UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
INSTITUTO DE INGENIERÍA
MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

" OBJECT RECOGNITION BASED ON DISTRIBUTED SCANNING OF A MOBILE ROBOTIC GROUP USING 3D OPTICAL TECHNICAL VISION SYSTEM "

TESIS PARA OBTENER EL GRADO DE:
DOCTOR EN CIENCIAS

PRESENTA
MYKHAILO IVANOV

DIRECTOR
DR. OLEG SERGIYENKO

CODIRECTOR
DRA. VIRA TYRSA

Mexicali, B. C. ***Agosto 2020***

OBJECT RECOGNITION BASED ON DISTRIBUTED SCANNING OF A MOBILE ROBOTIC GROUP USING 3D OPTICAL TECHNICAL VISION SYSTEM

By

Mykhailo Ivanov

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF SCIENCE

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
Instituto De Ingeniería

August 2020

To the Engineering Institute of Autonomous University of Baja California:

The members of the Committee appointed to examine the dissertation of Mykhailo Ivanov find it satisfactory and recommend that it be accepted.

Oleg Sergyienko, Ph.D., Chair

Vira Tyrsa, Ph.D., Chair

Julio Cesar Rodríguez-Quinonez, Ph.D.

Wendy Flores-Fuentes, Ph.D.

Fabian Natanael Murrieta-Rico, Ph.D.

ACKNOWLEDGMENT

We want to extend our gratitude to the Instituto de Ingeniería de Universidad Autónoma de Baja California (UABC), and the Consejo Nacional de Ciencia y Tecnología (CONACYT) for providing the resources that made this research possible.

OBJECT RECOGNITION BASED ON DISTRIBUTED SCANNING

OF A MOBILE ROBOTIC GROUP

USING 3D OPTICAL TECHNICAL VISION SYSTEM

Abstract

by Mykhailo Ivanov, Ph.D.
Universidad Autónoma De Baja California
August 2020

Chair: Oleg Sergyienko

Robotic group collaboration in a densely cluttered terrain is one of the main problems in mobile robotics control. This thesis describes the basic set of tasks solved to model a robotic group behavior during the distributed search of an object (goal) with the parallel mapping. The navigation scheme uses the benefits of the authors' original technical vision system (TVS) based on dynamic triangulation principles. According to the TVS output data, fuzzy logic rules of resolution stabilization were implemented; this with the aim to improve the data exchange. Modified dynamic communication network model and implemented the propagation of information with a feedback method to improve the data exchange inside the robotic group. For forming the continuous and energy-saving trajectory authors are proposing to use the two-steps post-processing method of path planning with polygon approximation. The combination of our collective TVS scans fusion and modified dynamic data exchange network forming method with adjustment of the known path planning methods can improve the robotic motion planning and navigation in unknown cluttered terrain.

Table of context

List of tables

List of figures

Objectives

General objective

Create a methodology of optimized path planning improvement by data transfer integration for the mobile homogeneous robotic group, using the dynamic network graph configuration and 3D laser technical vision system, for the conditions of group of robots equipped with the original 3D laser technical vision system, moving in concert to visit all the points of interest to create a 3D point cloud of the continuous environment.

Specific objectives

- Development of behaviour model for robotic group.
- Analysis of technical vision approaches and system selection.
- Development of the data transferring model for the robotic swarm.
- Development of modeling system for data transferring analysis.
- Optimization of data transferring for small amount of robots.

- Analysis of existing path planning methods.
- Selection and adoption of the path planning method according to the specifications of the selected technical vision system.
- Improvement of the selected path planning method by adding the steps of post processing.
- Development of simulation software for method analysis.
- Evaluation of the data transferring influences on robotic group path planning.
- Analysis of robotic group effectiveness.

Problem definition

The dynamic network forming system based on the leader changing method is able to improved data exchange within the robotic group and to improve navigation by sharing previously detected obstacles by other robots within a group.

This methodology can be used to define the robotic behavior and to develop a navigation system that allows the robot to move freely in an environment with obstacles the path is dynamically recalculated, in response to the detected changes in surroundings. Since robots are equipped with a laser-based technical vision system it allows robots to detect the exact position of detected obstacle and create projectiles on the navigation map to define impassable areas, that will further improve navigation.

The presented navigation system consists of two-step post-processing path planning system based on A* algorithm, technical vision system for obstacle detection and improved by leader

changing method that allows robots to calculate optimized path inside of pre-known environment (detected buy robots in a group) or even avoid parts during navigation in case of a dead-end. These allow robots to navigate successfully and without collisions through an unknown environment and safely reach a robot's points of interest and goal.

Hypothesis

The dynamic network forming system based on leader changing method is able to improved data exchange within the robotic group and to distribute obtained map elements (obstacles detected by integrated technical vision system). The methodology can be implemented in a mobile robot to improve navigation of robotic group/swarm. By obtaining data from technical vision system, robot will update it's map and send data to other robots, then the navigation system will calculate the optimal trajectory using the merged map. This allows the mobile robot to navigate more efficiently in different types of environment without the need of revisiting the areas detected by other robots within the group.

Justification

The purpose of the thesis is to develop a data exchange methodology for the robotic group/swarm and to analyse it's effect on the navigation in the unknown environment. While other researches take into consideration only one problem at a time ([1], [2]) this thesis proposes the complex research of main aspect of mobile robot behavior based on the specified structure and constants.

This research was carried out entirely in the Optoelectronics and Measurements Laboratory

of the Engineering Institute of the Autonomous University of Baja California (UABC), Campus Mexicali, under the supervision of the Head of the Optoelectronics Laboratory and my thesis director, Dr. Oleg Sergiyenko.

Software design, experimentation, analysis, and presentation of results were made in the form of journal articles, publications and reports in international conferences, and copyrights.

The type of research that was carried out can be defined as an applied research project, which promotes both theoretical and practical applications, since the methodology used (leader changing method) to dynamically created data exchange network, it is applied in the developed modeling software. This methodology in turn is implemented in the behavior of the mobile robotic group presented in the research.

Chapter One

Introduction

Swarm Robotics is a research field that studies how systems composed of multiple autonomous agents (robots) can be used to accomplish collective tasks, where the tasks cannot be accomplished by each individual robot alone, or are carried out more effectively by the robots as a group. Dudek et al. [3] identified the following categories for tasks executable by robots: tasks that are inherently single-agent, tasks that may benefit from the use of multiple agents, tasks that are traditionally multi-agent, and tasks that require multiple agents. The swarm robotics discipline focuses on the last three categories, and past works have demonstrated in many application domains that using multiple agents to solve a task in a distributed manner allows working with significantly less complex behavior of agents at the individual level.

Swarm robotics, also known as Multi-Robot Systems (MRS) have been proposed in the last decade in a variety of settings and frameworks, pursuing different research goals, and successfully applied in many application domains. Special attention has been given to MRS developed to operate in dynamic environments, where uncertainty and unforeseen changes can happen due to the presence of robots and other agents that are external to the MRS itself.

An MRS can be referred to as a group of robots working in the same surroundings. Nevertheless, robotic systems can be presented in different ways one as a set of sensors that are used for data acquisition and processing to human-like as a more complex part of machine family, those have the ability to interact with surroundings in different ways. Also, to define a level of robot autonomy is another hard task, cause some of them have to interact with an environment and others just have to be able to accomplish a simple set of tasks programmed by an operator. The mobile platform discussed in the current thesis is a fairly complex platform equipped with the technical vision system that is able to execute tasks of detection, navigation and communication. In the end, three main aspects can be considered to describe the subset of MRS:

- The rationale for the design of the MRS;
- The basic functionalities and technologies (both hardware and software) used in the MRS development;
- The tasks that the robots should perform and the intended application domains.

The expression “swarm intelligence”, which is now widely used in the field of swarm robotics, refers to the superior capabilities of a swarm of agents compared to its single individuals. The local events triggered by swarm members during execution of a task translate into a global behavior which often transcends the individual capabilities, to the point that many collective tasks can be successfully done by robots that are not explicitly programmed to execute those tasks: the global, macroscopic dynamics is said to emerge from interactions of swarm members between each other and with the environment.

Nowadays, the use of swarm intelligence systems can be found in research articles of civil

related tasks, such as autonomous cars, unmanned aerial vehicles (UAV) etc. Driven by scientific research, swarm intelligence systems are prevalent and specialized for multipurpose tasks that require a group of robots to cover different types of unknown environments (cluttered or rugged terrains, indoor premises, etc.). In articles such robotic groups are referred as swarm robotics [4] [5] [6].

Swarm robotics is a promising technology that can be deeply involved in daily human life. As an example smart autonomous vehicles can be found in the use of Google [7], Tesla, Uber, etc. Right now they are not so affordable, however in future they will be a huge part of social life from smart cities and campuses with autonomous personal mobility vehicles [8] to simple use in smart buildings as janitors.

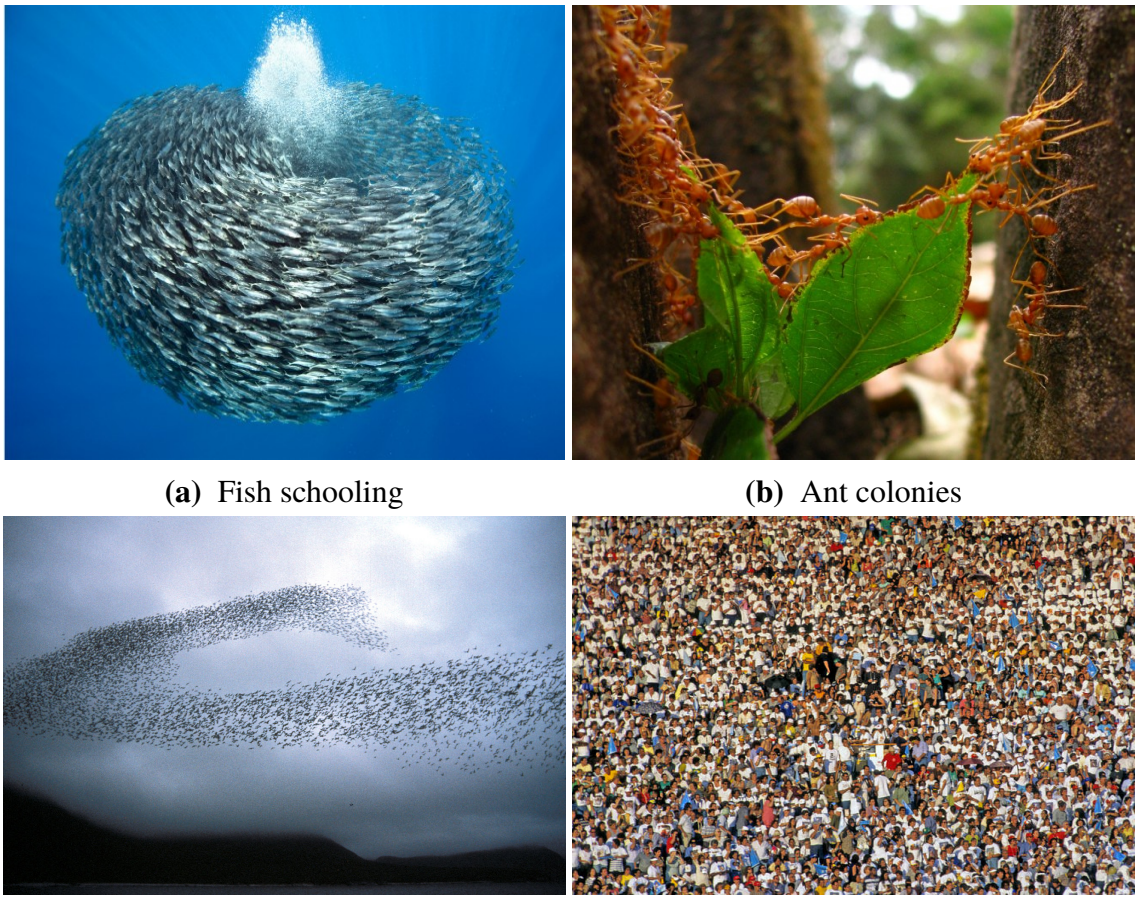
In these matters exists two principal tasks – navigation and communication. The first is used for obstacle avoidance and moving to the target location and second is to give the robot a tool to "talk". Communication helps a swarm to achieve more complete and structured information about the surroundings and improve their navigation as individuals.

Further sections will consider a solution for such problems and review the influence of data exchange on the path planning in terms of unknown surroundings.

1.1 Inspirational models for robotic behavior

Development of distributed artificial intelligence [9] (republished [10]) is a subject of many complex researches related to the multi-agent systems [11]. The behavior models used in these systems take their origin in adoption of social animal group activities. In bacteria colonies, fish schooling, animal herding, ants and etc. (Fig. 1.1) individuals have primitive abilities, but while in-group they

become a complex organization with improved surrounding interactions, signaling communication and data transmission [12]. Such natural swarms are based on a simple set of rules used by each individual. As a result with common efforts the homogeneous groups can complete complex tasks. Transferring of this behavior has created the principles of swarm intelligence ([13], [14]).



(a) Fish schooling **(b)** Ant colonies

(c) Bird crowds **(d)** Human beings

Figure 1.1 Natural swarms

Primates usually have complex collaboration inside the group, they can have different types of social interactions [15], recognize their relatives [16] and some of the species can use some of the human language aspects.

Bacteria and their colonies are usually functioning as biofilms. They have an ability to cell communication [17] and use the benefits of task distribution, collective defense, etc. The bacterial colony has higher resistance to antibacterial agents than individuals of the same type of bacteria [18].

Bird crowds during the migration, they can locate their destination point according to inner sensing, landmarks, etc. [19].

Ant and Bee colonies communication in such type of colonies is based mostly on pheromones [20]. Ants are able to path planning by leaving a pheromone trail, where in case of optimal rout more and more ants are using it [21]. According to suggestion in [22] ants implement role distribution based on their previous performance during foraging.

Locusts while increasing amount of insects in-group, they convert the type of their movement, from chaotic to aligned, this with the capacity for a quick transference of directional information [23].

Fish schools each fish by analyzing the neighbors' movement can avoid collision while swimming in phalanxes[24]. Fish schools are better in foraging [25] and predator evasion [26].

Human beings Dyer et al. in [27] have shown that in the group of humans can occur leadership without any verbal communication or other obvious types of signals. Such behaviorism shows the hierarchical structure with role distribution. In case of growing down of the population collaboration within the group becomes more complex and each role becomes more important. A swarm of individuals in this case can solve complex tasks easily while an individual cannot.

1.2 Tasks of robotic swarm implementation

According to the this brief review of natural swarms can be found a list of tasks that robotic swarm can be capable of.

The next list presents some examples according to their recursivity in descending order.

Navigation is a task where robot needs to find location of an object using his sensing capabilities and the help of other robots. In this task an individual robot (not a group) must reach an object of interest (target).

Foraging is another studied scenario in swarm robotics where a group needs to find a food source. It takes origin in the behavior of ants in colonies. Evolving robots [28] is a perfect example of this behavior implementation.

Multi-foraging is a more complex type of foraging task. A robotic group needs to find and collect different types of objects and after bringing them to a specific place for this object type. This task has an application in warehouses, rescue missions, hazardous terrain cleanups, etc.

Odor source localization is aimed to solve the problem of odor source search. In [29], was described as a project where robots are using binary odor sensor.

Collective decision-making is a study of many researches like [30] (authors proposed decision making based on majority rule model), [31] (charges of preferred labors), [32] (site selection). Collective decision-making is applied to swarm robotics during the flocking, path formation, clustering etc.

Object clustering and sorting are similar to foraging tasks, but in this case, the goal is to find objects in the environment and to place objects near to each other.

Object assembly is a task related to construction problems. The task has a relationship with clustering, but in this case, it is focused on relationships and physical connections between objects (robots have to create an object of predefined shape). In [33] is described an example of wall construction by robots, in [34] robots were trying to figure out if the block they use can be attached or not with help of local communication, in [35] were modeled a process of building using blocks with attached LEDs.

Collaborative manipulation refers to a task where robotic swarms have to interact with objects. For achieving this task and avoiding centralized coordination (control) robots can use simple behavior rules. Ant colonies can be used as an example.

Self-assembly is a task of local interaction within the group, where robots need to establish a physical connection to each other. The examples can be found in [36] (swarm-bots project), authors of [37] used s-bots with the ability to decide who will use its grip to attach to other s-bot. Self-assembly also takes its place during the hole avoidance [38] and navigation in hazardous terrains [39].

Human-swarm interaction task is devoted to increase the performance of robotic swarms. The task can be described in a next way: if human operation have an information that can help robotic swarm to achieve their goal he can share it, but it cannot be interpreted as a direct controlling. In [40] authors proposed a method whereby starting direct control over one robot can influence a group by changing robot behavior. In [41] were used a similar method but with leader system implementation. Kolling et al. [42] proposed methods of robot behavior switch and manipulation with the environment to force the robot to change its behavior.

Deployment is a scenario where a group of robots must autonomously enter the environment. The task is useful for the unknown terrain mapping.

Path formation is a process that refers to a collective movement formation from point to point while minimizing the time to reach the destination. This task usually occurs during the foraging and chain formations.

Coordinated motion is used during the modular robotic structure to achieve coordinated movement in a common direction.

Flocking is a task of the swarm to "stay together". It is based on local interaction within the group. Robots' sensing systems (vision, laser range finding, sonar, infrared or tactile sensors) and communication abilities are used to keep the group as a compact formation. Such behaviorism is adopted from birds, fish schools, etc.

Morphogenesis exists as an extension of self-assembly. In the case of morphogenesis, robots are tasked to create a specific shape. For example, [43] s-bots attached to a structure used LEDs for communication to tell how others should be attached.

Aggregation is the task of grouping individual robots in a dedicated place. The behavior model is based according to animal species observation.

Task allocation is a process of labor division within the group. This ability helps to increase the efficiency of work done by a swarm.

Group size estimation is a "sub-task" for different applications of swarm robotic, used to coordinate movement, self-assembly, morphogenesis, etc. Authors in [44] describe the method using the propagation of information through the group in case of impossible direct communication within

the group.

A more detailed review of the mentioned tasks with their solutions can be found in [45].

1.3 Swarm robotic projects

During the last decade, the growth of small and mobile computing devices' capabilities increased the interest in swarm robotics research topics. Yet still many of the swarm robotics projects are in the development or even modeling stage [46]. Earlier in the 80s existed other attempts to create robotic groups like SWARMS [47] and ACTRESS [48]. This section will review some of the existing projects and their design.

Pheromone robotics project (Fig. 1.2a) was initiated in 2000 [49]. The idea of the project is to use a large number of small robots for different tasks by achieving a swarm behaviorism [50]. In robots was implemented the pheromone idea, by using beacons and sensors mounted on robots.

iRobot swarm project (Fig. 1.2b) was made by Massachusetts Institute of Technology (MIT). Swarm includes more than 100 cooperating robots [51] [52].The main idea is to create the solution for graceful degradation of the swarm.

E-puck education robots is a group of small robots (Fig. 1.2c) that were made for educational purposes like programming, human-machine interaction, signal, and image processing. They are cheap, have a simple structure, and have possibilities to use extensions [53].

Kobot project (Fig. 1.2d) consists of mobile robotic platforms equipped with IR range finder system for obstacle detection [54].

Kilobot project (Fig. 1.2e) is aimed to create collective behaviorism with hundreds or more individuals in swarm [55]. Robots are easy to assemble and have capabilities for some simple operations like charging, moving, updating programs, etc.

I-Swarm (Fig. 1.2f) came from the University of Karlsruhe in Germany [56]. Swarm consists of micro-robots with the abilities of "collective thinking" and to recognize its kin.

Multi Robot Systems (Fig. 1.2g) [57] initially were developed in University of Alberta in Edmonton, USA, studies robots collective behavior. This institution has several robot systems (Multi-Robot Systems (MRS)) in development. The project is devoted to problems of collective decision-making.

Project SwarmBot (Fig. 1.2h) was made by iRobot company [58]. Swarm uses small robots that can work together to carry out certain actions. It is expected that the SwarmBot robots can join a group of up to ten thousand and perform tasks such as a search for mines, research of unknown territory (including on other planets), detection of harmful substances, etc.

Project Centibots (Fig. 1.2i) uses small robots that can work in-group and autonomously [59], they are not using a centralized management system. Their goal is to map the enclosed space and perform some tasks. Robots are using the roles distribution system based on interaction and depending on the circumstances.

Project Swarmanoid (Fig. 1.2j) studies the behavior of inhomogeneous groups of robots [60]. The considered task was in which a team of wheeled robots, a flying robotic spy and the handling robot jointly found the object (book), and manipulated it.

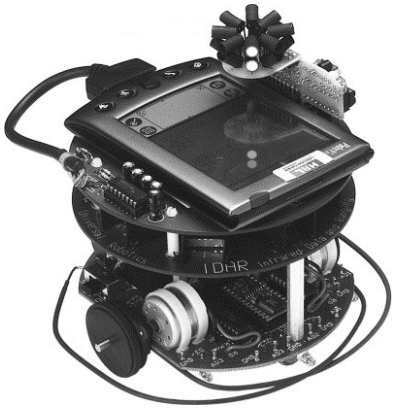
Evolving robots (Fig. 1.2k) were created in the Swiss Laboratory of Intelligent Systems (Polytechnic School, Lausanne) during the "evolution" of robots studies [28]. Robots were evolving

gene that determines behavior. A group of 10 robots competed for food. The challenge was to find robots "food source", which is a luminous ring on one end of the arena. Robots can "communicate" with each other by light signals. The evolution of robots in experiments sometimes leads that even the robots were taught to deceive opponents, letting the "wrong" light, being near the trough.

Robots scouts (Fig. 1.2l) were designed for intelligence [61]. The project was developed in the distribution centers of robotics, University of Minnesota, USA., the robot has a very high quality from a technical point of view. The robot can work in a team. Its design allows you to "shoot" using a device resembling an automatic grenade launcher. The robot is designed to help the police and rescue services in carrying out dangerous operations.

According to the mentioned projects, [12] and [62] the next advantages of swarm robotics can be described:

- Parallel processing – swarm can perform various tasks simultaneously (each of swarm can perform its scheduled task), this would save time for achieving a common goal.
- Scalable group – including a new individual to a swarm can be performed without any modification of the software or hardware.
- Tasks enlargements – swarm robotics systems can solve the tasks that are impossible for individuals.
- Fault tolerance (graceful degradation) – swarm can continue performing its tasks even what the part of the swarm is unable to work. Useful during the tasks in a hazardous environment.
- Distributed sensing and action – distributed on the terrain swarm robotics system can be used as a sensing network for data accumulation and action performing system.



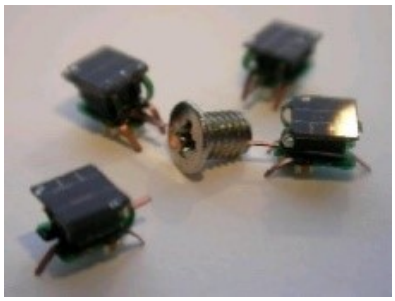
(a) Pheromone robot

(b) iRobot swarm project

(c) E-puck education robot

(d) Kobot project

(e) Kilobot project



(f) I-Swarm project

(g) Multi Robot Systems

(h) Project SwarmBot

(i) Project Centibots

(j) Project Swarmanoid

(k) Evolving robots

(l) Robot scout

Figure 1.2 Swarm robotics projects

However, can be allocated a several specific problems that need to be resolved. Among them are:

- Unpredictable environmental dynamics;
- Imperfect and inconsistent knowledge of the environment;
- Variety of options to achieve the goal, the team structures, roles, etc.;
- A complex behavior pattern of teamwork;
- Problems related to the territorial distribution of swarm and its localization;
- Communication problems or data exchange (network architecture, protocols, etc.);
- Data loss and storage redundancy.

So next section will cover the main questions of robotic group behaviour: a vision system for robot as a part of simultaneous localization and mapping (SLAM), route planning, data transferring (communication) within the group and complex modelling with analysis.

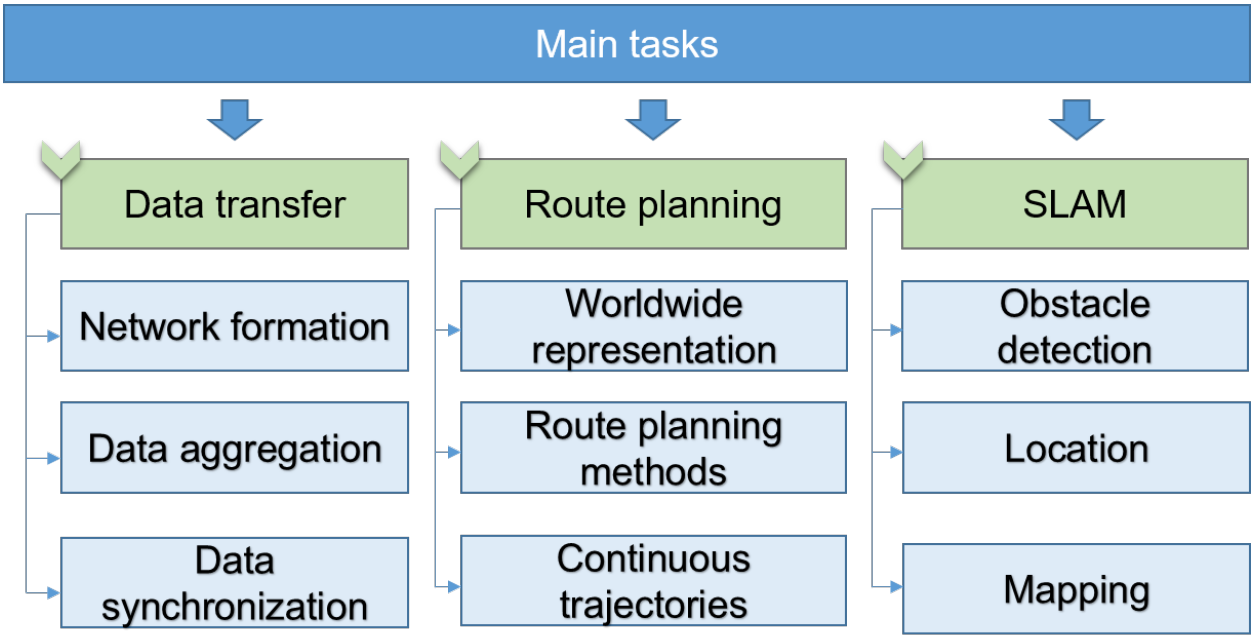


Figure 1.3 Main tasks

Chapter Two

Technical vision system

Each mathematical / computing unit in the conventional algorithm of autonomous robot (Fig. 2.1) during its work is based on data obtained from its sensors.

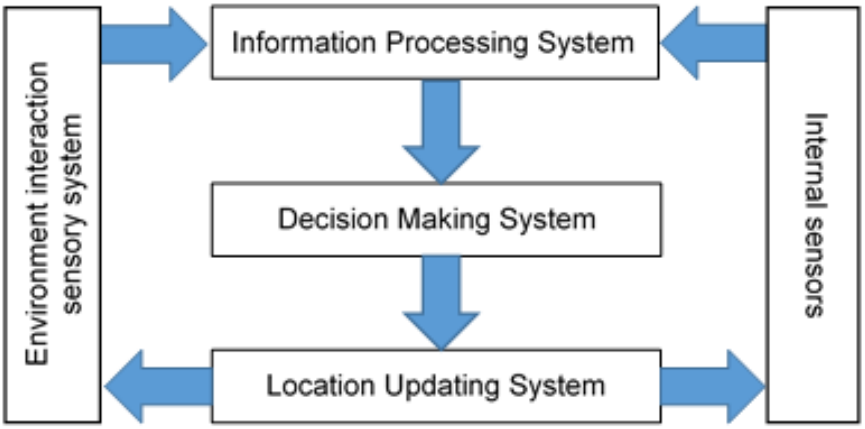


Figure 2.1 Autonomous robotic system

For the task of navigation are generally used sonars as a basic solution, laser rangefinders as a more accurate alternative for sonars, LIDARs (can return the detailed surroundings, but depends on its form factor), camera-based robots ([63], [64]) and drones [65] or more expensive equipment like ToF (Time-of-Flight) cameras [66]. For some tasks, it is possible to use only the inertial navigation system [1].

Plenty of the researches and solutions in the field of obstacle detection and navigation, as mentioned before, are based on cameras and laser systems. As an advance will be reviewed some works.

In [67] authors presenting a lightweight, inexpensive insect-based stereo-vision system. They used two cameras placed very similar to honey bee eyes and received a field-of-view around 280°by 150°. In [68] authors are using a camera vision for real-time obstacle avoidance with biped robots. Article [69] describes the obstacle avoidance for a pocket drone based on data from the stereo camera. In [70] solution uses Arduino with pixy camera by a wheeled robot for the line tracking task and obstacle avoidance. The solution mentioned in [71] can be applied for autonomous cars and shows a method of vehicle detection system design based on stereo vision sensors.

Article [72] describes the benefits of the use of long-wavelength infrared stereo vision and 3D-LIDAR combination in case of fire environments. Another publication [73] presenting MEMS-based LIDAR systems for use in an autonomous vehicle. In [74] were a proposed method of real-time LIDAR odometry and mapping and its application on drones and cars.

Also, such systems can be used for similar tasks in other areas. Authors of [75] used an industrial robotic hand with a mounted 3D camera-based vision system for object scanning, similar to them in [76] are using industrial robots with a camera to track motions of the second robot, in [77] position of the robot is controlled by tracking marker with LEDs and in [78] were proposed human posture tracking and classification using cameras stereo vision and 3D model matching. Also can be mentioned [79] Stereo vision-based automation for a bin-picking solution.

As can be seen, laser-based systems are more suitable for automobile navigation purposes. It is explained by their surrounding representation principles that helps to avoid long post-processing

like in cameras.

However, our robotic group highly probable moves in low light conditions with a large number of obstacles, where the aforementioned vision systems may not always give the correct results during post-processing. Therefore, working in such difficult conditions, the authors' novel real-time vision system TVS can satisfy with its accuracy and data representation.

The next section will provide a detailed description of the real-time vision system used in the research.

2.1 Vision systems

2.1.1 Traditional Vision-Based Collision Detection Methods

Vision-based collision detection is widely used in robotics [80], [81]. For example, Saha et al. [82] proposed a monocular obstacle detection and avoidance method for UAV. They used a mathematical model to estimate the relative distance from the UAV's camera to an obstacle by detecting the feature points in the UAV's field of view, which is not an on-board system. Yaghmaie et al. [83] proposed a novel method for robots to navigate in dynamic environments called escaping algorithm which is based on force field method which belongs to the family of simultaneous localization and mapping. In their algorithm, the movement of dynamic obstacles is predicted by the Kalman filter for collision detection combined with a potential field approach. Traditional visual-based collision detection methods need to process the massive volume of images in real-time or need a real-world model created in advance, which is either difficult to be completed on-board for a micro-robot with limited resources or hardly able to cope with dynamic environments.

2.1.2 Bio-Inspired Collision Detection Methods

There are additionally a few bio-inspired collision avoidance and route planning techniques, a large portion of which depend on elementary motion detector (EMD), for instance, Zhang et al. [82], Badia et al. [83], and Franceschini et al. [84]. EMD-based techniques could be hard to apply because of its intrinsic character -— the presentation is limited inside certain visual paces. The lobula giant movement locator (LGMD) based techniques can adapt to a large portion of the future collisions, without paying attention on the visual speed. Blanchard et at. [85] was the first to bring LGMD-based neuron systems into robots for constant impact recognition and tried it with Khepera I robots. Badia et al. [86] proposed one type of LGMD based impact recognition model and tried it on a "Strider" robot with a remote camera to catch and transmit pictures to PC for handling. Silva et al. [87] proposed another LGMD model which joined two works from [88] and [89] for increasingly strong collision detection, which concentrated more on modeling rather than embedded systems development. There has been put forth an attempt on actualizing bio-inspired techniques in enormous scope joining chips like field-programmable cluster (FPGA), Meng et al. [89] added extra cell to recognize the development top to bottom, Harrison [90] proposed a simple integrated circuit for collision detection dependent on EMD, and Okuno and Yagi [91] actualized blended analog/digital incorporated circuits with FPGA. These attempts are not reasonable for smaller scale robots, either due to the enormous size or the powerful utilization of the FPGA circuits.

2.1.3 ToF camera principle

Profundity estimations depend on the notable time-of-flight (ToF) standard [92] by utilizing either pulsed or ceaseless wave (CW) modulation. Some of ToF cameras are using both of the modulation

types. To demonstrate more deep principles will be reviewed CW-based ToF cameras. Sensors based on discrete pulse modulation measure the time that the optical pulse trips to calculate depth, while sensors based on lock-in measure the phase difference between the transmitted and received signals.The system emits near-infrared light (NIR) through a LED and then reflects it back to the sensor. The amount of light reflected by the scene is sampled by each pixel on the sensor at even intervals in each cycle (m_0, m_1, m_2, and m_3) so that its phase (Eq.2.1), offset (Eq.2.2) and amplitude (Eq.2.3) can be measured in parallel.

$$\phi = arctan(\frac{m_3 - m_1}{m_0 - m_2}) \tag{2.1}$$

$$B = \frac{m_0 + m_1 + m_2 + m_3}{4} \tag{2.2}$$

$$A = \frac{\sqrt{(m_0 + m_1)^2 + (m_2 + m_3)^2}}{2} \tag{2.3}$$

This phase demodulation procedure is ordinarily alluded to as "four barrel" examining, which can easily calculate the target depth (Eq.2.4).

$$D = L\frac{\phi}{2\pi} \tag{2.4}$$

To predict the quality of the measurement used the intensity (B) and amplitude (A). The modulation frequency (f_m) of the emitted light determines the unambiguous distance range of the sensor (Eq.2.5) where c is the speed of light in vacuum.

$$L = \frac{c}{2f_m} \tag{2.5}$$

2.1.4 Camera-based Systems

To these group fits several techniques [92], for example, profundity fromfocus/defocus/obscure, profundity-from-movement, profundity-from-shape, sound system and organized light triangulation strategies [93]. Profundity-based techniques are using focus variation, motion estimation, and shape change determination. Those methods produce ambiguities and singularities, and frequently require utilizing numerous pictures to solve the problem, which infers extra time, space and computational expenses. On the other hand, profundity data acquired with ToF cameras has high accuracy using single frame of data (image). Triangulation strategies can be isolated into passive (stereo vision) and active (structured light methods techniques).

Passive triangulation methods require a par of cameras isolated by a standard that determines a restricted working profundity range (the higher profundity, the bigger base is the required). These methods have to solve the next issue: determine corresponding points from cameras pictures that are a references of a similar 3D point. This is a computationally costly and complex issue, as stereo vision systems frameworks can't comparing points in homogeneous surroundings [94]. ToF cameras and Laser-based systems does not have this problem cause of they have these data instantaneously.

Active triangulation methods Oppositely to the former techniques, active triangulation ones require just a single camera and an organized light emitter that gives minimum of one pattern line. Comparing with with ToF cameras and laser-based vision systems current method has some disadvantages: incomplete/missing profundity estimations, a need of exceptional power supply and concentrated light, checking of the light through the scene, and an exceptionally controlled light conditions that prompts a major limitation in local or open air application of autonomy vehicles.

2.2 Historical background

All the solutions are based on the novel author's technical vision system (TVS)[95] that uses a dynamic triangulation principle [96]. In [97] proposed method for improving resolution of 3D TVS and its implementation for surface recognition. This approach to obstacle recognition was implemented for the single robot navigation in work [98]. The concepts of data transferring within robotic group firstly presented in [99]

Further advances of author's 3D TVS found its use and development in works [100] and [101]. Here system received its internal changes and appliance as a machine vision system for UAV.

Despite all of researches and results presented in this section still exist a common problem: all of them are dedicated to one problem at a time. That is why the presented research is aimed to a joint solution taking into consideration problems of machine vision, path planning and data transferring using the mentioned earlier 3D TVS.

The herein presented real-time technical vision system (TVS) is appropriated novel tool for optimization of the considered task of the robotic group collective behavior improvement. The advantages of our system when compared with other methods of 2D/3D laser sensors, possessed in the following points:

- Original TVS possess the property of natural physical filter [96], [102] of redundant information about robot surrounding, adjusting the quantity of the scanned points within FOV according to the requirements of current practical application.

- This TVS in comparison to other methods of 2D/3D laser sensors has significantly wider FOV [96], [103] due to its original patented rotational sensory part. This circumstance permits two advantages: better simultaneous capture of the detailed data about surrounding [104], and wider

possibility to vary the scenario sectorization between n participants equipped with identical TVS [99], [105].

- Present TVS, as shown in [106] represents the data naturally in the same Cartesian Coordinate System where operating the robot, which delete the necessity of any post-processing time, and in the same notation system as robot state matrix, which simplify any additional transformation within scenario.

- The multiple experimental results [107], [106], [97], [108] shows that obtained scanning data are very appropriated for neuronal networks application in scanned surface rectification, and after this are significantly better that any surface reconstruction from stereo vision systems.

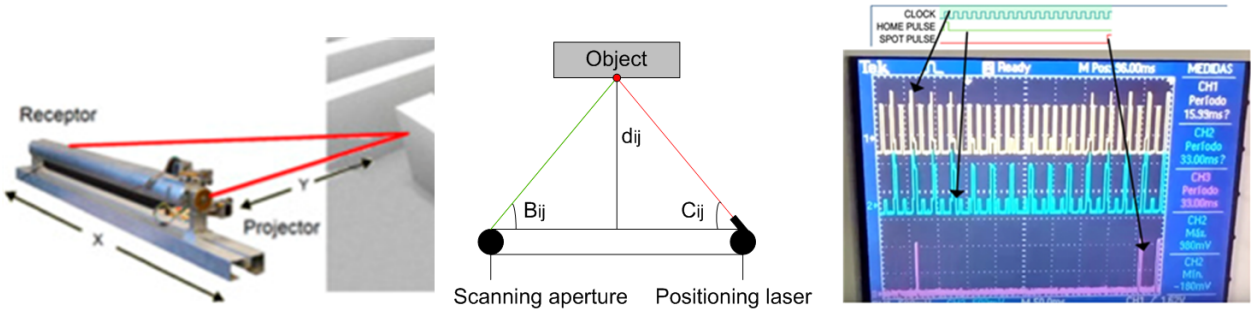
- The performance convenience in this case is given including the frequency of operation. According to [109] our TVS possess the feature of variable scanning step, which permits on the different stages of swarm mission adjust the fastness/accuracy to the current challenge; or even provide the variable scanning velocity based on enhanced control algorithms of DC motor of laser ray positioner, introduced in detail in [100] and [101].

2.3 Structure and working principles

According to current task robotic group move in low light conditions with a large number of obstacles, the aforementioned vision systems may not always give the correct results during post processing. Therefore, when working in such difficult conditions, authors TVS [96] can satisfy with its accuracy and data representation. 3D TVS (Fig. 2.2a) is able to work in complete darkness, and can obtain the real 3D coordinates of any point highlighted by laser ray on real, not imaginable, surfaces. The theory is based on a dynamic triangulation method. The main components of the

TVS are the positioning laser (PL) and the scanning aperture (SA) (Fig. 2.2b).

System works in the following way: the laser emits its beam toward a fixed 45°mirror than makes orthogonal redirection of the beam into a rotating 45°mirror driven by a stepper motor. For the guaranteed positioning of the laser direction PL is driven by a stepper motor. SA receive the reflected laser rays, this indicates that system had detected an obstacle. However, stepper motor has one weak point: on the long distances of scanning dead-zones between two adjacent points of scanning are provoked. Solution of the problem can be found in [110] and [111].



(a) Technical Vision System **(b)** Dynamic triangulation method **(c)** Principle of codes N forming in laser scanning TVS

Figure 2.2 Technical Vision System

Dynamic triangulation ([102], [103]) consists of detection of laser instantly highlighted point coordinates based on two detected angles B_{ij} and C_{ij} and fixed distance between projector and receptor. Here ij means the number of horizontal and vertical scanning steps consequently. In such triangle (Fig. 2.2b), if three parameters are known, it makes possible to calculate all others. Angle B_{ij} is calculated as simple ratio of two counters codes: number of clock pulses between two home pulses and in interval "home pulse – spot pulse" (Fig. 2.2c) (Eq. 2.6).

$$B_{ij} = \frac{2\pi N_A}{N_{2\pi}} \tag{2.6}$$

where N_A is the number of reference pulses when laser rays are detected by the stop sensor and

$N_{2\pi}$ is the number of reference pulses when the 45°mirror completes a 360°turn detected by the zero sensor. To calculate x, y and z coordinates the next equations are used (Eq. 2.7-2.10):

$$x_{ij} = a\frac{\sin B_{ij} \cdot \sin C_{ij} \cdot \cos \sum_{j=1}^{j} \beta_j}{\sin[180^o - (B_{ij} + C_{ij})]} \tag{2.7}$$

$$y_{ij} = a(\frac{1}{2} - \frac{\sin B_{ij} \cdot \sin C_{ij}}{\sin[180^o - (B_{ij} + C_{ij})]}) \; at \; B_{ij} \leq 90^o \tag{2.8}$$

$$y_{ij} = -a(\frac{1}{2} + \frac{\sin B_{ij} \cdot \sin C_{ij}}{\sin[180^o - (B_{ij} + C_{ij})]}) \; at \; B_{ij} \geq 90^o \tag{2.9}$$

$$z_{ij} = a\frac{\sin B_{ij} \cdot \sin C_{ij} \cdot \cos \sum_{j=1}^{j} \beta_j}{\sin[180^o - (B_{ij} + C_{ij})]} \tag{2.10}$$

2.3.1 Surface Recognition Improvement

The Levenberg-Marquardt algorithm is designed to optimize the parameters of nonlinear regression models. It is assumed that the root-mean-square (RMS) error of the model is used as the optimization criterion on the training set. The algorithm consists of the given parameters initial values sequential approximation to the desired local optimum.

Regression sampling is given – many pairs $D = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ of free variable $\mathbf{x} \in \mathbb{R}^M$ and dependent variable $y \in \mathbb{R}$. Regression model set – function $f(\mathbf{w}, \mathbf{x}_n)$ continuously differentiated in the field $W \times X$

It is required to find a value of the parameter vector $\mathbf{w}$, which would deliver the local minimum of the error function

$$E_D = \sum_{n=1}^{N} \big(y_n - f(\mathbf{w}, \mathbf{x}_n)\big)^2. \tag{2.11}$$

Before the algorithm starts, the initial parameter vector $\mathbf{w}$ is set. At each iteration step, this vector is replaced by a vector $\mathbf{w} + \Delta\mathbf{w}$. To estimate the increment of $\Delta\mathbf{w}$ is used the linear approximation of the function

$$f(\mathbf{w} + \Delta\mathbf{w}, \mathbf{x}) - f(\mathbf{w}, \mathbf{x}) \approx J\Delta\mathbf{w}, \tag{2.12}$$

where J is the Jacobian functions $f(\mathbf{w}, \mathbf{x}_n)$ at the point $\mathbf{w}$. Matrix J has dimensions $N \times R$ and can be visualized as

$$J = \begin{bmatrix} \frac{\partial f(\mathbf{w},\mathbf{x}_1)}{\partial w_1} & \cdots & \frac{\partial f(\mathbf{w},\mathbf{x}_1)}{\partial w_R} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{w},\mathbf{x}_N)}{\partial w_1} & \cdots & \frac{\partial f(\mathbf{w},\mathbf{x}_N)}{\partial w_R} \end{bmatrix} \tag{2.13}$$

here $\mathbf{w} = [w_1, \ldots, w_R]^T$ is the parameter vector.

Increment $\Delta\mathbf{w}$ at the point $\mathbf{w}$, delivering a minimum of E_D is zero. Therefore, to find the subsequent increment value $\Delta\mathbf{w}$ equating to zero the vector of partial derivatives E_D by $\mathbf{w}$. for this Eq.2.11 presented in the form

$$E_D = \|\mathbf{y} - \mathbf{f}(\mathbf{w} + \Delta\mathbf{w})\|^2, \tag{2.14}$$

where $\mathbf{y} = [y_1, \ldots, y_N]^T$ and $\mathbf{f}(\mathbf{w} + \Delta\mathbf{w}) = [f(\mathbf{w} + \Delta\mathbf{w}, \mathbf{x}_1), \ldots, f(\mathbf{w} + \Delta\mathbf{w}, \mathbf{x}_N))]^T$.

Transforming this expression

$$\|\mathbf{y}-\mathbf{f}(\mathbf{w}+\Delta\mathbf{w})\|^2 = \big(\mathbf{y}-\mathbf{f}(\mathbf{w}+\Delta\mathbf{w})\big)^T\big(\mathbf{y}-\mathbf{f}(\mathbf{w}+\Delta\mathbf{w})\big) = \mathbf{f}^T(\mathbf{w}+\Delta\mathbf{w})\mathbf{f}(\mathbf{w}+\Delta\mathbf{w})-2\mathbf{y}^T\mathbf{f}(\mathbf{w}+\Delta\mathbf{w})+\mathbf{y}^T\mathbf{y}$$

and differentiating, will be obtained

$$\frac{\partial E_D}{\partial \mathbf{w}} = (J^T J)\Delta\mathbf{w} - J^T\big(\mathbf{y} - \mathbf{f}(\mathbf{w})\big) = 0. \tag{2.15}$$

So, to find the value $\Delta\mathbf{w}$ need to solve a system of linear equations

$$\Delta\mathbf{w} = (J^T J)^{-1} J^T\big(\mathbf{y} - \mathbf{f}(\mathbf{w})\big) \tag{2.16}$$

Since the condition number of the matrix $J^T J$ is the square of the condition number of the matrix J, then the matrix $J^T J$ may be significantly degenerate. Therefore, Marquardt proposed to introduce the regularization parameter $\lambda \geq 0$,

$$\Delta\mathbf{w} = (J^T J + I\lambda)^{-1} J^T\big(\mathbf{y} - \mathbf{f}(\mathbf{w})\big), \tag{2.17}$$

where I is the identity matrix. This parameter is assigned at each iteration of the algorithm. If the error value E_D decreasing rapidly than small value λ reduces this algorithm to the Gauss-Newton algorithm.

The algorithm stops, in the event that the increment $\Delta\mathbf{w}$ in the subsequent iteration is less than the specified value, or if the parameters $\mathbf{w}$ deliver an error E_D smaller than the specified value. Value of the vector $\mathbf{w}$ at the last iteration is considered to be desired.

The disadvantage of the algorithm is a significant increase in the parameter λ with poor approximation speed. In this case, the matrix inversion $J^T J + \lambda\ I$ becomes meaningless. This disadvantage can be eliminated using the matrix diagonal $J^T J$ as a regularizing term:

$$\Delta \mathbf{w} = \left(J^T J + \lambda \text{diag}(J^T J)\right)^{-1} J^T \left(\mathbf{y} - \mathbf{f}(\mathbf{w})\right). \tag{2.18}$$

2.3.2 Data reduction

According to the specifics of proposed TVS it returns the scanned surface as a point cloud (Fig. 2.3). on short distances it gives high detailed object, while on a distance it loses its resolution depending to the opening angle of each step of scanning. As the TVS uses stepper motors, on short distances it gives high detailed objects (Fig. 2.3c contains 10663 point), while on higher distances it loses its resolution depending to the opening angle of each step of scanning (Fig. 2.3b (3206 point) scan of "Mayan pyramid", on sides the point cloud density is less than a part of a cloud with stairs has).

In the memory of a robot each point obtained by TVS is represented with three variables – x, y, z of Cartesian coordinate system. Each of them stored using double data type that is equals to 64bit per number, so to store one point of environment 192 bit of memory is used. Using simple calculation it is possible to say that to store in memory "Object A" (Fig. 2.3a) 493,632 bits (60.3 kB) are used, "Object B" (Fig.. 2.3b) – 75.14kB and "Object C" (Fig. 2.3c) – 249.9kB.

However, a robot needs to move and to explore environment sizes of what can be limitless. So the data that need to be processed can reach gigabytes and more for one autonomous vehicle, but for the RG it needs to be multiplied on the number of individuals. That's the main reason, why it is necessary to reduce the amount of stored point in memory to a minimum that is required for obstacle avoidance and dead reckoning (low-density scanning), and detailed object scanning (high-density scanning) to use on-demand.

During the movement and mapping of environment the data that need to be processed can reach

gigabytes. For the navigation robots need a minimum amount of points to describe an object. That is why it is necessary to implement the method of low density scanning for dead reckoning and to use high density scanning on demand.

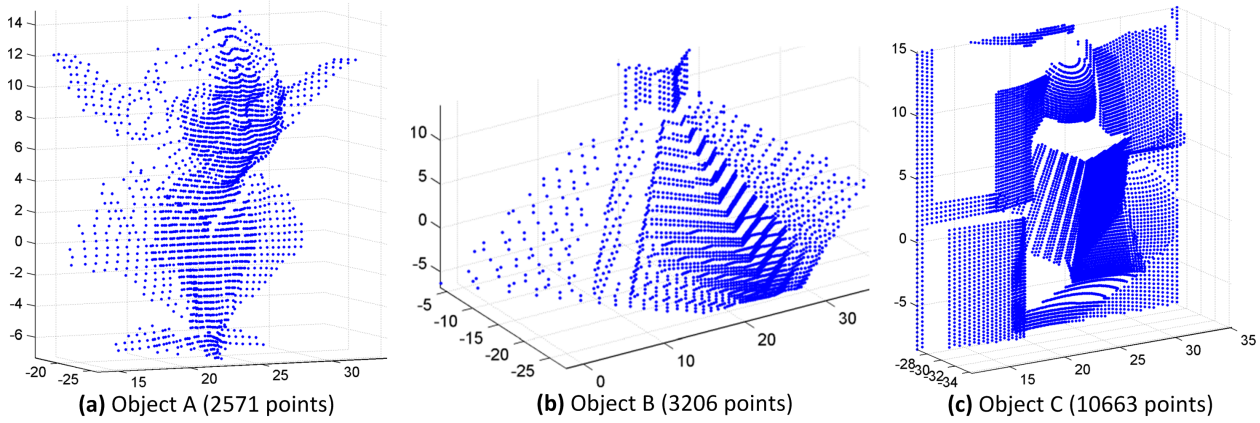


Figure 2.3 Examples of surfaces scanned by TVS

In previous work [112] were described the method of real-time data reduction during robots' movement. According to the allocated accuracy zones (Fig. 2.4) were determined an opening angle [109] equivalent to store points on detected obstacles.

As a starting point will be used and arc of one meter according to possible 160°FOV of TVS. Using the research data and type of robots described in work [98] the cloud point density (ρ) of image is 11 points per meter.

$$\rho = \frac{\lambda}{\beta p}, \tag{2.19}$$

where λ – FOV angle, β – opening angle equivalent (14.5636°for initial calculations), p – length of an arc (one meter for initial calculations). In general the length of an arc can be calculated as follows:

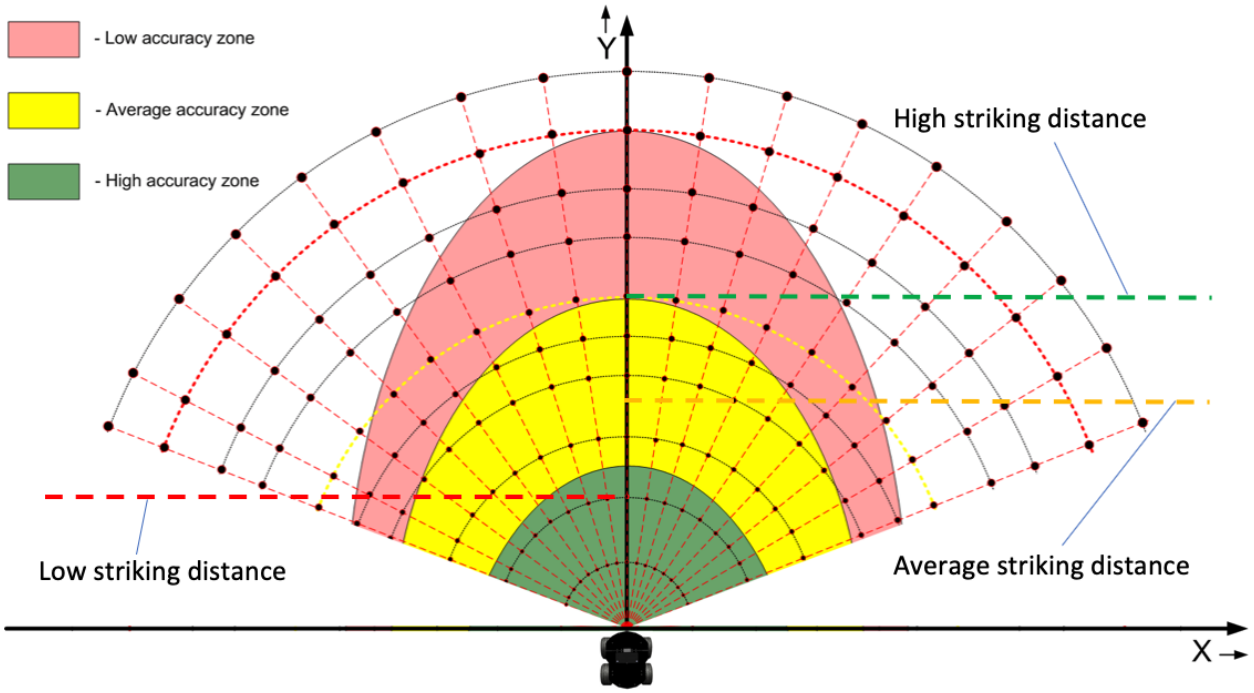


Figure 2.4 Field of view fragmentation

$$p = \frac{\pi r \lambda}{180}, \tag{2.20}$$

where r is radius of an arc (striking distance). To prevent the changes in selected resolution, the opening angle will be calculated using Eq. 2.21:

$$\beta = \frac{180}{\rho \pi r}, \tag{2.21}$$

Average opening angle for each of the striking distance zone:

$$\beta_i = \frac{\sum_{j=0}^{n} \beta_{ij}}{n}, \tag{2.22}$$

where β_{i} is an opening angle for each accuracy zone, β_{ij} is an opening angle for each striking distance in zone i. Example of the calculation are represented on Fig. 2.5. Here we have a radius

of an one meter arc equal to 0.358m with resolution of 11 points per meter.

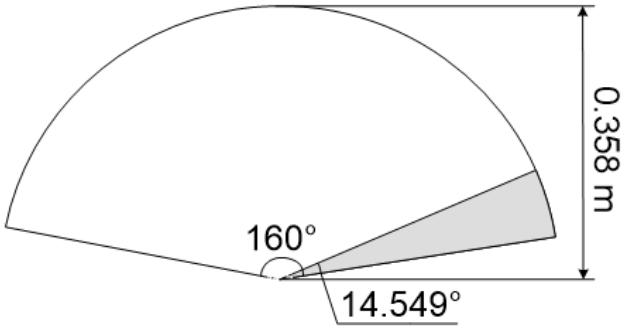


Figure 2.5 FOV with opening angle for low resolution for one meter arc length

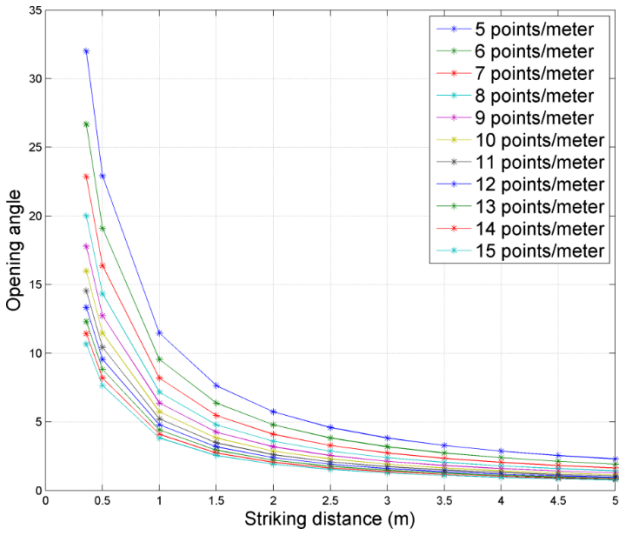
Accuracy zones are separated in a next way: from 0 to 1 meter, from 1 to 3 meter and from 3 to 5 meter for high, average and low accuracy zones respectively. Taking into account such partitioning of zones the results of the calculation are represented in table Fig. 2.6.

				Resolution (points/m)										
				5	6	7	8	9	10	11	12	13	14	15
		Striking distance	Length of arc	Opening angle (deg.)										
Striking distance range	High	0,358	1,000	32,009	26,674	22,863	20,006	17,783	16,004	14,549	13,337	12,311	11,432	10,670
		0,500	1,396	22,918	19,099	16,370	14,324	12,732	11,459	10,417	9,549	8,815	8,185	7,639
		1,000	2,793	11,459	9,549	8,185	7,162	6,366	5,730	5,209	4,775	4,407	4,093	3,820
	Average	1,500	4,189	7,639	6,366	5,457	4,775	4,244	3,820	3,472	3,183	2,938	2,728	2,546
		2,000	5,585	5,730	4,775	4,093	3,581	3,183	2,865	2,604	2,387	2,204	2,046	1,910
		2,500	6,981	4,584	3,820	3,274	2,865	2,546	2,292	2,083	1,910	1,763	1,637	1,528
		3,000	8,378	3,820	3,183	2,728	2,387	2,122	1,910	1,736	1,592	1,469	1,364	1,273
	Low	3,500	9,774	3,274	2,728	2,339	2,046	1,819	1,637	1,488	1,364	1,259	1,169	1,091
		4,000	11,170	2,865	2,387	2,046	1,790	1,592	1,432	1,302	1,194	1,102	1,023	0,955
		4,500	12,566	2,546	2,122	1,819	1,592	1,415	1,273	1,157	1,061	0,979	0,909	0,849
		5,000	13,963	2,292	1,910	1,637	1,432	1,273	1,146	1,042	0,955	0,881	0,819	0,764
		Average opening angle for width	High	22,129	18,441	15,806	13,831	12,294	11,064	10,058	9,220	8,511	7,903	7,376
			Average	5,443	4,536	3,888	3,402	3,024	2,722	2,474	2,268	2,094	1,944	1,814
			Low	2,744	2,287	1,960	1,715	1,525	1,372	1,247	1,206	1,181	1,096	1,091

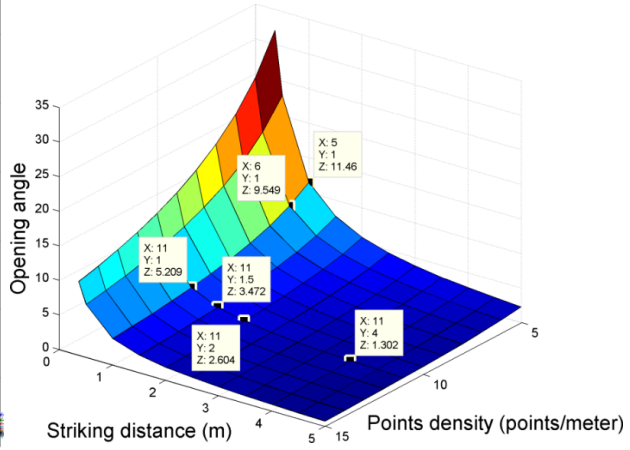
Figure 2.6 Opening angle comparison

Fig. 2.7(a) represents the general relation between opening angles and striking distances for the reviewed density of points. Fig. 2.7(b) represents the same but also taking into account point cloud

density. Using the defined accuracy zones Fig. 2.8 demonstrates average values of such a ratio.



(a) Dependencies of opening angle and striking distance



(b) Dependencies of opening angle, striking distance and point cloud density

Figure 2.7 Opening angle equivalents

According to the calculation the average angles based on the initial point cloud density (11 points/meter) are 10.059°for "High accuracy zone", 3.011°for "Average accuracy zone" and 1.34°for "Low accuracy zone". The average angle for the "High accuracy zone" range will give a small resolution equal to 5-6 points per meter. So the low edge value of an opening angle for "High accuracy zone" was taken. The set of angles changed to 5.209°, 3.011°, 1.34°.

Identification of the obtained point on the obstacle surface, belonging to a specific accuracy zone, is simplified to the solution of a point belonging to an ellipse in a Cartesian coordinate system. Using the extended algebra of algorithms it can be written as the next expression:

```
RESOLUTION=<INPUT(striking_distance)*[zone_state]
(dcn v don v den)*RULES>
RULES={[critical]( RESOLUTION_LOW) *
```

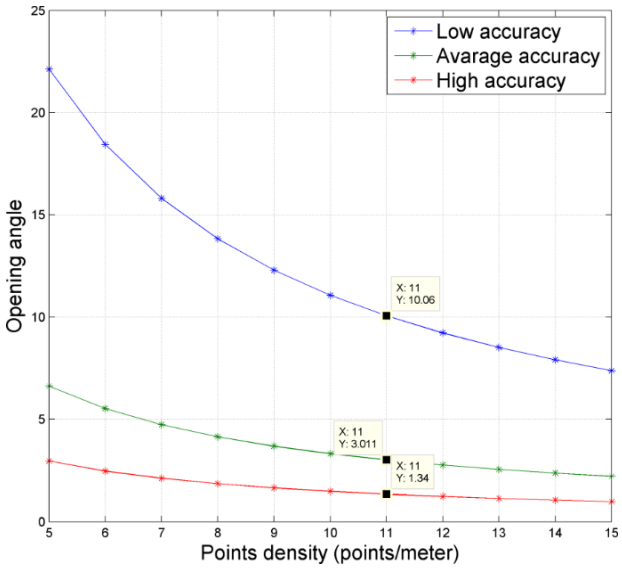


Figure 2.8 Average values of point cloud density

* [optimal](RESOLUTION_MEDIUM) *

* [effective](RESOLUTION_HIGH)}

Chapter Three

Data exchange for robotic group

Communication within the robotic group is one of the main tasks in swarm robotics. Its implementation helps to expand the possibilities of a swarm by improving tasks of flocking, foraging, navigation, etc.

Communication works distinctively in most autonomous robotic group systems and depends on the most part upon factors like the surroundings, robots' dimensions, the financial possibilities of the development, on project restrictions, or other external factors. The size of the robot has a direct influence on the possible hardware that can be installed, so it increases a variety of communication devices that can be used during development and improve communication within the group. As an example of communication can be mentioned bluetooth, wireless LAN (Local Area Network), communication using the environment (stigmergy).

Looking at different natural swarms it is possible to notice that some of them are not using messages (voice) for communication. However, those swarms are still communicating with each other. This type of communication is called stigmergy [113] where the swarm is using the environment to interchange the data. For example, ants are placing pheromones to solve the task of

foraging and finding the shortest path. Ants start to follow the most odorous route and continue to leave their pheromone, after some time the ants start to follow an optimal path. For robotic swarm the stigmergy is not so efficient, more common ways for data exchange are bluetooth, wireless LAN (WiFi), or infrared.

Reviewing the infrared communication has several advantages and disadvantages. To use infrared communication robots must be in the direct vision of each other. But this technology can be adopted for swarms and according to its specifics robots will have a natural filter of information that comes outside of their group. On the other hand, infrared communication also has its benefits in the micro-robotics application cause of there low usage of the energy resources. However, different sources of light can interrupt data transference.

For mid-size robotic groups, wireless LAN communication is applied. Bluetooth type of communication can be placed between infrared and LAN according to its size and range capabilities. Bluetooth gives the fundamental possibilities of inter-robot communication. Each participant must have assigned a unique identifier. The content of messages can be simple and complex. Simple messages can contain commands like 'start', 'stop', or some other directives, and complex messages can contain additional descriptive information about the environment or requests for providing a list of operations based on the robot's capabilities. Messages can be distributed through the robotic group or swarm using the protocol like a spanning tree.

Exists two types of communication, it can be with global or local interaction. During the global communication received message contains local information of the sender. In most cases this information is useless. In swarm or group robotics is used local communication. This type of communication takes its origin in nature (herding is a good example, where the local interaction

helps to survive the predators' attacks by signaling to their kin with movement or sounds).

In turn, local communication can be direct and indirect. Direct communications are real-time data transferring within the group. In this case, a robot sends a message to the group and they have to process it immediately. For direct communication can be used WiFi connection, bluetooth, or more primitive types of communications like light and sound. Indirect communications use different types of mediums that can be used for late access information storing (mail services). For example, in swarm robotics, it is implemented during the task of SLAM by living an NFC (Near Field Communication) card on detected landmark (implementation of pheromones used by ants).

There are a few characteristics of communication [114] that have a significant impact on how it can be functioning and should be designed:

Communication Range has a direct impact on what distance should be between robots to achieve complete data exchange. On the other hand, the large communication range can cause the communication to jam, cause each participant need to listen for all of the existing messages.

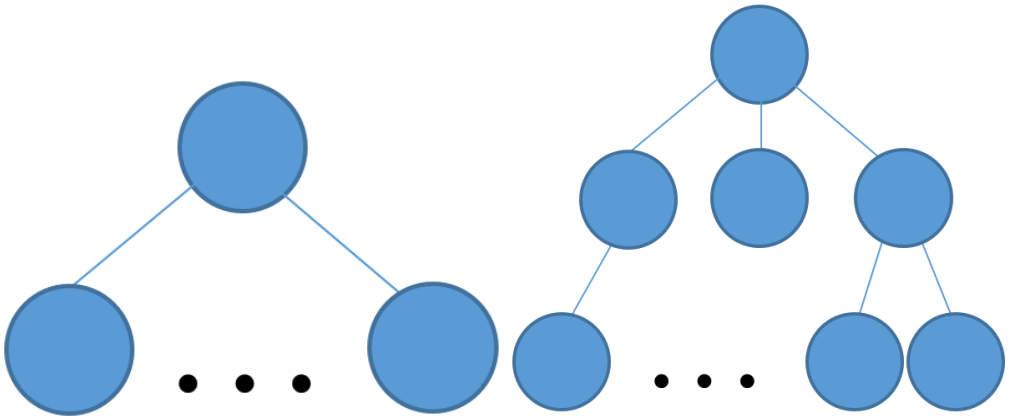
Communication Area in the perfect case covers all 360°, this case is possible if only one device for data transferring exists on the top of a robot.

Length of Messages also has an influence on the communication process. In the case of small messages, the network becomes overloaded with small pieces of data and creates queues to process, on the other hand, the bigger message is the more chance to fail it has.

Propagation Time is the parameter that reflects how much time needed to distribute the data/message within the whole robotic group/swarm.

Interference and other external influences is what each communication network is dealt with. For example, external Wireless LAN, or like was mentioned before the sunlight for the infrared. Types of interference based on the location and environment can vary.

Research considers two models of data transferring: information exchange with centralized management (Fig. 3.1aa) and strategies of centralized hierarchical control (Fig. 3.1bb).



(a) Information exchange with centralized management

(b) Information exchange using strategies of centralized hierarchical control

Figure 3.1 Models of data transferring

In the next sections will be presented solutions for data transferring task in swarm robotics. Solutions are inspired by Spanning Tree Protocol (STP) [115] and Shortest Path Bridging (SPB)[116].

Spanning Tree Protocol is a data link layer protocol ([117], [118]). The main task of STP is to eliminate loops in the topology of an arbitrary Ethernet network, in which there are one or more network bridges connected by redundant connections. STP solves this problem by automatically blocking connections that are currently redundant for the full connectivity of switches.

Shortest Path Bridging is the IEEE 802.1aq standard, it is a network technology that simplifies the building and network configuration while taking advantage of multipath routing.

3.1 Spanning tree forming for swarm robotics

Consider a general case of the swarm that can be proposed as a method of network forming based on creating a spanning tree. The algorithm consists of seven steps and includes the use of classical approaches. Steps of dynamic network forming for robotic swarm are next:

- Build a fully-connected network graph;
- Use the Kruskal algorithm to build the minimum spanning tree;
- In the obtained tree, use the Floyd-Warshall algorithm to receive the list of all possible routes in the network;
- Calculate the average route length for each node;
- Select the node with the lowest average length and configure it as a high-level node;
- Nodes with "one side" connection configured as low-level nodes;
- Other nodes configured as mid-level nodes.

Applying this method it is possible to obtain both of the network types automatically (depends on the robot placement, a number of robots in a swarm, and signal of the network). In particular cases (Fig. 3.2) in calculations is considered open space without obstacles, so the distances between nodes were used. In more complicated scenarios, distances should be replaced with the wireless network signal levels.

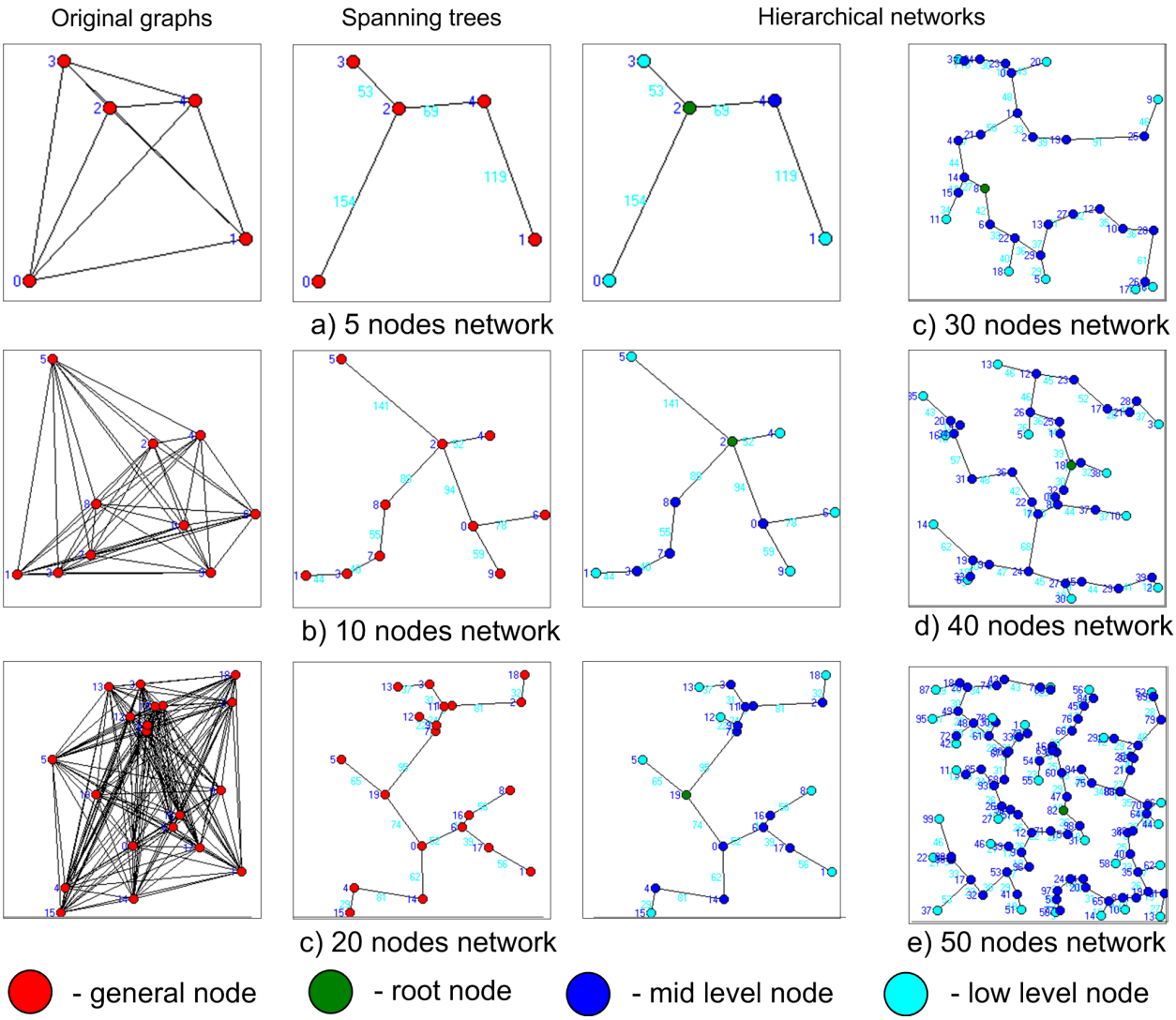


Figure 3.2 Calculated networks

This method is useful for a large swarm, but in the case of a small group can be used methods that include more behavior control and solve several tasks simultaneously.

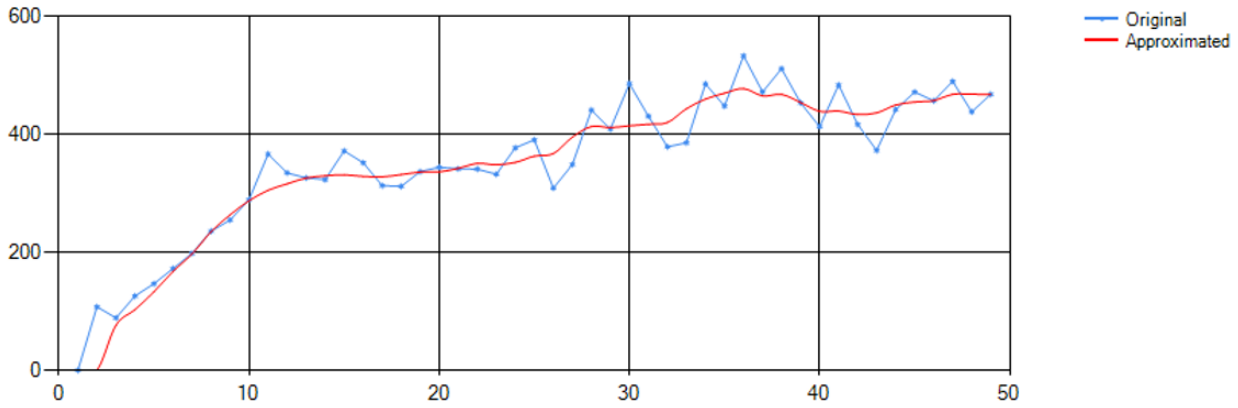
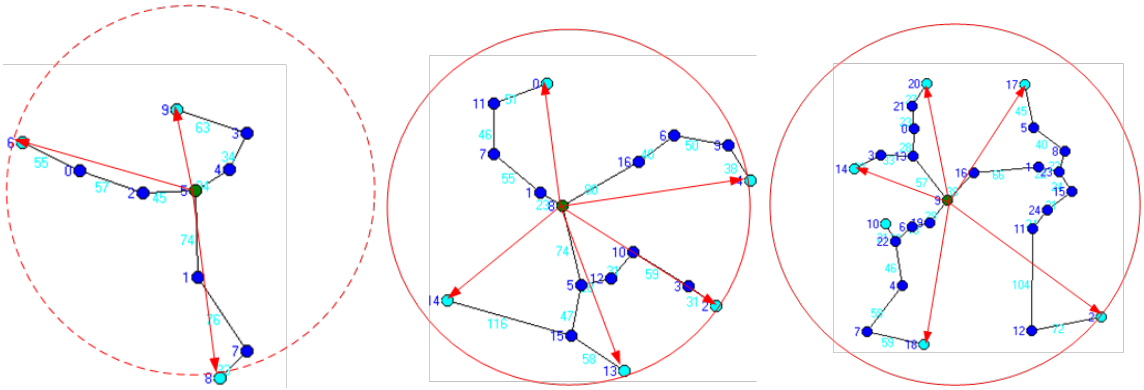


Figure 3.3 Network modeling results: 1 to 50 nodes with cross-validation



(a) Example # 1 **(b)** Example # 2 **(c)** Example # 3

Figure 3.4 Examples of network levels estimation based on geometrical center search

3.2 Leader based communication

Leadership is one of the principal features of swarm robotics is the local nature of the interaction of robots with each other, as well as the robot with the environment. This interaction is called implicit communication. The idea is that each robot group interacts directly only with its neighbors, in a restricted area of visibility. In such a system, the robots make their own decisions on how to proceed, based on some simple rules for local interaction.

In the "follow the leader" method is considered that the group has a priori defined leader, who

sets this movement. Terms of local interaction can be very different: from a purely formal to very exotic. For example, rules of schooling movement based on the model of the springs and shock absorbers. "Spring" component model defines the attraction of individuals to the leader (and not to each other), and "Shock absorbers" – pushing away from the leader. On the other hand, there is an option that allows determining the leader to solve the problem of coordinated motion.

One of the models that describe the locally interacting robots organization is static swarm [119]. It is characterized by the absence of a given control center and is some kind of a fixed network – a set of agents. The basic properties of a static swarm are activity, local interactions, and functional heterogeneity. That is why will be reviewed the method of role distribution based on the task of selecting a leader. Under the term "Leader" we will understand the central node of data exchange (robot for a short period of time will become server to store and marge data). For choosing a leader robots will be using a voting process. Each robot can be described as a set of parameters:

$$R = (I, L, E, N) \tag{3.1}$$

where I – identifier of robot, L – identifier of voted leader, E – evaluation of the leader L (amount of voices that have to be given for a leader), N – list of connections available for robot (its neighbors).

The voting process on the initial step goes the following way: each robot evaluates his neighbors for the role of leader according to the set of previously defined its characteristics; each of these characteristics have their own weight; using the membership function robot selects the neighbor with the highest value.

For the vote value will be implemented a linguistic variable e = "evaluation of robot". Its value

is based on the scale of M = "very low", "low", "medium", "high", "very high" or it can have a digit equivalent M=1, 2, 3, 4, 5. After voting process many alternatives for E will be generated, so it will have next form:

$$E = \{e_1, e_2, ..., e_n\} \tag{3.2}$$

where e_i – alternative "candidate" at and n is amount of visible neighbors. For robot evaluation are offered the following characteristics: 1) surroundings: c_1 = "the number of neighbors evaluable for candidate"; 2) territorial: c_2 = "the distances to the neighbors or levels of signals"; 3) status: c_3 = "the physical state of the robot". Each of these characteristics are estimated by a voting robot for each of its neighbors:

$$C_i = \{c_{i1}, c_{i2}, ..., c_{ik}\} \tag{3.3}$$

where c_j – characteristic value of i-th "candidate" at j=1..k. Each of the characteristics has its weight:

$$W = \{w_1, w_2, ..., w_k\} \tag{3.4}$$

where – the j-th characteristics weighting $\sum w_i = 1$, Evaluation of the i-th neighbors uses the following formula:

$$e_i = \sum_{j=1}^{k} w_j c_{ij} \tag{3.5}$$

To determine the value of linguistic variable we use three types of membership functions (Eq. 3.6 – 3.8), where a general view is represented in Fig. 3.5.

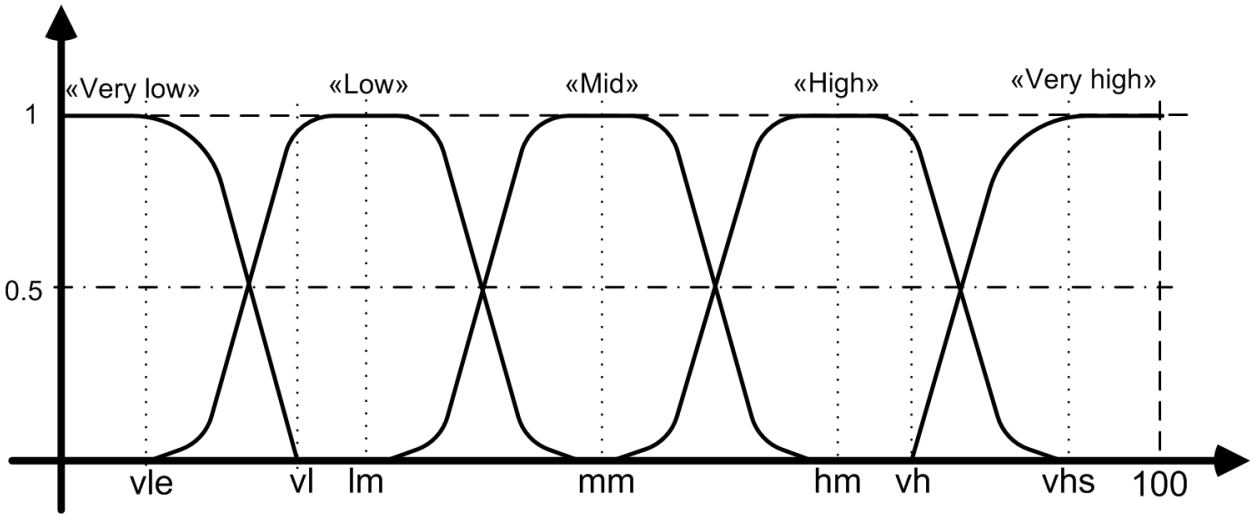


Figure 3.5 Membership functions

$$f_{vl}(e_i) = \left\{ \begin{array}{l} 1, x < vle \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{e_i - vle}{vl - vle}), vle \le x \le vl \\ 0, x > vl \end{array} \right\} \tag{3.6}$$

where vle is the threshold to which the membership function is equal to "1", vl is the threshold, after which the membership function is equal to "0".

$$f_{vh}(e_i) = \left\{ \begin{array}{l} 1, e_i > vhs \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{e_i - vhs}{vhs - vh}\pi), vh \le e_i \le vhs \\ 0, e_i < vh \end{array} \right\} \tag{3.7}$$

where similar to Eq. 3.6 vhs is the threshold for "1" and vh is for "0".

$$f_{gb}(e_i) = \frac{1}{1 + \left|\frac{e_i - c}{a}\right|^{2b}} \tag{3.8}$$

where c is the middle part of a membership function, a is the value at which $f_{gb}(c + a) = 1$ and $f_{gb}(c - a) = 1$, b is the value of function smooth regulation.

3.3 Feedback implementation and method improvement

In the case where some of the information need to be transferred between all robots within the group based on that certain of them (robots) have to be notified that transfer is complete. This task is the dissemination of information feedback (PIF – propagation of information with feedback) is formulated as follows: a subset is formed by robots of those which know message M (the same for all robots) which should be spread, that is, all robots must take M. Certain processes must be notified of the transfer is complete, a special event notification must be done, and it can only be done when all processes have already received the M. Alert in PIF-algorithm can be viewed as a return (OK) event.

Any wave algorithm can be used as PIF-algorithm. For example, let A be a wave algorithm. To use A as a PIF-algorithm, we take the robots, initially knowing the M is an initiator of A. Information M is added to each message A. This is possible because, by construction, starters of A know M initially, and the followers do not send messages until they receive at least one message, that is until they get M. When return (OK) events occur, each process knows the M, and return event (OK) can be considered as the required notify event in PIF-algorithm.

Two models of data transferring were previously reviewed: information exchange with centralized management (Fig. 3.1a) and strategies of centralized hierarchical control (Fig. 3.1b). When using the strategy of centralized management of a robotic group R, every robot r_i (i = 1, 2,..., N) of group transmits data about its state and information obtained about the environment in the central control device (robot chosen by the estimation process).

The hierarchical strategy of a centralized management network between robots can be represented with layers. Layers can be separated into three types: the top layer is a single central control

device which merges data and initiates backward propagation; a middle layer is a group of control devices for existing to send their data and data from lower levels (layers) to top layer; low layer can communicate only with the elements of the middle layer, sending the data and receiving the data after merging.

The leader-changing method can be simplified for the layers distribution inside the group. To define network roles will be implementing linguistic variable p = "pattern of layer". It uses three levels scale of M = "lower layer", "middle layer", "top layer". Correspondingly, many alternatives of P can be represented in the following form:

To determine the value of the linguistic variable (Fig. 3.6) we use three types of membership functions [120], where the extreme values ("Low-level" and "Top-level") will determine Z- shaped (31) and the S-shaped (32) functions, the degree of belonging to the "Middle-level" value is based on trapeze-like membership function (33) (general formulas are represented).

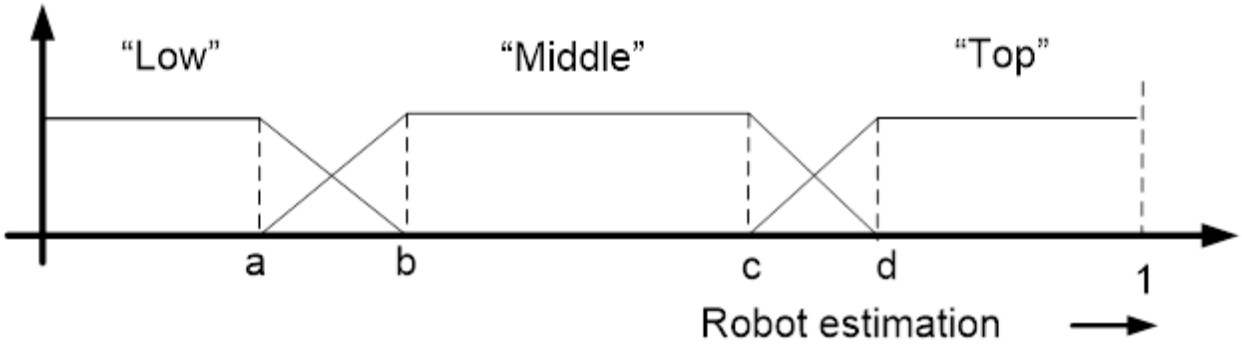


Figure 3.6 Functions for network layer determination

$$f_{low} = \left\{ \begin{array}{l} 1, e_i \le a \\ \frac{b-e_i}{b-a}, a < x \le b \\ 0, x > b \end{array} \right\} \tag{3.9}$$

$$f_{top} = \left\{ \begin{array}{l} 0, e_i \leq c \\ \frac{e_i - a}{b - a}, c < e_i \leq d \\ 1, x > d \end{array} \right\} \tag{3.10}$$

$$f_{mid} = \left\{ \begin{array}{l} 0, e_i \leq a \\ \frac{e_i - a}{b - a}, a < e_i \leq b \\ 1, b < e_i \leq c \\ \frac{d - e_i}{d - c}, c < e_i \leq d \\ 0, e_i > d \end{array} \right\} \tag{3.11}$$

where e_i is evaluation of the i-th robot, it takes the following form

$$e_i = \sum_{j=1}^{k} w_j c_{ij} \tag{3.12}$$

In terms of fuzzy logic, it can be described using next IF – THEN rules type:

```
IF robot evaluation IS top level,
    THEN network level EQUALS host
IF robot evaluation IS mid level,
    THEN network level EQUALS level 1
IF robot evaluation IS low level,
    THEN network level EQUALS level 2
```

where "NET HOST" – robot becomes a host for data transferring (top level), "NET LVL 1" and "NET LVL 2" for determining the network level for communication (middle and low levels) and n – fixing statement.

Schematic representation of PIF for the current case is in Fig. 3.7.

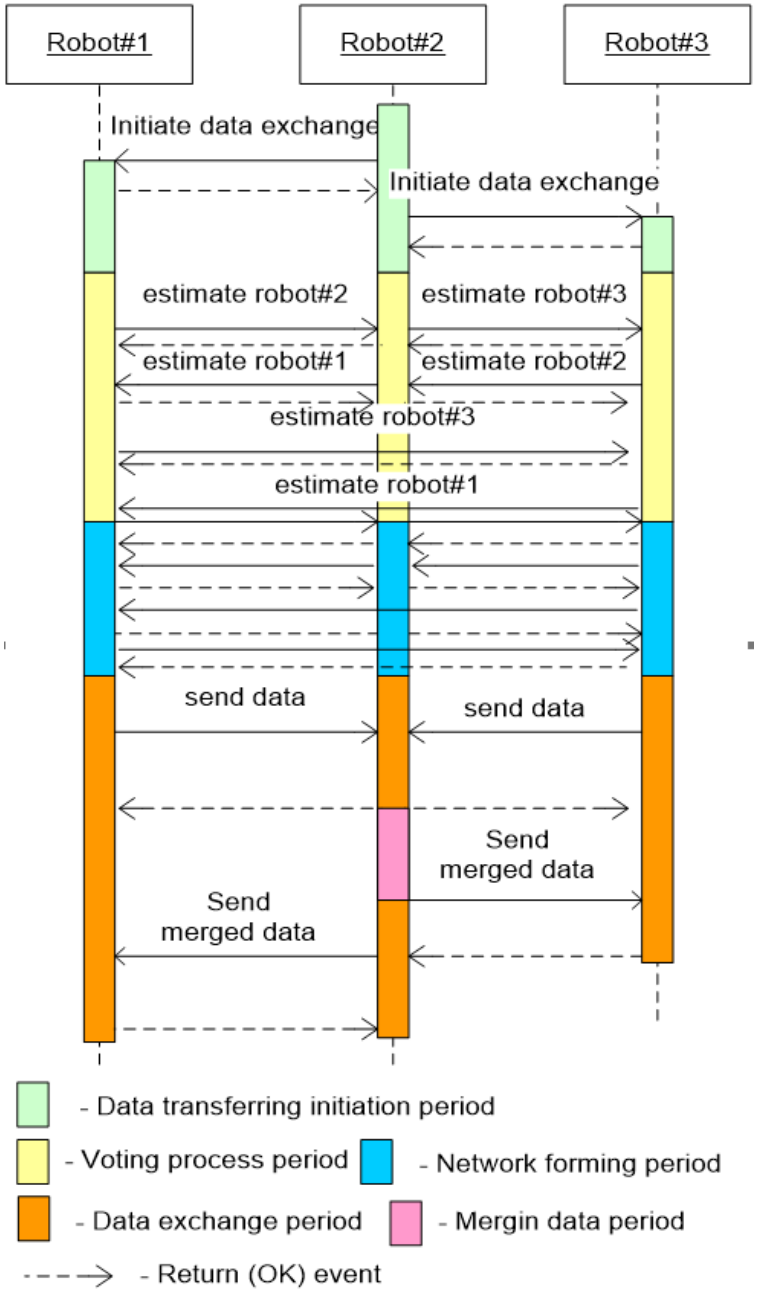


Figure 3.7 Margin data about environment (sequence diagram)

Data transferring initiation period is at a state when one of the robots in group sends messages to others to start data transferring, and occurs when: 1) robot needs additional data for further navigation, or 2) robot has collected enough of a portion of information from TVS that seems to

be transferred to others in group. The voting process period is used for evaluation of each robot in group. Compilation of data transferring channels happens at network forming period. The data exchange period is used to interchange the accumulated data according to the topological structure of the network. After this comes the data merge. The last two periods have floating time depending on the amount of data accumulated by each robot.

3.4 Implementation results

When using the strategy of centralized management of a robots group R, every robot of group transmits information about its condition, collects information about the environment in the central control device, and receives commands from the central control unit (Fig. 3.8). In group of N Robots, where every robot transfers on central control device message with size of Kout and receive commands with size of Kin, the volume of transferred info will be:

$$I = N(K_{in} + K_{out}) \tag{3.13}$$

Thus, the load on the communication channel is directly proportional to the number of robots in the group.

Lest make the modeling of such system consider the case N = 5 (where n – the number of robots) in two sates – where all robots have a good connection to a server (Fig. 3.8) and when robot #1 and robot #2 have a bad connection that causes a package lost (Fig. 3.12). Modeling is based on queue theory, where information about an obstacle that was found by robots is sent to the server for further processing.

Modeling in MATLAB was made in the period of 500 seconds for all cases and gave us the

next results represented on the next figures: Fig. 3.9 shows us the total amount of sent requests for processing and processed requests from all five devises, their difference explained as some of the requests are duplicated; Fig. 3.10 represents a timeout in request processing (amount of time needed for the next request to be processed by a server)(Timeout for robot #1, Timeout for robot #2 and etc.); Fig. 3.11 represents modeling for the second case, where the bad signal to the server causes the loss of requests (the difference between sent requests for processing and received requests).

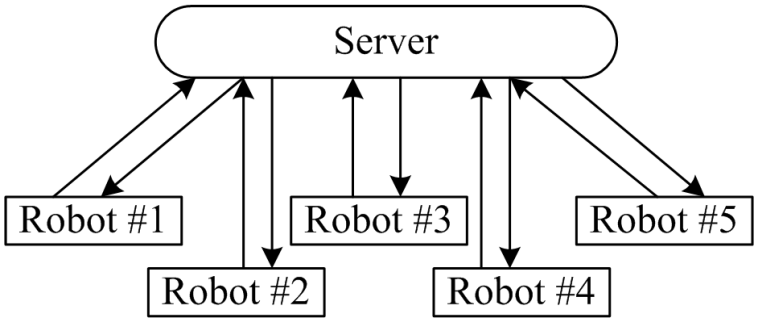


Figure 3.8 Information exchange with centralized management

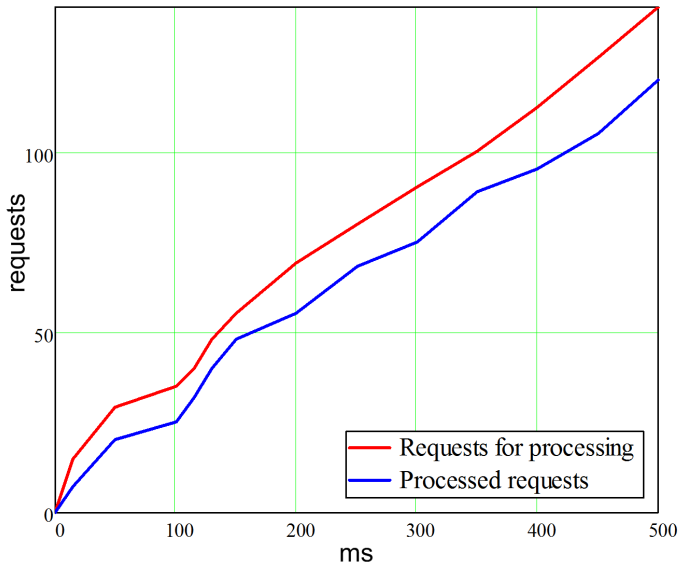


Figure 3.9 Total amount of sent and processed requests

Such data loss causes troubles in a movement of robots and increases the time of task imple-

mentation. That is why the network structure must be changed.

When using a hierarchical strategy of centralized management central control device is subject to a top-level hierarchy of the robots, each of them is subject to several robots of lower-level hierarchy (Fig. 3.12 represents a hierarchy for our case).

Such complex control schemes require very high performance of communications, because all the robots, except the lowest level of the hierarchy, are interacting with robots of lower and upper levels of the hierarchy. Displacement information through the transceivers of the robot may be estimated as:

$$I = n(k_{in} + k_{out}) + K_{in} + K_{out} \tag{3.14}$$

where n is the number of robots in the lower level of hierarchy; k_{in} in is the amount of information in the incoming messages from the lower level robot of hierarchy; k_{out} – the amount of information in outcoming messages to the lower level robot of hierarchy.

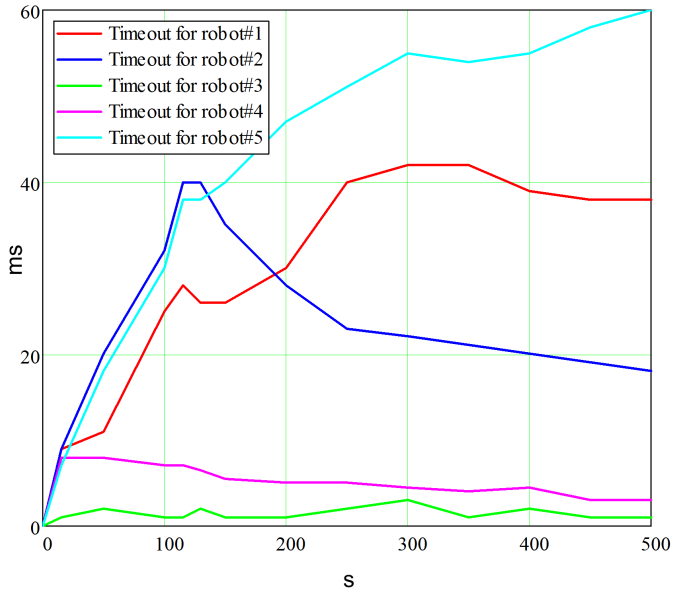


Figure 3.10 Timeout in request processing

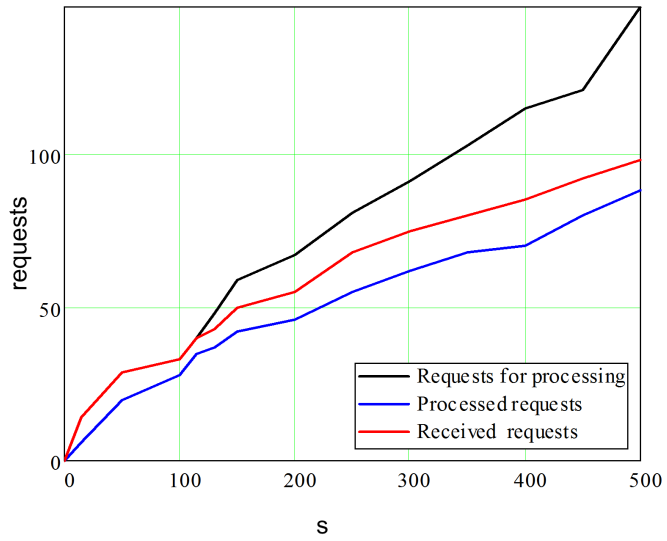


Figure 3.11 Total amount of sent, received and processed requests with signal loss

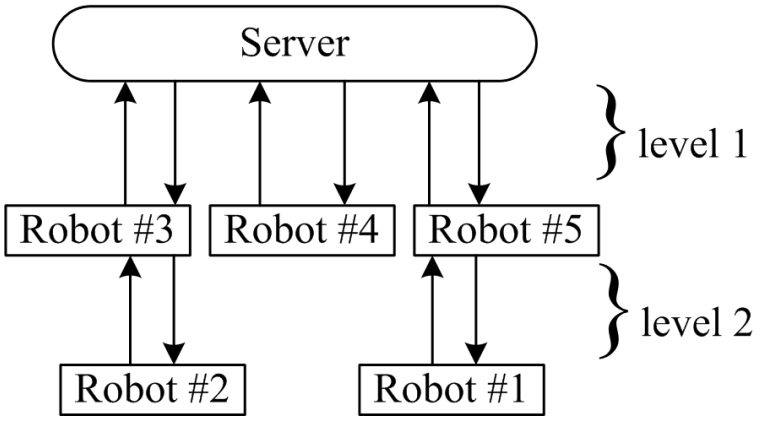


Figure 3.12 Information exchange using strategies of centralized hierarchical control (shown 2 levels)

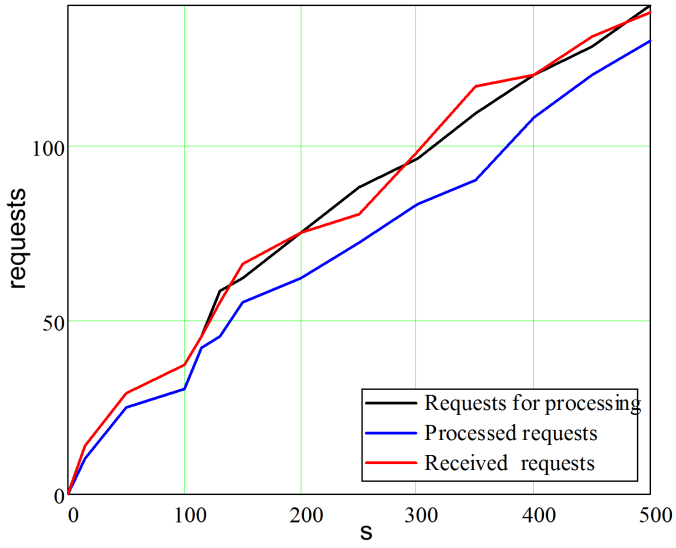


Figure 3.13 Total amount of sent, received and processed requests with signal loss using centralized hierarchical control

This robot swarm is processing and transferring the geometrical data of our original TVS which are enough precise and naturally represent information about swarm surrounding in the same Cartesian system. It significantly simplifies the decisions for navigation performance. The decision of dynamic data exchange is supported by average timeout in request processing that at the end of the modeling is suspended to the time of our TVS obstacle detection time (0.039 s in both cases of network structure (Fig. 3.10, Fig. 3.14)).

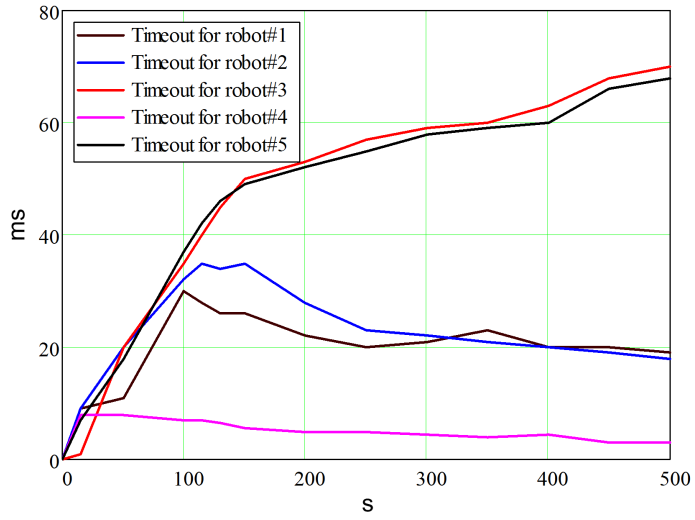


Figure 3.14 Total amount of sent, received and processed requests with signal loss using centralized hierarchical control

The leader changing system that we implemented improves the process of data transferring by dynamically changing the network model from centralized management to centralized hierarchical control and backward.

Results of our system implementation are shown in Fig. 3.15 by comparison the number of processed requests before and after the leader changing system implementation.

Such approach helps to increase the movement speed of a group of robots and decrease the time of their task accomplishment.

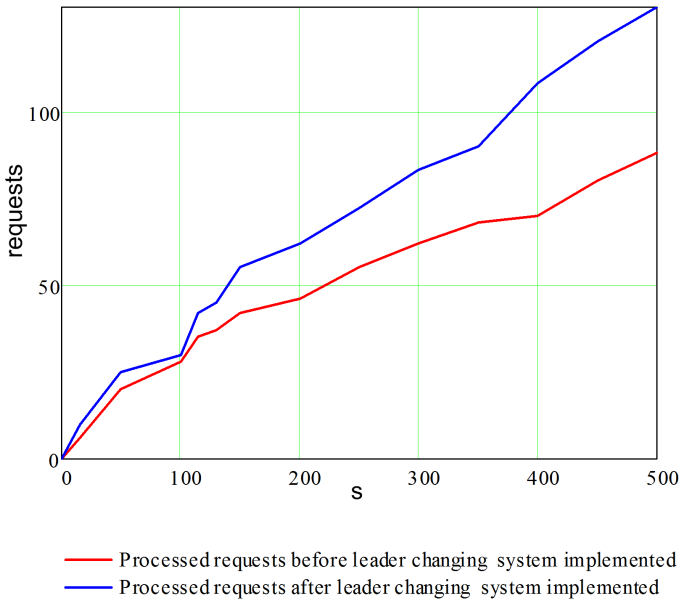


Figure 3.15 Comparison of the number of processed requests before and after leader changing system implementation

Proposed dynamic data exchange network forming method extends the potential of our novel TVS. It overlaps an ability of single robot navigation with a cloud-like common knowledge base within the robotic group to improve the efficiency of dead reckoning.

The proposed methods allow the elimination of topological loops in the data network in a group of robots. A fully connected graph of a real network with a high probability leads to endless repetitions of the same messages in a group, while network bandwidth is almost completely occupied by these useless replays. In these conditions formally the network can continue to operate but in practice its performance becomes so low that may lead to a complete network failure. Therefore, the proposed methods ensure the full propagation of information within the group and help to improve the movement coordination of a robotic group by exchanging information about the missing sectors.

Chapter Four

Path planning methods

The most common way to define the path planning (motion planning) problem as an agent needs to move from its initial position to the goal by avoiding obstacles and achieving cost minimization. The cost definition to find the optimal path varies based on the system's criterias, in some systems, it is time, in other distance, energy, etc. But as the most general approach is to minimize distance (shortest path) between the start and the goal points. However, the definition of optimality changes in some situations. As an example for robot matters the time used for calculation (amount of iterations and there's complexity). So when the path calculations take too much time it is harder to achieve the task continuit. Such reason causes to choose the most suitable algorithm based on desired optimality criterion. In some cases, it is better to use more complex criteria – combine the criteria. So motion planning is one of the key tasks in robotics.

During the task of navigation can be allocated three main branches:

The Path Planning task when robots are provided with the goal point and/or points of interest and the map of the surrounding area is converted to the quadtree model using the distance transformation according to the desired resolution. After that calculates path from initial position through the points

of interest to the goal.

The Path Execution is the task where a robot starts to move from its initial position to the goal using the path calculated during the path planning.

The Model Update is triggered when a robot detects any kind of obstacle in its field of view (FOV) and then updates the quadtree model of the surroundings.

In mathematics, there are well-developed algorithms for finding the way in an unknown or partially known environment (Optimal and heuristic algorithms). For this purpose, discrete mathematics (graph theory) and linear programming are usually used. Tasks of the shortest path search in the graph are known and studied (for example, Dijkstra's, Floyd-Warshell's, Prim's, Kruskal's, algorithms, etc. [121] and [58]). Algorithms can be separated into two categories:

- Classic [122] [123] (Dijkstra's, Floyd–Warshell's, Prim's, Kruskal's, algorithms, etc.)
- Heuristic [124] [125] (A* algorithm, ant algorithm, genetic algorithm, etc.)

Exists many types of researches in the frame of path-planning. For example [119] where authors represented an approach that uses motion primitive libraries. In [126], representing an attempt to implement animal motion for robot behavior, or [2] suggested an algorithm of collision free trajectory for robots.

This section will cover the main aspects of path planning used in the research according to the specifics of the used technical vision system, environment and tasks.

4.1 Algorithm review

In mathematics are well-developed algorithms (Fig. 4.1) for finding the way in an unknown or partially known environment (Optimal and heuristic algorithms). For this purpose, usually used discrete mathematics (graph theory) and linear programming. Tasks of the shortest path search in the graph have known and studied (for example, Dijkstra's, Floyd-Warshell's [127], Prim's [128], Kruskal's [129] algorithms, etc.).

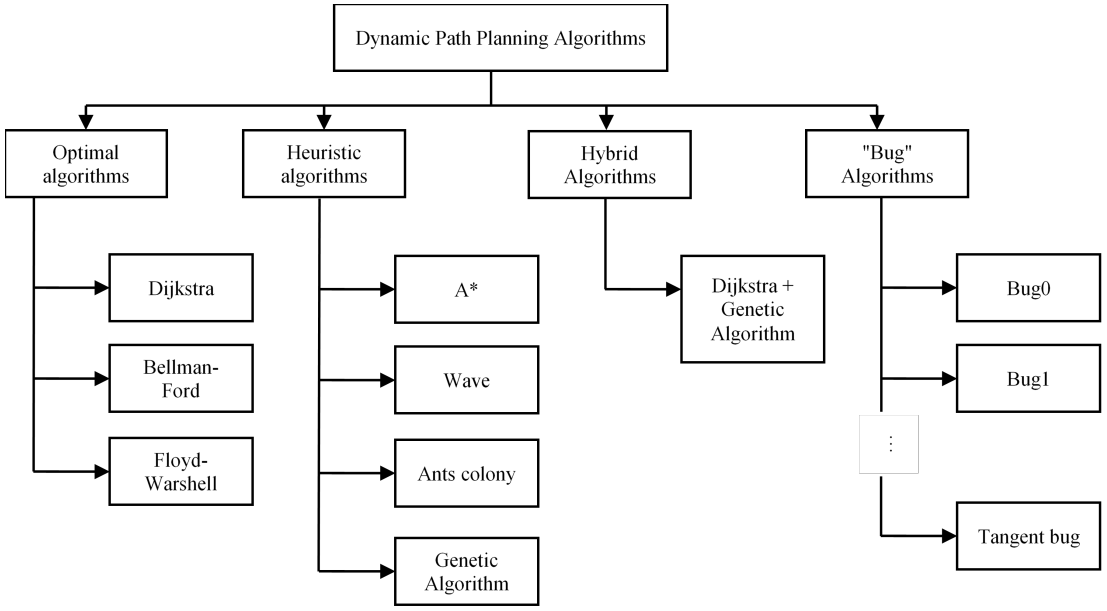


Figure 4.1 Dynamic Path Planning Algorithm

Dijkstra's algorithm [130] is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds the shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

Floyd–Warshall [127] in computer science is an algorithm for finding the shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm. Versions of the algorithm can also be used for finding the transitive closure of a relation R, or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph.

A* or A-star or A* search [131] is one of the well-known and basic heuristic algorithm. It is a combination of Dijkstra's algorithm. The algorithm tries to minimize the function formulized as f(n) = g(n) + h(n) heuristically considering the relationship between the nodes and the edges. g(n) refers to the cost of starting point or node and h(n) implies a heuristic cost estimation related to the remaining path. h(n) hereby constitute the heuristic base of the algorithm [132].

Genetic algorithm (GA) [133] is the other popular heuristic approach. This method imitates the evolutionary process and attempts to acquire the best individuals (chromosome) which represent the optimal path for this type of problem with regard to the defined objective function. The crossover strategy is applied to the parents in order to create new individuals. In addition to this, the mutation process which prevents the algorithm to converge on local minima is one of the crucial parts, so that the variety of the solution space is always preserved. GA and similar heuristic methods are an alternative way for the solution of the optimal path problem and they are often utilized.

Rapidly-Exploring Random Tree (RRT) is another probabilistic based algorithm which is improved by Lavalle and Kuffner [134], [135]. It is aimed at the solution of the path planning problems which have nonholonomic constraints. The algorithm generally gives effective results

in non-convex and high-dimensional spaces. The tree which implies the solution space is built incrementally and two different functions which are the creation and the expansion of the tree are used all over the scheme. The expansion of the tree is one-way which is from start to endpoint. On the other hand, the idea of the expansion of the tree in a bidirectional way is improved the time efficiency of the whole solution. This technique is called bidirectional RRT (bRRT or B-RRT) and the tree is established in both starting and target points and then expanded into the unsearched parts. The search process is terminated when the junction point of these two trees is found and this path which is the output of the algorithm is an optimal route [136].

Probabilistic Roadmap (PRM) is one of the probabilistic-based and last path planning algorithm used in this study. The optimal path is acquired by the distance calculation of the edges which are the connections between randomly created nodes on the map. First of all, the nodes are generated by sampling from the non-obstacle points on the map and then these nodes are linked to each other, and lastly, the path cost is evaluated. The rapid random cluster strategy may provide faster results than the other algorithms but not guarantee the shortest path [137].

The Bug Algorithm Family solves the navigation problem by storing only a minimal number of waypoints, but without generating a full map of the environment. If no solution path exists, the algorithm is able to recognize this situation and terminates reporting that the target is unreachable, instead of endlessly wandering about.

The Bug model makes three simplifying assumptions about the robot [138]. First, the robot is a point object. Second, the robot has a perfect localization ability. Third, the robot has perfect sensors. These three assumptions are unrealistic for real robots, and therefore Bug algorithms are usually not directly applied for practical navigation tasks but can be considered as a higher-level

supervisory component of a system that incorporates all three assumptions. Bug algorithms can be seen as a first logical step towards solving a robotic 2D navigation task

The following algorithms from the Bug family have been implemented and evaluated: Bug1[138], Bug2[138], Alg1[139], Alg2[140], DistBug[141], Class1[142], Rev1[143], Rev2[143] and TangentBug[144].

4.2 Navigation algorithms analysis

The analysis based on the results in [99] are presented in graphics (in the case of "one to all" Fig. 4.2 and "all to all" Fig. 4.3 search) were represented the amount of time (in milliseconds) needed to find the shortest path in case of 100 to 2500 nodes for each of the algorithms. The difference in the performance of Dijkstra's and Floyd-Warshell's algorithms at "all to all" search is insignificant until we have less than 500 nodes (Fig. 4.3), but the resulting routing matrix has different parameters (length of the route is the same, but the ribs used for its passage are different). Bellman-Ford's algorithm is unacceptable for use in a task with lots of nodes because of its high calculation time

4.3 Navigation using technical vision system

The task of path-planning can be presented the next way:

- Robot is deployed in an unknown environment;
- Its current position marked as a starting point and target location as an endpoint;
- Robot calculates the heuristic rout and starts moving towards the target;

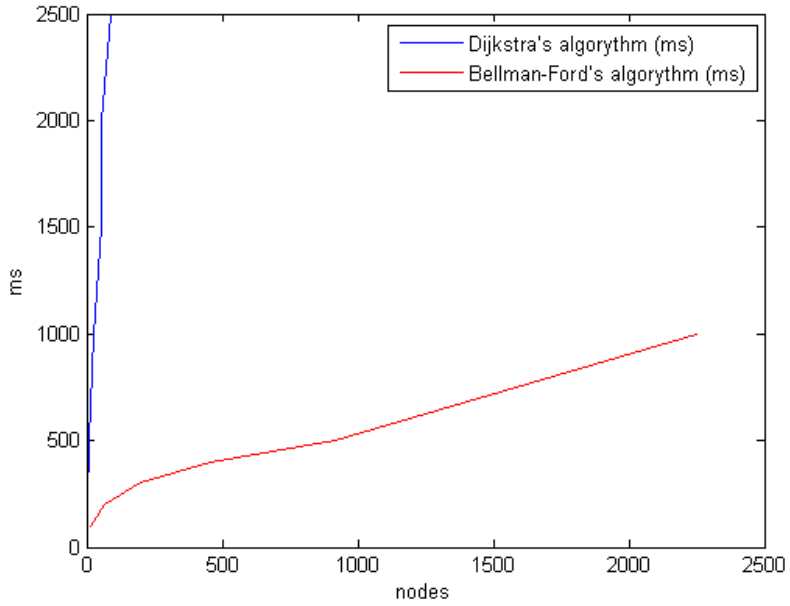


Figure 4.2 Results for "one to all search"

- In case of obstacle detection by the vision system, robot updates his navigation map and recalculates the route;
- processes of obstacle detection and path update continues until the target is reached.

Additionally, the obtained path should be approximated to obtain a continuous and energy-saving trajectory (Fig. 4.4).

As was mentioned each iteration the heuristic path is calculated according to the current position of the robot and its surroundings. During the calculation of the path, the robot will place additional shape points (where the direction of movement is changed). To avoid the collision robot should take into consideration the safe distance to an obstacle (Fig. 4.5).

In the case of our TVS, all the obstacles can be separated into two classes: positive obstacles (above the level of local zero of robot's coordinate system, Fig. 4.6a) and negative (below the local zero of robot's coordinate system, Fig. 4.6b)

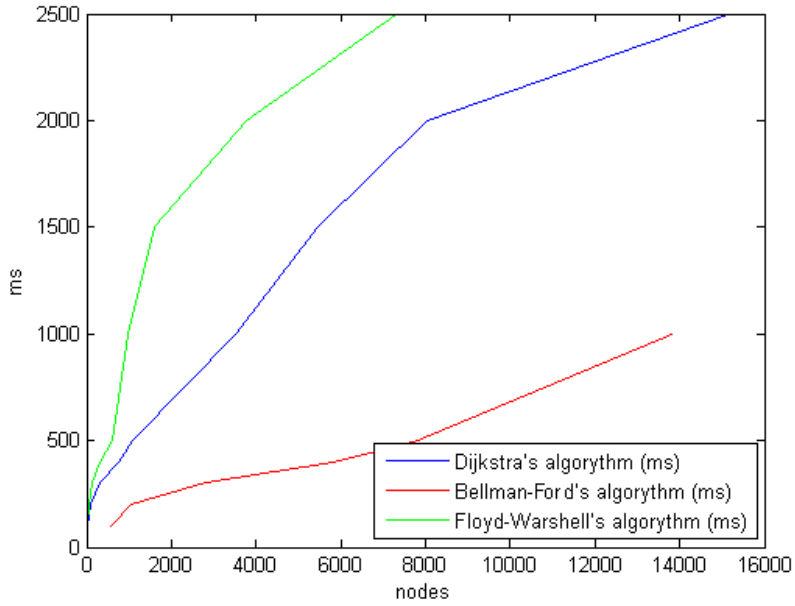


Figure 4.3 Results for "all to all search"

4.4 Collision detection and obstacle avoidance

In the case of our TVS, all the obstacles can be separated into two classes: positive obstacles (above the level of local zero of robot's coordinate system, Fig. 4.6a)

Traditional Vision-Based Collision Detection Methods. the collision detection using on vision-based systems is used in different types of robotic applications [145], [146]. As an example can be mentioned [80], the authors proposed a monocular obstacle detection and avoidance method for UAV. The mathematical model, used in the publication, estimates the relative distance from the UAV's camera to an obstacle in the FOV by finding the feature points. In [81] authors proposed a method for robotic navigation in dynamic environments, the proposed solution is called escaping algorithm it is using the force field method which belongs to the SLAM family.

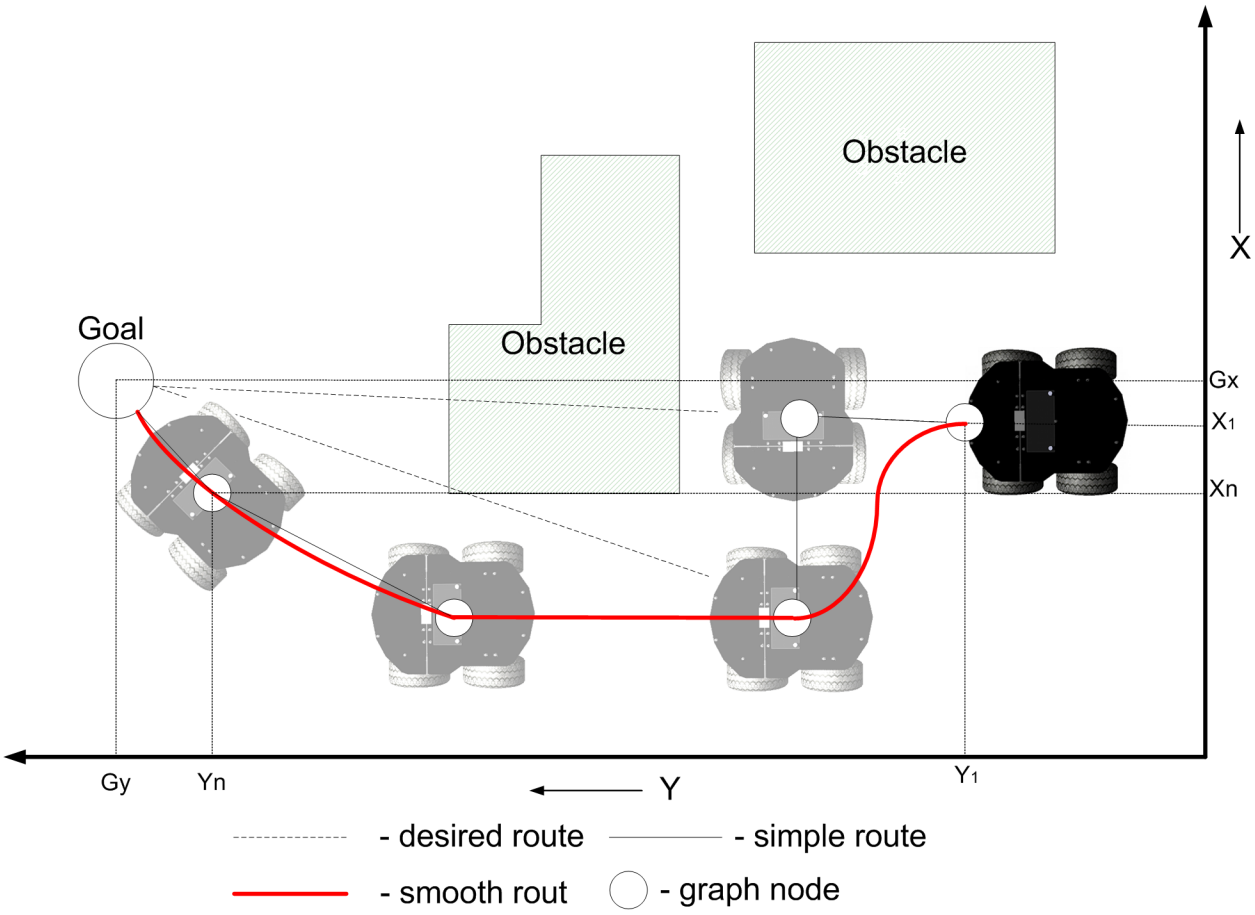


Figure 4.4 Path planning

Bio-Inspired Collision Detection Methods. There are also several bio-inspired collision avoidance and navigation methods, most of which are based on elementary motion detector (EMD), for example, Zhang et al. [82], Badia et al. [83], and Franceschini et al. [84]. However, in many cases, EMD-based methods could be difficult to apply due to its inherent character—the performance is strictly restricted within certain visual speeds. LGMD-based methods, on the other hand, can cope with most of the upcoming collisions, regardless of the visual speed. Blanchard et at. [85] was the first to bring LGMD-based neuron networks into robots for real-time collision detection and tested it with Khepera I robots. Badia et al. [86] proposed one form of LGMD based collision detection model and tested it on a high-speed robot "Strider" with a wireless camera to capture and

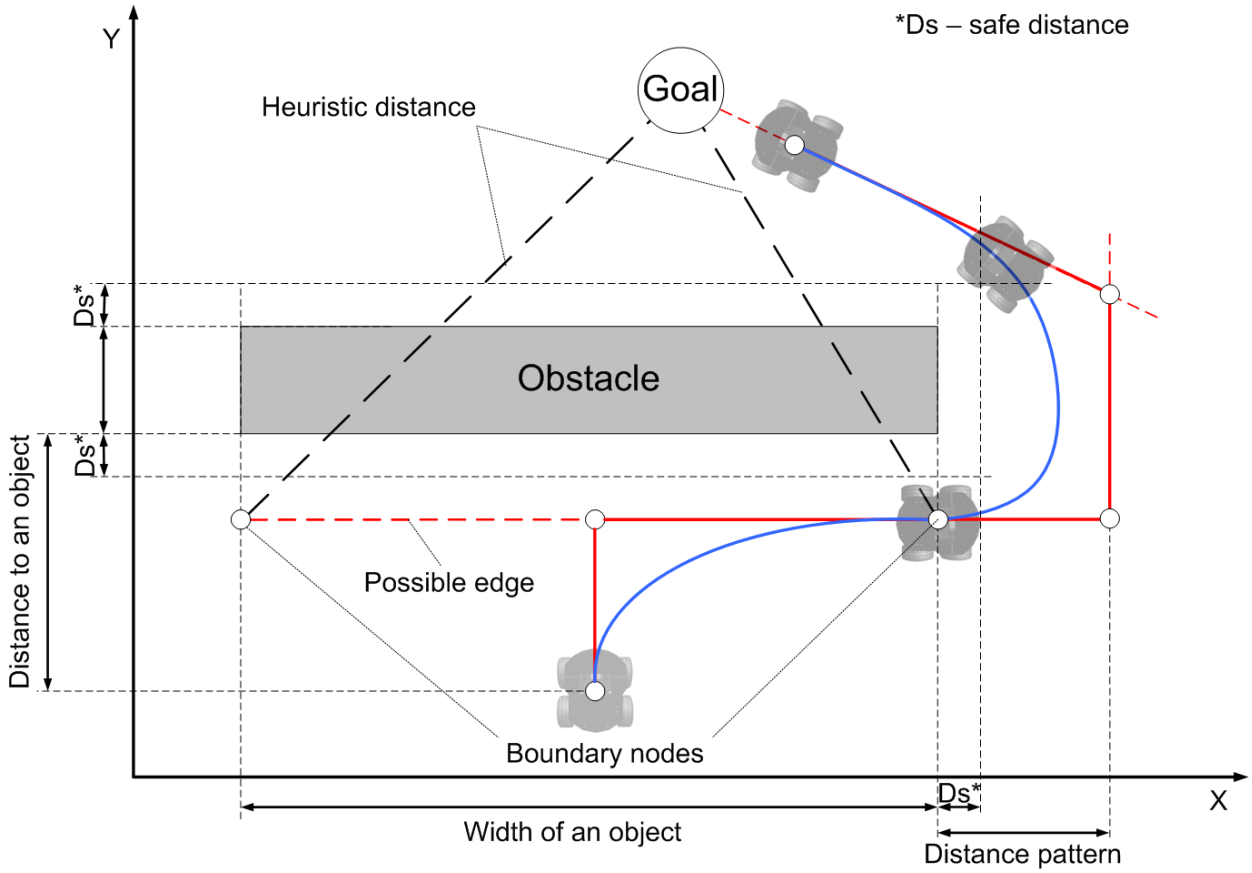
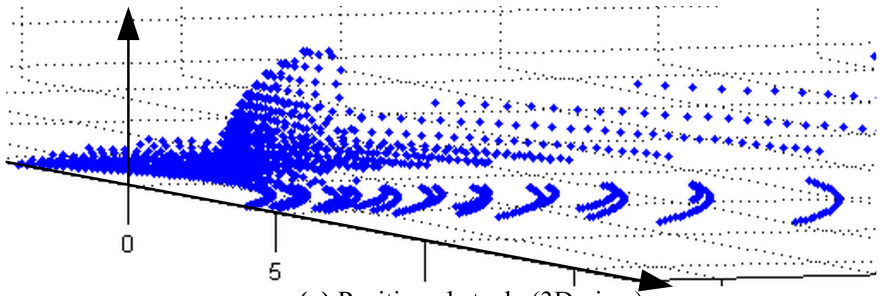
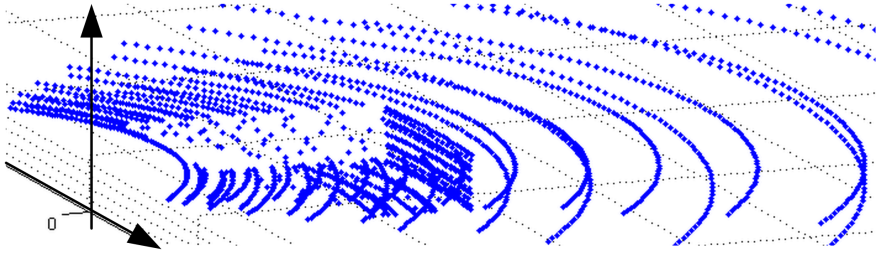


Figure 4.5 Obstacle avoidance

transmit images to a PC for processing. Silva et al. [87] proposed another modified LGMD model which combined two previous works from[88] and [89] for more robust collision detection, which focused more on modeling instead of embedded system development. There has been an effort on implementing the bio-inspired method in very large scale integration chips like field-programmable gate array (FPGA), for example, Meng et al. [89] added additional cell to detect the movement in-depth, Harrison [90] proposed an analog IC for visual collision detection based on EMD, and Okuno and Yagi [91] implemented mixed analog-digital integrated circuits with FPGA. However, these attempts are not suitable for micro and mini-robots, either because of the large size or the high power consumption of the FPGA circuits. and negative (below the local zero of robot's coordinate



(a) Positive obstacle (3D view)



(b) Negative obstacle (3D view)

Figure 4.6 Obstacles types

system, Fig. 4.6b)

According to the principles of TVS and previous research [112], it is expedient to use the algorithm A* [147] as the tool of obstacle avoidance in this research. The terrain can be represented as a matrix where each cell will have a linear size of the robot's half diagonal. Cells are having two possible states: available for the path through (walkable) or saturated by obstacle without the possibility to the path through (non-walkable). Initially, each cell is walkable. After the detection of the obstacle within the cell, it changes the state. All the cells around it also change the state to "non-walkable" in order to create a safe zone and avoid collisions during the robot's turns (Fig. 4.7a). After execution of such operations set, the robot obtains a matrix of surrounding state, where the results of TVS scanning (after z-coordinate omission) are replaced by a matrix of the scanned sector (binary map) with a defined status of all inside cells (walkable/non-walkable, or occupied/empty). Preliminary, the robot's control unit decides the ability to pass the sector under

the attitudinal constraints: it is available if the obstacle height is lower than 0.1 of the wheel radius or it is above the 1.2 of robot's height (basing on the omitted z-coordinate). Such operation permits to simplify the further path-planning, using a reduced 2D model.

A*, according to its classical form [123], performs the wave propagation toward the goal (points of interest), searching for the first best match, which is the set of nearest "walkable cells". This set of cells supposed to be visited by the robot during its movement (Fig. 4.7b). The trajectory in Fig. 4.7b represents the first match solution. However, this solution is not the best due to two reasons: it contains redundant information (5 low-informative points, which are not strictly required for trajectory planning) and non-smooth trajectory in this case, which increase the energetic load on driving mechanism and robot's wheels and which finally leads to undesired additional losses of robot's power source lifetime. To avoid these problems, we propose to perform additional post-processing. For continuous (smoother) trajectories in the first step, we will remove all unnecessary nodes from the trajectory, remaining only nodes where the direction of movement is changed (Fig. 4.7c). In the second step of the post-processing, the path trajectory approximation should be executed (Fig. 4.7d), in order to improve the robot's movement smoothness. In this case, as the approximation, we consider the replacement of the polygonal line with a smooth curve. The selection of the appropriated method of individual trajectory approximation will be provided below. It was done to obtain coherence between the decisions interrelation of the navigation system actions and the ability to anticipate and provide feedback to events with sufficient speed.

The present research was reviewed and compared with several methods of approximation: point to point [98], Beziers approximation and Dubins path.

In [98] was shown that ten points are enough for building a smooth trajectory. This amount

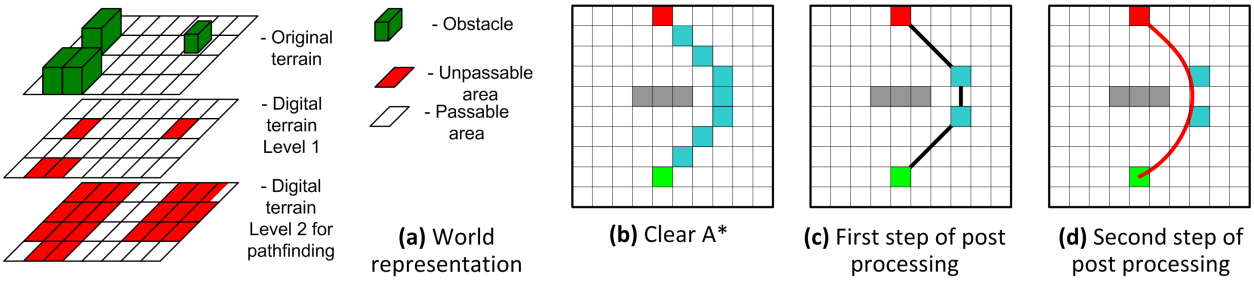


Figure 4.7 World representation and dead reckoning with two-step post processing

of data requires a repetitive calculation of the movement vector at every point, which is time-consuming. One of the simplified solutions is to use fewer points with Bezier curve as an approximation function ([148], [149]). Such approach is useful and found its application in various tasks of path-planning for autonomous vehicles [150] and [151]. Mathematical parametric representation of a Bezier curve has the form:

$$P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t), \ 0 \leq t \leq 1 \tag{4.1}$$

where t is a parameter, n is the degree of Bernstein's polynomial basis, i is the summation index, $J_{n,i}(t)$ basis functions of the Bezier curve, also known as Bernstein basis polynomials of degree n, and B_i represents the i-th vertex of the Bezier polygon.

The i-th Bezier or Bernstein function (approximation function) in Eq. 4.2 is given as:

$$J_{n,i}(t) = \begin{pmatrix} n \\ i \end{pmatrix} t^i (1-t)^{n-i}, \tag{4.2}$$

with

$$\begin{pmatrix} n \\ i \end{pmatrix} = \frac{n!}{i!(n-i)!} \tag{4.3}$$

Bezier curve equation can be written in matrix form, as well as the equation for cubic spline interpolation and parabolic:

$$P(t) = [T][N][G] = [F][G] \tag{4.4}$$

$$[F] = [J_{n,0},\ J_{n,1},\ \ldots,\ J_{n,n}] \tag{4.5}$$

$$[G]^T = [B_0,\ B_1,\ \ldots,\ B_n] \tag{4.6}$$

where T – vector of time, N – Bezier's basis matrix, G – matrix of vertices. In general case for Eq. 4.4:

$$[T] = [t^n,\ t^{n-1},\ \ldots,\ t^1] \tag{4.7}$$

$$[N] = \begin{bmatrix} \binom{n}{0}\binom{n}{n}(-1)^n & \ldots & \ldots & \binom{n}{n}\binom{n-n}{n-n}(-1)^0 \\ \binom{n}{0}\binom{n}{n-1}(-1)^{n-1} & \ldots & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \binom{n}{0}\binom{n}{0}(-1)^0 & \ldots & \ldots & 0 \end{bmatrix} \tag{4.8}$$

Let us compare an example represented on Fig. 4.8 Bezier curve based on three points. A and B are initial and final points. Point C is a virtual temporal point to make a curve. Matrix [N] for

this case written in Eq. 4.9.

$$[N] = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{4.9}$$

The results of calculation are shown in Fig. 4.8 (Fig. 4.8a shows 3D view, Fig. 4.8b – 2D view). It proves that in order to smooth the trajectory only three points are need.

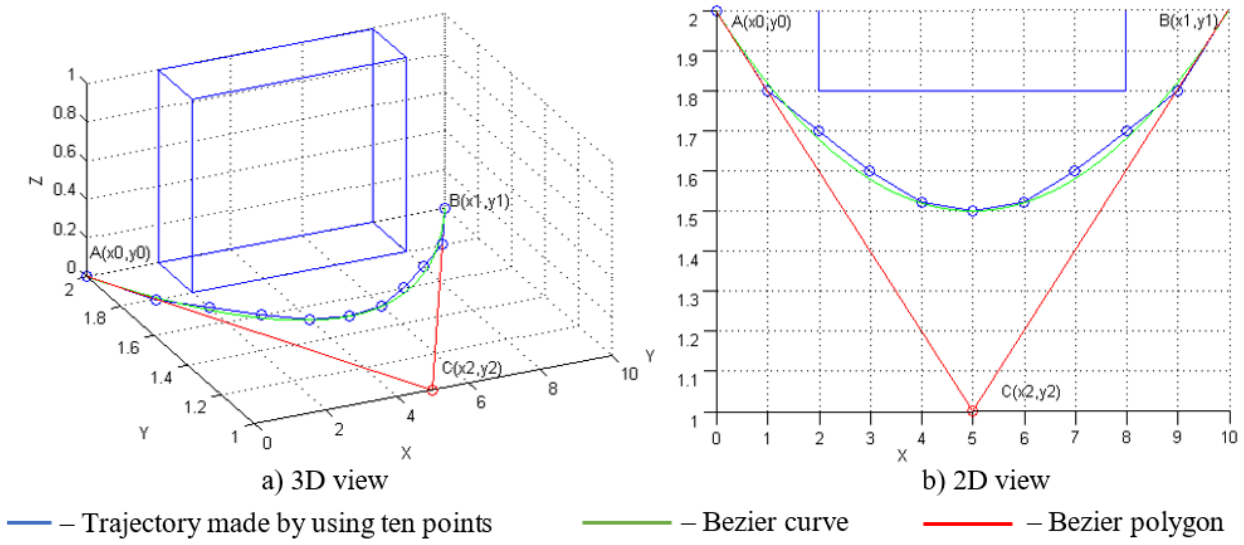


Figure 4.8 Obstacle avoidance with Bezier curve

Lets return to example represented on figure Fig. 4.9. Using the calculation based on three points (Fig. 4.9a) helps to avoid movement in situ on point B but not on point A. That is why will be used the Bezier matrix for four-point (Eq. 4.10). Result of calculations is shown on figure Fig. 4.9b. By adding temporal points C and D trajectory becomes smooth and it avoids the need

for movement in situ on all parts of the trajectory.

$$[N] = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.10}$$

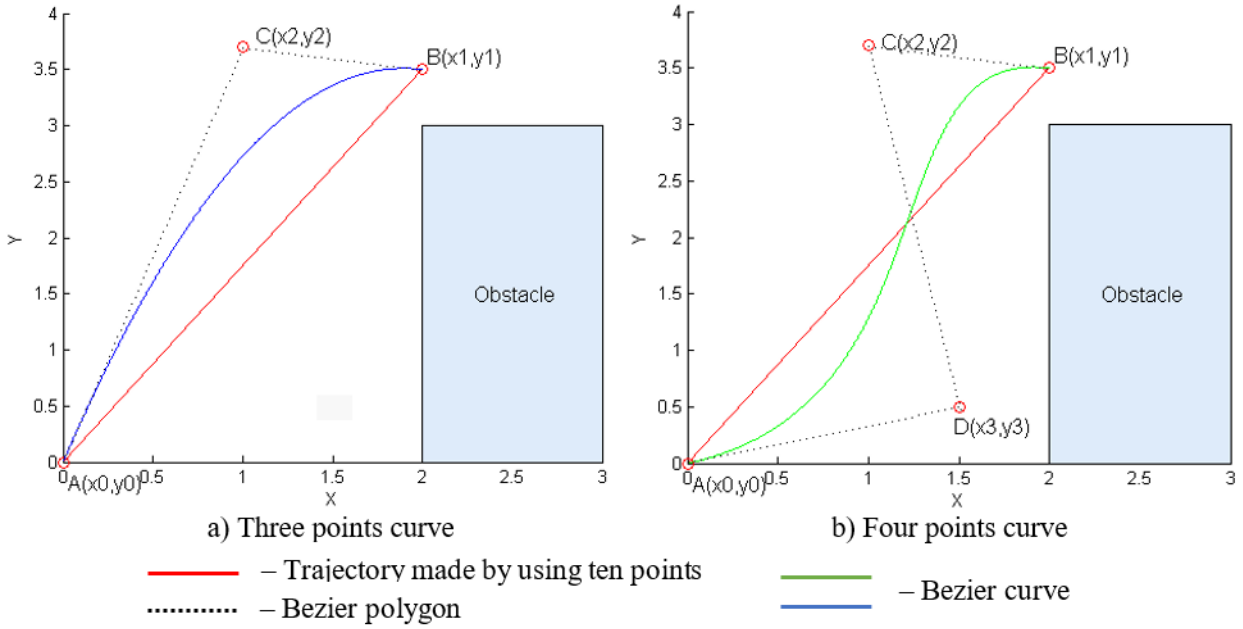


Figure 4.9 Obstacle avoidance with Bezier curve

It this case only three points are needed to get the smoothen trajectory. As an example, let us solve the same task using a Dubins path [152]. The following equations describe it:

$$\begin{cases} x = V * cos\,(\theta) \\ y = V * cos\,(\theta) \\ \theta = u \end{cases} \tag{4.11}$$

where (x, y) is the position of the robot, θ is the heading, a robot is moving using the constant speed V. The optimal path type always is based on the robot's ability to perform the next basic

actions: “right turn (R)”, “left turn (L)” or “driving straight (S)”. Every time the optimal path will be one of the next six types: RSR, RSL, LSR, LSL, RLR, and LRL (Fig. 4.10).

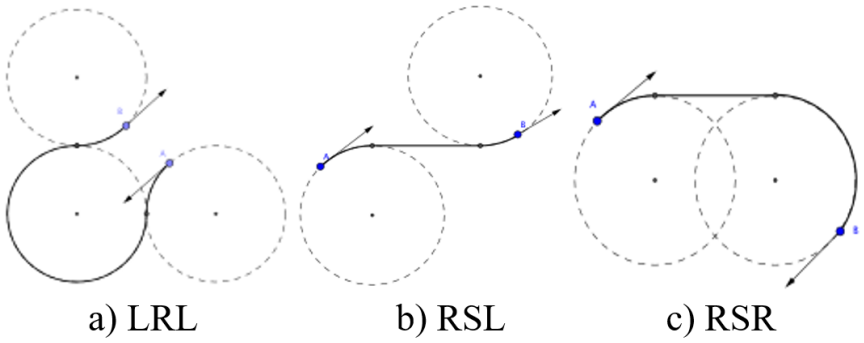


Figure 4.10 Obstacle avoidance using Dubins path

Using the Dubins path (Fig. 4.11) the distance becomes longer and at the initial point, the robot by default is in an inconvenient position to start a movement along the recommended trajectory (additional inconvenient movement required). Making a detailed comparison of these two algorithms, the result of the path made by Bezier approximation is 10.3% – 12.7% more effective than the solution of the same situations using Dubins path.

Further, these methods were compared using the parameters of total path length and trajectory smoothness, represented as an amount of trajectory bending energy [153].

The bending energy is a curvature function k, used to estimate the smoothness of the robot’s navigation trajectory. Bending energy is calculated as the sum of the squares of the curvature at each point of the line along its length. The bending energy of the trajectory of a robot is given by:

$$BE = \frac{1}{n}\sum_{i=1}^{n} k^2(x_i, f(x_i)) \tag{4.12}$$

where: $k(x_i,y_j)$ is the curvature at each point of the robot’s trajectory and n is the number of points within the trajectory. Here the curvature of the trajectory at the n-th point is the inverse

value of the circle radius built to the point of arc where it moves along at the given instance of time. The results are shown in Tab. 4.1. The 10 point curve was taken as the basis of comparison (all represented as 100%). The summarized results are represented as an average value between the total length and bending energy characteristic. In other words, each of them has the same weight for calculation.

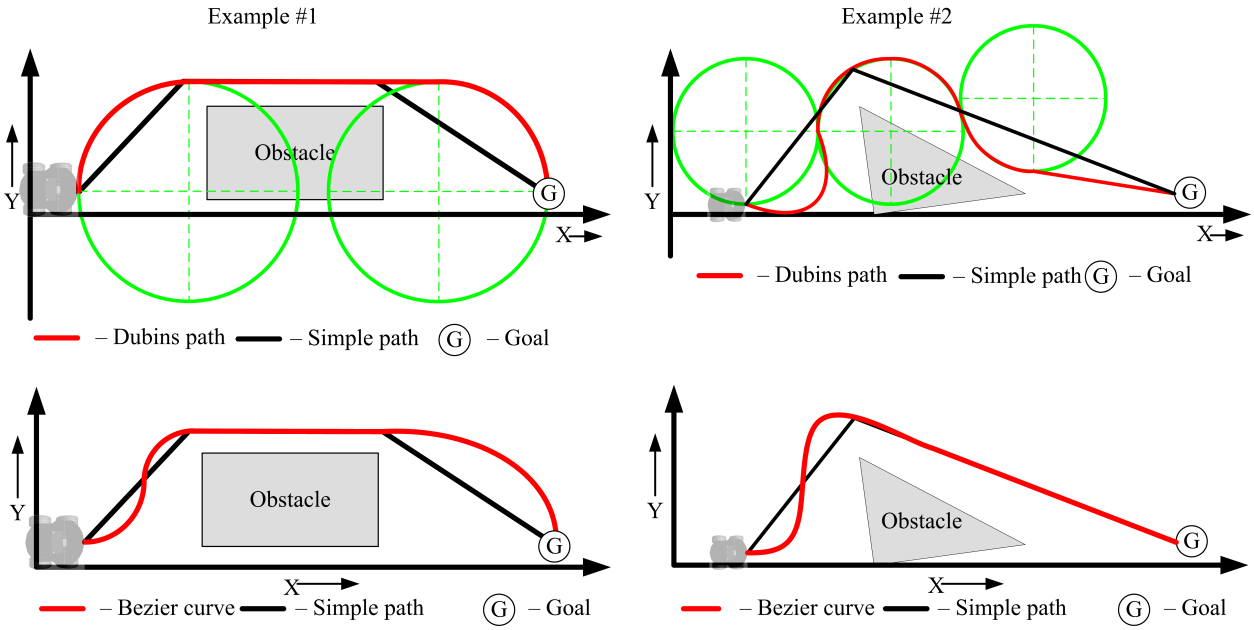


Figure 4.11 Obstacle avoidance with Dubins path and Bezier curves

Table 4.1 Motion planning comparing results

	10 points curve	Dubins path	Bezier with 3 points	Bezier with 4 points
Total length	100%	141%	113%	105%
Bending energy	100%	17%	75%	29%
Total	100%	79%	94%	67%

Comparing three motion planning methods (Dubins path, Bezier approximation using 3 and 4 points) we can get the next conclusions. The Dubins path received the highest length, but the best bending energy saving. The Bezier approximation using 3 points polygon shows the worst

ability for optimized path planning. Ultimately, Bezier approximation using 4 point polygon gives the most satisfactory average result. In other words, its application permits to get the minimally extended trajectory, providing at the same time the maximum savings of the robot's source and ware of mechanisms.

4.5 Section conclusion

In general, these methods are solving the task of motion planning for the independent robot in a group. It is obvious that data exchange between the n robots in a group is a good tool to obtain additional information. It can serve for more efficient implementation of all mentioned above methods. The main idea is to give each individual robot in a group more knowledge about the sector as quickly as possible. Moreover, in some cases, a certain portion of information can be unavailable for the i-th individual from its own position.

Chapter Five

Simulations and experiments

5.1 Modeling system structure and group behaviour

5.1.1 Basic behaviour scenario for robotic group

Swarm robotics is a new approach to the coordination of multi-robot systems. In contrast with traditional multi-robot systems which use centralised or hierarchical control and communication systems in order to coordinate robots' behaviours, swarm robotics adopts a decentralised approach in which the desired collective behaviours emerge from the local interactions between robots and their environment. Such emergent or self-organised collective behaviours are inspired by, and in some cases modelled on, the swarm intelligence observed in social insects.

The potential for swarm robotics is considerable. Any task in which physically distributed objects need to be explored, surveyed, collected, harvested, rescued, or assembled into structures is a potential real-world application for swarm robotics. The key advantage of the swarm robotics approach is robustness, which manifests itself in a number of ways [154]. Firstly, robots within the swarm are mainly represented as a group of similar robots, initially, they do not have a dedicated

task, however after robots are deployed the swarm is able to reorganize itself taking into account the work they need to accomplish. Secondly, furthermore, the swarm approach has deep tolerance to the failure of individual agents. Thirdly, the swarm has a decentralized control, it means that robots have no common point of failure or similar type of vulnerability.

The acknowledgment of the capability of swarm robotics requires to solve of various extremely hard issues [155]: algorithm design, implementation and test, analysis and modelling.

Algorithm design: developers have a problem to design the hardware part (physical model) and behavior (logical/software part) of individual agents by providing the ability to establish communication within the swarm, interaction with surroundings and ability to interact with each other if necessary. Only then the desired collective behavior can be achieved.

Implementation and test: within the laboratory is required to build an experimental infrastructure. Experiments during this stage are used with simulations.

Analysis and modeling : create a behavior model for robotic swarm considering parametrical optimization and validation is a difficult task cause of is a stochastic, non-linear origin. Such models would definitely be a basic piece of developing a security contention for real-world applications.

This section will cover possible solutions for each of the problem and review their implementation according to the machine vision selection, methods of data transferring (communication) and navigation.

Based on the tasks presented in previous sections can be establish conceptual diagram for robotic group behaviour Fig. 5.1. The behaviour concept consists from next main steps:

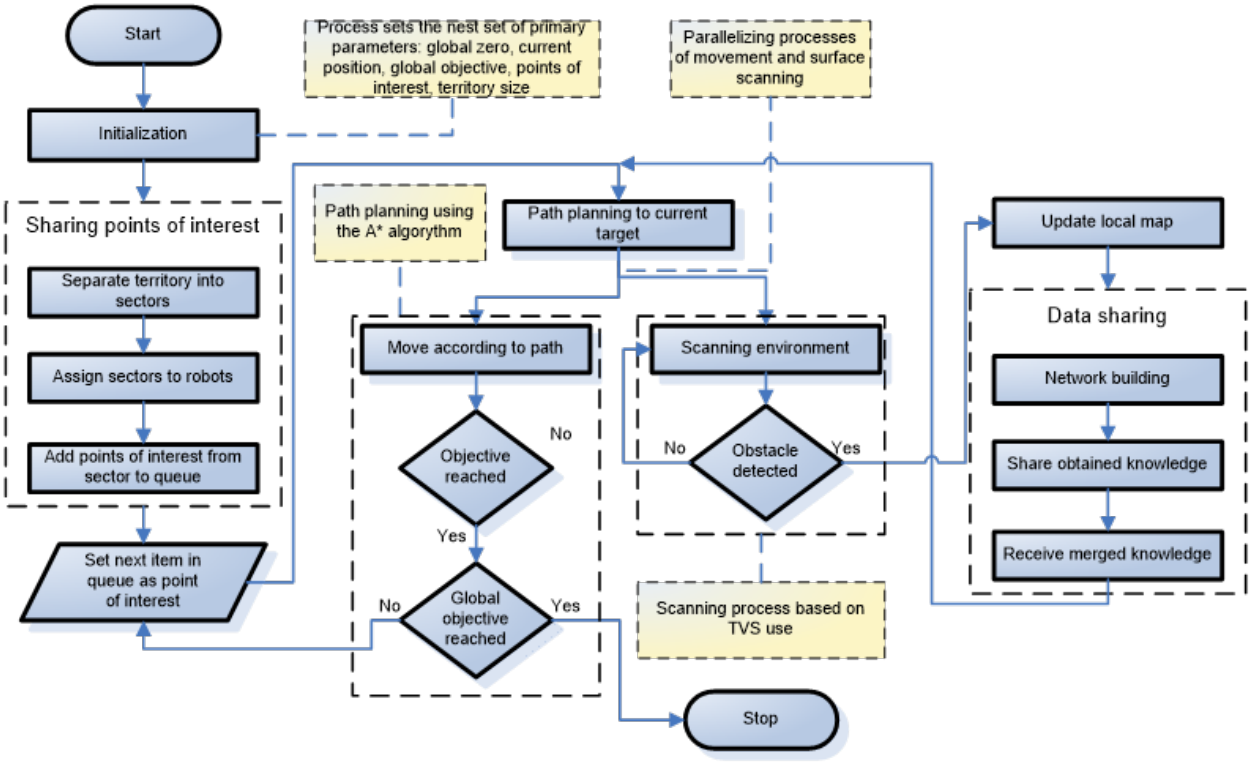


Figure 5.1 Behaviour model of robotic group

Initialization – the first step of the behavior algorithm. Here each robot sets the default values for the main parameters. Some of these parameters are predefined initially in the system by a human operator and they are used as constants. These parameters are included the robotic group size, terrain and its dimensions, speed limits, resolution, accuracy zones radiuses, and objective locations (defined automatically or manually points of interest). Other parameters are defined by robots individually (global zero and current position).

Sharing points of interest – on the previous step of initialization robots have the list of points of interest. Here they applied the logic for sectoring the terrain (n sectors) between the agents/robots (n robots) within the group. The corresponding points of interest of the i-th sector will be added to the queue for visiting of the i-th robot.

The next three steps of the algorithm are going in parallel.

Path planning – includes the process of navigation, localization, and motion planning. Here robot calculates the path using the A* between the current position and the next point of interest in a queue.

Environment scanning – on this stage robot is using 3D technical vision system for obtaining Cartesian coordinates and updating the existing map stored in the memory by creating projectiles on the matrix representation of a map.

Communication – the stage where a group of robots is involved in the data exchange process and data merge. Hero robots are improving their environmental knowledge using data of there "teammates".

According to the algorithm (Fig. 5.1) can be created the combined solution for robotic group and presented as pseudo code (Fig. 5.2).

```
foreach r_i in G
   Initialize v_i, a_i and l_i for robot
   Initialize map M = [t_11, ..., t_1a; ...; t_b1, ..., t_ba]
end foreach
Split T in s parts → T = {t_1, ..., t_s}
Generate secondary objectives locations
X_i = [x_i1, ..., x_im] for each robot r_i in G
X_i add x_m → X_i = [x_i1, ..., x_in, x_im]
while x_m not reached
   foreach r_i in G
      if received data exchange request then
         Initiate DataExchange (Algorithm 1)
      end if
      get frame F from TVS
      F = [f_11, ..., f_1r; ...; f_q1, ..., f_qr]
      foreach f in F.
         c ← C(f, dimensions(M))
         M(c) ← not walkable
      end foreach
      obstacle ← D({x_i,y_i,z_i}, F)
      if obstacle ≤ Z_low then
         {v_i, a_i} ← {S_low, R_low}
      end if
      if Z_low < obstacle ≤ Z_avg then
         {v_i, a_i} ← {S_avg, R_avg}
      end if
      if Z_avg < obstacle ≤ Z_high then
         {v_i, a_i} ← {S_high, R_high}
      end if
      Initiate DataExchange
      path ← A(T, x_ij)
      if x_ij is reached then
         x_ij ← pull (X_i)
      end if
   end foreach
end while
```

Figure 5.2 Basic algorithm of robotic group (pseudo code)

Behaviour of robotic group for the modeling system can be described by using the next set of variables:

- Robotic group: $G = r_1, \ldots, r_s$;
- Group size: s
- Terrain: T;
- Terrain dimensions: d
- Speed and limits: S_{low}, S_{avg}, S_{high};
- Resolution: R_{low}, R_{avg}, R_{high};
- Accuracy zones radiuses: Z_{low}, Z_{avg}, Z_{high};

- Main objective locations: x_m;
- Secondary objectives locations $X_i = [x_{i1}, \ldots, x_{in}]$;
- Pathfinding function: $A(T, x_{ij})$, where $T = t_1, \ldots, t_s$;
- Distance to obstacle function: $D(l_i, F)$, where F -– frame obtained from TVS;
- TVS to map coordinates function: C(x,y,z, b,a), where x,y,z – Cartesian coordinate from TVS, b,a – dimensions of map;

It considers an initial robotic group G. Each robot is described as a set of Cartesian coordinates, velocity v_i and scanning accuracy (resolution) a_i. For each robot, the algorithm initializes these variables and maps (lines 1-4). Next, according to terrain dimensions, it splits the map into equal sectors (line 5). Each of i robots for own sector creates the queue of secondary objectives X_i (for optimal trajectory planning) and pushes to the end of the queue the main objective x_m. While the robot did not reach the main objective in the queue, it will repeat the lines 10-34. The robot constantly provides the feedback for data exchange process initiation (lines 11 - 13), if such exchange requested it participates in the process. After visual data frame obtaining from TVS (lines 14 - 15) it reviews each point in the frame and sets as non-walkable all corresponding cells in the map (lines 16 - 19).

After obstacle detection (line 20) the algorithm uses a resolution and speed control based on the accuracy zone range (lines 21-29). Then the robot initiates data exchange to update and share its knowledge about the environment (line 30) with other group members, gets the path to the current objective in the queue (line 31). If the current objective is reached, it pulls next from the queue (lines 32-34)

5.1.2 Simulation frameworks

Before every complex mechanical system practical implementation it has to go throw two stages: theoretical justification and realistic simulation. Create a digital model of the entire system significantly affects the overall efficiency of the project. Process of simulation gives an opportunity to reduce the mistakes during the development, improve the output of the system according to the changes in environmental conditions and reduce costs of technical issues. Among other benefits of simulation are:

- Reduce the cost of manufacturing robots;
- Resource management and source code diagnostics;
- Simulate different alternatives;
- Before the implementation of the robot its components can be verified;
- Modeling can be done in stages, for complex projects favor;
- To determine the viability of the system;
- Compatibility with a wide range of programming languages;

However, disadvantages of simulation also can be found. An application can simulate the robotic behavior model only with predefined rules, it will not mimic factors that are not taken into account in the development phase. Real world experience can provide more scenarios than the computational model.

Nowadays simulation platforms covers a lot of tools and features that can provide close to real life simulations. Most of the use different C/C++ like algorithmic languages, LabVIEW, MATLAB and etc. In this section, several used simulation platforms are summarized.

Player/stage [156] is a project under which three robotics-related software platforms are currently being developed. It consists of the Player network robotics server, the Stage-2D robot simulation environment, and the Gazebo-3D robot simulation environment. The project was founded in 2000 by Brian Gerkey, Richard Vaughan and Nathan Koenig at the University of Southern California in Los Angeles, and is widely used in research and training within robotics.

The UberSim [157] is a simulator designed for quick testing before uploading program to football robot. UberSim uses the ODE physics engine. Software supports custom robots and sensors.

USARSim [158] is a simulation of urban search and rescue. It is based on Unreal Engine 2.0. USARSim can be interfaced with Player and runs on Windows, Linux and MacOS.

Breve [159] is a 3D simulation environment for distributed artificial life systems. behavioral models are defined using Python. Like UberSim, Breve uses ODE physics engine and OpenGL for 3D graphical representation.

V-REP [160] is a useful 3D simulator for educational process, allows modeling of complex systems, individual sensors, mechanisms, and so on.

Webots [161] is a software product of Swiss company Cyberbotics. It provides supports of different programming languages like C/C++, Java, Python, URBI and MATLAB. Moreover is compatible with third-party software through TCP/IP.

Gazebo [162] can simulate complex systems and a variety of sensor components. It is helpful in developing robots used in interaction, to lift or grab objects, to push and activities that require recognition and localization in space.

ARGoS [163] is a modular, multi-engine simulator for heterogeneous swarm robotics. System is able to use about 10,000 wheeled robots in real-time during the simulation.

TeamBots [164] is a multi-agent simulation program for robots that allows you to create multi-agent control systems in dynamic environments with visualization. You can develop your control system and implement it in a simulation program, and then test your control system in a real mobile robot.

To prove the theoretical basis of presented questions was used developed software for the simulation and robotic group collaboration. Presented framework has been developed in Unity 5 ([165],[62]), it is a multiplatform engine provided with different features and tools. Software was developed using programming language C# in MonoDevelop integrated development environment (IDE) for Windows 10. Software (Fig. 5.3) has three operating modes "Without common knowledge", "Pre-known territory", "With common knowledge". First two use only part of decision-making system for path planning and obstacle avoidance. The third one implements a full stack of decision-making process. In the end, the system returns data about environment and state of each robot in every moment of time.

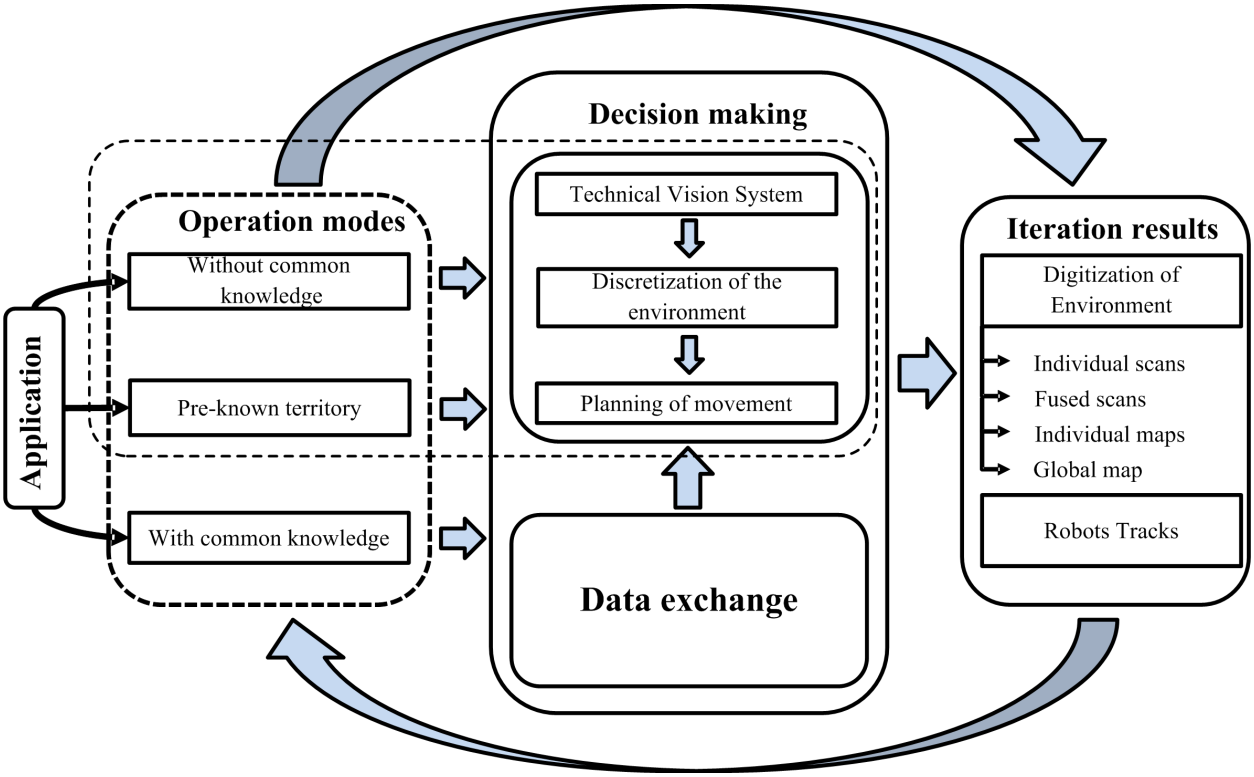


Figure 5.3 System structure

5.2 Robot entity

Robot within the system can be described with a set of variables (rotation speed, speed of movement, current position and spatial orientation), goal position and decision making system (Fig. 5.4).

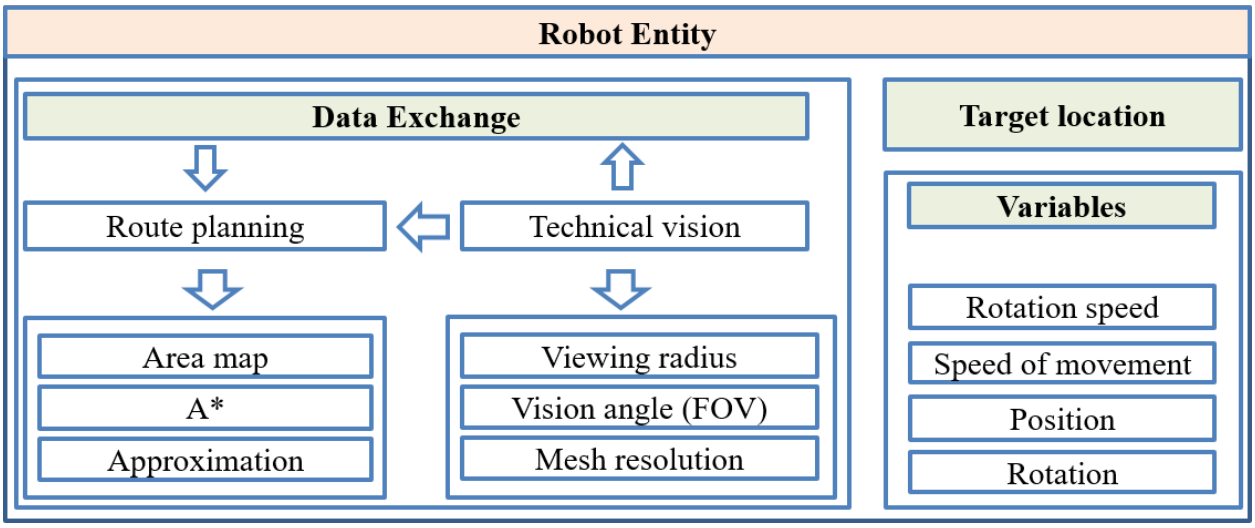


Figure 5.4 Robot entity

In current simulations were used four different random scenes presented in Fig. 5.6. Modeling includes the group of three robots using Pioneer 3-AT and TVS (Fig. 5.5). This robotic platform and its kinematic were reviewed previously in work [98]. The TVS used in the research had already a practical implementation and solid results presented in articles, during modeling the dynamical triangulation method will be simplified for the purpose of more efficient analysis of other problems.

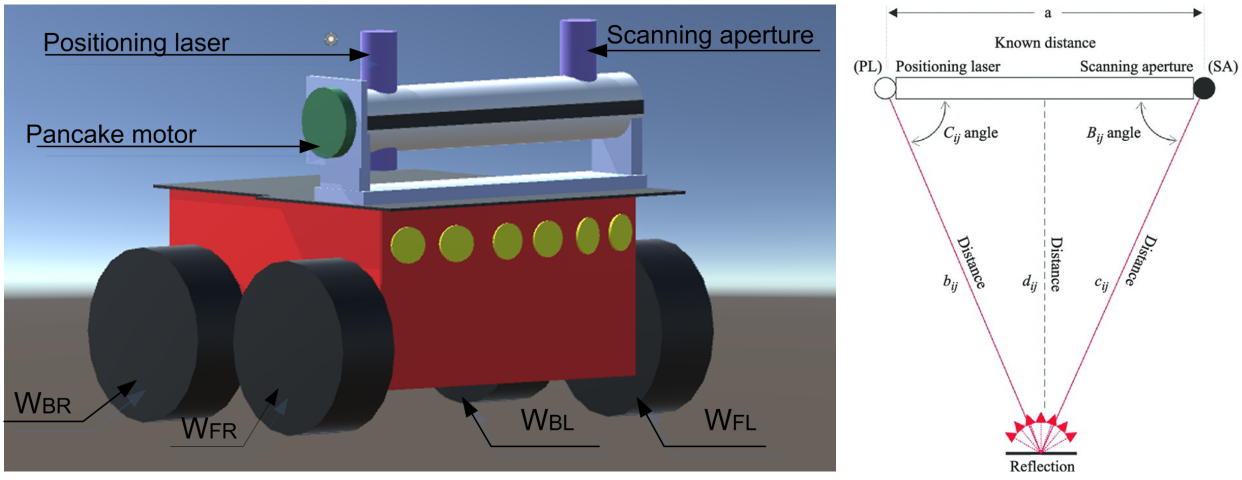
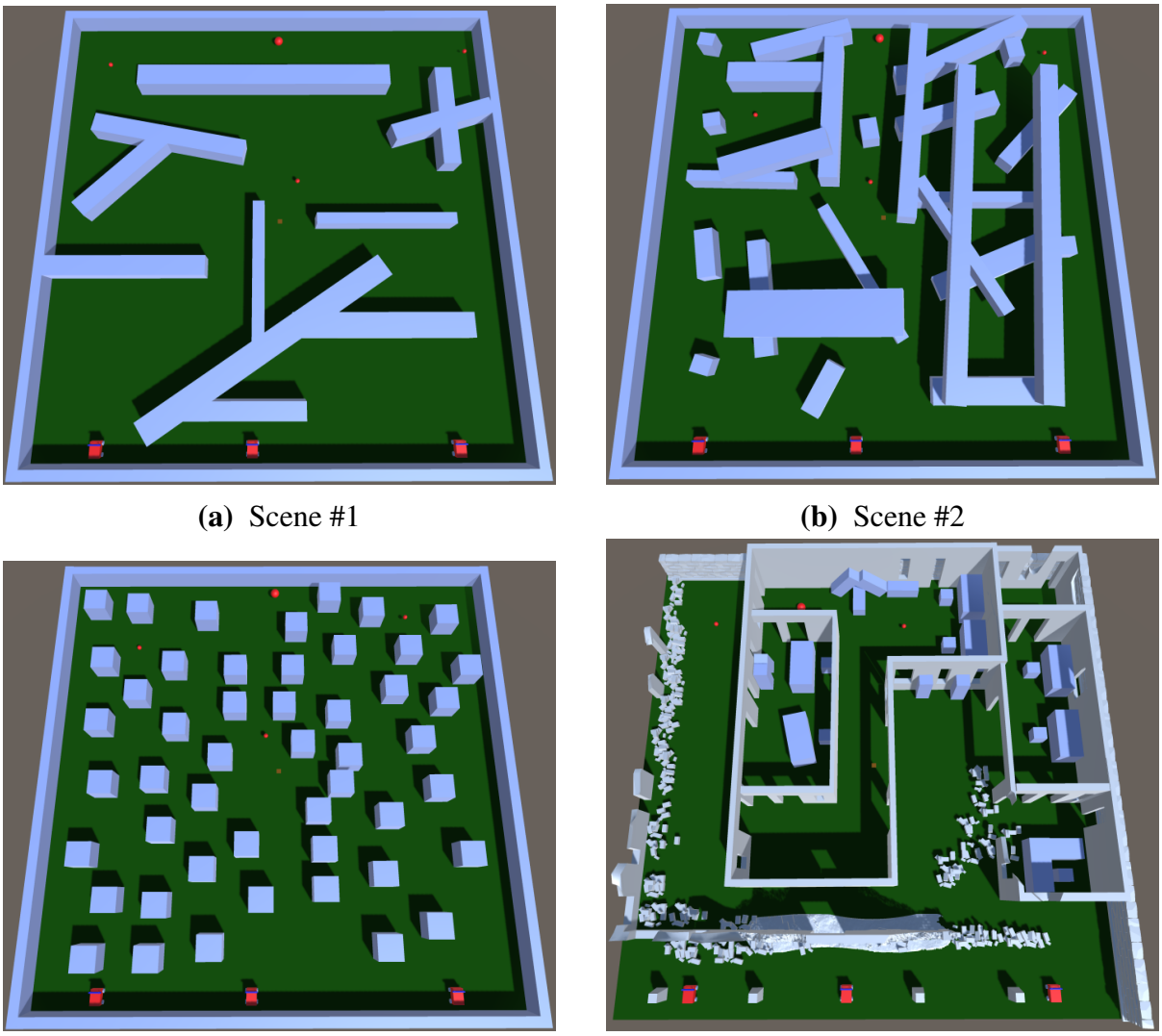


Figure 5.5 Pioneer 3-AT mobile robotic platform

5.3 Influence of data exchange on path planning

To receive the results for each scene three scenarios were modeled. In the first scenario robots are moving independently among each other (no knowledge sharing and data exchange) – “no common knowledge”. In the second scenario (“with common knowledge”) three robots are fusing obtained information and are using common knowledge about terrain for path planning (implemented data exchange method). In the last scenario, the group is moving with the map of terrain – “with the pre-known territory”. In each scenario, robots have to reach their personal goals and then get to a common point.

To make the modeling were used four different scenes of cluttered environments (Fig. 5.6). First two scenes (Fig. 5.6a and Fig. 5.6b) considers closed labyrinth-like environment with randomly placed walls, the difference between two of these is that in second scene used the second level of walls placed over the top of the first. Third scene (Fig. 5.6c) is closed to cluttered warehouse and the last one (Fig. 5.6d) reviews office building after natural disaster.



(a) Scene #1 **(b)** Scene #2

(c) Scene #3 **(d)** Scene #4

Figure 5.6 Scenes used for modeling

For all of the scenarios in each scene, 100 simulations were executed. Each simulation in the start point uses the same initial parameters (position, velocity, the accuracy of scanning, main and secondary goals) of robots. The results of the modeling and accumulated data are represented in Fig. 5.7

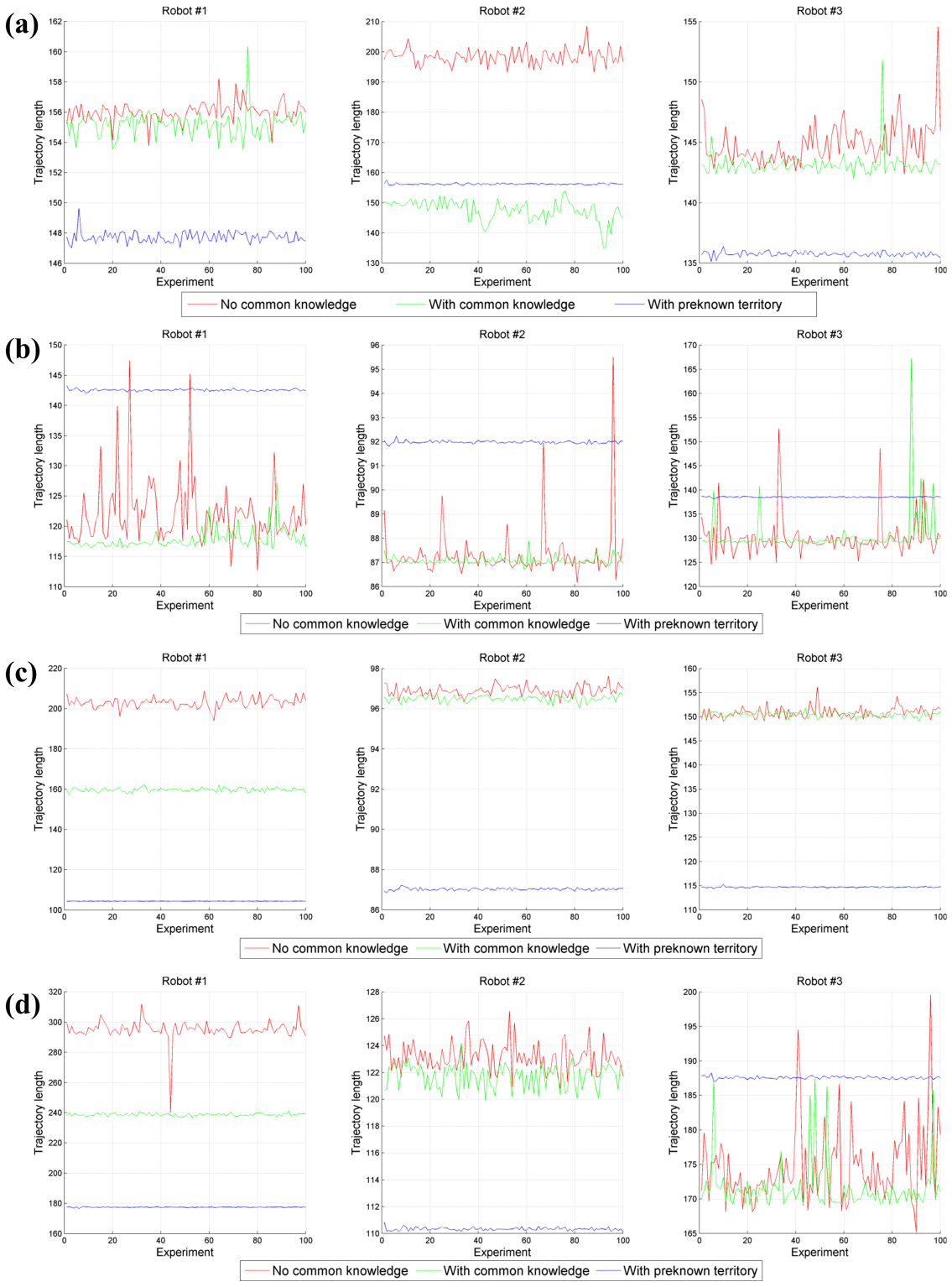


Figure 5.7 Length of trajectories

Reviewing the obtained results, it becomes visible on the graphs that exist trajectory lengths

deviation for each robot in all three scenarios (Tab. 5.3). For example Robot #1 in Fig. 5.7b for the case of "no common knowledge" has a deviation of 4.32%. It means that for each of the modeling iterations the robot had different routes to achieve its goals. In general, its average route maintains the same. Still for some cases, shown as high peaks in Fig. 5.7b, the route becomes an outlier, in regard to the average route length.

Another observation can be mentioned where pre-known territory does not always give a shorter trajectory length. This can be found in Fig. 5.7a (Robot #2), Fig. 5.7b (Robot #1, Robot #2, Robot #3), Fig. 5.7d (Robot #3) for the case "with the pre-known territory".

Both mentioned observations are caused by specialties of applied A* algorithm and, for the first case, by unknown territory uncertainties. According to the principle of used TVS, as mentioned earlier in Section 2, for the dead reckoning was decided to adopt the A* algorithm. Its heuristic nature and structure provides a suitable solution for our needs in path planning. The matrix-based orientation of the algorithm solves the task of discrete mapping used for robots dead reckoning and gives next benefits: fast conversions of coordinates obtained from TVS to path planning map reduces the need to post-process the point clouds from TVS to map in case of data exchange and gives a faster calculation time, compared to Dijkstra algorithm [166] for example.

Implementation of the common knowledge base decreases trajectories deviation and tends them to the optimal solution (not taking into account the individual anomalies that have occurred (peaks on Fig. 5.7a and Fig. 5.7b, Robot#3 "with common knowledge")). TVS cause all the deviations in robots routes, mostly because of the order of obstacle detection.

The summary of the modeling is presented in Tab. 5.3 and Fig. 5.8. Comparing averaged distances obtained during the modeling can be observed that the use of a common knowledge base

has advantages in all of the scenes. The result shows that RG with implemented data exchange method has averaged group trajectory length shorter from 6.2% (Tab. 5.3 Scene #1) up to 10% (Tab. 5.3 Scene #4), comparing to distances of individual autonomous robotic trajectories (when using non-group movement). Under the group trajectory length, a sum of the individual trajectories was considered. Scaling the results for individual robots in-group the improvement of trajectories length can reach up to 21.3% (Tab. 5.3 Scene #3, Robot #1).

Table 5.3 Motion planning comparing results (the normalized units of framework used for distances)

	Scene #1			Scene #2			Scene #3			Scene #4		
	Solo	Group	Ratio	Solo	Group	Ratio	Solo	Group	Ratio	Solo	Group	Ratio
Robot #1	155.99	155.17	0.0053	122.02	117.7	0.035	202.97	159.72	0.213	295	238.6	0.19
Robot #2	198.46	172.68	0.13	87.28	87.07	0.002	96.9	96.5	0.0042	123.2	121.7	0.012
Robot #3	147.76	143.16	0.0312	149.69	130.46	0.13	150.83	150.45	0.0025	174.6	171.6	0.018
Total	502.21	497.86	0.062	359.0	335.32	0.066	450.71	406.67	0.098	592.8	531.9	0.1

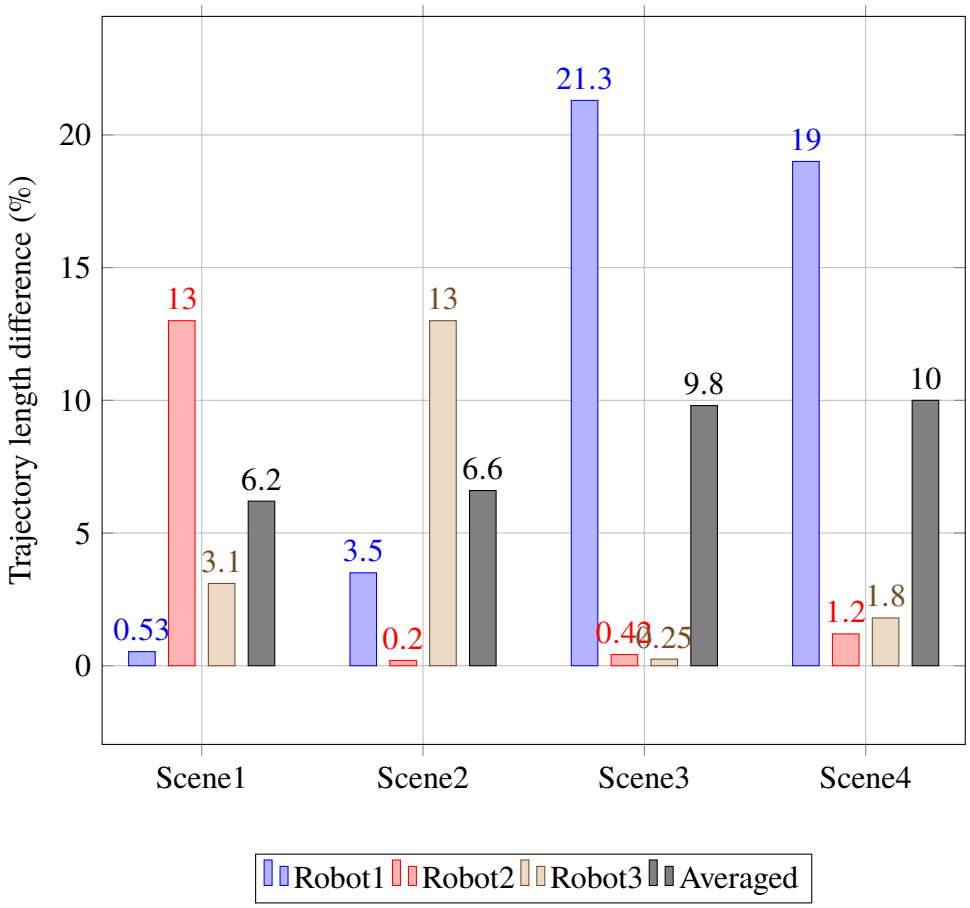


Figure 5.8 Comparing trajectory lengths for each of the scenes in percent

Now it's obvious that the use of the data exchange method for merging individual knowledge into common has a positive influence on robotic group movement and dead reckoning. Another question occurs: What is the effectiveness of individual robots in the group while sectoring the terrain in case of a dispersed initial placement (sectoring the terrain).

The most important goal of the present research is to offer the best solution to the stated problem. Fig. 5.7 in general shows that our solution (green graph, applying the common knowledge obtained by fusion of scanned data from n group members) always propose appropriated solution in regard to path planning, which sometimes even can be better than "almost ideal case", when the configuration of environment is preliminary known (particular case b) in Fig. 5.7. This case

corresponds to the practical situation when the real scene is cluttered by a mix of continued and small obstacles randomly placed within the considered frame.

In our opinion, such a variant is most probable in real life for the considered task. Moreover, in the considered task the existence of preliminary known environment (inspection after disasters) is almost impossible. So, we consider this circumstance as one of the significant obtained contributions in our research.

Regarding the reasoning of such unexpected behavior of our algorithm, we can mention the next. Search A* (in a pre-known territory) in computer science and mathematics, is the search algorithm for the first best match on a graph that finds the least cost route from one vertex (initial) to another (target, final). The particular Scene#2 in Fig. 5.7(b) demonstrates this principle: in the case of "With pre-known trajectory" the algorithm does not consider all the possible set of trajectories, but stops on the first arbitrary obtained an appropriated solution (by the nature of any heuristic algorithms), i.e. robots are not using the closest path but the first best match existed on the map in current environment. However, such a conclusion based only on simulation analysis cannot be final, for its proof strongly requires real experimentation, because any complex phenomena in real life can have many intercrossed reasons or sub-components.

5.4 Effectiveness of robotic group

5.4.1 Terrain sectoring

Laser scanning TVS, as an alternative to camera vision, can give the exact coordinates of any selected point on the surface of the obstacle. However, they cannot process all the surfaces at the

same time and require a certain time to scan a 3D sector. The good solution to this problem is to split the terrain into sectors, sharing the task among robots in the group. Such a distributed scanning will give more explicit information about the position of obstacles inside this sector, and possible dead-ends for robots, which maybe cannot view this sector part from its current position.

According to the TVS specifics, the RG will split terrain into sectors for their movement. To back this up the example (Fig. 5.9) will be considered. While a single robot moves (Fig. 5.9a), it detects an obstacle “A” and the goal point still is in a “blind spot”. Fig. 5.9b represents another situation, similar to the group movement in [167]. Here the first robot still detects obstacle “A”, the second robot moves closer to it. In the FOV of the second robot, part of obstacles “A” and “B” appears, but not both of them completely. In this case, the robots have over-detailed knowledge about one part of obstacle “A” and not so complete knowledge about the obstacle “B”. The goal is still invisible in this particular case. In Fig. 5.9c the zone is separated by distancing of the robots into two sectors (“sector A” for the first robot, “sector B” for the second). As can be observed in the case of Fig. 5.9c, agents have enough detalization of an obstacle “A”, information about the existence of obstacle “C”, and moreover, they have found the goal (object of interest). Using this information, the trajectories of each robot are recalculated for reaching the goal. Let us review the effectiveness of robots based on the “Scene #4” (Fig. 5.9d). Here under effectiveness will suppose the amount of unique data obtained by a single robot compared to the common data fusion.

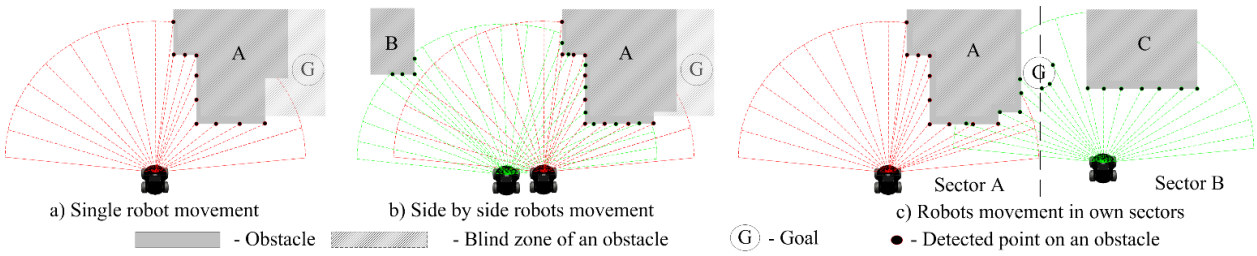


Figure 5.9 Terrain sectoring

5.4.2 Effectiveness calculation

Let us review effectiveness of robots based on the "Scene #4" (Fig. 5.6) . As effectiveness will be understood the amount of unique data obtained by robot comparing to common data fusion. On Fig. 5.10 are shown four different binary maps that in the end were known by the robotic group (Three individual maps for each robot and fused map).

Overlapping one individual binary map on other is possible to say that some sectors were detected only by one robot, other by two or three robots (Fig. 5.11).

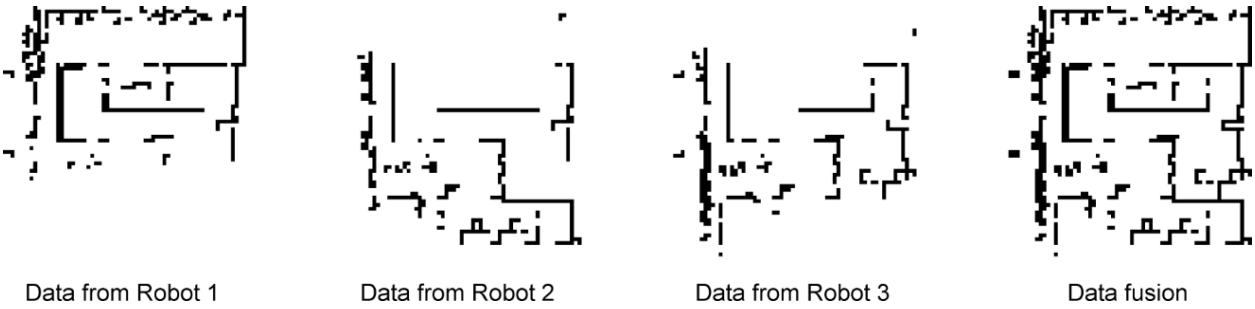


Figure 5.10 Binary maps of environment

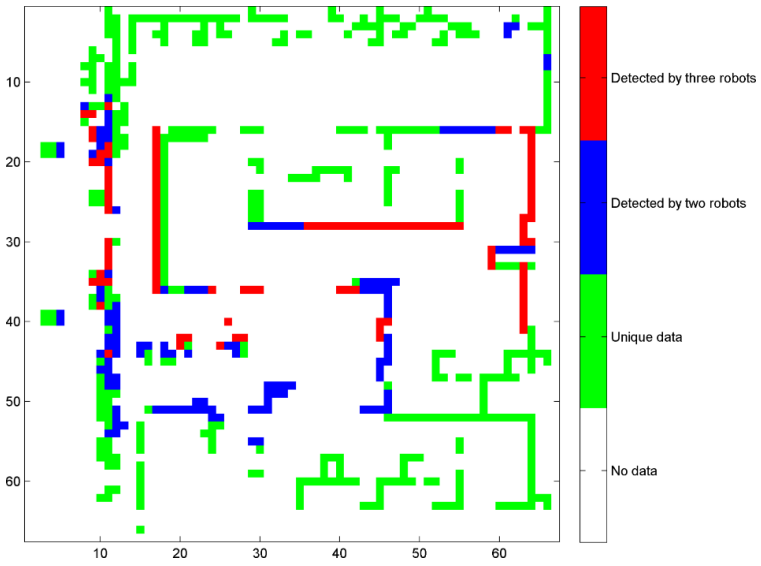


Figure 5.11 Overlapped individual binary maps

As was mentioned above each robot creates during its movement a binary map of the obtained

data about obstacles (i.e. individual maps Rm1, Rm2, . . . , Rmn of all n participants of the robotic group, for the current particular case n = 3). The map of obstacles is:

$$M = \sum_{i=1}^{n} Rm_i,\ where\ M_{xy} = \left\{ \begin{array}{l} 1,\ if\ non-walkable \\ \\ 0,\ if\ walkable \end{array} \right\} \tag{5.1}$$

For further calculations obtained matrixes will be considered as matrixes of integer values (for this purpose were used operation ConvertToIneger where “true” considered as 1 and “false” as 0). Applying the simple operation of matrix addition (Eq. 5.2) we will have the resulting matrix (Mr) of fused data with overlapped density (the overlapping density, in this case, is ranked by discrete values 1,2. . . n, and it can be defined as the higher the number, the more data repetitions occur in this location). On Fig. 5.12a,b,c are presented the individually obtained maps by n (n = 3) robots and Fig. 5.12d are fused data. The white areas are the locations where obstacles are not detected, green where obstacles were detected once, blue by two robots, and red by three. So, Fig. 5.12 shows, that according to the overlapped results of the individually obtained maps by each robot can be estimated the efficiency of environment sectoring and territorial group distribution.

$$Mr = \sum_{i=1}^{n} ConvertToIneger(Rm_i),\ where\ Mr_{xy} = 0...n \tag{5.2}$$

By subtracting the overlapped data from the resulting matrix Mr (cells with values more than 1) will get the unique data obtained in the group. The sum of these values will give a total amount of unique values. To calculate the group effectiveness (Ge) of terrain sectoring will be used in the next equation:

$$Ge = \frac{Tou}{To} \tag{5.3}$$

where Tou is a total amount of unique detected obstacle (Mr_{xy} = 1) and To is a value of total detected obstacles on a scene.

$$Tou = \sum_{x=0}^{w} \sum_{y=0}^{h} Mr_{xy},\ where\ Mr_{xy} = 1 \tag{5.4}$$

$$To = \sum_{x=0}^{w} \sum_{y=0}^{h} ConvertToInteger(M_{xy}) \tag{5.5}$$

where w and h are width and height of the matrix Mr correspondingly.

Besides the overlapped data, it can be allocated another characteristic – the ratio of individual data obtained by the robot to total data. Fig. 5.12 shows the results of the analysis of the obstacles

For these scenes, the average group efficiency equals 75.98%, the data obtained by two robots is 15.53% and by 3 is 8.48% (Fig. 13a). Comparing the results of obtained unique data from each robot (Fig. 13b) will receive the following values of 36.29%, 13.82%, and 25.86% for first, second, and the third robot respectively.

The approach of sectoring the terrain improves the individual FOV's of each robot to the extended joint FOV of RG. It gives a sufficiently detailed point cloud of surrounding, combining data from each individual TVS. Such extended FOV helps to avoid unnecessary scanning and decreases the number of calculations required to pass through the already scanned territory for all robots as a whole.

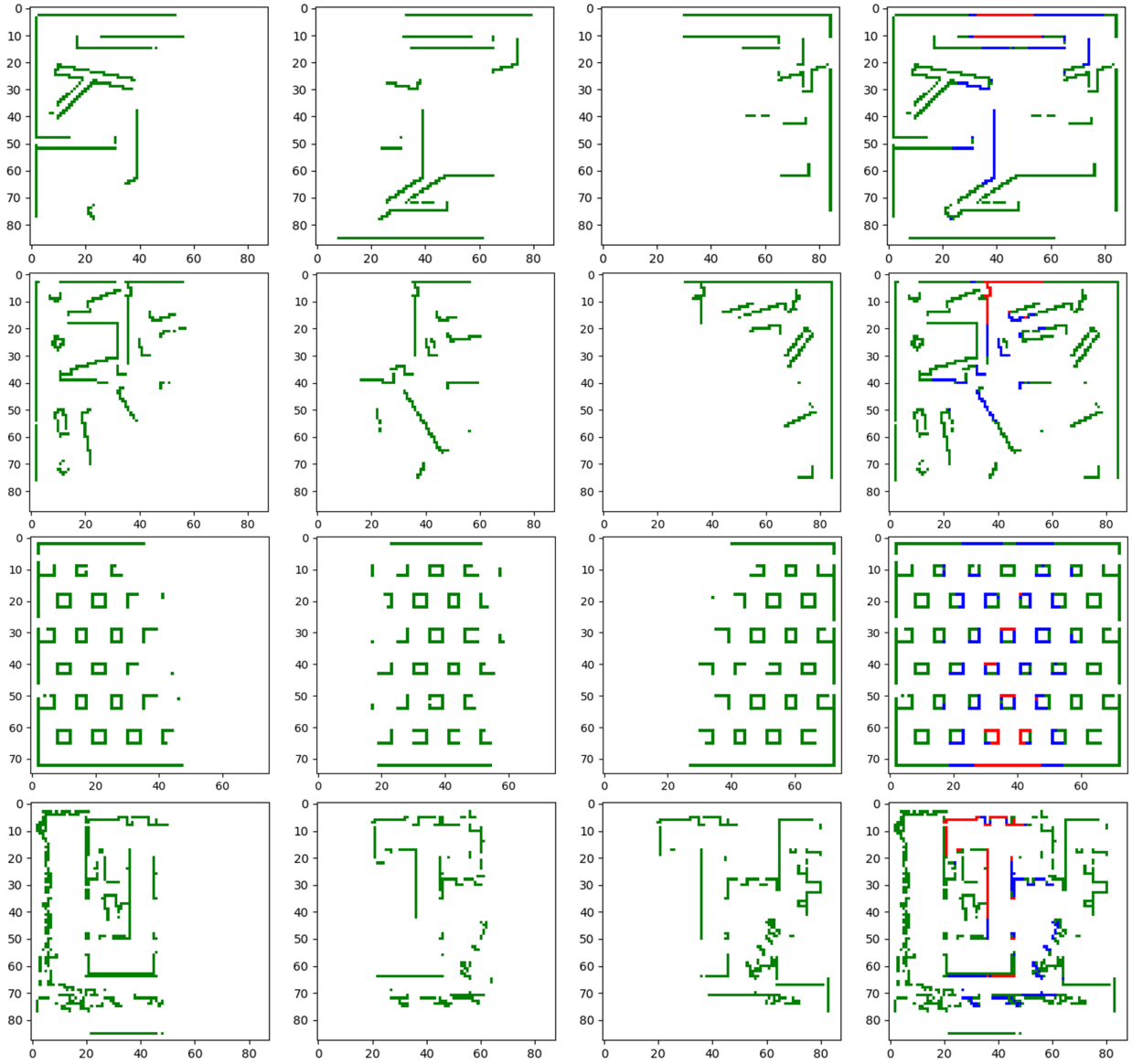
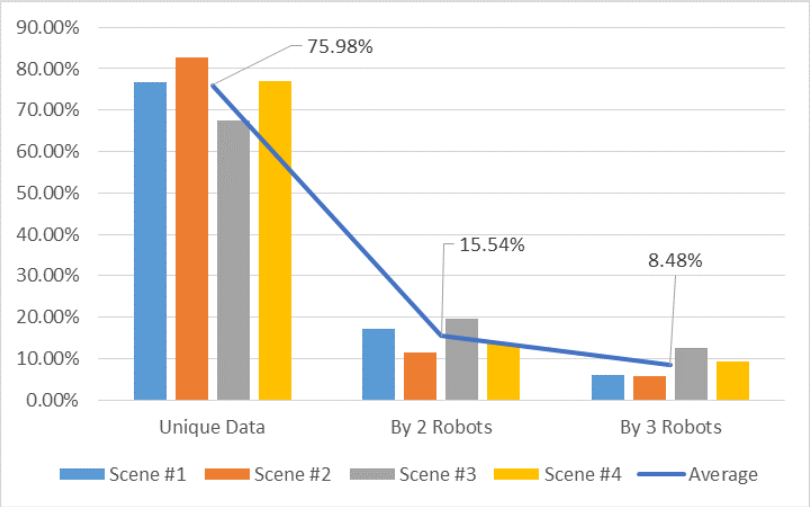
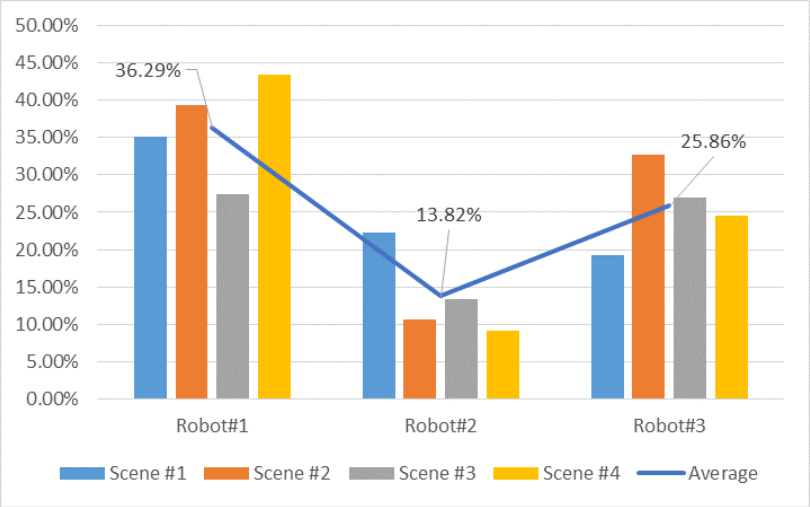


Figure 5.12 Individual maps overlapping



(a) Overlapped density values for each scene



(b) Unique data from each robot in group

Figure 5.13 Unique and general data comparison for each robot in group

5.4.3 Scenes description for modeling and analysis

For the analysis of group effectiveness was decided to make another modeling with different approach 5.17. Here environment has to be scanned with the group of robots (1 to 5 units in group). Each robot has a list of secondary objects needed to be visited in a specific order and return to initial position. The list of the scenes are presented in Tab. 5.4 and the visualized examples are

presented on Fig. 5.14.

Table 5.4 Experimental scenes description

Scenario	Scene	Obstacles on scene	Robots in group	Size
Scenario #1	Scene #1	39	1 to 5	100 x 100
Scenario #1	Scene #2	39	1 to 5	100 x 100
Scenario #1	Scene #3	17	1 to 5	100 x 100
Scenario #2	Scene #1	36	1 to 5	100 x 200
Scenario #2	Scene #2	92	1 to 5	100 x 200
Scenario #2	Scene #3	39	1 to 5	100 x 200
Scenario #3	Scene #1	63	1 to 5	200 x 200
Scenario #3	Scene #2	159	1 to 5	200 x 200
Scenario #3	Scene #3	159	1 to 5	200 x 200

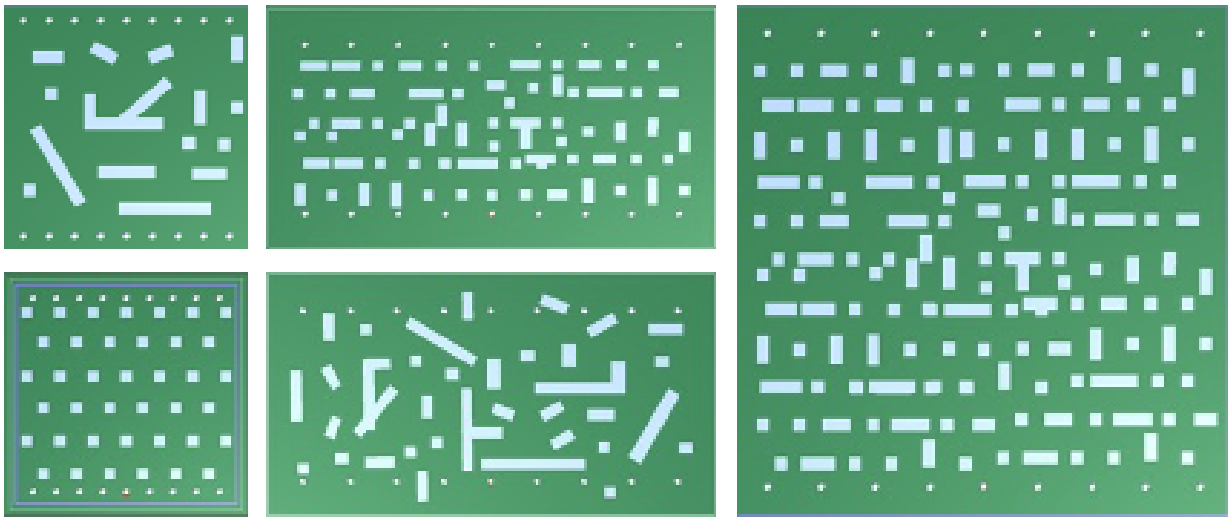


Figure 5.14 Examples of used scenes

5.4.4 Secondary objectives placement for surface mapping

For the task of terrain mapping it is necessary to locate additional points to visit. Solution can be adopted from surface mapping using UAV [168] or other autonomous surface vehicles [169]. These solutions are based on Dubins car principles ([152], [170]).

According to the Dubins principles, for the case of terrain mapping with single robot (Fig. 5.15a), territory is covered with pre-calculated trajectory. In place where the trajectory is changing its state (straight to round and round to straight) secondary points are placed (light grey dots on Fig. 5.15). Terrain mapping using a group of robots can be separated in two types: vertical movement (Fig. 5.15b) and horizontal movements (Fig. 5.15c). In both types territory is sliced into sectors (amount of sectors depends on amount of robots). Advantages of sectoring the terrain are described in [112].

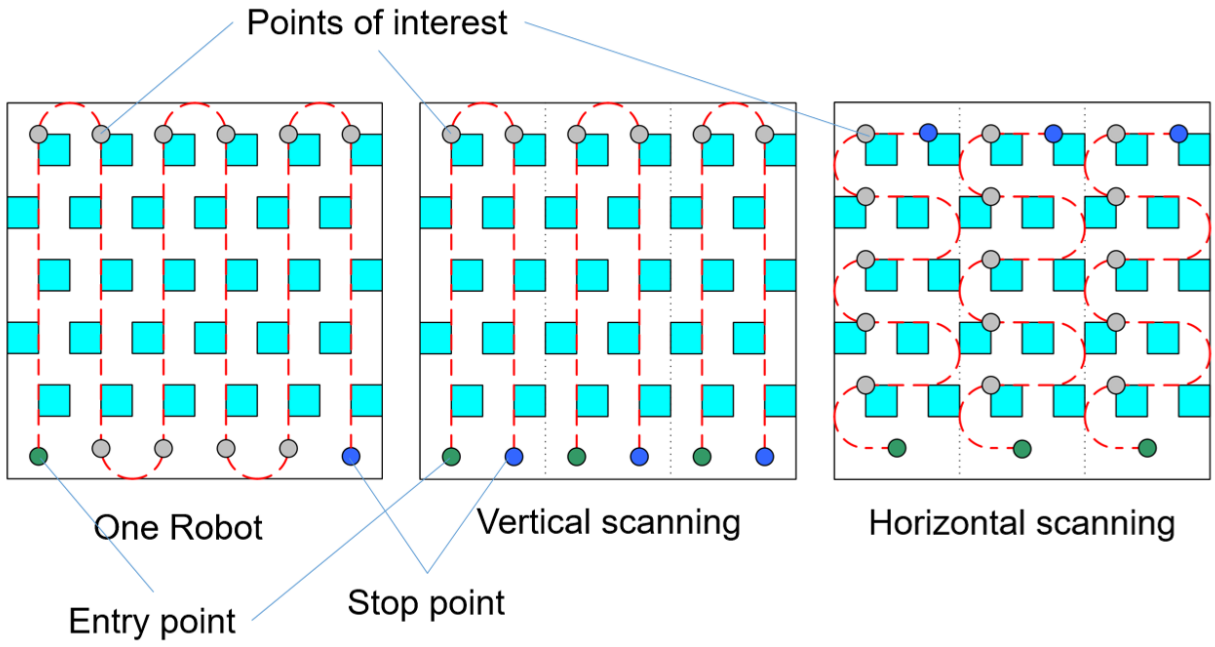


Figure 5.15 Secondary objectives placement

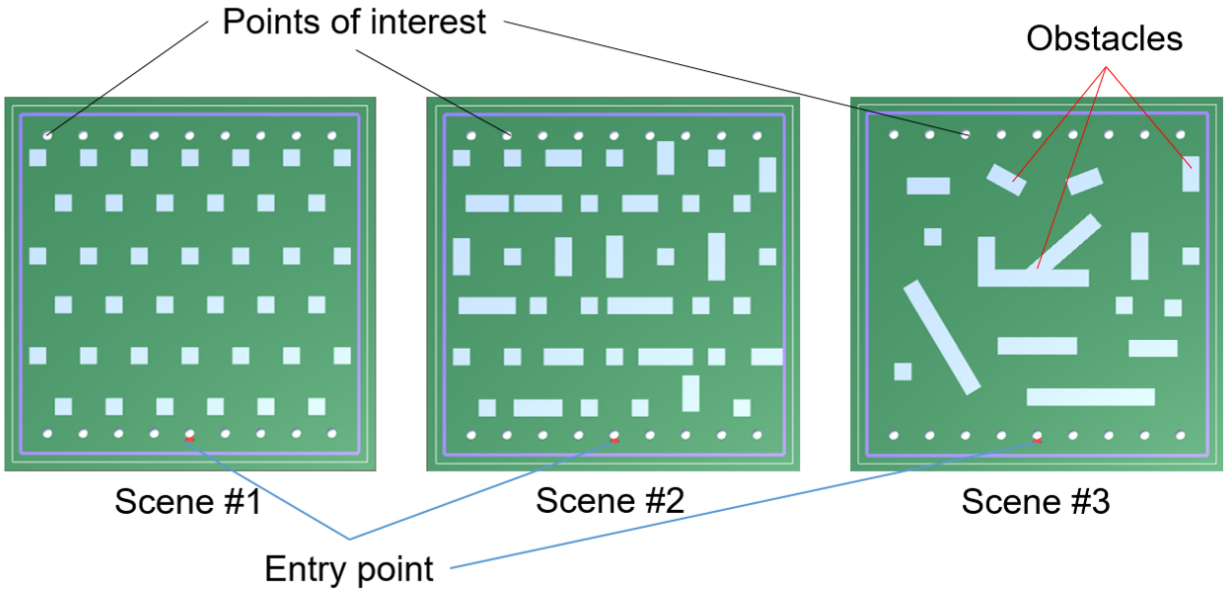
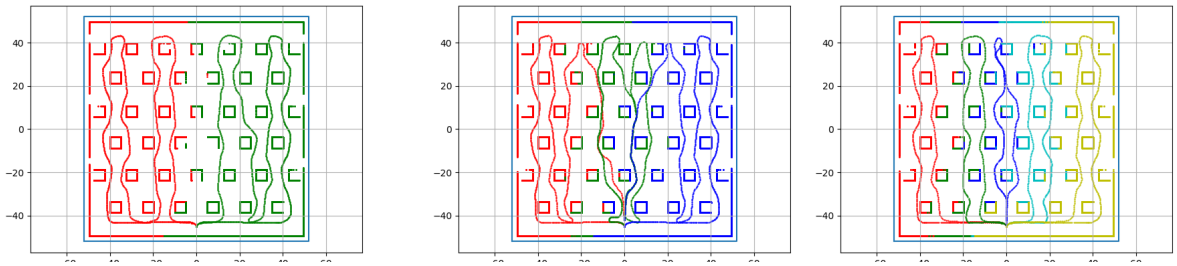


Figure 5.16 Secondary objectives placement on testing environment

5.4.5 Unique data as an index of effectiveness

After the modelling of each scene (Fig. 5.17) and closer look on the sectoring technique used in it was decided to use the unique data as the main characteristic of group effectiveness. Obtained results are presented in Fig. 5.18 - Fig. 5.21



(a) Example of obtained tracks for two robots

(b) Example of obtained tracks for three robots

(c) Example of obtained tracks for five robots

Figure 5.17 Example of obtained tracks and scanning sectors

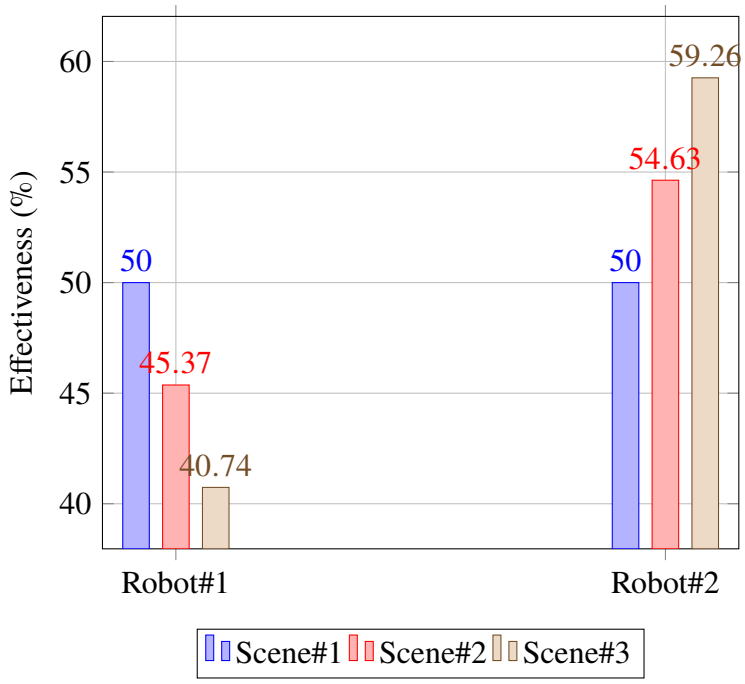


Figure 5.18 Effectiveness of two robots group

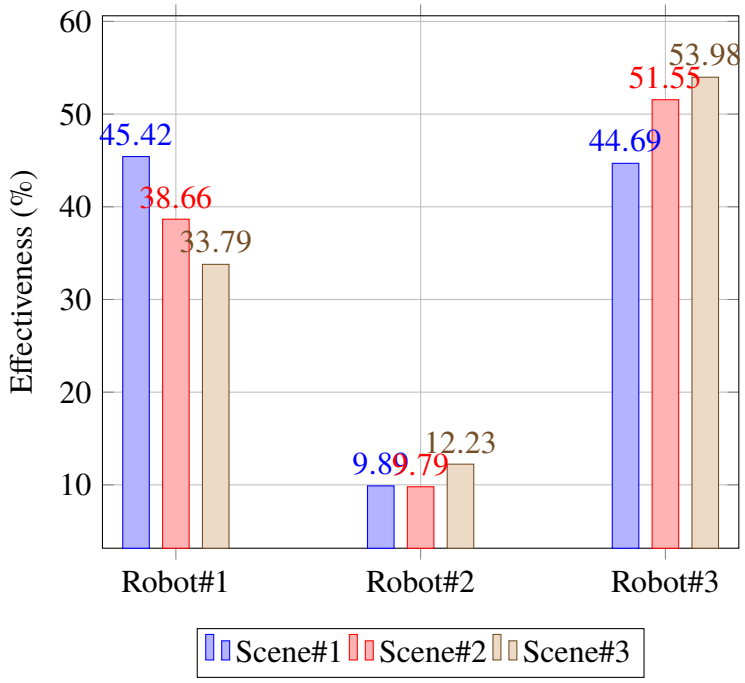


Figure 5.19 Effectiveness of three robots group

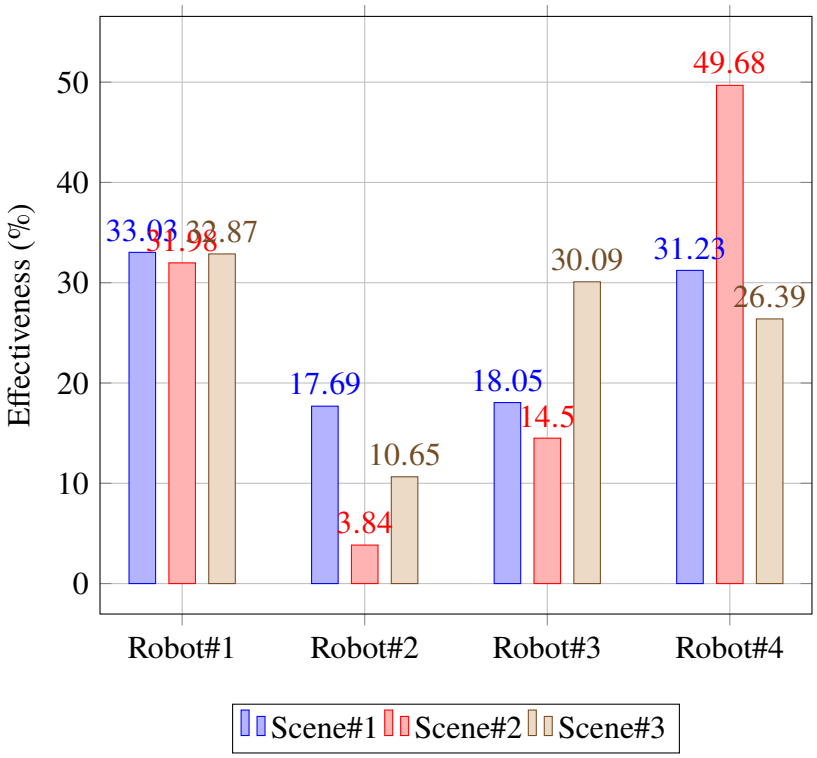


Figure 5.20 Effectiveness of four robots group

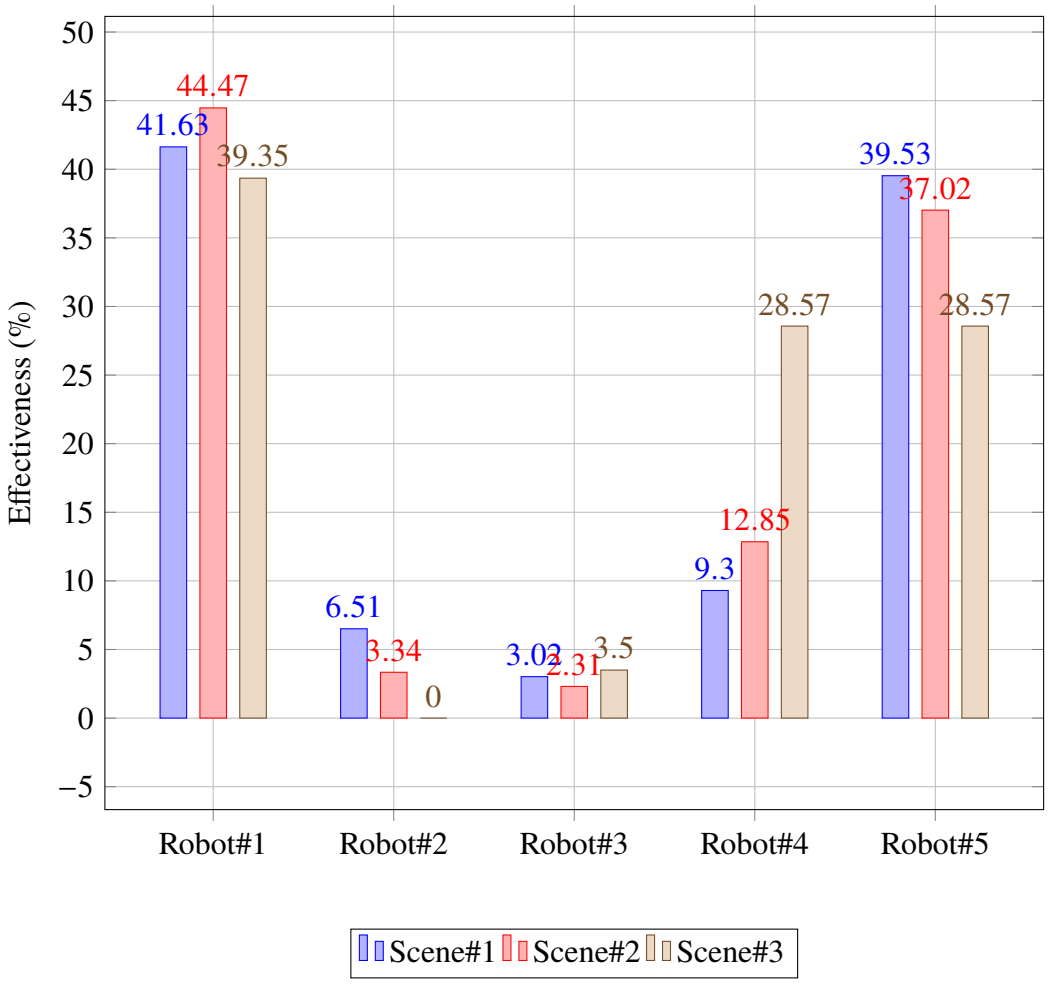


Figure 5.21 Effectiveness of five robots group

Fig. 5.22 shows the result of overlapped data as the averaged values of detected data. I shows that robots placed on the sector sides are more effective that robots that are moving in the middle of the map. It means that side robots are providing more data unique values (data in general) that other agents within the group.

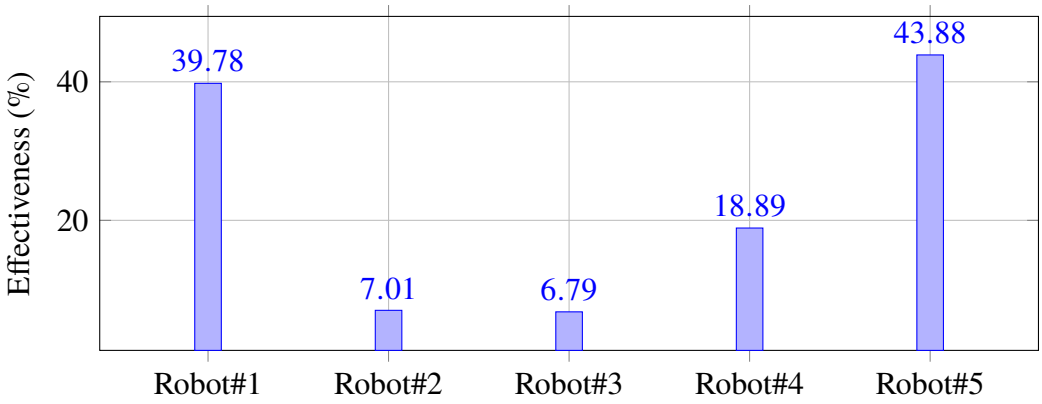


Figure 5.22 Avaraged overlapped values of group effectivenes

Additional analysis of detected data is shown on Fig. 5.23 - Fig. 5.31

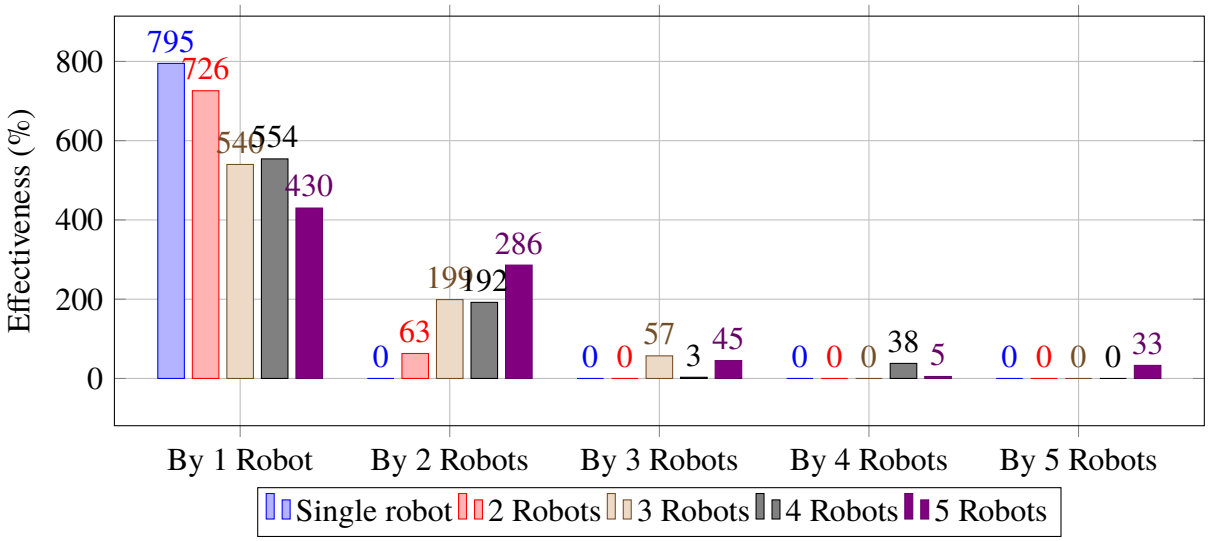


Figure 5.23 Effectiveness of five robots group (Scenario #1, Scene #1)

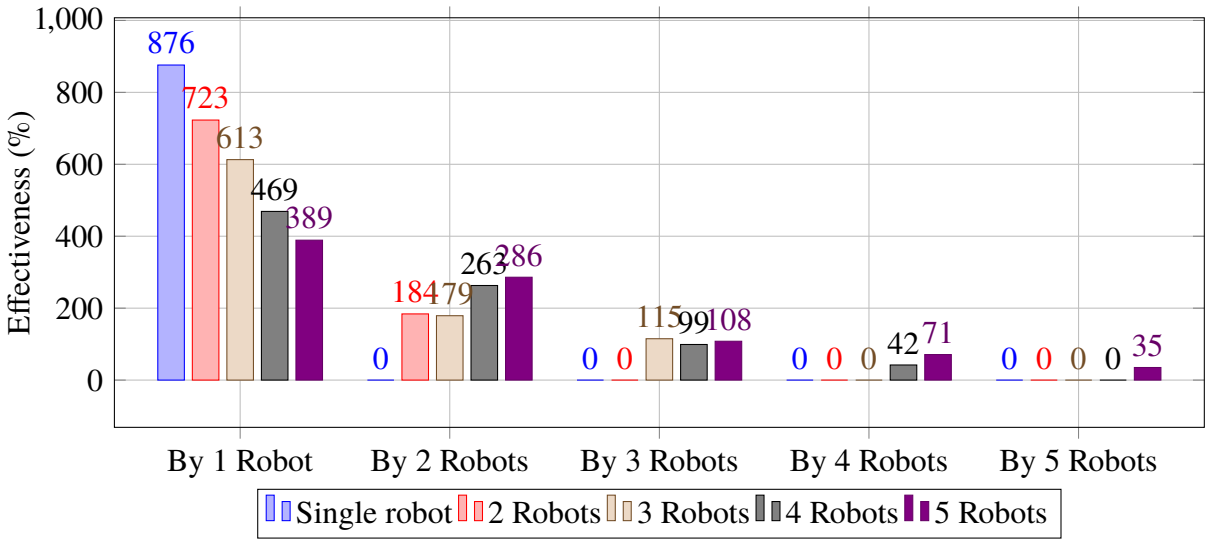


Figure 5.24 Effectiveness of five robots group (Scenario #1, Scene #2)

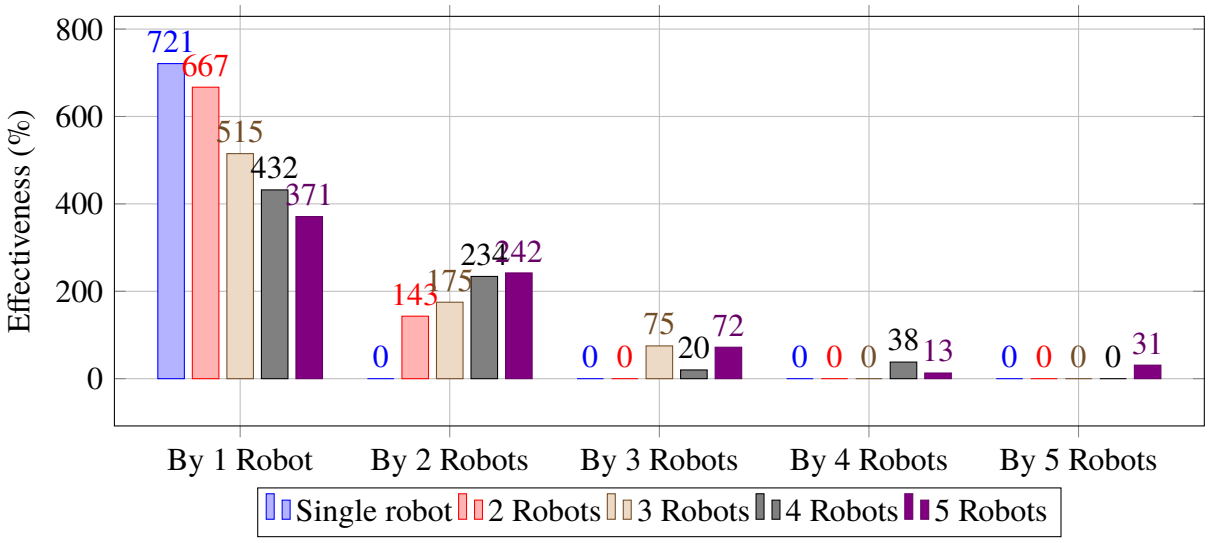


Figure 5.25 Effectiveness of five robots group (Scenario #1, Scene #3)

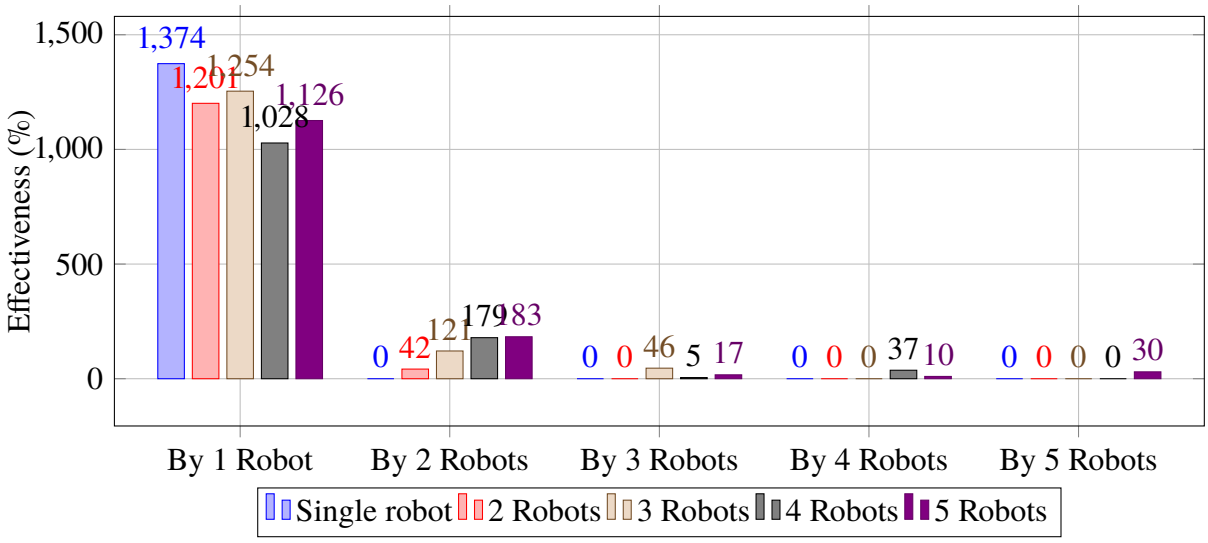


Figure 5.26 Effectiveness of five robots group (Scenario #2, Scene #1)

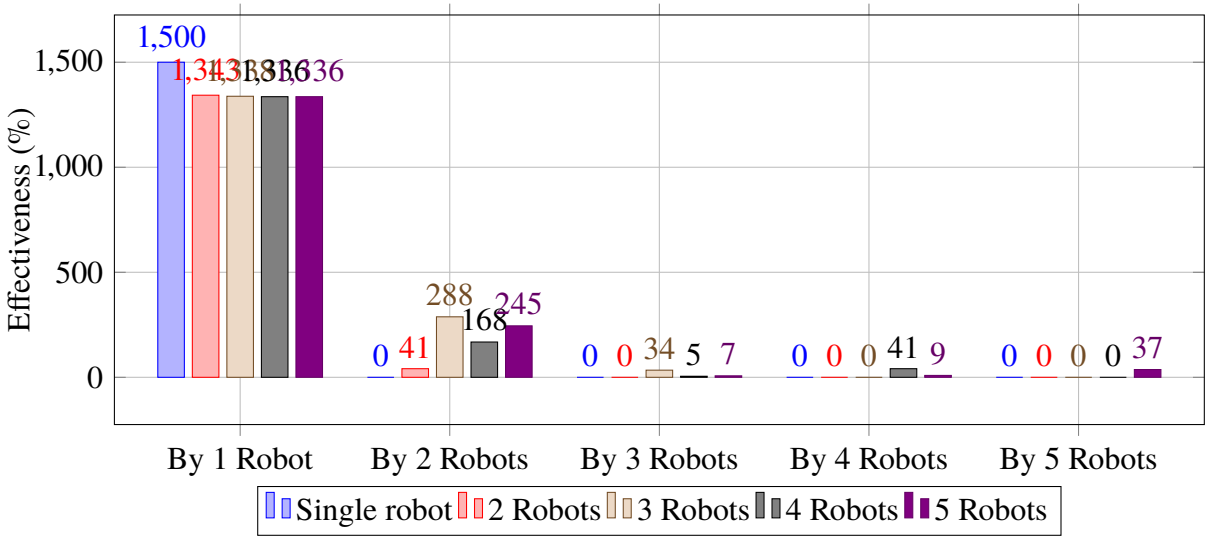


Figure 5.27 Effectiveness of five robots group (Scenario #2, Scene #2)

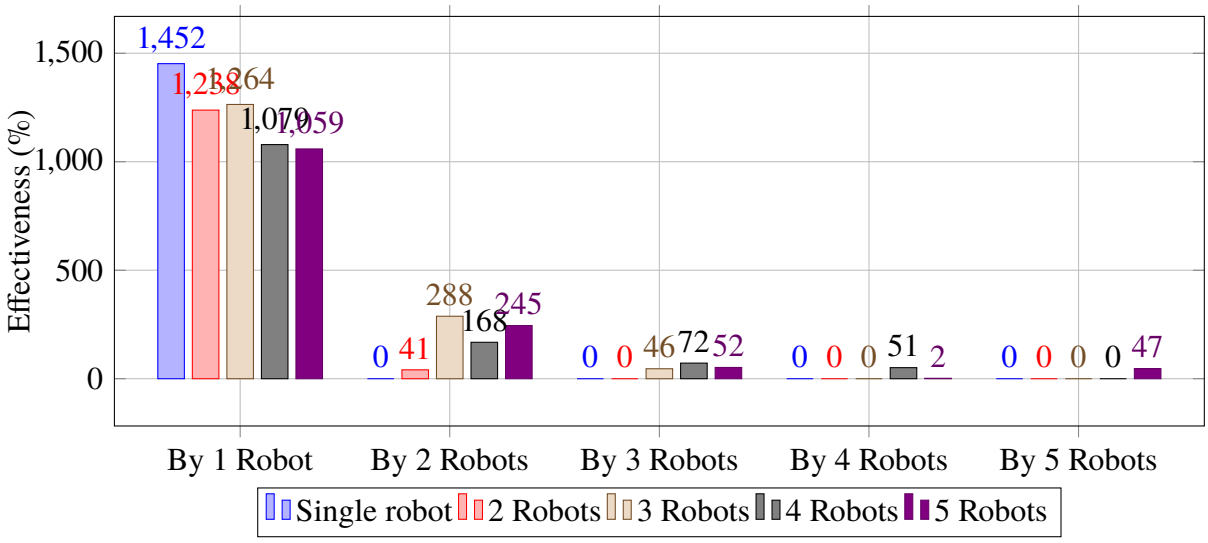


Figure 5.28 Effectiveness of five robots group (Scenario #2, Scene #3)

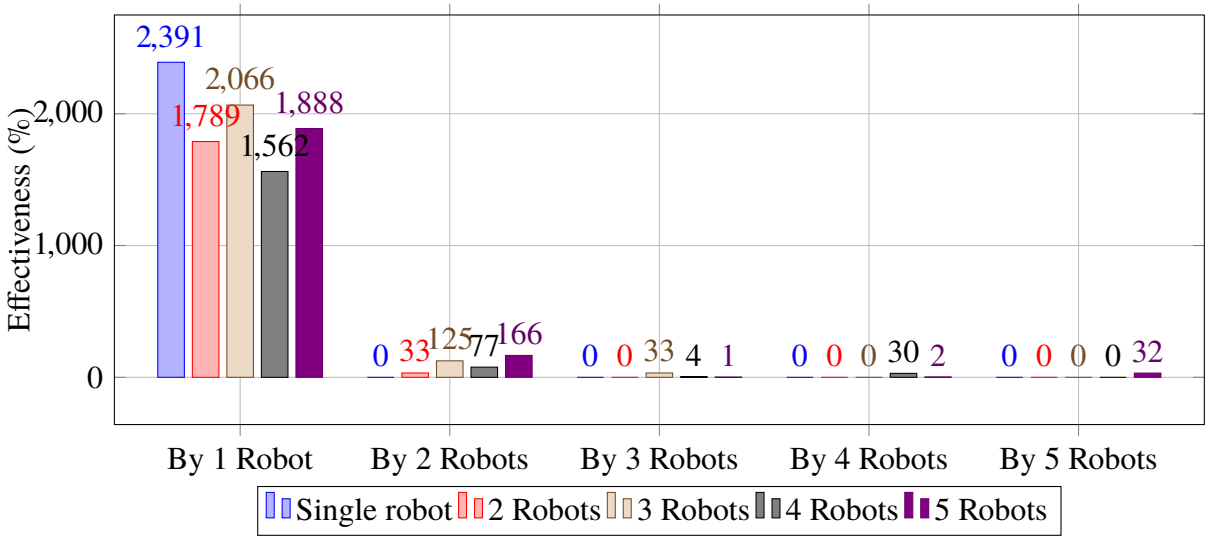


Figure 5.29 Effectiveness of five robots group (Scenario #3, Scene #1)

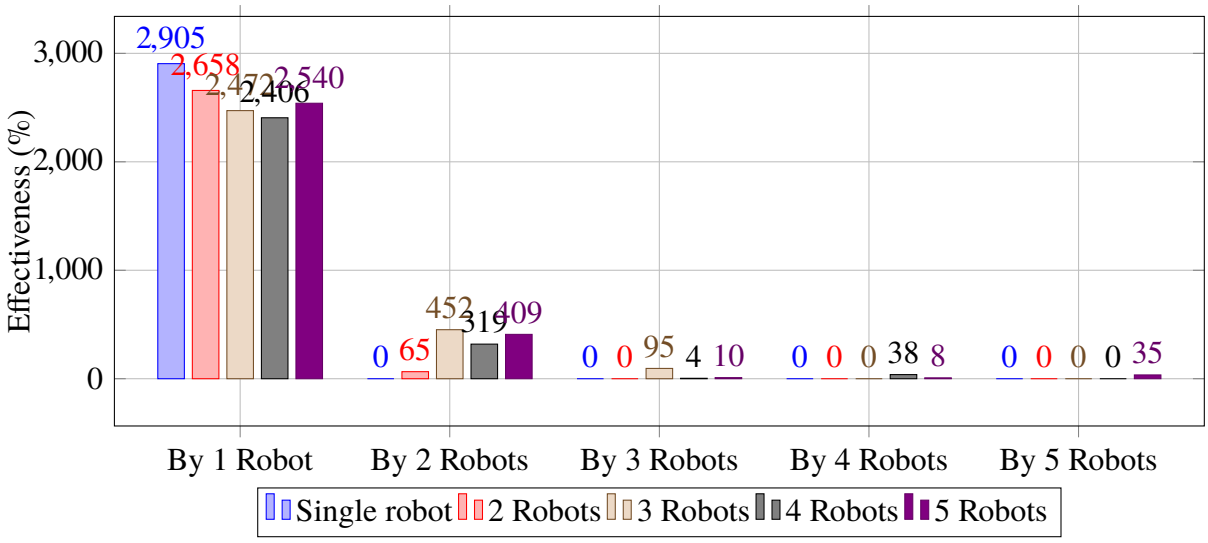


Figure 5.30 Effectiveness of five robots group (Scenario #3, Scene #2)

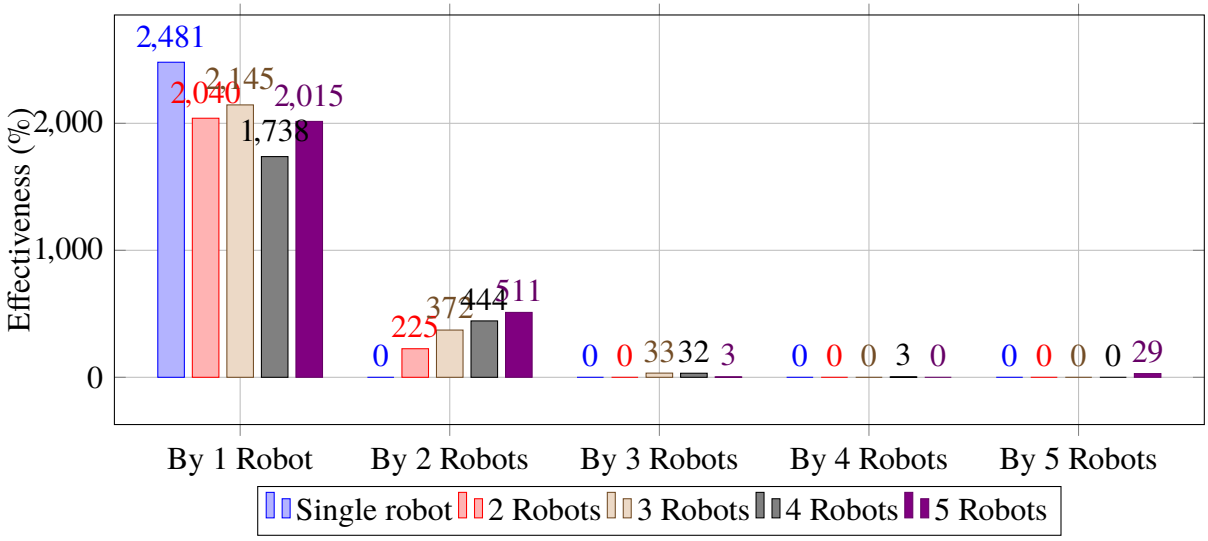
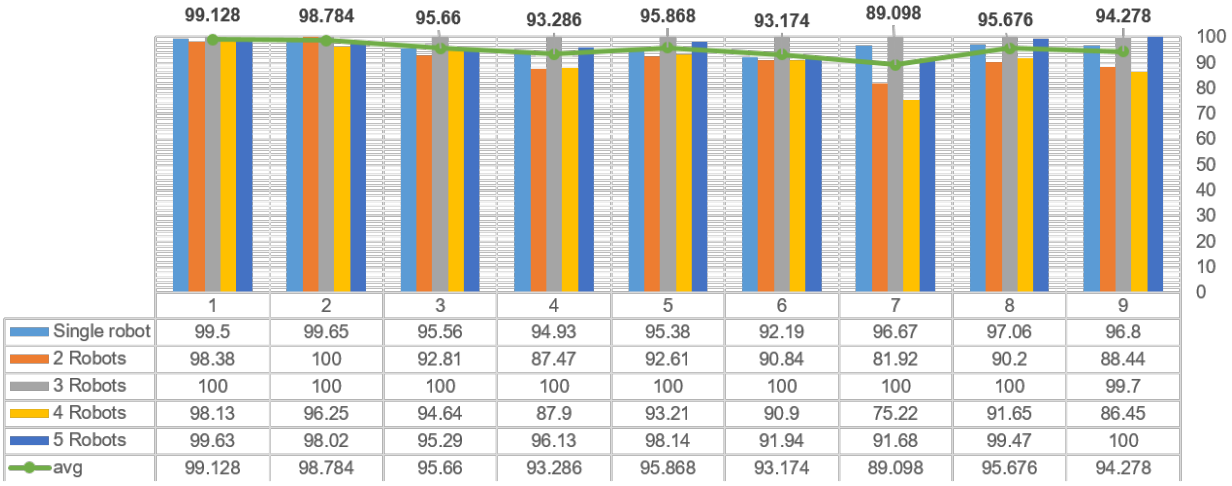


Figure 5.31 Effectiveness of five robots group (Scenario #3, Scene #3)

Comparing the obtained amount of data between the robotic groups in all scenes can be made the next conclusion that group of thee robots in all cases gives mode data that all other groups (Fig. 5.32).



	1	2	3	4	5	6	7	8	9
Single robot	99.5	99.65	95.56	94.93	95.38	92.19	96.67	97.06	96.8
2 Robots	98.38	100	92.81	87.47	92.61	90.84	81.92	90.2	88.44
3 Robots	100	100	100	100	100	100	100	100	99.7
4 Robots	98.13	96.25	94.64	87.9	93.21	90.9	75.22	91.65	86.45
5 Robots	99.63	98.02	95.29	96.13	98.14	91.94	91.68	99.47	100
avg	99.128	98.784	95.66	93.286	95.868	93.174	89.098	95.676	94.278

Figure 5.32 Comparing the detected data

5.4.6 Scene completion time

According to the specifics of the developed application was decided to use as time equivalence amount of iterations needed to finish the scene (Fig. 5.33). Next charts (Fig. 5.36 - Fig. 5.33) are representing the obtained data. Each chart represent each scenario separately.

Scene	Size	Obstacles on scene	Single Robot	Group of two robots	Group of three robots	Group of four robots	Group of five robots
Scene#1	100x100	39	3776	1696	1270	766	714
Scene#2	100x100	39	3508	1437	1394	911	704
Scene#3	100x100	17	3882	1785	1367	870	754
Scene#4	100x200	36	5050	2463	1942	1100	882
Scene#5	100x200	92	4596	2242	1642	1425	1047
Scene#6	100x200	39	5719	2507	1814	1241	874
Scene#7	200x200	63	4671	2057	1582	982	804
Scene#8	200x200	159	4255	1842	1523	869	719
Scene#9	200x200	67	4500	2013	1593	1280	1016
Scenario#1			3722	1639	1344	849	724
Scenario#2			5122	2404	1799	1255	934
Scenario#3			4475	1971	1566	1043	846
Total average			4440	2005	1570	1049	835

Figure 5.33 Tiempo de finalización de la escena

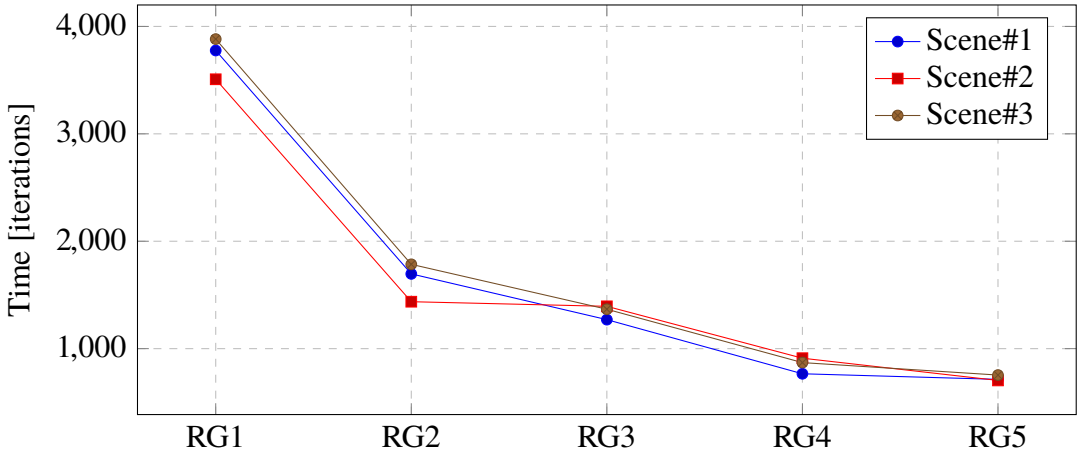


Figure 5.34 Scenario#1 completion time

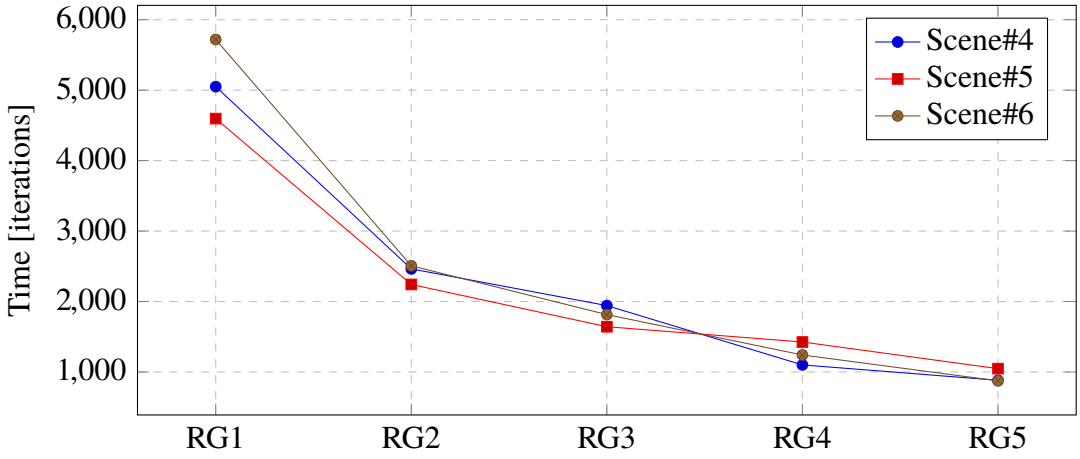


Figure 5.35 Scenario#2 completion time

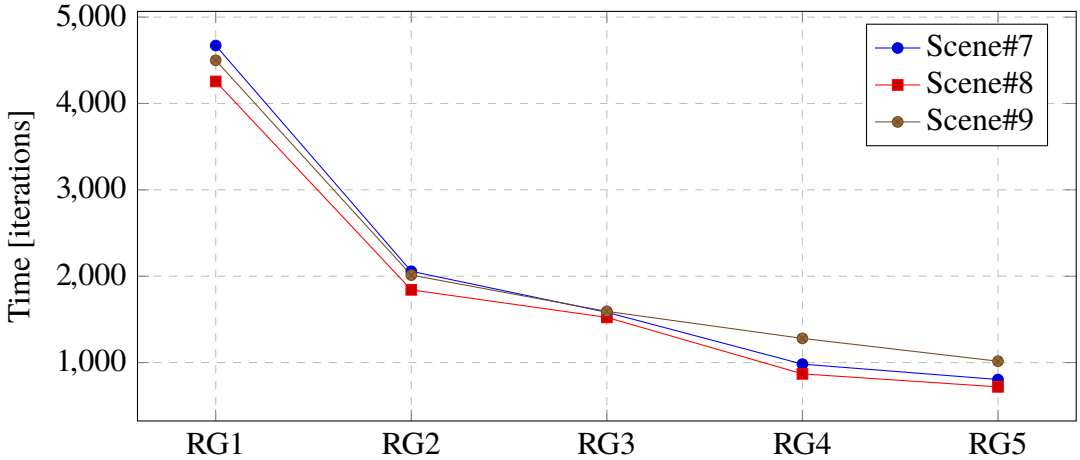


Figure 5.36 Scenario#3 completion time

To get more comparative view of averaged data (Fig. 5.37) it should be represented as normalized values (Fig. 5.38).

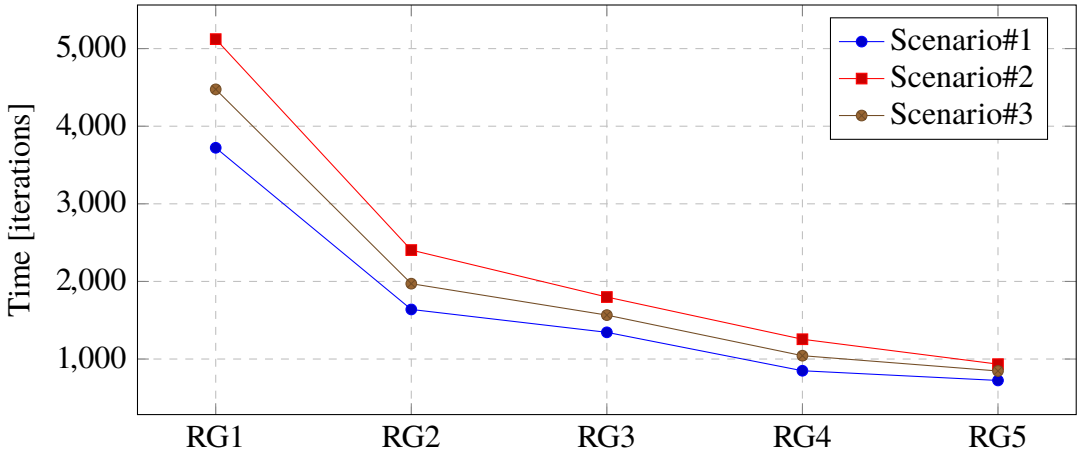


Figure 5.37 Average completion time

The results are showing that group of five robots gives the best time in performance, however the difference between group of three robots and five are just in 20% of time to complete the scene, but it is already two robots more that increase the complexity of calculations and system structure. That is why once more can be told that group of three robots are better to use.

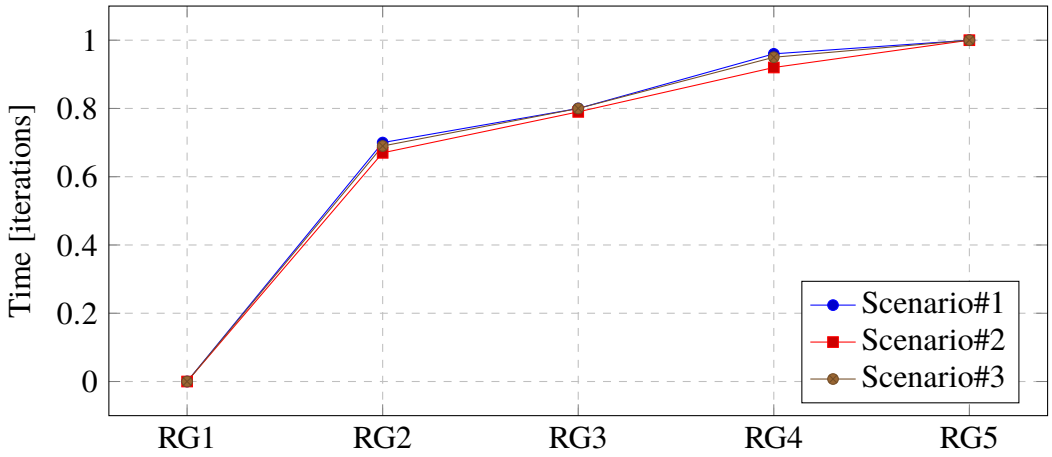


Figure 5.38 Normalized completion time

5.4.7 Informational entropy reduction analysis

Shannon proposed in [171] the notion of the average informativeness measure of the test (the unpredictability of its outcomes), which takes into account the probability of individual outcomes. Entropy (Eq. 5.6) is directly related to the "unexpectedness" of an event. From this follows its information content – the more predictable an event, the less informative it is. This means that its entropy will be lower.

$$H = -\sum_{i} P_i * \log_2 P_i \tag{5.6}$$

where Pi is the probability of the i-th outcome.

For our case is reviewed the event of obstacle detection. In the case of current task robots are moving in a partially unknown environment (only known boundaries of it). That is why the appearance of each new obstacle have equal value if entropy. Obstacles that were detected are not taken into account.

Fig. 5.39 are representing the obtained knowledge (in presents) during the time (iterations). Each figure represents the specific scene used during modeling for different robotic group sizes.

The presented graphics have the reversed logarithmic type of nature (were represented obtaining the knowledge and not directly decrease of entropy).

Reviewing the results can be noticed several patterns. On Fig. 5.39 the received knowledge has a step-like form. Is very noticeable in case of three robots from 500 to 1000 iterations, or on general the graphics form of single robot case. More over for the group of four and five robots the knowledge obtaining during time is almost the same as the difference in time of scene accomplishment (as were mentioned before).

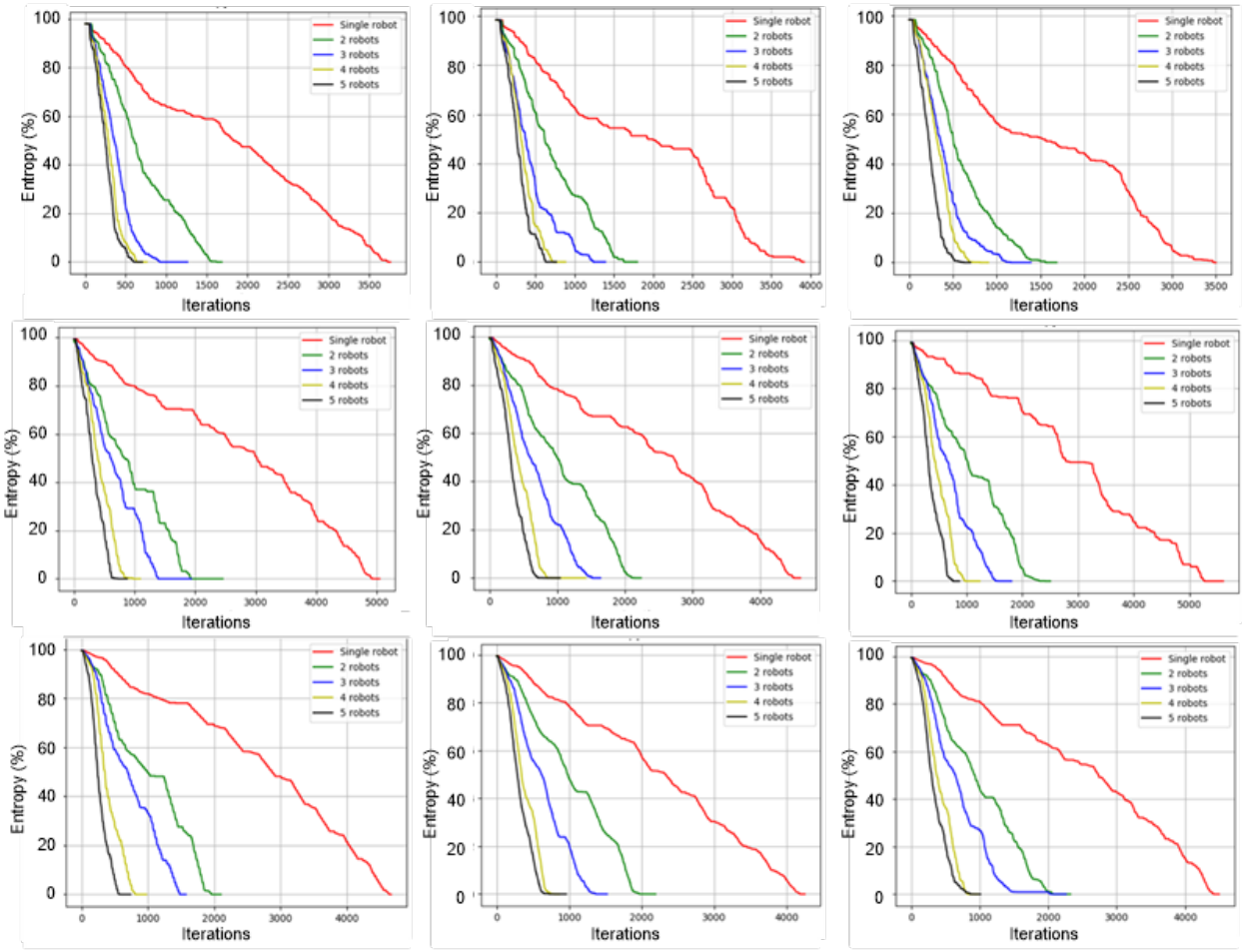


Figure 5.39 Entropy reduction speed

On the chart can be seen that the group of four and five robots are having almost the same entropy reduction speed (1%-3% difference). The group of thee robots are given the 7%-20% difference comparing to group of five robots (comparable with time completion scene).

5.5 Section conclusion

The presented section proposed the methodology to analyze the information entropy and evaluate the effectiveness of the robotic group. According to presented results it is possible to make the next conclusions: the location of the objects on the map has a minimal impact on the mapping; with the

same area, the scanning speed depends on the number of robots; The most complete scanning of the space gives a group of three robots (current case); Based on the current placement of robots, the extreme robots provide more information than the robots closer to the center (the average efficiency of the extreme robots is higher than 30%); the acquisition of data has a stepped form; the amount of data received directly affects the speed of obtaining a new knowledge (more robots know about the environment, the less new information it will be possible to obtain over time).

Chapter Six

Conclusions

6.1 Conclusions

The dissertation thesis was devoted to the methods of robotic group automation. The research was based on the three main aspects of optimization: the vision system that allows to work in different conditions and have a good accuracy; data transferring and communication within the group that allows to have an environment 3D map up to date and avoid additional movement planning; and the path planning approach that will allow to build minimized continuous trajectories and decrease curvatures banding energy while fitting the first two aspects.

All of the mentioned aspects were reviewed during the Chapters One to Chapter Four and analyzed in full within the implementation in Chapter Five. Based on the research the next conclusions can be done:

Initially were reviewed the origins and basics of swarm robotics. In thesis is overviewed the main types of behavior models and their application. According to them and existing projects were designed the behavior model of the robotic group for future research.

Analyzing all pros and cons of different types of vision systems were decided to use the 3D laser technical vision system. Based on it were made an improvement of 3D point cloud density stabilization stored in a memory of robot. With this aim into the system of robots, logic was added such variable as an opening angle. It keeps the density (resolution) of scanned surfaces in the necessary minimum resolution and applies to it the path planning, using the detailed scan on demand. Based on the obtained values were applied fuzzy logic rules defining three accuracy zones. According to Chapter Two, the initial point cloud density (11 points/meter) are 10.059°for "High accuracy zone", 3.011°for "Average accuracy zone" and 1.34°for "Low accuracy zone". The average angle for the "High accuracy zone" range will give a small resolution equal to 5-6 points per meter that is away from the initial goal, also robots start to miss the close positioned obstacles. So the low edge value of an opening angle for "High accuracy zone" was taken. The set of angles changed to 5.209°, 3.011°, 1.34°.

It is obvious that data exchange between the n robots in a group permits to get more additional information in a fuzzed dataset that has each individual robot itself. It can provide more complete monitoring results. The main idea is to give each individual robot in a group more knowledge about the sector as quickly as possible. Data exchange serves for more efficient implementation as a supporting tool of described methods in the thesis. For the purpose of communication were developed two approaches: based on spanning tree protocol principles and leader base communication with the propagation of information with feedback. The first method is embedding the Kruskal algorithm to calculate a spanning tree and Flowyd-Warshal to find the main (center) node the fuzzy logic rules to select a center (leader). It is established in this thesis that the introduced method has a proportional function with respect to the number of robots in a swarm. For small

groups, the second method is used.

The leader changing system that was implemented improves the process of data transferring by dynamically changing the network model from centralized management to centralized hierarchical control and backward. The proposed dynamic data exchange network forming method extends the potential of our novel TVS. It improves the ability of a single robot efficiency of dead reckoning with a cloud-like common knowledge base within the robotic group. The proposed methods allow the elimination of topological loops in the data network in a group of robots. A fully connected graph of a real network with a high probability leads to endless repetitions of the same messages in a group, while network bandwidth is almost completely occupied by these useless replays. In these conditions formally the network can continue to operate but in practice, its performance becomes so low that may lead to a complete network failure. Therefore, the proposed methods ensure the full propagation of information within the group and help to improve the movement coordination of a robotic group by exchanging information about the missing sectors. Analysis has shown that the implementation of leader based method reduced the data loss and the average timeout in request processing were less than the time of 3D TVS obstacle detection time (0.039s in both cases of network structure).

For the path, planning was adopted to our task conditions the use of the A* algorithm with two steps of post-processing. The algorithm suits the workflow of TVS and its output data to create the environmental matrix-based map. For the post-processing were compared three motion planning methods: Dubins path, Bezier approximation using 3 and 4 points. For the evaluation of those algorithms, we used two parameters: total length and bending energy. The Dubins path received the highest length, but the best bending energy saving. The Bezier approximation using 3 points

polygon shows the worst ability for optimized path planning. Ultimately, Bezier approximation using 4 point polygon gives the most satisfactory average result. Making a detailed comparison of the Bezier approximation and Dubins path, the result of the path made by the first algorithm is 10.3% – 12.7% more effective (bending energy efficient) than the solution provided by Dubins path. In the case of 3 point Bezier approximation, it looses in all aspects (7% more of a path length and 46% less energy efficient) comparing to 4 point Bezier approximation. In other words, its application permits to get the minimized trajectory, providing at the same time the maximum savings of the robot's source and ware of mechanisms. In general, these methods are solving the task of motion planning for the independent robot in a group.

The original simulation software was developed to verify the benefits of methods. It implements the behavior model while combining all of the vision, communication, and path planning methods together. The thesis offers an original solution that improves robotic group teamwork To achieve the result were used two types of modeling: first to review the influence of data transferring on dead reckoning (a group of 1 to 3 robots on 4 scenes and 100 simulations for each of 3 scenarios with/without common knowledge base and predefined environment); the second is to review the effectiveness of the group (with the groups of 1 to 5 robots, 3 scenarios and 9 scenes in total).

Comparing averaged distances obtained during the modeling can be observed that the use of a common knowledge base has advantages in all of the scenes. The result shows that the robotic group with implemented data exchange method has averaged group trajectory length shorter from 6.2% up to 10%, comparing to distances of individual autonomous robotic trajectories (when using non-group movement). Scaling the results for individual robots in-group the improvement of trajectories length can reach up to 21.3%. It gives the possibility to free a significant part of

memory on an individual robot for a complementary task solution in real-time and use the data only needed for navigation task. Simulations have shown that the use of mentioned improvements applied to the behavior of the robotic group allows stable functioning of the group at lower energy costs of motion (bending energy) and decreases trajectories length.

The proposed theoretical method is the subject to efficient application in many engineering tasks, such as surroundings mapping after a natural or human-caused disaster, indoor navigation, surface recognition, etc. while the reconstructed 3D image in its turn is possible to use for structural health monitoring.

6.2 Future works

The next tasks are next steps in environmental analysis: clusterization of 3D scans and structural health monitoring. The examples of the solutions are presented below.

Density-based spatial clustering of applications with noise (DBSCAN)[172] algorithm was proposed by Martin Esther, Hans-Peter Kriegel, and colleagues in 1996 as a solution to the problem of splitting (initially spatial) data into clusters of arbitrary shape. Most algorithms that produce a flat partition create clusters that are close to spherical in shape, since they minimize the distance of the documents to the center of the cluster. DBSCAN authors experimentally showed that their algorithm is able to recognize clusters of various shapes.

The idea of the algorithm is that inside each cluster there is a typical density of points, which is noticeably higher than the density outside the cluster, as well as the density in areas with noise below the density of any of the clusters. For each point of the cluster its neighborhood of a given radius must contain at least some number of points, this number of points is specified by a threshold

value (Fig. 6.1).

The algorithm can be presented as next:

- Given the dataset
- Label all points as core or non core
- Until all core points are visited:
  - Add one of non visited core point P to a new cluster
  - Until all points in cluster are visited:
    - For each non visited core point P within the cluster:
      - Add all core points within boundary of P to the cluster
      - Mark P as visited
- Until all non-core points are visited:
  - If a non core point P has a core point within its boundary, add it to the cluster corresponding to that core point
  - Else ignore

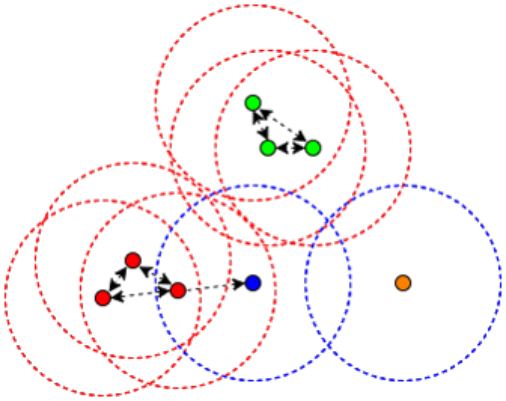
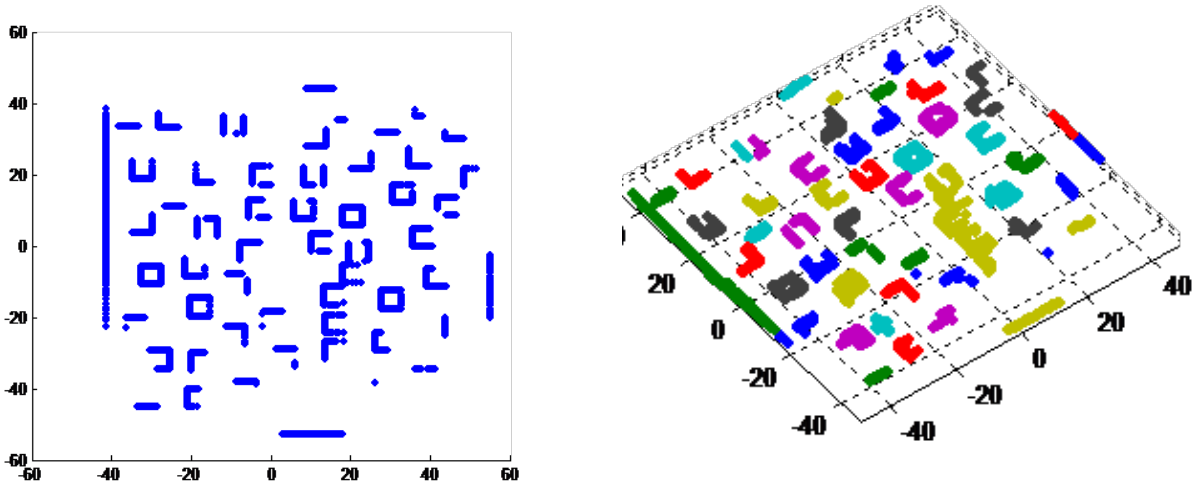


Figure 6.1 DBSCAN clustering illustration

Example of algorithm implementation presented in Fig. 6.2. Here can be seen a scanned environment using the group of robots and TVS (Fig. 6.2a). After implementation of DBSCAN can be seen clustered objects (Fig. 6.2b).



(a) Original point cloud of scene **(b)** Clustered scene

Figure 6.2 Example of DBSCAN implementation

According to the clustered data set objects can be extracted for further analysis (Fig. 6.3). Obtained data can be used in many applications like object classification and recognition, surface reconstruction and etc.

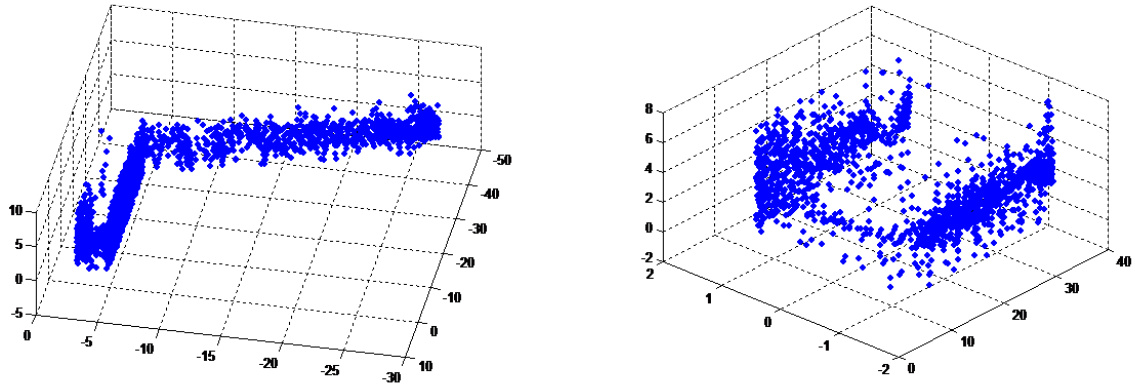
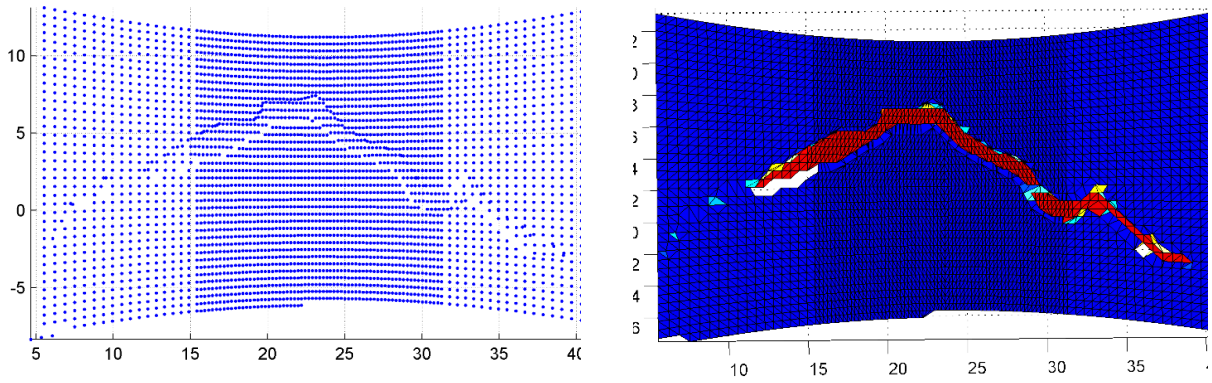


Figure 6.3 Extracted objects

One of the main sub applications is to use data for structural health monitoring. The scanned surface (Fig. 6.4a) can be analyzed and reconstructed to detect crack (Fig. 6.4b) or other problems that accrue.



(a) Scan of cracked surface **(b)** Detected crack

Figure 6.4 Implementation of approach for structural health monitoring

Publications

Articles with impact factor

- Ivanov, M., Sergiyenko, O., Tyrsa, V., Lindner, L., Rodriguez-Quinonez, J. C., Flores-Fuentes, W., Hipolito, J. N. (2019). Software Advances using n-agents Wireless Communication Integration for Optimization of Surrounding Recognition and Robotic Group Dead Reckoning. Programming and Computer Software, 45(8), 557-569.

- Ivanov, M., Sergyienko, O., Tyrsa, V., Lindner, L., Flores-Fuentes, W., Rodriguez-Quinonez, J. C., Mercorelli, P. (2020). Influence of data clouds fusion from 3D real-time vision system on robotic group dead reckoning in unknown terrain. IEEE/CAA Journal of Automatica Sinica, 7(2), 368-385.

- O.Yu. Sergiyenko, M.V. Ivanov, V.V. Tyrsa, V.M. Kartashov, M. Rivas-Lopez, D. Hernandez-Balbuena, W. Flores-Fuentes, J.C. Rodriguez-Quinonez, J.I. Nieto-Hipolito, W. Hernandez, A. Tchernykh, Data transferring model determination in robotic group, Robotics and Autonomous Systems, Volume 83, 2016, Pages 251-260

- Lars Lindner, Oleg Sergiyenko, Moises Rivas-Lopez, Daniel Hernandez-Balbuena, Wendy

Flores-Fuentes, Julio C. Rodriguez-Quinonez, Fabian N. Murrieta-Rico, Mykhailo Ivanov, Vera Tyrsa, Luis C. Basaca-Preciado, (2017) "Exact laser beam positioning for measurement of vegetation vitality", Industrial Robot: the international journal of robotics research and application, Vol. 44 Issue: 4, pp. 532-541

International conferences

- Mykhailo Ivanov, Oleg Sergiyenko, Vera Tyrsa, Vladimir Kartashov, Yelizaveta Tolstykh, Moises Rivas-Lopez, Daniel Hernandez-Balbuena, Paolo Mercorelli, Julio Rodriguez-Quinonez, Wendy Flores-Fuentes, Lars Lindner, “Individual scans fusion in virtual knowledge base for navigation of mobile robotic group with 3d TVS" 2017 Radar. Satellite Navigation. Radiomonitoring, Kharkov, 2017, pp. 55-60.

- Mykhailo Ivanov, Oleg Sergiyenko, Vera Tyrsa, Paolo Mercorelli, Vladimir Kartashov, Wilmar Hernandez, Sergiy Sheiko, Marina Kolendovska, Individual scans fusion in virtual knowledge base for navigation of mobile robotic group with 3D TVS, IEEE, IECON 2018

- Ivanov, M. V., Sergiyenko, O. Y., Tyrsa, V. V., Lindner, L., Rodriguez-Quinonez, J. C., Flores-Fuentes, W., Nieto Hipolito, J. I. (2019). Wireless integration to optimize environmental recognition and calculate the trajectory of a group of robots. Proceedings of the Institute for System Programming of the RAS, 31(2), 67-82.

- Ivanov, M., Sergiyenko, O., Mercorelli, P., Hernandez, W., Tyrsa, V., Hernandez-Balbuena, D., Iryna, T. (2019, June). Effective informational entropy reduction in multi-robot sys-

tems based on real-time TVS. In 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE) (pp. 1162-1167). IEEE.

- O. Sergiyenko, V. Kartashov, M. Ivanov, D. Hernandez-Balbuena, V. Tyrsa and J. I. Nieto-Hipolito, "Transferring model in robotic group,"2016 IEEE 25th International Symposium on Industrial Electronics (ISIE), Santa Clara, CA, 2016, pp. 946-952.

- L. Lindner, Oleg Sergiyenko, Moises Rivas-Lopez, Mykhailo Ivanov, Julio C. Rodriguez-Quinonez, Daniel Hernandez-Balbuena, Wendy Flores-Fuentes, Vera Tyrsa, Fabian N. Murrieta-Rico, Paolo Mercorelli, "Machine vision system errors for unmanned aerial vehicle navigation," 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), Edinburgh, 2017, pp. 1615-1620.

- Miguel Reyes-Garcia, Lars Lindner, Moises Rivas-Lopez, Julio C. Rodriguez-Quinonez, Wendy Flores-Fuentes, Mykhailo Ivanov, Fabian N. Murrieta-Rico, Alexander Gurko, Viktor I. Melnik, Reduction of Angular Position Error of a Machine Vision System using the Digital Controller LM629, IEEE, IECON 2018

- Reyes-Garcia, M., Sergiyenko, O., Ivanov, M., Lindner, L., Rodriguez-Quinonez, J. C., Hernandez-Balbuena, D., Murrieta-Rico, F. N. (2019, June). Defining the Final Angular Position of DC Motor shaft using a Trapezoidal Trajectory Profile. In 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE) (pp. 1694-1699). IEEE.

- Hernandez, W., Mendez, A., Ballesteros, F., Gonzalez-Posada, V., Jimenez, J. L., Tyrsa, V., Ivanov, M. Quezada-Sarmiento, P. A. (2019, October). A method of image classification

by using multidimensional scaling. In IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society (Vol. 1, pp. 5559-5565). IEEE.

Book chapters

- Mykhailo Ivanov, Lars Lindner, Oleg Sergiyenko, Julio Rodriguez-Quinonez, Wendy Flores-Fuentes, Moises Rivas-Lopez (2018). Mobile Robot Path Planning Using Continuous Laser Scanning, Optoelectronics in Machine Vision-Based Theories and Applications (pp. 300). Hershey, Pennsylvania: IGI-Global.

- Ivanov, M., Sergiyenko, O., Tyrsa, V., Lindner, L., Reyes-Garcia, M., Rodriguez-Quinonez, J. C., Hernandez-Balbuena, D. (2020). Data Exchange and Task of Navigation for Robotic Group. In Machine Vision and Navigation (pp. 389-430). Springer, Cham.

- Reyes-Garcia, M., Sepulveda-Valdez, C., Sergiyenko, O., Rivas-Lopez, M., Rodriguez-Quinonez, J. C., Flores-Fuentes, W., Ivanov, M. (2020). Digital Control Theory Application and Signal Processing in a Laser Scanning System Applied for Mobile Robotics. In Control and Signal Processing Applications for Mobile and Aerial Robotic Systems (pp. 215-265). IGI Global.

Copyrights

Nataniel Murrieta – Rico, Modelacion de la red para transferencia de datos en enjambre robotico

- Miguel Reyes Garcia, Oleg Sergiyenko, Julio Cesar Rodriguez Quinonez, Wendy Flores Fuentes, Moisés Rivas Lopez, Daniel Hernandez Balbuena, Fabian Nataniel Murrieta Rico, Mykhailo Ivanov, Lars Lindner, Interfaz para la implementacion del Integrado LM629

References

[1] Qigao Fan et al. “Data fusion for indoor mobile robot positioning based on tightly coupled INS/UWB”. In: *The Journal of Navigation* 70.5 (2017), pp. 1079–1097.

[2] Abduladhem A Ali et al. “An algorithm for multi-robot collision-free navigation based on shortest distance”. In: *Robotics and Autonomous Systems* 75 (2016), pp. 119–128.

[3] Gregory Dudek et al. “A taxonomy for multi-agent robotics”. In: *Autonomous Robots* 3.4 (1996), pp. 375–397.

[4] Adham Atyabi, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. “Navigating a robotic swarm in an uncharted 2D landscape”. In: *Applied soft computing* 10.1 (2010), pp. 149–169.

[5] Paul Levi, Eugen Meister, and Florian Schlachter. “Reconfigurable swarm robots produce self-assembling and self-repairing organisms”. In: *Robotics and Autonomous Systems* 62.10 (2014), pp. 1371–1376.

[6] Alan Oliveira de Sá, Nadia Nedjah, and Luiza de Macedo Mourelle. “Distributed and resilient localization algorithm for Swarm Robotic Systems”. In: *Applied Soft Computing* 57 (2017), pp. 738–750.

[7] Eric R Teoh and David G Kidd. “Rage against the machine? Google’s self-driving cars versus human drivers”. In: *Journal of safety research* 63 (2017), pp. 57–60.

[8] Yoichi Morales et al. “Passenger discomfort map for autonomous navigation in a robotic wheelchair”. In: *Robotics and Autonomous Systems* 103 (2018), pp. 13–26.

[9] Alan H Bond and Les Gasser. “A subject-indexed bibliography of distributed artificial intelligence”. In: *IEEE transactions on systems, man, and cybernetics* 22.6 (1992), pp. 1260–1281.

[10] Alan H Bond and Les Gasser. *Readings in distributed artificial intelligence*. Morgan Kaufmann, 2014.

[11] Jérémy Boes and Frédéric Migeon. “Self-organizing multi-agent systems for the control of complex systems”. In: *Journal of Systems and Software* 134 (2017), pp. 12–28. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.08.038. URL: http://www.sciencedirect.com/science/article/pii/S0164121217301838.

[12] Ying Tan and Zhong-yang Zheng. “Research advance in swarm robotics”. In: *Defence Technology* 9.1 (2013), pp. 18–39.

[13] Salima Nebti and Abdallah Boukerram. “Swarm intelligence inspired classifiers for facial recognition”. In: *Swarm and Evolutionary Computation* 32 (2017), pp. 150–166.

[14] Michalis Mavrovouniotis, Changhe Li, and Shengxiang Yang. “A survey of swarm intelligence for dynamic optimization: Algorithms and applications”. In: *Swarm and Evolutionary Computation* 33 (2017), pp. 1–17.

[15] Lisa A Parr et al. “Recognizing facial cues: individual discrimination by chimpanzees (Pan troglodytes) and rhesus monkeys (Macaca mulatta).” In: *Journal of Comparative Psychology* 114.1 (2000), p. 47.

[16] Lisa A Parr and Frans BM de Waal. “Visual kin recognition in chimpanzees”. In: *Nature* 399.6737 (1999), p. 647.

[17] James A Shapiro. “Thinking about bacterial populations as multicellular organisms”. In: *Annual Reviews in Microbiology* 52.1 (1998), pp. 81–104.

[18] J William Costerton et al. “Microbial biofilms”. In: *Annual Reviews in Microbiology* 49.1 (1995), pp. 711–745.

[19] Hans G Wallraff and Hans Georg Wallraff. *Avian navigation: pigeon homing as a paradigm*. Springer Science & Business Media, 2005.

[20] Duncan E Jackson and Francis LW Ratnieks. “Communication in ants”. In: *Current biology* 16.15 (2006), R570–R574.

[21] Simon Goss et al. “Self-organized shortcuts in the Argentine ant”. In: *Naturwissenschaften* 76.12 (1989), pp. 579–581.

[22] Fabien Ravary et al. “Individual experience alone can generate lasting division of labor in ants”. In: *Current Biology* 17.15 (2007), pp. 1308–1312.

[23] Jerome Buhl et al. “From disorder to order in marching locusts”. In: *Science* 312.5778 (2006), pp. 1402–1406.

[24] Quentin Bone and Richard Moore. *Biology of fishes*. Taylor & Francis, 2008.

[25] TJ Pitcher, AE Magurran, and IJ Winfield. “Fish in larger shoals find food faster”. In: *Behavioral Ecology and Sociobiology* 10.2 (1982), pp. 149–151.

[26] Peter B Moyle and Joseph J Cech. *Fishes: an introduction to ichthyology*. 597. 2004.

[27] John RG Dyer et al. “Consensus decision making in human crowds”. In: *Animal Behaviour* 75.2 (2008), pp. 461–470.

[28] Davide Marocco and Stefano Nolfi. “Origins of communication in evolving robots”. In: *International Conference on Simulation of Adaptive Behavior*. Springer. 2006, pp. 789–803.

[29] Adam T Hayes, Alcherio Martinoli, and Rodney M Goodman. “Swarm robotic odor localization: Off-line optimization and validation with real robots”. In: *Robotica* 21.4 (2003), pp. 427–441.

[30] Marco A Montes de Oca et al. “Majority-rule opinion dynamics with differential latency: a mechanism for self-organized collective decision-making”. In: *Swarm Intelligence* 5.3-4 (2011), pp. 305–327.

[31] Alexander Scheidler et al. “The k-Unanimity Rule for Self-Organized Decision-Making in Swarms of Robots”. In: *IEEE transactions on cybernetics* 46.5 (2016), pp. 1175–1188.

[32] Gabriele Valentini, Heiko Hamann, and Marco Dorigo. “Efficient Decision-Making in a Self-Organizing Robot Swarm: On the Speed Versus Accuracy Trade-Off”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’15. Istanbul, Turkey: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 1305–1314. ISBN: 978-1-4503-3413-6. URL: http://dl.acm.org/citation.cfm?id=2772879.2773319.

[33] Jens Wawerla, Gaurav S Sukhatme, and Maja J Mataric. “Collective construction with multiple robots”. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2002, pp. 2696–2701.

[34] Justin Werfel, Yaneer Bar-Yam, and Radhika Nagpal. “Building patterned structures with robot swarms”. In: *IJCAI*. 2005, pp. 1495–1504.

[35] Michael Allwright et al. “SRoCS: Leveraging stigmergy on a multi-robot construction platform for unknown environments”. In: *International Conference on Swarm Intelligence*. Springer. 2014, pp. 158–169.

[36] Roderich Groß et al. “Autonomous self-assembly in a swarm-bot”. In: *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005)*. Springer. 2006, pp. 314–322.

[37] Elio Tuci et al. “Self-Assembly in Physical Autonomous Robots-the Evolutionary Robotics Approach.” In: *ALIFE*. 2008, pp. 616–623.

[38] Vito Trianni, Stefano Nolfi, and Marco Dorigo. “Cooperative hole avoidance in a swarm-bot”. In: *Robotics and Autonomous Systems* 54.2 (2006), pp. 97–103.

[39] Rehan O’Grady et al. “Self-assembly strategies in a group of autonomous mobile robots”. In: *Autonomous Robots* 28.4 (2010), pp. 439–455.

[40] S. Bashyal and G. K. Venayagamoorthy. “Human swarm interaction for radiation source search and localization”. In: *2008 IEEE Swarm Intelligence Symposium*. Sept. 2008, pp. 1–8. DOI: 10.1109/SIS.2008.4668287.

[41] P. Walker et al. “Human control of robot swarms with dynamic leaders”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1108–1113. DOI: 10.1109/IROS.2014.6942696.

[42] Andreas Kolling et al. “Human-swarm Interaction: An Experimental Study of Two Types of Interaction with Foraging Swarms”. In: *J. Hum.-Robot Interact.* 2.2 (June 2013), pp. 103–129. ISSN: 2163-0364. DOI: 10.5898/JHRI.2.2.Kolling. URL: https://doi.org/10.5898/JHRI.2.2.Kolling.

[43] Rehan O’Grady, Anders Lyhne Christensen, and Marco Dorigo. “SWARMORPH: multi-robot morphogenesis using directional self-assembly”. In: *IEEE Transactions on Robotics* 25.3 (2009), pp. 738–743.

[44] Manuele Brambilla et al. “A reliable distributed algorithm for group size estimation with minimal communication requirements”. In: *Advanced Robotics, 2009. ICAR 2009. International Conference on*. IEEE. 2009, pp. 1–6.

[45] Levent Bayındır. “A review of swarm robotics tasks”. In: *Neurocomputing* 172 (2016), pp. 292–321.

[46] Shiming Chen and Huajing Fang. “Modeling and behavior analysis of large-scale social foraging swarm”. In: *Control and Decision* 20.12 (2005), p. 1392.

[47] Gerardo Beni. “The concept of cellular robotic system”. In: *Intelligent Control, 1988. Proceedings., IEEE International Symposium on*. IEEE. 1988, pp. 57–62.

[48] Hajime Asama, Akihiro Matsumoto, and Yoshiki Ishida. “Design Of An Autonomous And Distributed Robot System: Actress.” In: *IROS*. Vol. 89. 1989, pp. 283–290.

[49] David Payton et al. “Pheromone robotics”. In: *Autonomous Robots* 11.3 (2001), pp. 319–324.

[50] David Payton, Regina Estkowski, and Mike Howard. “Compound behaviors in pheromone robotics”. In: *Robotics and Autonomous Systems* 44.3-4 (2003), pp. 229–240.

[51] Erol Şahin. “Swarm robotics: From sources of inspiration to domains of application”. In: *International workshop on swarm robotics*. Springer. 2004, pp. 10–20.

[52] James McLurkin and Jennifer Smith. “Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots”. In: *in 7th International Symposium on Distributed Autonomous Robotic Systems (DARS*. Citeseer. 2004.

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

[54] Ali E Turgut et al. “Self-organized flocking in mobile robot swarms”. In: *Swarm Intelligence* 2.2-4 (2008), pp. 97–120.

[55] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. “Kilobot: A low cost scalable robot system for collective behaviors”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 3293–3298.

[56] Jörg Seyfried et al. “The I-SWARM project: Intelligent small world autonomous robots for micro-manipulation”. In: *International Workshop on Swarm Robotics*. Springer. 2004, pp. 70–83.

[57] Samuel N Beshers and Jennifer H Fewell. “Models of division of labor in social insects”. In: *Annual review of entomology* 46.1 (2001), pp. 413–440.

[58] Vito Trianni et al. “Evolutionary swarm robotics: A theoretical and methodological itinerary from individual neuro-controllers to collective behaviours”. In: *The horizons of evolutionary robotics* 153 (2014).

[59] Régis Vincent et al. “Centibots: Large-Scale Autonomous Robotic Search and Rescue Experiment”. In: (2008).

[60] Marco Dorigo, Mauro Birattari, et al. “Swarm intelligence”. In: *Scholarpedia* 2.9 (2007), p. 1462.

[61] Paul E Rybski et al. “System architecture for versatile autonomous and teleoperated control of multiple miniature robots”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 3. IEEE. 2001, pp. 2917–2922.

[62] Patricia Suárez, Andrés Iglesias, and Akemi Gálvez. “Make robots be bats: specializing robotic swarms to the Bat algorithm”. In: *Swarm and Evolutionary Computation* (2018).

[63] Claudio O Vilão et al. “A single camera vision system for a humanoid robot”. In: *Robotics: SBR-LARS Robotics Symposium and Robocontrol (SBR LARS Robocontrol), 2014 Joint Conference on*. IEEE. 2014, pp. 181–186.

[64] Nikolai Gryaznov and Alexander Lopota. “Computer vision for mobile on-ground robotics”. In: *Procedia Engineering* 100 (2015), pp. 1376–1380.

[65] Davide Scaramuzza et al. “Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in GPS-denied environments”. In: *IEEE Robotics & Automation Magazine* 21.3 (2014), pp. 26–40.

[66] Guillem Alenyà, Sergi Foix, and Carme Torras. “ToF cameras for active vision in robotics”. In: *Sensors and Actuators A: Physical* 218 (2014), pp. 10–22.

[67] Chelsea Sabo et al. “A lightweight, inexpensive robotic system for insect vision”. In: *Arthropod structure & development* 46.5 (2017), pp. 689–702.

[68] Daniel Wahrmann et al. “Fast object approximation for real-time 3D obstacle avoidance with biped robots”. In: *Advanced Intelligent Mechatronics (AIM), 2016 IEEE International Conference on*. IEEE. 2016, pp. 38–45.

[69] Kimberly McGuire et al. “Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone.” In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1070–1076.

[70] Jeng-Han Li, Yi-Shing Ho, and Jia-Jie Huang. “Line Tracking with Pixy Cameras on a Wheeled Robot Prototype”. In: *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*. IEEE. 2018, pp. 1–2.

[71] Albert S Huang et al. “Visual odometry and mapping for autonomous flight using an RGB-D camera”. In: *Robotics Research*. Springer, 2017, pp. 235–252.

[72] Joseph W Starr and BY Lattimer. “Evidential sensor fusion of long-wavelength infrared stereo vision and 3D-LIDAR for rangefinding in fire environments”. In: *Fire Technology* 53.6 (2017), pp. 1961–1983.

[73] Han Woong Yoo et al. “MEMS-based lidar for autonomous driving”. In: *e & i Elektrotechnik und Informationstechnik* (2018), pp. 1–8.

[74] Ji Zhang and Sanjiv Singh. “Low-drift and real-time lidar odometry and mapping”. In: *Autonomous Robots* 41.2 (2017), pp. 401–416.

[75] Peter Kinnell et al. “Autonomous metrology for robot mounted 3D vision systems”. In: *CIRP Annals* 66.1 (2017), pp. 483–486.

[76] Filip Šuligoj et al. “Object tracking with a multiagent robot system and a stereo vision camera”. In: *Procedia Engineering* (2014), pp. 968–973.

[77] Marcos Ferreira et al. “Stereo-based real-time 6-DoF work tool tracking for robot programing by demonstration”. In: *The International Journal of Advanced Manufacturing Technology* 85.1-4 (2016), pp. 57–69.

[78] Stefano Pellegrini and Luca Iocchi. “Human Posture Tracking and Classification through Stereo Vision and 3D Model Matching”. In: *EURASIP Journal on Image and Video Processing* 2008.1 (Dec. 2007), p. 476151. ISSN: 1687-5281. DOI: 10.1155/2008/476151. URL: https://doi.org/10.1155/2008/476151.

[79] Hema Chengalvarayan Radhakrishnamurthy et al. “Stereo vision system for a bin picking adept robot”. In: *Malaysian Journal of Computer Science* 20.1 (2017), pp. 91–98.

[80] Suman Saha, Ashutosh Natraj, and Sonia Waharte. “A real-time monocular vision-based frontal obstacle detection and avoidance for low cost UAVs in GPS denied environment”.

In: *2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology*. IEEE. 2014, pp. 189–195.

[81] F Adib Yaghmaie, A Mobarhani, and HD Taghirad. "A new method for mobile robot navigation in dynamic environment: Escaping algorithm". In: *2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*. IEEE. 2013, pp. 212–217.

[82] Zhuhong Zhang, Shigang Yue, and Guopeng Zhang. "Fly visual system inspired artificial neural network for collision detection". In: *Neurocomputing* 153 (2015), pp. 221–234.

[83] Sergi Bermudez i Badia, Pawel Pyk, and Paul FMJ Verschure. "A fly-locust based neuronal control system applied to an unmanned aerial vehicle: the invertebrate neuronal principles for course stabilization, altitude control and collision avoidance". In: *The International Journal of Robotics Research* 26.7 (2007), pp. 759–772.

[84] Nicolas Franceschini et al. *Optic flow based visual guidance: from flying insects to miniature aerial vehicles*. INTECH Open Access Publisher, 2009.

[85] Mark Blanchard, F Claire Rind, and Paul FMJ Verschure. "Collision avoidance using a model of the locust LGMD neuron". In: *Robotics and Autonomous Systems* 30.1-2 (2000), pp. 17–38.

[86] Sergi Bermudez i Badia, Ulysses Bernardet, and Paul FMJ Verschure. "Non-linear neuronal responses as an emergent property of afferent networks: A case study of the locust lobula giant movement detector". In: *PLoS computational biology* 6.3 (2010).

[87] Ana Carolina Silva, Jorge Silva, and Cristina Peixoto dos Santos. "A modified LGMD based neural network for automatic collision detection". In: *Informatics in Control, Automation and Robotics*. Springer, 2014, pp. 217–233.

[88] Shigang Yue and F Claire Rind. "Collision detection in complex dynamic scenes using an LGMD-based visual neural network with feature enhancement". In: *IEEE transactions on neural networks* 17.3 (2006), pp. 705–716.

[89] Hongying Meng et al. "A modified model for the Lobula Giant Movement Detector and its FPGA implementation". In: *Computer vision and image understanding* 114.11 (2010), pp. 1238–1247.

[90] Reid R Harrison. "A biologically inspired analog IC for visual collision detection". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 52.11 (2005), pp. 2308–2318.

[91] Hirotsugu Okuno and Tetsuya Yagi. "A visually guided collision warning system with a neuromorphic architecture". In: *Neural networks* 21.10 (2008), pp. 1431–1438.

[92] Sergi Foix, Guillem Alenya, and Carme Torras. "Lock-in time-of-flight (ToF) cameras: A survey". In: *IEEE Sensors Journal* 11.9 (2011), pp. 1917–1926.

[93] Elena Stoykova et al. “3-D time-varying scene capture technologies—A survey”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 17.11 (2007), pp. 1568–1586.

[94] R Hartley and A Zisserman. “Multiple view geometry in computer vision, 2nd edn Cambridge University Press”. In: (2000).

[95] Mikhail Ivanov et al. “Data Exchange and Task of Navigation for Robotic Group”. In: *Machine Vision and Navigation*. Springer, 2020, pp. 389–430.

[96] O Yu Sergiyenko. “Optoelectronic system for mobile robot navigation”. In: *Optoelectronics, Instrumentation and Data Processing* 46.5 (2010), pp. 414–428.

[97] Julio C Rodriguez-Quinonez et al. “Surface recognition improvement in 3D medical laser scanner using Levenberg–Marquardt method”. In: *Signal Processing* 93.2 (2013), pp. 378–386.

[98] Luis C Basaca-Preciado et al. “Optical 3D laser measurement system for navigation of autonomous mobile robot”. In: *Optics and Lasers in Engineering* 54 (2014), pp. 159–169.

[99] O Yu Sergiyenko et al. “Data transferring model determination in robotic group”. In: *Robotics and Autonomous Systems* 83 (2016), pp. 251–260.

[100] Lars Lindner et al. “Machine vision system for UAV navigation”. In: *Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles & International Transportation Electrification Conference (ESARS-ITEC), International Conference on*. IEEE. 2016, pp. 1–6.

[101] Lars Lindner et al. “Exact laser beam positioning for measurement of vegetation vitality”. In: *Industrial Robot: An International Journal* 44.4 (2017), pp. 532–541.

[102] Oleg Sergiyenko et al. “Remote sensor for spatial measurements by using optical scanning”. In: *Sensors* 9.7 (2009), pp. 5477–5492.

[103] Luis C Básaca et al. “Resolution improvement of dynamic triangulation method for 3D vision system in robot navigation task”. In: *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE. 2010, pp. 2886–2891.

[104] BK Patle et al. “Matrix-Binary Codes based Genetic Algorithm for path planning of mobile robot”. In: *Computers & Electrical Engineering* 67 (2018), pp. 708–728.

[105] Oleg Sergiyenko et al. *Transferring model in robotic group*. Santa Clara: 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE), 2016. DOI: 10.1109/isie.2016.7745018.

[106] Luis C. Basaca-Preciado et al. “Optical 3D laser measurement system for navigation of autonomous mobile robot”. In: *Optics and Lasers in Engineering* 54 (2014), pp. 159–169. ISSN: 0143-8166. DOI: 10.1016/j.optlaseng.2013.08.005.

[107] JC Rodriguez-Quinonez et al. “Improve 3D laser scanner measurements accuracy using a FFBP neural network with Widrow-Hoff weight/bias learning function”. In: *Opto-Electronics Review* 22.4 (2014), pp. 224–235.

[108] JC Rodriguez-Quinonez et al. “Improve a 3D distance measurement accuracy in stereo vision systems using optimization methods’ approach”. In: *Opto-Electronics Review* 25.1 (2017), pp. 24–32.

[109] XM Garcia-Cruz et al. “Optimization of 3D laser scanning speed by use of combined variable step”. In: *Optics and Lasers in Engineering* 54 (2014), pp. 141–151.

[110] Lars Lindner et al. “Continuous 3D scanning mode using servomotors instead of stepping motors in dynamic laser triangulation”. In: *Industrial Electronics (ISIE), 2015 IEEE 24th International Symposium on*. IEEE. 2015, pp. 944–949.

[111] Lars Lindner et al. “Mobile robot vision system using continuous laser scanning for industrial application”. In: *Industrial Robot: An International Journal* 43.4 (2016), pp. 360–369.

[112] Mykhailo Ivanov et al. “Mobile Robot Path Planning Using Continuous Laser Scanning”. In: *Optoelectronics in Machine Vision-Based Theories and Applications*. IGI Global, 2019, pp. 338–372.

[113] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. “Ant algorithms and stigmergy”. In: *Future Generation Computer Systems* 16.8 (2000), pp. 851–871.

[114] *Swarm Communication. Jasmine: swarm robot platform*. 2015. URL: http://www.swarmrobot.org/Communication.html (visited on 09/30/2015).

[115] Bart Braem et al. “The wireless autonomous spanning tree protocol for multihop wireless body area networks”. In: *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual International Conference on*. IEEE. 2006, pp. 1–8.

[116] D Fedyk et al. *IS-IS extensions supporting IEEE 802.1 aq shortest path bridging*. Tech. rep. 2012.

[117] Neil Briscoe. “Understanding the OSI 7-layer model”. In: *PC Network Advisor* 120.2 (2000), pp. 13–16.

[118] Yadong Li et al. “Research based on OSI model”. In: *2011 IEEE 3rd International Conference on Communication Software and Networks*. IEEE. 2011, pp. 554–557.

[119] David J Grymin, Charles B Neas, and Mazen Farhood. “A hierarchical approach for primitive-based motion planning and control of autonomous vehicles”. In: *Robotics and Autonomous Systems* 62.2 (2014), pp. 214–228.

[120] Hung T Nguyen and Elbert A Walker. *A first course in fuzzy logic*. CRC press, 2005.

[121] Regis Vincent et al. “Centibots: Large-scale autonomous robotic search and rescue experiment”. In: *2nd International Joint Topical Meeting on Emergency Preparedness & Response and Robotics & Remote Systems* (2008).

[122] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[123] Anany Levitin. *Introduction to the design & analysis of algorithms*. Boston: Pearson, 2012.

[124] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[125] Eric Bonabeau et al. *Swarm intelligence: from natural to artificial systems*. 1. Oxford university press, 1999.

[126] Bence Kovács et al. “A novel potential field method for path planning of mobile robots by adapting animal motion attributes”. In: *Robotics and Autonomous Systems* 82 (2016), pp. 24–34.

[127] Stefan Hougardy. “The Floyd–Warshall algorithm on graphs with negative cycles”. In: *Information Processing Letters* 110.8-9 (2010), pp. 279–281.

[128] Santiago Manen, Matthieu Guillaumin, and Luc Van Gool. “Prime object proposals with randomized prim’s algorithm”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2536–2543.

[129] Laurent Najman, Jean Cousty, and Benjamin Perret. “Playing with kruskal: algorithms for morphological trees in edge-weighted graphs”. In: *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. Springer. 2013, pp. 135–146.

[130] Hwan Il Kang, Byunghee Lee, and Kabil Kim. “Path planning algorithm using the particle swarm optimization and the improved Dijkstra algorithm”. In: *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*. Vol. 2. IEEE. 2008, pp. 1002–1004.

[131] Fan Hsun Tseng et al. “A star search algorithm for civil UAV path planning with 3G communication”. In: *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE. 2014, pp. 942–945.

[132] Junfeng Yao et al. “Path planning for virtual human motion using improved A* star algorithm”. In: *2010 Seventh international conference on information technology: new generations*. IEEE. 2010, pp. 1154–1158.

[133] Sarah Alnasser and Hachemi Bennaceur. “An efficient genetic algorithm for the global robot path planning problem”. In: *2016 Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP)*. IEEE. 2016, pp. 97–102.

[134] Lydia E Kavraki and Jean-Claude Latombe. “Probabilistic roadmaps for robot path planning”. In: (1998).

[135] Matthew Baumann et al. “Path planning for improved visibility using a probabilistic road map”. In: *IEEE Transactions on Robotics* 26.1 (2010), pp. 195–200.

[136] Karl Berntorp. “Path planning and integrated collision avoidance for autonomous vehicles”. In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 4023–4028.

[137] Ritu Tiwari. *Intelligent planning for mobile robotics: algorithmic approaches: algorithmic approaches*. IGI Global, 2012.

[138] Vladimir Lumelsky and Alexander Stepanov. “Dynamic path planning for a mobile automaton with limited information on the environment”. In: *IEEE transactions on Automatic control* 31.11 (1986), pp. 1058–1063.

[139] A Sankaranarayanan and M Vidyasagar. “A new path planning algorithm for moving a point object amidst unknown obstacles in a plane”. In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE. 1990, pp. 1930–1936.

[140] A Sankaranarayanar and M Vidyasagar. “Path planning for moving a point object amidst unknown obstacles in a plane: a new algorithm and a general theory for algorithm development”. In: *29th IEEE Conference on Decision and Control*. IEEE. 1990, pp. 1111–1119.

[141] Ishay Kamon and Ehud Rivlin. “Sensory-based motion planning with global proofs”. In: *IEEE transactions on Robotics and Automation* 13.6 (1997), pp. 814–822.

[142] H Noborio. “A path-planning algorithm for generation of an intuitively reasonable path in an uncertain 2D workspace”. In: *Proc. of the Japan-USA Symp. on Flexible Automation*. Vol. 3. 1990, pp. 477–480.

[143] Hiroshi Noborio, Keiichi Fujimura, and Yohei Horiuchi. “A comparative study of sensor-based path-planning algorithms in an unknown maze”. In: *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*. Vol. 2. IEEE. 2000, pp. 909–916.

[144] Ishay Kamon, Elon Rimon, and Ehud Rivlin. “Tangentbug: A range-sensor-based navigation algorithm”. In: *The International Journal of Robotics Research* 17.9 (1998), pp. 934–953.

[145] Mandyam V Srinivasan et al. “Vision and Navigation in Insects, and Applications to Aircraft Guidance”. In: (2014).

[146] Mandyam V Srinivasan et al. “From biology to engineering: insect vision and applications to robotics”. In: *Frontiers in sensing*. Springer, 2012, pp. 19–39.

[147] František Duchoň et al. “Path planning with modified a star algorithm for a mobile robot”. In: *Procedia Engineering* 96 (2014), pp. 59–69.

[148] Pierre E. Bézier. *How Renault Uses Numerical Control for Car Body Design and Tooling*. Detroit, MI, USA: Society of Automotive Engineers, 1968. DOI: 10.4271/680010.

[149] Pierre Bezier. "Example of an Existing System in the Motor Industry: The Unisurf System". In: *Proceedings of the Royal Society of London* A321 (1971), pp. 207–218.

[150] Kuniaki Kawabata et al. "A path generation for automated vehicle based on Bezier curve and via-points". In: *Robotics and Autonomous Systems* 74 (2015), pp. 243–252.

[151] L. Han et al. "Bézier curve based path planning for autonomous vehicle in urban environment". In: *2010 IEEE Intelligent Vehicles Symposium*. June 2010, pp. 1036–1042. DOI: 10.1109/IVS.2010.5548085.

[152] Bhargav Jha, Vladimir Turetsky, and Tal Shima. "Robust Path Tracking by a Dubins Ground Vehicle". In: *IEEE Transactions on Control Systems Technology* (2018).

[153] J. Rosenblatt. "DAMN: Distributed Architecture for Mobile Navigation". In: *Journal of Experimental and Theoretical Artificial Intelligence* 9 (2-3 1997), pp. 339–360.

[154] Paul Levi and Serge Kernbach. *Symbiotic multi-robot organisms: reliability, adaptability, evolution*. Vol. 7. Springer Science & Business Media, 2010.

[155] Erol Sahin and Alan FT Winfield. "Special issue on swarm robotics." In: *Swarm Intelligence* 2.2-4 (2008), pp. 69–72.

[156] Miguel Duarte et al. "JBotEvolver: A versatile simulation platform for evolutionary robotics". In: *Proceedings of the 14th International Conference on the Synthesis & Simulation of Living Systems. MIT Press, Cambridge, MA*. Citeseer. 2014, pp. 210–211.

[157] Brett Browning and Erick Tryzelaar. "Übersim: a multi-robot simulator for robot soccer". In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM. 2003, pp. 948–949.

[158] Su Zhibao, Zhang Haojie, and Zhu Sen. "A robotic simulation system combined USARSim and RCS library". In: *Intelligent Robot Systems (ACIRS), 2017 2nd Asia-Pacific Conference on*. IEEE. 2017, pp. 240–243.

[159] Jon Klein and Lee Spector. "3d multi-agent simulations in the breve simulation environment". In: *Artificial life models in software*. Springer, 2009, pp. 79–106.

[160] Eric Rohmer, Surya PN Singh, and Marc Freese. "V-REP: A versatile and scalable robot simulation framework". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 1321–1326.

[161] Olivier Michel. "Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation". In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5. DOI: 10.5772/5618.

[162] Fadri Furrer et al. “Rotors—A modular gazebo mav simulator framework”. In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[163] Carlo Pinciroli et al. “ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics”. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 5027–5034.

[164] Okan Aşık and H Levent Akın. “Solving multi-agent decision problems modeled as dec-pomdp: A robot soccer case study”. In: *RoboCup 2012: Robot Soccer World Cup XVI*. Springer, 2013, pp. 130–140.

[165] Patrick Felicia. “Unity 5 from Proficiency to Mastery: Artificial Intelligence”. In: (2017).

[166] Yong Deng et al. “Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment”. In: *Applied Soft Computing* 12 (3 2012), pp. 1231–1237.

[167] Elias K. Xidias and Philip N. Azariadis. “Computing collision-free motions for a team of robots using formation and non-holonomic constraints”. In: *Robotics and Autonomous Systems* 82 (Aug. 2016), pp. 15–23. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.04.008.

[168] Israel Lugo-Cárdenas et al. “Dubins path generation for a fixed wing UAV”. In: *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*. IEEE. 2014, pp. 339–346.

[169] Nare Karapetyan et al. “Multi-robot Dubins Coverage with Autonomous Surface Vehicles”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2373–2379.

[170] Zhu Wang et al. “Enhanced sparse A* search for UAV path planning using dubins path estimation”. In: *Control Conference (CCC), 2014 33rd Chinese*. IEEE. 2014, pp. 738–742.

[171] Claude Elwood Shannon. “A mathematical theory of communication”. In: *ACM SIGMOBILE mobile computing and communications review* 5.1 (2001), pp. 3–55.

[172] Erich Schubert et al. “DBSCAN revisited, revisited: why and how you should (still) use DBSCAN”. In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), p. 19.

Appendix A

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class Unit : MonoBehaviour {

 public int id;
 public bool reached { get; set; }
 const float minPathUpdateTime = .2f;
 const float pathUpdateMoveThreshold = .5f;
 public Transform[] targets;
 int targetID = 0;
 Transform target;

 float speed;
```

```
Path path;


 List<Vector3> playerPos = new List<Vector3>();
 LineRenderer lineRenderer;


 void Start() {


  System.IO.Directory.CreateDirectory(config.path
  + config.mode
  + "\\R" + GetComponent<Unit>().id.ToString()
  + "\\traks\\");



  string trackHead = "time,x,y,z,roll,pitch,yaw"+System.Environment.NewLine;


  System.IO.File.AppendAllText(config.path
  + config.mode
  + "\\R" + GetComponent<Unit>().id.ToString()
  + "\\traks\\"
  + "\\TE" + config.experiment.ToString() + ".txt", trackHead);
```

```
  Color red = Color.red;
  lineRenderer = GetComponent<LineRenderer>();
  lineRenderer.startColor = red;
  lineRenderer.endColor = red;
  lineRenderer.startWidth = 0.5f;
  lineRenderer.endWidth = 0.5f;

  speed = config.speedLow;
  target = targets[targetID];

  InvokeRepeating("RequestPath", 2.0f, 0.3f);
 }

public void OnPathFound(Vector3[] waypoints, bool pathSuccessful) {

  if (pathSuccessful) {
path = new Path(waypoints, transform.position,
config.turnDst, config.stoppingDst);
StopCoroutine("FollowPath");
StartCoroutine("FollowPath");
  }
```

```
}

 void Update()
 {
  playerPos.Add(transform.position);
  lineRenderer.numPositions = playerPos.Count;
  for (int i = 0; i < playerPos.Count; i++)
  {
lineRenderer.SetPosition(i, playerPos[i]);
  }

  string track = Time.realtimeSinceStartup.ToString() + ","
   + transform.position.x.ToString() + ","
   + transform.position.y.ToString() + ","
   + transform.position.z.ToString() + ","
   + transform.rotation.x.ToString() + ","
   + transform.rotation.y.ToString() + ","
   + transform.rotation.z.ToString()
   + System.Environment.NewLine;

  System.IO.File.AppendAllText(config.path
```

```
  + config.mode
  + "\\R" + GetComponent<Unit>().id.ToString()
  + "\\traks\\"
  + "\\TE" + config.experiment.ToString() + ".txt", track);
 }
 public void RequestPath()
 {
  if (Vector3.Distance(transform.position, target.position) < 5
  && targetID < (targets.Length - 1))
  {
targetID++;
target = targets[targetID];
  }

  if (Vector3.Distance(transform.position, target.position) < 5
  && targetID == (targets.Length - 1) )
  {
reached = true;
  }

  GameObject.Find("Grid").GetComponent<PathRequestManager>()
.RequestPath(new PathRequest(transform.position, target.position, OnPathFound));
```

```
 }

IEnumerator FollowPath() {

bool followingPath = true;
int pathIndex = 0;

float speedPercent = 1;

  float obstProx = GetComponent<TVSRay>().obstacleProximity;
  float maxVis = config.viewRadius;




  if (config.speedControll)
  {
if (obstProx > 2 * maxVis / 3)
 speed = config.speedMax;

if ((obstProx > (maxVis / 3)) & (obstProx <= (2 * maxVis / 3)))
 speed = config.speedMed;
```

```
if (obstProx > 0 & obstProx <= (maxVis / 3))
 speed = config.speedLow;
  }

  while (followingPath) {
Vector2 pos2D = new Vector2 (transform.position.x, transform.position.z);
while (path.turnBoundaries [pathIndex].HasCrossedLine (pos2D)) {
if (pathIndex == path.finishLineIndex) {
followingPath = false;
break;
} else {
pathIndex++;
}
}

if (followingPath) {

if (pathIndex >= path.slowDownIndex && config.stoppingDst > 0) {
speedPercent = Mathf.Clamp01 (
path.turnBoundaries [path.finishLineIndex].DistanceFromPoint (pos2D) /
config.stoppingDst);
if (speedPercent < 0.01f) {
```

```
followingPath = false;
}
}

Quaternion targetRotation =
Quaternion.LookRotation (path.lookPoints [pathIndex] - transform.position);
transform.rotation =
Quaternion.Lerp (transform.rotation, targetRotation,
Time.deltaTime * config.turnSpeed);
transform.Translate
(Vector3.forward * Time.deltaTime * speed * speedPercent, Space.Self);
}

yield return null;

}
}

public void OnDrawGizmos() {
if (path != null) {
path.DrawWithGizmos(transform.position);
  }
```

```
 }



}



using System;

using System.Collections.Generic;

using UnityEngine;


class TVSRay : MonoBehaviour

{

 public LayerMask obstacleMask;

 public float vertDispAngle = 90;


 float angle_ver;


 int stepCount;

 float stepAngleSize;

 //float angle;

 float angle2;
```

```
public GameObject commonGrid;



string scan;
string frames = "";
public float obstacleProximity { get; set; }


void Awake()
{
 stepCount = Mathf.RoundToInt(config.viewAngle * config.meshResolution);
 stepAngleSize = config.viewAngle / stepCount;
}
void Start()
{


 System.IO.Directory.CreateDirectory(config.path
 + config.mode
 + "\\R" + GetComponent<Unit>().id.ToString()
 + "\\proximities\\");
 System.IO.Directory.CreateDirectory(config.path
 + config.mode
 + "\\R" + GetComponent<Unit>().id.ToString()
```

```
 + "\\frames\\");
 //frames = "{"+ System.Environment.NewLine+"\t\"frames\":["+
 System.Environment.NewLine;
 //frames = "[";//+ System.Environment.NewLine;
}

void OnApplicationQuit()
{
 //frames = frames.Remove(frames.Length - 3) + System.Environment.NewLine;
 //frames += "\t]"+ System.Environment.NewLine;

 System.IO.File.AppendAllText(config.path
 + config.mode
 + "\\R" + GetComponent<Unit>().id.ToString()
 + "\\frames\\"
 + "\\FE" + config.experiment.ToString() + ".json", frames);
}
void LateUpdate()
{
 DrawFieldOfView();
 DrawFieldOfViewBeam();
}
```

```
 ViewCastInfo ViewCast(float globalAngle)
 {
  Vector3 dir = DirFromAngle(transform.rotation.y + globalAngle, true);
  RaycastHit hit;
  obstacleProximity =
  Vector3.Distance(transform.position, transform.position
  + dir * config.viewRadius);

  if (Physics.Raycast(transform.position, dir, out hit,
  config.viewRadius, obstacleMask))
  {
   Debug.DrawLine(transform.position, hit.point, Color.blue);
   Vector3 wp = new Vector3(hit.point.x, hit.point.y, hit.point.z);
   commonGrid.GetComponent<Grid>().
    UpdateGrid(wp, GetComponent<Unit>().id, false);
   obstacleProximity = Vector3.Distance(transform.position, wp);
   return new ViewCastInfo(true, hit.point, hit.distance, globalAngle);
  }
  else
  {
Debug.DrawLine(transform.position, transform.position +
```

```
dir * config.viewRadius, Color.blue);
return new ViewCastInfo(false, transform.position +
dir * config.viewRadius, config.viewRadius, globalAngle);
  }
 }

 ViewCastInfo ViewCastBeam(float globalAngle)
 {
  Vector3 dir = DirFromAngleBeam(transform.rotation.y + globalAngle, true);
  RaycastHit hit;
  obstacleProximity =
  Vector3.Distance(transform.position, transform.position +
  dir * config.viewRadius);

  if (Physics.Raycast(transform.position, dir, out hit,
  config.viewRadius, obstacleMask))
  {
   Debug.DrawLine(transform.position, hit.point, Color.red);
   Vector3 wp = new Vector3(hit.point.x, hit.point.y, hit.point.z);
   obstacleProximity = Vector3.Distance(transform.position, wp);
   return new ViewCastInfo(true, hit.point, hit.distance, globalAngle);
  }
```

```
 else
 {
  Debug.DrawLine(transform.position, transform.position +
  dir * config.viewRadius, Color.red);
  return new ViewCastInfo(false, transform.position +
  dir * config.viewRadius, config.viewRadius, globalAngle);
 }
}

 public Vector3 DirFromAngle(float angleInDegrees, bool angleIsGlobal)
 {
  if(!angleIsGlobal)
  {
angleInDegrees += transform.eulerAngles.y;
  }
  return new Vector3(Mathf.Sin(angleInDegrees * Mathf.Deg2Rad),
       0, Mathf.Cos(angleInDegrees * Mathf.Deg2Rad));
 }

 public Vector3 DirFromAngleBeam(float angleInDegrees, bool angleIsGlobal)
 {
  if (!angleIsGlobal)
```

```
  {
angleInDegrees += transform.eulerAngles.y;
  }
  return new Vector3(Mathf.Sin(angleInDegrees * Mathf.Deg2Rad),
    Mathf.Cos(angle_ver * Mathf.Deg2Rad),
    Mathf.Cos(angleInDegrees * Mathf.Deg2Rad));
 }


 public struct ViewCastInfo
 {
  public bool hit;
  public Vector3 point;
  public float dst;
  public float angle;


  public ViewCastInfo(bool _hit, Vector3 _point, float _dst, float _angle)
  {
hit = _hit;
point = _point;
dst = _dst;
angle = _angle;
  }
```

```
 }

 void DrawFieldOfView()
 {
  int stepCount = Mathf.RoundToInt(config.viewAngle
  * config.meshResolution);
  float stepAngleSize = config.viewAngle / stepCount;
  List<Vector3> viewPoints = new List<Vector3>();
  for (int i = 0; i <= stepCount; i++)
  {
float angle = transform.eulerAngles.y - config.viewAngle / 2 + stepAngleSize * i;
ViewCastInfo newViewCast = ViewCast(angle);
viewPoints.Add(newViewCast.point);
  }
 }

 void DrawFieldOfViewBeam()
 {
  int stepCount = Mathf.RoundToInt(config.viewAngle * config.meshResolution);
  int stepCount_ver = Mathf.RoundToInt(config.vertAngle * config.meshResolution);

  float stepAngleSize = config.viewAngle / stepCount;
```

```
float stepAngleSize_ver = config.vertAngle / stepCount_ver;

List<Vector3> viewPoints = new List<Vector3>();

//string proximities = "";

string frame = "";


frame +="\t{" ;//+ System.Environment.NewLine;

frame += "\t\"time\":"+ "\"" +

DateTime.Now.ToString("hh.mm.ss.ffffff").ToString()+ "\",";// +

System.Environment.NewLine;

frame += "\t\"position\":["

+ transform.position.x.ToString("G4") + ","

+ transform.position.y.ToString("G4") + ","

+ transform.position.z.ToString("G4")

+ "],";

//+ System.Environment.NewLine;

frame += "\t\"roation\":["

+ transform.rotation.x.ToString("G4")+","

+ transform.rotation.y.ToString("G4") + ","

+ transform.rotation.z.ToString("G4")

+ "]," ;

//+ System.Environment.NewLine;

frame +="\t\"points\":[";//+ System.Environment.NewLine;
```

```
  for (int i = 0; i <= stepCount; i++)
  {
for (int j = 0; j <= stepCount_ver; j++)
{
 float angle = transform.eulerAngles.y - config.viewAngle / 2
 + stepAngleSize * i;
 angle_ver = -transform.eulerAngles.x + vertDispAngle -
 config.vertAngle / 2 + stepAngleSize_ver * j;
 ViewCastInfo newViewCast = ViewCastBeam(angle);
 viewPoints.Add(newViewCast.point);
 if(obstacleProximity < (config.viewRadius-0.1)){
    frame += "\t\t["
  + newViewCast.point.x.ToString("G4") + ","
  + newViewCast.point.y.ToString("G4") + ","
  + newViewCast.point.z.ToString("G4") + ","
  + obstacleProximity.ToString("G4")
  + "],";// + System.Environment.NewLine;
 }

}
  }
```

```
 //frame = frame.Remove(frame.Length - 3) + System.Environment.NewLine;
 frame += "\t\t]";// + System.Environment.NewLine;
 frame += "\t}" + System.Environment.NewLine;

 frames += frame;
 /*
 proximities = proximities.Remove(proximities.Length - 1) +
 System.Environment.NewLine;
 Store proximities in seperate file
 System.IO.File.AppendAllText(config.path
 + config.mode
 + "\\R" + GetComponent<Unit>().id.ToString()
 + "\\proximities\\"
 + "\\PE" + config.experiment.ToString() + ".txt", proximities);

 System.IO.File.AppendAllText(config.path
 + config.mode
 + "\\R" + GetComponent<Unit>().id.ToString()
 + "\\frames\\"
 + "\\FE" + config.experiment.ToString() + ".txt", frame);
 */
}
```

```
 public struct EdgeInfo
 {
  public Vector3 pointA;
  public Vector3 pointB;

  public EdgeInfo(Vector3 _pointA, Vector3 _pointB)
  {
pointA = _pointA;
pointB = _pointB;
  }
 }
}
```

```
using UnityEngine;
using System.Collections.Generic;
using System.Diagnostics;
using System;

public class Pathfinding : MonoBehaviour {
```

```
Grid grid;
 float displacementY = 0.9f;
void Awake() {
grid = GetComponent<Grid>();
}



public void FindPath(PathRequest request, Action<PathResult> callback) {


Stopwatch sw = new Stopwatch();
sw.Start();


Vector3[] waypoints = new Vector3[0];
bool pathSuccess = false;


Node startNode = grid.NodeFromWorldPoint(request.pathStart);
Node targetNode = grid.NodeFromWorldPoint(request.pathEnd);
startNode.parent = startNode;


if (startNode.walkable && targetNode.walkable) {
Edge<Node> openSet = new Edge<Node>(grid.MaxSize);
HashSet<Node> closedSet = new HashSet<Node>();
```

```
openSet.Add(startNode);


while (openSet.Count > 0) {
Node currentNode = openSet.RemoveFirst();
closedSet.Add(currentNode);


if (currentNode == targetNode) {
sw.Stop();
//print ("Path found: " + sw.ElapsedMilliseconds + " ms");
pathSuccess = true;
break;
}


foreach (Node neighbour in grid.GetNeighbours(currentNode)) {
if (!neighbour.walkable || closedSet.Contains(neighbour)) {
continue;
}


  int newCToN = currentNode.gCost + GetDistance(currentNode, neighbour)
  + neighbour.movementPenalty;
if (newCToN < neighbour.gCost ||
 !openSet.Contains(neighbour)) {
```

```
neighbour.gCost = newCToN;
neighbour.hCost = GetDistance(neighbour, targetNode);
neighbour.parent = currentNode;

if (!openSet.Contains(neighbour))
openSet.Add(neighbour);
else
openSet.UpdateItem(neighbour);
}
}
}
}
if (pathSuccess) {
waypoints = RetracePath(startNode,targetNode);
pathSuccess = waypoints.Length > 0;
}
callback (new PathResult (waypoints, pathSuccess, request.callback));

}



Vector3[] RetracePath(Node startNode, Node endNode) {
```

```
List<Node> path = new List<Node>();
Node currentNode = endNode;

while (currentNode != startNode) {
path.Add(currentNode);
currentNode = currentNode.parent;
}
  Vector3[] waypoints;

  if (config.fullNodeList)
  {
waypoints = OriginalPath(path);
  }
  else
  {
waypoints = SimplifyPath(path);
  }

  Array.Reverse(waypoints);
return waypoints;

}
```

```
Vector3[] SimplifyPath(List<Node> path)
 {
  List<Vector3> waypoints = new List<Vector3>();
Vector2 directionOld = Vector2.zero;

for (int i = 1; i < path.Count; i ++) {
Vector2 directionNew =
new Vector2(path[i-1].gridX - path[i].gridX,
path[i-1].gridY - path[i].gridY);
if (directionNew != directionOld) {
     waypoints.Add(new Vector3(path[i].worldPosition.x,
     displacementY, path[i].worldPosition.z));
}
directionOld = directionNew;
}
return waypoints.ToArray();
 }

 Vector3[] OriginalPath(List<Node> path)
 {
  List<Vector3> waypoints = new List<Vector3>();
```

```
  Vector2 directionOld = Vector2.zero;


  for (int i = 1; i < path.Count; i++)
  {
waypoints.Add(new Vector3(path[i].worldPosition.x, displacementY,
path[i].worldPosition.z));
  }
  return waypoints.ToArray();
 }


 int GetDistance(Node nodeA, Node nodeB) {
int dstX = Mathf.Abs(nodeA.gridX - nodeB.gridX);
int dstY = Mathf.Abs(nodeA.gridY - nodeB.gridY);


if (dstX > dstY)
return 14*dstY + 10* (dstX-dstY);
return 14*dstX + 10 * (dstY-dstX);
}




}
```

```
using UnityEngine;
using System.Collections.Generic;
using System;
using System.Threading;

public class PathRequestManager : MonoBehaviour {

Queue<PathResult> results = new Queue<PathResult>();

static PathRequestManager instance;
Pathfinding pathfinding;

void Awake() {
instance = this;
pathfinding = GetComponent<Pathfinding>();
}

void Update() {
if (results.Count > 0) {
```

```
int itemsInQueue = results.Count;
lock (results) {
for (int i = 0; i < itemsInQueue; i++) {
PathResult result = results.Dequeue ();
result.callback (result.path, result.success);
}
}
}
}

public void RequestPath(PathRequest request) {
ThreadStart threadStart = delegate {
instance.pathfinding.FindPath (request, instance.FinishedProcessingPath);
};
threadStart.Invoke();
}

public void FinishedProcessingPath(PathResult result) {
lock (results) {
results.Enqueue (result);
}
}
```

```
}

public struct PathResult {
public Vector3[] path;
public bool success;
public Action<Vector3[], bool> callback;

public PathResult (Vector3[] path, bool success, Action<Vector3[], bool> callback)
{
this.path = path;
this.success = success;
this.callback = callback;
}

}

public struct PathRequest {
public Vector3 pathStart;
public Vector3 pathEnd;
public Action<Vector3[], bool> callback;
```

```
public PathRequest(Vector3 _start, Vector3 _end, Action<Vector3[], bool> _callback) {
pathStart = _start;
pathEnd = _end;
callback = _callback;
}

}



using UnityEngine;

public class Path {

public readonly Vector3[] lookPoints;
public readonly Line[] turnBoundaries;
public readonly int finishLineIndex;
public readonly int slowDownIndex;

public Path(Vector3[] waypoints, Vector3 startPos,
turnDst, float stoppingDst) {
lookPoints = waypoints;
turnBoundaries = new Line[lookPoints.Length];
```

```
finishLineIndex = turnBoundaries.Length - 1;


Vector2 previousPoint = V3ToV2 (startPos);
for (int i = 0; i < lookPoints.Length; i++) {
Vector2 currentPoint = V3ToV2 (lookPoints [i]);
Vector2 dirToCurrentPoint = (currentPoint - previousPoint).normalized;
Vector2 turnBoundaryPoint =
(i == finishLineIndex)?currentPoint : currentPoint -
dirToCurrentPoint * turnDst;
turnBoundaries [i] =
new Line (turnBoundaryPoint, previousPoint -
dirToCurrentPoint * turnDst);
previousPoint = turnBoundaryPoint;
}


float dstFromEndPoint = 0;
for (int i = lookPoints.Length - 1; i > 0; i--) {
dstFromEndPoint += Vector3.Distance (lookPoints [i], lookPoints [i - 1]);
if (dstFromEndPoint > stoppingDst) {
slowDownIndex = i;
break;
}
```

```
}
}

Vector2 V3ToV2(Vector3 v3) {
return new Vector2 (v3.x, v3.z);
}

public void DrawWithGizmos(Vector3 pos) {

Gizmos.color = Color.black;

  for (int i = 0; i < lookPoints.Length; i++)
  {
Gizmos.color = Color.black;
Gizmos.DrawCube(lookPoints[i], Vector3.one);
if (i == 0)
{
 Gizmos.DrawLine(pos, lookPoints[i]);
}
else
{
 Gizmos.DrawLine(lookPoints[i - 1], lookPoints[i]);
```

```
}
  }

  Gizmos.color = Color.white;
foreach (Line l in turnBoundaries) {
l.DrawWithGizmos (10);
}

 }

}
```

```
using UnityEngine;
public class Node : IEdgeItem<Node> {

public bool walkable;
public Vector3 worldPosition;
public int gridX;
public int gridY;
public int movementPenalty;
```

```
public int gCost;

public int hCost;

public Node parent;

int edgeIndex;


public Node(bool _walkable, Vector3 _worldPos,

int _gridX, int _gridY, int _penalty) {

walkable = _walkable;

worldPosition = _worldPos;

gridX = _gridX;

gridY = _gridY;

movementPenalty = _penalty;

}


public int fCost {

get {

return gCost + hCost;

}

}


public int EdgeIndex {

get {
```

```
return edgeIndex;
}
set {
edgeIndex = value;
}
}

public int CompareTo(Node nodeToCompare) {
int compare = fCost.CompareTo(nodeToCompare.fCost);
if (compare == 0) {
compare = hCost.CompareTo(nodeToCompare.hCost);
}
return -compare;
}
}

static class config
{
 //GENERAL
 public static string path = "E:\\experiments\\mode_";
 public static string scene = "Scene_6";
 public static string mode = "Group";
```

```
public static int experiments = 1;


public static int experiment = 1;
public static int activeRobotID = 0;


//UNIT
public static bool speedControll = false;
public static bool fullNodeList = false;


public static float speedMax = 7;
public static float speedMed = 4;
public static float speedLow = 7;


public static float turnSpeed = 4;
public static float turnDst = 1;
public static float stoppingDst = 5;


//TVS
public static float viewRadius = 20;
public static float viewAngle = 90;
public static float vertAngle = 20;
public static float meshResolution = 0.3f;
```

```
 public static float nodeRadius = 0.7f;
}

using UnityEngine;
using UnityEngine.SceneManagement;

class Monitor : MonoBehaviour
{
 public GameObject[] robots;
 public GameObject grid;

 void Start()
 {
  robots = GameObject.FindGameObjectsWithTag("robot");

  System.IO.Directory.CreateDirectory(config.path +
  config.mode + "\\common" + "\\maps\\");

  System.IO.Directory.CreateDirectory(config.path +
  config.mode + "\\R1" + "\\maps\\");
  System.IO.Directory.CreateDirectory(config.path +
```

```
  config.mode + "\\R2" + "\\maps\\");
  System.IO.Directory.CreateDirectory(config.path +
  config.mode + "\\R3" + "\\maps\\");

  if (config.mode.Equals("Group")|| config.mode.Equals("Preknown"))
  {
InvokeRepeating("MainTargetReachedForGroup", 1.0f, 1f);
  }

  if (config.mode.Equals("Single"))
  {
foreach (GameObject robot in robots)
{
 robot.SetActive(false);
}
//Debug.Log("Setting active robot to "+ config.activeRobotID.ToString());
robots[config.activeRobotID].SetActive(true);

InvokeRepeating("MainTargetReachedForOne", 1.0f, 1f);
  }

  if (config.mode.Equals("Preknown"))
```

```
  {


  }



 }


 void MainTargetReachedForGroup()
 {
  int reload = 0;


  foreach (GameObject robot in robots)
  {
if (robot.GetComponent<Unit>().reached) reload++;
  }


  //if (reload == robots.Length && config.experiment <= config.experiments)
  if (reload == robots.Length)
  {
string array = "";
string array_1 = "";
string array_2 = "";
```

```
string array_3 = "";

for (int i = 0; i < grid.GetComponent<Grid>().grid.GetLength(0); i++)
{
 for (int j = 0; j < grid.GetComponent<Grid>().grid.GetLength(1); j++)
 {
  array += (grid.GetComponent<Grid>().grid[i, j].walkable ? "1" : "0") + " ";
  array_1 += (grid.GetComponent<Grid>().grid_1[i, j].walkable ? "1" : "0") + " ";
  array_2 += (grid.GetComponent<Grid>().grid_2[i, j].walkable ? "1" : "0") + " ";
  array_3 += (grid.GetComponent<Grid>().grid_3[i, j].walkable ? "1" : "0") + " ";
 }
 array += System.Environment.NewLine;

 array_1 += System.Environment.NewLine;
 array_2 += System.Environment.NewLine;
 array_3 += System.Environment.NewLine;
}

string filename = "M" + config.experiment + ".txt";

System.IO.File.WriteAllText(config.path + config.mode +
"\\common" + "\\maps\\" + filename, array);
```

```
System.IO.File.WriteAllText(config.path + config.mode +
"\\R1" + "\\maps\\" + filename, array_1);
System.IO.File.WriteAllText(config.path + config.mode +
"\\R2" + "\\maps\\" + filename, array_2);
System.IO.File.WriteAllText(config.path + config.mode +
"\\R3" + "\\maps\\" + filename, array_3);



//config.experiment++;
//SceneManager.LoadScene(config.scene);
  }
  /*
  if (config.experiment>config.experiments)
  {
SceneManager.LoadScene("Menu");
  }
  */
 }

 void MainTargetReachedForOne()
 {
  foreach (GameObject robot in robots)
```

```
  {
if (robot.activeInHierarchy)
{
 if (robot.GetComponent<Unit>().reached && config.experiment <= config.experiments)
 {
  string array = "";

  for (int i = 0; i < grid.GetComponent<Grid>().grid.GetLength(0); i++)
  {
for (int j = 0; j < grid.GetComponent<Grid>().grid.GetLength(1); j++)
{
 array += (grid.GetComponent<Grid>().grid[i, j].walkable ? "1" : "0") + " ";
}
array += System.Environment.NewLine;
  }

  string filename = "M" + config.experiment + ".txt";

  System.IO.File.WriteAllText(config.path + config.mode
  + "\\R"+ (config.activeRobotID +1).ToString() + "\\maps\\" + filename, array);

  config.experiment++;
```

```
  SceneManager.LoadScene(config.scene);
 }



 if (config.experiment > config.experiments)
 {
  config.experiment = 1;
  config.activeRobotID++;


  if (config.activeRobotID > 2)
  {
SceneManager.LoadScene("Menu");
  }
 }
}
  }


 }


}


using UnityEngine;
```

```
public struct Line {

const float verticalLineGradient = 1e5f;

float gradient;
float y_intercept;
Vector2 pointOnLine_1;
Vector2 pointOnLine_2;

float gradientPerpendicular;

bool approachSide;

public Line(Vector2 pointOnLine, Vector2 pointPerpendicularToLine) {
float dx = pointOnLine.x - pointPerpendicularToLine.x;
float dy = pointOnLine.y - pointPerpendicularToLine.y;

if (dx == 0) {
gradientPerpendicular = verticalLineGradient;
} else {
gradientPerpendicular = dy / dx;
```

```
}

if (gradientPerpendicular == 0) {
gradient = verticalLineGradient;
} else {
gradient = -1 / gradientPerpendicular;
}

y_intercept = pointOnLine.y - gradient * pointOnLine.x;
pointOnLine_1 = pointOnLine;
pointOnLine_2 = pointOnLine + new Vector2 (1, gradient);

approachSide = false;
approachSide = GetSide (pointPerpendicularToLine);
}

bool GetSide(Vector2 p) {
return (p.x - pointOnLine_1.x) *
(pointOnLine_2.y - pointOnLine_1.y) > (p.y - pointOnLine_1.y) *
(pointOnLine_2.x - pointOnLine_1.x);
}
```

```
public bool HasCrossedLine(Vector2 p) {
return GetSide (p) != approachSide;
}

public float DistanceFromPoint(Vector2 p) {
float yInterceptPerpendicular = p.y - gradientPerpendicular * p.x;
float intersectX = (yInterceptPerpendicular - y_intercept) /
(gradient - gradientPerpendicular);
float intersectY = gradient * intersectX + y_intercept;
return Vector2.Distance (p, new Vector2 (intersectX, intersectY));
}

public void DrawWithGizmos(float length) {
Vector3 lineDir = new Vector3 (1, 0, gradient).normalized;
Vector3 lineCentre = new Vector3 (pointOnLine_1.x,
0, pointOnLine_1.y) + Vector3.up;
Gizmos.DrawLine (lineCentre - lineDir *
length / 2f, lineCentre + lineDir * length / 2f);
}

}
```

```
using UnityEngine;
using System.Collections.Generic;

public class Grid : MonoBehaviour {

public bool displayGridGizmos;
public LayerMask unwalkableMask;
public Vector2 gridWorldSize;
float nodeRadius;
int penalty = 100000;
LayerMask walkableMask;

 public Node[,] grid { get; set; }
 public Node[,] grid_1 { get; set; }
 public Node[,] grid_2 { get; set; }
 public Node[,] grid_3 { get; set; }
 public Node[,] grid_nsz { get; set; }

 float nodeDiameter;
int gridSizeX, gridSizeY;
```

```
void Awake()
 {
  nodeRadius = config.nodeRadius;

  nodeDiameter = nodeRadius*2;
gridSizeX = Mathf.RoundToInt(gridWorldSize.x/nodeDiameter);
gridSizeY = Mathf.RoundToInt(gridWorldSize.y/nodeDiameter);

  if (config.mode.Equals("Preknown"))
  {
CreateMappedGrid();
  }
  else
  {
CreateGrid();
  }

 }

 public int MaxSize
 {
get
```

```
  {
return gridSizeX * gridSizeY;
}
}

void CreateGrid()
 {

grid = new Node[gridSizeX,gridSizeY];

  grid_1 = new Node[gridSizeX, gridSizeY];
  grid_2 = new Node[gridSizeX, gridSizeY];
  grid_3 = new Node[gridSizeX, gridSizeY];

  grid_nsz = new Node[gridSizeX, gridSizeY];

  Vector3 worldBottomLeft = transform.position -
  Vector3.right * gridWorldSize.x/2 - Vector3.forward * gridWorldSize.y/2;

for (int x = 0; x < gridSizeX; x ++) {
for (int y = 0; y < gridSizeY; y ++) {
```

```
 int movementPenalty = 0;
 Vector3 worldPoint = worldBottomLeft + Vector3.right *
 (x * nodeDiameter + nodeRadius) + Vector3.forward *
 (y * nodeDiameter + nodeRadius);
 bool walkable = true;

 grid[x, y] = new Node(walkable,worldPoint, x,y, movementPenalty);

 grid_1[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);
 grid_2[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);
 grid_3[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);

 grid_nsz[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);
}
}
}

 void CreateMappedGrid()
 {
  grid = new Node[gridSizeX, gridSizeY];

  grid_1 = new Node[gridSizeX, gridSizeY];
```

```
grid_2 = new Node[gridSizeX, gridSizeY];

grid_3 = new Node[gridSizeX, gridSizeY];


grid_nsz = new Node[gridSizeX, gridSizeY];


Vector3 worldBottomLeft = transform.position -

Vector3.right * gridWorldSize.x / 2 - Vector3.forward * gridWorldSize.y / 2;


for (int x = 0; x < gridSizeX; x++)

{

for (int y = 0; y < gridSizeY; y++)

{

 Vector3 worldPoint = worldBottomLeft +

 Vector3.right * (x * nodeDiameter + nodeRadius) + Vector3.forward

 * (y * nodeDiameter + nodeRadius);

 bool walkable = !(Physics.CheckSphere(worldPoint,

 nodeRadius, unwalkableMask));


 int movementPenalty = 0;


 Ray ray = new Ray(worldPoint + Vector3.up * 50, Vector3.down);
```

```
  if (!walkable)
  {
   movementPenalty += penalty;
  }

  grid[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);

  grid_1[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);
  grid_2[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);
  grid_3[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);

  grid_nsz[x, y] = new Node(walkable, worldPoint, x, y, movementPenalty);
 }
 }
}

public void UpdateGrid(Vector3 obstacle_point, int id, bool walkable)
{
 int x, y;
 int movementPenalty = 0;

 x = Mathf.RoundToInt((obstacle_point.x + gridWorldSize.x / 2) /
```

```
  (nodeDiameter) - 0.5f);
  y = Mathf.RoundToInt((obstacle_point.z + gridWorldSize.y / 2) /
  (nodeDiameter) - 0.5f);
  Vector3 worldBottomLeft = transform.position -
  Vector3.right * gridWorldSize.x / 2 - Vector3.forward * gridWorldSize.y / 2;
  Vector3 worldPoint = worldBottomLeft +
  Vector3.right * (x * nodeDiameter + nodeRadius)
 + Vector3.forward * (y * nodeDiameter + nodeRadius);

  grid[x, y] = new Node(false, worldPoint, x, y, penalty);

  grid_nsz[x, y] = new Node(false, worldPoint, x, y, penalty);

  if (id == 1)
  {
grid_1[x, y] = new Node(false, worldPoint, x, y, penalty);
  }

  if (id == 2)
  {
grid_2[x, y] = new Node(false, worldPoint, x, y, penalty);
  }
```

```
  if (id == 3)
  {
grid_3[x, y] = new Node(false, worldPoint, x, y, penalty);
  }


  try
  {
for (int i = x - 1; i <= x + 1; i++)
 for (int j = y - 1; j <= y + 1; j++)
  grid[i, j] = new Node(walkable, worldPoint, i, j, movementPenalty);
  }
  catch
  {


  }
 }


 public List<Node> GetNeighbours(Node node) {
List<Node> neighbours = new List<Node>();


for (int x = -1; x <= 1; x++) {
```

```
for (int y = -1; y <= 1; y++) {
if (x == 0 && y == 0)
continue;

int checkX = node.gridX + x;
int checkY = node.gridY + y;

if (checkX >= 0 && checkX < gridSizeX &&
checkY >= 0 && checkY < gridSizeY) {
neighbours.Add(grid[checkX,checkY]);
}
}
}

return neighbours;
}

public Node NodeFromWorldPoint(Vector3 worldPosition) {
float percentX = (worldPosition.x + gridWorldSize.x/2) / gridWorldSize.x;
float percentY = (worldPosition.z + gridWorldSize.y/2) / gridWorldSize.y;
percentX = Mathf.Clamp01(percentX);
percentY = Mathf.Clamp01(percentY);
```

```
int x = Mathf.RoundToInt((gridSizeX-1) * percentX);
int y = Mathf.RoundToInt((gridSizeY-1) * percentY);
return grid[x,y];
}

void OnDrawGizmos() {
  Vector3 gridCubeSize = new Vector3((nodeDiameter - .1f),
  0.1f, (nodeDiameter - .1f));
  Gizmos.DrawWireCube(transform.position,
  new Vector3(gridWorldSize.x, 1, gridWorldSize.y));
  if (grid != null && displayGridGizmos)
  {
foreach (Node n in grid)
{
 Gizmos.color = (n.walkable) ? Color.white : Color.red;
 Gizmos.DrawCube(n.worldPosition, gridCubeSize);
}
  }
 }

}
```

```
using System;

public class Edge<T> where T : IEdgeItem<T> {

T[] items;
int currentItemCount;

public Edge(int maxEdgeSize) {
items = new T[maxEdgeSize];
}

public void Add(T item) {
item.EdgeIndex = currentItemCount;
items[currentItemCount] = item;
SortUp(item);
currentItemCount++;
}

public T RemoveFirst() {
T firstItem = items[0];
currentItemCount--;
items[0] = items[currentItemCount];
```

```
items[0].EdgeIndex = 0;

SortDown(items[0]);

return firstItem;

}


public void UpdateItem(T item) {

SortUp(item);

}


public int Count {

get {

return currentItemCount;

}

}


public bool Contains(T item) {

return Equals(items[item.EdgeIndex], item);

}


void SortDown(T item) {

while (true) {

int childIndexLeft = item.EdgeIndex * 2 + 1;
```

```
int childIndexRight = item.EdgeIndex * 2 + 2;

int swapIndex = 0;


if (childIndexLeft < currentItemCount) {

swapIndex = childIndexLeft;


if (childIndexRight < currentItemCount) {

if (items[childIndexLeft].CompareTo(items[childIndexRight]) < 0) {

swapIndex = childIndexRight;

}

}


if (item.CompareTo(items[swapIndex]) < 0) {

Swap (item,items[swapIndex]);

}

else {

return;

}


}

else {

return;
```

```
}


}
}

void SortUp(T item) {
int parentIndex = (item.EdgeIndex-1)/2;

while (true) {
T parentItem = items[parentIndex];
if (item.CompareTo(parentItem) > 0) {
Swap (item,parentItem);
}
else {
break;
}

parentIndex = (item.EdgeIndex-1)/2;
}
}

void Swap(T itemA, T itemB) {
```

```
items[itemA.EdgeIndex] = itemB;
items[itemB.EdgeIndex] = itemA;
int itemAIndex = itemA.EdgeIndex;
itemA.EdgeIndex = itemB.EdgeIndex;
itemB.EdgeIndex = itemAIndex;
}




}

public interface IEdgeItem<T> : IComparable<T> {
int EdgeIndex {
get;
set;
}
}

using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
```

```
public class ButtonManager : MonoBehaviour {



 //GENERAL

 public InputField path;

 public InputField experiments;


 public Dropdown scene;

 public Dropdown mode;


 //UNIT

 public Toggle speedControll;

 public Toggle fullNodeList;


 public InputField speedMax;

 public InputField speedMed;

 public InputField speedLow;


 public InputField turnSpeed;

 public InputField turnDst;


 //TVS
```

```
public InputField viewRadius;
public InputField viewAngle;
public InputField vertAngle;
public InputField meshResolution;

public InputField nodeRadius;
public void Start()
{
 speedControll.isOn = config.speedControll;
 fullNodeList.isOn = config.fullNodeList;

 path.text = config.path.ToString();
 experiments.text = config.experiments.ToString();

 speedMax.text = config.speedMax.ToString();
 speedMed.text = config.speedMed.ToString();
 speedLow.text = config.speedLow.ToString();

 turnSpeed.text = config.turnSpeed.ToString();
 turnDst.text = config.turnDst.ToString();

 viewRadius.text = config.viewRadius.ToString();
```

```
 viewAngle.text = config.viewAngle.ToString();
 vertAngle.text = config.vertAngle.ToString();
 meshResolution.text = config.meshResolution.ToString();

 nodeRadius.text = config.nodeRadius.ToString();
}

public void SceneLoadButton()
{
 config.scene = scene.options[scene.value].text;
 config.mode = mode.options[mode.value].text;

 config.speedControll = speedControll.isOn;
 config.fullNodeList = fullNodeList.isOn;

 config.path = path.text;

 int.TryParse(experiments.text, out config.experiments);

 float.TryParse(speedMax.text, out config.speedMax);
 float.TryParse(speedMed.text, out config.speedMed);
 float.TryParse(speedLow.text, out config.speedLow);
```

```
    float.TryParse(turnSpeed.text, out config.turnSpeed);
    float.TryParse(turnDst.text, out config.turnDst);

    float.TryParse(viewRadius.text, out config.viewRadius);
    float.TryParse(viewAngle.text, out config.viewAngle);
    float.TryParse(vertAngle.text, out config.vertAngle);
    float.TryParse(meshResolution.text, out config.meshResolution);

    float.TryParse(nodeRadius.text, out config.nodeRadius);

    SceneManager.LoadScene(config.scene);
  }

}
```

Appendix B

```
namespace NetworkModeling.Models
{
 class edge
 {
  public int id { get; set; }
  public int s_id { get; set; }
  public int d_id { get; set; }

  public int s_x { get; set; }
  public int s_y { get; set; }
  public int d_x { get; set; }
  public int d_y { get; set; }

  public double w { get; set; }
```

```
  public bool marked { get; set; }
 }
}



namespace NetworkModeling.Models
{
 class node
 {
  public node() { }
  public node(int x, int y)
  {
   this.x = x;
   this.y = y;
  }
  public node(int id, int x, int y)
  {
   this.id = id;
   this.x = x;
   this.y = y;
  }
  public int id { get; set; }
```

```
  public int x { get; set; }
  public int y { get; set; }
  public bool marked { get; set; }
 }
}



namespace NetworkModeling.Models
{
 class Subset
 {
  public int Parent;
  public int Rank;
 }
}



using System;
using System.Collections.Generic;
using System.Linq;

namespace NetworkModeling.Models
```

```
{
 class Graph
 {
  public edge[] _edges;
  public edge[] _kedges;
  List<node> _nodes = new List<node>();
  public IEnumerable<node> nodes { get { return _nodes; } }
  public edge[] edges { get { return _edges; } }
  public edge[] kedges { get { return _kedges; } }
  public void addNode(node argValue)
  {
   _nodes.Add(argValue);
  }

  public void addArc(edge argValue,int id)
  {
   _edges[id] = new edge();
   _edges[id].id = id;
   _edges[id].marked = false;
   _edges[id].s_id = argValue.s_id;
   _edges[id].d_id = argValue.d_id;
   _edges[id].s_x = argValue.s_x;
```

```
 _edges[id].d_x = argValue.d_x;
 _edges[id].s_y = argValue.s_y;
 _edges[id].d_y = argValue.d_y;
 _edges[id].w = argValue.w;



}
public void clearGraph()
{
 _edges = null;
 _nodes = new List<node>();
}



public void genGraph(int vc)
{
 _edges = new edge[Convert.ToInt32((vc * (vc - 1)) / 2)];
 Random rnd = new Random();

 for (int i = 0; i < vc; i++)
 {
  node n = new node(i, rnd.Next(10, 290), rnd.Next(10, 290));
```

```
 addNode(n);
}

int id = 0;
for (int i = 0; i < vc - 1; i++)
{
 node ns = nodes.ElementAt(i);
 for (int j = i + 1; j < vc; j++)
 {
  node dn = nodes.ElementAt(j);
  edge a = new edge();

  a.id = id;
  a.s_id = ns.id;
  a.s_x = ns.x + 5;
  a.s_y = ns.y + 5;

  a.d_id = dn.id;
  a.d_x = dn.x + 5;
  a.d_y = dn.y + 5;

  a.w = Math.Round(Math.Sqrt((a.s_x - a.d_x) *
```

```
   (a.s_x - a.d_x) + (a.s_y - a.d_y) * (a.s_y - a.d_y)));

   addArc(a, id);

   id++;
  }
 }
}
private static int Search(Subset[] subs, int i)
{
 if (subs[i].Parent != i)
  subs[i].Parent = Search(subs, subs[i].Parent);

 return subs[i].Parent;
}

private static void Union(Subset[] subs, int x, int y)
{
 int xR = Search(subs, x);
 int yR = Search(subs, y);

 if (subs[xR].Rank < subs[yR].Rank)
```

```
  subs[xR].Parent = yR;
 else if (subs[xR].Rank > subs[yR].Rank)
  subs[yR].Parent = xR;
 else
 {
  subs[yR].Parent = xR;
  ++subs[xR].Rank;
 }
}

public void Kruskal()
{
 int vc = _nodes.Count;
 edge[] result = new edge[vc];
 int i = 0;
 int e = 0;

 Subset[] subs = new Subset[vc];

 for (int j = 0; j < vc; j++)
 {
  subs[j] = new Subset();
```

```
}
for (int j = 0; j < vc; j++)
{
 result[j] = new edge();
}

Array.Sort(_edges, delegate (edge a, edge b)
{
 return a.w.CompareTo(b.w);
});



for (int v = 0; v < vc; ++v)
{
 subs[v].Parent = v;
 subs[v].Rank = 0;
}

while (e < vc - 1)
{
 edge nextEdge = _edges[i++];
 int x = Search(subs, nextEdge.s_id);
```

```
 int y = Search(subs, nextEdge.d_id);


 if (x != y)
 {
  result[e++] = nextEdge;
  Union(subs, x, y);
 }
}


//_kedges = result;
int ii = 0;
List<edge> _nkedges=new List<edge>();
foreach (edge ed in result)
{
 if (ed.d_id != ed.s_id)
 {
  _nkedges.Add(ed);
  edge b = new edge();


  b.id = ed.id;


  b.d_id = ed.s_id;
```

```
    b.s_id = ed.d_id;

    b.d_x = ed.s_x;
    b.d_y = ed.s_y;

    b.s_y = ed.d_y;
    b.s_x = ed.d_x;

    b.marked = ed.marked;
    b.w = ed.w;
    _nkedges.Add(b);
   }
  }
  _kedges = _nkedges.ToArray();
}

private static int MinKey(int[] key, bool[] set, int vc)
{
  int min = int.MaxValue, minIndex = 0;

  for (int v = 0; v < vc; ++v)
  {
```

```
    if (set[v] == false && key[v] < min)
    {
     min = key[v];
     minIndex = v;
    }
   }

   return minIndex;
  }

 }
}
```

```
using NetworkModeling.Models;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Security.Permissions;
```

```
using System.Threading;
using System.Windows.Forms;


namespace NetworkModeling
{
 public partial class frmMain : Form
 {
  static Graph gr = new Graph();
  edge a = new edge();
  node n = new node();


  double[] nd;
  double[] ndf;
  int nv;


  static double[][] dist;
  int mni;


  SolidBrush sb = new SolidBrush(Color.Red);
  SolidBrush br = new SolidBrush(Color.Blue);
  SolidBrush bw = new SolidBrush(Color.Cyan);
```

```
SolidBrush bt0 = new SolidBrush(Color.Green);
SolidBrush bt1 = new SolidBrush(Color.Blue);
SolidBrush bt2 = new SolidBrush(Color.Cyan);

StringFormat sf = new StringFormat();
Pen p = new Pen(Color.Black);

int v=5;

private Thread netThread;

public frmMain()
{
 InitializeComponent();
}

private void gengrBtn_Click(object sender, EventArgs e)
{
 gr.clearGraph();
 gr.genGraph(Convert.ToInt32(netsizeNUD.Value));
 printData();
 stonePanel.Invalidate();
```

```
}

private void panel1_Paint(object sender, PaintEventArgs e)
{
 Graphics g = stonePanel.CreateGraphics();
 if(gr.edges!=null)
  foreach (edge ac in gr.edges)
  {
   g.DrawLine(p, ac.s_x, ac.s_y, ac.d_x, ac.d_y);
   sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
  }
 foreach (node nd in gr.nodes)
 {
  g.DrawEllipse(p, nd.x, nd.y, 10, 10);
  g.FillEllipse(sb, nd.x, nd.y, 10, 10);
  sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
  g.DrawString(nd.id.ToString(), this.Font, br, nd.x, nd.y, sf);
 }
}

private void printData()
{
```

```
nodesRTB.Clear();

edgesRTB.Clear();

graphRTB.Clear();


if(gr.edges!=null)

 foreach (edge ac in gr.edges)

 {

  string st =

  String.Format("{0}: {1}; {2}; w:{3}",

                 ac.id, ac.s_id, ac.d_id, ac.w);

  edgesRTB.AppendText(st + Environment.NewLine);

 }


if (gr.kedges != null)

 foreach (edge ac in gr.kedges)

 {

  string st = String

  .Format("{0}: {1}; {2}; w:{3}", ac.id, ac.s_id

  , ac.d_id, ac.w);

  graphRTB.AppendText(st + Environment.NewLine);

 }
```

```
 foreach (node nd in gr.nodes)
 {
  string st = String.Format("{0}: {1}; {2}",nd.id,nd.x,nd.y);
  nodesRTB.AppendText(st + Environment.NewLine);
 }
}

private void simpgrBtn_Click(object sender, EventArgs e)
{
 sttwoPanel.Controls.Clear();
 sttwoPanel.Invalidate();
 gr.Kruskal();
 printData();
 sttwoPanel.Invalidate();
}

private void panel2_Paint(object sender, PaintEventArgs e)
{
 Graphics g = sttwoPanel.CreateGraphics();
 if(gr.kedges!=null)
  foreach (edge ac in gr.kedges)
  {
```

```
   g.DrawLine(p, ac.s_x, ac.s_y, ac.d_x, ac.d_y);
   sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
   g.DrawString(ac.w.ToString(),
       this.Font, bw,
       (ac.s_x + ac.d_x) / 2, (ac.s_y + ac.d_y) / 2, sf);
  }

 foreach (node nd in gr.nodes)
 {
  g.DrawEllipse(p, nd.x, nd.y, 10, 10);
  g.FillEllipse(sb, nd.x, nd.y, 10, 10);
  sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
  g.DrawString(nd.id.ToString(), this.Font, br, nd.x, nd.y, sf);
 }
}

private void buildntwBtn_Click(object sender, EventArgs e)
{
 ////////////////////////////////////////////
 dist = FloydWarshal(gr);
 mni = Weights(dist);
 stthreePanel.Controls.Clear();
```

```
 stthreePanel.Invalidate();
 stthreePanel.Invalidate();
 ////////////////////////////////////////////
}

private static double[][] FloydWarshal(Graph gr)
{
 int v = gr.nodes.Count();
 double[][] dist = new double[v][];
 for (int i = 0; i < v; i++)
 {
  dist[i] = new double[v];
  for (int j = 0; j < v; j++)
  {
   dist[i][j] = 16000;
  }
 }

 for (int j = 0; j < v; j++)
 {
  dist[j][j] = 0;
 }
```

```
 foreach (edge ed in gr.kedges)
 {
  dist[ed.d_id][ed.s_id] = ed.w;
  dist[ed.s_id][ed.d_id] = ed.w;
 }
 for (int k = 0; k < v; k++)
 {
  for (int i = 0; i < v; i++)
  {
   for (int j = 0; j < v; j++)
   {
    if (dist[i][j] > dist[i][k] + dist[k][j])
    {
     dist[i][j] = dist[i][k] + dist[k][j];
    }
   }
  }
 }
 return dist;
}
private void button1_Click(object sender, EventArgs e)
```

```
{
 v = Convert.ToInt32(netsizeNUD.Value);

 ////////////////////////////////////////////
 gr.clearGraph();
 gr.genGraph(v);
 printData();
 stonePanel.Invalidate();
 ////////////////////////////////////////////

 ////////////////////////////////////////////
 sttwoPanel.Controls.Clear();
 sttwoPanel.Invalidate();
 gr.Kruskal();
 printData();
 sttwoPanel.Invalidate();
 ////////////////////////////////////////////

 ////////////////////////////////////////////
 dist = FloydWarshal(gr);
 mni = Weights(dist);
 ////////////////////////////////////////////
```

```
 stthreePanel.Controls.Clear();
 stthreePanel.Invalidate();

 stthreePanel.Invalidate();
 ////////////////////////////////////////////
}

private static int Weights(double[][] dist)
{
 int v = dist.Count();
 double[] nodes = new double[v];
 for (int i = 0; i < v; i++)
 {
  nodes[i] = dist[i].Average();
 }
 return nodes.ToList().IndexOf(nodes.Min());
}

private static double Weights2(double[][] dist)
{
 int v = dist.Count();
 double[] nodes = new double[v];
```

```
 for (int i = 0; i < v; i++)
 {
  nodes[i] = dist[i].Average();
 }
 return nodes.Min();
}
bool search(List<int> nu, int nid)
{
 foreach (int nui in nu)
 {
  if (nui == nid)
  {
   return true;
  }
 }
 return false;
}


int childrenCount(Graph gr, int nid)
{
 int child = 0;
 foreach (edge ed in gr.kedges)
```

```
 {
  if (ed.s_id == nid) child++;
 }
 return child;
}

private void panel3_Paint(object sender, PaintEventArgs e)
{
 if (gr.edges != null)
 {
  Graphics g = stthreePanel.CreateGraphics();
  List<int> nu = new List<int>();

  foreach (edge ac in gr.kedges)
  {
   g.DrawLine(p, ac.s_x, ac.s_y, ac.d_x , ac.d_y);
   sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
   g.DrawString(ac.w.ToString(),
   this.Font, bw, (ac.s_x + ac.d_x) / 2, (ac.s_y + ac.d_y) / 2, sf);
  }

  foreach (node nd in gr.nodes)
```

```
{
 if (childrenCount(gr, nd.id)==1)
 {
  g.DrawEllipse(p, nd.x, nd.y, 10, 10);
  g.FillEllipse(bt2, nd.x, nd.y, 10, 10);
  sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
  g.DrawString(nd.id.ToString(), this.Font, br, nd.x, nd.y, sf);
 }
 if (childrenCount(gr, nd.id) > 1)
 {
  g.DrawEllipse(p, nd.x, nd.y, 10, 10);
  g.FillEllipse(bt1, nd.x, nd.y, 10, 10);
  sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
  g.DrawString(nd.id.ToString(), this.Font, br, nd.x, nd.y, sf);
 }
 if (nd.id == mni)
 {
  g.DrawEllipse(p, nd.x, nd.y, 10, 10);
  g.FillEllipse(bt0, nd.x, nd.y, 10, 10);
  sf.FormatFlags = StringFormatFlags.DirectionRightToLeft;
  g.DrawString(nd.id.ToString(), this.Font, br, nd.x, nd.y, sf);
 }
```

```
  }
 }
}

void calcNetwork()
{
 for (int i = 1; i < nv; i++)
 {
  double fmni = 0;
  double fdist = 0;
  int cv = Convert.ToInt16(crosvalidNUD.Value);
  for (int j = 1; j < nv; j++)
  {
   gr.clearGraph();
   gr.genGraph(i);

   gr.Kruskal();

   dist = FloydWarshal(gr);

   mni = Weights(dist);
   fmni += Weights2(dist);
```

```
  for (int ii = 0; ii < dist.Length; ii++)
  {
   for (int jj = 0; jj < dist.Length; jj++)
   {
    fdist += dist[ii][jj];
   }
  }
 }
 fmni /= cv;
 nd[i] = fmni;

 if (chart1.IsHandleCreated)
 {
  this.Invoke((MethodInvoker)delegate { UpdateChart(); });
 }
 Thread.Sleep(100);
}

ndf = Filterd(nd);
if (chart1.IsHandleCreated)
{
```

```
  this.Invoke((MethodInvoker)delegate { UpdateChart2(); });
 }
 Thread.Sleep(100);


 KillTheThread();
}
private void UpdateChart()
{
 chart1.Series[0].Points.Clear();
 for (int i = 0; i < nd.Length - 1; ++i)
 {
  chart1.Series[0].Points.AddXY(i, nd[i]);
 }
}
private void UpdateChart2()
{
 chart1.Series[1].Points.Clear();
 for (int j = 0; j < ndf.Length - 1; ++j)
 {
  chart1.Series[1].Points.AddXY(j, ndf[j]);
 }
}
```

```
[SecurityPermissionAttribute(SecurityAction.Demand,
ControlThread = true)]
private void KillTheThread()
{
 netThread.Abort();
}



private void button2_Click(object sender, EventArgs e)
{
 chart1.Series[0].Points.Clear();
 chart1.Series[1].Points.Clear();



 nv = Convert.ToInt16(maxnodesNUD.Value);
 nd = new double[nv];
 ndf = new double[nv];


 netThread = new Thread(new ThreadStart(this.calcNetwork));
 netThread.IsBackground = true;
 netThread.Start();
```

```
}




private static void fullDemo()
{


}


private static Bitmap DrawControllToBitmap(Control control)
{
 Bitmap bitmap = new Bitmap(control.Width, control.Height);
 Graphics g = Graphics.FromImage(bitmap);
 Rectangle rect = control.RectangleToScreen(control.ClientRectangle);
 g.CopyFromScreen(rect.Location, Point.Empty, control.Size);
 return bitmap;
}


private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
 DateTime localDate = DateTime.Now;
 Bitmap bitmap = DrawControllToBitmap(stonePanel);
```

```
 bitmap.Save("images//ornet_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(sttwoPanel);
 bitmap.Save("images//basetree_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(stthreePanel);
 bitmap.Save("images//main_" + v.ToString() + ".bmp", ImageFormat.Bmp);
}

private void button3_Click(object sender, EventArgs e)
{
 DateTime localDate = DateTime.Now;
 Bitmap bitmap = DrawControllToBitmap(stonePanel);
 bitmap.Save("images//ornet_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(sttwoPanel);
 bitmap.Save("images//basetree_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(stthreePanel);
 bitmap.Save("images//main_" + v.ToString() + ".bmp", ImageFormat.Bmp);
}
```

```
public float getCPUCounter()
{

 PerformanceCounter cpuCounter = new PerformanceCounter();
 cpuCounter.CategoryName = "Processor";
 cpuCounter.CounterName = "% Processor Time";
 cpuCounter.InstanceName = "_Total";

 // will always start at 0
 float firstValue = cpuCounter.NextValue();
 System.Threading.Thread.Sleep(500);
 // now matches task manager reading
 float secondValue = cpuCounter.NextValue();
 return secondValue;

}

int tiks = 0;
private void timer1_Tick(object sender, EventArgs e)
{
 float cpuPercent = getCPUCounter();
 chart1.Series[0].Points.AddXY(tiks,cpuPercent);
```

```
 tiks++;
}

private static double[] Filterd(double[] dist)
{
 int v = dist.Count();
 int w = 7;
 int w2 = 3;
 double[] nodes = dist;

 for (int j = 0; j <w2; j++)
 {
  nodes[j] = 0;
 }

 for (int j = v-w2; j < v; j++)
 {
  nodes[j] = nodes[v-1];
 }

 for (int i = w2; i < v- w2; i++)
 {
```

```
  double sum = 0;
  for (int j = i-w2; j <= i+ w2; j++)
  {
   sum += dist[j];
  }
  nodes[i] = sum / w;
 }

 return nodes;
}

private void graphImagesToolStripMenuItem_Click(object sender, EventArgs e)
{
 bool exists = System.IO.Directory.Exists("images//");
 if (!exists)
  System.IO.Directory.CreateDirectory("images//");

 DateTime localDate = DateTime.Now;
 Bitmap bitmap = DrawControllToBitmap(stonePanel);
 bitmap.Save("images//ornet_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(sttwoPanel);
```

```
 bitmap.Save("images//basetree_" + v.ToString() + ".bmp", ImageFormat.Bmp);


 bitmap = DrawControllToBitmap(stthreePanel);
 bitmap.Save("images//main_" + v.ToString() + ".bmp", ImageFormat.Bmp);
}


private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
 DialogResult result =
 MessageBox.Show("Do you want to exit application?", "Warning",
 MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning);
 if (result == DialogResult.Yes)
 {
  Application.Exit();
 }
 else if (result == DialogResult.No)
 {
  //code for No
 }
 else if (result == DialogResult.Cancel)
 {
  //code for Cancel
```

```
 }
}

private void saveAllToolStripMenuItem_Click(object sender, EventArgs e)
{
 bool exists = System.IO.Directory.Exists("images//");
 if (!exists)
  System.IO.Directory.CreateDirectory("images//");

 DateTime localDate = DateTime.Now;
 Bitmap bitmap = DrawControllToBitmap(stonePanel);
 bitmap.Save("images//ornet_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(sttwoPanel);
 bitmap.Save("images//basetree_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 bitmap = DrawControllToBitmap(stthreePanel);
 bitmap.Save("images//main_" + v.ToString() + ".bmp", ImageFormat.Bmp);

 exists = System.IO.Directory.Exists("text//");
 if (!exists)
  System.IO.Directory.CreateDirectory("text//");
```

```
 nodesRTB.SaveFile("text//" + "nodes_" + v.ToString() + ".txt",
 RichTextBoxStreamType.PlainText);
 edgesRTB.SaveFile("text//" + "edges_" + v.ToString() + ".txt",
 RichTextBoxStreamType.PlainText);
 graphRTB.SaveFile("text//" + "graph_" + v.ToString() + ".txt",
 RichTextBoxStreamType.PlainText);
}

private void graphInTextToolStripMenuItem_Click(object sender, EventArgs e)
{
 bool exists = System.IO.Directory.Exists("text//");
 if (!exists)
  System.IO.Directory.CreateDirectory("text//");

 nodesRTB.SaveFile("text//"+"nodes_" + v.ToString() + ".txt",
 RichTextBoxStreamType.PlainText);
 edgesRTB.SaveFile("text//" + "edges_" + v.ToString() + ".txt",
 RichTextBoxStreamType.PlainText);
 graphRTB.SaveFile("text//" + "graph_" + v.ToString() + ".txt",
 RichTextBoxStreamType.PlainText);
}
```

```
private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
{
 About moreForm = new About();
 //moreForm.Modal = true;
 moreForm.ShowDialog(this);
}

private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
 Graph gr = new Graph();
 edge a = new edge();
 node n = new node();

 nd = null;
 ndf = null;
 dist = null;

 StringFormat sf = new StringFormat();

 v = 5;
```

```
    stonePanel.Refresh();
    sttwoPanel.Refresh();
    stthreePanel.Refresh();

    nodesRTB.Clear();
    edgesRTB.Clear();
    graphRTB.Clear();
   }
  }
}
```