MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE NATIONAL AVIATION UNIVERSITY
FACULTY OF AIR NAVIGATION, ELECTRONICS AND TELECOMMUNICATIONS
AEROSPACE CONTROL SYSTEMS DEPARTMENT

APPROVED FOR DEFFENCE
Head of the Department
_____Yurii MELNYK
“_____” _____ 2024

QUALIFICATION PAPER
FOR THE ACADEMIC DEGREE OF
BACHELOR

Title: «Autonomous navigation system of a wheeled robot»

Performer: student of group CS-404 Yelyzaveta HULBINAS

Supervisor:_____ Yurii MELNYK

Srandarts inspector:_____ Mykola DYVNYCH

Kyiv 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА АЕРОКОСМІЧНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Юрій МЕЛЬНИК
«_____» _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО РІВНЯ
«БАКАЛАВР»

Тема: «Система автономної навігації колісного робота»

Виконавець: студент групи СУ-404_____Єлизавета ГУЛЬБІНАС

Керівник: _____Юрій МЕЛЬНИК

Нормоконтролер:_____Микола ДИВНИЧ

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра ***Аерокосмічних систем управління***

Освітній ступінь: бакалавр

Спеціальність: 151 Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Юрій МЕЛЬНИК

« ____» _____ 2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Гульбінас Єлизавети Сергіївни

1. **Тема роботи (проекту):** «Система автономної навігації колісного робота». Затверджена наказом ректора від «1» квітня 2024 р. No 511/ст.
2. **Термін виконання роботи (проекту):** з 13.05.2024 по 13.06.22024
3. **Вихідні дані до роботи (проекту):** розробка модуля автономної навігації колісного робота з побудовою оптимальних алгоритмів обходу перешкод з використанням карт місцевості
4. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):** РОЗДІЛ І. НАВІГАЦІЙНА СИСТЕМА ДЛЯ МОБІЛЬНИХ РОБОТІВ; РОЗДІЛ 2. ПЛАНУВАННЯ ШЛЯХУ МОБІЛЬНОГО РОБОТА; РОЗДІЛ 3. РЕЗУЛЬТАТИ ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ; ВИСНОВОК; СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА РЕСУРСІВ.
5. **Перелік обов'язкового графічного матеріалу:** Рисунки результатів

моделювання та розрахунків. Матеріали презентації в Power Point.

6. Календарний план-графік

1	Отримання завдання	13.05.2024-16.05.2024	Виконано
2	Формування мети та основних завдань дослідження	17.05.2024-18.05.2024	Виконано
3	Аналіз існуючих методів	19.05.2024-23.05.2024	Виконано
4	Теоретичний розгляд рішення проблеми	24.05.2024-28.05.2024	Виконано
5	Розробка методів планування траєкторії рухомого об'єкта, здатного здійснювати автономний рух у різних середовищах	30.05.2024-2.06.2024	Виконано
6	Оформлення пояснювальної записки	2.06.2024-4.06.2024	Виконано
7	Підготовка презентації роздаткового матеріалу	5.06.2024-11.06.2024	Виконано

7. Дата видачі завдання: «13» травня 2024 р.

Керівник дипломної роботи _____ Юрій МЕЛЬНИК
(підпис керівника)

Завдання прийняв до виконання _____ Єлизавета ГУЛЬБІНАС
(підпис випускника)

NATIONAL AVIATION UNIVERSITY

Faculty of Air Navigation, Electronics and Telecommunications

Aerospace Control Systems Department

Educational level: bachelor

Specialty: 151 "Automation and Computer-integrated Technologies"

APPROVED BY

Head of Department

_____Yurii MELNYK

"____" _____2024 y.

Qualification Paper Assignment for Graduate Student

Hulbinas Yelyzaveta Serhiyivna

1. **The qualification paper title** « Autonomous navigation system of a wheeled robot »

   Approved by the rector order from «01» April 2024 № 511/ст.
2. **The paper to be completed between:** 13.05.2024 and 13.06.2024
3. **Output data to the work (project):** development of a module for autonomous navigation of a wheeled robot with the construction of optimal algorithms for avoiding obstacles using terrain maps.
4. **Contents of the explanatory note (list of questions to be developed):**

   SECTION I. NAVIGATION SYSTEM FOR MOBILE ROBOTS; SECTION 2. PLANNING THE MOBILE ROBOT PATH; SECTION 3. RESULTS OF SOFTWARE IMPLEMENTATION; CONCLUSION; LIST OF REFERENCES AND RESOURCES.

5. **List of required graphic material:** Figures of simulation and calculation results. Presentation materials in Power Point.

6. **Planned schedule:**

№	Task	Execution term	Execution mark
1	Task receiving	13.05.2024-16.05.2024	Executed
2	Purpose formation and describing the main research tasks	17.05.2024-18.05.2024	Executed
3	Analysis of existing methods	19.05.2024-23.05.2024	Executed
4	Theoretical consideration of the problem solution	24.05.2024-28.05.2024	Executed
5	Developing of methods for planning the trajectory of a moving object capable of autonomous movement in various environments	30.05.2024-2.06.2024	Executed
6	Making an explanatory note	2.06.2024-4.06.2024	Executed
7	Preparation of presentation and handouts	5.06.2024-11.06.2024	Executed

7. **Date of task receiving:** “13” May 2024

Diploma thesis supervisor_____ Yuriy MELNYK.

(signature)

Issued task accepted_____ Yelyzaveta HULBINAS.

(signature)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Система автономної навігації колісного робота»: 58 ст., 29 рисунків, 13 використаних джерел.

Актуальність теми. Проблема створення навігаційної системи, що дозволяє рухомим об'єктам здійснювати автономний рух у реальних середовищах, дуже важлива в сучасному світі. Масове виробництво автономних роботів, здатних працювати у складних умовах, значно спростить життя людей. Нікому не доведеться ризикувати своїм життям, виконуючи роботу.

В даний час найчастіше використовуваними автономними рухомими об'єктами через зручність їх використання є мобільні роботи. У зв'язку з цим розглядається завдання побудови системи навігації саме для даного типу роботів.

Об’єкт дослідження – автономні мобільні роботи.

Предмет дослідження – система автономної навігації робота.

Мета дослідження – створення навігаційної системи, що дозволяє здійснювати автономний рух на площині (2D).

Наукова новизна – розроблено модуль автономної навігації робота з побудовою оптимальних алгоритмів обминання перешкод з використанням карт місцевості.

Методи дослідження: методи аналізу, аналітичного та комп’ютерного моделювання, методи випробувань і експерименту.

Матеріали дипломної роботи можуть бути використані для вирішення задач автономного виконання завдань колісними роботами різного призначення по картам місцевості з обминанням перешкод по найбільш оптимальному, в даних умовах, маршруту.

ABSTRACT

Explanatory note to the thesis "Autonomous navigation system of a wheeled robot": 58 p., 29 figures, 13 references.

Actuality of the theme. The problem of creating a navigation system that allows moving objects to move autonomously in real environments is very important in the modern world. Mass production of autonomous robots capable of working in difficult conditions will greatly simplify people's lives. No one should have to risk their life doing the job.

Currently, mobile robots are the most frequently used autonomous moving objects due to their ease of use. In this regard, the task of building a navigation system specifically for this type of robots is considered.

The object of research is autonomous mobile robots.

The subject of research is a system of autonomous robot navigation.

The purpose of the research is to create a navigation system that allows for autonomous movement on a plane (2D).

Scientific innovation – a robot autonomous navigation module was developed with the construction of optimal algorithms for bypassing obstacles using terrain maps.

Research methods: methods of analysis, analytical and computer modeling, methods of tests and experiments.

The materials of the thesis can be used to solve the problems of autonomous performance of tasks by wheeled robots of various purposes on maps of the area with the bypassing of obstacles along the most optimal, under the given conditions, route.

PLAN

INTRODUCTION

The problem of creating a navigation system that allows moving objects to move autonomously in real environments is very important in the modern world. More and more tasks are being performed by some service robots instead of people. Over time, most processes for the production of material assets, exploration of new territories (including in space) and servicing people will be performed by autonomous robots.

Mass production of autonomous robots capable of working in difficult conditions will greatly simplify people’s lives. No one should have to risk their life doing the job.

Creating some universal method that can automate robot movement in various environments will be a huge step towards creating fully autonomous and multifunctional robots. In this regard, this task is currently truly relevant and requires finding more optimal solutions in many respects, such as reducing the error in calculations by sensors of distances to environmental objects and the ability to create groups of robots that can jointly perform one task that a mobile robot cannot can do it alone.

SECTION 1

NAVIGATION SYSTEM FOR MOBILE ROBOTS

The main problem of all currently existing mobile devices that move independently, without human control, remains navigation. To successfully navigate in space, the robot's on-board system must be able to build a route, control movement parameters (set the angle of rotation of the wheels and the speed of their rotation), correctly interpret information about the surrounding world received from sensors, and constantly monitor its own coordinates.

Computer route planning systems are quite well developed. Initially, they were created for the simplest virtual environments, and the program simulating the robot's actions quickly found the optimal path to the goal in two-dimensional labyrinths and rooms filled with simple obstacles.

When fast processors appeared, it became possible to form a movement trajectory on complex three-dimensional maps and in real time. A significant contribution to this algorithmic direction has been made by companies that develop computer games and finance relevant research. In modern games, each of the conflicting sides involves several hundred combat units operating on randomly generated three-dimensional maps, and each unit quickly and quite efficiently finds its way to the goal. Therefore, in real operating conditions such algorithms are ineffective.

A full-fledged robot must determine its own coordinates and choose the direction of a movement only based on the indicators of on-board sensors, therefore, artificial intelligence systems created for autonomous machines are focused on supporting a continuous cycle of "poll of sensors – making an operational decision to change the route".

There can be several such cycles – one is responsible for following the main route, the other for avoiding obstacles, etc. In addition, at the hardware level, each cycle can be supported by sensors of different types and different operating principles, generating data streams of different volumes and intensity.

Mobile robot navigation covers a wide range of different technologies and applications. It relies on both very old technologies and the most advanced achievements of science and technology.

The navigation system in robotics is divided into three levels:

- global – determination of the absolute coordinates of the device when moving along long routes;
- local – determining the coordinates of the device in relation to some (usually starting) point. This scheme is in demand by developers of tactical unmanned aircraft and ground robots performing missions within a pre-known area;
- personal – the robot positions parts of its body and interacts with nearby objects, which is important for devices equipped with manipulators.

Navigation systems are classified according to one more criterion – they can be passive or active. A passive navigation system involves receiving information about one's own coordinates and other characteristics of one's movement from external sources, while an active one is designed to determine the location only on one's own. As a rule, all global navigation schemes are passive, local ones are both, and personal schemes are always active [6].

In the process of local navigation, a number of tasks arise:

- Trajectory motion control;
- Detection localization of obstacles;
- Ensuring movement along a given route: along a strip, in a labyrinth, on a map of the area;
- Determination of own coordinates in local space;
- Scanning space;
- Drawing up a map of the area and linking it to it.

1.1. Hierarchical structure of the robot control system

When building a robot navigation system, many technical difficulties appears, the solution of which is assigned to the control system. To move towards a goal, the robot needs to form a fairly accurate image of the space around it. Today this is achieved mainly by using laser rangefinders and ultrasonic emitters (sonars). However, the laser beam will help to obtain an image of the environment only in the line of sight. In addition, small interference often appears along the path of the beam, introducing an error into such an image**.** And ultrasonic sensors are characterized by a long response time (if the robot is in a large and open space), on the order of tenths of a second, which does not allow the robot to move quickly. The speed of sound in different conditions can also "float", affecting the accuracy of distance estimation, as a result, the overall picture of the environment in the robot's "head" is distorted.

Creating three-dimensional maps using lasers in real time is even more difficult and, at a minimum, requires significant computing power, which has not yet been implemented in the form of compact on-board boards. For these reasons, the value of the information coming from on-board sensors is low. The robot needs to translate it into a formal and structured "verbal" description of the world (recognition task), which so far turns out rather poorly. Technical vision systems promise to provide the greatest effect here but they are also still imperfect. However, this drawback has already been overcome in projects where robots operate in buildings and in any other predetermined environment [6]..

A promising idea turned out to be storing a complete map of the area in the machine's memory. Usually it is presented in a geometric (very detailed, but also very voluminous) or topological (compact, symbolic, but less detailed) form.

The best results are obtained from three-dimensional maps, but their storage and processing by the robot's on-board system is difficult: the computing resources required are too large by today's standards. And most importantly, the robot is not always able to correctly determine its real location on such a map.

A lot of research is being done to train autonomous vehicles in methods of independently constructing terrain maps. This area is heavily funded by the military, who are interested in automating the processes of constructing maps of any area of the Earth. The obstacle to this lies not so much in the weakness of the algorithms, but in the relatively slow on-board processors. During movement, the robot must quickly and accurately control the motor and the position of the wheels.

Some robotics problems, in principle, do not allow an exact solution (this is, for example, the problem of controlling the torque of an electric motor so that the robot strictly follows the route). In other problems related to the dynamics of robot motion (the field of theoretical mechanics), finding the answer is still very far away, and the search for approximate coefficients that determine the motion parameters requires the on-board device to constantly solve systems of differential equations. The robot must know its real location and it is almost always different from that stored in the on-board system.

Determining coordinates is a fundamental navigation problem, the answer to which is of interest not only to roboticists, but also to specialists from many other fields – primarily space, aviation and automotive.

The general functional diagram of a sensing robot equipped with a technical vision system (possibly together with other external information sensors) is shown in fig. 1.1. The robot's sensory system must supply its control system with information about the current situation in the external environment: the presence, type, parameters, location and orientation of objects of manipulation (impact); the correctness and quality of the robot's performance of technological operations and/or other actions; the existence of obstacles and ways to bypass them, etc. The control system also receives tasks (commands) to the robot from a human operator, other robots, higher-level computers, technological equipment or other devices. In accordance with the above diagram, information from the technical vision system can be supplied to different levels of the control system [3].

For example, data on the location of obstacles is needed to build up a model of the working environment in order to plan the robot's actions; the results of object

classification are necessary at the strategic level for dividing the general plan of action into specific manipulation operations, setting their sequence and parameters; information about the location and orientation of objects is necessary for the formation at the tactical level of the required movements of the robot, according to which, in turn, software laws for coordinated changes in the corresponding degrees of mobility are constructed: information about the deviation of the actual trajectory from the programmed one can be used directly at the executive level to generate control signals to the drives when executing the program in order to correct the movement of the robot's working body.

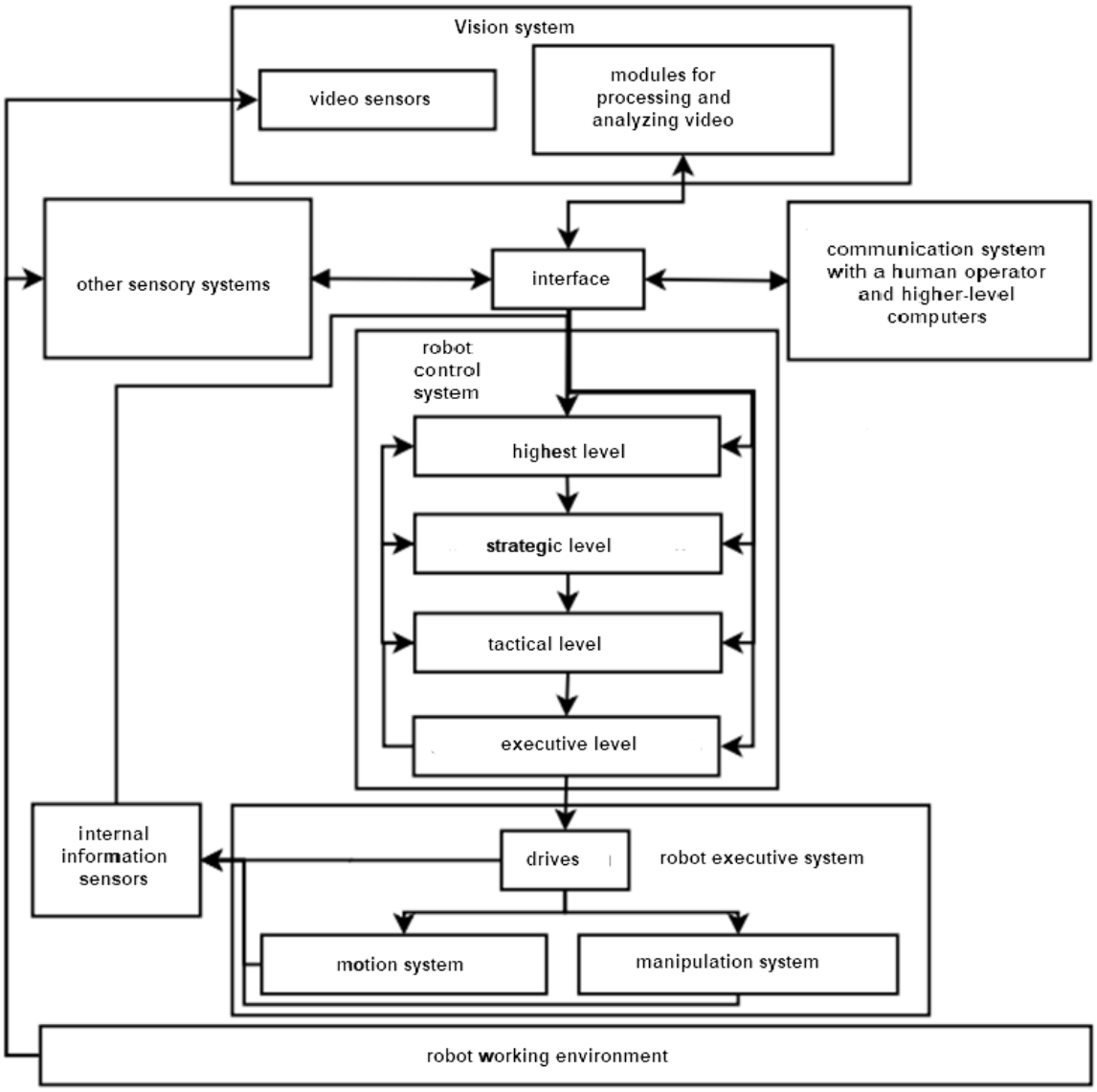

Figure 1.1 – Functional diagram of the robot's hierarchical control system

Regardless of how the video sensors are placed, the information obtained with their help is used to control the adaptive robot in accordance with one of the following two principles. The first is based on continuous (or quasi-continuous) input of a video feedback signal to correct the robot's trajectory. Servo systems operate on this principle, minimizing deviations of the current position from the desired one, which is determined based on visual observation of the target object.

The second principle is based on inputting information from the video sensor into the robot control device in discrete portions. Each such portion serves to develop programmed movements for the next time interval, during which the robot moves to the next target position "blindly," i.e., without continuous visual feedback. Each of these principles of using video information to control robots has its own area of application.

Although systems with "visual servo control" are undoubtedly promising, video sensors based on the second principle described above, "kissing," are still much more widespread. In the simplest case, based on the results of the operation of the video sensor, based on a priori specified conditions, a decision is made only on starting or stopping (interrupting) a predetermined program of robot actions, switching to one or another rigid subroutine, changing the sequence of execution of commands of the control program, complete information about each of which should be entered into the control device before the robot starts operating [3]. Wider capabilities are provided by adaptively changing the robot's control programs themselves in accordance with the actual situation, determined from information from the video sensor. Adaptive robots are capable of automatically generating movements during operation without the need for a priori human indication of detailed laws of change in all controlled coordinates.

1.2. Obstacle detection

There are many ways to detect obstacles. These may include ultrasonic, radar, optical and other sensors.

In the practice of controlling a mobile robot, the problem appears of quickly collecting information about the space around it in order to correctly complete the task. To increase the speed of completing a task, it is necessary to increase both the speed of movement of its mechanical parts and the speed of collecting and processing information [2]. As a result, the mechanical impact of physical contact between the robot and an obstacle or objects being manipulated can be destructive to the system. In addition, a mobile robot must be able to obtain information about all objects around it, both stationary and moving, which would allow it to learn and plot the shortest course to speed up the task. Cases when information about the environment is pre-installed by developers into the program are increasingly becoming an exception, since such devices are not sufficiently universal and safe [10].

The main methods for detecting obstacles and identifying their coordinates are considered. Let’s assume that the video sensor is located on the mobile base and is directed in the direction of its movement. In this case, obstacles that must be avoided or navigation beacons come into view. To do this, it is necessary to determine the presence of the obstacle itself and the distance to it. Let’s assume that the outline or characteristic points of the obstacle can be determined from the image. In this case, there are two main approaches to determining the distance to this obstacle. One of them is based on the use of stereo vision methods and a method related to stereo vision – triangulation, and the second is based on the use of a number of frames in processing, rather than just one, along with measuring the parameters of the robot’s movement.

1.3. Stereo vision method in the problem of determining the distance to an obstacle

The stereo vision method for determining the distance to an obstacle is based on the fact that a video camera, just like a photo camera, is essentially an angular device. Each point of the real object O is mapped to point I of the real image, located behind the focus f of the objective or lens, by means of the light emitted or reflected by this object, as shown in fig. 1.2.

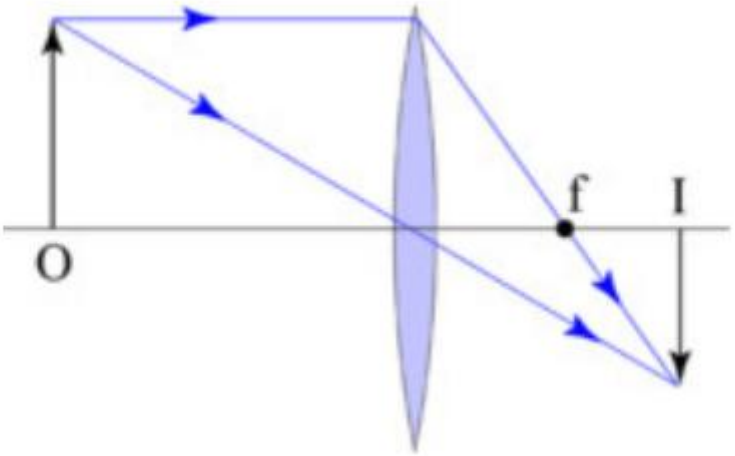

Figure 1.2 – Formation of an optical image point

To construct an optical image, an optical objective or a collecting lens is used. In optics, these devices are considered equivalent, but the lens consists of a set of lenses (in some lenses, mirrors), designed to mutually compensate for aberrations and assembled into a single system inside the frame [10].

The image of one plane perpendicular to the main optical axis of the lens is constructed in one plane of the actual image, which for the imaged plane is called the plane of best vision. The image of any other plane will be shifted along the main optical axis of the lens. To input an image into a computer (convert an optical image into a graphic image), in most cases, a charge-coupled device (CCD) is used. Devices with charge injection (CI) and others can also be used. The CCD is located in a plane perpendicular to the main optical axis behind the lens focus. In the CCD plane, a real image of a plane located at a certain predetermined distance from the lens will be

constructed. This distance is determined by the lens and the location of the CCD relative to it. Any other plane will be depicted in the CCD plane as a spot, the larger the further its plane of best vision is from the CCD plane. As long as the spot produced on the CCD does not exceed the size of one photosensitive element, the image will be sharp, otherwise it will begin to blur. The distance between the extreme optical image planes in which the CCD image remains sharp is called the depth of field of the lens. Each of these planes corresponds to a plane in the space of objects. The image of a scene constructed in the space of objects between these planes, taken from a CCD, will be sharp. Objects located outside these planes will be blurred by the (X) axis of the graphic image. If a feature point is identified in each image, an equivalent triangle can be constructed, shown in fig. 1.3.

The coordinate of a point along each axis in the image is a function of the angle between the optical axis and the projection of the ray forming this point onto the plane formed by the optical axis and the perpendicular to it corresponding to the considered axis of the image. The idea of the stereo vision method is: there are two cameras, each of which has a characteristic obstacle point in its field of view, and the cameras themselves are located at a known distance from each other. Conveniently, let’s assume that the cameras are oriented in such a way that the optical axes of the lenses are collinear, the line connecting the focal points of the lenses is perpendicular to both optical axes, and the lines parallel to it, passing through the optical axes and falling into the field of view of both cameras, coincide with the horizontal axis (X) graphic image. If a feature point is identified in each image, an equivalent triangle can be constructed, shown in fig. 1.3.

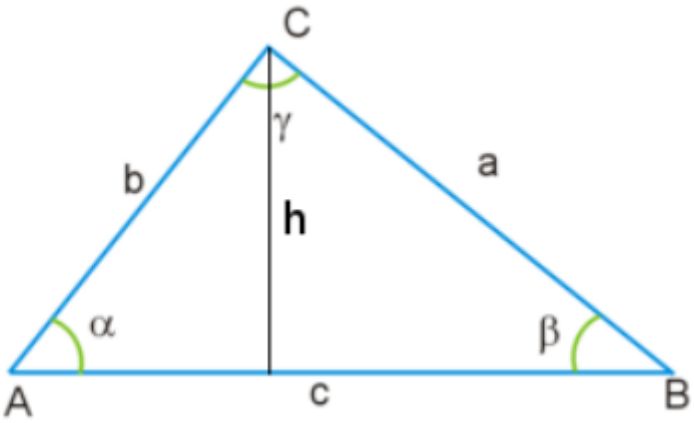

Figure 1.3 – Equivalent triangle (point A is the position of camera 1, point B is camera 2, point C is the projection of a point in object space onto the plane in which the optical axes of the lenses lie).

The base "c" of the triangle is equivalent to the distance between the cameras, and the angles at basis – determined by the image from each camera. Now that one side and two angles have been defined for a triangle, any of its characteristics can be calculated, including its height [10]. The height corresponds to the distance from the line connecting the centers of the CCD matrices to the projection of a point in the space of objects onto the plane in which the optical axes of the lenses lie, and is determined by the equation:

$$h = \frac{c \cdot tg\beta \cdot tg\alpha}{tg\beta + tg\alpha} \quad (1.1)$$

Now, using the angle γ of the elevation of a point in space above its projection onto the plane in which the optical axes of the lenses lie, the required distance L to the point itself in the space of objects can be calculated using the equation:

$$L = \frac{h}{tg\gamma} \quad (1.2)$$

The angle γ is determined by the vertical coordinate (Y) on the graphic image obtained from any camera or by the average value. One of the problematic areas in stereo vision is determining the relationship between the coordinates in the image and the angular coordinates of the ray coming from the imaged point. The most rational method of solving this problem is experimental.

For a camera-lens pair, in a plane perpendicular to the optical axis of the camera, at a given distance, a calibration surface with a uniform grid applied to it is placed. Grid nodes located on the image axes have, on the one hand, a known coordinate in the image, and on the other, a known coordinate in space, which means a table of correspondence between angles and coordinates along each image axis can be compiled.

At intermediate points the function is found by linear interpolation. Another problem is the very definition of a common point in two images. The most straightforward way to solve this problem is to directly recognize patterns from two images. However, this task is quite complex, and its solution directly from images is complex, resource-intensive, and has a significant error in determining coordinates. In addition, even with successful detection of objects (especially those of similar shape and color), if several of them fall into the field of view of the camera, especially in cases where some of them fall into the field of view of only one camera, the problem of establishing correspondence with each other arises images of the same object detected on different cameras. To solve this problem, active or passive beacons are used, which mark obstacles or navigation beacons [10]. These can be multi-colored emitters or colored corner reflectors, if there is a spotlight on the mobile robot itself. The main condition is the ability to easily distinguish one beacon from another and from background radiation in the image. What is used for is a variation of simple geometric shapes, easily recognizable, and colors. In the case where you intend to work near sources of interference, it is advisable to choose beacons or a searchlight that emit in a wavelength range different from the one inherent in the interference. For example, if the robot is working in a crowded environment, it is not wise to use infrared. In addition, it is necessary to equip the system with optical filters that cut off wavelengths not used by beacons. These can be bandpass filters or, in the case of using beacons of the same color, line filters – allowing you to isolate almost only one wavelength and completely filter out background radiation at other wavelengths. Since a typical CCD matrix perceives radiation in a fairly wide range of wavelengths, even with significant brightness of a monochromatic beacon, the intensity of the light flux from a seemingly insignificant interference, when integrated over the entire wavelength range, can

generate false images of beacons. Thus, the use of optical filters makes it possible to organize the search for beacons even in a very noisy environment.

1.4. Triangulation method in the problem of determining the distance to an obstacle

Another method used to detect points belonging to an object in a video image and the distance to them is the triangulation method [6,9], using illumination with a laser beam or scanning of a laser beam. Similar methods are good for determining distances to objects at which the beam is directed, but are increasingly used in rangefinders or profilometers.

In the orientation of mobile robots, triangulation using laser illumination is not very suitable, since it is necessary to organize laser scanning of space, which, along with the low speed of household video cameras (25 frames per second), gives a rather low resolution. In addition, additional drives and moving parts appear, which means the weight increases, additional dynamic moments and forces arise, as well as weak points of the structure, since it is the mechanical components that have the least reliability. The use of a video sensor in such a system is redundant, since instead of a video sensor/laser pair, a laser lidar [8] can be used, which provides greater scanning speed and accuracy. Another approach is the triangulation method using a light strip. The use of illumination, made in the form of a laser beam scan in a plane parallel to the floor, similar to that shown in figures 1.4, 1.5, except in cases where the presence of obstacles is expected below or above this plane, is quite acceptable.

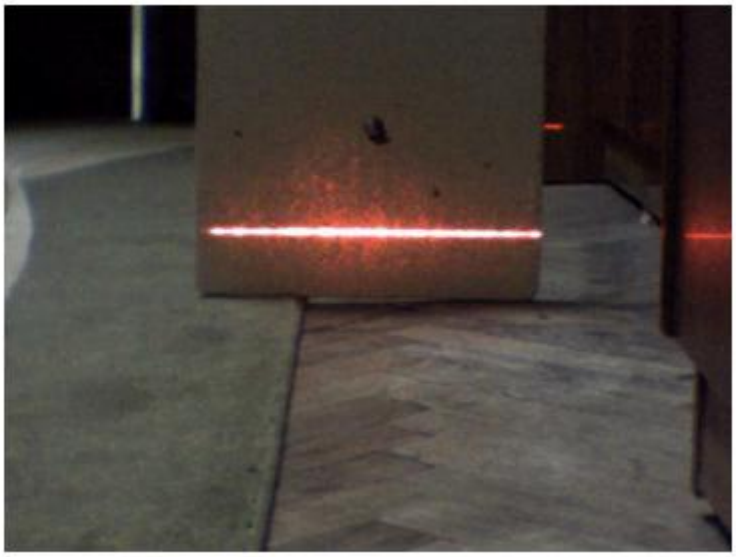

Figure 1.4 – Image of a nearby object illuminated by a laser scan

From Figures 1.4-1.5 it can be seen that the laser scan line in the image has a broken character. The areas between the line breaks in the image correspond to individual objects illuminated by the laser scan. This provides a mechanism for selecting individual objects and estimating their size. In addition, from Figures 1.4-1.5 it is clear that when an object is removed, the image of the intersection of the object with the laser illumination plane moves away from the latter, which provides a mechanism for estimating the distance to the identified object. This method looks quite simple, but it should be remembered that when using household cameras and lenses, the image of the line will be sharp enough only within the finite limits determined by the depth of field of the lens.

Figure 1.5 – Image of a distant object illuminated by a laser scan

The computational part of the triangulation method practically repeats the computational part of the stereo vision method. An additional advantage of this approach is that there is no need to use two cameras. Instead, the sweep is arranged so that in the equivalent triangle in Fig. 1.3, it originates from the base and coincides with the height. Then three quantities are known: the angle α, as before determined from the image, the distance from the vertex A to the height h (from the focus of the video camera to the laser scan), which will note as l, as well as the angle between the height and the base, equal to 90 degrees. Thus, the following equation is valid:

$$h = l \cdot tg\alpha \quad (1.3)$$

The main disadvantage of this approach is the detection and determination of distance only to objects in the laser scanning plane, which is why instead of two equations (1.1)–(1.2), one (1.3) appears here. In order to capture the return beam in such a system, suitable lighting conditions and reflectivity of the target surface are necessary so that the fringe is the brightest in the image. In practice, this is achieved by treating the target surface with a matte finish, using high-contrast cameras, or reducing the level of ambient lighting. When using the system in a robot to recognize objects in a room environment, there is a large amount of noise that makes work difficult: smooth

surfaces can cause secondary reflections, edges and textures can have a hatched appearance, and in the end, crosstalk is possible when more than one robot is working with a similar system. Several approaches have been found to increase the robustness of such systems. Using a stereo pair to register a band allows you to remove false reflections but is highly dependent on user information on the object given a priori [8]. You don't have to mix images from two cameras but use one as a checker for the other. Robustness is achieved through sequential checks of the received ranging information, the most important of which require independent conversions from one of the cameras. Another limitation is that the error correction method does not provide a way to get rid of multiple fringe images resulting from secondary reflections. As a result, all measurements indiscriminately during secondary reflections are not taken into account. Also, the elimination of incorrect information can be achieved through the interaction of independent scanning devices [12]: two laser stripes and one camera are used. In addition to robust ranging, surface normals can be obtained. The disadvantage of this method is that the distance can only be obtained from the image to the point where the stripes intersect, so it will take much more time to obtain a complete distance map. Some other methods of interest propose using a single camera with a single stripe. Reflections can be recognized by moving the sensor relative to the environment and analyzing changes in the converted ranging information. By modulating the periodic intensity of the beam, extraneous noise is eliminated [12]. Both of these methods obtain ranging information from multiple images, making them prone to errors. Moreover, modulating the beam intensity does not eliminate secondary reflections, which change in unison with the primary reflection. To eliminate secondary reflections from metal surfaces, linearly polarized light can be used, so that with each new reflection such a beam changes its polarization. However, the registration method at the receiver is difficult: several measurements through different polarization filters are required.

CONCLUSION

The task of navigation remains a key problem in mobile robotics, which implies determining the position of a mobile robot in the workspace – localization and drawing up a representation, description of the surrounding world – cartography. Information about the current position of the robot is necessary to solve most encountered control problems: passing a given path, finding a path to a given point, returning to the starting position. Information about the surrounding world, which is most often presented in the form of a map or terrain plan is necessary to remember the route taken, plan a trajectory around static obstacles and track dynamic objects.

SECTION 2

PLANNING THE PATH OF A MOBILE ROBOT

Every modern enterprise is looking for all possible ways to make its work as efficient as possible. Autonomous mobile robots (AMRs) are coming to the aid of healthcare institutions, agricultural companies, manufacturing warehouses and logistics organizations. Such devices successfully replace outdated equipment and improve characteristics such as speed, accuracy and safety.

The task of creating robots that can move without human assistance from point A to point B, avoiding collisions with obstacles, consists of many different subtasks.

2.1. Description of the movement of a moving object

Two types of moving objects are considered [11]: tracked and wheeled mobile robots. The robot's movement was studied in a Cartesian coordinate system centered at point (0,0) on the environment map.

Equations of motion of a moving object:

$$\{\dot{v} = a\ \dot{\omega} = \varepsilon\ \dot{\varphi} = \omega\ \dot{x} = v * cos(\varphi)\ \dot{y} = v * sin(\varphi) \quad (2.1)$$

Where x, y are the coordinates corresponding to the center of the circle describing the mobile robot, φ is the heading angle, v, a are the linear speed and acceleration, ω, ε are the angular speed and acceleration.

A tracked mobile robot can move in any direction on a plane, subject to the constraints imposed by the equations of motion.

Unlike a tracked robot, a wheeled robot has limited capabilities in choosing the direction of movement (fig. 2.1). The robot is presented in the shape of a rectangle with an aspect ratio of 1:2.

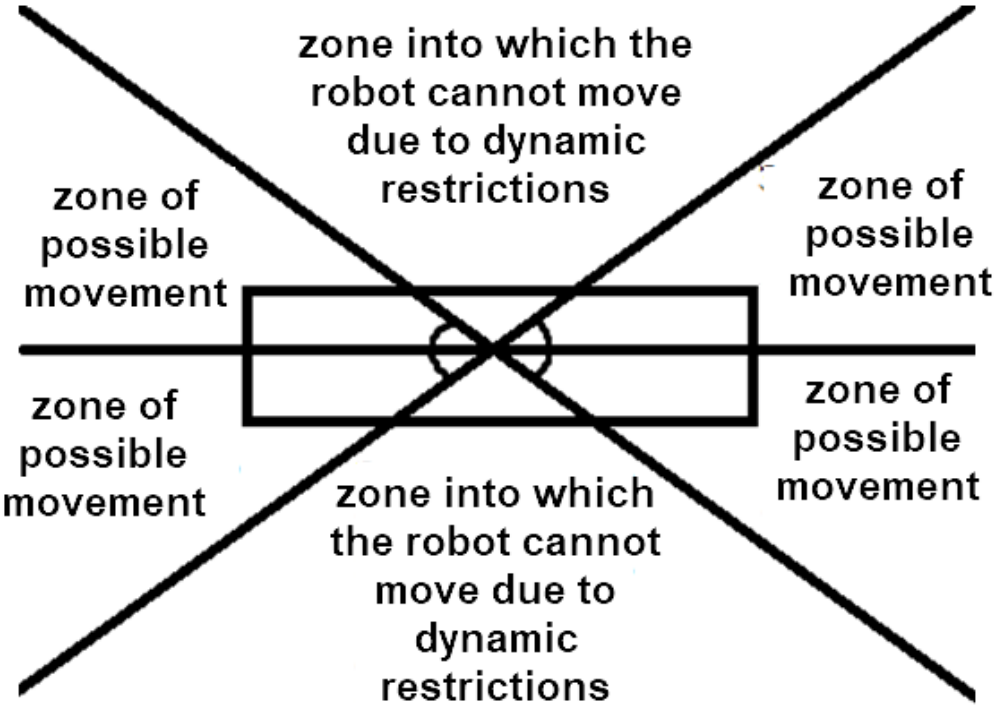

Figure 2.1 – Possible directions of movement of a wheeled robot

It was considered a movement in which the angle of rotation relative to the coordinate system under consideration is maintained after moving in any of the possible directions.

2.2. Navigation system structure

The resulting structure of the navigation system of an autonomous mobile robot is shown in the fig. 2.2.

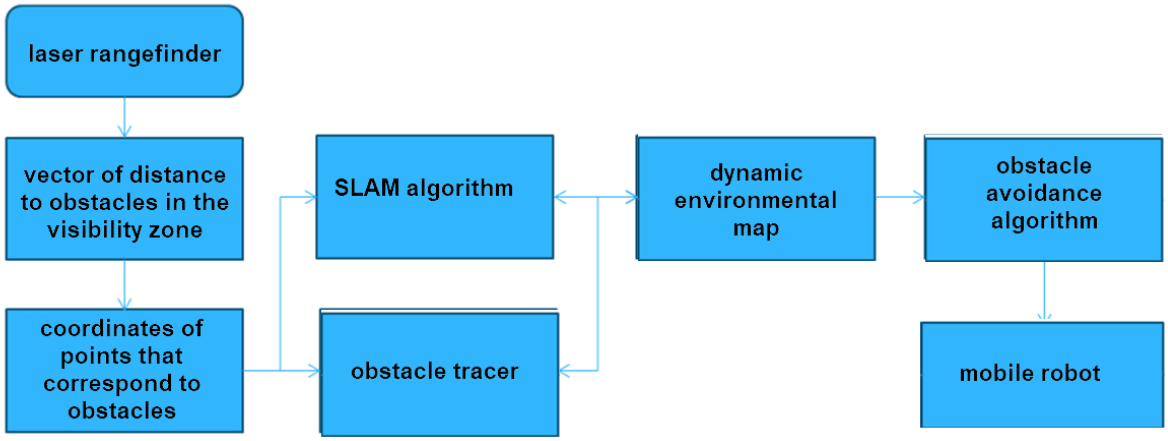

Figure 2.2 – Navigation system structure

Elements of this system are [2]:

- Laser range finder – a device that allows to scan the surrounding space and receive data about objects (obstacles) located in a given space in the form of distance vectors;
- SLAM algorithm (Simultaneous Localization and Mapping) – an algorithm developed for localizing a mobile robot in space, as well as constructing a dynamic map of the environment;
- The obstacle tracer, based on the available data on the current location of the robot and the current scan of the environmental map, builds a list of moving obstacles and predicts their location at the next time;
- Obstacle avoidance algorithm – an algorithm that allows to construct a trajectory for a mobile robot to avoid obstacles given the available data on the robot's location at a given time and the current scan of the environment map.;
- Mobile robot – an autonomous robot moving along a given trajectory with the help of some control.

2.3. SLAM algorithm

The SLAM algorithm is necessary to create mobile robots that can move autonomously in a non-deterministic environment.

There are two kinds of fundamental characteristic SLAM navigation methods. These are methods that use various types of filtering and methods using Bundle Adjustment.

MonoSLAM. The very first visual monocular SLAM method was an algorithm developed back in 2002 called MonoSLAM [13]. This method is a typical representative of VSLAM methods that work using a filtering process. The MonoSLAM algorithm has six degrees of freedom of the camera position (DOF) and the coordinates of the position of singular points in three-dimensional space are represented as a state vector of an extended Kalman filter (EKF).

The camera position calculation is based on the received motion model data. As a result of the calculated new camera position, new special points are added. It is also worth noting that the initial construction of the environment map occurs based on the visible special points on the current frame.

The following stages of this algorithm can be distinguished:

- the initialization process occurs with previously known special points on the map;
- estimation of the camera movement and three-dimensional positions of the object's special points is performed using an extended Kalman filter.

The main disadvantage of this algorithm is the increase in the number of calculations as the surrounding space expands, which leads to an increasing number of calculations of new singular points. Because of this, the size of the state vector increases, which does not allow the use of this algorithm in real-time systems.

PTAM. To solve the main problem of the MonoSLAM method, an algorithm called PTAM took the path of dividing the tasks of mapping and tracking between two CPU threads [13]. Because of these two threads execute in parallel, the computational cost of mapping does not have a significant impact on the tracking task. Therefore,

during the mapping process, can use Bundle Adjustment. This means that the tracking task can be performed in real time while 3D modeling of singular points occurs at a high computational cost.

The PTAM algorithm was the first algorithm to use Bundle Adjustment in real time. Subsequently, the multithreading approach began to be often used in other VSLAM algorithms. The construction of the initial map of the environment in PTAM occurs using the five-point algorithm [13]. To clarify the camera position, environmental map points are projected onto the image and a cloud of supposed visible points is constructed from them. During the matching process, the position of new points is determined by triangulation on specific frames, which are called key points. An important development in the history of the development of SLAM algorithms, thanks to the experience of developing PTAM, is the introduction of a keyframe-based mapping system. To determine a key frame, the input frame is compared with another key frame and if the difference is large, then the input frame is taken as a key frame. To perform triangulation, there must be a significant difference between the input and key frames. In PTAM, optimization of the three-dimensional position of special points occurs by applying a global Bundle Adjustment with certain key frames, as well as with all key frames on the environment map. It is also worth noting that the PTAM algorithm uses a relocalization method for the camera tracking process [2].

In the task of finding the most suitable key frame of the resulting image, a randomized tree search classifier is used. As a result, the following modules can be distinguished in the PTAM method: 1) application of the five-point method for the task of initializing an environment map; 2) the position of the video camera is estimated using the coincidence of special points of the environment map and the resulting image; 3) using triangulation, the three-dimensional positions of object points are estimated, and the already estimated positions are optimized using Bundle Adjustment; 4) when using a randomized tree search classifier, the tracking process is restored.

The SLAM algorithm uses a Kalman filter to build a solution.

The Kalman filter is an efficient recursive filter that estimates the state of a dynamic system based on a series of imprecise measurements. It was developed in 1960

and named after Rudolf Kalman. This filter is necessary to eliminate the error of the SLAM algorithm caused by odometry. The Kalman filter processes the input data, i.e., the position of the robot and the singular points, and returns their estimated values.

The operating principle of the SLAM algorithm:

- The robot is in some unknown place. Using the data received by the sensors, a visible section of the map is constructed from a given position;
- Using the trajectory obtained at this step, the next position for movement is selected;
- There is a movement to a new position and a comparison of the current position with the expected one obtained at the previous step;
- Based on the received data and data from the previous iteration, the map is updated.

2.4. Constructing an environmental map in the form of a grid map

The environmental map is presented in the form of a grid map. The cell size is selected depending on the required accuracy. Moreover, the smaller the cell size, the longer the obstacle avoidance algorithm will work.

The resulting cells are divided into two types (fig. 2.3):

- free – these are the cells through which the robot can move unhindered);
- cells containing obstacles.

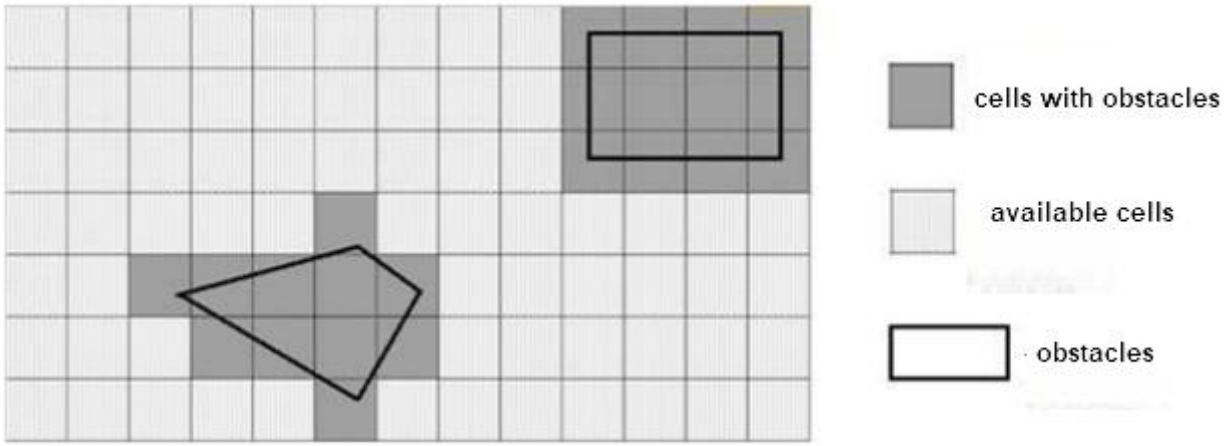

Figure 2.3 – Example of an environmental map

For robots that can move in all directions on a plane, the cell size usually corresponds to the dimensions of the robot, that is, the robot's outline can be fit into this cell. To achieve higher accuracy, it is possible to reduce the cell size, but the algorithm for constructing the trajectory must be changed.

2.5. Obstacle tracing

Tracing moving obstacles allows to determine the location at the following moments of time and the movement parameters of a mobile robot in a coordinate system related to the environment. Most obstacles move without changing direction. Thanks to this it is able to use obstacle tracing to predict their movement.

It may be apply tracing to an obstacle if:

- The obstacle is a rigid body;
- Submits to the equations of robot motion;
- Cannot change speed abruptly;
- It can be inscribed in a circle.

Each such obstacle has the following set of parameters:

$$\{\dot{v} = a\ \dot{\omega} = \varepsilon\ \dot{\varphi} = \omega\ \dot{x} = v * cos(\varphi)\ \dot{y} = v * sin(\varphi) \quad (2.2)$$

Where x, y are the coordinates corresponding to the center of the circle describing the obstacle, φ is the heading angle, v, a are linear speed and acceleration, ω, ε are angular speed and acceleration.

The obstacle state vector looks like this:

$$s = (x\ y\ \varphi\ v\ \omega\ a\ \varepsilon\ R)^T$$

Operating principle of the obstacle tracer:

- Creating a list of moving obstacles;
- Determining the radius of the circle R describing obstacles from the resulting list;
- Constructing a state vector s for each obstacle.

2.6. General planning structure

Although different algorithms are designed to construct a trajectory in different situations. The flow diagram of the rapid path planning algorithm is as shown in fig. 2.4

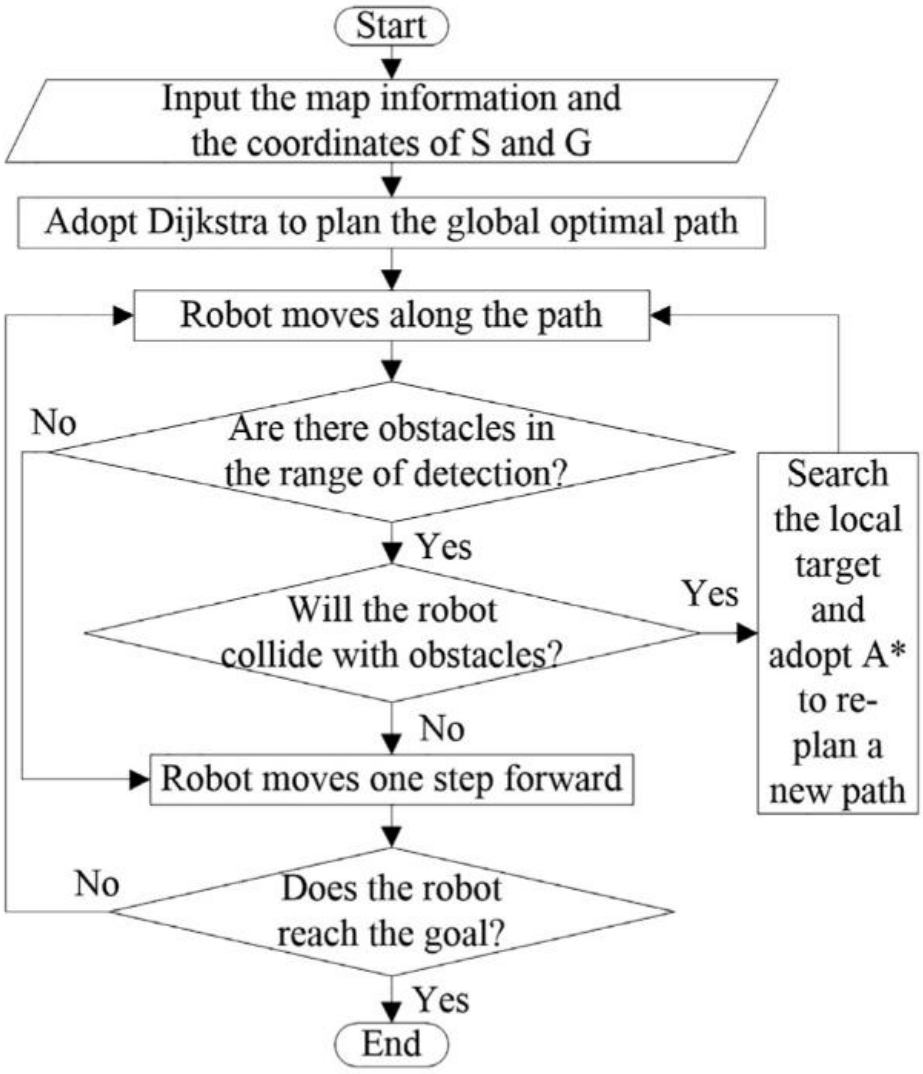

Figure 2.4 – Flow diagram of the rapid path planning algorithm

2.7. Algorithms for avoiding obstacles

To build trajectories in different situations, it is necessary to use different algorithms:

- Deterministic environment with stationary obstacles – Dijkstra's algorithm, Algorithm A*;
- Non-deterministic environment – D* algorithm.

In addition, the algorithm can change due to various dynamic properties of the moving object under consideration, for example, when considering problems of constructing a trajectory for a wheeled and tracked robot.

2.7.1. Dijkstra's algorithm

Dijkstra's algorithm is currently one of the most popular algorithms for finding the shortest path in a graph.

How the algorithm works [5]:

1. Building a map of the environment, also marking on it the initial position of the robot and the target point (fig. 2.5)

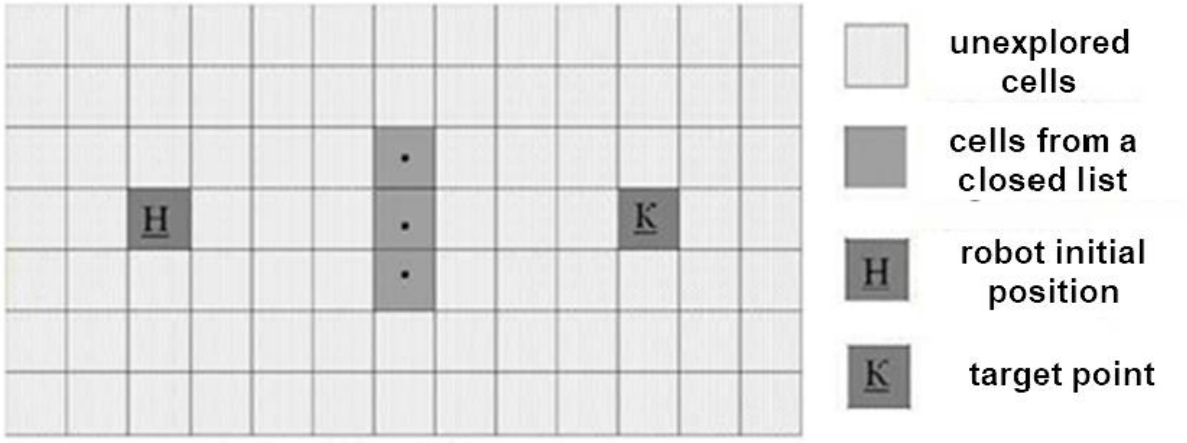

Figure 2.5 – An initial environmental map

2. Creating two lists of cells – open and closed. The open list initially contains only the cell corresponding to the initial position of the robot, while the closed list contains all cells containing obstacles (fig. 2.6).

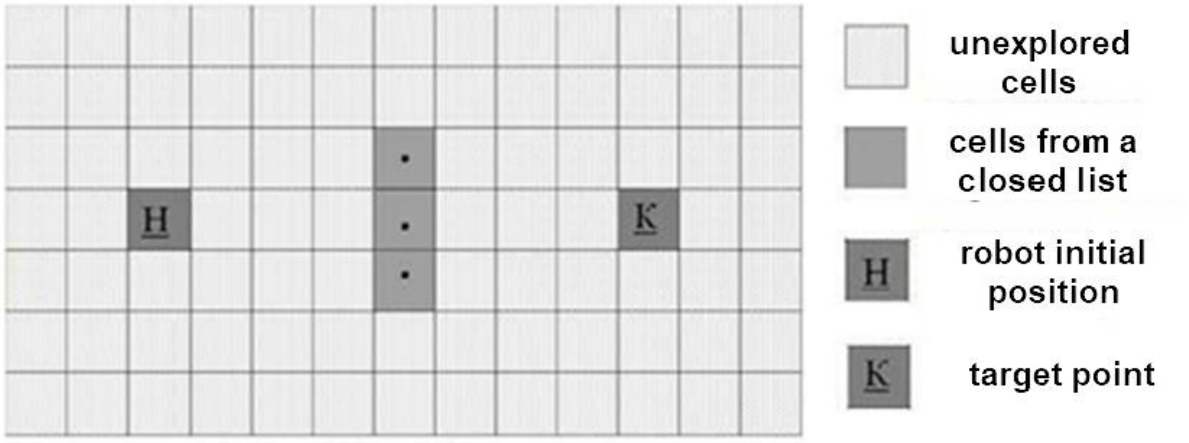

Figure 2.6 – An initial graphical display of the environment

3. Introducing the function l, which displays the distance from the initial cell to the given one.

4. Sorting the open list in ascending order by function l. Taking the first cell from this list as the cell in question.

5. Placing this point in a closed list. We consider all neighboring points that do not belong to the closed list (fig. 2.7). Adding them to the open list. If one of the points considered is the target, go to point 6, otherwise – to point 4.

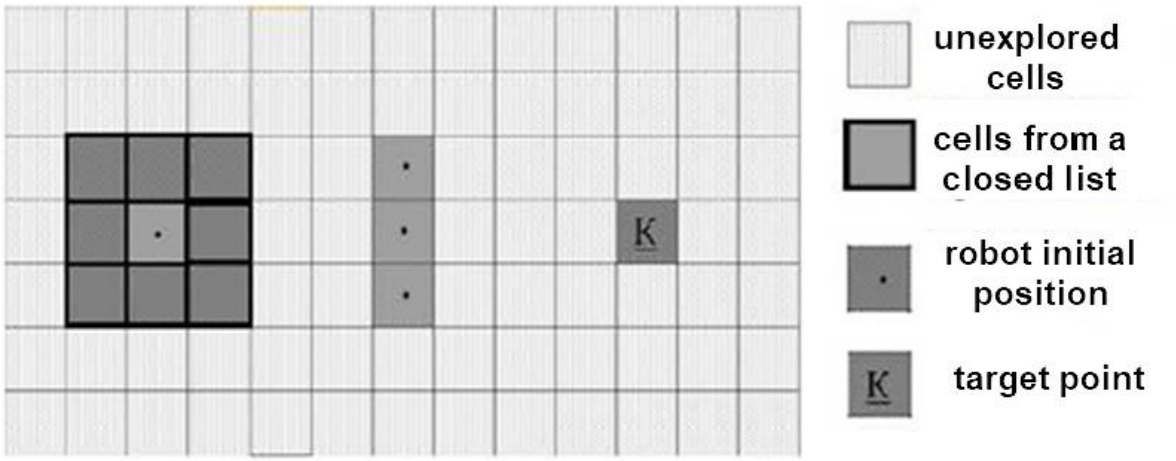

Figure 2.7 – Changing open and closed lists

6. Based on the data obtained, is building the initial trajectory.

2.7.2. Algorithm A* (A-star)

The "A*" algorithm is an improved modification of Dijkstra's algorithm, designed to more quickly find the optimal trajectory using heuristics. This paper considers the heuristic function $H = \sqrt{(x - x_к)^2 + (y - y_к)^2}$, where x, y – coordinates of the point under consideration, x_K, y_K – coordinates of the target point.

How the algorithm Works [1]:

1. Building a map of the environment, also marking on it the initial position of the robot and the target point (fig. 2.5)

2. We create 2 lists of cells – open and closed. The open list initially contains only the cell corresponding to the initial position of the robot, while the closed list contains cells containing obstacles (fig. 2.6).

3. We introduce the function $L = H + l$, where H is a heuristic function, l is the distance from the robot's initial position to the current one, which displays the minimum distance from the starting point to the target when constructing a trajectory through a given point.

4. We sort the open list in ascending order of the function L. We take the first cell from this list as the cell in question.

5. We place this point in a closed list. We consider all neighboring points that do not belong to the closed list (fig. 2.7). Add them to the open list. If one of the considered points is the target, go to point 6, otherwise – to point 4.

6. Based on the data obtained, build a trajectory.

2.7.3. Algorithm A* (A-star) for wheeled robots

Due to the limitations imposed on the ability to choose the direction of movement of a wheeled mobile robot, six main directions of movement were identified (fig. 2.8).

To determine the location of the robot at the current time, it is only need to know the coordinate corresponding to the upper left edge of the robot.

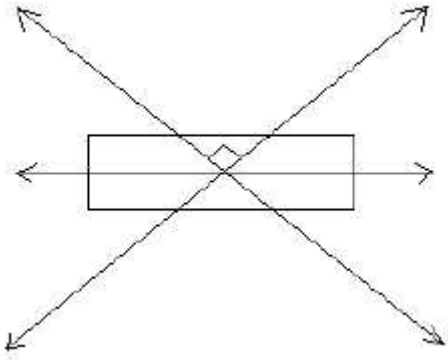

Figure 2.8 – Possible directions of movement of a wheeled robot

Since the standard A* algorithm examines eight directions of possible movement, and we consider only six, it needs to be changed.

Operating principle of the modified A* algorithm:

1. We build a map of the area, marking on it the initial position of the robot and the target point (fig. 2.5). In this modification, the initial position of the robot will correspond to the upper left cell belonging to the robot.

2. We create two lists of cells – opened and closed. The open list initially contains only the cell corresponding to the robot's initial position, while the closed list contains cells containing obstacles. An initial graphical display of an open and closed list is shown in fig. 2.6.

3. Let's introduce the function $L = H + l$, where H is a heuristic function, l is the distance from the initial position of the robot to the current one.

4. We sort the open list in ascending order of the function L corresponding to the cell.

5. Take the first cell from the resulting list as the cell in question.

6. Consider six neighboring cells that can be reached in one iteration. If these cells, as well as the cells necessary to make a movement from the previous cell to the current one, do not belong to the closed list, then add them to the opened list, otherwise – to the closed list (see fig. 2.9).

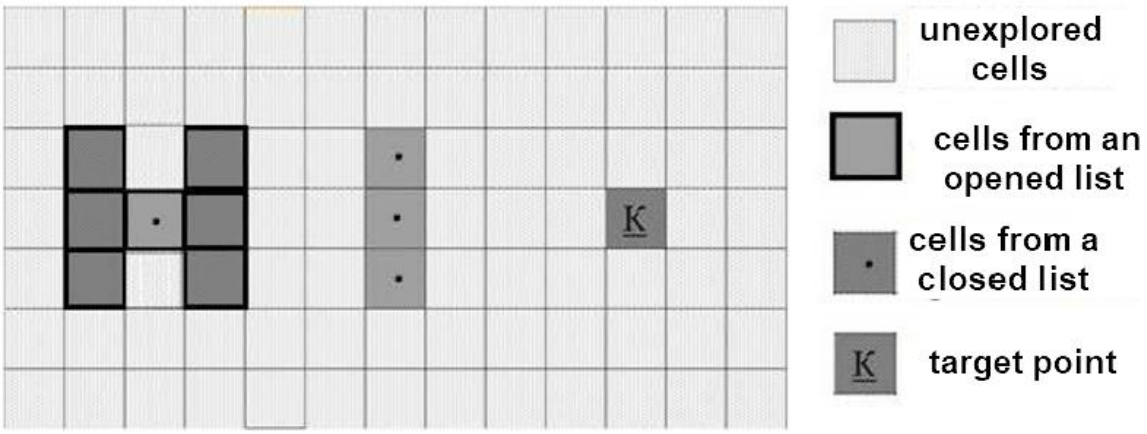

Figure 2.9 – Changing opened and closed list

7. If there is a cell in the open list corresponding to the target point, then go to the next point, otherwise we return to point 2.

8. Based on the data obtained, we build the desired trajectory.

2.7.4. Algorithm D* (D-star)

The robot can only move along a given grid, that is, at each new step it has four different options for making a movement – up, down, left, right (fig. 2.10).

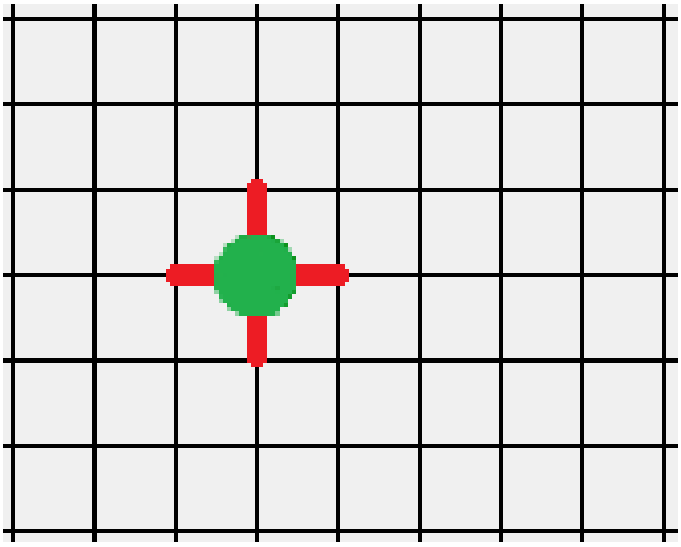

Figure 2.10 – Possible directions of movement

2.7.5. Algorithm D*

The “D*” algorithm is based on the use of the “A*” algorithm, which allows to build a trajectory around obstacles that is optimal in terms of the path length.

How the algorithm works:

1. We build an initial map of the environment, marking on it the initial position of the robot and the target point (Fig. 2.11).

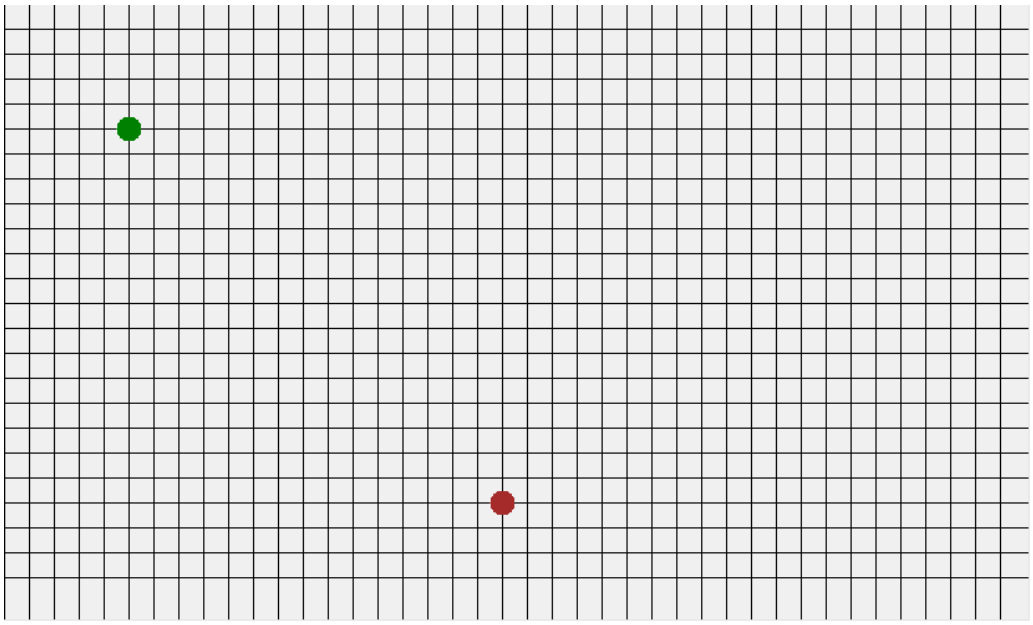

Figure 2.11 – An anitial environmental map

2. Create a history list that stores information about all cells considered during each trajectory construction. Using the “A*” algorithm, we build a trajectory from the target point to the starting point (fig. 2.12), adding all the considered cells from the history list. All elements of this list have the following characteristics:

a. Weight:

$\{x = 1, if\ the\ cell\ does\ not\ contain\ an\ obstacle\ x = 999999, if\ the\ cell\ contains\ an\ obstacle$;

b. Previous $(k-1)$ cell;

c. path length from target cell: $\rho(k) = \omega(k) + \rho(k-1)$;

d. current coordinates x и y.

3. The movement along the resulting trajectory from the starting point to the target point is considered. At each step, a check is made (fig. 2.13): if the next cell of this trajectory does not contain an obstacle and is not a target cell, move on; if it contains an obstacle, we stop moving and go to step 4; if it is a target cell, we complete the algorithm.

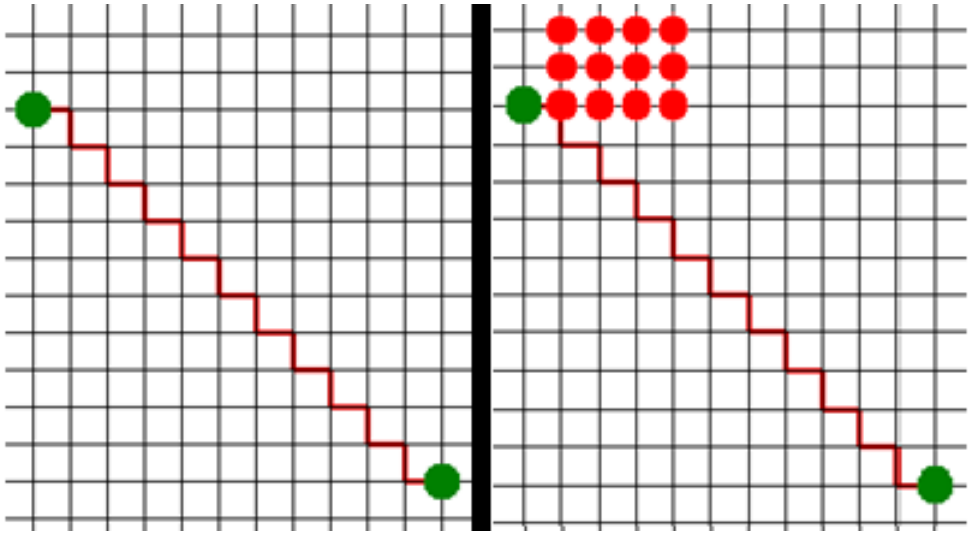

Figure 2.13 – Making a movement and checking the next point

4. If at least one of the neighboring cells (k_{new}) available in the history list and updated distance from this cell to the target $\rho(k_{new}) \leq \rho(k) + 3$, then we go to step 5, otherwise we make the current cell the initial position of the robot and go to step 2.

5. From the neighboring cells that satisfy the condition from step 4, select the one whose distance from the target point is the smallest and go to step 6.

6. Using information about the previous cell known for each cell, we recursively build a new trajectory passing through the cell obtained in step 5 and go to step 3.

2.8. Efficient Path Method

The effective path method presented in [11] makes it possible to optimize the resulting trajectory in terms of length.

As a reference point, select the second bend of the trajectory obtained using one of the considered algorithms. Representing the robot in the form of a circle describing

it. Denote the distance between the reference point and the current position of the robot as R. Due to the dynamic limitations of the robot, R is limited from above and below:

$$\{R_{max}(v_c) = (v_c + \Delta t * \dot{v}_a) * T_{max}\ R_{min} = v_{max} * T_{b_{max}} - \frac{1}{2} * \dot{v}_b * T_{b_{max}}{}^2\ ,$$

Where v_c – initial speed, $\dot{v}_a$ – maximum robot acceleration, T_{max} – time path to the nearest obstacle, $T_{b_{max}}$ – maximum braking time.

If the robot can move unhindered in a straight line from the current position of the robot to the reference point, then we change the trajectory, replacing the broken line leading to the reference point with a straight line. Otherwise, we must find the shortest possible curve along which the robot can move unhindered. To do this, consider the robot at the current moment and when reaching the reference point (fig. 2.14).

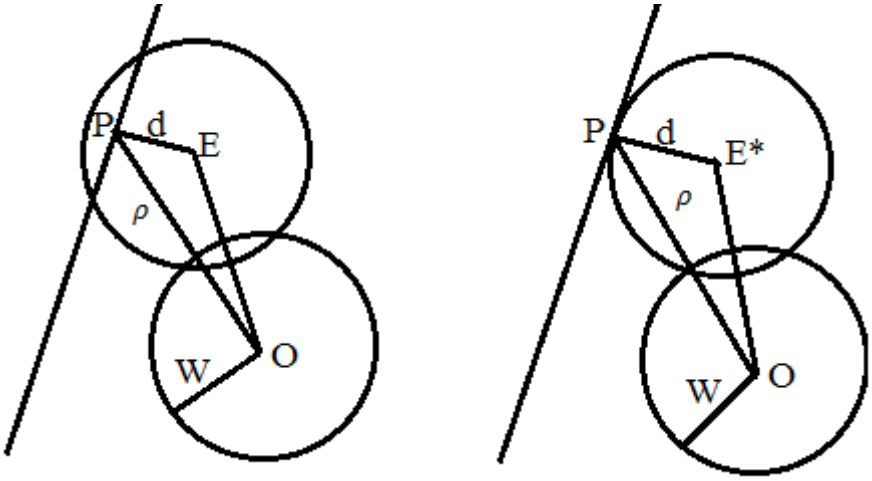

Figure 2.14 – An efficient path method

To find the shortest trajectory, is need to consider the case when the robot moves as close as possible to the obstacle, that is $d = |\overrightarrow{EP}| = |\overrightarrow{OE^*} - \overrightarrow{OP}| = W$. Based on the information received, using geometric calculations, a new polyline is found connecting the current position of the robot with the reference point.

CONCLUSION

Algorithms for constructing trajectories in different situations are analyzed, necessary for deterministic and non-deterministic environments.

It is concluded that it is impossible to create a universal robot movement algorithm. The algorithm used will depend on the conditions and purpose of the robot in each specific case.

A method of using the effective path method is shown to optimize the robot's trajectory according to the criterion of the minimum route length for a deterministic and non-deterministic environment.

SECTION 3

RESULTS OF SOFTWARE IMPLEMENTATION

Software has been developed in C# language that implements the considered algorithms for constructing a trajectory to avoid obstacles.

A fragment of the program code is shown in the fig. 3.1-3.8

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="12.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props" Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{BC166F77-00E8-4F49-8A07-0898C6D47153}</ProjectGuid>
    <OutputType>WinExe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>Motion_Control</RootNamespace>
    <AssemblyName>Motion_Control</AssemblyName>
    <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
```

Figure 3.1 – A fragment of the program code

```
54              this.button2 = new System.Windows.Forms.Button();
55              this.label14 = new System.Windows.Forms.Label();
56              this.velocity1 = new System.Windows.Forms.NumericUpDown();
57              this.label15 = new System.Windows.Forms.Label();
58              this.velocity2 = new System.Windows.Forms.NumericUpDown();
59              ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
60              ((System.ComponentModel.ISupportInitialize)(this.velocity1)).BeginInit();
61              ((System.ComponentModel.ISupportInitialize)(this.velocity2)).BeginInit();
62              this.SuspendLayout();
63              //
64              // pictureBox1
65              //
66              this.pictureBox1.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
67              this.pictureBox1.Location = new System.Drawing.Point(590, 37);
68              this.pictureBox1.Margin = new System.Windows.Forms.Padding(7, 6, 7, 6);
69              this.pictureBox1.Name = "pictureBox1";
70              this.pictureBox1.Size = new System.Drawing.Size(321, 185);
71              this.pictureBox1.TabIndex = 24;
72              this.pictureBox1.TabStop = false;
73              //
74              // label9
75              //
76              this.label9.AutoSize = true;
77              this.label9.Location = new System.Drawing.Point(551, 18);
78              this.label9.Margin = new System.Windows.Forms.Padding(7, 0, 7, 0);
79              this.label9.Name = "label9";
80              this.label9.Size = new System.Drawing.Size(50, 13);
81              this.label9.TabIndex = 25;
82              this.label9.Text = "( x1 ; y1 )";
```

Figure 3.2 – A fragment of the program code

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace Motion_Control
12  {
13      public partial class AddRectangle : Form
14      {
15          public AddRectangle()
16          {
17              InitializeComponent();
18          }
19
20          public string X1
21          {
22              get { return textX1.Text; }
23              set { textX1.Text = value; }
24          }
25
26          public string X2
27          {
28              get { return textX2.Text; }
29              set { textX2.Text = value; }
30          }
31
32          public string X3
```

Figure 3.3 – A fragment of the program code

```
54              this.button2 = new System.Windows.Forms.Button();
55              this.label14 = new System.Windows.Forms.Label();
56              this.velocity1 = new System.Windows.Forms.NumericUpDown();
57              this.label15 = new System.Windows.Forms.Label();
58              this.velocity2 = new System.Windows.Forms.NumericUpDown();
59              ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
60              ((System.ComponentModel.ISupportInitialize)(this.velocity1)).BeginInit();
61              ((System.ComponentModel.ISupportInitialize)(this.velocity2)).BeginInit();
62              this.SuspendLayout();
63              //
64              // pictureBox1
65              //
66              this.pictureBox1.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
67              this.pictureBox1.Location = new System.Drawing.Point(590, 37);
68              this.pictureBox1.Margin = new System.Windows.Forms.Padding(7, 6, 7, 6);
69              this.pictureBox1.Name = "pictureBox1";
70              this.pictureBox1.Size = new System.Drawing.Size(321, 185);
71              this.pictureBox1.TabIndex = 24;
72              this.pictureBox1.TabStop = false;
73              //
74              // label9
75              //
76              this.label9.AutoSize = true;
77              this.label9.Location = new System.Drawing.Point(551, 18);
78              this.label9.Margin = new System.Windows.Forms.Padding(7, 0, 7, 0);
79              this.label9.Name = "label9";
80              this.label9.Size = new System.Drawing.Size(50, 13);
81              this.label9.TabIndex = 25;
82              this.label9.Text = "( x1 ; y1 )";
```

Figure 3.4 – A fragment of the program code

```
57        mimetype: application/x-microsoft.net.object.bytearray.base64
58        value   : The object must be serialized into a byte array
59                : using a System.ComponentModel.TypeConverter
60                : and then encoded with base64 encoding.
61        -->
62      <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
63        <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
64        <xsd:element name="root" msdata:IsDataSet="true">
65          <xsd:complexType>
66            <xsd:choice maxOccurs="unbounded">
67              <xsd:element name="metadata">
68                <xsd:complexType>
69                  <xsd:sequence>
70                    <xsd:element name="value" type="xsd:string" minOccurs="0" />
71                  </xsd:sequence>
72                  <xsd:attribute name="name" use="required" type="xsd:string" />
73                  <xsd:attribute name="type" type="xsd:string" />
74                  <xsd:attribute name="mimetype" type="xsd:string" />
75                  <xsd:attribute ref="xml:space" />
76                </xsd:complexType>
77              </xsd:element>
78              <xsd:element name="assembly">
79                <xsd:complexType>
80                  <xsd:attribute name="alias" type="xsd:string" />
81                  <xsd:attribute name="name" type="xsd:string" />
82                </xsd:complexType>
83              </xsd:element>
84              <xsd:element name="data">
85                <xsd:complexType>
86                  <xsd:sequence>
87                    <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
88                    <xsd:element name="comment" type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
```

Figure 3.5 – A fragment of the program code

```
1       using System;
2       using System.Collections.Generic;
3       using System.Linq;
4       using System.Threading.Tasks;
5       using System.Windows.Forms;
6
7       namespace Motion_Control
8       {
9           static class Program
10          {
11              /// <summary>
```

Figure 3.6 – A fragment of the program code

```
# User-specific files
*.suo
*.user
*.userosscache
*.sln.docstates

# User-specific files (MonoDevelop/Xamarin Studio)
*.userprefs

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
x64/
x86/
bld/
[Bb]in/
[Oo]bj/
[Ll]og/
```

Figure 3.7 – A fragment of the program code

Examples of the operation of this software and the results obtained during the comparison of the studied algorithms are shown.

3.1. Building an obstacle map

An example of constructing a grid map obtained using the created software (Fig. 15):

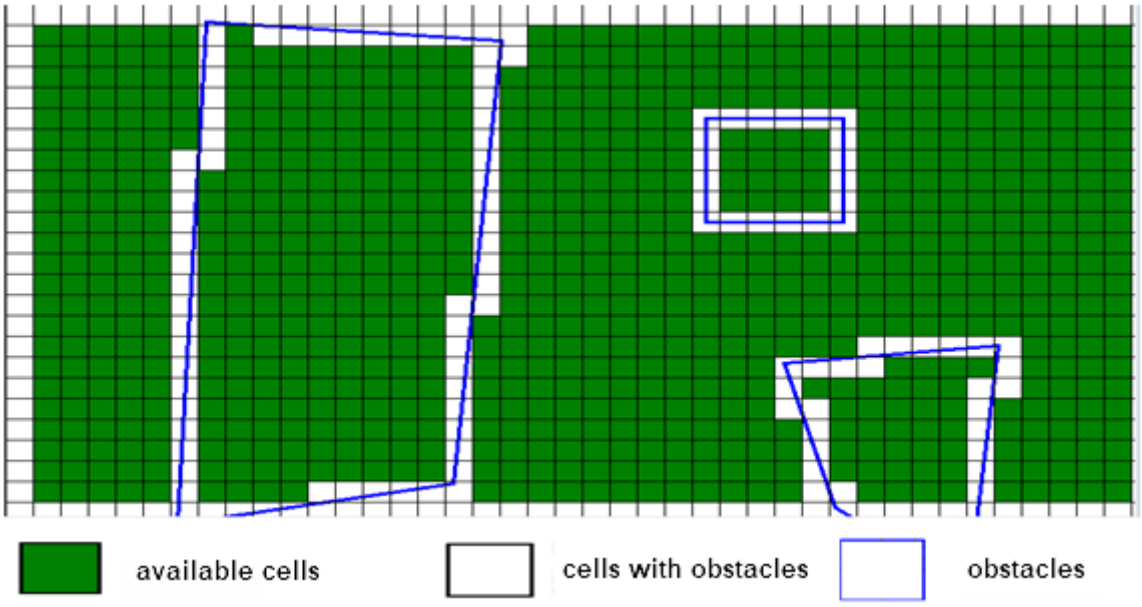

Figure 3.1. – An example of a grid map

3.2. Implementation of the considered algorithms

This section presents the key points in the operation of the following algorithms:

- Dijkstra's algorithm (Fig. 3.2);
- Algorithm A* (Fig. 3.3);
- Algorithm D* (Fig. 3.4-3.8).

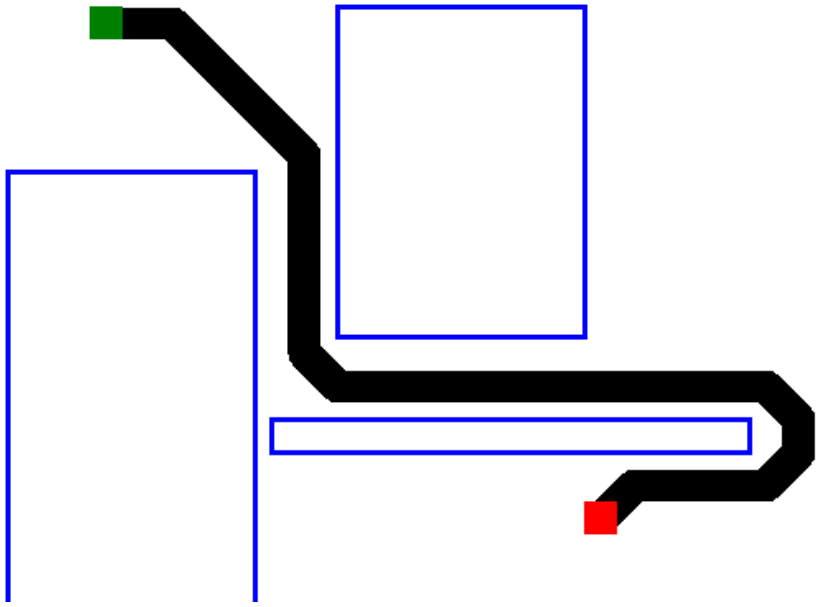

Figure 3.2 – Trajectory construction using Dijkstra's algorithm

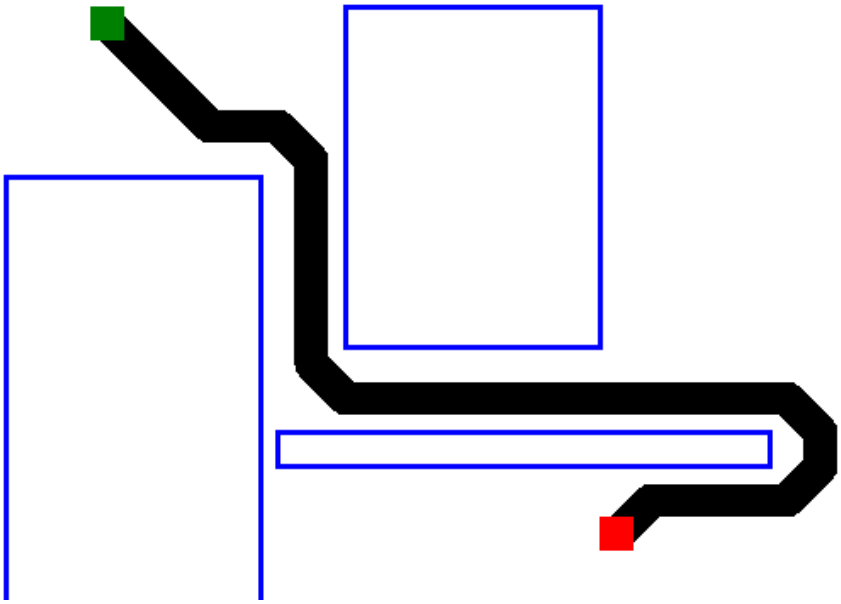

Figure 3.3 – Trajectory construction using the A* algorithm

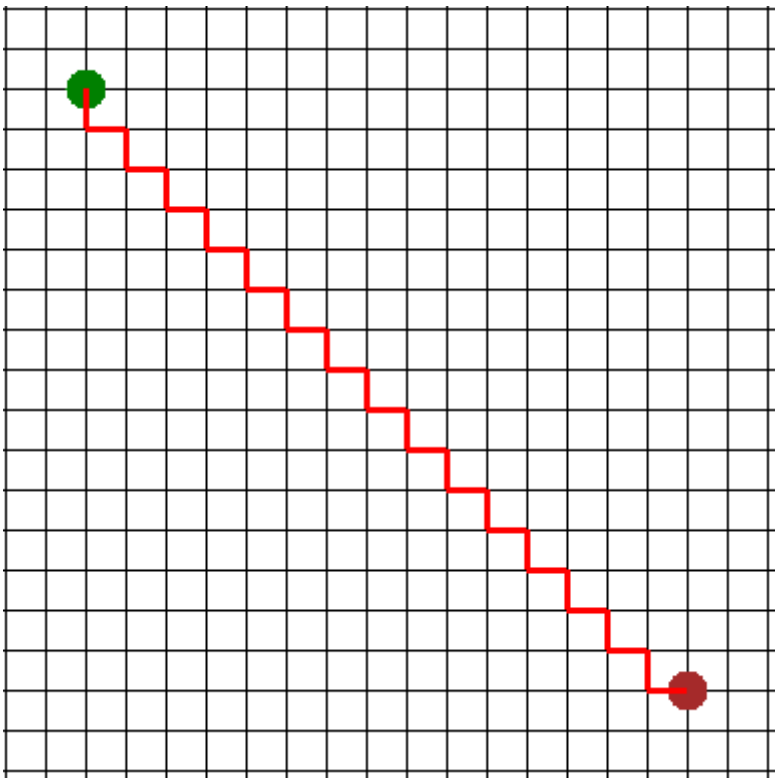

Figure 3.4 – The robot builds an initial trajectory in a non-deterministic environment

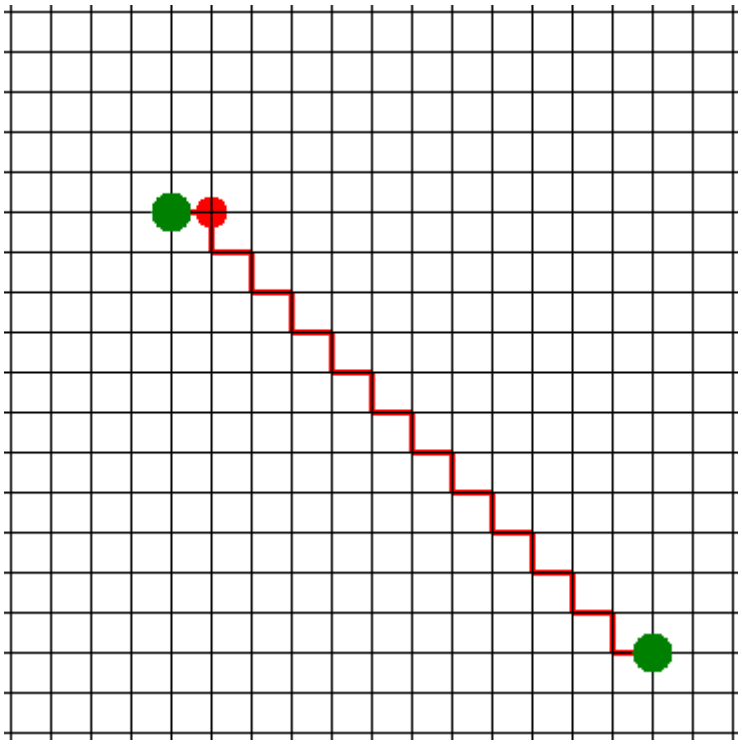

Figure 3.5 – While moving, the robot finds an obstacle in the next step

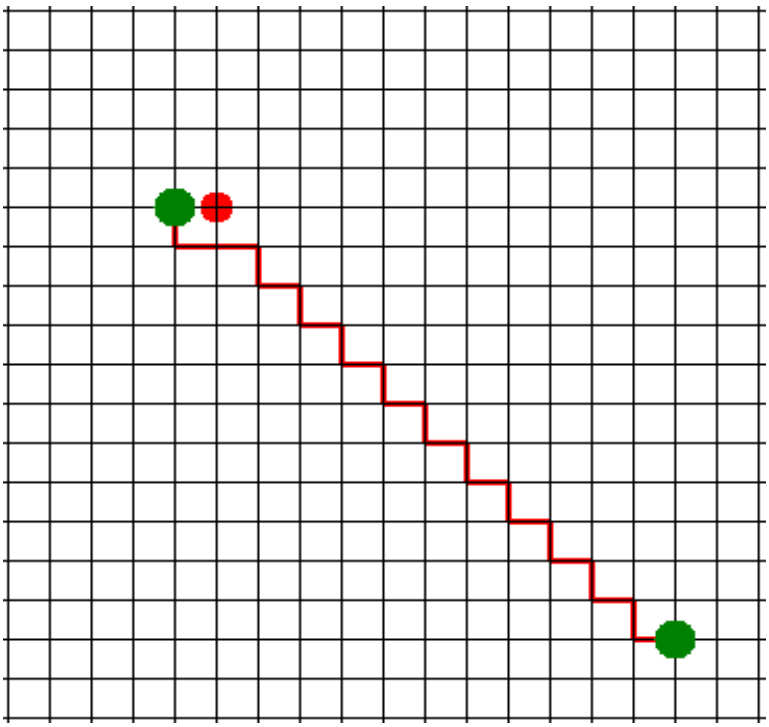

Figure 3.6 – The robot avoids an obstacle using a neighboring point from the history list

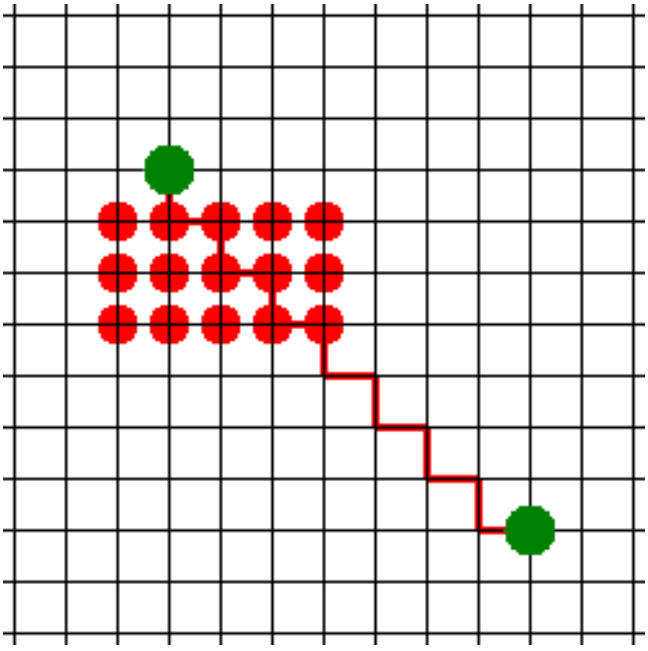

Figure 3.7 – While moving, the robot finds an obstacle in the next step

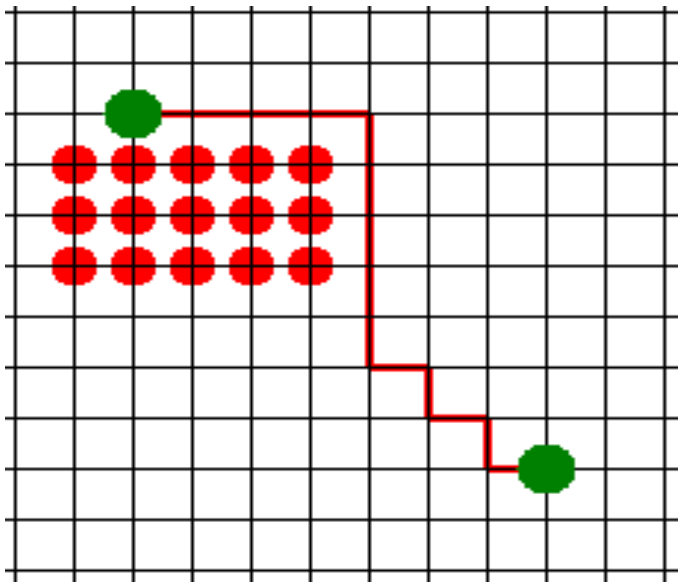

Figure 3.8 – The robot avoids an obstacle by constructing a new trajectory.

3.3. Comparison of accuracy and speed of algorithms

Based on the data obtained using software that implements these algorithms, we can conclude that the "A*" algorithm in most cases builds a trajectory around obstacles in a shorter time than Dijkstra's algorithm (fig. 3.9). The length of the resulting path is the same for the algorithms.

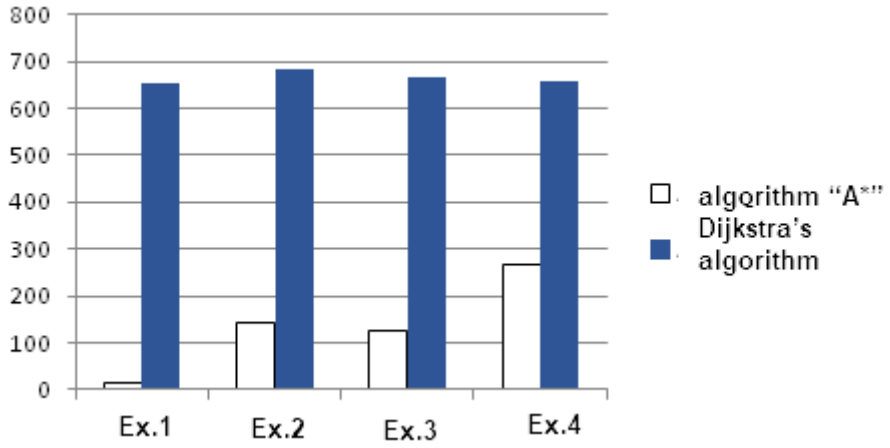

Figure 3.9 – Comparison of the running time of Dijkstra's algorithm and the A* algorithm.

Using the developed software that implements the operation of the D*, A* and LPA* algorithms in a non-deterministic environment, the operating time of these

algorithms was compared in different situations. The information obtained is shown in fig. 3.10 (time is in milliseconds).

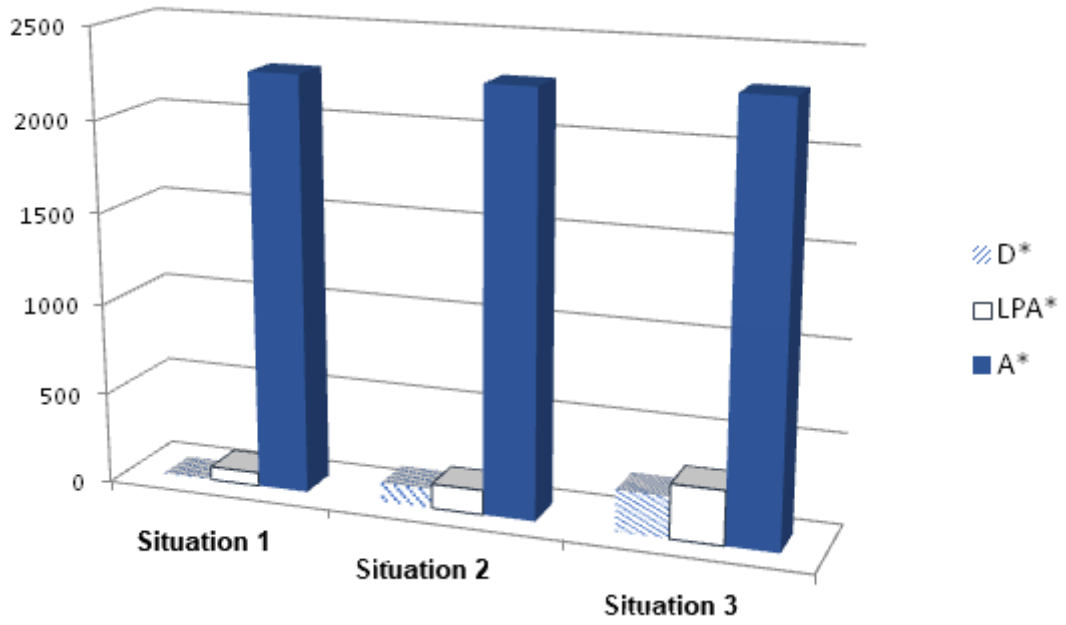

Figure 3.10 – Comparison of running time of algorithms in a non-deterministic environment.

CONCLUSION

- A navigation system is presented that allows a moving object to autonomously build an optimal path length to avoid obstacles from the starting point to the target point.
- The considered algorithms have been researched and implemented in software. Some results of the software are shown.
- A comparative analysis of the operating time of the algorithms was carried out.
- A navigation system is considered that allows you to apply the obtained theoretical information in practice.

REFERENCES

1. A* Pathfinding Algorithm. [Electronic resource]. – URL: https://www.baeldung.com/cs/a-star-algorithm

2. Alferov G. V., Malafeyev O. A. The robot control strategy in a domain with dynamical obstacles // Lecture Notes in Computer Science, 2003. №35. С. 4-23.

3. An architecture for a robot hierarchical system. [Electronic resource]. – URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nbsspecialpublication500-23.pdf

4. All You Need To Know About Obstacle Detection Sensor. [Electronic resource]. – URL: https://dreamvu.com/all-you-need-to-know-about-obstacle-detection-sensor/

5. Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction. [Electronic resource]. – URL: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

6. Dissanayake M.W.M.G., Newman P., Clark S., Durrant-Whyte H.F., Csorba M. A. Solution to the Simultaneous Localisation and Map Building (SLAM) Problem // Australian Centre for Field Robotics Department of Mechanical and Mechatronic Engineering The University of Sydney NSW, 2006. С 1-14.

7. Kalman filter: simple words about digital mathematics. [Electronic resource]. – URL: https://mp-lab.ru/filtr_kalmana_dlya_nachinayushchih/

8. Lee T-L., Wu C-J. Fuzzy motion planning of mobile robots in unknown environments // Journal of Intelligent and Robotic Systems, 2003. Vol. 37 (2), P. 177-191.

9. Montaner M. B., Ramirez-Serrano A. Fuzzy knowledge-based controller design for autonomous robot navigation//Expert Systems with Applications, 1998. Vol. 14 (1-2), P. 179-186.

10. Obstacle Avoidance Based on Stereo Vision Navigation System. [Electronic resource]. – URL: https://journal.umy.ac.id/index.php/jrc/article/viewFile/17977/8265

11. Optimal and efficient path planning for partially-known environments A Stentz - Robotics and Automation, 1994. P. 3310-3317.

12. Stereo vision. [Electronic resource]. – URL: https://link.springer.com/article/10.1007/s42452-020-2815-z

13. SLAM algorithm. [Electronic resource]. – URL: https://medium.com/@nahmed3536/the-types-of-slam-algorithms-356196937e3d