Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Institute of Computer Science

Bachelor's diploma thesis

in the field of study Computer Science
and specialisation Computer Systems and Networks

A Swarm of Robots Performing Delivery Tasks Based on Embodied Agent Approach

Wiktor Bara
student record book number 293046

thesis supervisor
M.Sc. Maksym Figat

WARSAW 2024

A Swarm of Robots Performing Delivery Tasks Based on Embodied Agent Approach

Abstract. Swarm robotics is developed rapidly and conducted in this field research showed that multiple robot approach performs better, than single robot one, when faced with specific types of tasks. The superiority is mostly visible in fields where swarm of robots can take advantage from its assumptions: scalability, flexibility and robustness. The tasks such as exploration, search and rescue or object clustering can obviously enhance performance by using groups of robots. In this engineering thesis under consideration are taken algorithms used in collaborative transportation, and examined is impact of different aggregation algorithms on general outcome and effectiveness of the main transport task.

Keywords: Swarm Robotics, Aggregation, Collaborative Manipulation, Embodied Agent Approach

Rój robotów realizujący zadania dostawcze bazujący na podejściu agentowym

Streszczenie. Robotyka roju rozwija się szybko, przeprowadzone w tym polu badania pokazały, że podejście wielorobotyczne sprawdza się lepiej niż podejście z jednym robotem w przypadku określonych rodzajów zadań. Wyższość podejścia wielorobotycznego jest widoczna przede wszystkim w dziedzinach, w których robotyka roju może czerpać korzyści z jej założeń: skalowalności, elastyczności i solidności (ang. robustness). Zadania takie jak eksploracja, poszukiwanie i ratowanie (ang. search and rescue) lub grupowanie obiektów mogą w sposób oczywisty zwiększyć wydajność dzięki użyciu grup robotów. W tej pracy inżynierskiej uwzględniono algorytmy stosowane w transporcie zbiorowym oraz zbadano wpływ różnych rodzajów algorytmów agregacji na ogólny wynik i efektywność głównego zadania transportowego.

Słowa kluczowe: Robotyka Roju, Agregacja, Manipulacja Zespołowa, Agent Upostaciowiony

załącznik nr 10 do zarządzenia
nr 46 /2016 Rektora PW

Politechnika Warszawska
Warsaw University of Technology

…….....
miejscowość i data
place and date

……..
imię i nazwisko studenta
name and surname of the student

……..
numer albumu
student record book number

…….…….
kierunek studiów
field of study

OŚWIADCZENIE

DECLARATION

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.
Under the penalty of perjury, I hereby certify that I wrote my diploma thesis on my own, under the guidance of the thesis supervisor.

Jednocześnie oświadczam, że:
I also declare that:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- *this diploma thesis does not constitute infringement of copyright following the act of 4 February 1994 on copyright and related rights (Journal of Acts of 2006 no. 90, item 631 with further amendments) or personal rights protected under the civil law,*
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- *the diploma thesis does not contain data or information acquired in an illegal way,*
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- *the diploma thesis has never been the basis of any other official proceedings leading to the award of diplomas or professional degrees,*
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- *all information included in the diploma thesis, derived from printed and electronic sources, has been documented with relevant references in the literature section,*
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.
- *I am aware of the regulations at Warsaw University of Technology on management of copyright and related rights, industrial property rights and commercialisation.*

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.
I certify that the content of the printed version of the diploma thesis, the content of the electronic version of the diploma thesis (on a CD) and the content of the diploma thesis in the Archive of Diploma Theses (APD module) of the USOS system are identical.

.....
czytelny podpis studenta
legible signature of the student

Contents

1. Introduction . . . 9
1.1. Work Motivation . . . 9
1.2. Objective of the Work . . . 9
1.3. Scope of Work . . . 10
1.4. Work Layout . . . 11
2. Review of the Literature . . . 12
2.1. Multiple Robot Systems . . . 12
2.2. Swarm Robotics . . . 13
2.2.1. Robustness . . . 14
2.2.2. Flexibility . . . 14
2.2.3. Scalability . . . 14
2.2.4. Heterogeneity vs Homogeneity . . . 15
2.2.5. Basic Tasks . . . 15
2.2.6. Implementations . . . 16
3. Project Requirements . . . 19
3.1. Functional Requirements . . . 19
3.2. Non-Functional Requirements . . . 19
4. Embodied Agent Approach . . . 21
4.1. General Inner Structure of an Embodied Agent . . . 21
4.2. Activity of the System . . . 23
5. Description of Swarm Robotics Basic Subtasks . . . 25
5.1. Basic Tasks . . . 25
5.1.1. Random Walk . . . 25
5.1.2. Following . . . 27
5.2. Subtasks . . . 28
5.2.1. Resource Search . . . 28
5.2.2. Wait for Another Robot . . . 28
5.2.3. Group Aggregation . . . 29
5.2.4. Group Search . . . 30
5.2.5. Transporting the Resource . . . 31
5.3. Main Task . . . 32
5.3.1. Search - FormTeam . . . 32
5.3.2. FormTeam - Search . . . 33
6. System Specification . . . 34
6.1. Robot Structure . . . 34
6.2. Structure of Single Embodied Agent . . . 36
6.3. Single Embodied Agent Activities . . . 37

7. Tools . . . 38
7.1. Robot Operating System . . . 38
7.2. Stage ROS . . . 39
7.3. Gazebo . . . 41
7.4. Python: rospy . . . 42
8. System Implementation . . . 43
8.1. Resource Aggregation . . . 44
8.2. Group Aggregation . . . 46
8.3. Searching in Group . . . 48
8.4. Transporting the Resource . . . 51
9. Experiments . . . 54
9.1. Resource Aggregation . . . 54
9.2. Group Aggregation . . . 58
9.3. Group Search . . . 61
9.4. Algorithms Comparison . . . 63
9.5. Transportation . . . 65
10. Summary . . . 67
10.1. Achieved Goals . . . 67
10.2. Conclusions . . . 67
10.3. Future Works . . . 68
References . . . 69
List of Figures . . . 72

1. Introduction

1.1. Work Motivation

Mankind's tendency to making the work easier by attempt to make everything automated is one of the major motors of progress. Due to that tendency the various fields of technology are developed. One of those fields is robotics, and in this work that branch of technology is taken under consideration.

The thesis is the product of understanding the part of robotics branch, the swarm robotics. That approach assumes system consisting of relatively high number of simple, if not trivial, units, that operate basing on local interactions between them and the environment without centralized control. The main assumptions of swarm robotics are: the *scalability*, that allows system to grow by increasing the number of robots without loss of the efficiency, the *robustness*, that make the system fault tolerant, and the *flexibility* that makes the system to be universal and able to perform various types of tasks. That system should be also able to perform complex tasks that are impossible, or cost a lot of resources, in case of being performed by human or single highly complicated robot unit.

In this work the problem of collaborative gathering and transportation of resources is considered. The problem assumes that every resource has to be transported by two robots at the same time. Having given a specific number of resources to be gathered and a specific number of robots gathering the resources, we want to gather the resources as fast as possible.

There were proposed two approaches to solve this problem. The first one assumes the individual search for resources by the robots and then the merge into pairs near the resource. The second approach, on the other hand, prioritize the pairs creation. The robots search for each other and after the finding and merge they search for resources together in previously defined formation. In both cases, after the merging and resource finding, the robots transport the resource, to previously defined location.

We want to examine the influence of the order of executing, given in those two approaches, subtasks on the effectiveness of the whole algorithm of searching and transporting the resources.

1.2. Objective of the Work

The work describes previously mentioned issue of collaborative transportation of resources with usage of swarm robotics. The objective of work is to design the algorithms and implement them within the robotic system performing the tasks in the simulated environment. Under consideration are taken, described in the **Chapter 1.1**, two approaches to

solve the task. The comparison between those two approaches should show the differences between their efficiency in case of given task. The comparison would be based on the analysed data, previously obtained with usage of implemented system and the conducted simulations in robotic environment.

The comparison of those two approaches would provide answers about the quality and efficiency of both ways to solve the problem of collaborative transportation. The obtained results should also show the connection of presented system to the main assumption of the swarm robotics: the scalability, the robustness and the flexibility.

1.3. Scope of Work

The work includes several aspects that are designed, created and implemented. To achieve the objectives and goals set in previous subsection, several steps has to be taken.

The main and most crutial part of the project is the actual system development and implementation. The subtasks, mentioned in description of both approaches to perform the task, have to be developed into more specified algorithms. Those algorithms are implemented with designed robotic system with usage of embodied agent approach. The approach is described in **Chapter 4**, where on the robot is run controller expressed in the form of agent, that handles the subtasks of the system.

The task of the collaborative transportation can be divided into smaller parts that can be simulated separately. Those different scenarios, that are taken under consideration as the self executable, are: *Resource Aggregation, Group Aggregation, Group Search* and *Transporting the Resource*. They correspond to given subtasks of the main task where: *Resource Aggregation* corresponds to the first approach - finding the resource as the individual and then merging into a pair, *Group Aggregation* corresponds to first part of the second approach - finding of the partner and merging into a pair, *Group Search* corresponds to the second part of the second approach - finding the resource as the pair in the formation, *Transporting the Resource* corresponds to final part of both approaches - collaborative transportation of resource by two robots.

The system and the simulation scenarios presented above are designed and used to gather the data, that allow us to present the dependencies of the order of the subtasks execution to the efficiency of the approach in respect to another one. That data is analysed, and the results of that analysis will provide us possibility to form the thesis about the superiority of one approach over another, in case of collective resource transportation.

1.4. Work Layout

The work consists of 10 chapters. In Chapter 2 the concept of swarm robotics is introduced and generally described. Chapter 3 presents the requirements for the system, both functional and non-functional. Chapter 4 describes the term of embodied agent approach. In Chapter 5 the algorithms used in the system are described. Chapter 6 takes under consideration the structure of the designed system and elements used in it. The tools used to develop the system are listed and described in Chapter 7. The implementation of the system is shown in Chapter 8 while Chapter 9 includes data discussion and comparison between different approaches. Chapter 10 concludes the information presented throughout the work and indicates the future work.

2. Review of the Literature

Robotics as a whole branch of technology is a wide range issue that appears in many fields, from production, through medicine, to military implementations and is used to provide better outcomes due to superior effectiveness or to prevent any harm to humans. The research conducted on robotics caused enormous growth of different subbranches that develop rapidly giving us varied functionalities and tools. This chapter describes an approach that takes under consideration systems that consist of multiple robots. It presents features of multiple robot systems (MRS), their advantages and motivations. Additionally it describes the reasons why MRS approach was developed and the swarm robotics emerged.

2.1. Multiple Robot Systems

Multiple robots systems is one of sub-branches of robotics and its motivations are advantages over single robot approach. The tasks where the distribution over some area is needed or the tasks are complex enough to be divided into many subtasks, that can be executed parallel, are perfect examples of superiority of multi robot approach. In most of cases where there is an option between single powerful robot and multiple resource-bounded ones, the better way is the choice of the second one, the simplicity and cost is an important factor. The perfect example of tasks where multiple robot systems work better than the single robot ones is the area coverage tasks. The tasks where robots need to find some resources and gather them, or so called *search and rescue* tasks where there is missing a person and the robotic system has to find it as fast as possible. In that cases the superiority of the multiple robot systems is obvious due to their area coverage ability. Another example of usage of multiple robot systems is assembly line. The specialised (heterogeneous - robots that differ in structure and capabilities) robots assemble something like a car or household appliances. That line consists of different robots that cooperate and at the end their collective work results in finished product. The single robot system that has the same utility would be incomparable slower and as the result of that significantly more expensive. The error probability is also one of the main issues that we can face in robotics. The multiple agent systems limit this probability to minimum by ability to replace dysfunctional robot by another one.

Dudek in [1] distinguishes four types of tasks, taking under consideration differences between single robot and multiple robots approach: tasks that require multiple robots, tasks that are traditionally multi-robot, tasks that may benefit from the use of multiple robots and tasks which are inherently single robot. Dudek gives examples of these types and explains why and in what situations the collectives should be applied instead of single, more complex robot. Dudek gives an example of task that require multiple robots. The task consists of turning two keys at the same time, where keys are a large distance

apart. The usage of multiple robot system is the best option because of the complexity and the costs of the tasks in case of single robot system. The example of the tasks that are traditionally multi-robot are the tasks that do not need the communication between robots, but the information is obtained via central control module. Dudek as the direct example gives truck delivery or forestery as the task that benefits from paralell work of robots. The example of task that may benefit from the use of multiple robots is presented by Dudek as the tasks where the collective is searchning for the object on given area. The single robot can achieve the goal of that task, but the system that consists of several robots will find the object significantly faster. The example of the task that is inherently single robot is not given by Dudek, but it is easy to find. That example can be the surgical robot, the robot has to carry out a surgical operation, that is efficiently performed if the robot makes it on its own.

In [1] is also given a taxonomy for multi-robot systems using several dimensions along which robotic collectives can be classified: collective size, communication (range, topology, bandwidth), collective reconfigurability, processing ability, collective, composition. That taxonomy can be used to generally classify group of robots in some of given by Dudek dimentions.

However, the multiple robot system is only the part of the robotics. There was created more specialized type of that approach. The swarm robotics, that is multiple robot system with several assumptions.

2.2. Swarm Robotics

Swarm robotics is a special approach to multiple robot systems. That approach concentrates on coordination of multiple, relatively simple robots as a whole. The difference between the swarm robotics and multiple robot system are the assumptions. The swarm robotics is the more specialised system that is decentralised, while multiple robot systems usually have a coordinator. Moreover, the multiple robot systems often consist of specialised (heterogeneous) robots, but in swarm robotics there is tendency to simplify the system structure and the systems tend to consist of robots that are exactly the same (homogeneous). Another aspect is the communication between robots. Swarm robotics assumes that there is no communication or that communication is limited to local one, usually by stigmergy - indirect communication with usage of environment, but sometimes used is explicit communication - via dedicated channels. To sum up, the swarm robotics is the multiple robot system that is decentralised, in most of cases homogeneous and the communication is limited to local.

The inspirations of swarm robotics comes from observations of swarm insects such as ants, termites, etc. The research in this field showed us that insects, that are very simple

organisms, can without central control achieve goals that are far beyond abilities of single unit. However, swarm robotics is very specific approach, the misinterpretation and taking term "Swarm Robotics" into different meaning is common problem, to avoid it Şahin in [2], defined this term as:

"Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between agents and the environment".

Despite the fact that Şahin highlights that, the definition is not complete and does not perfectly explain what swarm robotics is, it captures all nature of that branch of robotics, and allow to understand the basis of it. To provide more information and extend presented above definition Şahin in [2] presented main motivations of the swarm robotics: robustness, flexibility and scalability.

2.2.1. Robustness

Robustness - the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [3].

That property is relatively easily achieved in any of multiple robots systems provided by number of robots. The redundancy that emerges from the assumptions of such a system can be performed by simple replacement of damaged or dysfunctional robot. Another aspect is the simplicity of the robot itself that makes it less prone to failure in comparison to single, more advanced unit.

2.2.2. Flexibility

Flexibility - the property of the swarm robotic system which allows to adapt system rapidly to new operating conditions.

It can be obtained by the system when the group of units, despite their individual limits, can perform complex tasks by dividing them into less complex ones achievable by smaller groups in swarm or even the individuals. In simple words the flexibility is the universality level of the system in performing various types of tasks.

2.2.3. Scalability

Scalability - the measure of the robotic system's ability to adapt to changes in number of robots by increasing or decreasing the performance of the entities in the system.

The scalability is the feature of the robotic system that allows system to adapt to changes in number of robots, either by increasing that number or decreasing. The system has to adjust to current number of robots and, based on that, perform tasks in the way

that gives the best efficiency for the whole system. The main field where scalability creates issues is communication between the robots and the type of coordination approach. Those two problems should be solved to obtain the algorithm and abilities that can be used in the swarm of different number of robots. The assumption of scalability is the theoretical possibility to increase number of robots up to infinity. To allow such a case the coordination of whole swarm has to be decentralized and the communication can be both explicit (communication directly e.g. via predefined channels) or implicit (communication indirectly e.g. via environment), but has to be somehow limited and assumed to be the local one.

2.2.4. Heterogeneity vs Homogeneity

The aspect that has to be taken under consideration in the swarm robotics is the similarity of the robots to each other. The system, to achieve those three previously mentioned motivations (robustness, flexibility, scalability), has to be as uniform as possible. The robots in that group can be homogeneous (the capabilities and the structure of individuals are the same) or heterogeneous (otherwise). The swarm should of course tend to be more homogeneous to simplify implementation, decrease costs etc., however, some deviations from the norm can be acceptable. Instead of creating one homogeneous swarm we can develop few different homogenous groups that will make the swarm heterogenous but allows to fulfill basic assumptions. The example and inspirations of such an approach are some swarms of ants that possess at least two casts of units (neglecting the queen and males): workers and warriors, that differ in many fields from body to tasks. In some hives there are also very specific and strange types of ants, we can call them even the abominations. The physical appearance significantly varies from other ones, overgrown head makes them looks strange and their only purpose is to die under the attack on the nest previously putting the head into entrance and blocking the passage. Heterogeneity may be better approach in swarms where the group should be able to execute two or more task that need hardware that differs in significant way. Additionally, it can be useful to prevent building more complex and more expensive units. Therefore, a better solution is division swarm into homogeneous groups.

2.2.5. Basic Tasks

As previously mentioned, collectives are used in specific types of tasks that need parallel execution and synchronization or can benefit from using multiple agents. Bayindir in [4] created a review about basic tasks examples that can be used not only as the goal task but also as one of the subtasks used in more complex algorithms. Bayindir presents those as basic swarm algorithms:

- **aggregation** - gathering in groups,
- **flocking** - moving in formation,
- **foraging** - gathering resources to the predefined location,
- **object clustering and sorting** - gathering resources to not predefined location, and sorting them by object type,
- **navigation** - navigating one robot from the swarm to the predefined location,
- **path formation** - creating the track to the interest point,
- **deployment** - spreading the swarm over the area,
- **collaborative manipulation** - manipulating an object in synchronized way.

Bayindir describes those algorithms, shows the methods used to implementation, and analyses metrics and models to measure the efficiency of task with given parameters and approach. He also considers more precise tasks as: odor source localization, object assembly, self-assembly and morphogenesis, coordinated motion, group size estimation, distributed rendezvous, collective decision making or human-swarm interaction. Those tasks are less basic and less often used as a subtasks but are important issue in swarm robotics fields in specific situations. The aspect that is also taken under consideration by Bayindir is task allocation problem. The problem that emerges in most of situations where is a need for optimalization and resource deficit. In my work that considers transportation and delivery tasks with use of swarm robotics I mainly focus on aggregation, flocking, deployment and collaborative manipulation, the tasks that can be used as subtasks to more complex one. I will describe them in detail in **Section 5**.

2.2.6. Implementations

The real life implementation of swarm robotic were widely proposed in many theoretical papers, are seen also in the pop culture, especially in science fiction branch. The examples proposed by space agencies like NASA or ESA that want to use swarm robotic to exploration of the universe, or in the closest future the Mars are an interesting idea.

Another interesting topic is considered in [5], usage of swarm robots as an micro or nanotechnology gives a lot of possibilities and enormous number of implementations, mainly in medicine because of the size comparable to the ingredients of our blood or the causes of the diseases: bacteria and viruses. In the **Fig 2.1** and the **Fig 2.2** we can see an example of project for the artificial respirocyte (red cell) designed by Robert A.Freitas Jr.

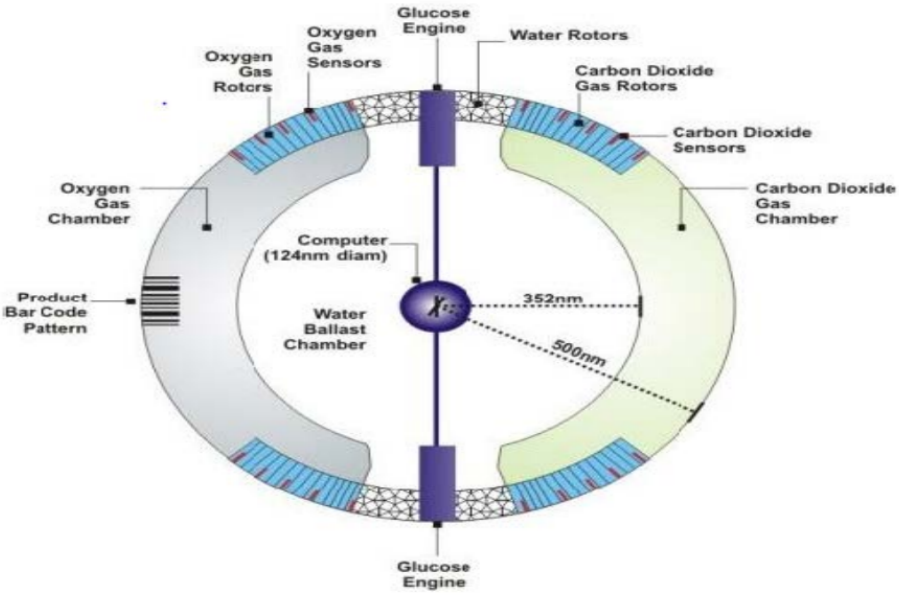


Figure 2.1. Internal cutaway view of respirocyte - equatorial view [5]

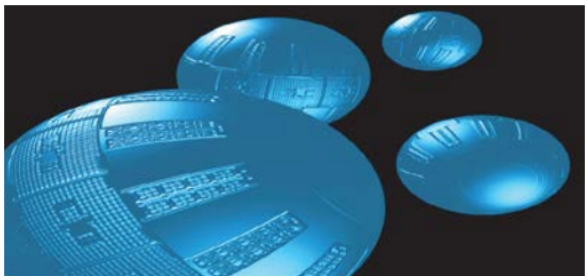
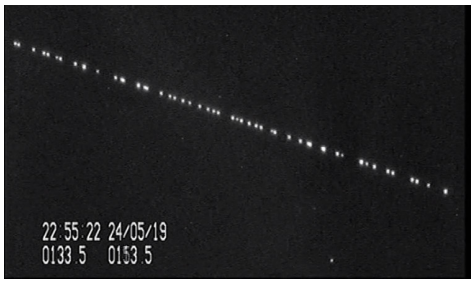


Figure 2.2. An artificial red cell - the respirocyte designed by Robert A.Freitas Jr [5]

The example of existing already swarm of robots is Starlink initiative [6] that is the mesh of satellites providing the global internet all over the world. The Starlink can be considered not to be the swarm because of the global communications channels, that exists because of the purpose of that initiative. Despite that fact, however, the Starlink fullfills the main motivations of swarm robotics, it is scalable due to enourmous number of entities in the mesh that is still increased. Due to the same fact it is robust, because the malfunction of one satelite will not cause significant loss in the efficiency of the whole system, and due to lower orbit than the geostationary satelites that provide satellite internet, there is achieved the flexibility. The starlink has lower latency than satellite internet and is available everywhere, even on the Arctic Circle.

In the **Fig 2.3** is visible the single satelite from the Starlink initiative and the photo of the launch of a group of satelites on their way to orbit.

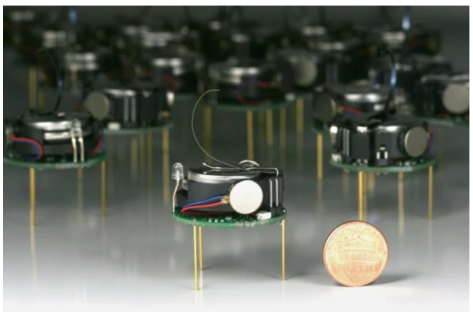
(a) Starlink's Satelite



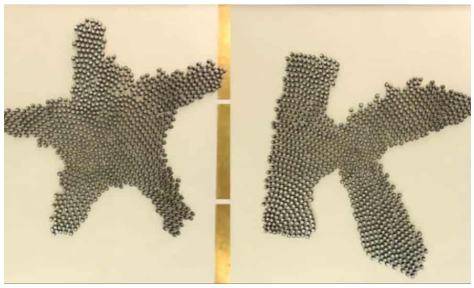
(b) Starlink launch

Figure 2.3. Starlink

Another example is the project with 1000-kilobots presented by Zastrow in [7]. The project consists of collective of 1024 small robots called kilobots. The robots basing only on local communication create 2 dimentional patterns. Two of that patterns and single robot with comparison to the coin are visible in the **Fig 2.4**



(a) Single kilobot with comparison to coin



(b) Kilobots that formed two paterns, star and letter 'K'

Figure 2.4. 1000-kilobots

3. Project Requirements

In this section project requirements are taken under consideration. The vast majority of the project bases on the simulator, simulation and the issues directly connected to it. The algorithms prepared and conducted using that simulator have to have direct usage and the data collected should be clear and useful. The requirements are divided into two groups: functional requirements and non-functional requirements. The first group defines what functionality the project is supposed to provide and the second determines how does the system should do it.

3.1. Functional Requirements

- **Algorithm Simulation** - The system should provide possibility to run simulation of algorithm of searching the resource and collective delivery by robots. The algorithm should be divided into subalgorithms that will be simulated separately: box pushing, group aggregation, resource aggregation and group search.

- **Box Pushing** - The system should provide possibility to run simulation of two robots transporting the resource in the form of the box. The box should be transported collectively without explicit communication between robots.

- **Aggregation Algorithms** - The system should provide possibility to run simulations of two types of aggregation algorithms: group aggregation and resource aggregation.

- **Simulation Settings** - Any simulation should have possibility of setting different parameters: number of robots in simulation or types of subalgorithms.

- **Data Collecting** - The system after finishing the single simulation, either by fulfilling all assumptions of the algorithm or after execution time threshold, should write the data gathered during simulation to the csv file.

3.2. Non-Functional Requirements

- **Usage of Robot Operating System (ROS)** - The system uses Robot Operating System [8]. ROS is one of the most popular middleware for robotic programming. It also provides several packages useful in swarm robotics, such as *stage* that is used as simulator.

- **Usage of ROS package Stage** - The system uses Stage ROS package as a simulator. The package performs very well when taking under consideration simulations with

usage of multiple robots and provides useful tools to gather data and execute simulations in convenient time.

- **Usage of Gazebo** - The system uses Gazebo as a simulator. Gazebo provides functionalities needed to execute one of the scenarios - box pushing. It is also basic and simple simulator that cooperates with ROS and it is commonly used in the robotic experiments.

- **Usage of Python Language** - The system is implemented with usage of Python language. It is one of the most popular and convenient programming language, it is also commonly used in ROS environment.

- **Execution Time Threshold** - The system should treat the simulation as the finished one when all assumptions of algorithm are fulfilled or the simulation lasts longer than the 500 seconds of simulation time. After that time the execution is terminated and the data is written.

- **Automatization of Simulation Execution** - The system should be able to run multiple simulations in sequence. There should be a script that executes several simulations of one type and gather all data obtained from executed simulations.

- **Maximum Number of Robots** - The system should not perform simulations with number of robots exceeding 10, because of the decrease in performance and real time simulation execution is significantly longer.

4. Embodied Agent Approach

Before introduction of term ***Embodied Agent*** the definition of actual ***Agent*** has to be established. As Zieliński and Kornuta in [9] explain: *"An agent is understood as entity that perceives its environment in order to decide on its further action upon environment"*. Further in their work is mentioned term of ***Rational Agent***, an entity that possesses an additional imperative for reaching the desired goal. This imperative influences the undertaken actions in order to maximize the agent's expected utility [9]. Simple visualisation of interactions between agent and environment are visible in the **Fig 4.1**.

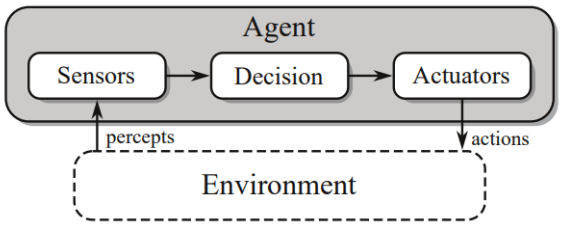


Figure 4.1. Interaction between agent and its environment [9]

Embodied Agent is described in [10] as: *"An agent which has a physical body that posses an effector that affects the state of the environment"*

Taking that thesis under consideration as a result the ***Embodied Agent*** is: *"An entity that perceives its environment in order to decide of its further action upon environment, that possesses additional imperative for reaching the desired goal, which has a physical body"*.

4.1. General Inner Structure of an Embodied Agent

General inner structure of an Embodied Agent is widely described in [9]–[14] and showed in the **Fig 4.2**. The robotic system consists of agents a_j for $j \in \{j_1..j_{n_a}\}$ where n_a is number of agents in this robotic system and j is the name of agent. Every agent consists of: control subsystem - c_j, virtual effectors - $e_{j,n}$ where n is designator of effector, virtual receptors - $r_{j,k}$ where k is designator of receptor, real effectors - $E_{j,h}$ where h is designator of effector and real receptors(sensors) - $R_{j,l}$ where l is designator of receptor.

- c_j **- Control Subsystem** - every agent has only one instance of this entity, the system is responsible for performing operations, basing on agent inner imperative and data obtained from virtual receptors($r_{j,k}$), and taking under considerations those operations sending commands to virtual effectors($e_{j,n}$),

- $e_{j,n}$ **- Virtual Effector** - n is the name of virtual effector (agent can have them more than one), each virtual effector is responsible for translating commands given to it

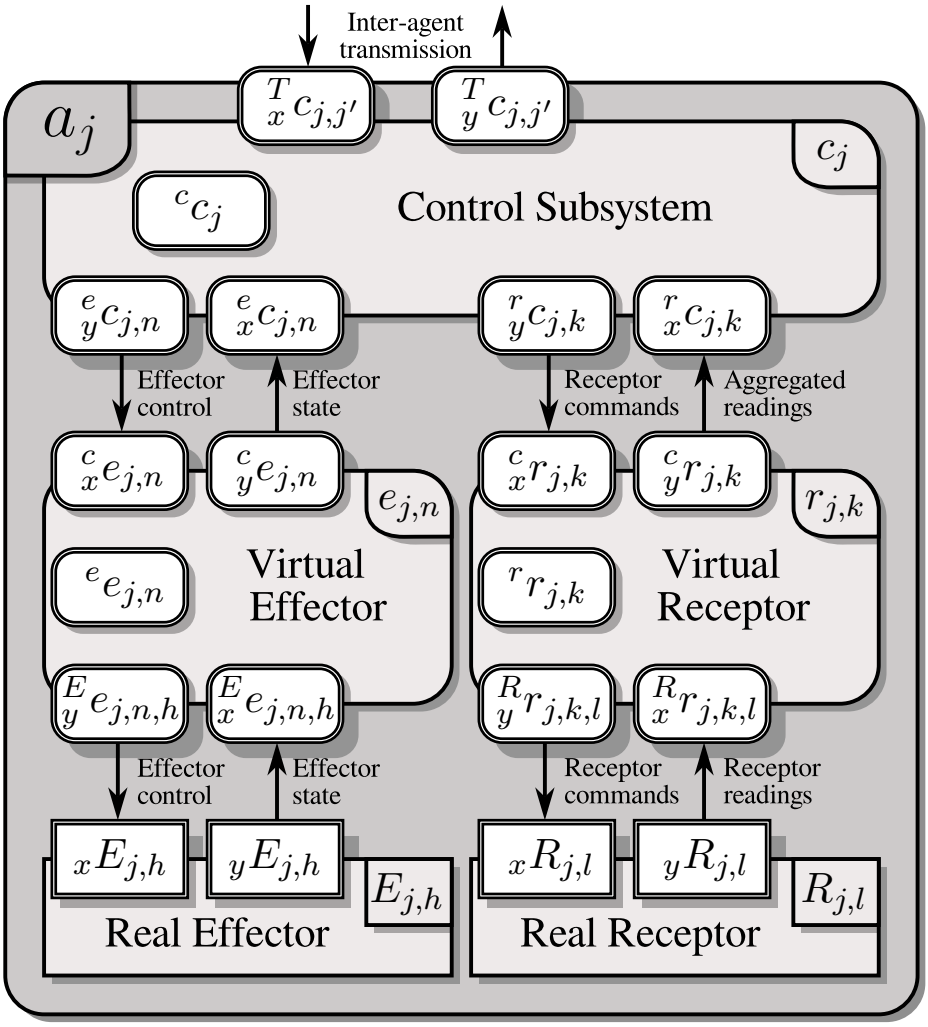


Figure 4.2. General structure of Embodied Agent a_j [12]

by control subsystem(c_j) to signals driving real effectors($E_{j,h}$),

- $r_{j,k}$ **- Virtual Receptor** - k is the name of virtual receptor (agent can have them more than one), each virtual receptor is responsible for translating signals given to it by real receptor($R_{j,l}$) to data that can be processed by control subsystem(c_j),

- $E_{j,h}$ **- Real Effector** - h is the name of real effector (agent can have them more than one), each real effector is physical device that can affect the environment of the agent,

- $R_{j,l}$ **- Real Receptor(Sensor)** - l is the name of real receptor (agent can have them more than one), each real receptor is a physical sensor that gathers information from environment.

Embodied agent is largely based on relations between agents and subsystems of an agent. In the **Fig 4.2** that relations are marked. Agent a_j can communicate with other agent $a_{j'}$ via transmission buffer where $j \neq j'$. The internal relations between subsystems of agent are even more significant for this structure. The communication is implemented with usage of buffers. The systematic denotation method [9]–[14] is used to present those buffers. The letters describing subsystems of agent (c, e, r, E, R) are used as a center sign describing the buffer. The right subscript describes respectively, name of the agent, name of subsystem and buffer element. The left subscript denotes the application of the buffer. If it is communication buffer it is described as x in case of input and y in case of output. The lack of this subscript indicate that the buffer is treated as the internal memory of the subsystem. Left superscript is describing the type of the subsystem where the data is send or from the data is gathered, or in case of internal memory the superscript is the same as the center sign. Right superscript determines the discrete time of the operation. The example of the buffer description is ${}^{E}_{x}e^{i}_{j,n,h}$ and it denotes the input buffer of virtual effector $e_{j,n}$ of agent a_j that acquires the data from real effector $E_{j,h}$.

4.2. Activity of the System

The system consisting of embodied agents, performs predefined tasks, that have affect on environment of agents. The outcome of the system is based on interactions between system's agents and behaviour of each agent emerges from interactions and activities of subsystems of the agent. In [10] those activities were described. The activity of each subsystem $s \in \{c, e, r\}$ can be represented by hierarchical Finite State Machine. For sybsystem $s_{j,v}$ the FSM is denoted by ${}^{s}F_{j,v}$. Every FSM consists of nodes and the edges. Every node corresponds to subFSM that describes the behaviour that is performed with given subsystem. Every edge is correlated with the condition. If the condition is always fulfilled there is no description on the edge.

In the **Fig 4.3** is presented an example of such FSM for control subsystem of robot described in [14]. The robot has to fulfill the task for collecting balls scattered over some area. The robot has three real receptors: two cameras, one that detects balls and another one that check whether the ball is in container, and sonar that detects the obstacles. The robot has also two real effectors: vacuum that sucks founded balls into its container and body that controls the movement of the robot. The FSM ${}^{c}F_{core}$ presents the activities of control subsystem for previously described robot. The task starts with search behaviour that uses the cameras to detect balls and sonar to detect obstacles. If the obstacle is found the behaviour is changed to obstacle avoid. If there is no obstacle detected and the ball is found the behavior changes to movement that moves robot to detected ball. During this state the detection of obstacle also changes behaviour to avoidance. If there is no obstacle the robot approaches ball and changes behaviour to start that gives signal to vacuum to

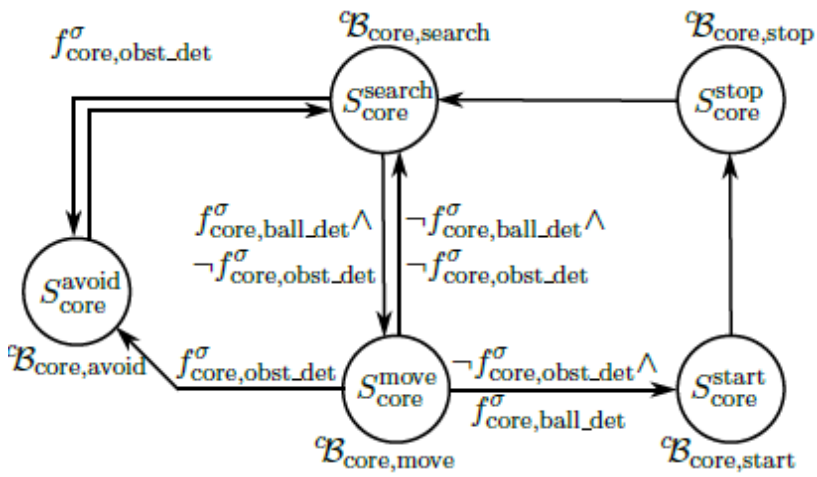


Figure 4.3. Activities of the control subsystem represented by an FSM ${}^{s}F_{core}$ [14]

suck the ball. After that action the behaviour is changed to stop the vaccum and proceeds the whole actions circle again.

States of the FSM describing given subsystem are linked to the corresponding behaviours that can be presented on subFSMs. Notation of subFSM, that describes given behaviour ${}^{s}B_{j,v,\omega}$, where ω is the name of the behaviour, looks as follows ${}^{s}F^{B}_{j,v,\omega}$. The exemplary subFSM is presented in the **Fig 4.4**.

The behaviour relies on iterating the same workflow that includes computing the transition function ${}^{s}f_{j,v,\omega}$ basing on data located in internal memory of subsystem ${}^{s}s^{i}_{j,v}$ and the input buffers ${}_{x}s^{i}_{j,v}$, and sending the output by the output buffers ${}_{y}s^{i+1}_{j,v}$. Right superscript is visualization of time laps, the increment by one symbolizes the duration of one iteration. After transition function is computed, the error condition ${}^{s}f^{\epsilon}_{j,v,\beta}$ and terminal condition ${}^{s}f^{\tau}_{j,v,\xi}$ are tested, and the output buffers and the internal memory are updated. The frequency of iteration of the behaviour is set basing on the time duration of one iteration.

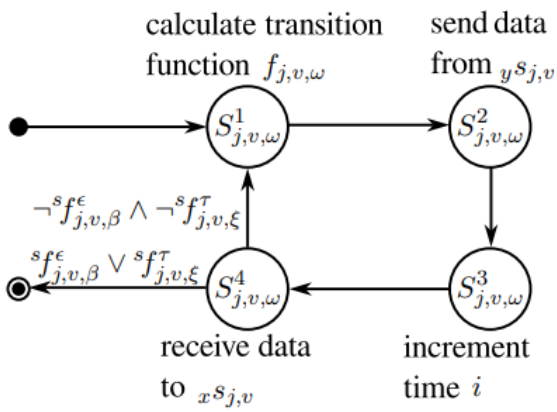


Figure 4.4. ${}^{s}F^{B}_{j,v\omega}$ specifying the behaviour ${}^{s}B_{j,v,\omega}$ [10]

5. Description of Swarm Robotics Basic Subtasks

In this chapter basic and typical subtasks for swarm robotics systems are presented and described. Those tasks are the core of this engineering thesis.

At the begininnig of this chapter the basic algorithms are presented, that are widely used in robotics as the essential ones. In the **Section 5.1.1** two types of random walk are presented, and the **Section 5.1.2** describes the algorithm of following that is responsible for going toward previously defined point.

Further in this chapter more complex algorithms are included. The algorithms correspond to the subtasks being the components of the main algorithms, that perform the tasks of gathering the resources and transporting them to defined location. In **Section 5.2.1** Resource Search algorithm is presented, in next section (**Section 5.2.2**) Waiting for Another Robot algorithm is showed. **Section 5.2.3** presents the Group Aggregation algorithm and the **Section 5.2.4** contains the information about algorithm named Group Search. The last subtask's algorithm is described in **Section 5.2.5**, the Transporting the Resource algorithm.

In the next, and the last part of this chapter presented are two algorithms that executes the main task. The algorithms perform the same tasks using slightly different approaches. The first one, Search-FormTeam algorithm, described in **Section 5.3.1**, prioritizes finding the resource over the merging robots into pairs. The second algorithm, FormTeam-Search, described in **Section 5.3.2**, firstly searches for other robots to merge into pairs to later searches for resource together.

5.1. Basic Tasks

5.1.1. Random Walk

The random walk is above all simple behaviour. That is the main cause of the common usage of this algorithm in the swarm robotics, where robots are also as simple as possible and are not designed to perform complex computations. Despite the fact that the main advantage of random walk is its simplicity, random walk is also flexible, scalable and robust that overlaps with the three main motivations of swarm robotics.

After small research, several types of random walk were found, and taken under consideration: Brownian Motion, Correlated Random Walk, Lévy Walk, Lévy Taxis and Ballistic Motion. Those behaviours are presented, described and examined in [15], [16]. Those algorithms can be divided into two groups. First one contains Brownian Motion, Correlated Random Walk, Lévy Walk and Lévy Taxis, and the second contains only Ballistic Motion. The first group can be described with general behaviour: robots move at random direction for a specific amount of time to change that direction after the time is up or an obstacle was found. Despite the fact, that every of these four algorithms differs slightly in their exploration parameters by different values movement time and direction change,

there was taken the decision to merge those algorithms into one and modify it slightly, creating the Alternating Walk. It can be visible in **Fig 5.1**, the time(**t**) of forward movement is predefined and its obstacle avoidance algorithm is a simple rotation. The second group, **Fig 5.2**, that contains only one behaviour, differs from previous group in the fact that there is no time of movement before changing the direction. In that case the direction is changed only when the obstacle is detected. Those two groups provide slightly different usage, the first one specialises in intensification, what means gathering deeper information on smaller area, while the second one goes toward coverage that takes smaller amount of data but on the bigger area. In both figures the inputs and outputs are presented in general way, because of the fact that both algorithms are used in different situations and tasks, and the input and output conditions, as well as states differ depending on the given scenario.

Finite State Machines **Fig 5.1** and **Fig 5.2** were created basing on the figures included in [16].

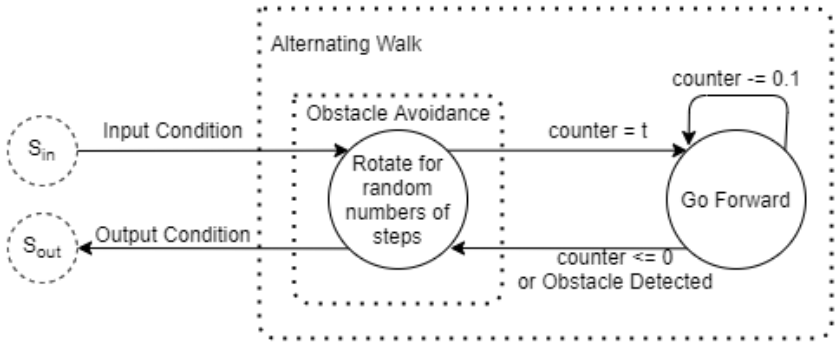


Figure 5.1. Alternating Walk

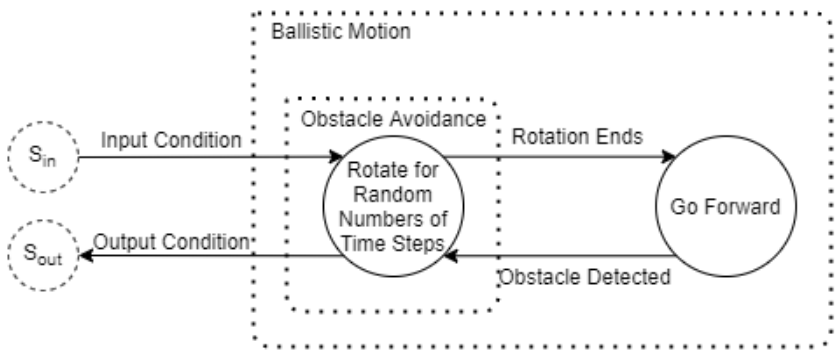


Figure 5.2. Ballistic Motion

5.1.2. Following

The following algorithm bases its core on the movement toward the direction of its goal. The position of that goal can be known to the robot or can be given to it by other robots or by the environment analysis. In [4] the Bayindir described some of the tasks that the communication between robots is used to transfer the data about the common target or even the target for specific robot, that case is taken under consideration by Bayindir in Navigation section of his work, where robots navigate one of them to the target position.

Despite the fact that, most of the algorithms used in this work are based on changing direction whenever the obstacle is detected, in this case the same approach will not be sufficient. The robot that moves to specific given position cannot change its direction in random way. The obstacle avoidance mechanism has to be included. One of the *"bugs"* algorithms, so called *"common sense algorithm"*, described and compared with similar ones in [17], looks as the perfect one for this task. It is very good choice due to its simplicity, and low error occurrence possibility.

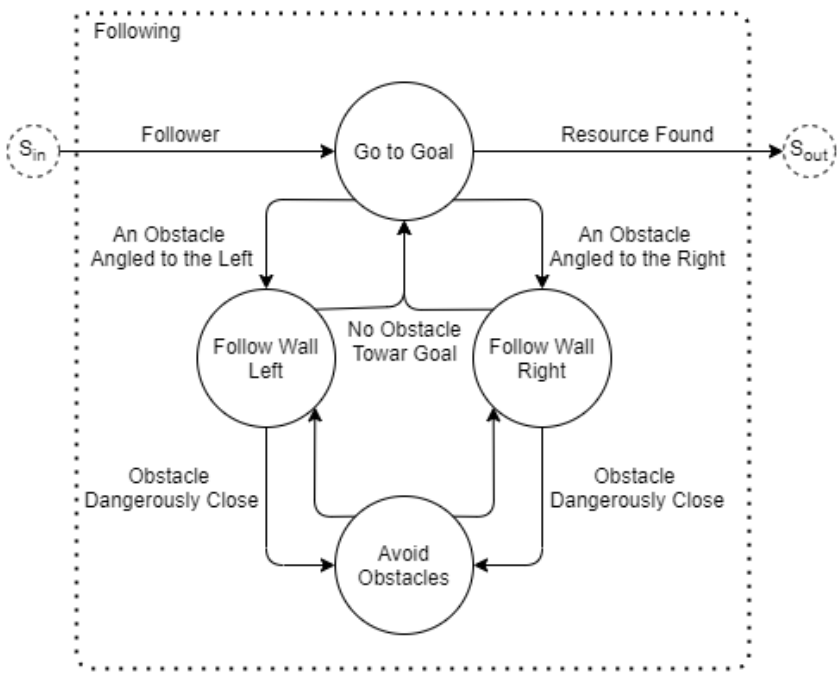


Figure 5.3. Following [18]

There was used, however, the modified version of the *"common sense algorithm"* that slightly improves that algorithm. It is described by Nick Mccrea in [18], and the general mechanism is presented in the **Fig 5.3**. The robot moves toward the destination and when the obstacle is detected it starts to follow the wall of that obstacle. The modified version differs from the *"commons sense algorithm"* in robot choice of the direction in which it follows that wall. The robot takes the direction in which it will start to approach the target

destination in the first place. When there is no longer any obstacle on the way to the goal destination, the robot leaves the wall following state and starts to directly go to the target position.

5.2. Subtasks

5.2.1. Resource Search

In the **Fig 5.4** can be observed an algorithm named Searching for Resources. To find multiple items scattered over some area using multi robot system, searching algorithm has to be developed. Apparently, one of the best algorithms providing deployment and area coverage is random walk, due to its simplicity and low resources usage. In case of resource searching random walk is more than sufficient and was described in details in **Section 5.1.1**. The state is changed when robot finds resource that is not already transported by a pair of robots. If the resource that is found is occupied by another robot that is in waiting state, robots change the state to the **S3**, *Merging Into a Pair* that is showed in the **Section 5.3.1**. When there is no robot waiting close to the resource, robot changes its state to *Wait for Another Robot* (**S2**), described in next paragraph.

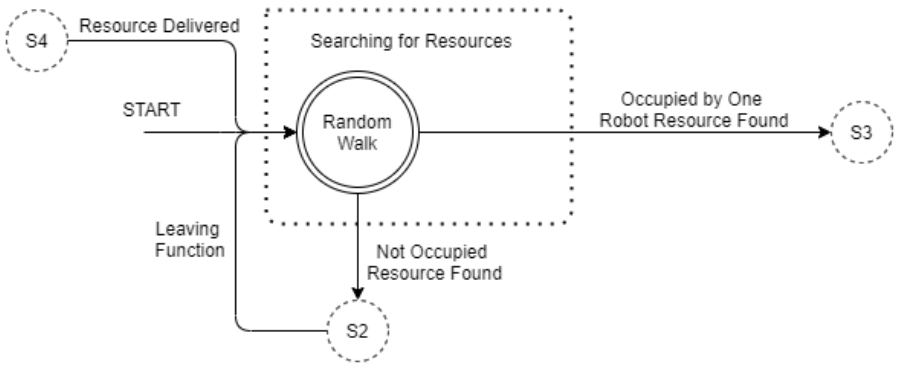


Figure 5.4. Searching for Resource Algorithm

5.2.2. Wait for Another Robot

The waiting state is one of the simplest tasks and its behaviour is visible in the **Fig 5.5**. That algorithm provides possibility to create a group of robots. The robot does not move, sends only the message that it is waiting for a pair. There are many ways to emit such a message, like sound or light impulse in case of implicit or dedicated channel in case of explicit. Despite the fact that the algorithm is simple there can appear a problem. The problem that emerges when there is relatively high number of resource items in comparison to number of robots. The situation where every robot waits for help from others is possible and there can occur deadloop. To prevent such an incident

waiting algorithm has to include abandonment of the task mechanism. There are several options of such a mechanism: timeout, that can be both predefined or being changed dynamically, or probabilistic algorithm where probability can be changed dynamically or be predefined too. After decision to leave the resource robot changes its state back to Searching for Resource (**S1**, **Section 5.2.1**), and starts looking for other resources. If the leaving mechanism is not used, the state changes to the **S3**, *Merging Into a Pair*, mentioned also in previous section, and showed in **Section 5.3.1**.

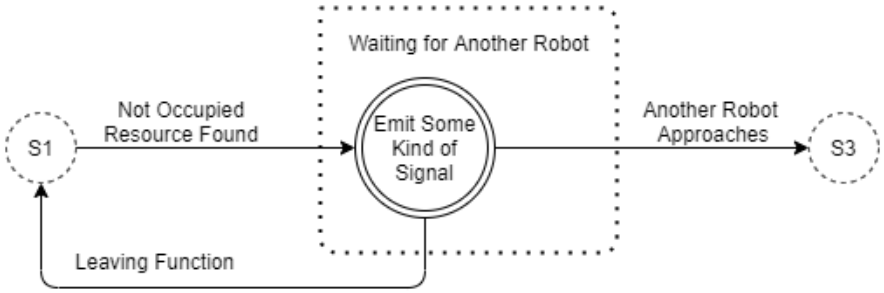


Figure 5.5. Wait Algorithm

5.2.3. Group Aggregation

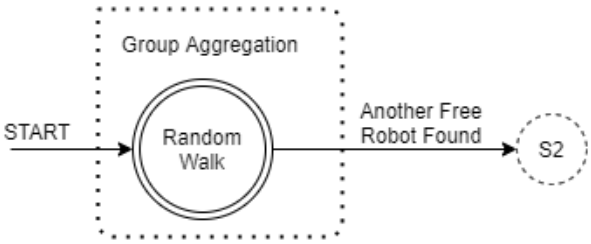


Figure 5.6. Group Aggregation

In the second scenario the robots start with the Group Aggregation algorithm, **Fig 5.6**. They are grouping with each other into small collectives (pairs). The robots can emit signals during searching phase. That phase relies on Random Walk (**Section 5.1.1**). When robot finds another robot, that is already grouped, it treats that robot as an obstacle. Already created collectives have to somehow communicate with each other to uphold this connection. The simplest way is via dedicated channels using explicit communication, but implicit communication like light or sound of specific for the group frequency works well. After merging to the pair, robots change the state to Searching in Group (**S2**), described in next section.

5.2.4. Group Search

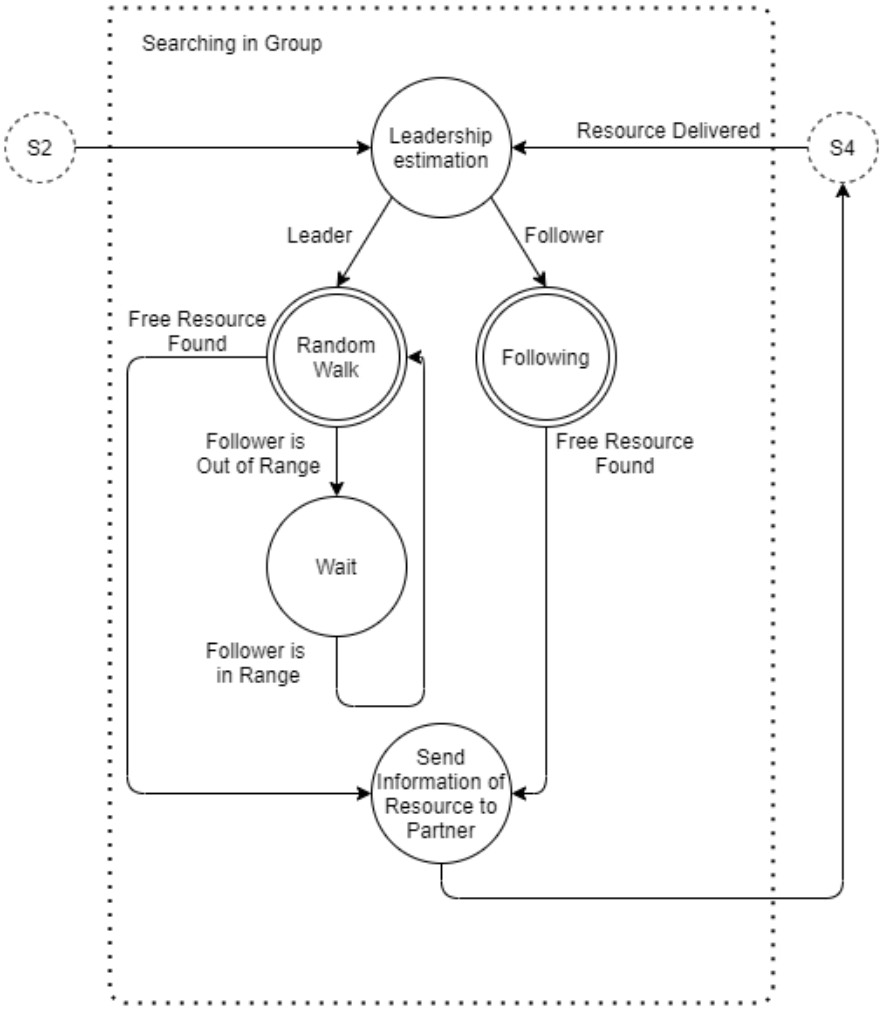


Figure 5.7. Group Searching

Task Searching in Group is very similar to the algorithm from the first scenario (**Section 5.2.1**), however, the problems of synchronized movement have to be taken under consideration. There can occur situation where one of robots will have to stop or change direction of movement because of some obstacle. In that case another robot has to wait or change the direction accordingly, to still move with its partner. The simplest solution is the division of robot team into two subgroups: leader and follower. The leadership estimation has to take under consideration collective decision making. Valentini in [19] presented solution to the problem as a whole. After leadership estimation the leader has almost the same behaviour as "Search for resources" in the **Section 5.2.1**. The only difference is that additional state is required which ensures that the leader will move with the follower. The leader has to wait for the follower if it is too much behind. The follower on the other hand has to include Following algorithm (**Section 5.1.2**) that includes the obstacle avoidance

needed to successfully follow the leader. The algorithm of Group Search can be observed in **Fig 5.7**. That stage changes to Transporting the Resource (**Section 5.2.5**, **S4**) state when not occupied item (item that is not already transported by another pair of robots) is found by any of a pair of robots.

5.2.5. Transporting the Resource

Transporting the Resource algorithm is common for both approaches to perform the task of gathering the resource. No matter which one is chosen, the robots, after finding the resource, execute exactly the same algorithm of transportation. In this work the transportation mechanism that is considered is the cooperative box pushing. The algorithm is based on the example presented in [20].

The algorithm assumes that robots do not communicate with each other, however they know the direction in which they should push the box. There is taken the assumption that in real life implementation there can be another robot, that moves ahead the box and informs the pushers about the goal direction. The robots perform the tasks basing on the state that is chosen. The visualisation of the behaviour of this algorithm can be visible in the **Fig 5.8** The algorithm consists of four states: *Maintain Box Velocity, Maintain Pusher Direction, Maintain Box Direction* and *Align Pusher to the Box.*

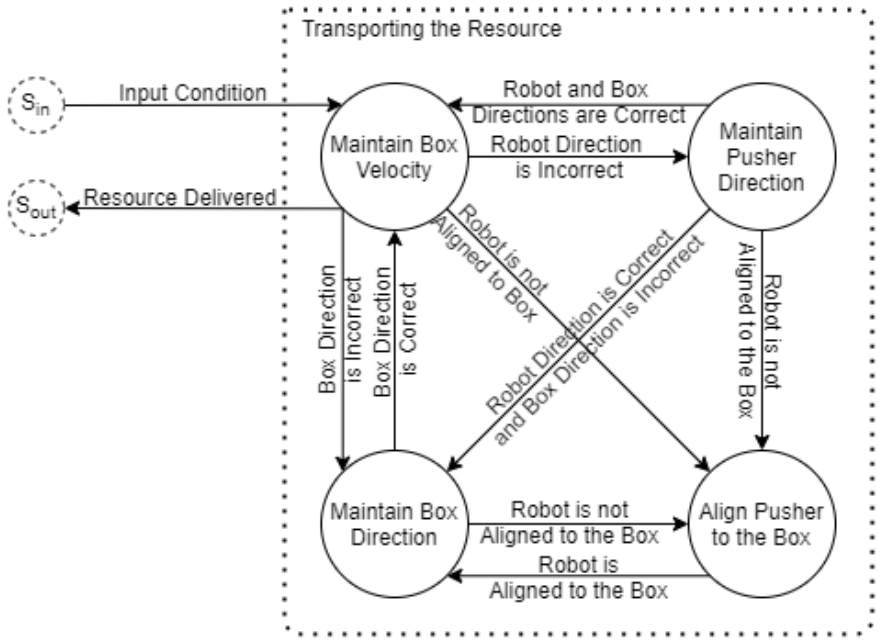


Figure 5.8. Transportation

Maintain Box Velocity - the simplest state where both the heading of the box and the heading of the robot are placed toward the goal direction. The robots just maintain the

linear velocity and push box in the straight line.

Maintain Pusher Direction - the state where the robot is not directed toward the goal direction. In that case the robot simply turns itself toward the goal direction.

Maintain Box Direction - in that case the box does not point toward the goal direction and the pushers are heading toward the goal direction. To adjust the box heading one robot does not move, letting the other one to rotate the box around him. Another robot rotates it by simply moving forward.

Align Pusher to the Box - the state that takes priority over the others. Whenever the robot looses the connection to the box the heading of the box or the robot are neglected and the robot starts to search the box. The box is found by the detection of the corner with usage of laser sensor. After that detection the robot approaches the box and after the alignment it changes the state basing on the current placement of the robot and the box.

5.3. Main Task

All algrithms mentioned and described earlier in this chapter are used as a subtasks to perform the main task, that relies on finding and gathering scattered over some area resources. There were selected two approaches to solve this tasks and both are described below.

5.3.1. Search - FormTeam

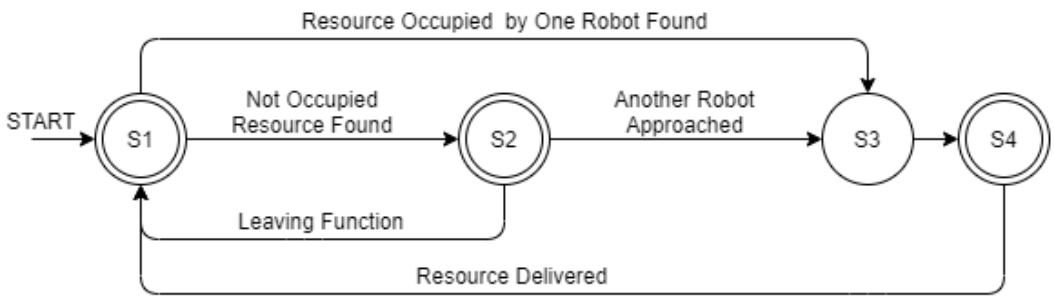


Figure 5.9. Search - FormTeam Algorithm

The first approach, **Search-FormTeam**, prioritizes the search for resources, it is executed in **S1** with usage of algorithm described in **Section 5.2.1**. That algorithm is changed, when the resource, that is not occupied by two robots, is found. If the resource is completely free (any robot does not wait close to the resource) the state is changed to **S2**, *Wait*

for Another Robot, described in **Section 5.2.2**. If the resource is occupied by another robot the state is changed to **S3**, *Merging Into a Pair*, where robots create a group and then the robots collectively transport the resource in **S4**, *Transporting the Resource*, described in **Section 5.2.5**. After resource delivery the algorithm starts from the **S1**, and the whole operation is executed again. The algorithm is presented in the **Fig 5.9**.

5.3.2. FormTeam - Search

The second approach, **FormTeam-Search**, **Fig 5.10**, at the first place creates the pairs of robots. Single robots search for other ones, **S1** described in **Section 5.2.3**, and when they find it they merge into a pair, **S2**. After merge the robots collectively search for resources moving in formation that increases their field of view. That part is executed in **S3**, and that algorithm was described in **Section 5.2.4**. When the resource is found the robots transport it (the resource) in the same way as it is transported in the previous approach (**Section 5.3.1**). The algorithm of transportation was described in **Section 5.2.5**. After successful resource delivery the algorithm is not executed from the beginning. The robots are already in pairs so the *Group Aggregation* and *Merge Into a Pair* can be neglected and the state changes directly to *Group Search*, **S3**.

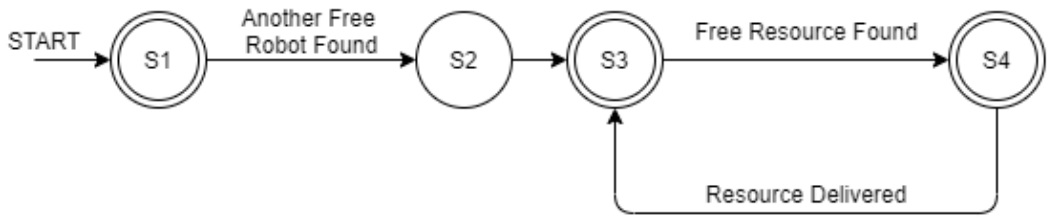


Figure 5.10. FormTeam - Search Algorithm

6. System Specification

The system presented in this engineering thesis takes under consideration the tasks and activities that should be performed by the group of robots preserving the assumptions of swarm robotics. The robots used to conduct the simulations and execute the given tasks are homogeneous, and the structure of the system expressed using embodied agent approach is presented in **Section 6.2**.

The system consists of embodied agents that communicate with each other via environment. Because of relatively high number of robots, there was taken a decision that part of receptors and effectors used by robots would be simulated by aritificially created environment. The assumption for this approach was simplicity and resource saving. Instead of real effector, e.g. a light source used to inform other robots in local area what is the state of the robot owning the effector (determined by frequency of the light), the environment gives that information in an explicit way to the robots that could know that information in "normal" circumstances (the robots that are in close area range). The creation of the artificial environment is implementation procedure. High number of robots and the simulation environment where the experiments are performed needs simplification. Otherwise, the system can work with significant delay. The general purpose of the artificial environment is to simulate the implicit signals that would be transferred in the robots world into the explicit equivalents that needs less computing resources and are easier to handle while behaving the basic assumptions of swarm robotic: flexibility, robustness and the most linked to this approach scalability.

6.1. Robot Structure

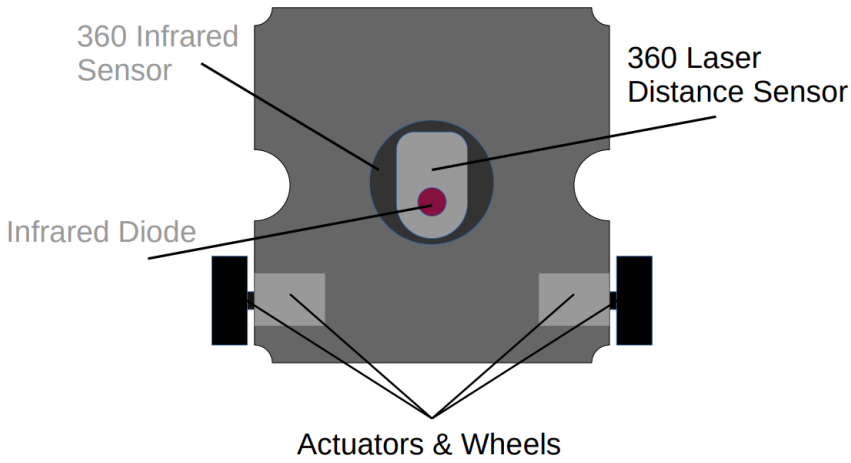


Figure 6.1. Turtlebot3 Waffle Pi (simulated parts in grey font)

The robot model which is used in the system to perform simulations is based on the popular model of robot, Turtlebot3 Waffle Pi, its simple sketch can be visible on **Fig 6.1**. That model is a ROS standard platform robot. It is commonly used both in the real life experiments and in the simulations. As the effectors robot posseses two wheels with actuators to move. The receptor is the 360 distance laser sensor. Moreover, there are effector (infrared diode) and receptor (infrared sensor) simulated by the environment that provide implicit communication between robots. Despite the fact that diode and sensor are simulated, they are presented in the **Fig 6.1**, because the driver treats them as the real ones.

Robot has two wheels with actuators, one on each side of it. That placement of the wheels determines type of movement model for this robot, the differential one. That model takes under consideration the velocities that are set to each of the wheel. If the values of velocities are the same and not zero on both wheels then the robot moves forward in the straight line in case of the positive values, and backward if those values are negative. If those values differs for every wheel, the robot turns toward left or right. The common practice in case of simulations and controllers to differential drive is to translate that model into the unicycle model. The unicycle model also has two variables but one is linear velocity - v, and the other one is angular velocity - ω. The differences between those two models can be observed in the **Fig 6.2**.

(a) Unicycle Model **(b)** Differential Model

Figure 6.2. Comparison between Unicycle Model and Differential Model [18]

Robot is equipped with 360 Laser Distance Sensor. Its the receptor that senses the objects all around the robot. In case of this simulations the sensor posses 360 samples of laser beams and can determine the distance up to 3.2 meters, however the smaller range is taken under consideration in simulations and it's equal to 0.5 m.

6.2. Structure of Single Embodied Agent

As it was previously mentioned, the system consists of homogeneous robots, each controlled by single embodied agents where a_j is an agent of a name $j \in \{1..n\}$. The agent posses one real receptor, $R_{j,scan}$ that is an Laser Distance Sensor, and one real effector, $E_{j,body}$, it corresponds to the movement of robot. There are also two simulated devices, single receptor - infrared sensor $R_{j,sens}$, and the effector - infrared diode $E_{j,diode}$.

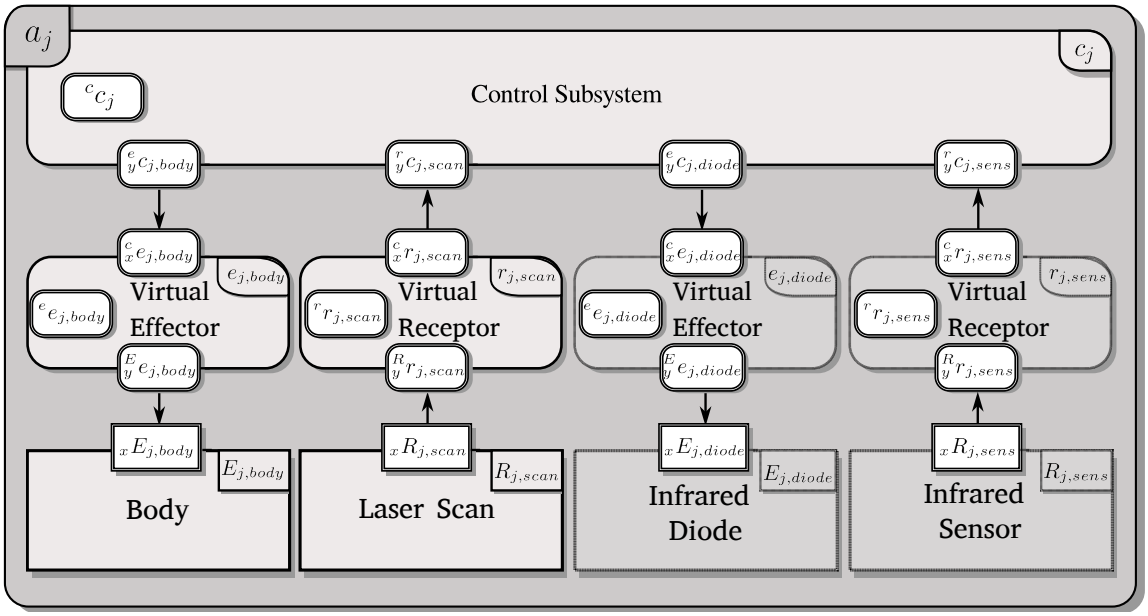


Figure 6.3. Structure of the agent a_j

The agent structure is presented in the **Fig 6.3**. It consist of five subsystems:

- c_j - the control subsystem that controlls the behaviours based on the data from the receptors and sending the controller instructions to the effectors,
- $e_{j,body}$ - the virtual effector named *"body"* that translates *unicycle model* (**Fig 6.2a**) that considers the movement values to be:
  -linear velocity - v
  -angular velocity - ω
  to the *differential model* (**Fig 6.2b**) that consider the movement values to be:
  -left wheel velocity - v_L
  -right wheel velocity - v_R
- $r_{j,scan}$ - the virtual receptor named *"scan"* translates the values from the real receptor to the convenient for calculations form of data,
- $e_{j,diode}$ - the virtual effector named "diode" that translates robot state to corresponding infrared light frequency,
- $r_{j,sens}$ - the virtual receptor named "sens" that after gathering the given infrared signal translates it to the information about the sender state.

6.3. Single Embodied Agent Activities

Subsystems, presented in the Structure of Embodied Agent, cooperate to achieve greater goal and fulfil the task that was set. Each subsystem has its own activities and behaviours that provide the possibility to perform that task.

Control Subsystem - c_j - basing on the data given to it by two receptors, Laser Scan and Infrared Sensor, performs specific action and sends commands to effectors of the system. The example of the action taken by the c_j is presented in the **Fig 5.4** in **Section 5.2.1**, where it changes states to *Resource Found* and *Wait*, or is still searching for the resource, basing on the information whether the found resource is fully free, is occupied by one robot or is occupied by a pair of robots.

The virtual effector named "body" - $e_{j,body}$ - is responsible for a movement of the robot. The commands given to it by the c_j are translated into the commands known to the real effector - the real actuators and the wheels. The example of usage of this effector can be visible in the **Fig 5.2**. There is a state named **Go Forward**. The effector makes the actuators move the robot by turning the wheels.

The virtual receptor "Laser Scan" - $r_{j,scan}$ - translates the information from the Distance Laser Sensor and based on that data the c_j makes decisions. The example of its usage is visible in the **Fig 5.3**. There can be seen condition *An Obstacle Angled to the Left*. That knowledge comes from the $r_{j,scan}$ that tells that there is detected an obstacle to the left, and basing on that knowledge the c_j changes state.

Next to subsystems - $e_{j,diode}$ and $r_{j,sens}$ - are the simulated ones. Their only purpose is the local communication between robots. Basing on the detected infrared signal the robots know that the other robot that sends that signal is in specific state. The algorithm that uses those subsystems is presented in the **Fig 5.6**. The information whether the found robot is free is known due to the infrared sensor. That information allows c_j to change the state and merge the robots into a pair.

7. Tools

In the following section, details about tools and technologies used in the project are presented. In the first part there is description of Robot Operating System, an environment that enables to build robot applications. The second part is about a ROS package that provides simulator named "Stage ROS". In next, third, section the Gazebo simulator is presented. In the last section the library from Python, rospy, is taken under consideration. Rospy enables to write Python code using ROS middleware.

7.1. Robot Operating System

ROS is complex environment that provide a lot of utility. As we can see on its page:

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms" [8].

The basic structure of this framework looks as follows:

- **Node** - in simple words, an executable file inside of ROS package, that using ROS tools can communicate with other nodes. The communication can be achieved with usage of topics, to which node can publish or subscribe, or with usage of services that are direct form of communication.

- **Topic** - Topic is an entity used in nodes' communication. It receives messages from nodes that publish to the topic and sends those messages to nodes that subscribe to it. Topics are indirect connection between nodes that does not make nodes directly dependent to each other.

- **Service** - direct type of communication provided by ROS. Nodes can communicate using them by sending request to given service and receiving the response from another node in predefined data type.

ROS is commonly used in robotic programming, because of its universal application and huge set of tools and libraries. Despite the fact that, ROS is not actual simulator and has to use external one, usually Gazebo [21] (**Section 7.3**), or one of its packages like Stage [22] (**Section 7.2**), it is perfect for most of robotic research and provides good tools to develop own packages or libraries.

7.2. Stage ROS

Stage [22] is ROS package that provides mobile robot simulator. The simulator consists of virtual world with mobile robots and sensors. The data of robot's positions and states, and readings from sensors are available via ROS topics where Stage publishes data. The commands controlling robots also use topics but in this case the robot controllers subscribe to them. Example of such a topic is *cmd_vel* that takes movement commands for robots.

The simulation and the world with objects is generated based on *.world* file, below is visible example of world of size 10x10m with 2 robots:

Listing 1. 10x10_2rbts.world example

```
include "turtlebot.inc"

name            "10x10_2rbts"
interval_sim    10
resolution      0.02
threads         1

define floorplan model
(
  color "gray30"
  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0
  gui_outline 0
  gripper_return 0
  fiducial_return 0
  laser_return 1
)
# parameters of window of application
window
(
    size    [ 10 10 ]
    center  [ 0 0 ]
    rotate  [ 0 0 ]
    scale   10
)
```

```
# parameters of the world
floorplan
(
  name "swarm"
  bitmap "./swarm.png"
  size [ 10.0 10.0 2.0 ]
  pose [  0.0  0.0 0.0 0.0 ]
)
# initialization of two robots
turtlebot( pose [ -3.0 0.0 0 0 ] name "r00" color "red")
turtlebot( pose [ 3.0 0.0 0 0 ] name "r01" color "red")
```

The file above includes another file, that contains information about the robots that would be generated. The file of these robots can be visible below.

Listing 2. turtlebot.inc

```
define kinect ranger
(
  sensor
  (
    range_max [ 0.2 5.0 ]
    fov 360
    samples 360
  )
  # generic model properties
  color "black"
  size [ 0.06 0.15 0.03 ]
)

define turtlebot position
(
  pose [ 0.0 0.0 0.0 0.0 ]

  size [ 0.2 0.2 0.2 ]
  origin [ 0.0 0.0 0.0 0.0 ]
  drive "diff"
  color "gray"

  kinect(pose [ -0.1 0.0 -0.11 0.0 ])
)
```

Those two files with usage of Stage ROS generate the world with two robots that posses lasers sensors, on the arena whose size is 10x10m. The image of this world is visible in the **Figure 7.1**.

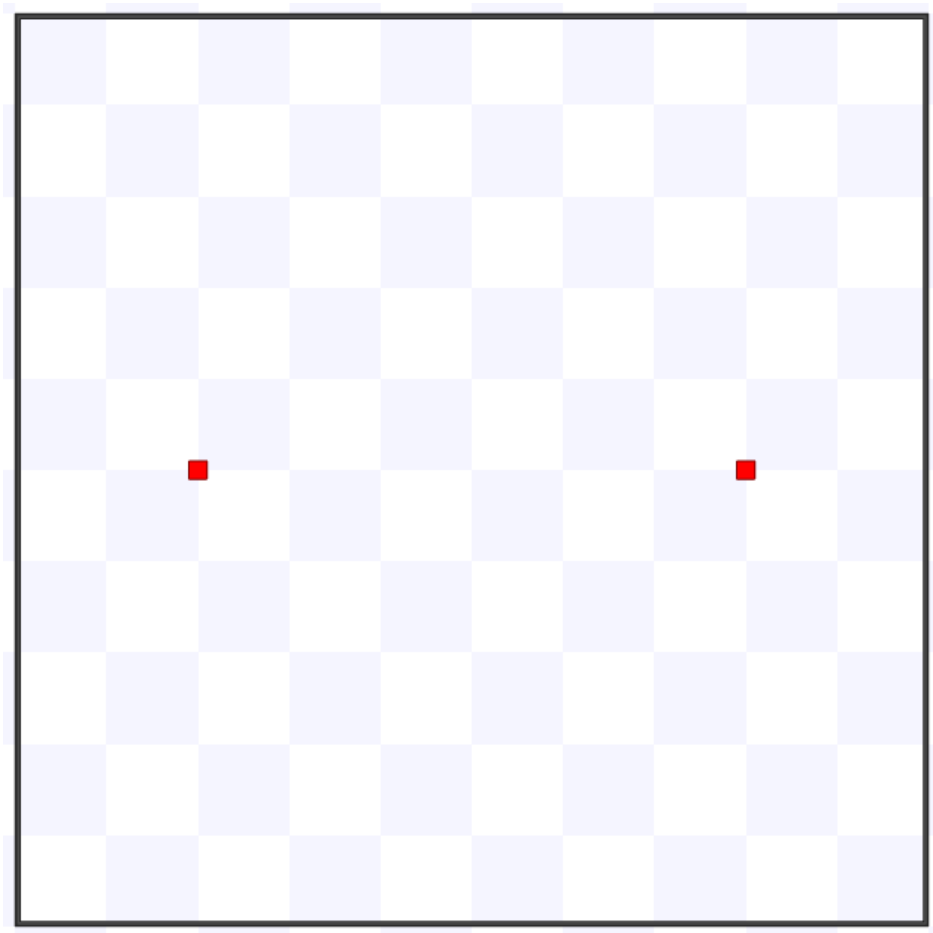


Figure 7.1. Two robots world 10x10

7.3. Gazebo

Gazebo [21] is an open-source 3D robotic simulator that is commonly used in robotics. It provides wide range of tools, and realistic environment with high quality models, light and shadows and textures. The Gazebo also provides the packages that enables to communicate with the ROS topics and services to control the simulation. Below, in the **Fig 7.2** is visible the simple world with one turtlebot3_waffle_pi.

Figure 7.2. Gazebo World with Turtlebot3_Waffle_PI

7.4. Python: rospy

Rospy [23] is the Python library for ROS. It enables creating python files using ROS middleware functionality like: topics, nodes, etc. It provides huge amount of tools useful in robotic programming and gives the opportunity to easily utilise the ROS environment, because of the simplicity of Python.

8. System Implementation

This work focuses mainly on the comparison between two different approaches, **Search - FormTeam** and **FormTeam - Search**. Both algorithms are described in **Section 5**. As we consider the transportation part (**Section 5.2.5**) to be the same in both approaches, hence the comparison can be performed taking under consideration only the part of algorithms that is responsible for aggregation close to resource that is to be transported.

However, there are two approaches to be considered, the aggregation part was divided into three simulations that can be executed separately. The resource aggregation that presents approach described in **Section 5.3.1**, is implemented as a one simulation that makes robots to aggregate close to scatter over arena resources. The second algorithm, described in **Section 5.3.2** is divided into two, standalone simulations that considers the algorithms of grouping into pairs (**Section 5.2.3**) and the algorithm of collective search of resource (**Section 5.2.4**).

The simulations consist of the simulator that provides the world and the actual models of robots to be controlled. The simulator and exemplary world file was described in **Section 7.2**. The execution of the simulation also needs the executable files that contain the scripts that perform the algorithms and behaviours needed to finish the task. The scripts create the nodes in ROS environment and also publish and subscribe to topics. The structures of those files and the connections between them are provided in following subsections.

There is also need to run the scripts and the relations between them. To do this, there is used a lunch file. The example of it is visible below.

Listing 3. resource_aggregation_bm_2.launch example

```
<launch>
    <arg name="r_num" value="2" />
    <param name="rbts_num" value="$(arg_r_num)" />
    <node pkg='turtlebot3_motion' type="resource_aggregation_environment.py" name="env" output="screen" />

    <!--ROBOT 0-->
    <group ns="r0">
        <arg name="num" value="0"/>
        <param name="num" value="$(arg_num)"/>
        <param name="rbts_num" value="$(arg_r_num)" />
        <node pkg="turtlebot3_motion" type="resource_aggregation.py" name="r$(arg_num)_resource_aggregation" output="screen" />
        <node pkg="turtlebot3_motion" type="ballistic_motion.py" name="r$(arg_num)_random_walk" output="screen" />
    </group>

    <!--ROBOT 1-->
    <group ns="r1">
        <arg name="num" value="1"/>
        <param name="num" value="$(arg_num)"/>
        <param name="rbts_num" value="$(arg_r_num)" />
        <node pkg="turtlebot3_motion" type="resource_aggregation.py" name="r$(arg_num)_resource_aggregation" output="screen" />
        <node pkg="turtlebot3_motion" type="ballistic_motion.py" name="r$(arg_num)_random_walk" output="screen" />
    </group>
</launch>
```

This file, *resource_aggregation_bm_2.launch*, launches scripts for environment and two robots to perform resource aggregation algorithm with usage of ballistic motion as random walk algorithm.

8.1. Resource Aggregation

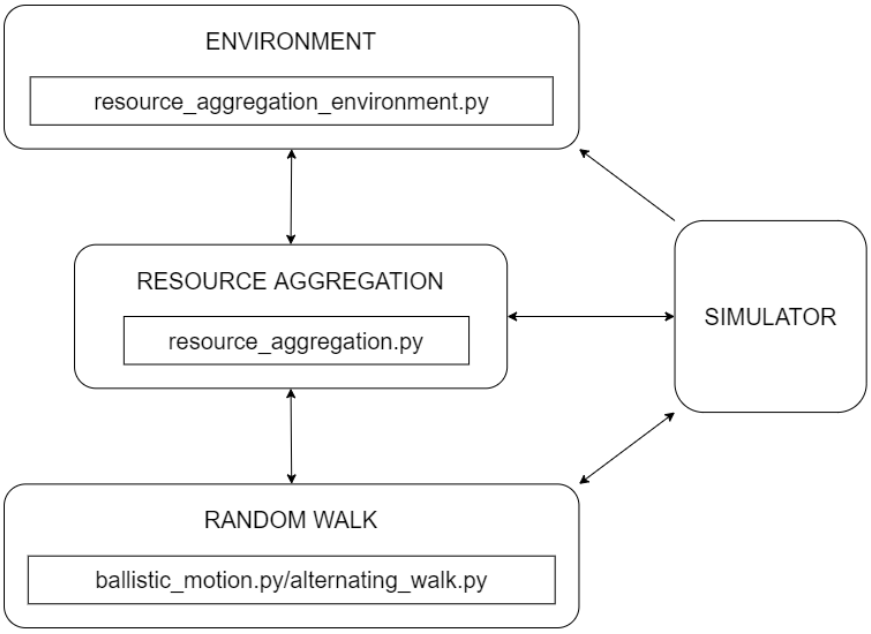


Figure 8.1. Resource Aggregation Nodes Structure

The resource aggregation simulation uses three types of nodes:

- **Environment** - every simulation needs only one instance of this node. It main task is to provide indirect communication between robots and to simulate in explicit way some events.
- **Resource Aggregation** - for every robot in the simulation there is a need for one instance of this node. That node provides controller, that executes the algorithm of resource finding, described in **Section 5.2.1**.
- **Random Walk** - in case of that node, there is also only one instance of it for every robot in simulation. That node performs execution of random walk tasks, described in **Section 5.1.1**. That node can be initialized by two types of scripts, the choice depends on the type of random walk that is to be tested, whether ballistic motion or alternating walk.

There exists the data transfer between nodes, and between nodes and the simulator. The connections are presented in the **Fig 8.1**. The data exchange between nodes is provided by services given by a ROS, while the exchange between nodes and the simulator is provided by topics to which nodes publish data or subscribe them.

Environment subscribes only to one topic - */clock*, that gives the information about time duration of the simulation, that provides output data, that is to be gathered. That node also has connection with every resource aggregation node via services. The con-

nection is used for resource finding and robot finding check. When the robot senses an obstacle, it sends the query whether that obstacle is a robot that waits close to the resource and does not have a pair yet. And other case when robot finds the resource, or rather location, because the resources were implemented as the virtual ones. In that case robot asks whether the resource is free or there is one robot that needs a pair. In both of this cases the environment responds to the query and basic on that the robot changes its state.

Resource aggregation subscribes to two topics and publishes to one. Publisher of that node sends the data to */cmd_vel* topic that is unique for every robot and is responsible for its movement. The topics to which the node subscribes are */odom* and */base_scan* that provides data of, respectively, position of the robot and the readings from laser sensor. The connection between this node and the environment were described but the node provides also service with random walk. The service is simple, it informs the random walk node to be active or not.

Random walk subscribes and publishes to the same topics as the resource aggregation node. And its activity determines whether the node publishes the data to move the robot, or not.

Despite the fact that there is a description of that algorithm, below is located simple pseudocode, that helps in understanding, how the scripts really works.

Listing 4. Resource Aggregation Pseudocode

```
while True:
    if state == random_walk:
        if obstacle_detected == True:
            state = wait
        resource_check()
        random_walk()
    elif state == wait:
        no_move()
        check_robots()
        if detected_obstacle != free_robot:
            state = random_walk
        else:
            state = resource_found
    elif state == resource_found:
        no_move()
```

The whole algorithm is performed in infinite loop, that is executed with certain frequency (in case of those simulations it is 20Hz). The agent actions are determined by the

state where the robot actually is. In case of resource aggregation there are 3 states: *random walk, wait* and *resource found*.

In the first state, *random walk*, the robot checks if it has found the resource, by resource_chcek() function. That implementation assumes that in the script are data about position of resources. The robot basing on its own position, given by the topic */odom* determines whether the resource was found. After that check the robot, asks the environment whether the found resource is free, and basing on the response changes its state to *resource_found* or remains in *random_walk*. The random_walk function makes robot to move, with usage of random walk node. In case when the laser sensor detects any obstacle the control subsystem changes state to *waiting* state.

The *wait* state provides the time to perform computation. The robot does not move and sends to the environment the question whether the obstacle is a robot and if so, what is its state (i.e. either it has found the resource or not). If it found the resource then the state of the searching robot is changed to *resource_found*. If not the state changes to the previous one, *random _walk*.

The *resource found* state is basically the end of task for given robot. The control scripts just send the message to not move.

8.2. Group Aggregation

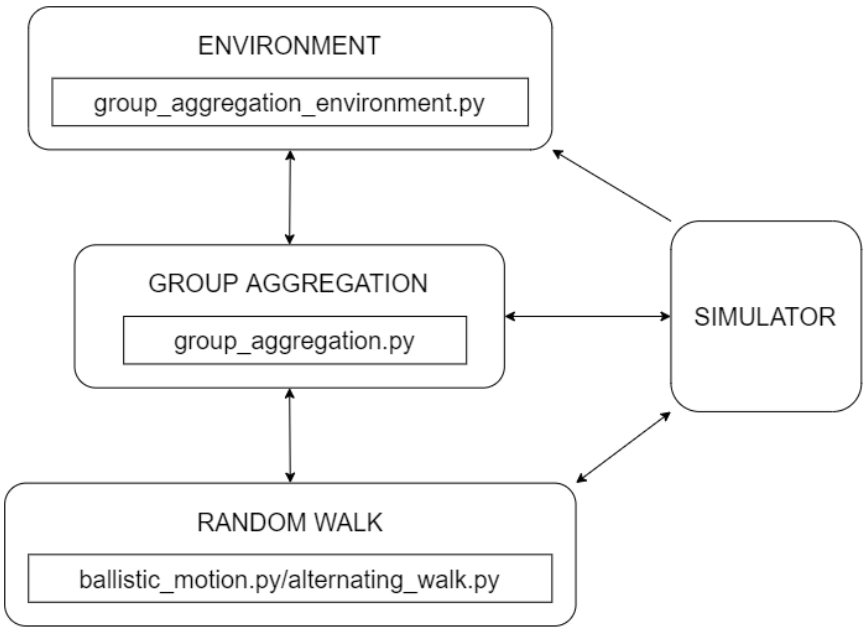


Figure 8.2. Group Aggregation Nodes Structure

Group aggregation simulation is very similar to the resource aggregation in case of the nodes structures. However, the connections between nodes differs slightly in comparison

to the structure described in **Section 8.1**. The structure is visible in the **Fig 8.2** and consists of:

- **Environment** - only one node that provides communication and simulates events.
- **Group Aggregation** - the node that corresponds to the robot and performs algorithm described in **Section 5.2.3**.
- **Random Walk** - the node that has only one instance for every robot and executes algorithm described in **Section 5.1.1**

Environment in that case also subscribes to */clock* but besides that the environment also subscribes to the */odom* for every robot in the simulation. The data of the position is needed when the query from the group aggregation node occurs. That provides the pair check, when the robot senses any obstacle sends the query to the environment to ask about that obstacle. If this object is the robot that is not in the pair yet the environment responds to the query positively and sends the other robot message that the pair was found.

Group aggregation and the random walk nodes posses the same node structure as was described in section above (**Section 8.1**), the only difference is the connection to the environment that was already discussed.

Listing 5. Group Aggregation Pseudocode

```
while True:
    if state == random_walk:
        if obstacle_detected == True:
            state = wait
        random_walk()
    elif state == wait:
        no_move()
        check_robots()
        if detected_obstacle != free_robot:
            state = random_walk
        else:
            state = aggregation
    elif state == aggregation:
        no_move()
```

In case of Group Aggregation, the pseudocode looks very similar to the Resource Aggregations. However, there are minor differences. The excution is also performed by infinite loop, there are also three states: *random walk, wait* and *aggregation*.

In the *random walk* state there is also movement performed by random walk node,

and there is change of state to *wait* if the obstacle is detected. There is, however lack of the resource check, determined by the algorithm specification.

Wait state is exactly the same, the robots don't move and send request to the environment. The difference is on the side of environment. In this case to change state to aggregation the found obstacle has to be the robot, the robot that also is in state of searching for another one. If this is true the robot requesting the information changes its state to *aggregation*. And environment informs the other robot via different channel that the pair was found and makes that robot to change state to *aggregation*.

The *aggregation* state is the final one where robot does not move and cannot change the state to different one.

8.3. Searching in Group

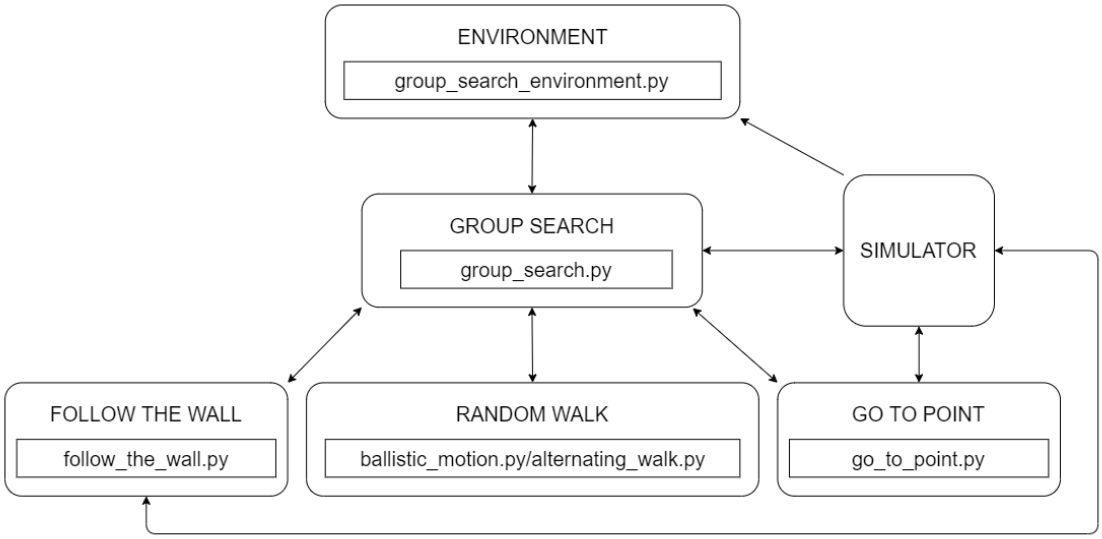


Figure 8.3. Search in Groups Node Structure

Searching in group is more complex algorithm from the previous ones, and the structure of nodes in this simulation is more extensive. The types of nodes and connections are visible in the **Fig 8.3**. There are five types of nodes:

- **Environment** - in general, the node is analogue to the environments in the previous scenarios. There is only one instance and its main tasks is to connect the robots and to simulate some events.
- **Group Search** - this is the basic algorithm node. There is one instance for every robot in the simulation. The algorithm that it executes is presented and described in **Section 5.2.4**.
- **Random Walk** - the node that executes the algorithm of random walk (**Section 5.1.1**). That node can be active only in case when the robot that uses it is the leader.

- **Go To Point** - the node executes part of the following algorithm (**Section 5.1.2**), it moves the robot towrd given point. That node can be active only in case when the robot that uses it is the follower.
- **Follow The Wall** - the node executes the second part of the following algorithm (**Section 5.1.2**). Is activated when the robot has to omit the obstacle and moves robot along the wall of the obstacle. That node can be active only in case when the robot that uses it is the follower.

Environment subscribes to */clock* to gather the data and that is the only connection with the simulator. The connection with group search node provides the service that answer the question whether the found resource is free or is already occupied by another pair of robots.

Group search publishes to the movement topic - */cmd_vel*, and it subscribes to */base_scan* and */odom* corresponding to its robot, and as the algorithm assumes that robots are in pairs, the node also subscribes to */odom* topic for robot that it is in pair with. The node has two services that are connected with a corresponding node of this type for its pair partner. The one that determines the leadership, and another one that informs partner that the resource was found. This node has also three services that activates the nodes below him: random walk, go to point and follow the wall.

All of the nodes that determines the movement of the robot: random walk, go to point and follow the wall, publishes data to */cmd_vel*. Follow the wall subscribes only to */base_scan* topic. Random walk is the same as in previous simulations so it uses both */base_scan* and */odom*. And go to point needs its partner position so it uses */odom* for its own robot and the partner one. Any of those three nodes do not posses service beside the activation one.

Listing 6. Group Searching Pseudocode

```
while True:
    if state == leadership_estimation:
        leadership_estimation()
        if leader == True:
            state = random_walk
        else:
            state = go_to_point
    elif state == go_to_point:
        resource_check()
        if on_way_obstacle_detected == True:
            state = follow_the_wall
        go_to_point()

```

```
    elif state == follow_the_wall:
        if on_way_obstacle_detected == False:
            state = go_to_point
        follow_the_wall()
    elif state == random_walk:
        if follower_away:
            state = wait_for_robot
        resource_check()
    elif state == waiting_for_robot:
        no_move()
        if follower_away == False:
            state = random_walk
    elif state == resource_found:
        no_move()
```

The Group Search, is significantly more complex than two previous algorithms. Not only in files connections but also as in code implementation. In this case there are six state: *leadership estimation, go to point, follow the wall, waiting for robot, random walk* and *resource found*. The whole operation is also considered to be infinite loop. We assume that at the beginning of simulation the robots are in pairs and and the pair is located close to each other.

The *leadership estimation* is short and simple state, the robots set which one of them is the leader and which one is the follower. In this case the robots slightly rotate by random angle and the one that rotated more is the leader. That state changes to *go_to_point* if robot is follower or to *random_walk* if it is a leader.

State *go to point* is simple walking toward the given position with usage of dedicated to it node. That position is the point to the left from the leader given by the title */odom* belonging to partner robot. In that state robots performs the resource_check, this is exactly the same approach as in the Resource Aggregation simulation. The robot checks its position, sends its state to the environment if it is close to resource, and the environment informs if it is free. In that case, however, the resource is either free or not. It cannot be occupied by one robot. After resource detection the robot informs its partner that the resource is found and both robots change their states to *resource_found*. In case when the laser sensor detects the object that blocks the passage to the point the state is changed to *follow_the_wall*. *Go to point* state is available only for follower.

Follow the wall state is performed by the dedicated to it node. It is simple algorithm, of movement close to the wall. That state changes back to *go_to_point* if there is no obstacle that blocks the passage to the destination point. *Follow the wall* state is available only for follower.

State *random walk* is similar to the previous examples. The robot moves randomly

in a way controlled by specific node. It performs resource_check and informs its partner if the resource is found and changes the state to *resource_found*. There is, however, synchronisation problem. When the partner from the pair is to far away from the destination point, the state is changed to *wait_for_robot* state. *Random walk* state is only available for leader.

Wait for robot state is just lack of movement, waiting for partner and changed back to *random_walk* state if partner is close to destination. *Wait for robot* state is available only for leader.

Resource found is finite state where robots do not move.

8.4. Transporting the Resource

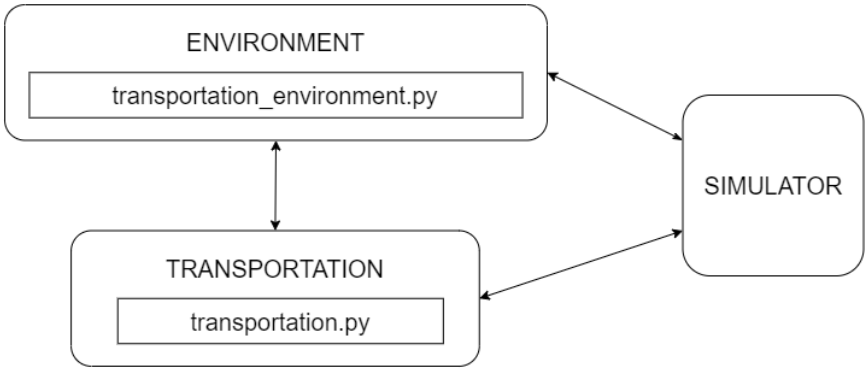


Figure 8.4. Transporting the Resource Node Structure

Transporting the Resource algorithm differs significantly from the previous ones. Due to the computational and software limits the algorithm is executed with usage of Gazebo [21] and the simulation takes under consideration the case with two robots and one box. The simulator is launched with usage of the launch file similar to the one presented in the beginning of the **Chapter 8**.

The node structure is presented in the **Fig 8.4**. There are only three nodes initialised where two are the twins that corresponds to the robots behaviour.

- **Environment** - environment, similarly to the previous algorithms implementations, is the support node that simulates the sensor readings to make the simulation more convenient to development.
- **Transportation** - that node provides the control to robots behaviour and performs the actions that ensures that box is moving.

Environment node subscribes to set of topics. The functional part is the subscription to */box/odom* that gives the information about the position of the resource. Based on that data the environment updates the desired direction of movement and uploads it to the robots by the topic */desired_yaw*. Other topics are */clock*, */robot_0/odom* and */robot_1/odom*

that gives information used to present the run of the task execution. The Environment node also has a connection to the transportation nodes. It informs the robots about the start of the tasks to synchronise them.

Transportation nodes subscribe to their own odometry to know the direction of the robot movement. Their also subscribe to */box/odom* to get the information about the resource position. The last of the topics that is subscribed by this node is the */robot_x/scan*, where *x* is the number of robot. That topic provides information from the laser sensors that are needed while executing the task of transporatation.

Listing 7. Transporting the Resource Pseudocode

```
while True:
    if state == maintaining_box_direction:
        if robot_aligned_to_the_box == False:
            state = aligning_pusher_to_the_box
        elif box_direction_correct == True:
            state = maintaining_box_velocity
        else:
            if is_ahead() == True:
                no_move()
            else:
                move_forward()
    elif state == maintaining_pusher_direction:
        if robot_aligned_to_the_box == False:
            state = aligning_pusher_to_the_box
        elif robot_direction_correct == True:
            if box_direction_correct == False:
                state = maintaining_box_direction
            else:
                state = maintaining_box_velocity
        else:
            turn()
    elif state == maintaining_box_velocity:
        if robot_aligned_to_the_box == False:
            state = aligning_pusher_to_the_box
        elif robot_direction_correct == False:
            state = maintaining_pusher_direction
        elif box_direction_correct == False:
            state = maintaining_box_direction
        else:
            move_forward()
```

```
    elif state == aligning_pusher_to_the_box:
        if robot_aligned_to_the_box == True:
            state = maintaining_box_direction
        else:
            align_to_the_box()
```

Presented above pseudocode describes in more details the algorithm presented in **Section 5.2.5**. It is performed in infinite loop, and the states corresponds to specific behaviour.

Maintaining Box Direction at the beginning checks whether there is need to change the box direction and if the robot is aligned to the box. If the box is beyond the robots range the state is changed to aligning_pusher_to_the_box and if the heading of the box points toward a goal direction the state is changed to maintaining_box_velocity. If none of those happens the robot determines if the box should be turned toward him or from him. The choice whether the robot turns is made based on the function is_ahead.

Maintain Pusher Direction also begins with checking if there is need to perform the turn. Firstly the alignment of robot to the box is checked. If the robot is not aligned the state is changed to aligning_pusher_to_the_box. Later the heading of the robot is checked. If the robot is not facing the goal direction the robots turns, otherwise the heading of the box is checked. If it points toward the goal direction the state is changed to maintaining_box_velocity otherwise to the maintaining_box_direciton.

Maintaining box velocity is also similar to the previous states. It firstly checks the box alignment, subsequently the robot heading direction and at the end the box heading direction. If there is something to fix the state is changed. In the case when there is nothing to adjust the robots just go forward. Aligning Pusher to the Box is the last behaviour of the robot. At first there is checked whether the robot fixed the problem with alignment. If yes then the state is changed to the maintaining_box_direciton otherwise the robot moves to the corner of the box that it detected with usage of its sensors.

The code of the whole project is available on the GitHub:
`https://github.com/lewiktorbara/Swarm_Robotics.git`

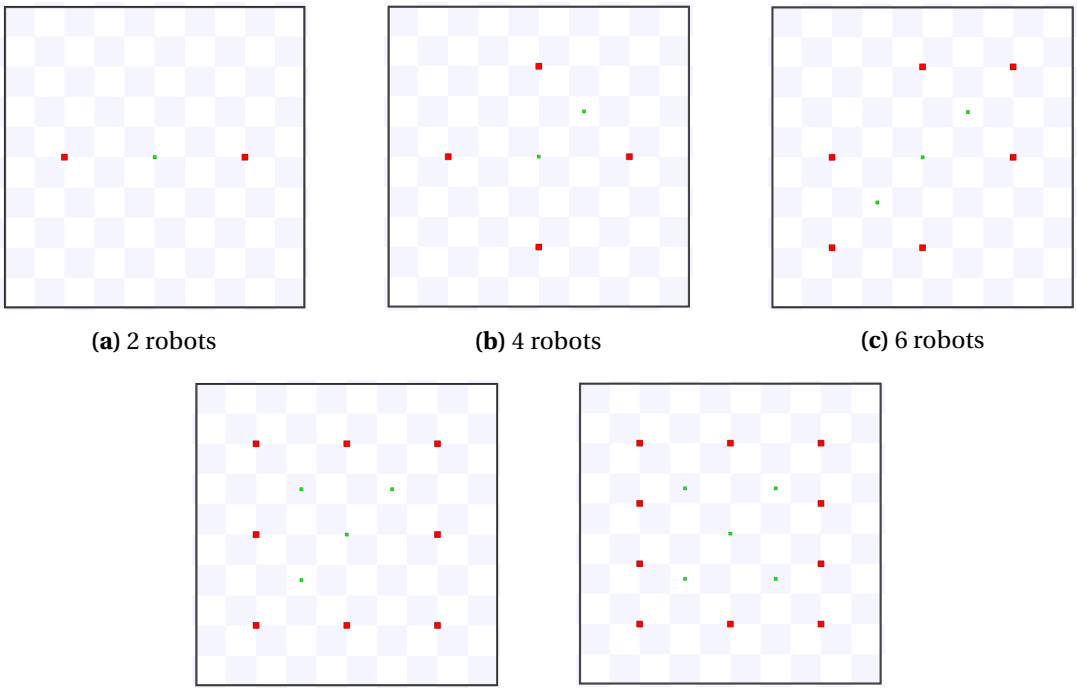
9. Experiments

In this section the data that was gathered during simulations is presented and discussed. The graphs in the following subsections shows the differences in cases what type of random walk was used and how different number of robots scattered over some area affects the time of the task. There are also taken under consideration the two approaches, that were designed and discussed earlier in this work. There is a comparison between them and the outcome of that comparison is presented.

In the first three subsections every type of simulation is analysed separately. The fourth subsection (**Section 9.4**) compares the aggregation algorithms and the fifth (**Section 9.5**), presents the transportation simulation.

9.1. Resource Aggregation

The first simulation type that is discussed is Resource Aggregation. Under consideration are taken several scenarios. Every simulation is performed on the world of size 10x10, however, the number of robots changes. That provides different density of the robots and the outcome of the experiment can depend on that variable. The number of robots that are considered are $\{2, 4, 6, 8, 10\}$. The number of resources also changes, but the ratio between number of robots and number of resources stay the same. There is always one resource for each pair of robots.

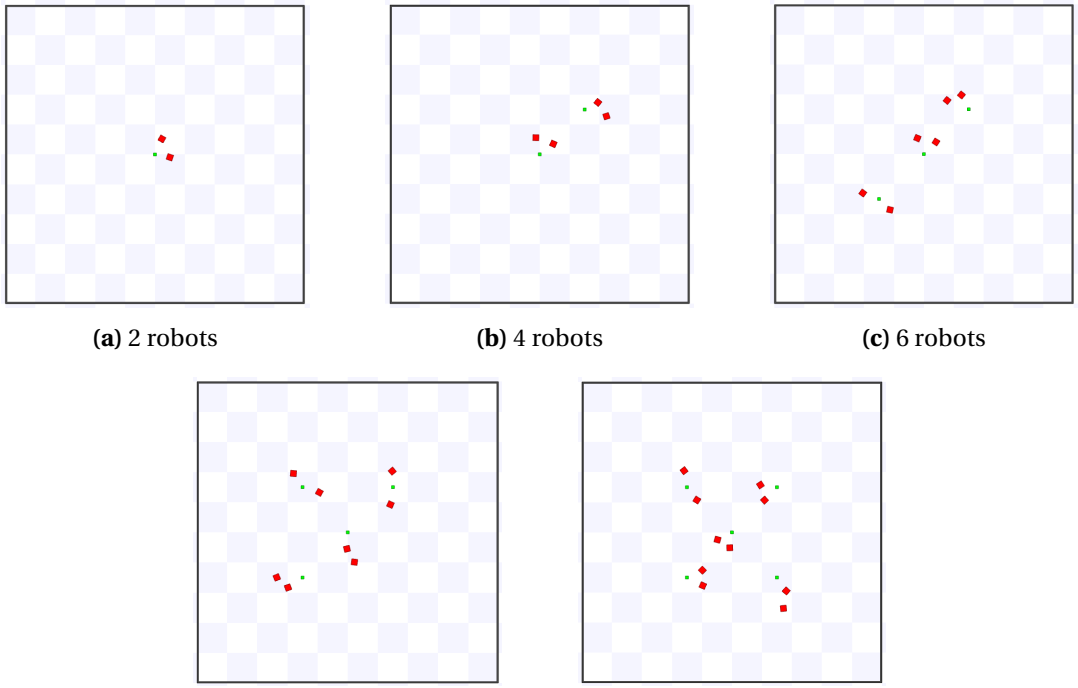


(a) 2 robots **(b)** 4 robots **(c)** 6 robots

(d) 8 robots **(e)** 10 robots

Figure 9.1. Worlds visualisation for Resource Aggregation simulation

In the **Fig 9.1** are presented the visualizations of those worlds at the beginning of the task with respect to number of robots. The red squares represent the robots while the green represent resources. In the **Fig 9.2**, on the other hand, we can observe the examples of successful runs of the Resource Aggregation Algorithm.



(a) 2 robots **(b)** 4 robots **(c)** 6 robots

(d) 8 robots **(e)** 10 robots

Figure 9.2. Worlds visualisation for Resource Aggregation simulation after successful execution of task

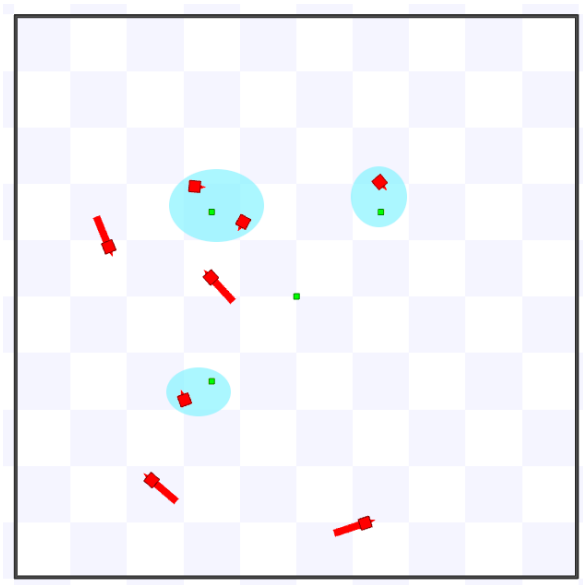


Figure 9.3. Visualization of world during execution of task of Resource Aggregation

In the **Fig 9.3** we can observe the task of Resource Aggregation during the execution. The red line after the robot visualize the movement of the robot, it shows the previous positions of the unit. As we can see there is already one pair of robots that found the resource and two single robots that are still waiting for a partner.

Another aspect that creates different scenarios, despite the difference in number of robots and resources, is the type of random walk algorithm that is used. There are two of them, presented and described in **Section 5.1.1**, *Ballistic Motion* and *Alternating Walk*. As the one of them is specialized in coverage (*Ballistic Motion*) and the other one in intensification (*Alternating Walk*) the results can differ significantly basing on type of random walk used.

Summing up, there are 10 different types of simulations in case of Resource Aggregation. Each type were run 30 times and the outcome of them can be visible in the **Fig 9.4**.

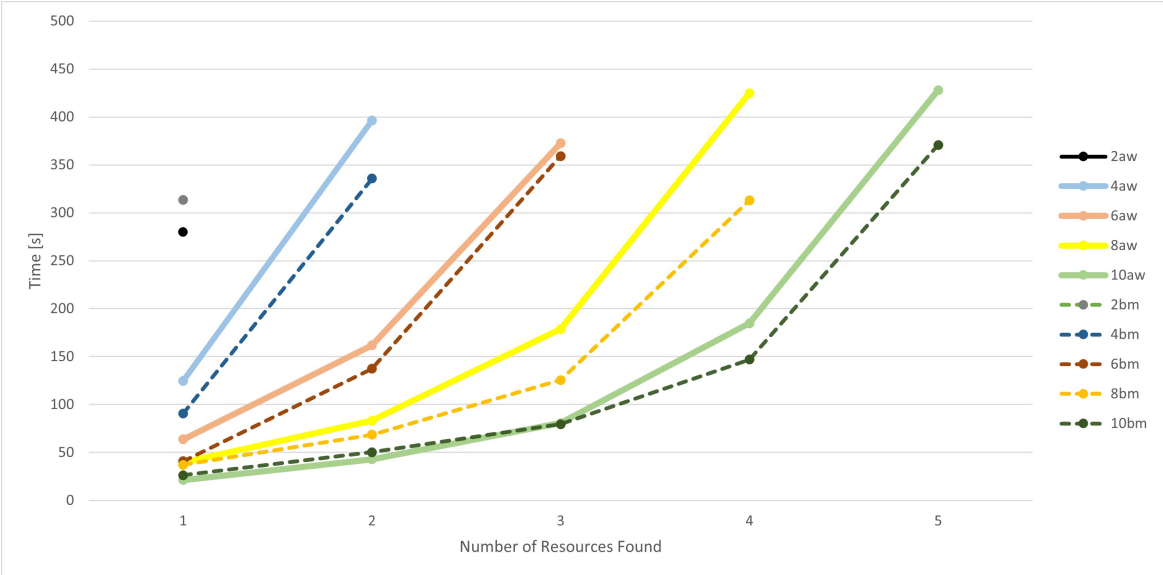


Figure 9.4. Resource Aggregation comparison

The points marked on the graph represent the average time that the given number of resources was found. The number of resources that can be found, and the number of measurements, depends on the number of robots in simulation. The solid lines corresponds to the usage of *Alternating Walk* and is described as **aw** while the dotted ones represents *Ballistic Motion* and is described as **bm**. In both cases the description is preceded by the number corresponding to number of robots used in simulation.

As we can see the data gathered is quite similar to each other and efficiency in overall is affected just slightly by the change of number of robots or type of random walk movement. Despite the fact that in general the measurement are very similar, the differences exist.

In almost all cases the type of random walk that seems more convenient for this task is *Ballistic Motion*. The time to find the resource, in most of scenarios is smaller in case of that movement algorithm. That difference tends to growth in case of bigger number of robots and finding last resources on the arena, while the difference hardly exist when taken under consideration are the firstly found resources. The only case where the time of finding for *Alternating Walk* is visibly smaller that the time for *Ballistic Motion* is the scenario with two robots and one resource. That outcome is clearly achieved by the type of specification of random walk type. The *Ballistic Motion* specializes in coverage while the *Alternating Walk* in intensification. That characteristic is clearly visible in case of the last found resource, where the *Ballistic Motion* moves on the whole arena, while the *Alternating Walk* tends to close itself in smaller parts of that area additionally blocked by the obstacles that are robots that found the resource.

The difference between those two random walk algorithms, can also be observed in the **Fig 9.5**.

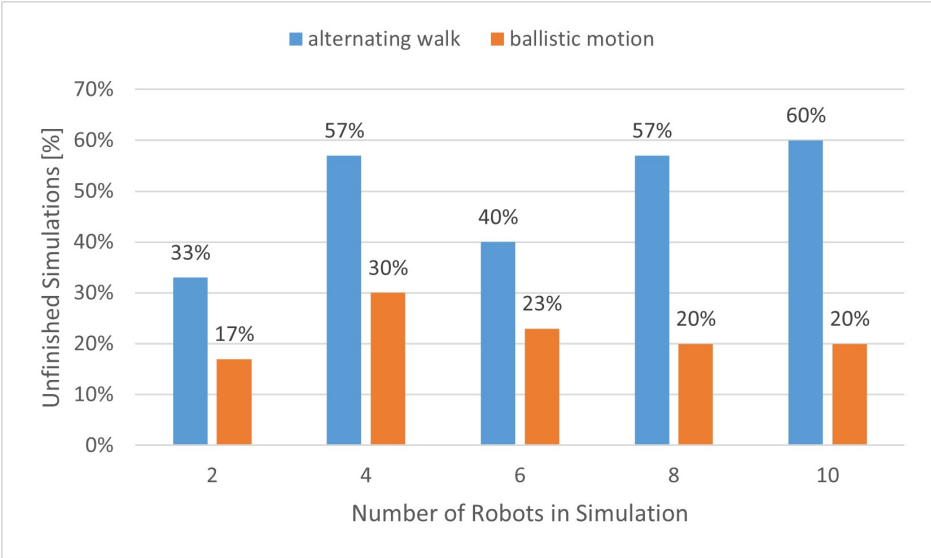


Figure 9.5. Unfinished Resource Aggregation

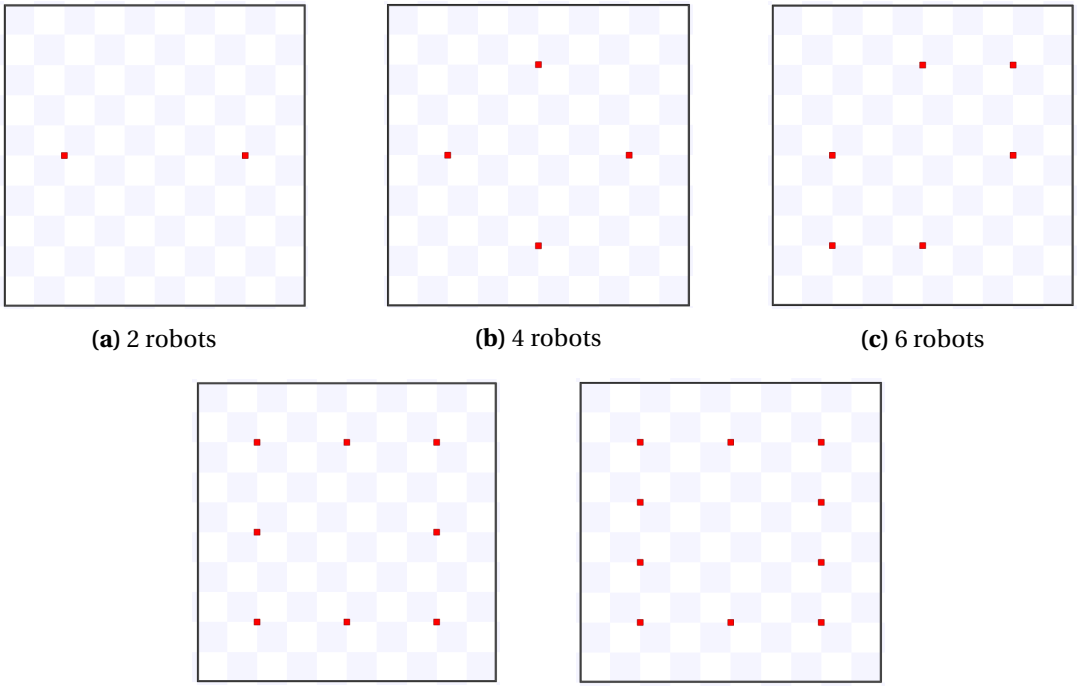
The simulations have the cap for the time spent on each simulation. After 500 seconds the simulation ends and saves that time to the resource that was not found yet. That graph presents percentage of unfinished resource findings for the last resource. That fact lowers the average time of that finding, presented in the **Fig 9.4**. As we can see, the *Ballistic Motion* looks much better that the other algorithm. In some cases *Alternating Walk* reaches 60%, so the actual average time for finding fifth resource in ten robots scenario can be much more higher.

Despite the fact that the type of random walk affects the outcome, the density of robots in the arena does not change that outcome so much. The average efficiency and the percentage of the robots that found the resource in given time is quite similar for all

simulations. We can assume that the change of density does not affect that tasks in the significant way.

9.2. Group Aggregation

In this subsection the Group Aggregation is taken under consideration. As in previous example of simulation there are 5 types of worlds with respectively $\{2, 4, 6, 8, 10\}$ number of robots on the arena 10x10. In that case, however, the robots search for each other so there is no resource on the arena. In the **Fig 9.6** are visible representations of those worlds.



(a) 2 robots **(b)** 4 robots **(c)** 6 robots

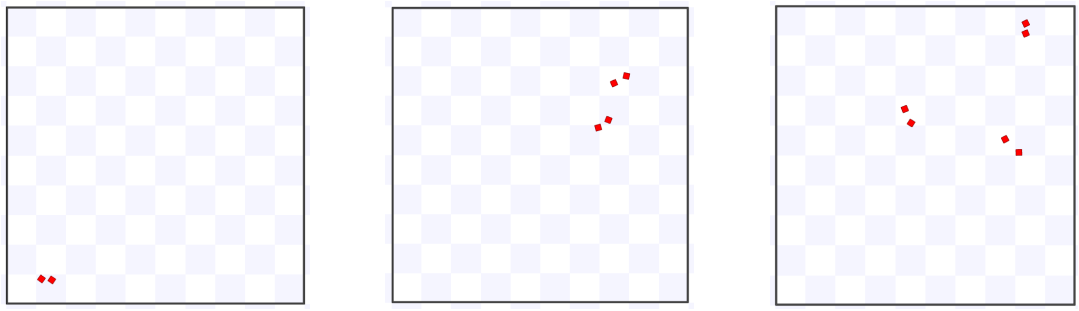
(d) 8 robots **(e)** 10 robots

Figure 9.6. Worlds visualisation for Group Aggregation simulation

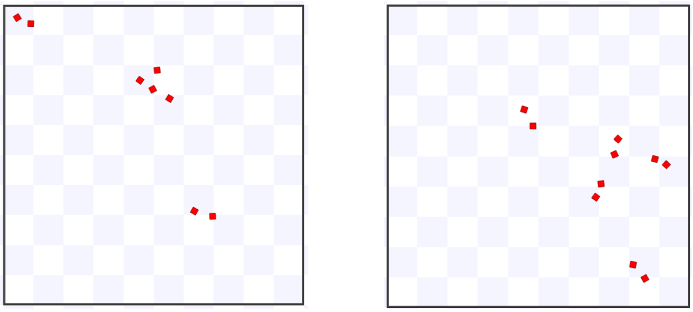
In the **Fig 9.7**, as in previous algorithm, we can see examples of finished task of group aggregation. And in the **Fig 9.8** is presented the world during the execution of the algorithm. There are marked already created pairs, we can also observe the movement of other four robots that still search for a pair. That movement can be visible thanks to the red lines that shows the previous positions of the robots.

In that case also the two random walk algorithms were used, *Ballistic Motion* and *Alternating Walk*. There were performed 10 types of simulations and every one of them was performed 30 times. The results are visible in the **Fig 9.9**.

As in case of Resource Aggregation there can be clearly visible the difference between the types of random walk. The *Ballistic Motion* also seems much better than *Alternating Walk*. The difference is visible especially in examples with 2, 6 and 8 robots. In the case



(a) 2 robots **(b)** 4 robots **(c)** 6 robots



(d) 8 robots **(e)** 10 robots

Figure 9.7. Worlds visualisation for Group Aggregation simulation after successful execution of task

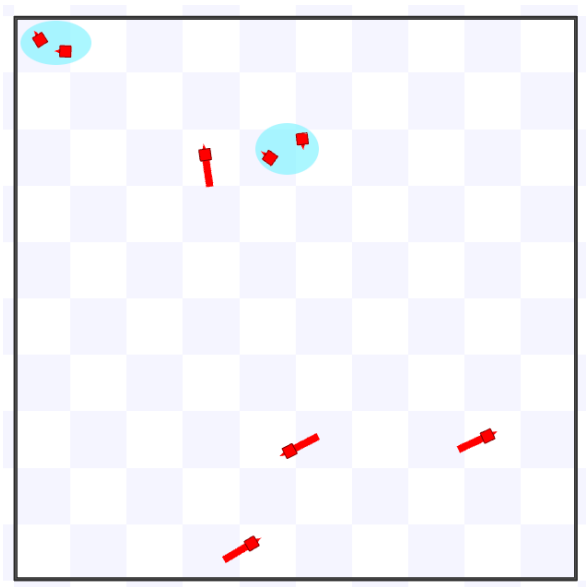


Figure 9.8. Visualization of world during execution of task of Group Aggregation

of the 2 robots the superiority of the *Ballistic Motion* is overwhelming. The time is over 100% smaller than in the *Alternating Walk* case. That is caused by the searching the robot in the relatively small part of the arena and the robots does not meet each other, while in the *Ballistic Motion* usage scenario, robots move around whole arena and the probability that they find each other is significantly larger. The 6 and 8 robots scenarios

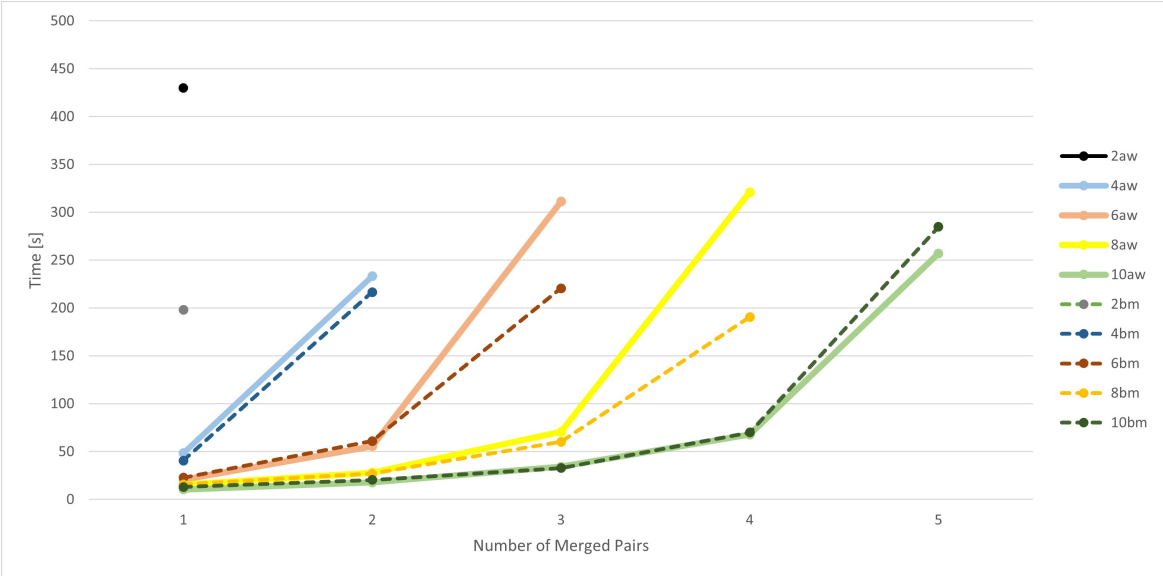


Figure 9.9. Group Aggregation comparison

also shows that *Ballistic Motion* is better choice. The difference is not so big like in 2 robots example, and that difference is visible only in case of last pair merging. The scenario with 4 robots is almost the same in both random walk algorithms, but the case with 10 shows that the *Alternating Walk* is slightly better in examples with high density of robots. That fact, however, does not affect the conclusion, that significantly better choice is to include *Ballistic Motion* in the Group Aggregation Algorithm. That thesis is also confirmed by the percentage of unfinished pair merging, visible in the **Fig 9.10**.

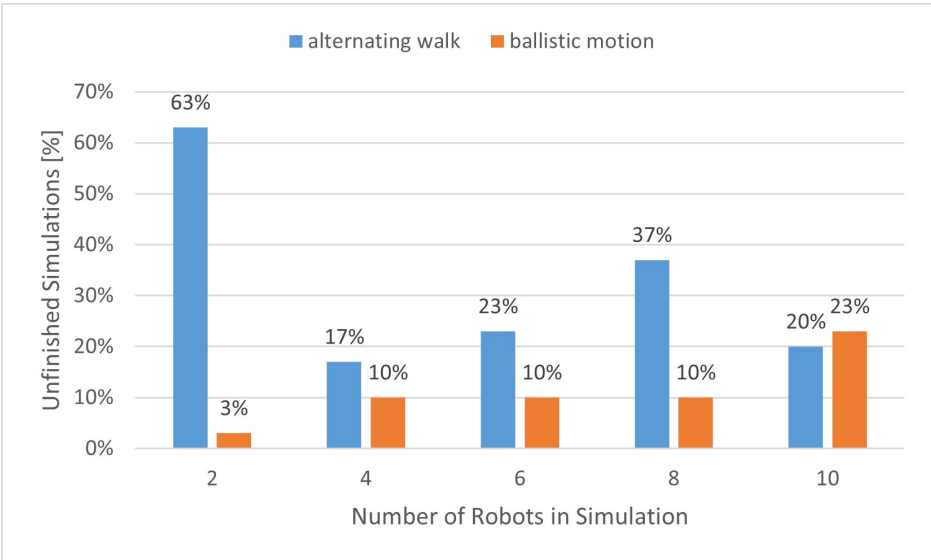


Figure 9.10. Unfinished Group Aggregation

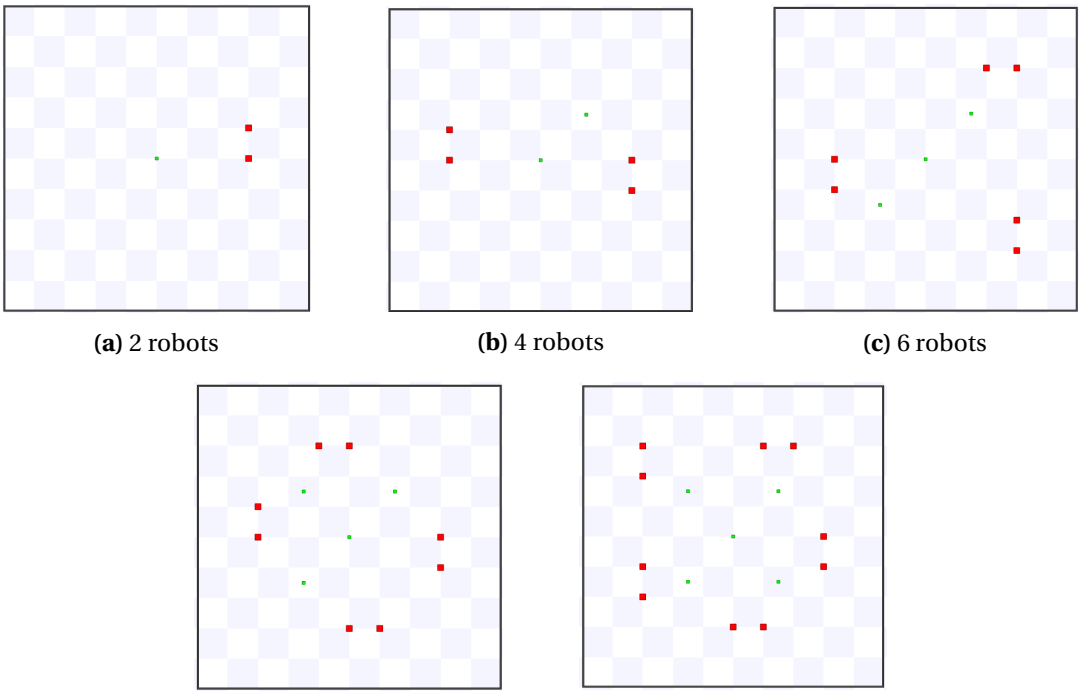
The superiority of *Ballistic Motion* is obvious and the only example where those two algorithm are on the same level is 10 robots scenario, that is not enough to not exclude

Alternating Walk from this task.

The change of density in that type of simulation, as in the scenario presented in **Section 9.1**, does not affect in the significant way the results of the tasks. In general the robots merge in similar times and the efficiency is almost the same. The only example that differs from tendency is the big tensity case with 10 robots. We can observe that in that case, the last merging takes little longer time than the other scenarios. We can form, basic on this, conclusion that this task performs better in **not** dense examples.

9.3. Group Search

The final simulation that refers to the aggregation algorithm is Group Search. In this case also the scenarios with $\{2,4,6,8,10\}$ number of robots are taken under consideration. The robots, however, are placed close to its partners so we can see several aggregates of two robots. The target of this simulation, similarly to the Resource Aggregation, is to find the resources. The number of resources also corresponds to the number of pairs in the simulation. The appearance of worlds is visible in the **Fig 9.11**.



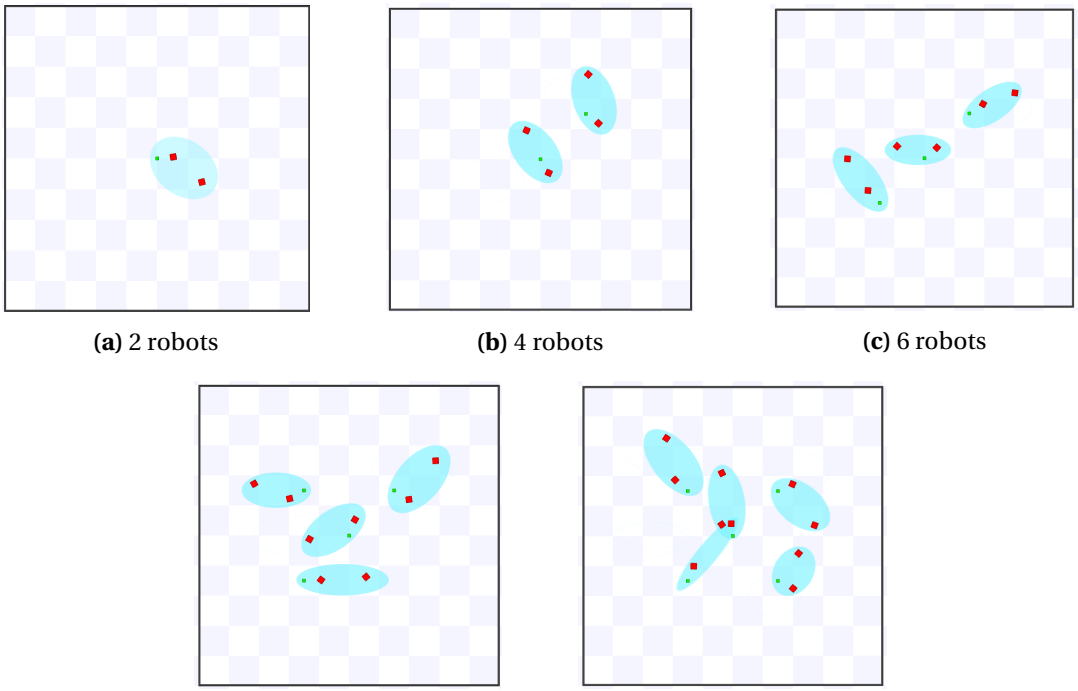
(a) 2 robots **(b)** 4 robots **(c)** 6 robots

(d) 8 robots **(e)** 10 robots

Figure 9.11. Worlds visualisation for Group Search simulation

In the **Fig 9.12** there are presented the examples for the finished task. To make it more visible and clear there are marked pairs of robots and the resources that were found by them. Next in the **Fig 9.13** we can observe the task of Group Search during the execution.

As we can see, till the time of observation only one pair of robots found the resource, while the other ones still search.



(a) 2 robots **(b)** 4 robots **(c)** 6 robots

(d) 8 robots **(e)** 10 robots

Figure 9.12. Worlds visualisation for Group Search simulation after successful execution of task

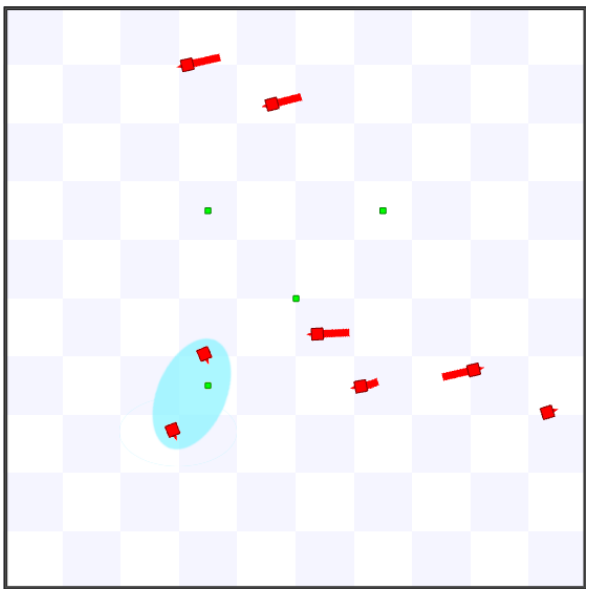


Figure 9.13. Visualization of world during execution of task of Group Search

That simulation type also compares the random walk types. The every type of simulation is carried out 30 times and the number of types, as in previous examples is 10. The data, gathered during those experiments is presented in the **Fig 9.14**.

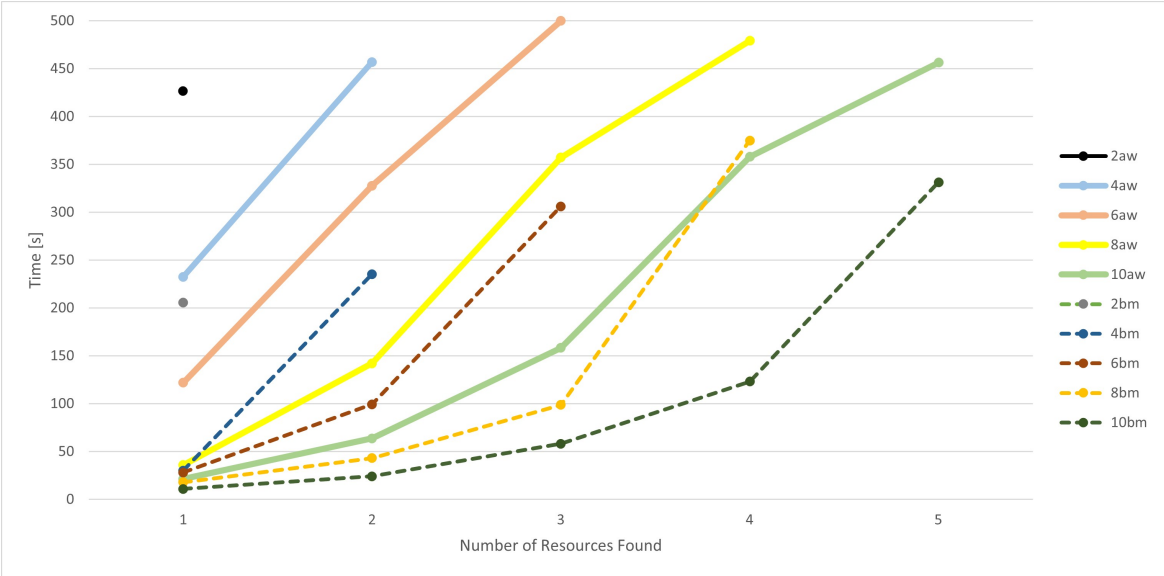


Figure 9.14. Group Search comparison

In the case of Group Search, there is clearly visible that each type of the simulation has different outcome. Let's first focus on the type of the random walk. The superiority of the *Ballistic Motion* is indisputable and much more obvious than in Group Aggregation scenario. For every number of robots the time in case of *Alternating Walk* is overwhelmingly longer. The cause of this is the number of interactions between robots. The pair of robot has to synchronise and the frequent change of the direction movement makes them to spend more time on this issue. This and the fact that there is increasing number of the interactions between robots cause to even increase that time in case of the increased density. The amount of unfinished resource search is huge, and visible in the **Fig 9.15**. Even for the *Ballistic Motion*, some values oscillate around the 50%, when the examples with *Alternating Walk* reach even 100% in scenario with 6 robots. What is completely unacceptable and the *Alternating Walk* **cannot** be use in this task.

Unlikely to previous simulations the number of robots, and density of them in the arena also affects the efficiency in significant way. The time is visibly increased in the growth of number of robots and in the scenario with 8 robots does not differs much regardless of used random walk algorithm. Group Search is much better for simulations with lower density and only with usage of the *Ballistic Motion*.

9.4. Algorithms Comparison

This section takes under consideration two algorithms, described in **Section 5**, concerning the aggregation issue. Based on the previous sections, both algorithms are performed with usage of *Ballistic Motion* as the random walk, due to the fact that this random walk algorithm performed significantly better.

In the **Fig 9.16** we can see the differences time of searching the resource using both

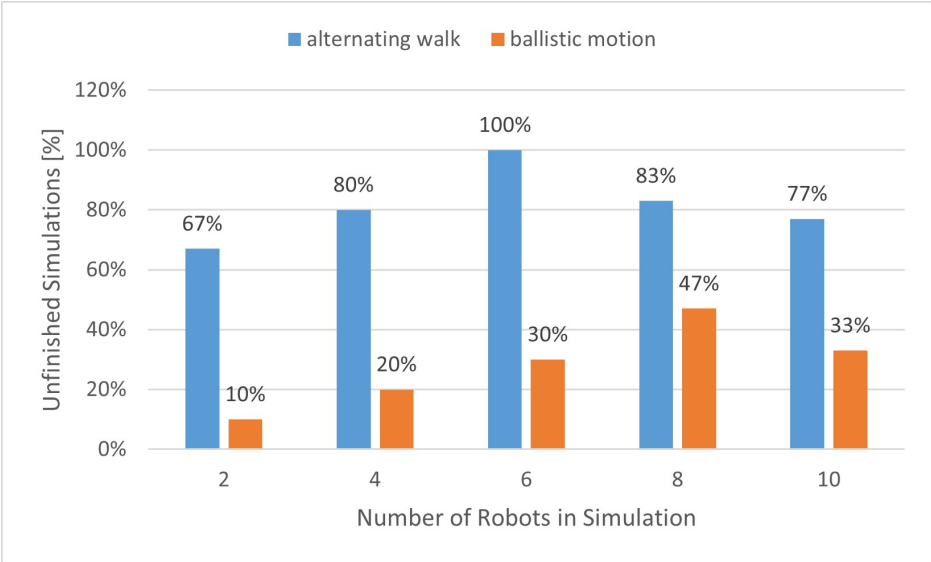


Figure 9.15. Unfinished Group Search

FormTeam - Search algorithm (marked as the solid lines) and the Search - FormTeam algorithm (marked as dotted lines). As we can see, in case of the finding the first resource the difference is almost zero, exception is the scenario with two robots where difference is almost 100 seconds. That little difference in case of other scenarios is growing in significant way while considering further resources. In case of the last resources that difference is enormous, especially in case of the scenarios with greater number of robots. In this statement algorithm that firstly searches for resources and then groups with other robots turns out to be much better one.

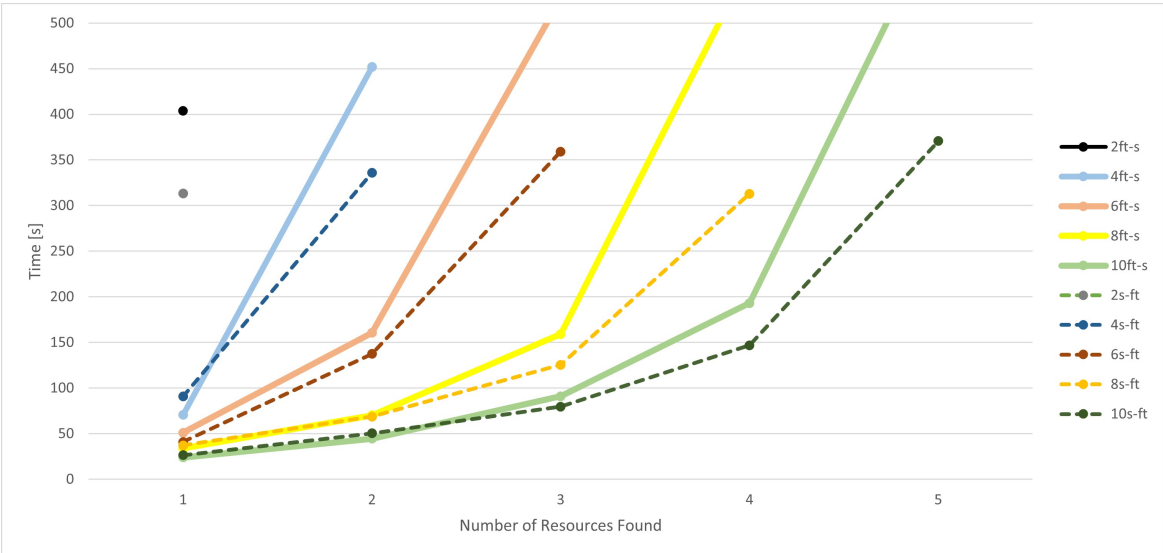


Figure 9.16. FormTeam - Search [ft-s] vs Search - FormTeam [s-ft] comparison

In the figure below (**Fig 9.17**) comparison between the resource searching part of the algorithms is presented.

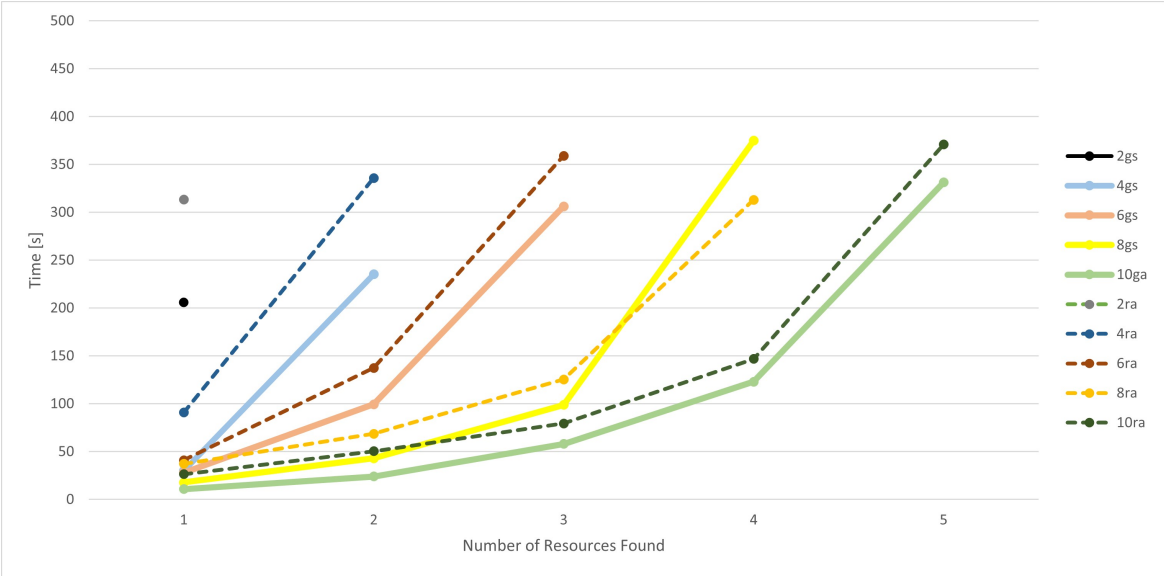


Figure 9.17. Group Search [gs] vs Resource Aggregation [ra] comparison

As can be observed the Group Search in most scenarios performed better. The only example where Resource Aggregation achieved shorter time, was in finding the last resource in scenario with 8 robots. The Group Search algorithm works faster, because the robots do not have to rely on the other one that has to find the resource, but with wider sensing range they find the resource quicker and the resource is found by two robots in one time, there is no need for single robot to wait. The disadvantages resulting from synchronization issues, takes less time that this algorithm gains because of its characteristic.

The main conclusions to the graphs is that the **Group Search algorithm is the better one** in case of using only the searching part of the task. We have to, however, take under consideration the fact that those two algorithms, presented in the **Fig 9.17**, are the small parts of larger whole. The **Fig 9.16** shows indisputably **that the algorithm where the robots prioritize finding the resource and then merge into a pairs is the superior one**. We can consider the other algorithm with searching the resource in the formation only in case where certain assumptions are considered. The robots have to start the task being already merged in pairs. That assumption is not problematic in most of cases and can open us the way to more efficient approach of finding the resource.

9.5. Transportation

The simulation that performs the tasks of collective transportation of resource is executed with usage of Gazebo [21] simulator. In the **Fig 9.18** we can observe the overall run of that simulation. On the left side we can observe the beginning of the transportation task, while on the right is visible the result after the turn of the box. As we can see, the robots were able to perform full 90° turn without communication and any central control.

They were basing only on the laser sensors and the effectors and receptors simulated by the environment.

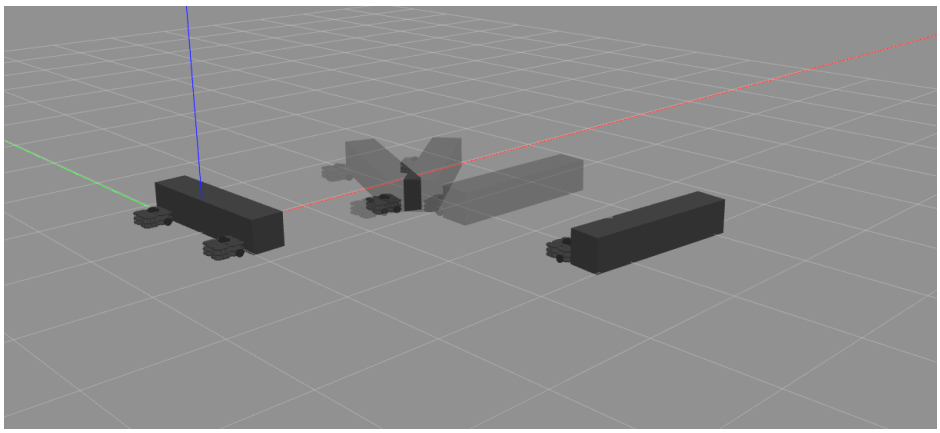


Figure 9.18. Visualisation of Transportation task

In the **Fig 9.19** there are visible the headings of the robot and the box. There is also presented the desired direction of movement. We can see that after the change in the desired heading of the box the robots started to align pushed object to given trajectory. The right robot was visible a bit late in case of its heading. That's because of the prioritizing the aligning robot to the box over maintaining robot direction in respect to the desired one. At the end of the movement we can also observe the little fluctuations of the headings of both robots and box in respect to desired one. After that fluctuation the movement stabilizes and the resource is transported along the given trajectory.

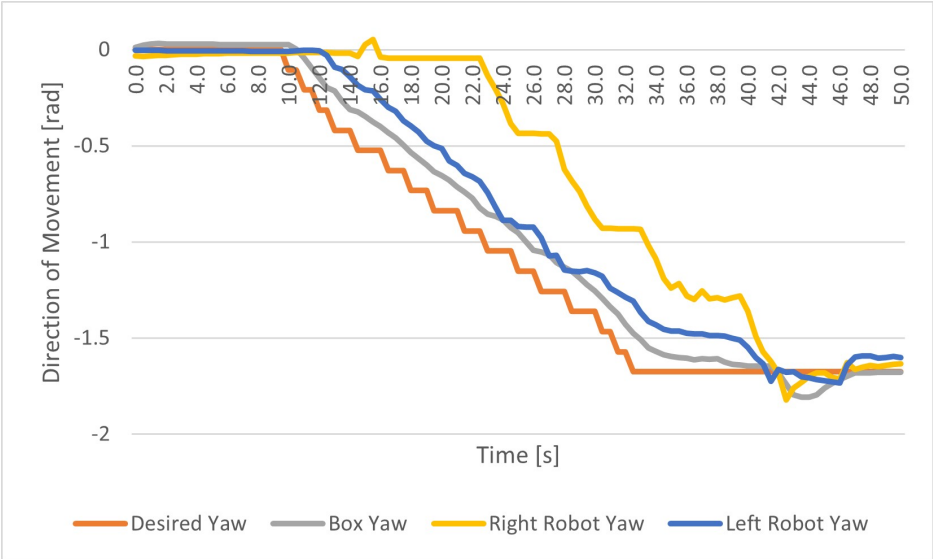


Figure 9.19. Trajectory of Transportation

10. Summary

This chapter sums up the information presented in whole the thesis. The most important facts that were introduced and researched, the conclusions that were made basic on gathered data. It also concludes project as a whole and shows that end of this work is not the end of the work in this field.

10.1. Achieved Goals

This thesis as a whole answers several questions asked in the **Chapter 1**. It also presents the process of development of the system that provides us information and data, that turned out to be useful in our research.

The system creation and implementation is one of the most crutial and meaningful aspects of the work. That system allowed us to perform several types of simulations. Another aspect, without which the experiments won't be possible, is development of the algorithms responsible for executing the task presented at the beginning of the work. The task of collaboratively transportation of resources by swarm of robots. The algorithms that was developed and possible to run are: *Resource Aggregation, Group Aggregation, Group Search* and *Transporting the Resource.*

The information in previous chapters showed us also that the comparison between two different approaches of solving the task answered the question which one of them is, objectively, the better one. Those two approaches concerned the aggregation part of the task. The first one prioritize the grouping with other robots and then collective search of resource, while the other one assumed that the resource search is more important. In the next section, the results of those comparison and system development are concluded.

10.2. Conclusions

After performing the experiments in **Section 9** the problem, set at the beginning of the work, was solved and now it gives us several answers. Therefore, we can make the following conclusions. The algorithms differ in the assumptions and their difference is also visible in performance. Despite the fact that the Group Search appeared as the better algorithm to search resources, it is the only part of algorithm with earlier merging into pairs. That fact makes it impossible for us to state that this algorithm is the more efficient one. The algorithm Search-FormTeam described in **Section 5.3.1** performs in more efficient way as a whole, from the point where single robots are scattered over some area. However, there is possibility to assume that the robots are in pairs at the beginning and those pairs are scattered over some area. In that case the Group Search algorithm looks like much better choice. The Group Search algorithm is, on the other hand, much more complex than the Resource Aggregation and in case of the swarm robotics, the simpler algorithms are often the best choice. The swarm relies on the simplicity of the individuals.

To make it clear and unambiguous, in most of situations the choice of the Search - FormTeam (Resource Aggregation) algorithm is the better one, due to the main motivations of swarm robotics. That algorithm is more universal one, the less complex is the approach the better due to its scalability, flexibility and robustness. That algorithm is in simple words: usually better.

10.3. Future Works

Despite the fact that the answer for the question about quality of the algorithms used has been given, the several enhancements can be implemented to develop the research.

In this work the hybrid approach was not taken under consideration, what can be considered. The robot prioritizes the algorithm that can be applied quicker to its run. For example the robots walk randomly and find the other one that also searches. They merge and start executing the algorithm that was used in that scenario was the FormTeam - Search (**Section 5.2.4**). In the case when the robot finds the free resource it uses the other algorithm. That brings new solutions for the task of searching and transportation, and can be considered in future works.

In this work, due to simplicity the algorithms were divided into separate cases and simulations. To make the research more accurate there can be implemented solution that simulates whole approach and considers more examples. That can give new data that will widen the knowledge about that issue.

There also can be used a system equipped with real robots and the tasks can be performed in real environment. The simplification that assumes simulating effectors and receptors to increase the efficiency can also be removed.

That project has potential to be developed and extended. It can give us new perspective on the case of aggregation, transportation, the whole swarm robotics.

References

[1] G. Dudek, M.R.M. Jenkin and E. Milios, "A taxonomy for multi-agent robotics", *Autonomous Robots*, vol. 3, pp. 375–397, 1996.

[2] E. Şahin, "Swarm Robotics: From Sources of Inspirations to Domains of Application", *Swarm robotics, Springer*, vol. 3342, 2005.

[3] H.-N. Chu, J. Arlat, M.-O. Killijian, B. Lussier, and D. Powell, "Robustness testing of robot controller software", May 2009.

[4] L. Bayindir, "A Review of Swarm Robotics Tasks", *Neurocomputing*, vol. 172, pp. 292–321, 2016.

[5] A. Manjunath and V. Kishore, "The Promising Future in Medicine: Nanorobots.", *Biomedical Science and Engineering*, vol. 2, no. 2, pp. 42–47, 2014.

[6] "Starlink", `https://www.spacex.com/updates/starlink-mission-04-07-2021/index.html`.

[7] M. Zastrow, "Researchers create 1,000-robot swarm", *Nature*, Aug. 2014.

[8] "Robot operating system", `https://www.ros.org/about-ros/`.

[9] C. Zieliński, T. Kornuta, "Robot Control System Design Exemplified by Multi-Camera Visual Servoing", *Journal of Intelligent & Robotic Systems*, pp. 499–523, 2015.

[10] C. Zieliński, M. Figat & R. Hexel, "Communication Within Multi-FSM Based Robotic Systems", *J Intelligent & Robotic Systems*, vol. 93, no. 3, pp. 787–805, 2019.

[11] M. Figat and C. Zieliński, "Robotic System Specification Methodology Based on Hierarchical Petri Nets", *IEEE Access*, vol. 8, pp. 71 617–71 627, 2020.

[12] M. Figat and C.Zieliński, "Automatic Robotic System Controller Code Generation Utilising Robotic System Specification Language", *International Conference on Robotics and Automation (ICRA)*, pp. 3363–3369, 2019.

[13] M. Figat, C. Zieliński and R. Hexel, "Robotic Systems Implementation Based on FSM's", *Advances in Intelligent Systems and Computing*, vol. 743, 2018.

[14] M. Figat, C. Zieliński and R. Hexel, "FSM Based Specification of Robot Control System Activities", *11th International Workshop on Robot Motion and Control (RoMoCo)*, pp. 193–198, 2017.

[15] C. Dimidov, G. Oriolo and V. Trianni, "Random Walks in Swarm Robotics: An Experiment with Kilobots", *Swarm Intelligence, Springer*, vol. 9882, pp. 185–196, 2016.

[16] M. Kegeleirs, D. Garzón Ramos and M. Birattari, "Random Walk Exploration for Swarm Mapping", *Towards Autonomous Robotic Systems (TAROS)*, vol. 11650, 2019.

[17] K.N. McGuire, G.C.H.E. de Croon and K. Tuyls, "A Comparative Study of Bug Algorithms for Robot Navigation", *Robotics and Autonomous Systems*, vol. 121, 2019.

[18] "An introductory robot programming tutorial", `https://www.toptal.com/robotics/programming-a-robot-an-introductory-tutorial?fbclid=IwAR3c0D1Hn30ZpYmAa7VoWe768e9kaSXwyEuS23X-cA-HD-AbEfE98A3QBRQ`.

[19] G. Valentini, “How robots in large group make decisions as a whole? From biological inspiration to the design of distributed algorithms”, *ArXviz*, vol. abs/1910.11262, 2019.

[20] Piotr Trojanek, Cezary Zieliński, Wojciech Szynkiewicz, “Definition and Composition of Individual Robot Behaviuors in Cooperative Box Pushing”, *Proceedings of the 13th IEEE IFAC International Conference on Methods and Models in Automation and Robotics (on CD) Technical University of Szczecin*, 2007.

[21] “Gazebo”, `http://gazebosim.org`.

[22] “Ros stage”, `http://wiki.ros.org/stage`.

[23] “Rospy”, `http://wiki.ros.org/rospy`.

List of Figures

2.1 Internal cutaway view of respirocyte - equatorial view [5] 17
2.2 An artificial red cell - the respirocyte designed by Robert A.Freitas Jr [5] 17
2.3 Starlink 18
2.4 1000-kilobots 18
4.1 Interaction between agent and its environment [9] 21
4.2 General structure of Embodied Agent a_j [12] 22
4.3 Activities of the control subsystem represented by an FSM ${}^{s}F_{core}$ [14] 24
4.4 ${}^{s}F^{B}_{j,v\omega}$ specifying the behaviour ${}^{s}B_{j,v,\omega}$ [10] 24
5.1 Alternating Walk 26
5.2 Ballistic Motion 26
5.3 Following [18] 27
5.4 Searching for Resource Algorithm 28
5.5 Wait Algorithm 29
5.6 Group Aggregation 29
5.7 Group Searching 30
5.8 Transportation 31
5.9 Search - FormTeam Algorithm 32
5.10 FormTeam - Search Algorithm 33
6.1 Turtlebot3 Waffle Pi (simulated parts in grey font) 34
6.2 Comparison between Unicycle Model and Differential Model [18] 35
6.3 Structure of the agent a_j 36
7.1 Two robots world 10x10 41
7.2 Gazebo World with Turtlebot3_Waffle_PI 42
8.1 Resource Aggregation Nodes Structure 44
8.2 Group Aggregation Nodes Structure 46
8.3 Search in Groups Node Structure 48
8.4 Transporting the Resource Node Structure 51
9.1 Worlds visualisation for Resource Aggregation simulation 54
9.2 Worlds visualisation for Resource Aggregation simulation after successful execution of task 55
9.3 Visualization of world during execution of task of Resource Aggregation 55
9.4 Resource Aggregation comparison 56
9.5 Unfinished Resource Aggregation 57
9.6 Worlds visualisation for Group Aggregation simulation 58
9.7 Worlds visualisation for Group Aggregation simulation after successful execution of task 59
9.8 Visualization of world during execution of task of Group Aggregation 59
9.9 Group Aggregation comparison 60

9.10 Unfinished Group Aggregation 60
9.11 Worlds visualisation for Group Search simulation 61
9.12 Worlds visualisation for Group Search simulation after successful execution of task 62
9.13 Visualization of world during execution of task of Group Search 62
9.14 Group Search comparison 63
9.15 Unfinished Group Search 64
9.16 FormTeam - Search [ft-s] vs Search - FormTeam [s-ft] comparison 64
9.17 Group Search [gs] vs Resource Aggregation [ra] comparison 65
9.18 Visualisation of Transportation task 66
9.19 Trajectory of Transportation 66