more values shall be set and more index shall be defined.

To start, there are four situations in the robot to move in the room, 1. Horizontal right movement. 2. Horizontal left movement. 3. Vertical up movement. 4. Vertical down movement. As shown in Fig. 33.

```
1. finish=1;
2. robmove=1;
3. row_right=1;
4. columns_up=2;
5. row_left=3;
6. columns_down=4;
```

So that movement can be set in the program, and here those numbers only represent the movement status.

“A=[0,0]”, within the entire program, A is a special index that represents currently the map location of the under-laying roof panel. In the beginning, this value is set as zero.

Also importantly, there is a need for an index that can put the loops in the end when certain pre-conditions are fulfilled. So, the flag finish indicates whether the roof laying is completed, and when it is 0, it indicates that the roof laying is completed. And at the beginning, use “finish=1;” to set this value equal to 1.

Represent robot motion with “robmove”, this is a special index that represents the moving condition of the robot. There will be a switch function used in the loop that “robmove” is also changing when the movement switches from one to another.

To decide when is a good time to stop the program, it is reasonable to say that when all the required area is laid by roof panels, this robot can stop functioning.

Thus, If the search range has reached the limit value (full search in all four directions reaches the boundary), and there is still no result, it means that the robot has completed

laying, and it will end. Use the flag bit “isend” to determine whether the check is complete. Here set all four direction’s flag as 0.

```
1. isend1=0;
2. isend2=0;
3. isend3=0;
4. isend4=0;
```

And, to control the switch of “clockwise” and “anti-clockwise” movement of the roof panel laying robot. An index “k” is given the value of 0, “k=0”, which will be used in the program loop.

Also, to plot the graph clearly, use the “title('pathfinding')” command to plot “pathfinding” at the top of the map before the program loop.

c. Program loop

Here is the most important part of the program, where the clockwise and anticlockwise movement and dead zone detection procedures are laid.

To start with, use the combination of “while” and the previously mentioned “finish” to create the main loop, “while(finish)” can represent the start of the main loop, and when some conditions are fulfilled, the entire loop can be finished.

Then, the switch which controls the robot’s movement from clockwise to anticlockwise or from anticlockwise to clockwise is added as: “if mod(K,2)==1”, as mentioned before, “k” is an index that controls the robot movement. Define when k is an even number, run the program clockwise, and when k is odd, run the program counterclockwise.

Here, the “mod” command is used to get the remainder after division (modulo operation). So the previous “mod(K,2)” represent k divided by 2 and outputs the remainder. “if” is the condition sentence, as mentioned before, when the remainder is even (here is 1), run the clockwise program and when the remainder is not even “else”, run the anticlockwise program. “K”’s value is given by those predefined “switchpoint”.

```
1. hold on
2. while(finish)
3.       if mod(K,2)==1
4.             switch(robmove)
```

About the clockwise part, there exist four states when the robot moves. “switch(robmove)” can create a loop that evaluates an expression and selects one of several sets of statements to execute. Each option is a case. The switch block tests each case until a case expression is true. Thus to put four directions movement is possible.

```
case row_right
                if(map(m+1,n)==1)
                    robmove=columns_up;
                elseif(map(m,n+1)==0||map(m,n+1)==2)
                    i=i+1;
                    x(i)=n;
                    y(i)=m;
                    map(m,n)=2;
                    robmove=columns_down;
                else
                    i=i+1;
                    x(i)=n;
                    y(i)=m;
                    map(m,n)=2;
                    n=n+1;
                end
```

Firstly, define the horizontal right movement, “case row_right” means when the case expression is true, Matlab executes the corresponding statement and then exits the switch block, and the following is the detail of the “case row_right” expression.

Considering the robot is moving the clock wisely, there exists the Influence of gravity fields. When the robot is moving toward the right, first thing is to check and determine whether there is a space on the left side of the robot (which actually is upside from the overall coordinate point of view), if so, the robot turns to its own left. By doing this, when the robot is moving toward the left, there exists only one condition that is: the upper part of the robot is an obstacle.

So, the upper gravity field’s influence can be accomplished by “map” which is the map coordinate representation of the upper cell and using the “if” command to decide whether this upper cell is an empty state or not, as mentioned before, 1 means empty state. “robmove=columns_up” can change the movement of the robot and when this upper cell is empty, make robot moves up.

To make the map coordinate representation clearer, it is shown in Fig. 34.

Then, to check and determine whether there is an obstacle or cell that is already laid by the roof panel on the front direction of the robot (which actually is the right side from the overall coordinate point of view), if so, the robot turns to its own right. By doing this, when the robot is moving toward the right, there exists a condition that is: the front part of the robot is an obstacle or roof panel, and the upper part is not empty.

This is accomplished by the sentence “elseif”. That means this part can be initiated when the upper part cell is not empty, and the right part cell is also not empty. So move the robot to the lower cell.

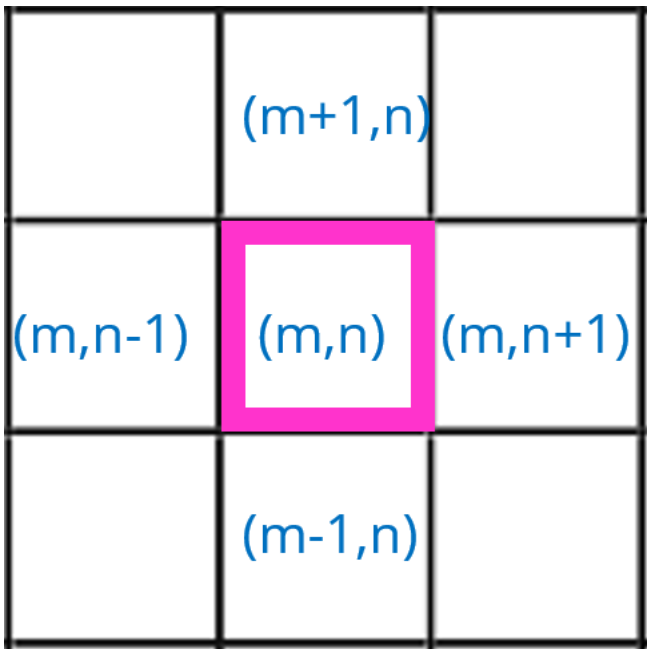

Fig. 34 – Map coordinate representation

“i=i+1; x(i)=n; y(i)=m;” can be explained as follow: firstly the “i” index is the current parameters for coordinate counting which are used in x(i) and y(i). And set the map value to 2 means this cell is already laid by the roof panel. When the precondition is fulfilled, make the robot moves down. And here is the lase precondition that not only the upper cell is occupied but also the right cell is empty, move the robot directly to the robot front (which is the right side from the overall coordinate point of view). But to visualize the actual movement of the roof panel laying robot, there are a few things that need to be clarified here. First, between the robot itself and the roof panel exist a cell’s distance. That is to say, the actual coordinate location of the robot is always one cell away from the currently laying roof panel. Shown in Fig. 35.

That means to get the moving coordinate of the robot, several extra steps are required. Here the robot is moving toward the right, and the movement is clockwise. So, it is reasonable to say that the robot is one cell below the currently laying roof panel.

```
1. obs = [obs;x(i),y(i)];
2. E=[x(i)-0.5,y(i)-1.5];
3. F=ismember(E,unwantedpoint1,'rows');
```

“E=[x(i)-0.5,y(i)-1.5]; F=ismember(E,unwantedpoint1,'rows'); C=true; “, using “E” to represent the coordinate location of robot and “x(i), y(i)” represent the coordinate location

of currently laying roof panel. And “ismember” can determine whether an array element is a set array member, as predefined before, some unwanted points were set and compared here to the coordinate location of the robot, if the result is true, the program will delete those points automatically.

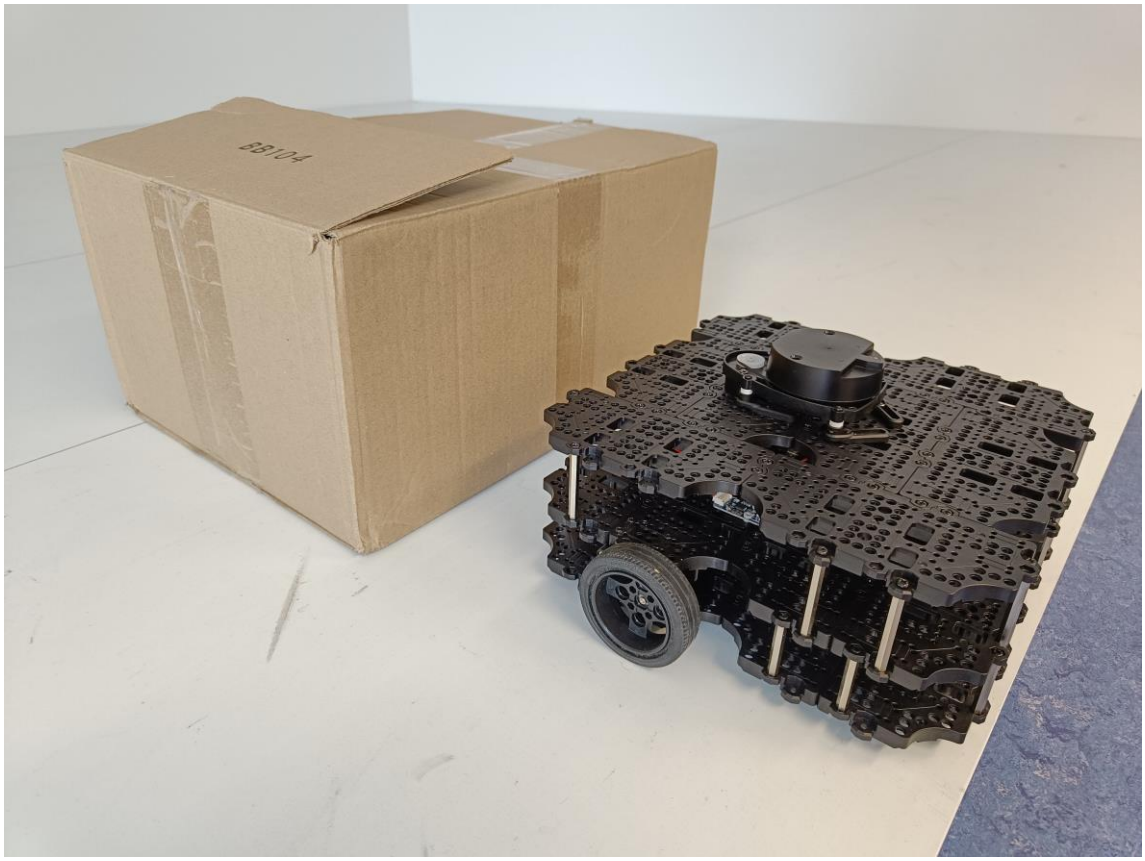

Fig. 35 – Brief explanation using turtle bot and box

```
1. if (F~=C)
2.             plot(x(i)-0.5,y(i)-1.5,'ro');
3.             G=[x(i)-0.5;y(i)-1.5]
4.             hold on
5.             fid=fopen('1.txt','a');
6.             fprintf(fid,'%d ',E)
7.             fprintf(fid,'\r\n');
8.             fclose(fid);
9.             pause(0.05)
10. end
```

Using the “if ” command to judge whether the coordinate location of the robot belongs to pre-set unwanted points, and plot the scatter into the map with the color red.

“pause(0.05)” means after each step, the program will pause 0.05 second to give the viewer enough time to react the movement of points.

And the other three directions: left, up, and down’s movement can be similarly defined.

```
case columns_up
                if(map(m,n-1)==1)
                    robmove=row_left;
                elseif(map(m+1,n)==0||map(m+1,n)==2)
                    i=i+1;
                    x(i)=n;
                    y(i)=m;
                    map(m,n)=2;
                    robmove=row_right;
                else
                    i=i+1;
                    x(i)=n;
                    y(i)=m;
                    map(m,n)=2;
                    m=m+1;
                end
```

Then, to check and determine whether there is an obstacle or cell that is already laid by the roof panel on the front direction of the robot (which actually is the lower side from the overall coordinate point of view), if so, the robot turns to its own right. By doing this, when the robot is moving toward the down, there exists a condition that is: the left part of the robot is an obstacle or roof panel, and the front part is not empty.

This is accomplished by “elseif(map(m-1,n)==0||map(m-1,n)==2)” which is controlled by the sentence “elseif”. That means this part can be initiated when the lower part cell is not empty, and the lower part cell is also not empty. So move the robot to the left cell.

“i=i+1; x(i)=n; y(i)=m; map(m,n)=2; robmove= row_left;” can be explained as follow: firstly the “i” index is the current parameters for coordinate counting which are used in x(i) and y(i). And set the map value to 2 means this cell is already laid by the roof panel. When the precondition is fulfilled, make the robot moves left.

And here is the lase precondition that not only the right cell is occupied but also the lower cell is empty, move the robot directly to the robot front (which is the lower side from the overall coordinate point of view).

“i=i+1; x(i)=n; y(i)=m; map(m,n)=2; m=m-1;” can be explained as follow: setting the map value to 2 means this cell is already laid by the roof panel. When the precondition is fulfilled, make the robot moves toward the down.

This is the logic behind the robot's clockwise movement, and the anticlockwise part can explain similarly.

If there are no grids to be cleaned, upper, lower, left, and right, it means that the robot is stuck in a dead zone or has finished roof panel laying. So the following explanation is designed to end the program loop when necessary.

```
1. while(map(m+1,n)~=1&&map(m-1,n)~=1&&map(m,n+1)~=1&&map(m,n-1)~=1&&map(m,n)==2)
```

"while" is a pre-conditions that can control the program end, and the explanation is to find whether the upper left down and right direction have already been laid by the roof panel or obstacle.

```
1. if m-h>=1
 2.                 r1=m-h;
 3.             else
 4.                 r1=1;
 5.                 isend1=1;
 6.             end
 7.             if m+h<=roomwidth
 8.                 r2=m+h;
 9.             else
10.                 r2=roomwidth;
11.                 isend2=1;
12.             end
13.             if n-h>=1
14.                 c1=n-h;
15.             else
16.                 c1=1;
17.                 isend3=1;
18.             end
19.             if n+h<=roomlength
20.                 c2=n+h;
21.             else
22.                 c2=roomlength;
23.                 isend4=1;
24.             end
```

And check if there is still 1 in the array, that is, the grid to be laid by the roof panel, if not, it means that the installation is completed, then jump out of the main loop. In detail, spread out layer by layer with the current grid as the center, and find the grid position with a grid value of 1. Use the variable h to represent the number of diffusion layers, and its initial value is set to 1. Determine the inspection range of each layer (a box composed of rows and columns), and use r1, r2, c1, and c2 to represent the range of rows and columns.

If the installation work is not completed, that is, the robot falls into the dead zone, and the operation of jumping out of the dead zone is performed. First, find the grid to be laid by the roof panel closest to the current grid, and then plan the shortest path. This is what the following “inspection” means.

“if” is to calculate along the room width direction, what is the range of the inspection. Another “if” is also to calculate the range of the inspection but in different conditions. Similarly, the third “if” is to calculate along the room length direction, what is the range of the inspection. And last “if” is also to calculate the range of the inspection but in different conditions.

After defining the range of inspection, to actually inspect whether the robot is stuck in a dead zone or has finished roof panel laying. Sequentially check the raster values in the range: Check the upper and lower lines first, and check from the middle to both sides.

```
1. for r=r1:(r2-r1):r2
2.                 if finish
3.                     for c=n:c2
4.                         if map(r,c)==1
5.                             x1=c;
6.                             y1=r;
7.                             finish=0;
8.                             break;
9.                         end
10.                    end
11.                end
12.                if finish
13.                    for c=n:-1:c1
14.                        if map(r,c)==1
15.                            x1=c;
16.                            y1=r;
17.                            finish=0
18.                            break;
19.                        end
20.                    end
21.                end
22.            end
```

This is accomplished by: “ for “ and “break ” here to terminate the execution of a for or while loop. Similarly, check the left and right columns, and check from the middle to both sides. If not checked, expand one layer to continue checking. Use the variable h to represent the number of diffusion layers, and its initial value is set to 1, and here If not checked, expand one layer to continue checking. Which can be using “h=h+1” to accomplish.

If the search range has reached the limit value (full search in all four directions reaches the boundary), and there is still no result, it means that the robot has completed cleaning, and it will end. Use the flag bit isend to determine whether the check is complete.

```
1. if isend1==1&&isend2==1&&isend3==1&&isend4==1
2.                 finish=0;
3.                 break;
4. end
```

“ if isend1==1&&isend2==1&&isend3==1&&isend4==1; finish=0; break; end”. Is to finish the entire loop. And it will redefine the value of “isend”.

To accomplish the final goal, Combining the above two programs enables the roof panel laying robot to replenish the material when the loading material is insufficient by the carriage robot.

Set “panelnumber = 10” at the beginning which defines the number that the roof panel laying robot can carry at one time. And this value can be user-defined at any time. Also, define “p=0;” at the beginning, this is the counting index that counts the number of roof panels which already been constructed, before the program loop, set as 0.

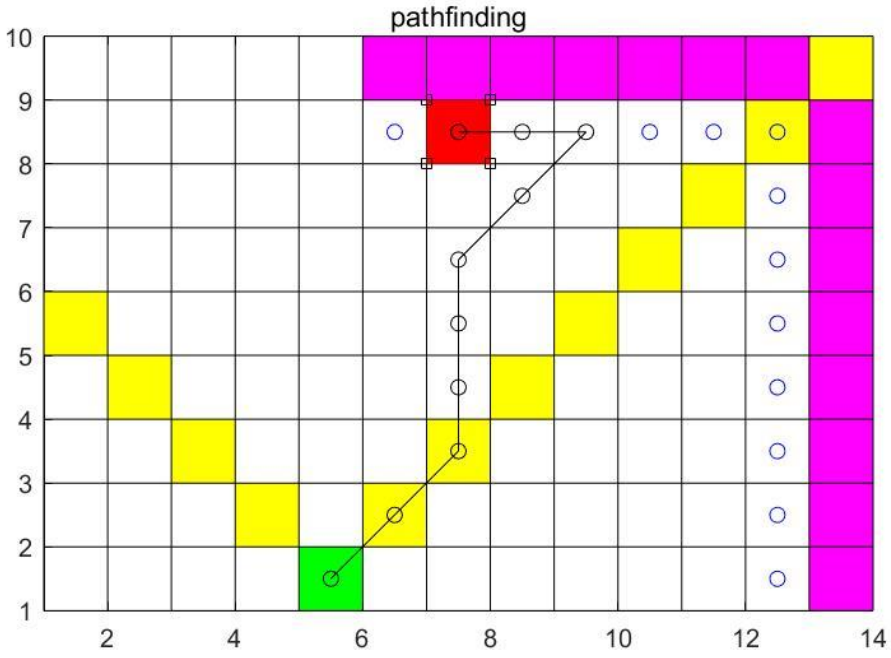

Fig. 36 – Carriage robot’s orientation problem

And import the previous A-star program as a subroutine into the existing program. This is because it is important to consider the two robots as one group, they move and cooperate together in reality, so the program is also the same. Set the stockpile point as the start point for the carriage robot and the location where the roof panel laying robot runs out of the roof panel as the target point. “obs = [obs;x(i),y(i)];” can set those laid roof panels as obstacles and this info will be further introduced into the A-star subroutine. And previous is the information that A-star will need when it runs as a subroutine.

But in this project, it is also important to consider that there exists an orientation problem when the carriage robot delivers the roof panel, there exists an orientation of the feed port at the carriage robot and roof panel laying robot.

Thus, by turning the carriage robot at an angle before it meets the roof panel, this orientation problem can be solved. So it is possible to call the A-star subroutine three times to make the carriage robot turn its direction.

```
1. if (p==panelnumber)
2.                          A=[x(i)+3,y(i)-1];
3.                          parent_node=[6,2];
4.                          target_node=A;
5.                          start_node=parent_node;
6.                          fill([start_node(1)-1, start_node(1), start_node(1), start_node(1)-1],...
7.                          [start_node(2)-1, start_node(2)-1 , start_node(2), start_node(2)], 'g');
8.                          n=roomlength;
9.                          m=roomwidth;
10.                         A_star_cal(start_node,parent_node, target_node,m,n,obs);
11.                         A=[x(i)+1,y(i)-1];
12.                         parent_node=[x(i)+3,y(i)-1];
13.                         target_node=A;
14.                         start_node=parent_node;
15.                         fill([target_node(1)-1, target_node(1), target_node(1), target_node(1)-1],...
16.                         [target_node(2)-1, target_node(2)-1 , target_node(2), target_node(2)], 'rs');
17.                         A_star_cal(start_node,parent_node, target_node,m,n,obs);
18.                         finish=0;
19. end
```

And the result is shown in Fig. 36, magenta part represents the area where already been laid by the roof panel, the yellow part means the roof panel Is in regular that will be planned by humans, and the blue scatter is the path of the roof panel laying robot, the black scatter is the path of the carriage robot. And as mentioned before, green cells represent the stockpile point and also the start point for the carriage robot, while the red cell is one cell behind the current location of the laying robot for material delivery.

By doing this, the orientation problem is solved.

The program explanation is: “A=x(i)-2,y(i)-3]”, this defines the first turning point, “x(i),y(i)” is the current carriage robot’s location, and the number behind is defined by the current location. For example in Fig. 36, the current carriage robot is located in (7,9), so the first turning point become (8,7).

Then, set “parent_node=[6,2]” which is the location of the stockpile point. “target_node=A; start_node=parent_node;” to give the target point and start node for the previous A-star subroutine.

“A=[x(i)+4,y(i)-1]” defines the second turning point, this point also varies according to location (here use Fig. 36 as an example). Then, set “parent_node=[x(i)-2,y(i)-3]” which is the location of the last turning point. “target_node=A; start_node=parent_node;” to give the target point and start node for the previous A-star subroutine. “A=[x(i)-1,y(i)-1]” defines the third turning point, this point also varies according to location. Then, set “parent_node= [x(i)+4,y(i)-1]” which is the location of the last turning point. “target_node=A; start_node=parent_node;” to give the target point and start node for the previous A-star subroutine.

At this moment, the carriage reaches the cell next to the roof panel laying robot, the feed port at the carriage robot. By doing the previous step, the orientation problem is solved.

Finally, to visualize the movement of both robots, “ A=[x(i),y(i)];” first define every cell that is under laying currently, “B=ismember(A,humanpart, 'rows');” is judging whether the cell belongs to the human-part. “C=true; if (B~=C)” then “p=p+1; fill([x(i)-1, x(i), x(i), x(i)-1]; [y(i)-1, y(i)-1, y(i), y(i)], 'm'); pause(0.05)” not just only plot the roof panel but also define the color “m” that is magenta. Also, “p=p+1” is the previous counting index that serves for counting the roof layer which already is constructed.

Here, all the program detail in the roof panel laying robot and the combination of both robot is explained.

5 IMPLANTATION

Here is the implantation part of the previous program.

Firstly, a Fortran program was designed to extract the txt file generated by the final calculation of the Matlab, and automatically write the coordinate statement required by the subsequent python control program according to the coordinate points in the txt file.

The following is a detailed explanation:

```
 1. open (10,file='data.txt')
 2. read(10,*) n
 3.             do i=1,n
 4.                             read(10,*) x(i),y(i)
 5.             enddo
 6. close(10)
 7.
 8.
 9. do i=1,n
10.             open(11,file='output.txt')
11.             write(11,*)    x(i)*3,y(i)*3
12.             !write(11,*) 'waypoints.append(Pose(Point(' x(i),y(i) ',0.0),quaternions[0]))'
13. enddo
```

In Matlab, “E=[x(i),y(i)]” is to give the current coordinate of the roof layer laying robot into index E. Then, “fid=fopen('input.txt','a'); fprintf(fid,'%d ',E);” whenever the point changed, written it into a txt file, eventually this txt file becomes the input of the Fortran program.

In Fortran, first, use the” open” command to open the previous input.txt file. And row by row, read the coordinate information until everything was input into Fortran. Then, put this part into a “do” loop so that the previous reading can be accomplished. As for the writing part, similarly, use the ” loop to write. “ open” command to open an output file. And use the “ write” command to put the Turtlebot control Python program required sentence. “

“write(11,*) 'waypoints.append(Pose(Point(', x(i)/1.48,',',y(i)/1.48,',0.0),quaternions[0]))'” is the standard output format. It includes x,y,z, coordinate, and steering control commands.

To check whether the generated point fulfills the requirement, here in the Fortran program set a visualization part: the Gnuplot subroutine was used as before. Set the x

range and the y range to create the coordinate system, all the points were put into Gnuplot to check the result as shown in Fig. 37.

But it is worth mentioning here, that in the actual control interface of ROS, it is necessary to remove redundant points and add some extra points so that the Turtlebot won't move out of control. Here is a picture representation of the adjusted points of the Turtlebot in Fig. 38.

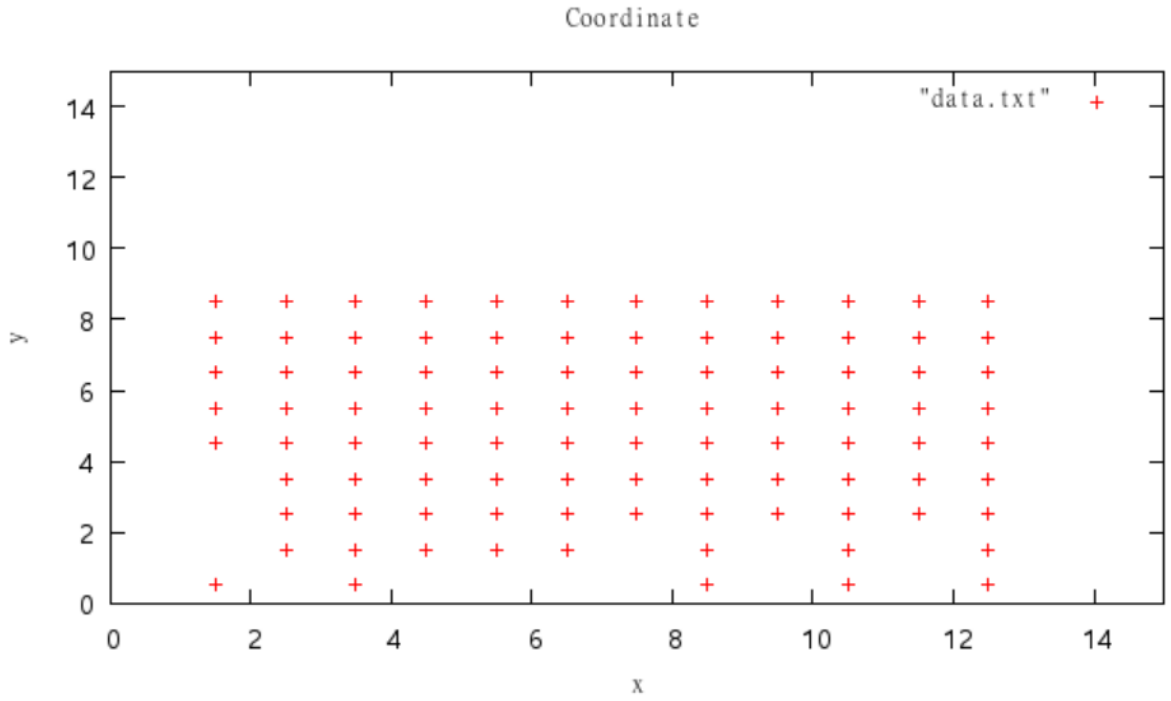

Fig. 12 – Coordinate of extracted points

Those points will be directly given as input data for ROS to guide the movement of the Turtlebot. And the action movement track shall be shown in Fig. 39. After all previous presetting, a txt file will be generated by Fortran which will be further used in the Turtlebot control Python program.

The following is the detailed process of ROS control and implantation.

In ROS, there exists the need to inputting the map and generate the map environment for further usage. Given are the IFC file which concludes an abundance of building information, but only the geometry and interaction between geometry is needed.

After extracting and putting the required information into Gazebo, a map environment was created. To be further used in the Turtlebot, the robot's initial coordinate shall be manually set into this environment.

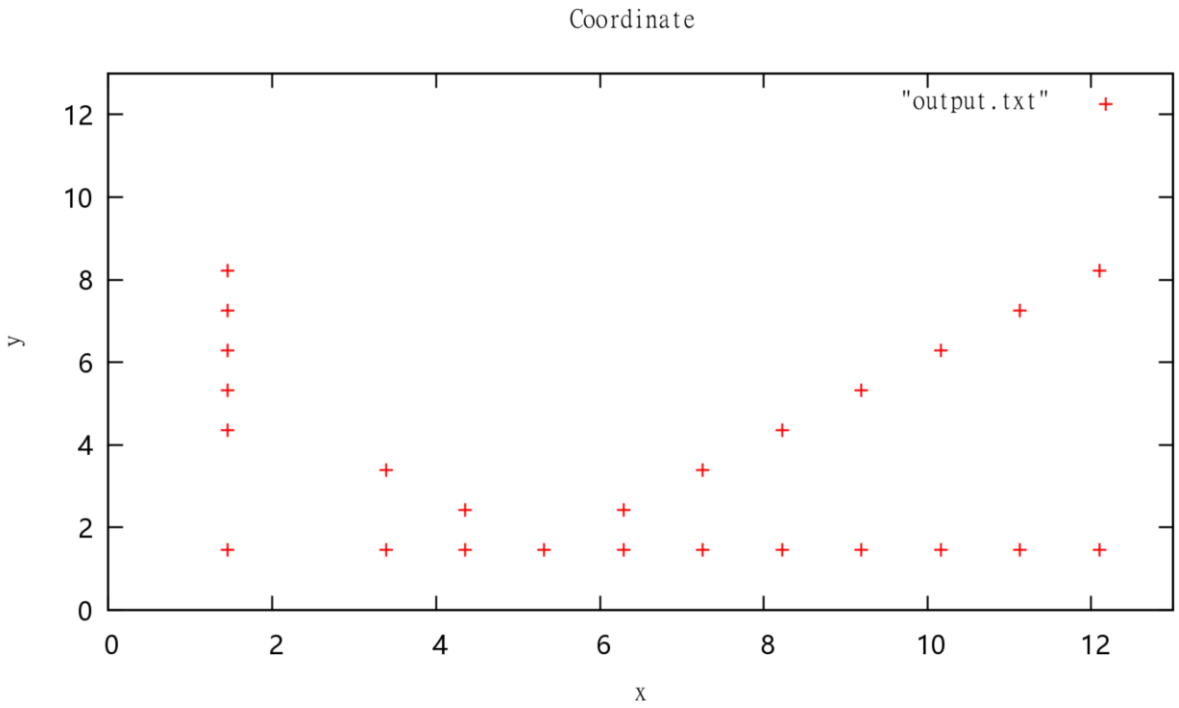

Fig. 38 – Adjusted points

But it is worth mentioning here, the Turtlebot needs a subroutine "Rviz" to merge the simulated sensor map and the real room map. It's important to note that this process is typically used in robotics to improve the accuracy and precision of the mapping process, and it may require some human intervention and manual control to achieve the desired results, which is, the Turtlebot was manually controlled to go through the room to make a fit as shown in Fig. 40.

By using "Rviz," the two maps can be overlaid on top of each other to create a more accurate and complete representation of the environment in which the Turtlebot is operating. This process can help the Turtlebot to better navigate and interact with its surroundings.

Then, input the initial point and coordinate system for the Turtlebot so that the robot's movement can have some reference and the guiding and mapping part for the robot's movement is now complete.

Here is a room map of Bauinfomatic institute in the simulated environment in Fig. 41.

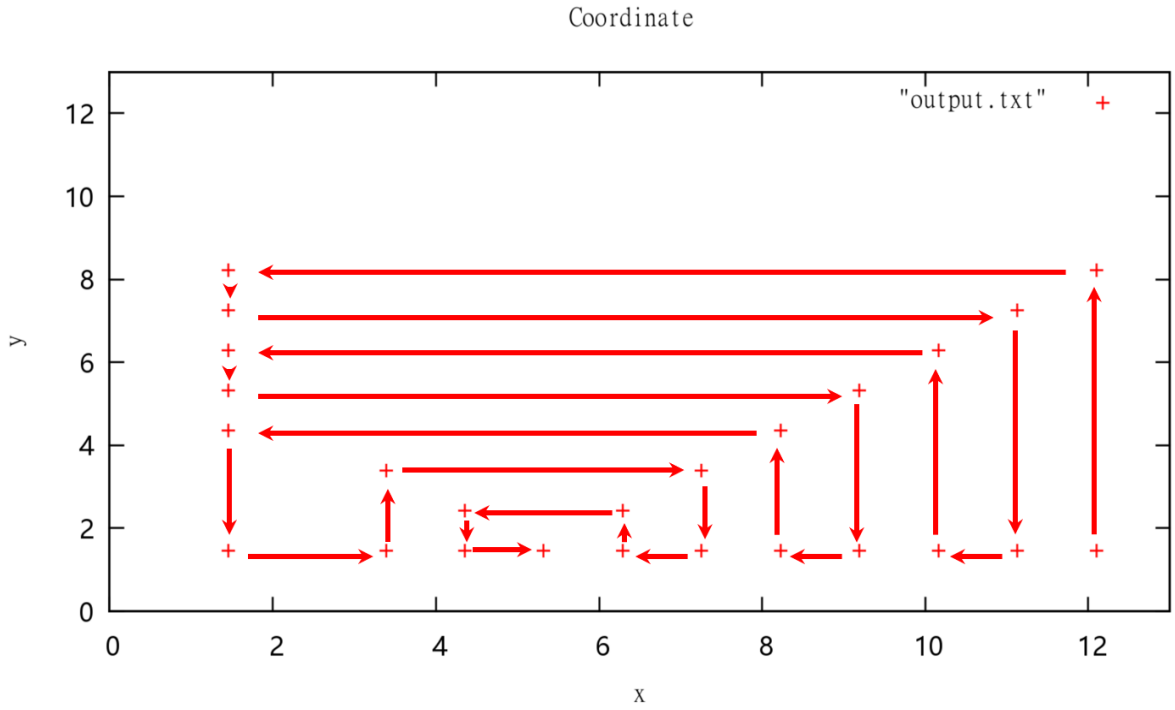

Fig. 39 – Action track

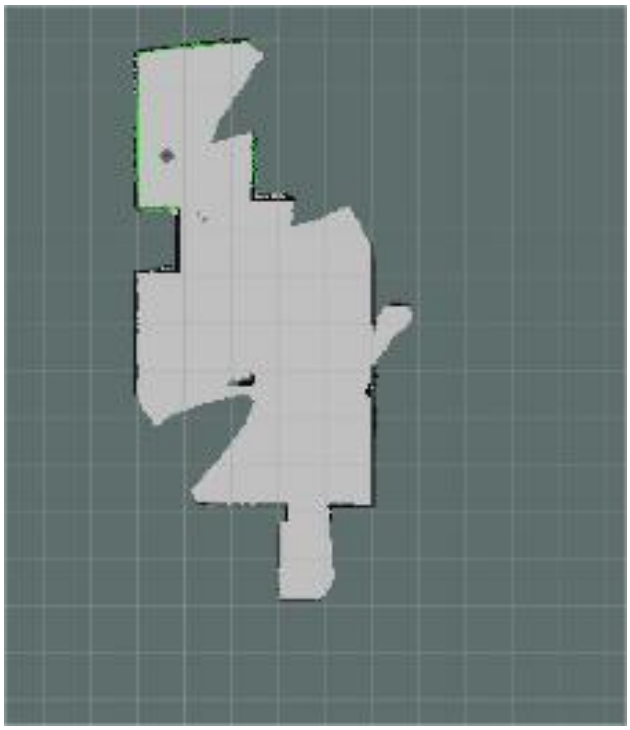

Fig. 40 – "Rviz" process

As shown in Fig. 41, some obstacles were added to the room to not only accomplish automatic obstacle avoidance but also to give the Turtlebot some room features that it can locate on the map.

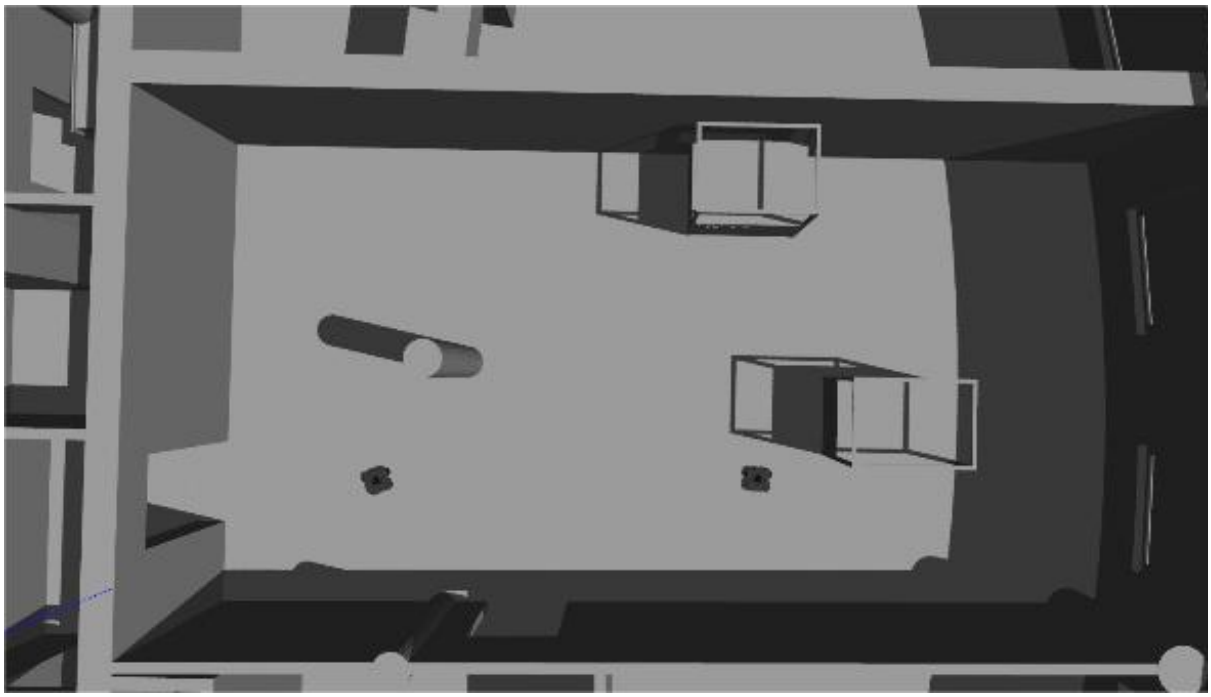

Fig. 41 – Room map in Gazebo enviroment

First, the input point coordinates are generated by Matlab. These inputs are then fed into the Python control program for the Turtlebot.

Once the inputs have been entered, the Turtlebot begins to move toward the target endpoint, following the calculated points step by step. This movement is accomplished using the A-star program, which allows the bot to detect and avoid obstacles in its path. By using this program, the Turtlebot can autonomously navigate through the environment while ensuring that it avoids any obstacles or hazards that may be present.

During the simulation, the Turtlebot will continue to move toward the endpoint until it reaches its destination. Throughout this process, the A-star program will continue to analyze the environment and make adjustments to the bot's path as needed, to ensure that it remains on course and avoids any obstacles that may arise. By using this combination of map-based inputs, Python control software, and the A-star program, the Turtlebot can move through the environment autonomously and with a high degree of accuracy and safety.

Here is the result visitation in “Rviz” shown in Fig. 42. It shows the Turtlebot moving towards one of the set points.

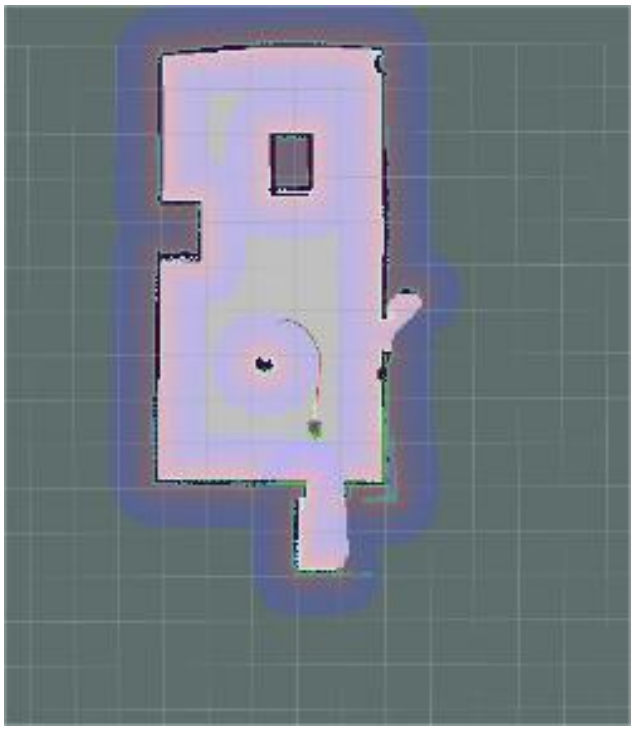

Fig. 41 – Room map in Gazebo enviroment

But there exist some control issues when Turtlebot is running, for example, the Turtlebot doesn’t go directly to the given point, instead, it turns the wheels and tries to find a better path to the given point.

This behavior of the Turtlebot is intentional and is a result of its navigation algorithm. The navigation algorithm used by the Turtlebot is called the DWA (Dynamic-Window Approach) algorithm, which is designed to handle complex and dynamic environments. The DWA algorithm works by generating a set of feasible trajectories based on the current state of the Turtlebot and the desired goal location. The algorithm then evaluates each trajectory based on factors such as safety, speed, and distance to the goal, and selects the best trajectory to follow(Kaiyu Zheng).

As a result of this algorithm, the Turtlebot may not always move directly toward the given point, especially if there are obstacles in its path or if the environment is complex. Instead, it may choose to turn its wheels and take a detour to find a safer or more efficient path toward the goal.

While this behavior may appear like a control issue to some users, it is a feature of the Turtlebot's navigation algorithm and is intended to make the robot more adaptable and responsive to its environment.

6 SUMMARY

The project work is a challenging but essential task that aims to build a path-finding mechanism for construction robots on roof sites. The objective of the mechanism is to ensure the safety of workers and equipment while reducing energy consumption during construction tasks. By developing an effective path-finding mechanism, construction sites can improve the efficiency and effectiveness of their construction processes, reduce accidents and injuries, and minimize their environmental impact.

The mechanism will need to take into account various factors, such as the layout of the construction site, and the location of obstacles.

In this project work, an Autocad file that includes BIM information was provided, and a Fortran program was written specifically to extract the middle point from the Autocad file, and further input into the Matlab program to create the roof map in the Matlab.

In Matlab, this project work applies two robots in the roof construction field to perform tasks such as lifting and placing roofing materials and installing insulation. One robot's mission is to deliver roof panels, which refers to lifting panels, the process of determining the optimal path for a robot to navigate from one stockpile point to another robot while avoiding obstacles and adhering to construction constraints and rules. And another robot's mission is to plan the panels according to the DIN stander, in the actual construction field, each roof condition has a different roof brick laying sequence, so the robot is expected to plan the roof along the set path and achieve some level's automatic.

Also, considering one robot has a carry limit of roof panels, this second robot Is expected to send signals of missing panels to another robot so that the supply can be delivered by another robot. And in order to accomplish this goal, this project uses Matlab to visualize the entire process which includes the roof laying process and the roof panel delivering process when the set carry limit is reached. In the Matlab part, two robots' behaviors were programmed and combined eventually, realized automatic obstacle avoidance and automatic pathfinding under a laying plan sequence.

Then a Robot "Turtlebot" is controlled by the ROS system to actually simulate this roof planning process. The coordinate information was processed by a Fortran program, and output a txt file which can be used in the Turtlebot controlling Python program. By inputting the Bauinfomatic institute's map into the ROS environment, this simulation process can be performed on this map.

Eventually, the Turtlebot can accomplish the simulation of the movement of the previous construction robot. By moving along the Matlab-generated path.

In conclusion, this project work has accomplished the goal of building the path-finding mechanism for construction robots on the roof site, and it heavily relies on BIM-provided information. Also, the A-star and complete coverage path planning method was used.

And, the Turtlebot along with the ROS system provides a suitable platform to visualize the movement of construction robots. Hopefully, a safety and energy consumption-reduced system in construction tasks are achieved.

But there exist some issues in the combination of the two Matlab programs, due to the difference between amount two grid maps, some extra loop must be written to combine them. And the point output from Matlab is not so perfect, also there exist some problems in the ROS when trying to make Turtlebot follow the exact input point.

Those problems require further work in this research, which hopefully can be solved in the future.

7 REFERENCE

Acharya, Debaditya; Ramezani, Milad; Khoshelham, Kourosh; Winter, Stephan (2019): BIM-Tracker: A model-based visual tracking approach for indoor localisation using a 3D building model. In *ISPRS Journal of Photogrammetry and Remote Sensing* 150, pp. 157–171. DOI: 10.1016/j.isprsjprs.2019.02.014.

Adekunle, Peter; Aigbavboa, Clinton; Akinradewo, Opeoluwa; Oke, Ayodeji; Aghimien, Douglas (2022): Construction Information Management: Benefits to the Construction Industry. In *Sustainability* 14 (18), p. 11366. DOI: 10.3390/su141811366.

Alhussein, Hasnaa; Shehab, Lynn; Hamzeh, Farook (2022): Improvisation in Construction Planning: An Agent-Based Simulation Approach. In *Buildings* 12 (10), p. 1608. DOI: 10.3390/buildings12101608.

Al-Kaff, Abdulla; Moreno, Francisco Miguel; Hussein, Ahmed (2019): ROS-Based Approach for Unmanned Vehicles in Civil Applications. In Anis Koubaa (Ed.): Robot Operating System (ROS), vol. 778. Cham: Springer International Publishing (Studies in Computational Intelligence), pp. 155–183.

Automate: UNIMATE // The First Industrial Robot.

Azhar; S: Building Information Modeling (BIM): Trends, Benefits, Risks, and Challenges for the AEC Industry.

Bernard, Alain; Rivest, Louis; Dutta, Debasish (Eds.) (2013): Product Lifecycle Management for Society. Berlin, Heidelberg: Springer Berlin Heidelberg (IFIP Advances in Information and Communication Technology).

Bernhard Dieber; Severin Kacianka; Stefan Rass; Peter Schartner (2016): Application-level Security for ROS-based Applications. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems : October 9-14, 2016, Daejeon Convention Center, Daejeon, Korea. Piscataway, NJ: IEEE. Available online at http://ieeexplore.ieee.org/servlet/opac?punumber=7743711.

Blanes, F.; Benet, G.; Pérez, P.; Simó, J. E. (2000): Map Building in an Autonomous Robot using Infrared Sensors. In *IFAC Proceedings Volumes* 33 (25), pp. 263–268. DOI: 10.1016/S1474-6670(17)39349-7.

Bloss, Richard (2016): Collaborative robots are rapidly providing major improvements in productivity, safety, programing ease, portability and cost while addressing many new applications. In *IR* 43 (5), pp. 463–468. DOI: 10.1108/IR-05-2016-0148.

Boadu, Elijah Frimpong; Wang, Cynthia Changxin; Sunindijo, Riza Yosia (2020): Characteristics of the Construction Industry in Developing Countries and Its Implications for Health and Safety: An Exploratory Study in Ghana. In *International journal of environmental research and public health* 17 (11). DOI: 10.3390/ijerph17114110.

Botea, Adi; Bouzy, Bruno; Buro, Michael; Bauckhage, Christian; Nau, Dana (Eds.) (2013): Pathfinding in Games. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany. With assistance of Michael Wagner. 11 pages.

Cantone, Domenico; Faro, Simone (2014): Fast shortest-paths algorithms in the presence of few destinations of negative-weight arcs. In *Journal of Discrete Algorithms* 24, pp. 12–25. DOI: 10.1016/j.jda.2013.03.005.

Castañeda, Karen; Sánchez, Omar; Herrera, Rodrigo F.; Pellicer, Eugenio; Porras, Hernán (2021): BIM-based traffic analysis and simulation at road intersection design. In *Automation in Construction* 131, p. 103911. DOI: 10.1016/j.autcon.2021.103911.

Chitta, Sachin; Marder-Eppstein, Eitan; Meeussen, Wim; Pradeep, Vijay; Rodríguez Tsouroukdissian, Adolfo; Bohren, Jonathan et al. (2017): ros_control: A generic and simple control framework for ROS. In *JOSS* 2 (20), p. 456. DOI: 10.21105/joss.00456.

Choi, Young-Ho; Lee, Tae-Kyeong; Baek, Sang-Hoon; Oh, Se-Young (2009): Online complete coverage path planning for mobile robots based on linked spiral paths using constrained inverse distance transform. In : 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009). St. Louis, MO, USA, 2009/10/10 - 2009/10/15: IEEE, pp. 5788–5793.

Christian Herold: Die neue DIN 18531 »Abdichtung von Dächern« und die neue Flachdachrichtlinie des ZVDH – anerkannte Regeln der Technik?

Church, Richard: Finding shortest paths on real road networks: the case for A*.

Craig W. Reynolds: Flocks, herds and schools: A distributed behavioral model.

D'Angelo, Vincenzo; Folino, Paolo; Lupia, Marco; Gagliardi, Gianfranco; Cario, Gianni; Gaccio, Francesco Cicchello; Casavola, Alessandro (2022): A ROS-Based GNC Architecture for Autonomous Surface Vehicle Based on a New Multimission Management Paradigm. In *Drones* 6 (12), p. 382. DOI: 10.3390/drones6120382.

David Baccarini: The concept of project complexity - a review.

Davila Delgado, Juan Manuel; Oyedele, Lukumon; Ajayi, Anuoluwapo; Akanbi, Lukman; Akinade, Olugbenga; Bilal, Muhammad; Owolabi, Hakeem (2019): Robotics and automated systems in construction: Understanding industry-specific challenges for adoption. In *Journal of Building Engineering* 26, p. 100868. DOI: 10.1016/j.jobe.2019.100868.

Dehnavi, Saeid; Sedaghatbaf, Ali; Salmani, Bahar; Sirjani, Marjan; Kargahi, Mehdi; Khamespanah, Ehsan (2019): Towards an Actor-based Approach to Design Verified ROS-based Robotic Programs using Rebeca. In *Procedia computer science* 155, pp. 59–68. DOI: 10.1016/j.procs.2019.08.012.

DIN: A brief introduction to standards.

Dorothea Wagner and Thomas Willhalm: Speed-Up Techniques for Shortest-Path Computations.

Ebrahim, Anju; Wayal, A. S. (2020): Green BIM for Sustainable Design of Buildings. In Vinit Kumar Gunjan, Sri Niwas Singh, Tran Duc-Tan, Gloria Jeanette Rincon Aponte, Amit Kumar (Eds.): ICRRM 2019 – System Reliability, Quality Control, Safety, Maintenance and Management. Singapore: Springer Singapore, pp. 185–189.

Fbr: Hadrian X.

Filipe Barbosa: improving-construction-productivity.

Foead, Daniel; Ghifari, Alifio; Kusuma, Marchel Budi; Hanafiah, Novita; Gunawan, Eric (2021): A Systematic Literature Review of A* Pathfinding. In *Procedia computer science* 179, pp. 507–514. DOI: 10.1016/j.procs.2021.01.034.

Gran Vía (2000): The state of occupational safety and health in the European Union. Pilot study summary report. Luxembourg, Lanham Md.: Office for Official Publications of the European Communities; Bernan Associates [distributor].

Guillermo Moral: Robots and construction automation.

Hans Josef Pesch: The Cold War and the Maximum Principle of Optimal Control.

Huang, Lei; Zhu, Zihan; Zou, Zhengbo (2023): To imitate or not to imitate: Boosting reinforcement learning-based construction robotic control for long-horizon tasks using virtual demonstrations. In *Automation in Construction* 146, p. 104691. DOI: 10.1016/j.autcon.2022.104691.

Hui-Ping Tserng: An operations planning system for asphalt pavement compaction.

Iswanto: Artificial Potential Field Algorithm Implementation.

J. Moy: OSPF Version.

Jayadeva; Shah, Sameena; Bhaya, Amit; Kothari, Ravi; Chandra, Suresh (2013): Ants find the shortest path: a mathematical proof. In *Swarm Intell* 7 (1), pp. 43–62. DOI: 10.1007/s11721-013-0076-9.

jaywu: Analysis Framework For The Interactuon Between Lean Construction And Building Information Modelling.

Kaiyu Zheng: ROS Navigation Tuning Guide.

Kamel, Ehsan; Kazemian, Ali (2023): BIM-integrated thermal analysis and building energy modeling in 3D-printed residential buildings. In *Energy and Buildings* 279, p. 112670. DOI: 10.1016/j.enbuild.2022.112670.

Krzysztof Apt: Edsger Dijkstra The Man Who Carried Computer Science on His Shoulders.

Lawande, Sharmad Rajnish; Jasmine, Graceline; Anbarasi, Jani; Izhar, Lila Iznita (2022): A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games. In *Applied Sciences* 12 (11), p. 5499. DOI: 10.3390/app12115499.

Le, Anh Vu; Nhan, Nguyen Huu Khanh; Mohan, Rajesh Elara (2020): Evolutionary Algorithm-Based Complete Coverage Path Planning for Tetriamond Tiling Robots. In *Sensors (Basel, Switzerland)* 20 (2). DOI: 10.3390/s20020445.

Li Yan: Application of BIM in the protection of historical buildings based on Riegel historic value theory.

Lingard, Helen (2013): Occupational health and safety in the construction industry. In *Construction Management and Economics* 31 (6), pp. 505–514. DOI: 10.1080/01446193.2013.816435.

Luo, Tongyuan (2020): Safety climate: Current status of the research and future prospects. In *Journal of Safety Science and Resilience* 1 (2), pp. 106–119. DOI: 10.1016/j.jnlssr.2020.09.001.

michael: Barriers to implementing information technology in developing countries.

MIT: Dynamic Programming.

Mora, Isabel: Prevention of Street Harassment Through Constrained Shortest Path Algorithms.

Morgan Quigley: ROS: an open-source Robot Operating System.

Mosler, Jürgen: A smarter way of building with mobile robots.

Mullen, R. J.; Monekosso, D.; Barman, S.; Remagnino, P. (2009): A review of ant algorithms. In *Expert Systems with Applications* 36 (6), pp. 9608–9617. DOI: 10.1016/j.eswa.2009.01.020.

ninakutzbach: The Impact of Robots on Productivity, Employment and Jobs.

Nuzul Azam Haron: Improving Cost and Time Control in Construction BIM A Review.

Occupational Safety and Health Administration: Census of Fatal Occupational Injuries Summary, 2021.

Reetie Multani: Robotics in Construction Industry in 2022 | Use, Benefits & Types.

RICHARD UGGELBERG: Comparative Analysis of Weighted Pathfinding in Realistic Environments.

Rondinel-Oviedo, Daniel R. (2021): Construction and demolition waste management in developing countries: a diagnosis from 265 construction sites in the Lima Metropolitan Area. In *International Journal of Construction Management*, pp. 1–12. DOI: 10.1080/15623599.2021.1874677.

Silvester Dian Handy Permana: Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game.

Tang, Llewellyn; Chen, Chao; Tang, Shu; Wu, Zhuoqian; Trofimova, Polina (2017): Building Information Modeling and Building Performance Optimization. In : Encyclopedia of Sustainable Technologies: Elsevier, pp. 311–320.

Tarik Terzimehic (Ed.) (2011): 2011 XXIII International Symposium on Information, Communication and Automation Technologies. 2011 XXIII International Symposium on

Information, Communication and Automation Technologies (ICAT). Sarajevo, Bosnia and Herzegovina, 2011/10/27 - 2011/10/29: IEEE.

The Business Research Company: Global Construction Market Report 2022 – Market Forecast, Trends And Strategies.

Theresa Marie Driscoll: Complete coverage path planning in an agricultural environment.

Thomas Vorbeck; Nadine Wills: The Current State of BIM on Existing Buildings: The Case of Germany.

Vincent: Dynamic Pathfinding Using Genetic Algorithm.

Wang, Huanwei; Qi, Xuyan; Lou, Shangjie; Jing, Jing; He, Hongqi; Liu, Wei (2021): An Efficient and Robust Improved A* Algorithm for Path Planning. In *Symmetry* 13 (11), p. 2213. DOI: 10.3390/sym13112213.

Wang, Juan; You, Hongyu; Qi, Xin; Yang, Na (2022a): BIM-based structural health monitoring and early warning for heritage timber structures. In *Automation in Construction* 144, p. 104618. DOI: 10.1016/j.autcon.2022.104618.

Wang, Peng; Li, Qian; Zhang, Lei; Xu, Yulong (2022b): Complete Coverage Path Planning of Underwater Wall-climbing Cleaning Robot. In *J. Phys.: Conf. Ser.* 2258 (1), p. 12072. DOI: 10.1088/1742-6596/2258/1/012072.

Wang, Ximing; He, Hongwen; Sun, Fengchun; Zhang, Jieli (2015): Application Study on the Dynamic Programming Algorithm for Energy Management of Plug-in Hybrid Electric Vehicles. In *Energies* 8 (4), pp. 3225–3244. DOI: 10.3390/en8043225.

Wong Chong, Oscar; Zhang, Jiansong; Voyles, Richard M.; Min, Byung-Cheol (2022): BIM-based simulation of construction robotics in the assembly process of wood frames. In *Automation in Construction* 137, p. 104194. DOI: 10.1016/j.autcon.2022.104194.

Xiao, Bo; Chen, Chen; Yin, Xianfei (2022): Recent advancements of robotics in construction. In *Automation in Construction* 144, p. 104591. DOI: 10.1016/j.autcon.2022.104591.

Xiao Cui: A*-based Pathfinding in Modern Computer Games.

Xiong, Ni; Zhou, Xinzhi; Yang, Xiuqing; Xiang, Yong; Ma, Junyong (2021): Mobile Robot Path Planning Based on Time Taboo Ant Colony Optimization in Dynamic Environment. In *Frontiers in neurorobotics* 15, p. 642733. DOI: 10.3389/fnbot.2021.642733.

Yang, Simon X.; Luo, Chaomin (2004): A neural network approach to complete coverage path planning. In *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society* 34 (1), pp. 718–725. DOI: 10.1109/tsmcb.2003.811769.

Yang, Wenlin; Wu, Peng; Zhou, Xiaoqi; Lv, Haoliang; Liu, Xiaokai; Zhang, Gong et al. (2021): Improved Artificial Potential Field and Dynamic Window Method for Amphibious Robot Fish Path Planning. In *Applied Sciences* 11 (5), p. 2114. DOI: 10.3390/app11052114.

Yaralidarani, Muhammad; Shahverdi, Hamidreza (2016): An improved Ant Colony Optimization (ACO) technique for estimation of flow functions (k r and P c) from core-flood experiments. In *Journal of Natural Gas Science and Engineering* 33, pp. 624–633. DOI: 10.1016/j.jngse.2016.05.067.

Zeng, W.; Church, R. L. (2009): Finding shortest paths on real road networks: the case for A*. In *International Journal of Geographical Information Science* 23 (4), pp. 531–543. DOI: 10.1080/13658810801949850.

.