Dynamic task allocation with multiple demands and comunication failures for a multi-robot system

por

Ojilvie Avila Cortés

Tesis sometida como requerimiento parcial obtener el grado de

Maestro en Ciencias Computacionales

por el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Noviembre, 2015

Tonantzintla, Puebla

Supervisor:

Dra. Angélica Muñoz Meléndez

Coordinación de Ciencias Computacionales

INAOE

Nehhuatl, Ojilvie Avila Cortés, ni mexihca quemen nohpalli, icanon, nic pia huei mahuiliznelhuayotl inpampa to huehcacoltzintzin tolteca huan azteca, axcan nic nequi niquin tlazohcamatiz ohzapa ninque mahuiliztlacatzintzin aquinque o nech macahquen no nemiliz, no mahuiliz huan tlamatiliz, tlen o qui chihquen ninque tlacatl nopampa huan ipampa to altepetlacame mexican o tech macaquen ce tlapalehuiliztli tlen o huehca ome xihuitl o tlahto no pampa CONACYT. Nican ni mech tlahpalohua ican nin nahuatlahtolli tlen o nic zalo itech ineca teotlahtzintli itech metztli mayo huan macuilli centzontli huan ce xihuitl ican no temachtianitzin, axcan no teopanquixtitzin.

Tlenica:

No cochmaihnitzin:	*Karina García (ni mitz tlazohtlaz... cenca),*
To cihuapiltzitzin:	*Astrid Tlanextli huan Siegrid Iztacmixtli (yehhuan no yolo),*
No nantzin:	*María Luisa Cortés (ti mati to ilnamiquiliz)*
No tahtzin:	*Ojilvie Avila (cepa),*
No cihuatemachtianitzin:	*Angélica Muñoz (tlazohcamati ipampa mo palehuiliz),*
To cihuateopanquixtitzin:	*Elodia Popoca (aco yolpatzmiqui),*
To teopanquixtitzin:	*Genaro Medina (cenca yolpaqui),*

Miac tlazohcamati...

Axto ni mo tlazohcamati nanmo ixpantzinco, nehhuatl niquin yolehua nochtin mexihca tlazalozquen nin cuacualtzin huan nelmelahuac tlahtolli (nozo ohze itech nin mexicantlalpan) tlen tech maca xochiyolotl huehca itech ilnamiquiliztli i papalo... huan, icuac tic pian nahuatlahtolli, tehhuan ti cualtizquen tlanonotzquen ica to teco Teotzin, tlalticpac, huan nochi cemanahuac...

Yo, Ojilvie Avila Cortées, soy mexicano como el nopal, por eso, tengo una gran raíz cultural a causa de nuestros honorables ancestros toltecas y aztecas, hoy quiero agradecer nuevamente a estas cultas personas quienes me dieron mi vida, mi honor y conocimiento, que hicieron este hombre en mi y al pueblo de México por ayudarnos durante dos años a través del CONACYT. De aquí les saludo con esta lengua nahuatl que aprendí desde aquél mes de mayo del año 2001 con mi culto maestro, hoy mi venerado padrino.

Para:

Mi venerada esposa:	*Karina García (te amaré... siempre),*
Nuestras hijas:	*Astrid Tlanextli y Siegrid Iztacmixtli (ellas son mi corazón),*
Mi respetable madre:	*María Luisa Cortés (conoces nuestra historia),*
Mi respetable padre:	*Ojilvie Avila (primero),*
Mi respetable asesora:	*Angélica Muñoz (gracias por su ayuda,)*
Nuestra culta madrina:	*Elodia Popoca (ya no triste),*
Nuestro culto padrino:	*Genaro Medina (siempre contento)*

Muchas gracias...

Antes de que me despida de su honorable presencia, yo quiero invitar a todos los mexicanos que aprendan esta hermosa y verdadera lengua (u otra de esta tierra mexicana) que da un corazón honesto lejos de la mentira de la historia... y, cuando tengan la lengua nahuatl, ustedes podrán conversar con nuestro amo Dios, la naturaleza, y todo el universo...

Resumen

El uso de un robot para acceder a lugares peligrosos para las personas es común y necesario en estos tiempos. De la misma manera, los sistemas de más de un robot conocidos como Sistema Multi-Robot (abreviado MRS por sus siglas en inglés) son también comunes en la época actual y ventajosos con respecto a los primeros, esto debido a la posibilidad de operación simultánea y redundante que ofrecen. Nuestra motivación para realizar este trabajo es programar un sistema compuesto por varios robots autónomos, móviles y homogéneos que puedan detectar minas explosivas en un campo donde se presume que hay estos dispositivos dañinos, aunque la localización exacta y el número de dichos dispositivos sean desconocidos.

Un sistema como este ayudaría a evitar que la gente que habita en lugares con remanentes de explosivos sufra mutilación o muerte por alguno de estos dispositivos dañinos. Además, el mecanismo subyacente a esta operación puede llevarse a otros dominios de asignación automática de un conjunto de recursos y tareas a un conjunto de procesadores, máquinas o robots, para lo cual, la comunicación constante entre las máquinas es un requerimiento indispensable.

Pero, ¿qué sucede si estas máquinas fallan y no pueden realizar las tareas encomendadas? Una posible solución es que si hay más máquinas habilitadas disponibles, éstas puedan atender las tareas que las máquinas que fallaron no pudieron realizar. La detección de las causas de falla o la corrección misma de una falla en un robot están fuera del alcance de este trabajo; sin embargo, consideramos los casos en los

que las fallas evitan que los robots se mantengan comunicados entre sí y causan que el objetivo del grupo de robots no se pueda completarse.

En los Sistemas Multi-Robot con asignación de tareas del trabajo relacionado casi siempre se supone una comunicación infalible. Para llevar a cabo este trabajo, nosotros nos enfocamos en proponer una solución a fallas que se reflejen en la pérdida de comunicación en un Sistema Multi-Robot con Asignación de Tareas. Esto se logra mediante el uso de un mecanismo que hemos denominado *shaking calls*.

Nosotros proponemos un Sistema Multi-Robot centralizado y temporalmente decentralizado en caso de ser necesario, que incluye a un **servidor** que tiene la función de **asignar tareas** a un **conjunto de robots** mediante un **protocolo de comunicación**, mismo que **reasigna tareas** en caso de que haya fallas tanto en los robots como en el servidor. Para ello se estudian tres casos particulares de fallas: que existan robots que pierdan la comunicación con el servidor y no la recuperen; que existan robots que pierdan la comunicación con el servidor y la recuperen; y que el servidor pierda totalmente la comunicación con los robots y la restablezca posteriormente. En el último caso, el sistema pasa temporalmente a ser decentralizado dado que el servidor no puede cumplir con su función de asignación de tareas.

Para el funcionamiento del Sistema Multi-Robot se utilizan *instantáneas* del ambiente en combinación con el mencionado protocolo de comunicación, y un algoritmo de tipo voraz para la asignación de tareas. Dado que la pérdida de comunicación puede ocurrir en cualquier momento una vez iniciadas las operaciones, la asignación de tareas no puede realizarse desde un inicio y se considera de tipo **dinámico**.

Si bien nuestro algoritmo de asignación de tareas no busca realizar la asignación óptima, nosotros suponemos una independencia funcional entre el algoritmo de asignación y el protocolo de comunicación, por tanto, es posible combinarlo, con otros algoritmos de mayor eficacia con nuestro sistema como se demuestra con la incorporación al Algoritmo Húngaro para asignación de tareas para sistemas multi-

robot. Cabe mencionar que consideramos que cada tarea se compone de demandas, lo que quiere decir que cada tarea debe ser atendida en repetidas ocasiones por robots diferentes.

Los resultados que mostramos en este documento corresponden a simulaciones en el ambiente de simulación Webots comparando los casos de fallas contra simulaciones sin falla alguna. Estos resultados demuestran la **resiliencia a las fallas que ocasionan pérdida de comunicación** de nuestro sistema porque a pesar de que éstas ocurren, nuestro sistema es capaz de completar el mayor número de tareas en el caso que el número de demandas sea mayor o igual al número de robots (es decir, que los robots que fallaron no pudieron realizar las demandas de tareas encomendadas), o bien, completar todas las tareas en caso que el número de robots sea mayor al número de demandas. Finalmente, nótese que nuestro trabajo puede aplicarse a cualquier Sistema Multi-Robot con asignación de tareas que requiera **tolerancia a fallas**, por lo que la motivación expuesta al inicio de este resumen es sólo una de sus posibles aplicaciones.

Abstract

The use of robots for accessing dangerous places for people is common and necessary nowadays. Systems with more than one robot known as Multi-Robot Systems (abbreviated MRS) are common in the present time and they offer advantages to the latter because of their capability to operate simultaneously and redundantly. Our motivation for achieving this work is programming a system composed by several autonomous, mobile and homogeneous robots capable of detecting explosive mines in a field where it is presumed there are these harmful devices, though their exact location and their number are unknown.

A system such as this might avoid people who inhabit in places that have explosive remnants suffer mutilation or are killed by the explosion of mines and these harmful devices. Also, the underlying mechanism of this operation can be applied to other domains of automatic assignment of a set of resources or tasks, to a set or processors, machines or robots, for which the communication is a mandatory requirement.

But, what happens if these machines fail and they are not able to achieve the committed tasks? A possible solution is that if there are more capable machines available, they might serve the tasks that the failed machines could not achieve. The detection of the causes of failure in a robot as well as the correction itself of these failures are beyond the focus of this work; however, we consider the cases where the failures, whichever their cause, prevent the robots to keep communicated and caused

that the goal of the group of robots can not be accomplished.

In the revised related Multi-Robot Task Allocation systems, communication is supposed to be mostly infallible. In this work we propose a solution to failures which result in the loss of communication in a Multi-Robot Task Allocation system. This is achieved by using a mechanism that we have called *shaking calls*.

We propose a centralized Multi-Robot System that can be temporarily decentralized if necessary, which includes a **server** with the function of **allocating tasks** to a **set of robots** through a **communication protocol** which **reassigns tasks** in case there exist faults on both, robots and the server in three particular cases of failure: there exist robots that lose communication with the server and the communication is not recovered; there exist robots that lose communication with the server and later the communication is recovered; and that the server completely loses communication with robots and subsequently the communication is restored. In the latter case, the system temporarily becomes decentralized since the server can not fulfill its function of task allocation.

In order to operate, the Multi-Robot System uses *snapshots* of the environment in combination to the afore-mentioned communication protocol and a greedy Algorithm for task allocation. Since the loss of communication might happen at any time during operations, the task allocation can not be solved at the beginning and is considered **dynamic**.

While our task allocation Algorithm does not have the purpose of finding the optimal assignment, we assume a functional independence between the assignment Algorithm and the communication and protocol making it possible to combine it with other more effectiveness algorithms as it is demonstrated in this work by incorporating the Hungarian Algorithm for Task Allocation in Multi-Robot Systems. It is worth to mention that we consider each task is composed by demands, this means that each task must be attended in repeated times by different robots.

The results shown in this research are simulations performed in the Webots simulation environment, comparing cases of failures against simulation of ideal situations. These results demonstrate the **resilience to failures that cause loss of communication** of our system because, despite failures happen, our system is able to complete as many tasks as possible in the case that the number of demands is greater or equal to the number of robots (*i.e.*, robots that failed were not able to achieve the demands of committed tasks), or complete all tasks when the number of robots is greater than the number of demands. Finally, note that our work can be applied to any Multi-Robot Task Allocation system that requires **fault tolerance**, so the motivation exposed at the beginning of this abstract is just one of its possible applications.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Explosive mines represent a danger even though they were created and located for war conflicts nowadays ended. In the report of the year 2013 of L&CMM, short term for *Landmine and Cluster Munition Monitor*[1], it is reported that there were 3,628 victims during 2012, from which 1,066 were fatal casualties caused by the remains of explosive devices in 62 countries: Afghanistan, Colombia and Yemen in first, second and third place, respectively, in number of victims. It is necessary to highlight that most of these victims were not related to the military.

The number of undetected mines currently grounded in post-conflict countries is unknown. However, it is estimated that at least 59 countries have mined fields, according to the L&CMM organization. The process of detection and remotion of explosive remnants is made by qualified personal and sophisticated equipment such as metal detectors. Initially motivated for investigating a robotic solution to this problem, we propose a feasible solution for mine detection using a Multi-Robot System that will help to reduce the risks of people who is exposed to, inhabit near or have to traverse fields that were once mined. The problem is investigated in this research using a scenario where a team of autonomous robots have to search mines

[1] http://www.the-monitor.org

within an enclosed environment, and a number of repeated inspections or demands in charge of different robots are combined into a hypothesis about the presence of mines at a target position or region.

Multi-Robot Task Allocation (abbreviated MRTA) is not new and different approaches have already been proposed in the literature, as we will revise in Chapter 2. However, there exist failures that will result in communication issues. We consider communication failures as an abstraction of other kind of failures. Thus, consider, for instance, there exist radio interference or a stuck robot for a mechanical problem or for a drained battery while trying to reach its goal. This robot will eventually lose communication with the system and the tasks currently assigned will never complete. This situation might certainly cause that the goal of the system cannot be accomplished. These cases of failures in underlying MRTA algorithms have been poorly studied in these approaches. Most of the proposed solutions, as far as we know, assume that the members of a robot team are able to communicate using a reliable system. And it is precisely in this niche where our work is circumscribed: **how to propose a mechanism to deal with communications fails among the members of a robot team that can enrich MRTA algorithms**.

It is also worth to mention that MRTA is an abstraction that can be extrapolated to many other domains beyond the design of demining robot teams. For instance, in domestic or service scenarios, automated robots might look for products located in shelves in order to fulfill several users' orders.

From now on, we will call "communication failures" to any failure in a set of robots from a MRS that will result in lose of communication with the rest of the system. Note that these kind of failures might be very common in a real-world scenario, and that they can be the result of several external factors that we will not treat in this work. Instead, we show one feasible solution to deal with these failures by proposing a mechanism that will reassign and complete tasks whenever required. Note that the proposed mechanism is independent of the task allocation

algorithm used, therefore, the mechanism might be set to work with other algorithms if required.

1.2 Problem definition

In this section we begin by introducing some conditions in order to understand our objectives and methodology. The complete definitions, the formal definition and assumptions are given in Sections 2.1, 4.1 and 5.1, respectively.

1.2.1 Considerations and hypothesis

Below we describe the conditions and properties considered in this research:

- The environment represents a mined field on the ground, without obstacles, delimited and previously divided into m parts not necessarily of equal size.
- The m partitions are abstractions of **tasks** to be allocated to the robots.
- Each task is composed by at least x and at most y **demands** ($x < y$).
- The **tasks** and their **demands** are announced by a **central server** which assigns tasks to the robots. This function can be temporarily transfered or assigned to the mobile robots in case of interruption of communication, and reassigned to the central server when communication is recovered, including all pertinent updates of the system if any.
- There exist n homogeneous, autonomous and mobile robots in the system, to which demands are allocated. These robots conform the **multi-robot system**.
- There exists a **communication mechanism** among robots, and between robots and the central server. There exists, also, a positioning and orientation mechanism for the location of the robots.

- Each robot is able to detect using its own sensors (sensors available in the market, for example, metal detectors or temperature detectors), the presence or absence of objectives and the result must be reported to the central server.
- A task can have one and only one of two possible states: "open" (while the task has unsolved demands) or "close" (when a minimum number of demands has been completed).
- A task will remain "open" in case of disagreement of results reported by robots in different demands of the task, in such case. A demand for a task cannot be allocated to the same robot twice.
- The overall goal of the multi-robot system is achieved when there exists a **minimum number of demands** for each and all of the existing tasks (*i.e.*, all task are in the state "close"); or when a **maximum run time** is reached.
- There are two measures to be taken into account: the time required for the system to complete all the tasks (or reach the maximum run time) and the overall accumulated distance traveled by the robots, called metrics of **completion time** and **total distance**, respectively.

We state here the **hypothesis** underlying this research:

A simple mechanism based on the exchange of continuous messages or "shaking calls" is enough for ensuring the resilience to communication failures of a MRS in a context of a MRTA problem.

1.2.2 Task Allocation in a Multi-Robot System

Task allocation refers to assign a set of jobs or tasks to a set of robots following some optimality criterion, since the execution of any task has an associated cost. The calculations where the optimal case (*i.e.*, the **overall minimum cost**, for example)

require to **compute all possible combinations** of allocating task to robots in order to obtain the optimal result. Note that the computation of combinations implies a very high computational cost in the order of **O(n!)**. Thus, Muti-Robot Task Allocation from the perspective of a single robot can be modeled as an instance of the **Traveling Salesman Problem** (Miller et al., 2007). There exist suboptimal solutions for this problem which implies the use of *heuristics* and the computational order is set to *polynomial* ($O(n^k)$).

Furthermore, imagine the tasks are **created** in run time; or have a **due time**; or a task is never achieved because the robot which was assigned had a failure; in such case, the task allocation is considered to be **dynamic**, making the problem even more complicated. The dynamic task allocation of m tasks among n robots, it is known to be **NP-hard** problem (Miller et al. (2007), Dasgupta (2011)).

These conditions and hypothesis are specially relevant when dealing with task allocation algorithms intended to operate on physical robots. In this case, robots must be able to deal with common failures and constraints that are not necessarily considered in simulated experiments. These failures and constraints are, for instance, vision and local sensing only in the vicinity of the robots; limited processing capabilities, storage, sources of power and communication, noise and uncertainty in sensors as in actuators of the robots, just to mention a few examples.

1.3 Objectives

The main goal of this research is to **propose a mechanism for dealing with communication failures, as a counterpart to Algorithms for task allocation with multiple demands for a multi-robot system, implement it and evaluate it.**

The set of particular goals that have been defined for the achievement of the

general goal are enumerated below.

1. Propose and design a greedy Algorithm for task allocation with multiple demands relying on a fully reliable communication mechanism.
2. Implement previous Algorithm using the Webots© simulation environment.
3. Propose and design an extension to the previous Algorithm and adapt it to cases with failures in the communication when at most one third of robots fail and when the central server fails.
4. Implement the second Algorithm using the Webots© simulation environment.
5. Evaluate the performance of all variations of our proposed algorithms using standard metrics of completion time and total distance.

1.4 Methodology

Below we revise briefly the methodology that was followed for the achievement of our goals.

- Design a greedy Algorithm for task allocation with multiple demands

  As a basis for comparison or **baseline**, a centralized greedy Algorithm was proposed and implemented for task allocation with multiple demands. For allocating a task, a matrix of costs (where the rows correspond to robots and columns to tasks) is calculated based on two aspects: (1) the ratios of distance between a robot and a task, and (2) the current battery level of the robot. The assigned task corresponds to the minimum value in the robot's row. The matrix is updated according to current robots' locations and battery levels and known

tasks' locations. All these values are stored in tables (one for tasks and one for robots) in the central server. The tables and tasks allocations are transmitted to the robots in the system through the explicit communication system. The robots and the central server share the most up-to-date tables. When a task is assigned to a robot, it starts its way to the task's location based on its copy of the table of robots. In this case, the stop criterion is to complete all tasks.

- Implement an Algorithm using the Webots© simulation environment

  For testing whether the first Algorithm is successful, a simulation environment is required. The Webots© simulator has been selected for this purpose, which is capable of simulating multiple robots, the environment and the tasks. Results are obtained varying the number of tasks and robots (in combinations of 3, 6, 9 and 12 robots and 4, 8, 12 and tasks), and are analyzed in metrics completion time and total distance.

- Design an extension to previous Algorithm for dealing with failures in the communication mechanism

  With the task allocation Algorithm ready, the mechanism that deals with lose communication is implemented. Three cases of failure are considered: (1) at most a third part of mobile robots lose communication and there is not recovery; (2) at most a third part of mobile robots lose communication and there is later communication recovery; and (3) the server loses communication and later recovers. These cases were simulated by sending to the robots special codes in order to: (1) stay at current location, fail definitively and cease all communications when a failure without recovery is required; (2) stay at current location and fail during a period of time in which communication is lost and eventually recovers for the case of failure with recovery; and (3) the central server disables its transmitter in order to avoid sending any messages for a period of time and eventually, it restores communications by reenabling the

transmitter. The task allocation algorithm is executed by both, by the server whenever a robots fails; and by the robots when the server is unavailable and the robots must self-assign tasks.

- Implement the fault-tolerant mechanism using the Webots© simulator

  The mechanism that deals with communication failures along with the task allocation Algorithm is tested in a simulated environment. The previously generated environments (second step) are used again for testing this mechanism. New results were obtained using the previous combinations of tasks and robots and they were compared against the results obtained using the Algorithm without failures using the metrics mentioned above. The resulting mechanism (which deals with communication failures) is called "shaking calls".

  Note there are three cases of failure to be considered, so is the number of simulations per environment to perform.

- Evaluate the performance of our mechanism with respect to reliable communications

  Our mechanism is compared against the Algorithm's performance without communication failures based on the specified metrics (*c.f.* Subsection 1.2.1, last item). In terms of the achievement of a global goal, all the metrics are increased due to the failures and tasks reassignations. However, in most cases, the mechanism enable robots to finish all the required tasks.

  On the other side, when a robot fails out of three robots used, specifically in failures without recovery, the MRS completed as many tasks as possible since by definition all tasks are composed by three demands, thus there are unfinished tasks left by the failing robot (completion time grows while total distance decreases). The obtained results demonstrate that even though our mechanism is not the more efficient than other task allocation Algorithms, is effective because it is able to recover from failures by reassigning tasks.

1.5 Contributions

The main contributions of this research are summarized below.

1. A fully functional greedy Algorithm for dynamic task allocation that can be incorporated to a module capable of dealing with communication failures.
2. A communication protocol required for coordination among robots based on the exchange of messages.
3. An operable mechanism (shaking calls) capable of dealing with communication failures independent of the task allocation algorithm.

1.6 Outline

The rest of this document is organized as follows. Chapter 2 includes all the theoretical basis that supports our research. Chapter 3 shows all the relevant related work and a comparison with the proposed research. Chapter 4 gives a formal definition and establishes boundaries of our work. Chapter 5 proposes the mechanism for dealing with communication failures and the communication protocol for data exchange. Chapter 6 presents our experiments, results and interpretations. Finally, in Chapter 7, our conclusions, contributions and recommendations about future work in order to deepen into extensions of this research are discussed.

Chapter 2

Foundations

In this Chapter, we revise relevant definitions as well as underlying notions of this research: in Section 2.1 we introduce important concepts and definitions; we present a widely accepted taxonomy for multi-robot task allocation is presented in Section 2.2; in Section 2.3 we review different techniques applied to the problem of task allocation; in Section 2.4 we expose issues concerning communication among robots. Finally in Section 2.5, we discuss the criteria that are important to determine if an environment where a muti-robot system is deployed is static or dynamic.

2.1 Basic definitions

We begin this section by defining concepts required for understanding the meaning of this work.

2.1.1 Tasks and demands

Definition 1. A **task** is an abstraction of a finite set of actions (behaviors) executed by a machine on a target.

Gerkey and Matarić (2004) have also stated that a task is a subgoal required to achieve a global goal.

In our case, **robots** are the machines and **tasks** are points within an environment that must be visited by robots. Figure 2.1 shows an example of six tasks distributed within an environment represented as a 20 × 20 units square.

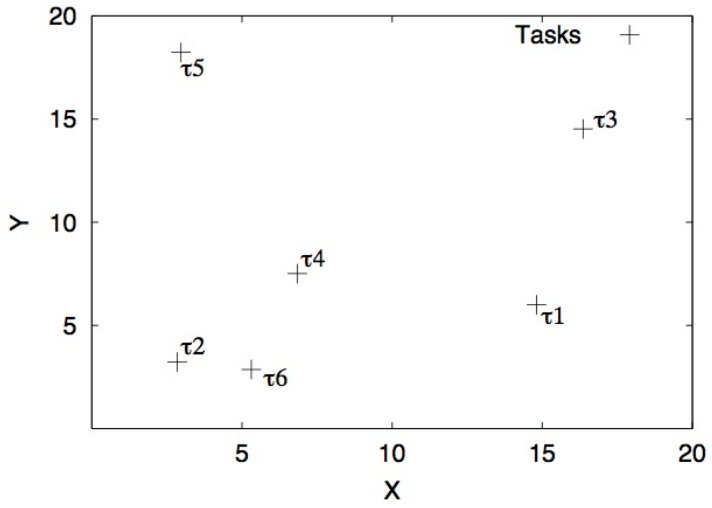


Figure 2.1: Tasks (τ_i) distributed within an enclosed environment. From: Munoz-Meléndez et al. (2012)

Definition 2. A **task** is said to be **executable** if previous conditions (such as availability, order of execution of tasks and time requirements) were achieved or are under execution (Botelho and Alami, 1999).

Definition 3. A **Dynamically Assigned Task** is a task that fulfills at least one of the following conditions: it is created, discovered, assigned or reassigned during **execution time**; or the number of these tasks is **unknown** *a priori*; or the tasks have an **expiration** or **due time**, priority or partial order requiring changes from the conditions initially known.

Definition 4. A **demand** is defined as *a requirement of an operation by a robot on or at a target* (**task**) (Munoz-Meléndez et al., 2012).

In this work, a demand is one visit to a point (task) in the environment that

must be executed by a robot. We require a minimum number of visits by different robots to each task in the environment in order to complete the task. We will refer to a visit by the name **demand** from now on.

2.1.2 Robots and behaviors

Definition 5. A **behavior** is the state or set of states that match one's robot action (for instance processing information, interaction with an object, moving).

Note that according to definitions 2 and 5, **behavior** is not equivalent to task. However, note that one or more behaviors might be required in order to fulfill a task.

Definition 6. A **Multi-Robot System**, abbreviated **MRS**, is any set which has more than one autonomous robot working for a common goal. A group of robots is **homogeneous** if their individual **capabilities** are **identical**. Otherwise the group of robots is **heterogeneous**, and their different **capabilities** complement the tasks they can achieve (Cao et al., 1997).

In this research we assume a **homogeneous** multi-robot system.

2.1.3 Communication

Definition 7. Communication in a MRS is the capability of robots for **exchanging information** in order to coordinate their actions.

Communication can be **explicit** or **implicit**. Explicit communication means the robots have a **direct mechanism** for exchanging messages in an ordered way and it usually involves a protocol, for example, wireless radio communication. **Implicit communication** requires that each robot be aware of the **actions** that other robots perform in the environment (Trojanek et al., 2007), for example information transmitted trough a trail of virtual pheromone.

Definition 8. A **protocol** is a convention of **rules** for exchanging any kind of information in a communication scheme between nodes in a network. For our case, the network is the MRS and the nodes are the robots and server of the system.

In this work we assume **explicit radio communication** and a **communication protocol** is required for exchanging information.

2.1.4 Failure and recovery

In the state-of-the-art literature, **failure** is rarely mentioned, since algorithms for MRTA rarely deal with any kind of failure. Thus we have to introduce a definition for this concept. We propose two kinds of failures in a MRS: **robot-related failure** (attributed to hardware or software problems but not its capabilities that are assumed to fulfill the requirements for the tasks' execution) and **environmental failure** (caused by factors that are external to robots and which might cause misbehavior of robots).

Definition 9. A **failure** is any reason or cause that has **prevented** a task from being **achieved**.

We consider a **communication failure** as any failure that results on the interruption in the communications in a MRS. Note that these failures might be caused by both, robot-related failure and an environmental failure.

Definition 10. Recovery is the action of returning to a **operational status** for all members that conform the MRS.

In this research we deal with both, cases where failures are definitive, and cases where failures are temporary and thus, the recovery and reincorporation of robots to the MRS are possible.

2.1.5 Centralization and decentralization

Definition 11. A MRS is said to be **centralized** if and only if there is a **unique single point** of calculation and/or repository of knowledge of the entire environment and **decisions** about actions of the systems are based on that knowledge. If this condition is not met, the system is said to be **decentralized**.

Note that single points existing in centralized systems are more prone to failures since there is a bottleneck in their unique point of control.

A fully centralized approach utilizes a single machine to **coordinate** the entire system. This approach is best suited for applications involving **static** environments or when **global knowledge** of the environment is available. On the other hand, in decentralized systems calculations and decisions are made by single robots based on the **perception** each robot has of a **partial environment** (Dias et al., 2006).

In this research we use a **centralized** approach since the work application is mine detection, meaning each mobile robot is under constant danger because the location of the mines is unknown and could explode at any time. A server located in a safe place nearby the mined field that is in charge to calculate and assign tasks to robots is considered in our scheme.

2.2 Multi-Robot Task Allocation and taxonomy

Definition 12. Multi-Robot Task Allocation, abreviated **MRTA**, is a system composed by m **tasks** (or subgoals) to be achieved by n available **robots** with a known and **required skill** (or ability) **optimizing** some criteria in a given **period of time**.

It is important to emphasize that there are a number of MRS applications, such as foraging, where robots usually look for **objects of interest** until one object is found. In this problem, the robots have to collect the objects to contribute

to an overall goal for the MRS. The main difference between foraging multi-robot systems and MRTA is that in the second problem, there is a specific calculation (**optimization**) of how the work is divided into available robots, taking into account the robots' capabilities and restrictions; environment; communication; among other key features, considerations that are not taken into account in applications of foraging robots.

A widely accepted **taxonomy** in MRTA problems is proposed by Gerkey and Matarić (2004). They present three dimensions for classifying the Multi-Robot Task Allocation problem that are detailed below.

-The number of tasks a robot is able to execute at a given time, this is **single-task robots** (**ST**) versus **multi-task robots** (**MT**). A single-task robot accomplish **one and only one** task per turn, whereas a multi-task robot is able to execute **more than one task** simultaneously.

-The number of robots that are necessary to complete a task: **single-robot task** (**SR**) versus **multi-robot task** (**MR**). A single-robot task requires interaction of **one and only one** robot to be considered as accomplished. On the other hand, a multi-robot task requires **more than one** robot to be fulfilled.

-The nature static or dynamic of the tasks: **instantaneous assignment** (**IA**) versus **time-extended assignment** (**TA**). Instantaneous assignment means that all the tasks are assigned once and there **will not be further changes** to these assignments. Time-extended assignment considers **changes might happen** over time.

The final class of a multi-robot system is given in any possible combination of these three dimensions. For instance: **ST-SR-IA** stands for a multi-robot system with robots that can perform only one task at a time, one robot is enough for achieving a task and the task assignment is made at the beginning and will not change as time passes over.

Our research work is a **ST-MR-TA** problem: Single Task robot because a robot is able to perform one and only one task at a given time; Multi-Robot task since the domain require the fulfillment of more than one demand per task by different robots; and Time-extended Assignment because we consider communication failures which lead to dynamic assignments in run-time.

2.3 Task assignment techniques

In this section we review the most significant techniques that have been proposed so far for the problem of MRTA.

2.3.1 Greedy assignment

This simple technique of assignment tries to solve the task allocation problem (find the global optimum) by choosing the local optimum (minimum in this case) in each iteration in a reasonable time. However, this technique lacks of further sightseeing even when there is an obvious global optimum. Greedy assignment is based on greedy scheduling algorithms, for instance Brassard and Bratley (1996) propose the example described below.

Consider a single server which has a finite fixed number n of customers to serve. Each customer i expends a known time t_i in the system, $1 \leq t_i \leq$ n. The objective is to minimize the average time that a customer spends in the system by minimizing the total time spent in the system by all the customers.

Let n=3 be customers, with t_1=5, t_2=10 and t_3=3.

Table 2.1 illustrates how much time is spent by customers in the system when they are served at different turns. Note that a customer's time is added to time of customers who are before in order. Also note that order 312 is optimal in time,

whereas 213 is the worst. So, if we have to minimize the time the last customer spends in the system, we have to minimize the time that previous customers spend in the system. Therefore, the Algorithm is simple: at each iteration add to the end of the schedule the remaining customer with shortest time.

Order	T	
123:	5+(5+10)+(5+10+3) =38	
132:	5 + (5+3)+(5+3+10)=31	
213:	10+(10+5)+(10+5+3)=43	← worst case
231:	10+(10+3)+(10+3+5)=41	
312:	3+(3+5)+(3+5+10)=29	← optimal
321:	3+(3+10)+(3+10+5)=34	

Table 2.1: All possible cases of order in a greedy example, where T represents the overall time spent by the customers. From Brassard and Bratley (1996)

As it can be seen, a greedy Algorithm can either minimize or maximize a given criterion of optimality. In case of MRTA, one optimal criterion is, for example, the Euclidean distance from tasks to robots.

The example seen in this section has been proven by the authors to be optimal (see Section 6.6.1, Brassard and Bratley (1996)). However, this Algorithm considers static conditions such a fixed number of customers and static time of attention. As a matter of fact, this information is not always available in real world applications.

2.3.2 Hungarian assignment

This Algorithm was proposed by Kuhn (1955) for n workers and n jobs and later revised by Munkres (1957) to be used for n workers and m jobs. It has been recently revisited for robots and tasks by Lenagh (2013).

The Algorithm has these steps:

Let n and m be a set of workers and a set of jobs, respectively. Each job has a specific cost (for instance time or distance) for each worker.

Step 0. A matrix called *matrix of costs* is created with rows representing workers and columns representing jobs. Each element of the matrix represents the cost of assigning worker n to job m. Let $k=min(n,m)$.

Step 1. For each row in the matrix, find the minor entry and subtract it from every value in that row.

Step 2. Find a zero in the new matrix and mark it (put an asterisk on it: 0*) if there is not a marked zero in the same column or row. Repeat this instruction for all entries in the matrix.

Step 3. Mark each column containing a marked zero. If the number of marked columns is equal to k then the **marked zeros (0*)** indicate the optimal assignments and the Algorithm ends; else, go to **step 4**.

Step 4. Find a non marked zero and prime it (put an apostrophe on i: 0'). If there is no marked zero in the row containing a primed zero go to **step 5**; else, mark the row and unmark the column containing this marked zero. Repeat this instruction until there are not unmarked zeros. Find the minimum in the unmarked entires. Go to **step 6**.

Step 5. Change primed zeros to marked zeroes as follows. Let Z_0 represent the primed zero (found in **step 4**); let Z_1 represent the marked zero in the column of Z_0 (if any); let Z_2 represent the primed zero in the row of Z_1 (there will always be one). Continue until there are not marked zeroes in a primed zero column. Unmark each marked zero, mark each primed zero, unprime them and unmark every line in the matrix, return to **step 3**.

Step 6. Add the value found in **step 4** to each entry of a marked row and

subtract it from each entry of unmarked columns. Return to **step 4** without any change to the matrix.

Table 2.2 illustrates the operation of the Hungarian Algorithm using these sets: Workers = {a, b, c}, Jobs = {p, q, r}.

2.3.3 Market-based assignment

The principle of a marked-based mechanism is to settle a market where tasks are for sale and robots act as selfish agents in search for individual profit. These robots are *paid* for tasks they achieve and they *pay* with *some electronic currency* for the resources they consume (Kalra and Martinoli, 2006). A market-based mechanism tries to optimize overall execution time and added costs. This objective is achieved by using a bidding protocol for the task assignment (Geppert et al., 1998).

The auction/bidding Algorithm works as follows:

Step 0. A dedicated agent or one of the robots auctions tasks during a finite period of time (auction period).

Step 1. During the auction period, each robot bids for winning *cheaper* tasks, taking into account the execution time and costs.

Step 2. All bids are announced and exchanged by robots, once received, these bids are taken into account for assigning a task.

Step 3. The *best bidder* is chosen and the auctioned task is, therefore, assigned to the winning robot. If a *tie* happens, there should be an untie mechanism based for instance on the robot's *id* or on its hierarchy.

Step 4. If there are left unassigned tasks, go to **step 0**, else finish auctions.

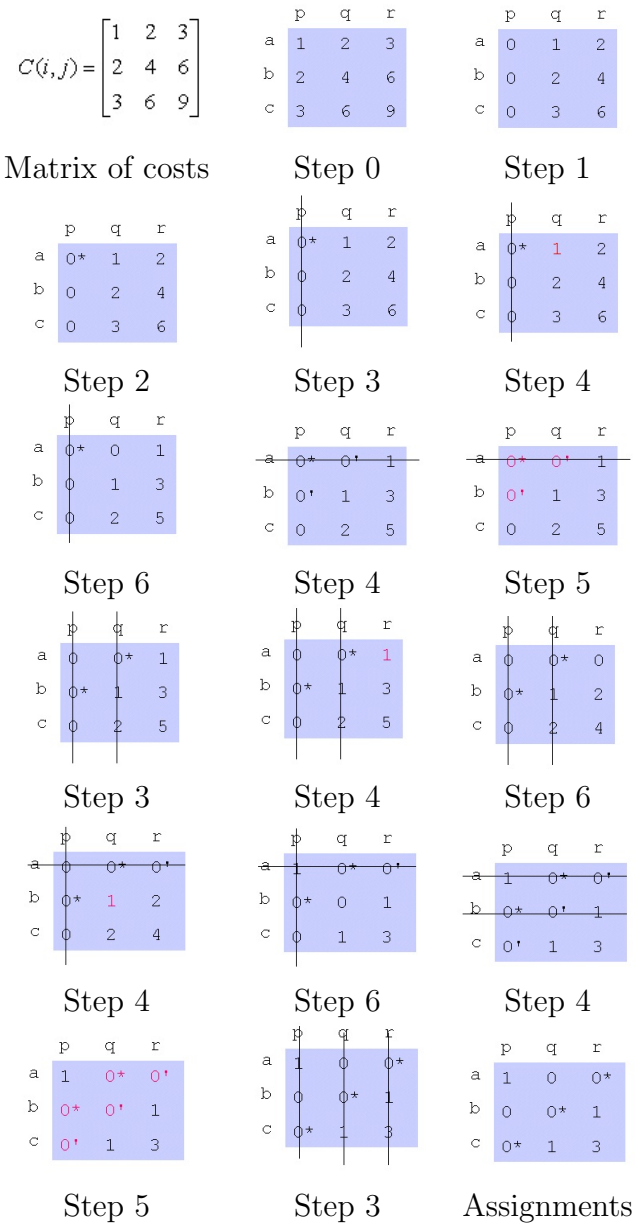


Table 2.2: Example of Hungarian Algorithm by steps on a 3×3 matrix. The assignments correspond to the zeros with an asterisk in the last matrix: task r is assigned to worker a; task q is assigned to worker b; and task p is assigned to worker c. From: http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html

The market-based mechanism can be either, centralized or decentralized as we detail in the chapter that reviews the state of the art. Also note that if there are tasks created at run time, the auction/bidding Algorithm resumes.

2.3.4 Stochastic assignment

In this section we review the Vehicle Routing Problem, abbreviated as VRP, in order to model the stochastic queuing according to Bullo et al. (2011):

1. A team of m vehicles is required to provide service to a set of n demands in a 2-dimensional space.
2. Each demand requires a certain amount of on-site service.
3. The goal is to compute a set of routes that optimizes the cost of servicing (according to some quality of service metric) the demands. Note that new demands are created dynamically, thus, this assignment is also known as Dynamic Vehicle Routing (DVR). See Figure 2.2.

The VRP is modeled trough a queuing model known in the literature as the m-vehicle Dynamic Traveling Repairman Problem (m-DTRP) defined as follows: consider m vehicles free to move, at a constant speed v, within $\mathbb{R}^2$. Demands are generated in a bounded and convex set Q, called the environment. A location of a demand becomes known at its arrival epoch; thus, at time t we know with certainty the locations of demands that arrived prior to time t, but future demand locations form an independent and identically distributed sequence.

At each demand location, vehicles spend some time $s \geq 0$ in on-site service, then, a served demand is removed from the system once one of the vehicles has completed its on-site service. The system time of demand j, denoted by T_j, is defined

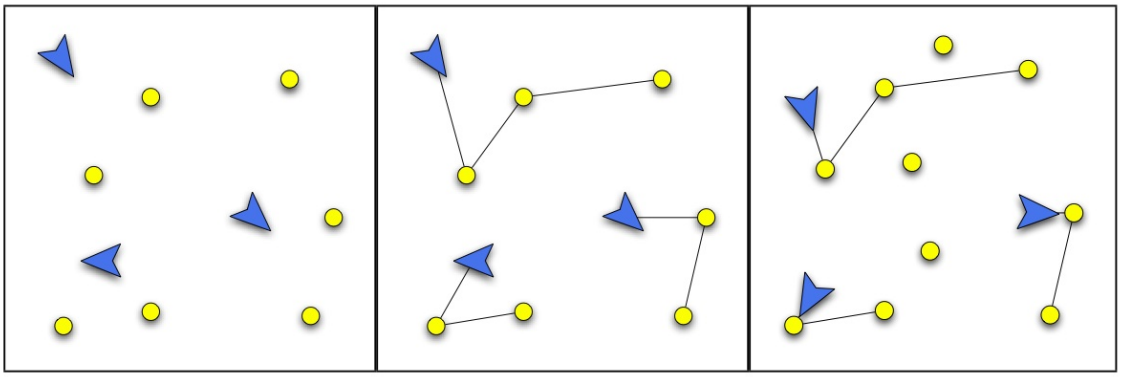


Figure 2.2: Example of Dynamic Vehicle Routing (DVR): left and middle: vehicles are assigned to customers and the routes planed are represented respectively as triangles, circles and lines. Right: the DVR problem is how to re-allocate and re-plan routes when new customers (new circles) appear. From Bullo et al. (2011)

$$\overline{T}^* := \overline{T}_{\pi}* = \inf_{\pi \in P} \overline{T}_{\pi}* \tag{2.1}$$

Formula 2.1. The optimal time ($\overline{T}^*$) is equal to the time of the optimal policy ($\overline{T}_{\pi}*$) which is the infimum of the times of all policies $\pi \in P$. From Bullo et al. (2011)

as the elapsed time between the arrival of demand j and the time one of the vehicles completes its service.

A policy for routing the vehicles is said to be stable if the expected number of demands in the system is uniformly bounded at any time. Let P be the set of all causal, stable, and time-invariant routing policies and $\overline{T}_{\pi}$ be the system time of a particular policy $\pi \in P$. The m-DTRP is then defined as the problem of finding an optimal policy $\pi* \in P$ (if one exists) as indicated in formula 2.1.

2.4 Communication techniques

As we mentioned in Subsection 2.1.3, the members of a MRS require an explicit mechanism of communication. In this section, we explain methods for transmitting information in a network. Also, a brief explanation about how to deploy a network for fulfilling the requirements of communication of this work.

2.4.1 Unicast versus broadcast methods

Unicast means that a message is transmitted to only one possible destination through a unique identification (address) in a network. The specific use of **unicast** is transmitting information to only one robot in the required case, for example, a task assigned to a robot concerns only to that robot. In **broadcast**, contrary to **unicast**, a message is sent to all possible destinations within a network simultaneously. A specific application of the broadcast method is to transmit the tasks' location to all robots in one step of transmission. There is not a method better than other, a suitable method is rather selected depending on what kind of messages that are to be transmitted in a network.

Gerkey and Matarić (2002) highlight the inefficient waste of bandwidth of transmitting **unicast** messages to every and each robot in a MRS compared to **broadcast**. However, an arguable statement is that not every robot should receive every message transmitted in the system and that discarding third parties' messages also might cause local overhead to a robot.

Our work uses both methods of communication, **unicast** and **broadcast**, depending on the type and application of information to be transmitted.

2.4.2 Ad-hoc and infrastructure wireless networks

An **ad-hoc wireless network** is a kind of network whose communication does not rely on a device such as an **access point** or a router as an intermediary. This gives a sense of decentralization since all the nodes (hosts) that compose the network are interconnected directly through their wireless network interfaces. However, the wireless network interfaces have usually limited power transmission. Figure 2.3 (left) shows an *ad-hoc* wireless network where all the nodes are laptops. Note the lack of an intermediary device connecting wirelessly the laptops.

An **infrastructure wireless network** requires an intermediary central device (**access point** or **wireless router**) that connects and provides connectivity and security to all the nodes in the wireless network. Also, an **access point** interconnects wired and wireless networks. Finally, their detachable antennas can be substituted by high gain antennas that provide a wider range of communication. Figure 2.3 (right) shows a wireless network connected to a wired network through an **access point**.

2.4.3 Communication network deployment

In our work, robots are mobile and autonomous. We have chosen the *Koala* robot for experimentation. This model of robot have been simulated within the Webots© environment, that provides realistic models of common robotic platforms.

In a real world scenario an ad-hoc network is the simplest wireless network. However, as this work uses a centralized system, a significant coverage of the environment is required. Therefore an infrastructure network will be deployed with an access point connected to the server which assigns tasks to robots. A **private IP C-class network version 4** is enough for addressing up to 254 hosts including all three mobile robots and the central server.

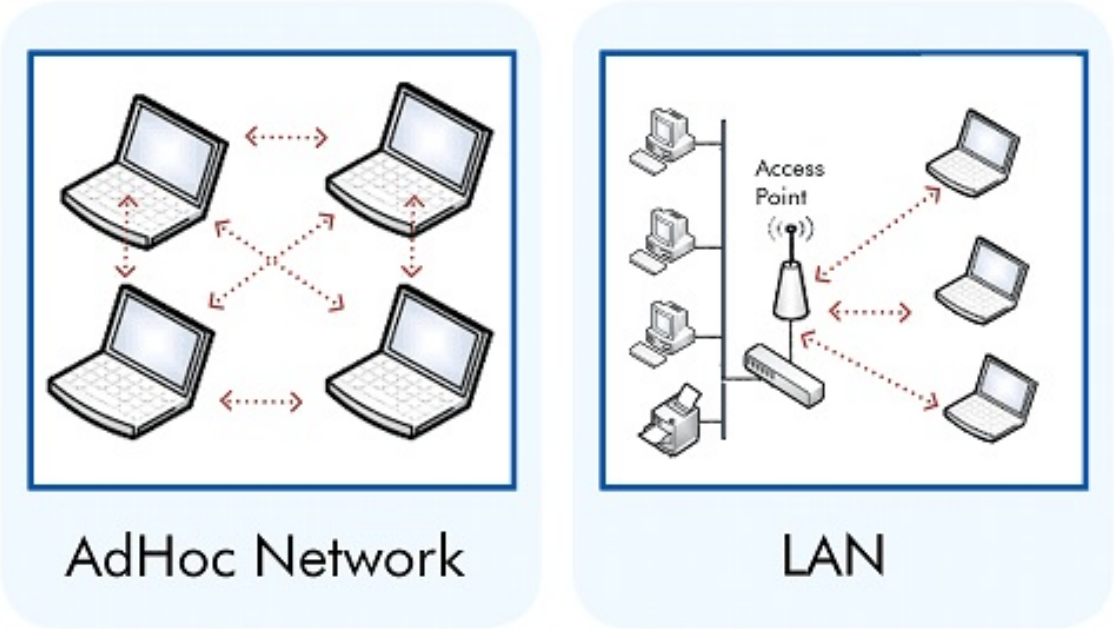


Figure 2.3: Two kinds of network: ad-hoc (left) and infrastructure (right). From: http://www.bb-elec.com/Learning-Center/All-White-Papers/Wireless-Cellular/How-to-Make-Devices-Communicate-in-a-Wireless-Worl.aspx

In case of unicast, a **specific IP address** is used for each host (server and robots), whereas **broadcast** subnet address (X.X.X.254) is used when broadcast is required.

Concerning the simulated environment in this research, the simulated robots have wireless radio mechanisms equivalent to an ad-hoc network. The receiver and transmitter modules for the robots in the simulator have **100 channels available** and they do not require any other module equivalent to a wireless access point for establishing communication.

In the case of unicast transmission, **channels 0 through 99** are used for allocating one channel per robot. In case of broadcast, **channel -1** is used and all receivers will catch the message despite their tuned channel.

As stated in Section 1.4, we model the communication failures in robots as follows: failures without recovery by sending a special code to the robots "designated" to fail to terminate operations; and failures with recovery by sending a special code to the robots "designated" to fail in which case robots cease all communications for a period of time. In the case of failure with recovery of the server, it will cease to send messages by disabling temporarily its transmitter and latter reenabling it. All calculations of time of failure are exposed in Subsection 6.1.3.

2.5 Dynamic and static environments

In order to differentiate a static versus a dynamic environment in the MRTA problem, at least one of the following conditions must be held in order to be considered as a dynamic environment:

1. Tasks are created, reassigned or known during run time.
2. Tasks have a due time, expiration and/or priority.
3. Tasks' locations are unknown or unavailable.
4. Any change to any starting plan for scheduling tasks is made.
5. Robots' failures during run time might happen.
6. Unexpected obstacles might be present in the environment.
7. An uncharted or partially known environment is used.
8. Communication is considered unreliable.
9. Communication failures force any previous allocation.

2.6 Summary

- We conclude this chapter by reminding that we have defined concepts that are required to understand this research in Section 2.1.
- Our work is considered a ST-MR-TA (Single-Task robot, Multi-Robot task, Time-extended Assignment) according to the taxonomy given in Section 2.3
- We also revised important algorithms for allocating task: a greedy approach; Hungarian assignment; market-based assignment; and stochastic assignment in Section 2.3.
- We have stated the use of unicast and broadcast in our work and how we modeled the failures of communication in Section 2.4.
- We give a list of conditions to distinguish between static and dynamic environments. Our case namely fills the condition that online reassignment of tasks is necessary due to communication failures, as presented in Section 2.5

Chapter 3

State of the art

In this chapter we revise the most significant literature of our field. We present a taxonomy of the multi-robot task allocation problem in Section 3.1; we revise the related work of muti-robot systems and the techniques proposed for task allocation in Sections 3.2, 3.3, 3.4 and 3.5, according to the previously mentioned taxonomy in Section 2.2; and finally, we summarize key features of related work and compare our work with respect to related work in Section 3.6.

3.1 Taxonomy for related work

For practical purposes, we have divided the related work in the three dimensions introduced by Gerkey and Matarić (2004) and presented in Section 2.2. Note that there are eight different divisions, depending on the combinations given.

1. **Single-Task robots - Single-Robot tasks - Instantaneous Assignment** (**ST-SR-IA**). Robots serve one task at a given time, tasks are served by robots only once, and there is only one round of assignations (static environment).

2. **Single-Task robots - Single-Robot tasks - Time-extended Assignment** (**ST-SR-TA**). Robots serve one task at a given time, tasks are served by robots only once, and there are more than one round of assignations (dynamic

environment).

3. **Single-Task robots - Multi-Robot tasks - Instantaneous Assignment** (**ST-MR-IA**). Robots serve one task at a given time, more than one robot is required to serve a task, and there is only one round of assignations (static environment).

4. **Single-Task robots - Multi-Robot tasks - Time-extended Assignment** (**ST-MR-TA**). Robots serve one task at a given time, more than one robot is required to serve a task, and there are more than one round of assignations (dynamic environment).

5. **Multi-Task robots - Single-Robot tasks - Instantaneous Assignment** (**MT-SR-IA**). Robots serve more than one task at a given time, tasks are served by robots only once, and there is only one round of assignations (static environment).

6. **Multi-Task robots - Single-Robot tasks - Time-extended Assignment** (**MT-SR-TA**). Robots serve more than one task at a given time, tasks are served by robots only once, and there are more than one round of assignations (dynamic environment).

7. **Multi-Task robots - Multi-Robot tasks - Instantaneous Assignment** (**MT-MR-IA**). Robots serve more than one task at a given time, more than one robot is required to serve a task, and there is only one round of assignations (static environment).

8. **Multi-Task robots - Multi-Robot tasks - Time-extended Assignment** (**MT-MR-TA**). Robots serve more than one task at a given time, more than one robot is required to serve a task, and there are more than one round of assignations (dynamic environment).

There are no works reported in these categories: ST-MR-IA, MT-SR-IA, MT-SR-TA, and MT-MR-IA and for that these categories are not included in next sections.

3.2 Single-Task robots - Single-Robot tasks - Instantaneous Assignment (ST-SR-IA)

Lagoudakis et al. (2005) propose a theoretical study of a multi robot system. The tasks (points in the environment which must be visited by robots) are known previously by the robots, however, a robot only knows its own location and does not know the other robots' locations.The robots exchange bids for tasks in rounds. The bids are proportional to the distance between a robot and a task: the "smallest" bid is the winner. Finally, when there are no tasks left, each robot computes its shortest path for reaching those tasks and begins its visiting route. This work supposes a decentralized and communicated system because there is no auctioneer who announces the tasks: the robots exchange bids through messages. Nevertheless the system is not dynamic since all robots know the tasks' location and tasks are attended only after they were allocated. According to Gerkey and Matarić (2004), this is a ST-SR-IA problem.

3.3 Single-Task robots - Single-Robot tasks - Time-extended Assignment (ST-SR-TA)

Botelho and Alami (1999) propose one of the earliest works of multi-robot task allocation: M+. They show a study of a hybrid approach combining reasoning ability and reaction for a simulated multi-robot system. The tasks are boxes which are carried from their original locations, transported through defined paths and relocated

to their new locations by the robots. Tasks are divided and mapped in case of failure or cooperative exchange trading. A global planner is always required for tasks execution and a local snapshot of a changing environment is held by each robot. An overlap exists between the task currently executed and the plan for the task to be executed next in a *look ahead* way. The M+ protocol is defined by three behaviors: allocation (refines and assigns tasks according to election by the current context), cooperative reaction (invoked by the failing robot when there are flaws in the execution of the task, it updates the state of the world, it manages the exchange of information between robots, controls replanning and asks for help if needed) and task execution (distributively controls the synchronization between robots and actions). Note that cooperative reaction can reassign an unachieved task previously assigned and for that three key features are supposed: a decentralized system, a changing environment and explicit communication among robots. It is worth to remark that the failing robot never loses communication with the system, in fact, the robot itself asks for help whenever needed. In Figure 3.1 there is snapshot of the simulated environment. According to Gerkey and Matarić (2004), this MRS is a ST-SR-TA problem.

Matarić et al. (2003) use both simulated and real robots. The tasks are fire alarms that can be activated at unpredicted times and unknown locations. The robots must approach these alarms for turning them off. The overall coordination formulation is divided into three parts: bid feature that determines the ability to perform a task based on the state of the robot; task allocation mechanism that determines which robot should perform a particular task based on bidding; and robot controllers that determine appropriated actions for each robot based on the commitment of the current task of the robot.

Fire alarms are simulated using sound signals. The bids are proportional to the sound intensity of the fire alarm perceived by the robots and depending on the distance between the alarm and each robot. In this case, because of the unknown

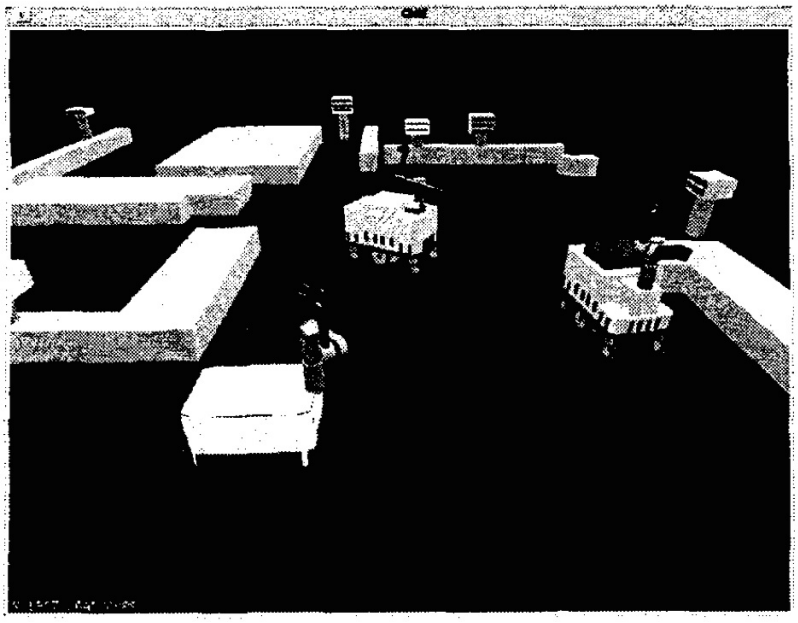


Figure 3.1: Simulated environment of M+. The robots move along lanes that connect "stations" where they pick-up and put-down boxes. From: Botelho and Alami (1999).

location of the alarms (tasks) and random time they activate, it is supposed a dynamic environment; since the robots bid for tasks and exchange those bids that means the scenario corresponds to a decentralized system; finally, communication among robots is a mandatory requirement for the exchange of bids. For the taxonomy given by Gerkey and Matarić (2004), this problem fits into the category of a ST-SR-TA problem.

Viguria et al. (2007) present another market-based work. In this case, the tasks are unknown points in an environment. There are two kinds of robots: auctioneers and bidders. In each auction a task is auctioned by only one auctioneer robot which has a token, meanwhile all other robots are bidders for that task. Later, the token is given to another robot which has more tasks to be auctioned or kept by the current robot. Tasks are discovered during run time. Since the tasks are unknown, this system is dynamic and there is explicit communication among robots for exchanging

bids. Also, this system is considered decentralized because there is no a unique robot assigning tasks. In the taxonomy given by Gerkey and Matarić (2004), this system is a ST-SR-TA problem.

Seow et al. (2010) propose a system where there are no mobile autonomous robots but agents. These agents reside in taxis which bid for customers (tasks) who are to be transported. The environment is a road network previously known by all the agents. A central announces a task's location to all available agents in the nearby area, then the available agents bid according to the distance between the taxi and the task location. A main feature of this work is that agents negotiate previously assigned tasks which are exchanged if agreed, thus the system is decentralized. This work has explicit communication, since all agents communicate decisions. Finally, the number of customers and their location is unknown, therefore, this system is also dynamic. This work is considered as a ST-SR-TA problem according to Gerkey and Matarić (2004).

Dahl et al. (2009) use a vacancy chain scheduling. One example of vacancy chain is in bureaucracy: an employee with high post retires leaving a vacancy filled with an employee with lower post than the first, then a second vacancy is created and the chain continues. This system uses indirect instead of explicit communication among robots. In this case, there are circuits with more than one robot each. In one edge of the circuits there is a provider of objects the robots must carry to another edge of the same circuit. Because the number of carried objects is unknown in the beginning, this system is considered dynamic. A schematic of the operation is seen in Figure 3.2. This research is ST-SR-TA problem, according to Gerkey and Mataric Gerkey and Matarić (2004).

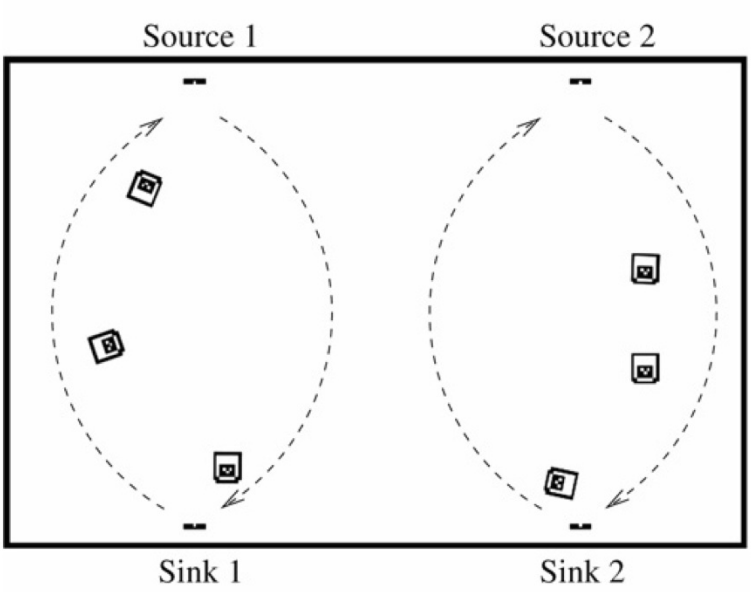


Figure 3.2: Vacancy chain simulated environment. The robots must carry objects (tasks) from sources to sinks. From: Dahl et al. (2009)

3.4 Single-Task robots - Multi-Robot tasks - Time-extended Assignment (ST-MR-TA)

Munoz-Meléndez et al. (2012) propose a multi-robot system that inhabit a divided environment. These divisions (tasks) must be processed (visited) by robots more than once. Visits are considered demands to accomplish a task. Each task can be visited by only one robot at a given time. The robots generate a transition matrix in which the Euclidean distances between available tasks are computed, as well as a state vector where a robot calculates the Euclidean distance to each available task. The matrix and the vector are multiplied giving as result a queue of sorted tasks for each robot. Then the robots bid for the tasks following the ordered queue. If a robot does not win any task, it will remain idle until the robot wins one available task. This work supposes a reliable explicit communication among robots for exchanging bids. Also, this system is decentralized because there is not a central manager. Finally,

the system is dynamic (stochastic) because the availability of a task is not known with certainty since the beginning and unavailable tasks are not considered in a bid round at a given time. In the taxonomy of Gerkey and Matarić (2004) this research is a ST-MR-TA problem.

Lenagh (2013) extends the work of Munoz-Meléndez et al. (2012) by adding the robots' batteries level to calculations. In this case, the location of tasks is unknown and are detected during run-time. As previously done in the work of Munoz-Meléndez et al. (2012), the robots generate a transition matrix in which the Euclidean distances between pairs of available tasks are computed and a state vector where a robot calculates the Euclidean distance to each available task. The matrix and the vector are multiplied giving as result a queue of sorted tasks for each robot. Then the robots bid for the tasks following the ordered queue. As this research is an extension to Munoz-Meléndez et al. (2012) has the same properties: explicit communication between robots, is decentralized and dynamic. As well, the taxonomy by Gerkey and Matarić (2004): this is a ST-MR-TA problem.

Miller et al. (2007) use muti-robot task allocation with a swarm-like system. A swarm has mobile entities (agents) which search and interact with objects of interest or targets (tasks). Since a single entity cannot achieve the whole interaction over the target, more entities are required. When a target is discovered, its existence, priority and location are communicated by pheromone. Other members of the swarm interact over the target until completion, then the members resume their search for targets.

The behaviors and capabilities of the agents of this system are: deploy (the agents are deployed upon request of a central manager, but the manager does not supervise the agents); search and discover (agents perform a blind search for targets, if a target is discovered it is marked with pheromone indicating priority); communicate (when an agent discovers a target, the agent must communicate the target's existence in a peer-to-peer manner to other agents within its communication range); task selection (one agent stores information about uncompleted tasks in a list and

chooses the tasks it will execute by establishing a path); execution of tasks (when reaching a task's location, the agents perform the required actions for achieving the task or share it with other agents and when its part is over, resume the search for targets).

The cited work has the following features: it is dynamic, since the agents do not know the tasks' location; there exists explicit communication even tough it is limited; and it is decentralized because the robots are autonomous and they have to discover the tasks. Note also that a single agent only performs a part of the task and it is analogous to the concept of demand used by Munoz-Meléndez et al. (2012). This is a swarm-like research, however, in the taxonomy by Gerkey and Matarić (2004), this system matches a ST-MR-TA problem.

The work of Dasgupta (2011) is similar to the work of Miller et al. (2007) that uses a swarm-like system. An object of interest or target is considered as a task. The robots must interact (discover, consume or carry to a nest) on tasks. The task is considered complete when a minimum of four robots have interacted with the task, analogous with the previous demand concept. Each robot uses three lists of tasks: allocated-task list (contains information of discovered tasks, it is updated during run-time, it is sorted according to some heuristics and the robot resumes search behavior if this list is empty); visited-task list (contains the tasks previously executed by the robot and these tasks must be ignored if rediscovered); completed-task list (contains closed tasks and those tasks must be ignored if rediscovered). Four heuristics are proposed. This work is also dynamic, decentralized and explicit communication exists between robots. As previously as with Miller et al. (2007) this research is a ST-MR-TA problem according to Gerkey and Matarić (2004).

3.5 Multi-Task robots - Multi-Robot tasks - Time-extended Assignment (MT-MR-TA)

Bullo et al. (2011) propose a theoretical study of algorithms and task allocation to dynamic routes. They also deal with movement restrictions for real robots or vehicles. Their main objective is not an optimal allocation of tasks but an optimal route for the robots given demands of visits created during run-time. They also deal with with these scenarios: explicit communication, without explicit communication between robots and time constraints policies. This research is very exhaustive. Gerkey and Matarić (2004) consider this work as a MT-MR-TA problem.

3.6 Comparison of related work

We summarize all previously mentioned works in Tables 3.1 and 3.2 in some comparable settings defined in Chapter 2. Note for example, similarities in the use of simulated robots and tasks. Almost every work reports one demand per task (single-robot task), however, there are few works (such as Miller et al. (2007), Munoz-Meléndez et al. (2012) and our research) that address problems where a task is divided into demands (multi-robot task).

Note also that Lagoudakis et al. (2005) use an instant-assignment scheme by assigning the tasks only once, meanwhile, all the other works consider a time-extended assignation of tasks meaning that their environments are dynamic. Also, note how all work which report the use of explicit communication always considered it flawless, the aspect where the main difference with our research relies: we assume that any failure that causes the loss of communication in a MRS results in a task reassignation.

Botelho and Alami (1999) deals with failures if a robot can not achieve its task, however, the robot never loses communication and is able to ask for help through the communication system. While we look for tasks completion, other works assume a reliable communication and look for an efficient task allocation algorithm. Furthermore, we assume that our mechanism might complement existing task allocation algorithms, so we focus our research in the communication field by analyzing different cases of failures as we expose in Chapter 5. The Tables shown are by no mean an exhaustive and complete comparison.

3.7 Summary

- We revised the most significant related work to our research according to the taxonomy introduced in Section 2.2.
- There are key differences in related work with our research: the communications field is poorly addressed or not addressed at all. We consider failures and task reassignation when robots fail and lose communication with the system.
- We presented a comparative chart with key features in order to highlight our problems and contributions.

Authors, year	System: centralized or decentralized	ROBOTS			TASKS			COMMUNICATION		Area of the environment	Class in the taxonomy by Gerkey and Matarić (2004)	Results
		Robots: simulated, physical or theoretical study	Size of robots by length: small (¡0.30m), medium (¡2m) or big (¿2m)	Number of robots	Task assignment	Number of tasks	Number of demands per task	Requirements: Reliable or unreliable	Among robots			
Bullo et al. (2011)	Decentralized	Theoretical algorithmic study	Not applicable	m vehicles	Dynamic	k (called services)	1	Not mentioned	Both: with and without explicit	Not reported	MT-MR-TA	No results reported but a study to calculate the optimal route for reaching places that arise in runtime (demands) which also must serve a task
Lenagh (2013)	Decentralized	Simulated	Medium	Groups of 5, 10, 15 and 20 robots	Dynamic	6, 12, 18 and 24	At least 3, at most 5	Reliable	Yes	$400m^2$ (20 x 20 m)	ST-MR-TA	Comparison with a greedy Algorithm, Hungarian and repeated bids of task allocation
Dasgupta (2011)	Decentralized	Simulated	Small	Experiments with 9, 18 and 27 robots	Dynamic	20, 24	At least 4	Not mentioned	Yes	$12.25m^2$	ST-MR-TA	Results of a cooperative foraging scheme and comparison with 4 different heuristics
Seow et al. (2010)	Decentralized	Simulated	Big	Fleet: 1000 and negociation groups of 5, 10, 15 and 20 agents	Dynamic	Not mentioned	1	High speed reliable	Yes	$150km^2$ (15 x 10 km)	ST-SR-TA	Comparison against a centralized system of task assignment
Botelho and Alami (1999)	Decentralized	Simulated	Not mentioned	3	Dynamic	10	1	Not mentioned	Yes	Not mentioned	ST-SR-TA	Experiments of docking stations connected to rails which the robots move containers from one station to another
Lagoudakis et al. (2005)	Decentralized	Theoretical study of routes	Not mentioned	n robots	Static	Not mentioned	1	Not mentioned	Yes	Not applicable	ST-SR-IA	The tasks' location is known and the number of robots but not their location; the robots bid for the all the unassigned tasks and when assigned, a route is drawn to reach the tasks' locations

Table 3.1: Related work comparison

Authors, year	System: centralized or decentralized	ROBOTS			TASKS			COMMUNICATION		Area of the environment	Class in the taxonomy by Gerkey and Matarić (2004)	Results
		Robots: simulated or physical or theoretical study	Size of robots by length: small (¡0.30m), medium (¡2m) or big (¿2m)	Number of robots	Task assignment	Number of tasks	Number of demands per task	Requirements: Reliable or unreliable	Among robots			
Matarić et al. (2003)	Decentralized	Simulated and physical	Medium	10 robots simulated and unknown number of real robots	Dynamic	Not mentioned	Not mentioned	Not mentioned	Yes	100 grids (10x10) in simulation and not mentioned the real environment dimensions	ST-SR-TA	Location of tasks are not known at start time, tasks arise at runtime and robots bid a estimated (heuristic) of the distance of the task. Comparison to four strategies by combining two axes: commitment and coordination
Miller et al. (2007)	Decentralized	Simulated	Not mentioned	18 robots	Dynamic	20	At least 3, 4 for confirmation	Not mentioned	Yes, limited	2500 grids (50x50)	ST-MR-TA	Comparisons among four heuristics for dynamic allocation of tasks with a minimum number of demands
Dahl et al. (2009)	Decentralized	Simulated	Medium	5 and 6 robots	Dynamic	Not mentioned	1	Not mentioned	No	$96m^2$ (12 x 8 meters)	ST-SR-TA	There exist two chains with of 3 robots each: a high priority and low priority for a basis case, other two experiments removing a robot from each chain and reports convergence time in the later cases
Viguria et al. (2007)	Decentralized	Simulated	Medium	Experiments with 3 and 5 robots	Dynamic	Experiments with 3, 5, 7, 9, 15, 20, 30 and 40 tasks	1	Reliable	Yes	$1,000,000m^2$ (1000 x 1000 meters)	ST-SR-TA	Compares 3 algorithms: *no local plan* against 2 with planning which are closer to the optimum than the first
Our research	**Centralized in a server or decentralized on multiple robots if needed**	**Simulated**	**Medium (robot Koala Silver)**	**Robots simulated 3, 6, 9 and 12**	**Dynamic**	**Experiments with 4, 8, 12 and 16**	**3**	**Unreliable**	**Yes**	$400m^2$ **simulated (20x20 meters)**	**ST-MR-TA**	**We use flawless simulations as a basis and compare with three cases of failures: robots fail without recover; robots fail with recover; and the central server fails with recover**

Table 3.2: Related work comparison (continued)

Chapter 4

MRS: formal definition and first Algorithm

This chapter begins by describing formally our system. It includes the **formal settings** of the environment and a formal definition of sets of tasks and robots in Section 4.1. In Section 4.2, we present the **technical features** of the environment, robots and tasks, as well as the robots' sensor capabilities. We introduce our first Algorithm in Section 4.3. A problem encountered in the first batch of simulations: *collision avoidance* among robots, and the **solution to collisions** is presented in Section 4.4. Finally, in Section 4.5 we summarize the content of the current chapter.

4.1 Formal definition of the problem

The formal definition of the problem states as following:

1. Let $\varepsilon \subset \mathbb{R}^2$ be an **environment** delimited and divided into m partitions not necessarily of equal size.
2. Let $A = \{a_i : 3 \leq i \leq n\}$ be a set of n processors: 1 fixed (**server**) and *n-1* mobile (**robots**) which move within ε.
3. Each mobile processor or robot a_i has associated these attributes: **position** at

a given time t, $p_{a_i}(t) \in \varepsilon$; **level of its battery** at a given time t, $b_{a_i}(t)$ with $0 < b_{a_i}(t) \leq 100$; and a unique **identifier** id_{a_i}.

4. Let $T = \{\tau_k : 1 \leq k \leq m\}$ be a set of m **tasks** which are abstractions of the partitions of the environment ε.

5. Each task τ_k has associated these attributes: a unique **identifier** id_{τ_k}; its **position** in the environment, $p_{\tau_k} \in \varepsilon$; a number of demands that compose the task, $nd_{\tau_k} \in \mathbb{Z}$; **number of demands** of the task that have been served, $sd_{\tau_k} \in \mathbb{Z}$; and a boolean value $disp_{\tau_k}$ that denotes if the task is **available** or **not**. A robot is able to serve a task once only.

6. Let $T_{open} = \{\tau_k \in T : sd\tau_k < nd_{\tau_k}\}$ be the set of incomplete tasks or **open tasks**.

7. Let $T_{closed} = T \setminus T_{open}$ be the set of complete tasks or **closed tasks**.

8. Let $T_{unavailable} = \{\tau_k \in T_{open} : disp_{\tau_k} = FALSE\}$ be the set of **unavailable tasks**.

9. Let $d_{ik}(t) = \| p_{a_i}(t) - p_{\tau_k} \|$ be the Euclidean distance between robot a_i and task $\tau_k \in T$ in a given time t.

10. There is a **mechanism of explicit communication** for passing **messages** between robots and server, and among robots, which **may suffer of failures** during run time.

11. The stop criterion is $T_{open} = \emptyset$, *i.e.*, there is **not any open task** or a **maximum run time** is reached (*operation time* $\geq$ *maximum run time*).

12. Task allocation is considered **dynamic** if tasks have a due time, if tasks are created during run time or if communication failures might happen, among other causes, as stated in Section 2.5.

13. **Starting time** and **finishing time** correspond to the *machine time* when all the simulations begin and end respectively.

14. **Idle time**$_{a_i}$ is any period of time when robot a_i is not serving a task and it is waiting for an available task to serve. This period of time is bounded by two events: when the task allocation Algorithm indicates to a robot to begin its current idle time (time(start current idle period)); and when the task allocation Algorithm assigns a task to the robot (time(finish current idle period)). At the end of the simulation the server accumulates all individual times into a global unique value.

15. $Avoidance_{a_i}$ is any period of time when robot a_i detours from its original path in order to avoid a collision with other robots. Its boundaries are two events: when an object in front of the robot is detected (time(start current avoidance request)); and when the avoidance routine is finished (time(finish current avoidance request)). As with idle time, this time is accumulated at the end of the simulation by the *server* into a global unique value.

4.2 Simulated environment

The simulation environment Webots© by Cyberbotics meets all the technical requirements mentioned in this section.

4.2.1 Dimensions and features

In this section we provide technical descriptions of the environment, robots and tasks, as they are represented in the Webots© simulation environment. Figure 4.1 illustrates the dimensions of these elements.

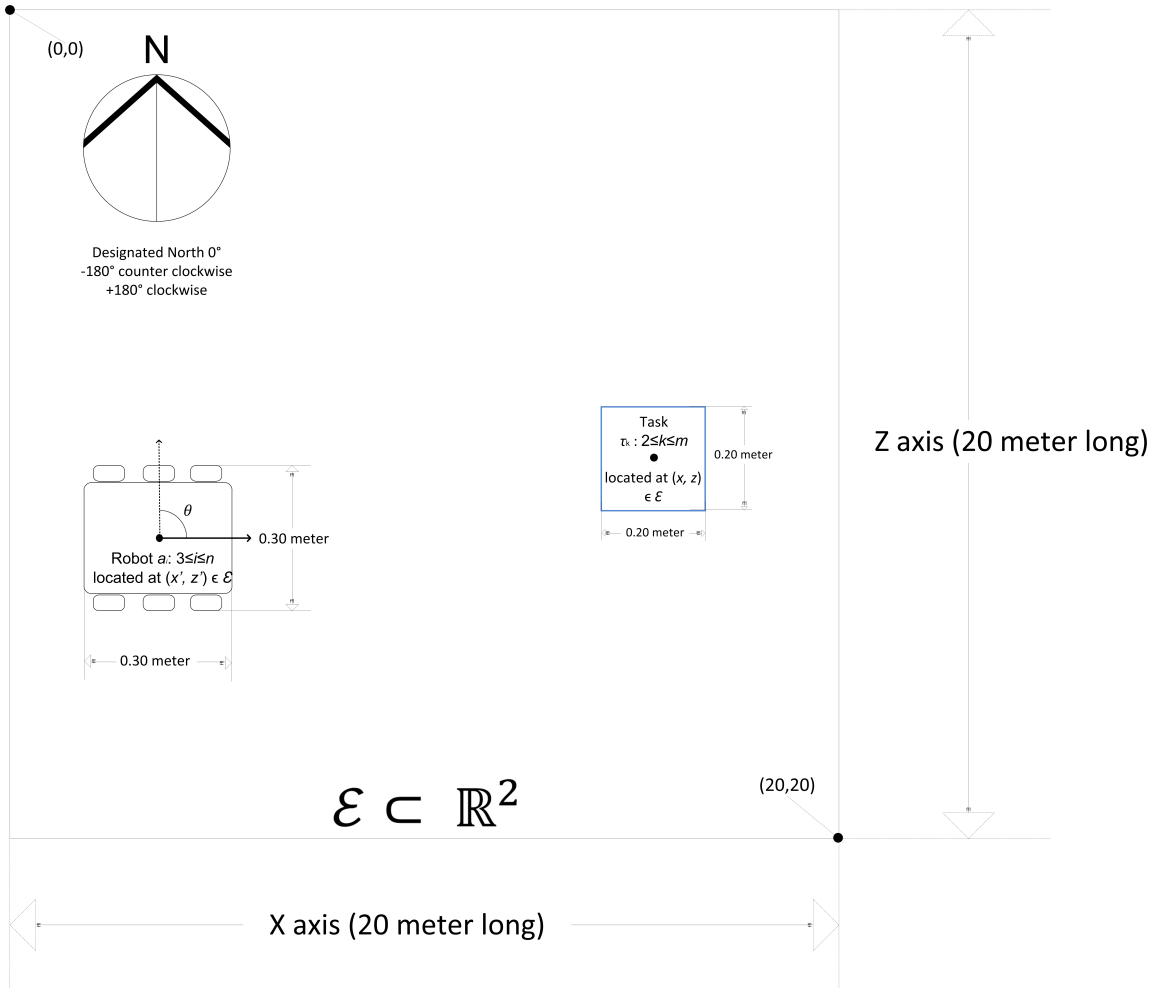


Figure 4.1: Dimensions of environment ε (20×20 meters, $400m^2$), robot a_i textcolorao(english)(0.30×0.30 meters, $0.09m^2$) and task τ_k (0.20×0.20 meters, $0.04m^2$). The robots detect with a compass the North Pole located in the upper side of the environment ε for orientation purposes

Environment

The simulated environment ε is the place where all the mobile robots are deployed along with the tasks. This environment has the following features set for all the possible scenarios involved in our experiments:

1. From now on, all the distance measures are given in meters unless stated otherwise.

2. The environment ε is a two-dimensional square with 20 meter long per side and it is located over the axes x and z.

3. The x axis has these properties: $x \in \mathbb{R}$, bounded in the [0,20] interval. The z axis is perpendicular to, and has the same properties of the x axis. The (0,0) coordinates lay on the upper left corner of the environment ε and the (20,20) coordinates lay on the bottom right corner of the environment. The Webots© simulation environment uses three axes thus simulations are done in three dimensions, however, the plane corresponding to the surface over which the robots move uses axes x and z, while the axis y corresponds to "up" and "down" the mentioned surface (see Figure 4.2). The origin of positive coordinates is just a convention defined by us.

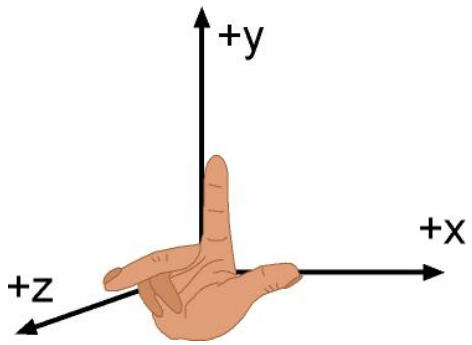


Figure 4.2: Right hand rule for simulated environments. The y axis is not used in our simulated environments. From https://share.ehs.uen.org/node/8473

4. There is a designated north pole located in the upper side of the environment ε.

5. The environment ε has no physical delimitations such as walls, therefore it is considered non-convex, *i.e.*, the robots are able to go beyond the limits of

the environment, however, a task is never located outside the environment's boundaries.

Robots

The set A contains the **server** and the **mobile robots**. The robots have the following features:

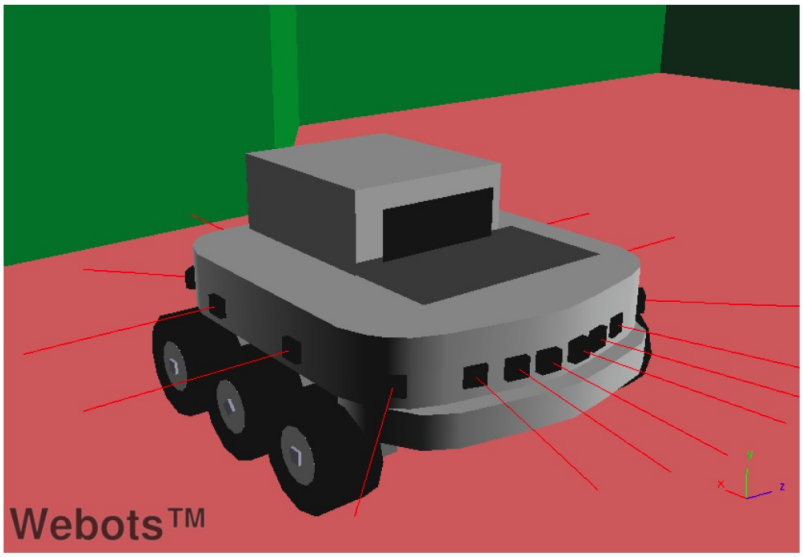


Figure 4.3: Simulated Koala robot with built-in infra-red sensors. From https://www.cyberbotics.com/guide/section3.5.php

1. All mobile processors are shaped according to the robot model Koala created by *The K-Team*.

2. The footprint of the Koala robot is 0.30×0.30 meters (see Figure 4.1), reported by the manufacturer, (medium size according to the comparison in Tables 3.1 and 3.2.

3. The Koala robot is a 6-wheeled vehicle with differential drive and two degrees of freedom, therefore, the Koala robot is considered a non-holonomic vehicle.

The radius of all wheels is 0.0425 meters (4.25 centimeters).

4. The Koala robot has 16 built-in infra-red proximity sensors spread on the left and right sides and on the front and rear, see Figure 4.3. For this research, we only use 8 out of 16 infra-red sensors located in the front of the robot (4 at each hemisphere) with an approximate scope of 1 meter and field of view of 60 degrees of opening detection as shown in Figure 4.4. As stated before, the environment ε does not have static obstacles, however, the robots themselves appear as obstacles for other robots, thus, these infra-red sensors are used for detecting partners.

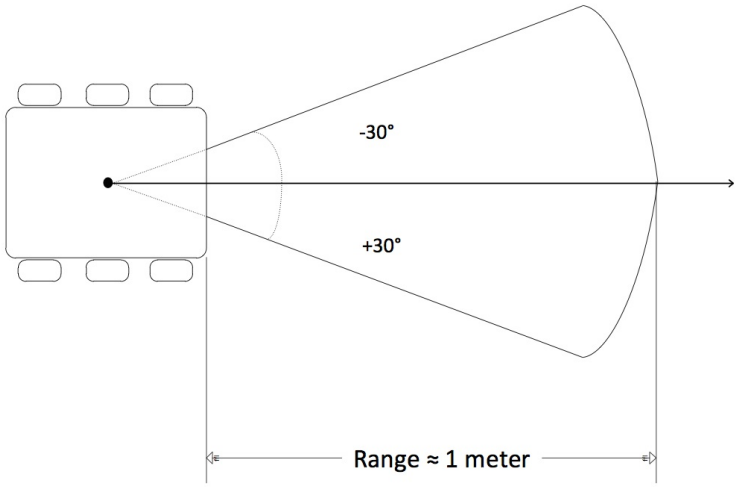


Figure 4.4: Dimensions and field of view of the 8 infra-red sensors located in the front of the robot that are used for detecting partners: opening of about 60 degrees and scope of approximately 1 meter

5. All the robots have two communication devices: Receiver and Emitter: their function is receive and send messages, respectively, in a full-duplex communication. They are set as Radio type, and originally set with unlimited range.

6. All mobile robots have two location devices: Compass and GPS. These devices are sensors of orientation and localization, respectively.

Central server

- As previously stated, there exists a server. This processor is also a Koala robot and it is located, for our purposes, outside the environment ε. It will remain static at any time. Its functions are: coordinate and assign tasks to the robots, reassign tasks if necessary, keep all records of progress, and send the exit signal to the robots, thus, making the MRS a **centralized system**.

Tasks

Finally, the tasks in set T have all the following features:

1. All the tasks have static coordinate points (x,z) in the environment ε. In order to be reached by robots, the tasks are expanded to squares of 0.20 meters per side with the points being in the center of the squares.
2. A task is considered as visited by a robot if the GPS of the robot reports a coordinate within the task's square, see Figure 4.5.

4.3 Task allocation Algorithm

Our first particular objective requires the implementation of a task-allocation Algorithm. Since this is not the core of our research, we decided to implement a greedy Algorithm. In this section we define two mandatory parts for proposing our Algorithm: the data structures and the Algorithm itself.

4.3.1 Data structures

As we formally defined in Section 4.1, each task is composed by specific attributes. We propose the use of a Table of m rows and a column of the required data

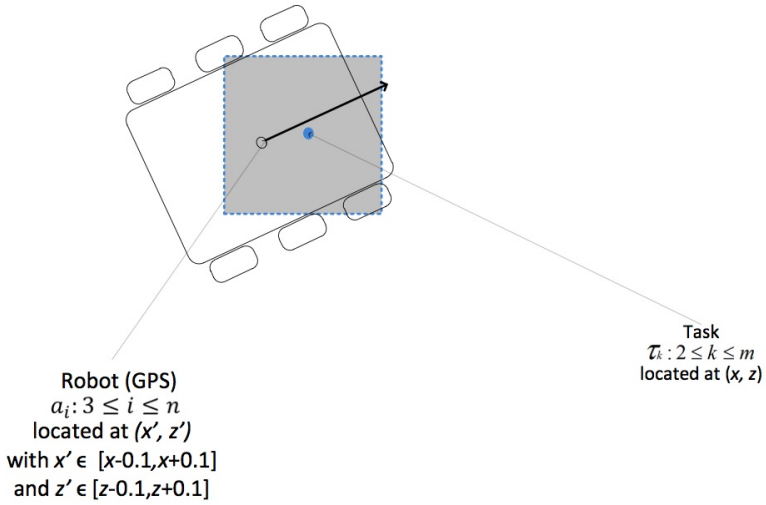


Figure 4.5: The robot perceives a task if $x' \in [x-0.1, x+0.1]$ and $y' \in [y-0.1, y+0.1]$, where (x,y) are the coordinates of task τ_k and where (x', y') represent the location of the robot

type for each of the attribute composing a task. The structure is illustrated in Table 4.1.

For the robots we also propose a Table of n rows for the data structure following the formal definition of Section 4.1 with a slightly difference: since the robots' data is constantly updated, we want to know which tasks have been served and which task is currently served by a robot, so we include these two extra attributes: **record of attended tasks** and **currently attended task**. See Table 4.2.

Notes:

Attribute	Name in data structure	Datatype
Identifier	id	integer from 0 through *m-1*
Position (x coordinate)	coord_x	real in the [0,20] interval
Position (z coordinate)	coord_z	real in the [0,20] interval
Number of demands:	nd	integer from 0 through *number of demands*
Attended demands:	ad	integer from 0 through *number of demands*
Available task	disp	boolean value

Table 4.1: Data structure for storing data related to tasks

Attribute	Name in data structure	Datatype
Identifier	id	integer from 0 through n-2
Position (x coordinate)	pos_x	real
Position (z coordinate)	pos_z	real
Battery level	bat	real in the (0,100] interval (%)
Record of attended tasks	record	vector of m boolean values
Identifier of currently attended task	current_task	integer from 0 through *m-1*

Table 4.2: Data structure for storing data related to robots

1. The identifier only reaches the number of processors minus one (*n-1*) because the server requires to store data related to mobile robots only.

2. The *record of server tasks* corresponding to the number of task served by the robot is set to one (1) if the task has been served, meanwhile if the task has not been served the entry is set to zero (0).

Finally, we require a calculation matrix in order to decide which task assign to which robot. This matrix is updated each time a task is assigned. The matrix has *n-1* rows, corresponding to the number of mobile robots and m columns corresponding to the number of tasks. Each entry stores the Euclidean distance between mobile

robot a_i and task τ_k divided by the battery level of robot a_i in a given time t, thus the data type is a positive real number. See Table 4.3.

Robot \ Task	τ_1	τ_2	...	τ_{k-1}	τ_k
a_1	2.53	83.26	...	10.04	0.28
a_2	14.11	70.03	...	92.01	84.09
...	...	...	...	...	...
a_{i-1}	86.27	78.17	...	12.08	30.62
a_i	34.67	78.84	...	10.66	60.14

Table 4.3: Matrix of calculations

Note that the battery level **must be greater than zero** at any time for avoiding a **division-by-zero** error. This condition was defined in Section 4.1

4.3.2 The first Algorithm

We propose a greedy Algorithm for task allocation based on assigning the task corresponding to the minimum value of the robot's row in the matrix of calculations. Note that no failures have been considered so far, thus, the only stop criterion is that all tasks must be closed. See Algorithm 4.1.

On the other hand, the Algorithm (4.2) which operates in each mobile robot is very simple: the mobile robots will only visit the tasks' locations the server has assigned to them and request new tasks as they fulfill the tasks they have visited.

It is worth to mention that the mobile robots use a local planner for them to move. This planner (function) receives the coordinates of the assigned task, the robot's current coordinate (from the GPS) and orientation (from the Compass) in order to calculate a straight path to the task's location, which is updated as the robot moves toward its goal. The planner also considers the infra-red sensors as we stated

previously in Subsection 4.2.1.2 for detecting other robots, and, if detected, the planner along with the protocol stated in Subsection 5.2.3 coordinates with other robots for avoiding collisions. See Section 4.4 for a detailed explanation in this matter.

Now we define the three required metrics used by our algorithms:

1. **Completion time** is the overall time of the program execution in seconds. This metric is measured by the server only. See formula 4.1.

$$Completion\ time = finishing\ time - starting\ time \tag{4.1}$$

(Unit: seconds)

2. **Total distance** is the cumulative Euclidean distance traveled by each mobile robot in the MRS. At the beginning, this metric is set to zero (0) for each robot. Every time a task is assigned to a mobile robot, the robot travels in the environment from its current position to the destination position where the task is located, the robot measures that distance using the formula 4.2.

$$d_{ik}(t) = d_{ik}(t) + \parallel p_{a_i}(t) - p_{\tau_k} \parallel \tag{4.2}$$

(Unit: meters)

When all tasks are closed the mobile robots report their traveled $textitdistance_{a_i}$ to the server which collects in the overall **total distance** metric as seen in formula 4.3.

$$Total\ distance = \sum_{i=1}^{n-1} distance_{a_i} \tag{4.3}$$

(Unit: meters)

3. **Idle time** is the cumulative time of all the moments a *mobile robot* enters in *idle* state. At the beginning, this metric is set to zero (0) for each robot. Each robot calculates its own *idle time* calculated with formula 4.4.

$$idle\ time_{a_i} = idle\ time_{a_i} + time(finish\ current\ idle\ period) - time(start\ current\ idle\ period) \tag{4.4}$$

(Unit: seconds)

As with total distance, the mobile robots report their $idle\ time_{a_i}$ to the server which calculates the overall **idle time** metric with the formula 4.5

$$Idle\ time = \sum_{i=1}^{n-1} idle\ time_{a_i} \tag{4.5}$$

(Unit: seconds)

4.4 Collisions cases

We encountered some issues when testing our first Algorithm that have to be solved before dealing with communication issues: collisions might happen among robots. Then a solution to this problem was designed and implemented. For practical issues, we consider two kinds of detection for robots derived from their obstacle detection capabilities, as stated in Subsection 4.2.1.2 that are described below.

Algorithm 4.1: Greedy Algorithm for task allocation: server

Input : Positions (coordinates) of *tasks* (p_{τ_k}) and *robots* (p_{a_i})

Output: Set of closed tasks (T_{closed}) and metrics of *Completion time*, *Total distance* and *Idle time*

```
1  begin
2      Table of tasks is created with coordinates of tasks p_τk, all number of demands
         (nd_τk) are set to a constant c, all served demands (sd_τk) set to 0 and all
         availability flags (disp_τk) set to TRUE;
3      Table of robots is created with coordinates of robots p_ai, all battery levels (b_ai) are
         set to 100, all entries of record vector set to 0 and all current tasks (current_task_ai)
         set to 0;
4      T_open = T;
5      while T_open ≠ ∅ do
6          Wait until a robot a_i requests a new task τ_k;
7          Update the Matrix of calculations with the distances from robots to tasks
             (d_ik(t)) at the current time;
8          Find the minimum value of row i corresponding to a column of an unassigned
             task τ_k and assign the task τ_k to robot a_i;
9          Update disp_k = 0;
10         Update current_task_ai = k;
11         if there are no available tasks then
12             Set state of robot a_i is idle;
13         else
14             Robot a_i fulfills its assigned task;
15             Update coordinates p_ai;
16             Update battery level b_ai;
17             Update record vector_ai corresponding to task τ_k to 1;
18             Increment nd_τk and decrement sd_τk ;
19             if nd_τk =0 and ad_τk =c then
20                 Move τ_k from T_open to T_closed;
21             else
22                 Set disp_τk to 1;
23             end
24         end
25     end
26     Send termination signal to all robots a_i;
27     Receive partial metrics of distance and idle time from robots;
28     Sum partial metrics of distance and Idle time into Total distance and Idle time;
29     Calculate Completion time;
30 end
```

Algorithm 4.2: Greedy Algorithm for task allocation: mobile robots

```
   Input  : Positions (coordinates) of tasks (p_{τ_k})
   Output: partial metrics of distance and idle time
 1 begin
 2     Set distance to 0;
 3     Set idle time to 0;
 4     Receive initial task τ_k from server;
 5     while NOT termination signal received do
 6         if task τ_k received then
 7             Move towards p_{τ_k};
 8             Fulfill τ_k;
 9             Request new task to server;
10         else
11             Enter idle state;
12         end
13     end
14     Send partial metrics of distance and idle time to server;
15 end
```

4.4.1 Single perception

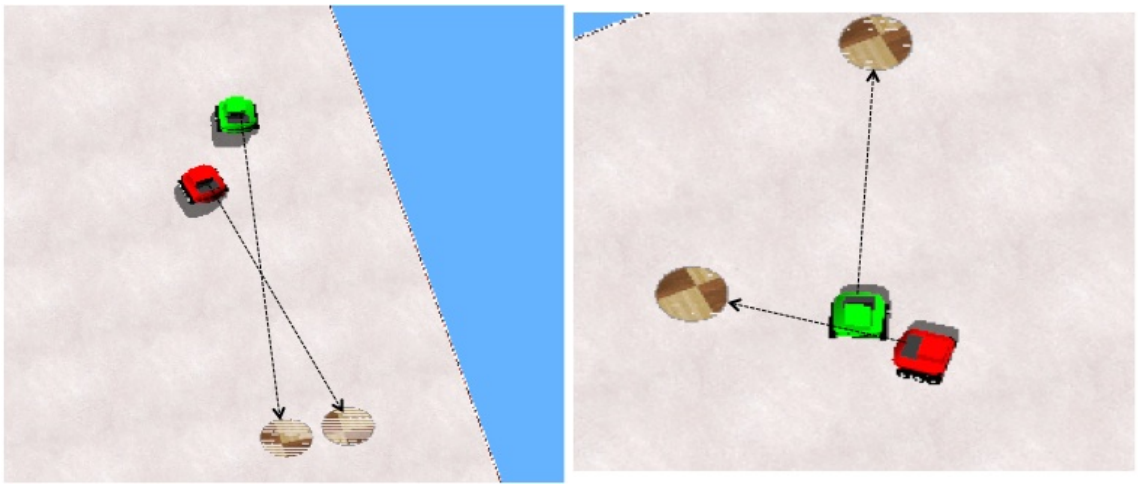
Single perception means only one robot detects (detecting robot) other robots (detected robots). If a detecting robot detects a partner, it sends a message in a restricted range asking for position and course (orientation) of its in-range neighbors, it waits during some time (wait-for-a-response time) for them to respond. In return, the detected robots send an acknowledge message responding the requested data if they are alive, then the detecting robot calculates probable collisions with all near detected robots and it decides if it has to surround its partners or stop for letting them pass. In the case the wait-for-a-response time is over, the perceiving robot surrounds the obstacle, assuming the other robots are dead, idle or have finished all their tasks and their program has already exited. From these kinds of perception we can identify three cases:

1. **Catch up collision**: if a detecting robot approaches detected robots from behind (the difference of their orientation angles is in the range from -30 to

30 degrees), the detecting robot stops for a while allowing the other robots to move away. See Figure 4.6(a).

2. **Lateral collision**: if a detecting robot approaches detected robots from their left or right side (the difference of their orientation angles is in the range from -150 to -30 degrees for left side, and from 30 to 150 degrees for right side), the detecting robot also **stops** for a while allowing the other robots to move away. See Figure 4.6(b).

3. **Default case**: finally, if a detecting robot does not receive response because the detected robots have finished their assigned tasks or have failed, the detecting robot surrounds the detected obstacles without transmitting more messages.



(a) Catch up

(b) Lateral collision

Figure 4.6: Single perception cases: (a) Catch up. The green robot detects and asks for its location and orientation of the red robot. The green robot determines it has to stop. (b) Lateral collision. The red robot detects and asks for its location and orientation of the green robot. The red robot determines it has to stop. Lines and arrows represent robots' paths, whereas circles represent their goals (tasks)

4.4.2 Multiple perception

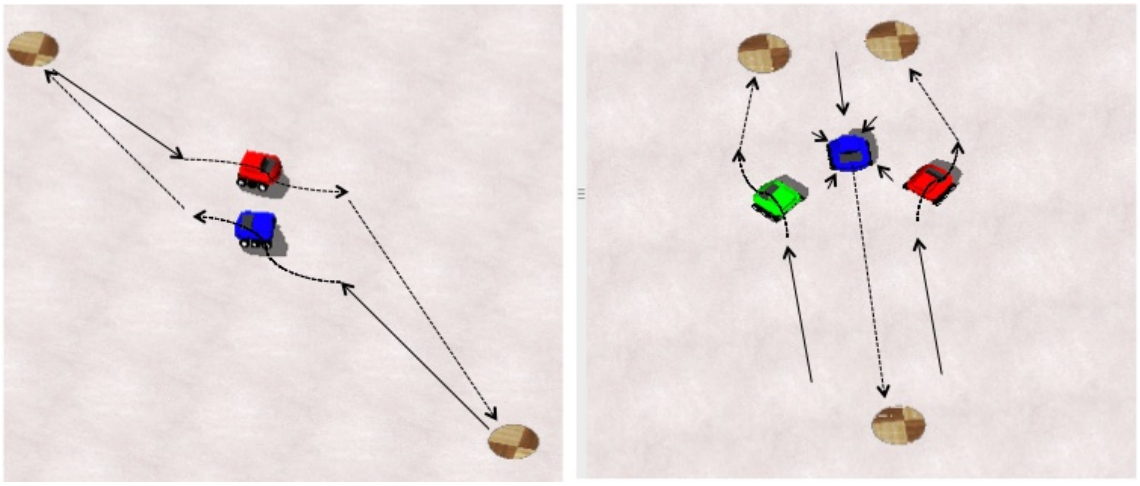
On the other hand, **multiple perception** involves more than one detecting robot, that means they detect each other in a particular case of frontal collision. This kind of perception involves two more steps of negotiation between robots for coordination: as in single perception, one robot detects parters and starts the negotiation asking for position and course of its neighbors; they respond the requested data if still no detection has happened, or requests position and course of their neighbors otherwise; then, they coordinate with two more messages asking for a detour or stop. For this kind of perception, the difference of their orientation angles is in the ranges from -180 to -150 degrees, and from 150 to 180 degrees. We deal here with two cases:

1. **Multiple perception with two robots**: the detecting robots coordinate their movements in order to surround each other on the same side detour. See Figure 4.7(a).
2. **Multiple perception with more than two robots**: the detecting robots coordinate their movements in such way that some robots (the least number in some flow direction) have to stop meanwhile the robots in counterflow surround the robots that have stopped. Say one robot goes left and two robots go right, the robot going left has to stop, meanwhile the robots going right surround the stopped robot. See Figure 4.7(b).

Note that this kind of perception requires a mandatory negotiation between at least two *alive* robots, otherwise, this situation is considered as single perception even though a frontal collision is foreseen.

4.4.3 A new metric

We have defined previously three metrics in order to record the overall progress, however, considering that the time required for avoiding collisions might affect all



(a) Catch up (b) Lateral collision

Figure 4.7: Multiple perception cases: (a) Two robots. Two robots detect each other and negotiate the side turn. (b) More than two robots. Three robots detect each other and negotiate that the blue robot must stop until the green and red robots have surrounded the blue one. Lines and arrows represent robots' paths, whereas circles represent their goals (tasks)

metrics: the *Completion time, Total distance* and *Idle time* we need to measure the time invested by robot to solve collisions. Thus, we propose a new metric called **avoidance time** given by formula 4.6.

$$avoidance_{a_i} = avoidance_{a_i} + time(finish\ current\ avoidance\ request) - time(start\ current\ avoidance\ request) \tag{4.6}$$

(Unit: **seconds**)

As with other metrics, the *mobile robots* start with the *avoidance time* set to zero (0) and by the end of the simulation, they report their *avoidance time*$_{a_i}$ to the *server* which calculates the overall **avoidance time** metric with the formula 4.7

$$Avoidance\ time = \sum_{i=1}^{n-1} avoidance\ time_{a_i} \tag{4.7}$$

(Unit: **seconds**)

4.5 Chapter summary

- We stated all previous properties of the environment, robots and tasks in Section 4.1.
- Our robots are equipped with: infra-red sensors for detecting other robots; GPS for location of the robot itself and detection of the tasks' area; and compass for orientation. Also, the robots have a transmitter and receiver radios in order to establish the communication mechanism. (Section 4.2).
- We designed a greedy Algorithm for Task Allocation. The description includes: data structures; metrics; and we have also described the path planner in Section 4.3.
- We studied and solved five cases of collisions avoidance: catch up; lateral collision; frontal with two robots; frontal with more than two robots; and the default case. The last case is the only one for which there is not any exchange of messages, in the rest of cases the message exchange is mandatory (Section 2.4).

Chapter 5

Extensions for failure recovery

The development of mechanisms for dealing with communication failures in MRTA problems is the main research point of this work. We classified the expected failures in Subsection 2.1.4. This chapter describes the proposed solution to the *communication problem* by defining first which cases of robots' failures might happen in our work in Section 5.1. Then we propose solutions by establishing a new Algorithm and an *ad hoc* communication protocol among robots for dealing with the failures; also an information exchange scheme in Section 5.2 is presented. We close by highlighting the main ideas of this chapter in Section 5.3.

5.1 Cases of failures proposed

We begin this section by reminding the formal definition of the problem given in Section 4.1 which establishes the need of explicit communication in MRS, communication which may suffer from failures. Also, we identify two kinds of processors: the stationary server and the mobile robots, both of them have one receiver and one emitter in order to exchange messages. Below we list the main assumptions, in terms of communication, for our MRS.

5.1.1 Assumptions

1. At the beginning of any deployment of the MRS, all processors (server and mobile robots) are fully functional because a human supervisor has prepared all the equipment previously.

2. A human supervisor is always aware and is located next to the server, thus, if a failure of server happens, the supervisor can reactivate the server in order to generate a global report. However, the human supervisor is unable to enter into the environment, meaning that the human supervisor can not control any robot during run time.

3. All communications are flawlessly working in order to start operations.

4. Neither the server nor the mobile robots know the number of tasks or the number of robots in the system before starting operations. We consider our mechanism scalable so the number of robots and tasks is different from environment to environment. The number of robots is known until there is an exchange of messages indicating starts of operations. The tasks' data is logically registered in a file held by the server.

5. The processors know their identifier when they wake up and use a unique channel related to that identifier for receiving messages through unicast method, or though broadcast in any channel. The processors are capable of emitting messages in any channel by unicast method, or by broadcasting messages in all channels at once.

6. We define **start operations** as the time when all robots enter in operation mode.

7. The total amount of mobile robots can decrease but it cannot increase. The total amount of robots equals the number of mobile robots reported at the

start of operations. Also, the mobile robots must report their initial location in the environment.

8. The mobile robots and the server must exchange data whenever an update has been made. The required data are the Table of tasks and the Table of robots (see Subsection 4.3.1).

9. There exists a reciprocal mechanism in the server and mobile robots for checking if their counterpart is operational: every 30 seconds the server will send a hello message in broadcast and the operational robots will answer in unicast to the server with a hello message ACKnowledge which includes their current location in the environment and battery level (see Subsection 5.2.3).

10. We say the status of a mobile robot is **dead** when it fails and does not recover; **asleep** if it recovers eventually during run time; **finished** if it has accomplished all its possible tasks; **idle** if there are no available tasks for it to be assigned; or else, **alive**. On the other hand, the server has the status of **alive** or **asleep**.

11. We use the term **maximum run time** introduced in Section 4.1. Note that this stop criterion must last longer than completion time, in order to allow to mobile robots to accomplish as many tasks as possible. The server has the function of calculating the maximum run time and send a termination program message to all active robots in the system.

12. We define **operation time line** as a period of time bounded by two events: a MRS starts operations and ends, either way, by closing all tasks or a maximum run time is reached. In both cases, the metric completion time equals the time that lasts the operation time line for the system being.

13. If there is a change in the environment, it triggers an **instant task assignment round**. Such events are, for example: when a robot just finished an assigned task with a number of demands left (*i.e.*, the task is still open); and when a

robot fails leaves an open task that must be reassigned. Note that during the first assignment round always there are at least *n-1* available robots and at least m open tasks.

14. From now on, for short, we will call **robots** to mobile robots and **server** to the static processor. Also, the receiver and the transmitter will be referred as Rx and Tx, respectively.

5.1.2 Single cases of failure

The cases we study in this subsection are called **single**, *i.e.*, one or several failures of the same type (see cases in Section 1.4) might happen during a simulation, however more than one case of failure can not happen simultaneously in a simulation environment. Multiple failures are not currently addressed and we explore some ideas to deal with them in the future (see Section 7.4). The cases we deal with in this research work are detailed below.

1. **Robots failure without recovery**. The first case we study happens when a robot ceases all communications with its partners or with the server, and communication is **never resumed**. The server takes into account 3 consecutive unanswered hello packets (90 seconds or more without communication) by a robot.

2. **Robots failure with recovery**. As with failure without recovery, a robot stops responding to 3 consecutive hello packets(90 seconds or more without communication is elapsed). However, in this case the robot will resume responding to hello packets at some point of the operation time line.

3. **Server failure with recovery**. In this case, the robots take into account if the server ceases to broadcast 3 hello packets (send messages for a period of 90 seconds or more). Note it is assumed the server will recover at some point of the

operation time line since human intervention is available as we mentioned in Subsection 5.1.1, therefore, a server failure without recovery is not considered.

5.2 Proposed solutions to cases of failures

In this section we give solutions to the single cases of failure described in Section 5.1, first in a descriptive manner, later in the application of algorithms and a communication protocol.

5.2.1 The *Shaking Calls* mechanism

We named this integral system **Shaking Calls** since we require all messages sent in any direction to have a mandatory acknowledge (or *handshake*). These acknowledges ensure in a bidirectional way that any message was received and guarantees that the sender has knowledge of the receipt of its request. Furthermore, there exists a message broadcast every 30 seconds (calls) from the server which robots must reply. The main function of this message is to know if a robot has failed giving the server the opportunity of reassigning the robot's task, or, as we describe in this section, that the server has failed, since the robots are aware of these messages also.

We describe below the solutions for each one of the three cases of communication failure exposed in Subsection 5.1.2.

1. Proposed solution to robots failure without recovery. Once the server has considered a robot as dead, the server simply reassigns the unaccomplished tasks left by the dead robot if it was not in idle state. If so, dead robots are not considered in further tasks assignments.

   In order to simulate this case of failure, we program the server with a key message (*die robot*) which produces an abnormal termination of the designated-

to-fail robots. In a real environment, factors such as radio interference, radio limited range or another cause inherent to the robots' malfunction may happen in order to trigger this kind of failure.

2. Proposed solution to robots failure with recovery. For this case, the solution is based on the case robots failure without recovery: the server assumes a robot is dead, although the robot is asleep and will eventually recover communication, thus, the status is reset to alive so a robot shall be considered in an instant task assignment round and in further tasks assignments. Note that in this case, an instant task assignment round applies because at least one robot is supposed to be available (the robot that just woke-up) and an environment change has happened.

   For simulating this case of failure, a message of the type *sleep robot* (for at least 90 seconds) has to be sent from the server to the designated-to-fail robot. The robot does not move and is not able to respond any incoming message during the time of sleep. After that period, the robot must reestablish communication. An intermittent interference may require this recovery mechanism in a real environment.

3. Proposed solution to server failure with recovery. If during the pre-defined (*c.f.* Section 5.1.2, item 3) period of time the server is considered off-line, the robots shall wait for the server to reestablish communication in the usual way indicated by Algorithm 4.1. However, we propose a temporal decentralization of the MRS contrary to the centralized system stated in Subsection 4.2.1: the robots shall be able to self-assign tasks in order to continue operations, and later, when the communication recovers, the server will receive the up-to-date data and reassume its functions.

   The server has to cease all communications with the robots to simulate this kind of failure for at least 90 seconds. Meanwhile, the robots will self-assign tasks (with the same greedy Algorithm of task allocation that executes in the

server) and broadcast their updates each time a task is served or self-assigned. A temporary **token** passes from robot to robot and it has the function of identifying the robot with the last update, *i.e.*, the last robot to make a change in the system. The robot with the token has the responsibility of updating the tables in the server when it comes on-line again. It is worth to remark that, if when dealing with this case of failure a robot fails too, *i.e.* while the server is in the offline status, the situation will not be properly detected. This case of failure is considered multiple and is beyond the scope of this research. An unexpected reboot in a physical server during run-time are examples of the application of this case of recovery in a real environment.

Note that it is mandatory that there is always a processor (server) capable of sending hello messages through broadcast meanwhile the robots answer with acknowledges (*c.f.* Subsection 5.1.1, item 9), thus, the task allocation algorithm might be decentralized but not the fail tolerant mechanism since every single robot should be connected with its partners, giving as a result a complete graph scheme if the robots were the vertices and the connections were the edges.

All these solutions fulfill our previous assumption that this system is considered dynamic since if any of these three cases happens, a task reassignation procedure must be initiated.

5.2.2 Algorithmic solutions

We begin this subsection by highlighting the condition that the **task assignment Algorithm is totally independent of the communication fault-tolerant mechanism and vice versa**. Furthermore, theoretically, if we substituted the task assignment Algorithm (without changing the data structures) in order to achieve a more efficient allocation scheme, there shall not be effects in the execution of the communication fault-tolerant mechanism.

The algorithms proposed use threads of parallel processing. There are four threads in the server Algorithm: for broadcasting hello messages; for receiving incoming messages and sending ACKs; a timer that stops when the maximum run time is reached and stops all operations; and for task allocation that assigns tasks to robots through messages. Meanwhile, there are three threads in the robots Algorithm: for receiving incoming messages and sending ACKs; for detecting the absence of hello messages received which trigger the self-assignment routine; and the main which executes the path planner. Note these threads are specified as **parallel while** in our Algorithms and some lines in their statement are invoked from the reception or transmission of messages in the protocol description from Subsection 5.2.4. All proposed solutions in Subsection 5.2.1 are now described as algorithms in this section. See algorithms 5.1 and 5.2.

5.2.3 An *ad hoc* protocol

For a correct operation of algorithms 5.1 and 5.2, we define a communication protocol for exchanging all pertinent data among all processors of the proposed system shaking calls.

We establish some conventions prior to show the protocol. We define the **direction type** as the flow direction of the generated message from the generator to the recipient, being only three and identified by the type number:

1. From *server* to *robot*.
2. From *robot* to *server*.
3. From *robot* to *robot*.

Finally, we abbreviate *Broadcast* as **BC** and *Unicast* as **UC**. See Table 5.1.

Algorithm 5.1: Fault tolerant Algorithm for communication failures: server

Input : Positions (coordinates) of *tasks* (p_{τ_k}) and *robots* (p_{a_i});
Parallel while indicates a separate thread of processing
Output: Set of closed tasks (T_{closed}) and metrics of *Completion time*, *Total distance*, *Idle time* and *Avoidance time*

1 **begin**
2 Table of tasks is created with coordinates of tasks p_{τ_k}, all *number of demands* (nd_{τ_k}) are set to constant c, all *served demands* (sd_{τ_k}) set to 0 and all *availability* flags ($disp_{\tau_k}$) set to TRUE;
3 Table of robots is created with coordinates of robots p_{a_i}, all *battery levels* (b_{a_i}) are set to 100, all entries of *record vector* set to 0 and all *current tasks* ($current\_task_{a_i}$) set to 0;
4 $T_{open} = T$;
5 Set *my Rx channel*;
6 Calculate and set *maximum run time*;
7 **parallel while** *1* **do**
8 **if** *maximum run time reached* **then**
9 goto **END** (end program);
10 **end**
11 **end**
12 **parallel while** *1* **do**
13 Listen to incoming messages through *Rx* in channel *my Rx channel* **OR** *any broadcast*;
14 **if** *robot* a_i *did not acknowledge* ***3*** *consecutive* ***hello*** **then**
15 Set status of robot a_i as **dead**;
16 Set $disp_{a_i.current\_task}$ as TRUE (available for reasignation);
17 **else**
18 **if** *status of robot* a_i *is* ***dead AND*** a_i ***did*** *acknowledge at least* ***1 hello*** **then**
19 Set status of robot a_i as **alive** (assigns task to this robot);
20 **end**
21 **if** *update received from robot* a_i **then**
22 Update nd_{τ_k}, sd_{τ_k} and $disp_{\tau_k}$ from Table *tasks* and p_{a_i}, b_{a_i} and $current\_task_{a_i}$ from Table*robots*;
23 **end**
24 Send requested **data** (task assignation, tables of tasks and robots) through robot's a_i **Rx** channel;
25 **end**
26 **end**
27 **parallel while** *1* **do**
28 Send *hello* through *Tx* in *broadcast*;
29 Sleep **30** seconds;
30 **end**
31 **end**

Algorithm 5.1: Fault tolerant Algorithm for communication failures: server (continued)

```
32 begin
33     parallel while T_open ≠ ∅ do
34         TASK ASSIGNMENT: ;
35         Update the Matrix of calculations with the distances from robots to
            tasks (d_ik(t)) at the current time;
36         Find the minimum value of row i corresponding to a column of an
            unassigned task τ_k and assign the task τ_k to robot a_i;
37         Update disp_k = 0;
38         Update current_task_{a_i} = k;
39         if there are no available tasks then
40             state of robot a_i is idle;
41         else
42             (Robot a_i completes its assigned task);
43             Update coordinates p_{a_i};
44             Update battery level b_{a_i};
45             Update record vector_{a_i} corresponding to task τ_k to 1;
46             Increment nd_{τ_k} and decrement sd_{τ_k} ;
47             if nd_{τ_k}=0 and ad_{τ_k}=3 then
48                 Move τ_k from T_open to T_closed;
49             else
50                 Set disp_{τ_k} to 1;
51             end
52         end
53         Send tables of task and robots through Tx in broadcast;
54     end
55 end
```

Algorithm 5.2: Fault tolerant Algorithm for communication failures: robots

Input : Positions (coordinates) of *tasks* (p_{τ_k}) and *robots* (p_{a_i});
Parallel while indicates a separate thread of processing
Output: partial metrics of *distance*, *idle time* and *avoidance time*

```
1 begin
2     Set my Rx channel;
3     Set my token as FALSE;
4     Set distance to 0;
5     Set Idle time to 0;
6     Set avoidance time to 0;
7     Receive initial task τ_k from server;
8 end
```

Algorithm 5.2: Fault tolerant Algorithm for communication failures: robots (continued)

```
9  begin
10   parallel while 1 do
11     Listen to incoming messages through Rx in channel my Rx channel
       OR any broadcast;
12     if server did not send 3 consecutive hello then
13       Set server status as asleep;
14       if I fulfilled my current_task τ_k then
15         Broadcast my updates related to τ_k;
16       else
17         Advance towards p_{τ_k};
18         Fulfill task τ_k;
19       end
20       goto SELF ASSIGNMENT;
21     else if hello received from server then
22       if my token = TRUE then
23         Send updates to server (tables of tasks and robots);
24       end
25       Set server status as alive;
26     else
27       Execute requested action (send hello_ACK, die or sleep);
28     end
29   end
30   parallel while NOT termination signal received do
31     if task τ_k received then
32       Move towards p_{τ_k};
33       Fulfill τ_k;
34       Request new task to server;
35     else
36       Enter idle state;
37     end
38   end
39 end
```

Algorithm 5.2: Fault tolerant Algorithm for communication failures: robots (continued)

```
40 begin
41   SELF ASSIGNMENT: parallel while server status = asleep do
42     Update the Matrix of calculations with the distances from robots to
        tasks (d_ik(t)) at the current time;
43     Find the minimum value of row i corresponding my id of an
        unassigned task τ_k and self assign the task τ_k ;
44     Update disp_k = 0;
45     Update current_task_{a_i} = k;
46     if there are no available tasks then
47       my state is idle;
48     else
49       Fulfill my assigned task τ_k;
50       Update coordinates p_{a_i};
51       Update battery level b_{a_i};
52       Update record vector_{a_i} corresponding to task τ_k to 1;
53       Increment nd_{τ_k} and decrement sd_{τ_k} ;
54       if nd_{τ_k}=0 and ad_{τ_k}=3 then
55         Set disp_{τ_k} to 0;
56       else
57         Set disp_{τ_k} to 1;
58       end
59       Broadcast my updates related to τ_k (table of task and table of
          robots);
60       Set my token as TRUE;
61     end
62   end
63   while NOT termination signal received do
64     if my state is not idle then
65       Move towards my self-assigned τ_k;
66       Fulfill task τ_k;
67     else
68       Set my state as idle;
69       Wait until received update;
70       if my status of server = asleep then
71         set my token as FALSE;
72       end
73     end
74   end
75   Send partial metrics of distance, idle time and avoidance time to
      server;
76 end
```

Note that the number of task is used in a wider way than just assigning a task. It is used also to indicate to a robot to enter in *idle state*, *asleep state*, *finish state* and even in *dead state*. The *server* will instruct to a robot to enter in any of the mentioned states whenever a **timer** for simulating a failure has elapsed. Also, the *server* will enter in *asleep state* and return to *alive state* depending on events of that **timer**, for which we require to add this *timer* to the new Algorithms (5.1 and 5.2).

Transmitting TASK-TABLE (see Table 5.1) and ROBOT-TABLE happen in very special cases since these tables can be sent through *broadcast* in both: when the *server* publishes an update of the conditions of the environment; and in the case of *Server failure with recovery*, the *robot* with the last *up-to-date tables* sends the tables through unicast to the *server*. The same situation happens with the *Direction type*: usually, the *server* sends the tables to the *robots* (type 1), but when the *server* has a failure, the *robots* publish any update to these tables resulting from self-assignment events decided by themselves (type 3). Note that, again, the *robot* with the last *up-to-date tables* sends the tables to the *server* (type 2). See these special cases in Table 5.1.

We proposed the protocol detailed in Table 5.1 in such a way that every *received message* will have a **mandatory ACKnowledge**. This mechanism (known as **handshaking** in *networking jargon*) will ensure the minimum proof that the counterpart has received the first sent message or else, take some action such as retransmitting the message, in order to **guarantee the same knowledge about the environment** or at least the **more recent version** of it.

5.2.4 Protocol description

In this section we describe the operation of the proposed communication protocol of Subsection 5.2.3.

Code	Parameters	Parameter type	BC or UC	Direction type
PREAMBLE	id_{a_i} $p_{a_i}(0)$ $b_{a_i}(0)$	integer in [0..n-2] interval coordinate pair (real,real) real in (0,100] interval	BC	2
PREAMBLE-ACK	*m-1* *n-2* *Rx channel* of *server*	integer: *number of tasks* integer: *number of robots* integer in [0..99] interval	BC	1
HELLO	*consecutive number*	integer > 0	BC	1
HELLO-ACK	id_{a_i} $p_{a_i}(0)$ $b_{a_i}(0)$ *consecutive number*	integer in [0..n-2] interval coordinate pair (real,real) real in (0,100] interval integer > 0	UC	2
TASK-TABLE	*Table of tasks*	Table of tasks	Both	All
TASK-TABLE-ACK	id_{a_i}	integer in [0..n-2] interval	UC	2
ROBOT-TABLE	*Table of robots*	Table of robots	Both	All
ROBOT-TABLE-ACK	id_{a_i}	integer in [0..n-2] interval	UC	2
TASK-ASSIGNATION	*number of task* *sleep time*	integer in [-10..m-1] interval real: time in seconds (1)	UC	1
TASK-ASSIGNATION-ACK	*number of task*	integer in [-10..m-1] interval	UC	2
TASK-FINISHED	*number of task* id_{a_i}	integer in [1..m] interval integer in [0..n-2] interval	UC	2
TASK-FINISHED-ACK	*number of task*	integer in [0..m-1] interval	UC	1
ALL-FINISHED	*partial idle time* *partial avoidance time* *partial distance*	real: time in seconds real: time in seconds real: distance in meters	UC	2 2
ALL-FINISHED-ACK	id_{a_i}	integer in [0..n-2] interval	UC	1
COLLISION	id_{a_i}	integer in [0..n-2] interval	BC	3
COLLISION-ACK	id_{a_i} $p_{a_i}(t)$ *orientation*	integer in [0..n-2] interval coordinate pair (real,real) real in (-180,180] interval	UC	3
DETOUR	id_{a_i} *case of detour*	integer in [0..n-2] interval integer in [-1..1] (2)	UC	3
DETOUR-ACK	id_{a_i}	integer in [0..n-2] interval	UC	3

Table 5.1: Proposed *Ad hoc* protocol for exchanging data. Notes: (1) This is the case when a robot is required to sleep during a specific period of time. If there is no such case, for instance a task assignation, the parameter sleep time is set to zero. (2) This message will be generated only if collision avoidance is detected by multiple robots. The robot which starts coordination requests to its parter(s) to avoid collision by cases: 1, detour to the right; 2, detour to the left; or 3, stop

- PREAMBLE. All robots send a report to the server including their current locations so the server is able to fill the Table of robots. Also the number of the PREAMBLE messages received by the server equals the number of robots (*n-1*) in the environment (line 3 of Algorithm 5.1).

- PREAMBLE-ACK. The server informs the robots how many tasks and robots

there are. Also it indicates the channel through which the server will be listening incoming messages by UC (line 6 of Algorithm 5.1; line 2 of Algorithm 5.2).

- HELLO. The server asks periodically the robots to report if they are alive. The consecutive number included in this message is required for knowing what version of the environment the system has (line 27 through 30 of 5.1).

- HELLO-ACK. The robots acknowledge HELLO messages by reposting the consecutive number received if their version of the environment matches the server's version; if the consecutive number posted by the robot is less than this, the robot has a previous version; if the robot posted a higher consecutive number, the server has been asleep and requires updates. The robot's position and battery level go to the Table of robots as updated data (lines 11 and 27 of Algorithm 5.2).

- TASK-TABLE and ROBOT-TABLE. These messages are considered as an snapshot of the environment. It is exchanged among all possible parts whenever is necessary, as described in Subsection 5.2.1 (line 53 of Algorithm 5.1; line 59 of Algorithm 5.2).

- TASK-TABLE-ACK and ROBOT-TABLE-ACK. Acknowledge that the counter part has received TASK-TABLE and ROBOT-TABLE.

- TASK-ASSIGNATION. The server after having run the task allocation Algorithm informs the robot its assigned task. The robot moves toward the task's location since it has a copy of Table of tasks. Also, it is used for instructing a robot a change of state: enter idle, die, sleep or finish (lines 38 through 40 of Algorithm 5.1).

- TASK-ASSIGNATION-ACK. The robot acknowledges the receipt of a TASK-ASSIGNATION message by confirming its assigned task.

- TASK-FINISHED. A robot confirms to the server whenever it has accomplished a task by letting know the server which task has been completed (line 34 of Algorithm 5.2).

- TASK-FINISHED-ACK. The server acknowledges to the robot the task it just finished.

- ALL-FINISHED. A robot reports its metrics to the server once it received a message of TASK-ASSIGNATION finish (line 75 of Algorithm 5.2).

- ALL-FINISHED-ACK. The server acknowledges the robot has delivered its individual metrics.

- COLLISION. A *robot a* that has detected an obstacle with is infra-red sensors asks which robots are within a radius of 1 meter of its location in order to calculate possible collisions.

- COLLISION-ACK. A *robot b* responds with its id, location and orientation to a parter who has asked for these data. In the counterpart, the *robot a* which started the negotiation (with a COLLISION message) computes a possible collision and determines if DETOUR message is required because it has also received a COLLISION message from *robot b*, implying multiple perception cases (Section 4.4). If not, *robot a* applies solution to collisions by single perception cases explained in Section 4.4.

- DETOUR. Subject only in case of multiple perception detected. The *robot a* which began negotiation indicates its partner(s) an evasive action explained in Section 4.4.

- DETOUR-ACK. Acknowledge of the robot(s) that received a DETOUR message.

5.2.5 State diagram: server

This subsection describes in detail of the state machine of the shaking calls mechanism. See Figure 5.2.

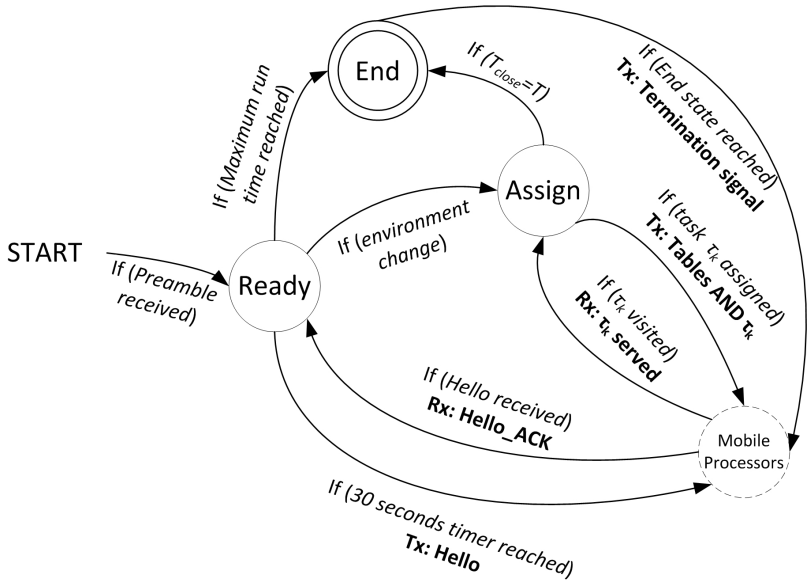


Figure 5.1: State machine: server

- **State Ready**. The server reaches this state after receiving (Rx) a preamble message from all robots transmitted in BC. With the number of robots, the server is able to initialize table of robots and the table of tasks is created by reading a file containing their locations. This state has the function of sending (Tx) hello messages to the robots and receiving (Rx) their correspondent acknowledges.

- **State Assign**. The server runs the task allocation algorithm in this state by receiving (Rx) a message of task finished. Then it determines the next task the robot should serve or if it goes to idle state. Note that if a robot has a communication failure, State Ready sends a signal to this state in order to

reassign the unserved task. This state transmits (Tx) the tables of tasks and robots in BC and the task assigned in UC for each robot.

- **State End**. The server arrives to this state if all tasks are closed ($T_{open} = \emptyset$) or if the maximum run time is reached. In both cases the server must send (Tx) a termination signal in BC to the robots, indicating the overall end of operations.

- **Mobile Processors**. Note this is not a state but destinations and origins of messages of the robots.

5.2.6 State diagram: robots

As a counterpart for the state diagram of the server, we also include the state diagram for the robots. See Figure 5.2.6.

- **State Idle**. The robots begin operations by receiving (Rx) the acknowledge to the preamble from the server. The robots set the transmission and reception channels and are aware of the number of tasks and robots in order to receive (Rx) the tables. This state reports to the server if a task has been served by sending (Tx) a message of task finished, acknowledges (Tx) incoming (Rx) hello messages and starts self assignation state if it detects that the server is off-line. When an task assignment has been made by either, the Self assign State or by receiving (Rx) it from the server, the Path planner State activates.

- **Self assign State**. This state is reached only if the robots detect that the server is off-line and as long as the server keeps that status. If a task is self assigned, it is started the path planner State.

- **Path planner State**. When a task is assigned, the robot initiates this state by establishing the goal's coordinates (task location). In this state the processors

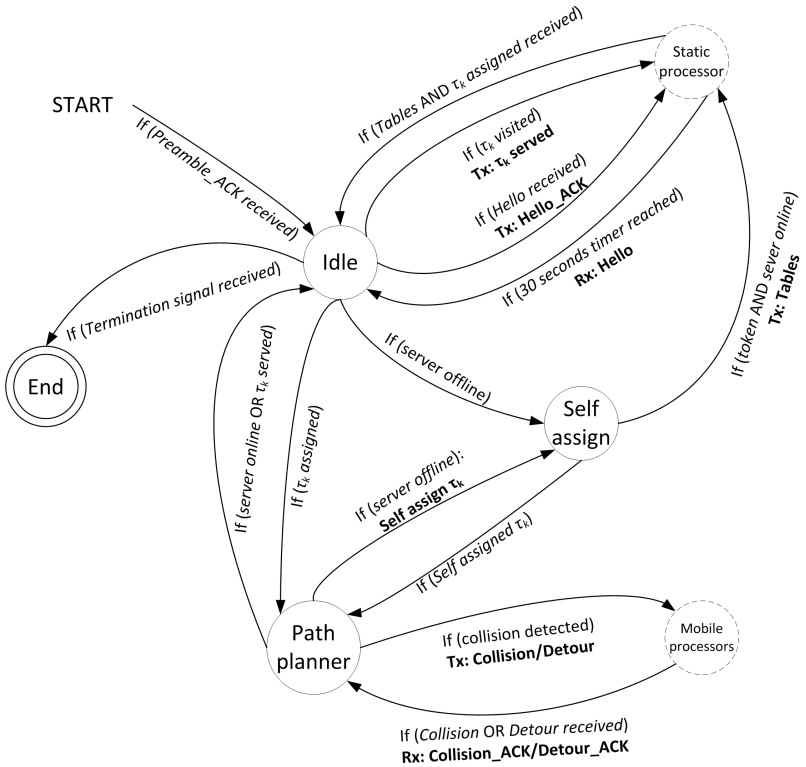


Figure 5.2: State machine: robots

detect obstacles and send (Tx) messages to other robots in order to avoid collisions.

- **End**. The robot enters this state by receiving (Rx) a termination signal triggered by two possible reasons: the tasks are all closed or the maximum run time is reached.

- **Mobile and Static Processors**. They are not states but destinations and origins of messages as indicated.

5.3 Summary

- We defined the cases of failure treated in this research: robots fail without recovery; robots fail with recovery; and server fails with recovery. Al these are cases of simple failures, *i.e.*, only one case of failure happens in a simulated environment. See Section 5.1.

- We have described the solutions given to the cases of failure stated above by asking periodic responses from the robots and detecting the lose of communication among the robots and the server. If failures happen, the MRS makes use of the mechanism shaking calls. The mechanism includes a communication protocol and a task allocation Algorithm, both specified in Section 5.2 and independent one of each other.

Chapter 6

Experiments and results

This chapter describes the simulations performed as part of our research. We also discuss the importance of the initial conditions of the environment for tests, and treat the effect of using random-generated environments in Section 6.1. Later, we cover the results we obtained in simulated environments by using controlled environments for tests in Section 6.2 and compare with our basis (simulated environments without failures) how the shaking calls mechanism is able to solve and reassigns tasks when communication failures happen in Section 6.3. Finally, we end by summarizing the ideas presented in this chapter in Section 6.4.

The goal of these experiments is to demonstrate how our mechanism is resilient to failures that lead to loss of communication in a MRS instead of proposing an efficient task allocation Algorithm. This work gives a feasible solution of task reassignation if failures happen. We base our results on four metrics specified in Chapter 4 (completion time, total distance, idle time and avoidance time). We also show how the mechanism reassigns tasks to robots by showing Gantt charts for the basis and the three cases of failure in the same simulated environment (*i.e.*, same location of tasks and same initial location of robots).

6.1 Setup

In this section we describe all the hardware and software required for achieving our set of experiments, how we calculate the maximum run time along with the periods of failures in an environment and finally we state how we named every environment of our experiments.

6.1.1 Technical specifications

All simulated environments with their respective cases were run in **batch** mode in a computer with the following features:

- AMD 8-core CPU, model FX-8350
- 4 gigabytes of DDR3 RAM memory running at 1600 MHz
- AMD Radeon GPU, model R7 240 with 2 gigabytes of DDR3 RAM memory
- Microsoft Windows version 7 Professional operating system
- Cyberbotics Webots version 8.0.5 with PRO trial version
- The *server* was simulated as a Koala robot with *Supervisor* mode.
- The *robots* were simulated as Koala robots with *differential wheels* mode including all nodes mentioned in Subsection 4.2.1.2

6.1.2 Environmental conditions

As it is usually the case for testing algorithms of MRTA such as Munoz-Meléndez et al. (2012) and Lenagh (2013), algorithms are tested establishing conditions that can be registered and repeated, and varying only a few number of parameters to see how the performance of the algorithms change.

In our case, the conditions that are fixed are initial conditions of the environment, *i.e.*, number of tasks and demands per task, and their positions, as well as initial locations of robots. Then we vary the number of tasks, ranging from 4 up to 16 with increases of 4, and run for each number of tasks, simulations involving a fix number of robots, ranging from 3 to 12 with increases of 3.

For each combination, for instance, 4 tasks and 3 robots or 16 tasks and 9 robots, we run a number of repetitions in order to obtain more significant results. For each combination we run 5 repetitions varying the initial conditions of the environment and then we average the results of these combinations to plot one value for the corresponding combination.

The first alternative to test our algorithms was to generate randomly 5 environments. However we soon realized that the spatial distribution of tasks and robots within the environment is an element that can impact the global metrics scored by the system in unexpected ways. This situation cannot enable us to distinguish clearly if metrics are impacted by environmental conditions or as a result of the solution to task allocations.

In order to test our algorithms, we decide to use controlled environments varying the combinations tasks-robots as explained above, but using for the repetitions environments with a similar distribution of tasks and robots.

In this section we provide more details about the sensitivity of metrics to the spatial distribution of tasks and robots. See Figure 6.1 for an example of an initial setup of a randomly-generated environment with 3 robots and 4 tasks and see Figure 6.2 also for a controlled environment for the same conditions.

Using randomly-generated environments robots score bigger values of metrics due to the variability in the distances between tasks compared with these distances in controlled environments where tasks were meant to be located equidistant. Some of the results of this set of results are compared in Appendix D. Also we realized

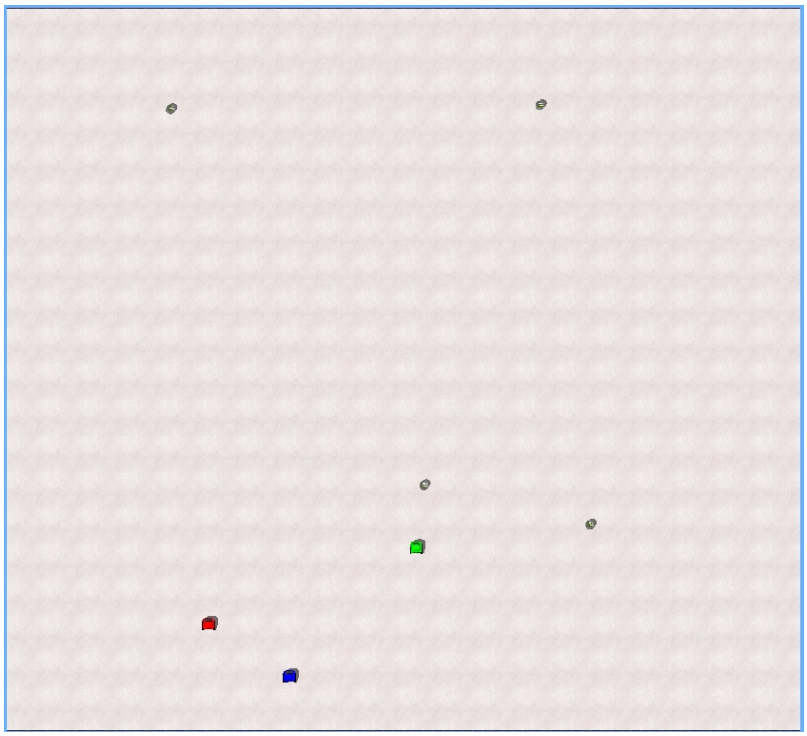


Figure 6.1: Simulated random-generated environment with 4 tasks (gray circles) and 3 robots (colored boxes)

that the completion time calculated using randomly-generated environments may give us a bound and find a suitable formula for the calculation of the maximum run time for controlled environments. See Sub-subsection 6.1.2.2 in order to see how we calculated the maximum run time.

Figure 6.2, gives an example of location of tasks for a 4-task environment. Later, when the number of tasks is increased, these locations are taken as a base for placing the tasks around (*i.e.*, around these places taken from the 4 tasks environments are placed for 8, 12 and 16 tasks, with a separation of 2.5 meters when aligned or parallel to an axis or about 3.5 meters diagonally).

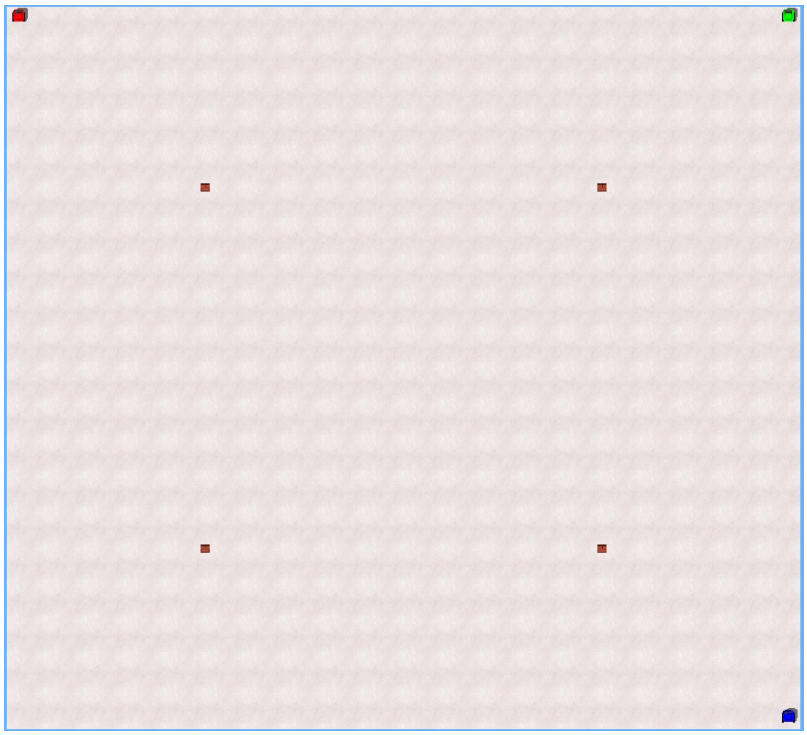


Figure 6.2: Simulated controlled environment with 4 tasks (red squares) and 3 robots (colored boxes), environment *a*

Examples of metrics of controlled versus random environments

A common situation that might happen in random environment is a robot starting operations in idle state over a task and another robot could not reach the task location, resulting in a long-time run of the avoidance routine by trying to reach that location. As a result of these delays the results for all metrics increase making the random-generated environments inconvenient for our demonstrative purposes.

Figure 6.3 shows an example of the results of controlled and random environments for 3 robots comparing completion time. Note that the random-generated environments present a sustained growth resembling a linear slope meanwhile the controlled environments keep a lower slope because of a more uniform distribution

of the tasks in completion time. This metric was taken as a basis for calculating the Maximum run time as we discuss below.

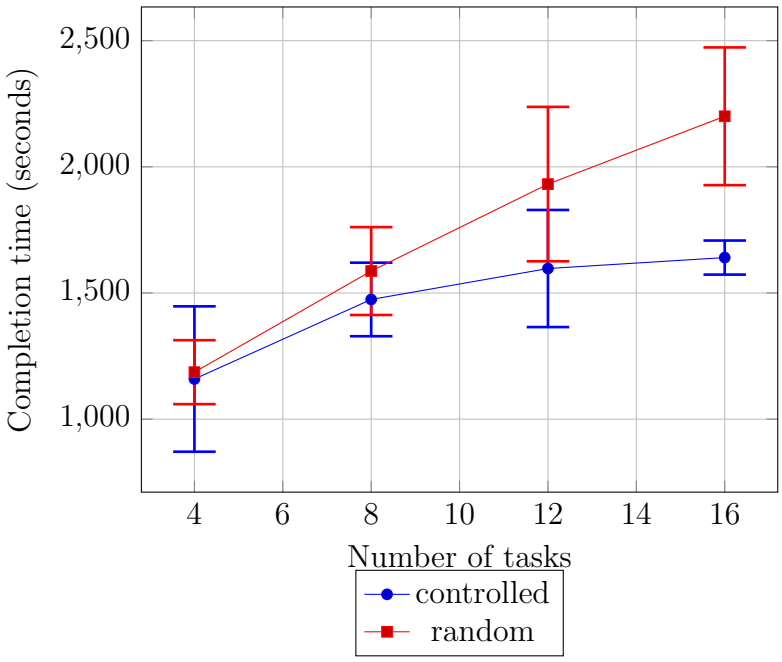


Figure 6.3: Comparison of completion time metric for 3 robots in controlled and random environments. The increase of the standard deviation is due to factors such as avoidance collisions and idle time metrics

Maximum run time

It is worth to remind that one stop criterion for our system was a maximum run time specified in formal definition (*c.f.* 4.1 item 11). To start, we proposed the use of a specific constant called alpha (α) of 665 seconds, calculated as the *approximate time invested by a robot to go across the main diagonal of a proposed environment at a fixed speed of 1 rad/sec*; the missing parameter in formula 6.1 is the radius of a (Koala) robot's wheel that is 0.0425 meters (4.25 centimeters):

$$linear\ velocity = angular\ velocity * radius \tag{6.1}$$

Units:
linear velocity: meters/seconds
angular velocity: radians/seconds
radius: meters

The application of formula 6.1 gives a speed of 0.0425 meters/sec. Now note that in a square of 20 meter by side the hypotenuse (or main diagonal) is approximately of 28.28 meters and from formula 6.2, we obtain the time of 665 seconds using formula 6.3.

$$velocity = \frac{distance}{time} \tag{6.2}$$

Units:
velocity: meters/seconds
distance: meters
time: seconds

$$time = \left\lceil \frac{distance}{velocity} \right\rceil \tag{6.3}$$

Units:
time: seconds
velocity: meters/seconds
distance: meters

Once calculated the maximum time for crossing an environment invested by a robot, we propose now a linear formula for estimating the maximum operation time by taking into account the number of robots, tasks and demands in a certain environment with formula 6.4.

$$maximum\ run\ time = \left\lceil \frac{\alpha * number\ of\ demands * number\ of\ tasks}{number\ of\ robots} \right\rceil \tag{6.4}$$

Units:
maximum run time: seconds

Where the number of tasks and robots varies from environment to environment; however, we have fixed the number of demands to 3 as shown in Table 3.2.

The calculated maximum run time with formula 6.4 is plotted in Figure 6.4. Note that the calculated maximum time overexceeds by approximately five times the actual completion time obtained in a simulation for 3 robots and 16 tasks. This led us to propose a second non-linear but logarithmic formula for exceeding only by an upper bound of approximately 10 minutes (600 seconds) the completion time. The new proposed formula is expressed by formula 6.5

$$maximum\ run\ time = \lfloor (\log_{10}(number\ of\ robots * number\ of\ tasks)) * \alpha * \beta \rfloor \tag{6.5}$$

Units:
maximum run time: seconds

Where β is an adjustment coefficient constant and equals to **2.5** for 3 robots; **1.7** for 6 robots; **1.4** for 9 robots; and **1.5** for 12 robots. See the comparison in Figure 6.4.

As noted, formula 6.5 gives a closer approach for calculating the maximum run time required as a forced program termination when maximum run time is reached. All comparative graphics for the completion time in random-generated environments and maximum run time calculated with formulas 6.4 and 6.5 are reported in Ap-

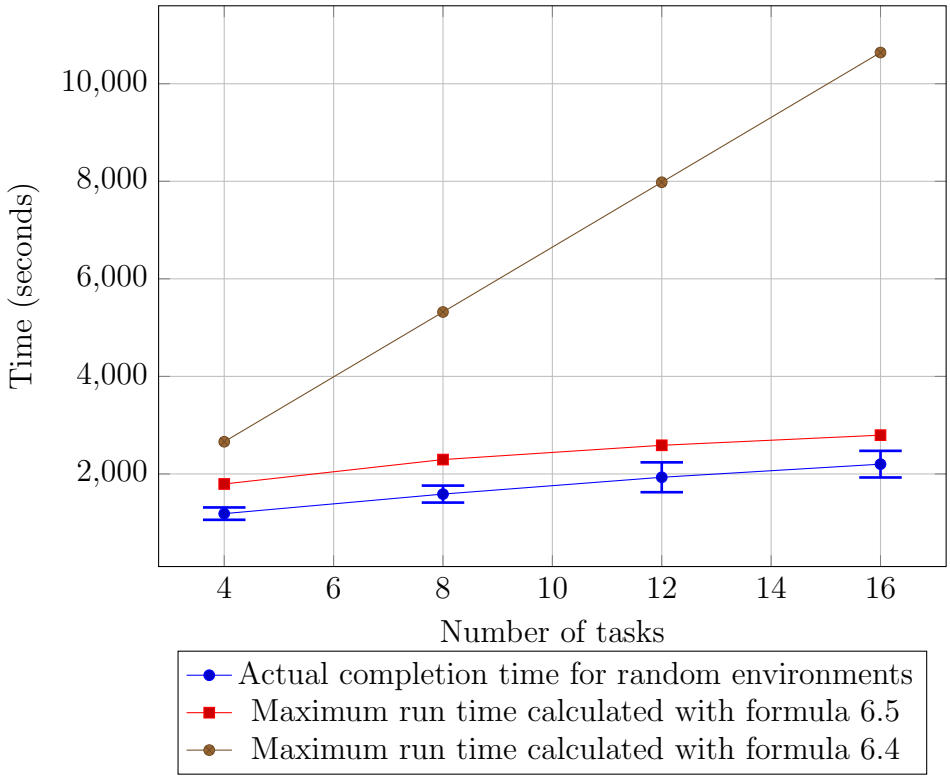


Figure 6.4: Calculation of maximum run time for 3 robots with equations 6.4 and 6.5 compared with simulation results of random environments. Note that the formula 6.5 (red line) gives a close calculation of maximum run time compared with the completion time (blue line) of random-generated environments

pendix A.

6.1.3 Simulation of failures

In this section we discuss the focus of our research: communication failures. We already have defined how we simulate the communication interruptions in Subsection 5.1.2, however, we have not mentioned when the failures happen.

We base all our calculations of failure periods on the previous calculation of the maximum run time as proposed in Subsection 6.1.2.2 since it is the upper boundary

of time when the whole system is operative. We also stated that a third part (1/3) of the total of robots would fail in our objectives in Section 1.3.

The server is programmed for losing communication only once in the run time. For controlling our tests, we also decided that the periods of failures shall not be calculated randomly but controlled, so a uniform distribution is proposed in formula 6.6 for robot failures without recovery and in formula 6.7 for robot failures with recovery.

In the case when the server fails, the periods of failure are calculated identically as if there were only 3 robots in the environment as can be observed in Figure 6.5. The distribution (which resembles the Poisson distribution) was stated uniform because there were initial experiments where the task allocation Algorithm served all tasks even before a simulated failure in a robot would happen. Also note that our system considers failures if there are three unanswered hello messages (90 seconds or more), thus, the simulation of failures must be longer than this period of time.

$$time\ interval\ of\ failure = \left\lfloor \frac{Maximum\ run\ time}{\frac{n\text{-}1}{3} + 1} \right\rfloor \tag{6.6}$$

$$time\ interval\ of\ failure = \left\lfloor \frac{Maximum\ run\ time}{(\frac{n\text{-}1}{3} * 2) + 1} \right\rfloor \tag{6.7}$$

Units:
time interval of failure: seconds
maximum run time: seconds

Where *n-1* is the total of robots in the system (minus the server), and divided into thirds (1/3). Examples for calculated uniformly distributed intervals for 6 robots (2 failures) and 9 robots (3 failures) are given in Figures 6.5 and 6.6 respectively.

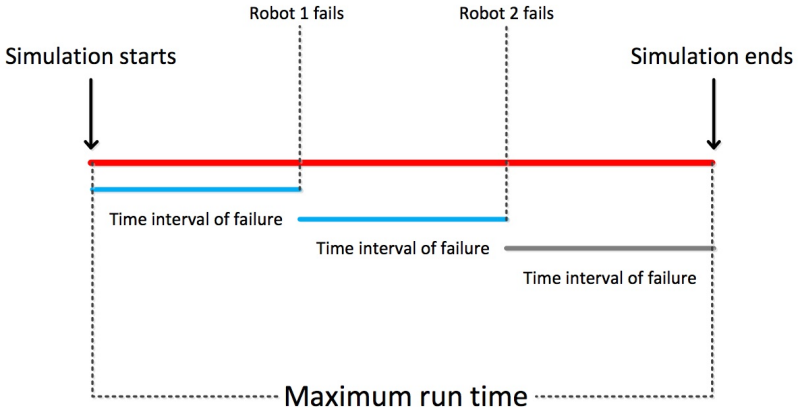


Figure 6.5: Distribution of interval of failure for 6 robots: 2 robots fail without recovery

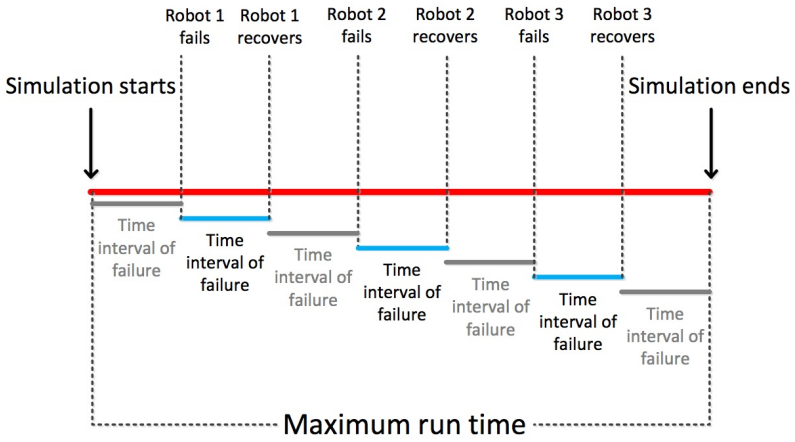


Figure 6.6: Distribution of interval of failure for 9 robots: 3 robots fail with recovery

For the first batch of simulations with failures we proposed formulas 6.6 and 6.7; however, we realized sometimes that robots finished all their assigned tasks without reaching the point of failure, so we made an adjustment to previous formulas and proposed formulas 6.8 and 6.9 for failures without recovery and failures with recovery, respectively, using the same variables and units.

$$time\ interval\ of\ failure = \left\lfloor \frac{\frac{Maximum\ run\ time}{2}}{\frac{n\text{-}1}{3} + 1} \right\rfloor \tag{6.8}$$

$$time\ interval\ of\ failure = \left\lfloor \frac{\frac{Maximum\ run\ time}{2}}{(\frac{n\text{-}1}{3} * 2) + 1} \right\rfloor \tag{6.9}$$

By applying formulas 6.8 and 6.9 we ensure that all possible failures for each case of failure happen in the first half of the maximum run time. The resulting distribution of failure intervals of time resembles a Poisson distribution (*i.e.*, the failures happen in the first half of the operation time line and they are not uniformly distributed along the time line).

For example, using an environment that has 6 robots and 8 tasks, according to formula 6.5 we calculate the maximum run time of 1,900 seconds; then we calculate a time interval of failure of 316 seconds with formula 6.8 for the case of failure without recovery and a time interval of failure of 190 seconds with formula 6.9 for the case of failure with recovery. In the case the server fails with recovery, the maximum run time is divided by 3 and the server failure lasts while the second period or time interval of failure runs. In the latter example with the maximum run time of 1,900 seconds, the periods of failure last 633 seconds.

6.1.4 Summary of experiments

The dimensions of all simulated environments are 400 m^2 in a 20 meter per side as previously mentioned in Subsection 4.2.1. We have designed the use of 3, 6, 9 and 12 robots and 4, 8, 12 and 16 tasks in all possible combinations designating the label beginning with the number of robots followed by the number of tasks. Every possible combination has 5 different environments (the location of tasks change for each of these environments) labeled with the suffix *a* through *e*, so for example *6-16e*

means that is an environment containing 6 robots with 16 tasks e.

- 3 robots with 4, 8, 12 and 16 tasks in 5 different environments.
- 6 robots with 4, 8, 12 and 16 tasks in 5 different environments.
- 9 robots with 4, 8, 12 and 16 tasks in 5 different environments.
- 12 robots with 4, 8, 12 and 16 tasks in 5 different environments.

All the possible combinations and changes of environments sum a total of 80 simulated environments and for each combination an average and standard deviation are calculated for all metrics introduced in Chapter 4: completion time, idle time, avoidance time and total distance. The same environment is used for the three reported cases of simulations: environments without failures; environments with failures and without recovery; and environments with failures and with recovery.

6.2 Comparison of global metrics

We present in this section graphics for the metrics completion time and total distance for two Algorithms: our greedy Algorithm and the Hungarian Algorithm (source code taken from [1] and modified from C# to C) and adapted by us for Time-extended Assignment, comparing results for all cases (robots without failures, failures of robots without recovery, failures of robots with recovery and failure of server with recovery) and environments for 3 robots in Figures 6.7, 6.8, 6.9 and 6.10; for 6 robots in Figures 6.11, 6.12, 6.13 and 6.14; for 9 robots in Figures 6.15, 6.16, 6.17 and 6.18; and for 12 robots in Figures 6.19, 6.20, 6.21 and 6.22. We take the metrics reported for the simulations of environments without failures, the ideal case, as a baseline for every comparison.

[1] http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html

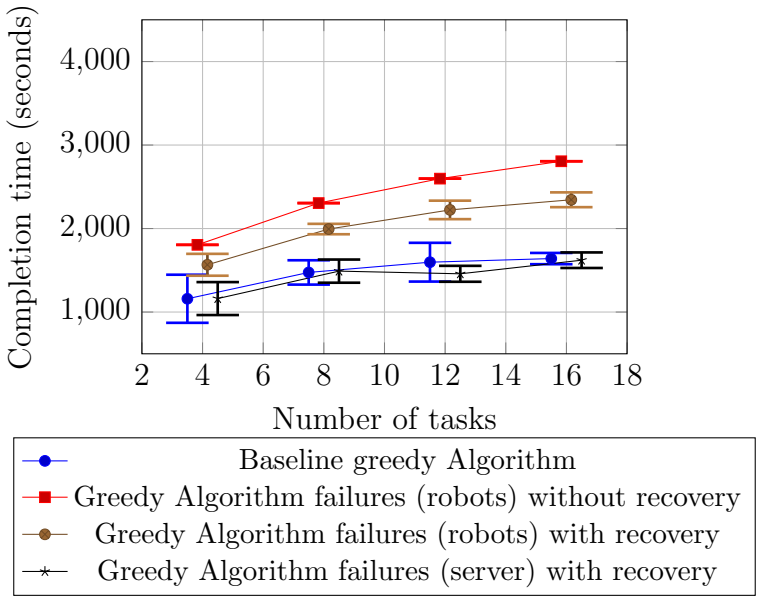


Figure 6.7: Comparison of completion time metric for 3 robots without and with failures and server with failures, greedy Algorithm

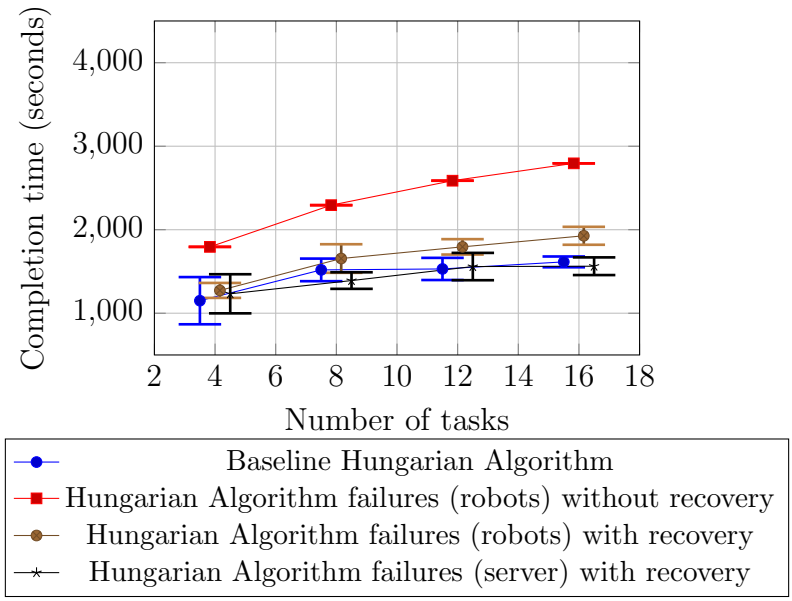


Figure 6.8: Comparison of completion time metric for 3 robots without and with failures and server with failures, Hungarian Algorithm

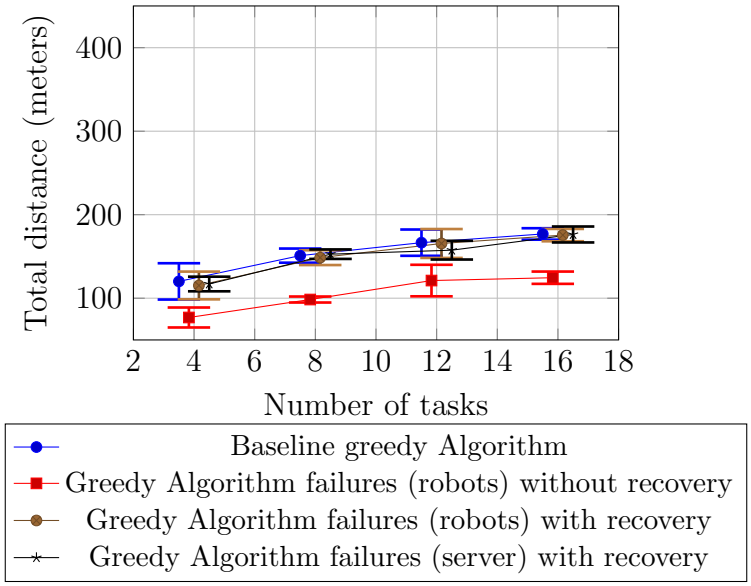


Figure 6.9: Comparison of total distance metric for 3 robots without and with failures and server with failures, greedy Algorithm

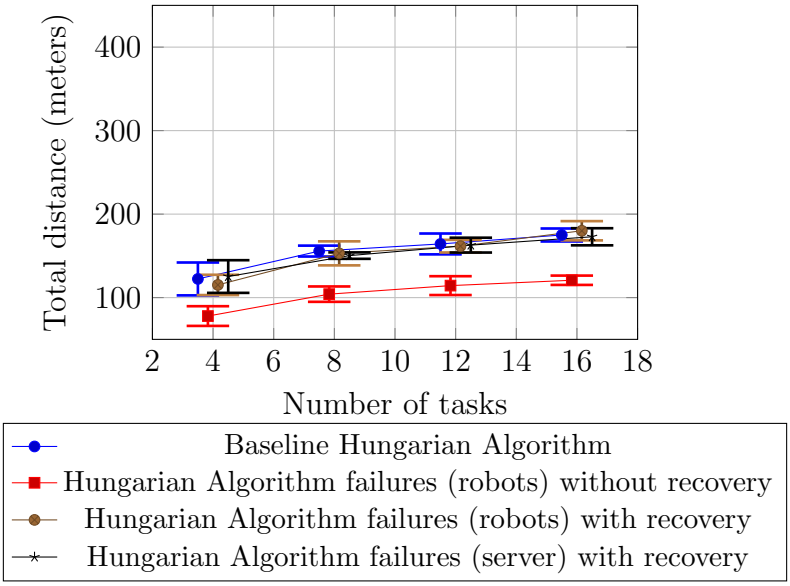


Figure 6.10: Comparison of total distance metric for 3 robots without and with failures and server with failures, Hungarian Algorithm

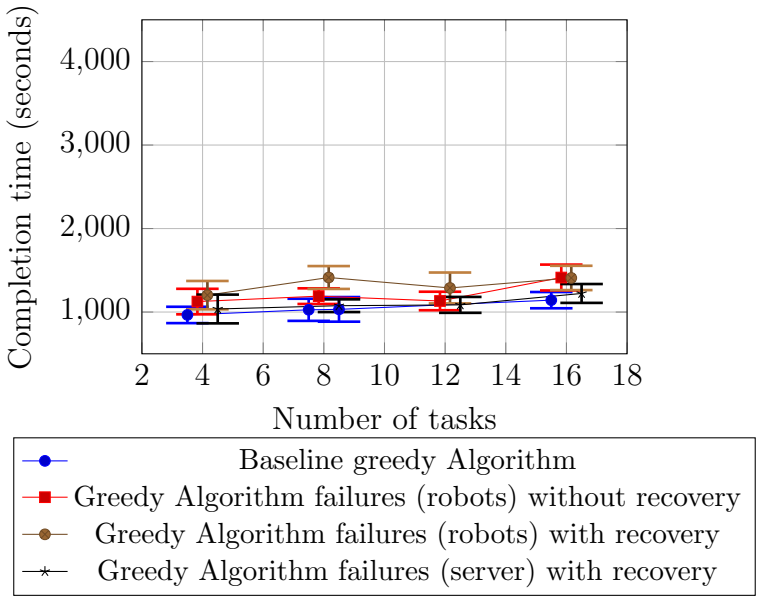


Figure 6.11: Comparison of completion time metric for 6 robots without and with failures and server with failures, greedy Algorithm

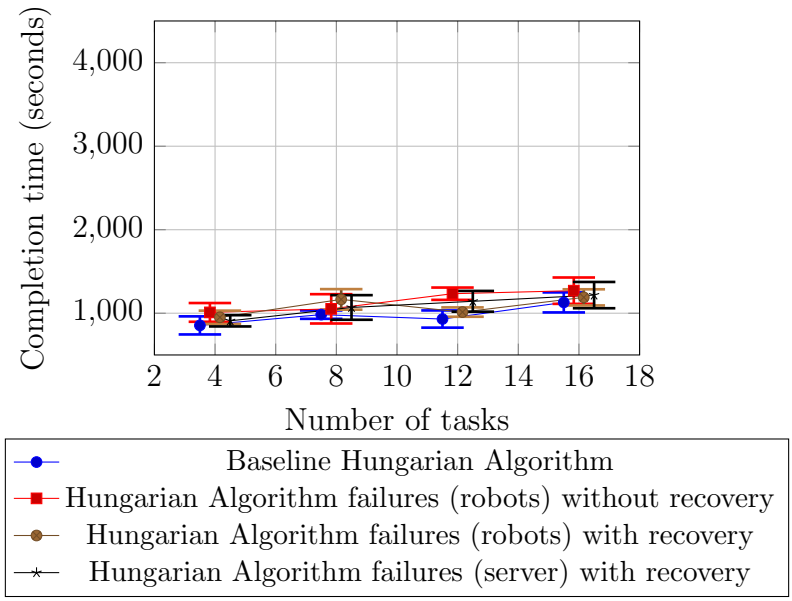


Figure 6.12: Comparison of completion time metric for 6 robots without and with failures and server with failures, Hungarian Algorithm

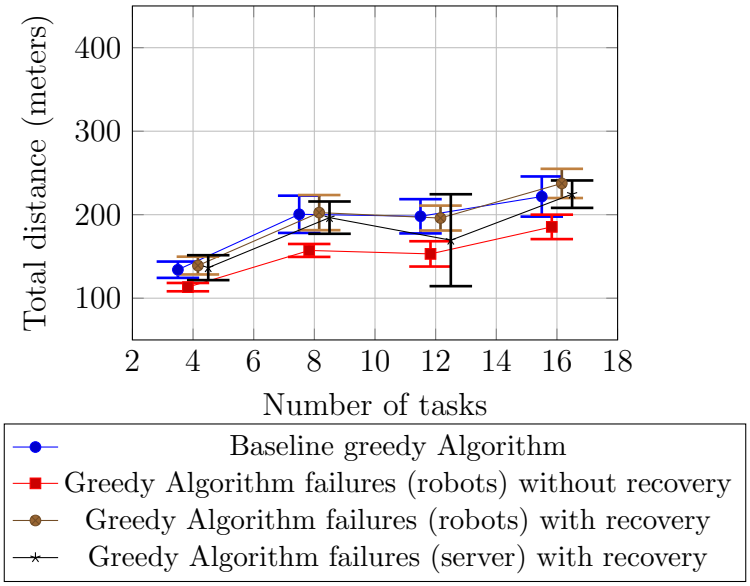


Figure 6.13: Comparison of total distance metric for 6 robots without and with failures and server with failures, greedy Algorithm

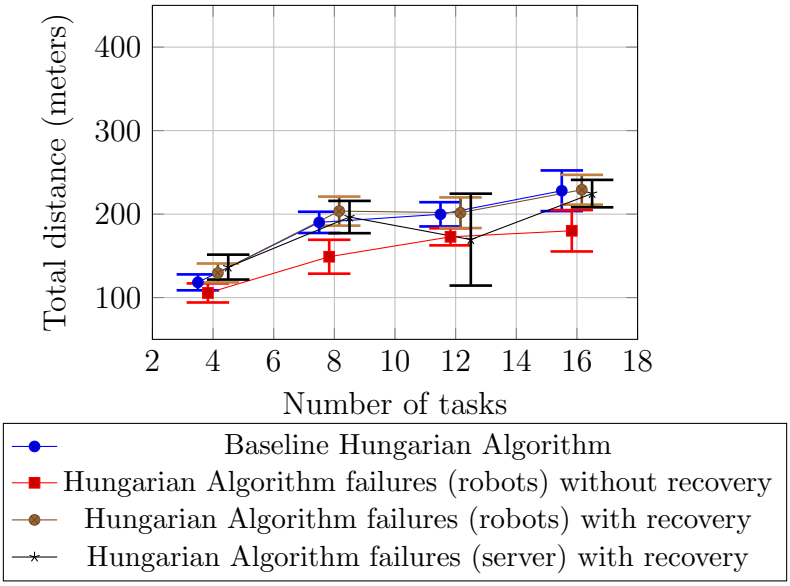


Figure 6.14: Comparison of total distance metric for 6 robots without and with failures and server with failures, Hungarian Algorithm

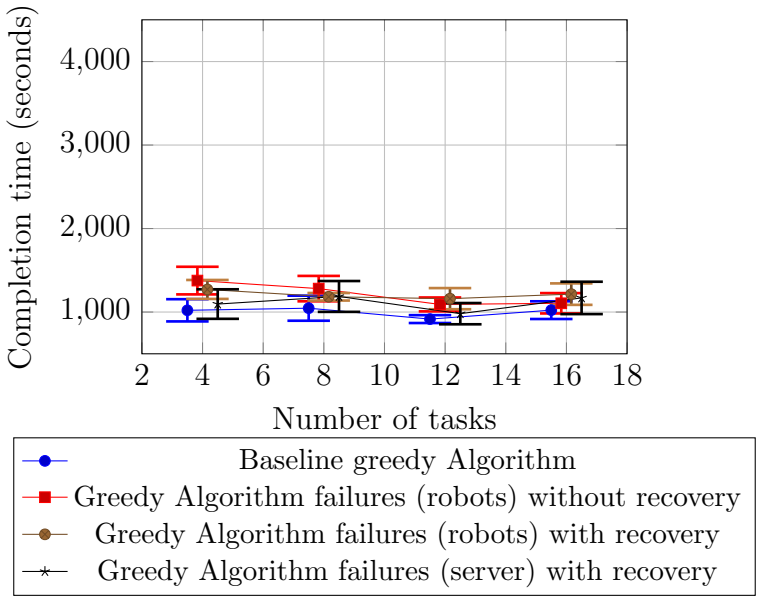


Figure 6.15: Comparison of completion time metric for 9 robots without and with failures and server with failures, greedy Algorithm

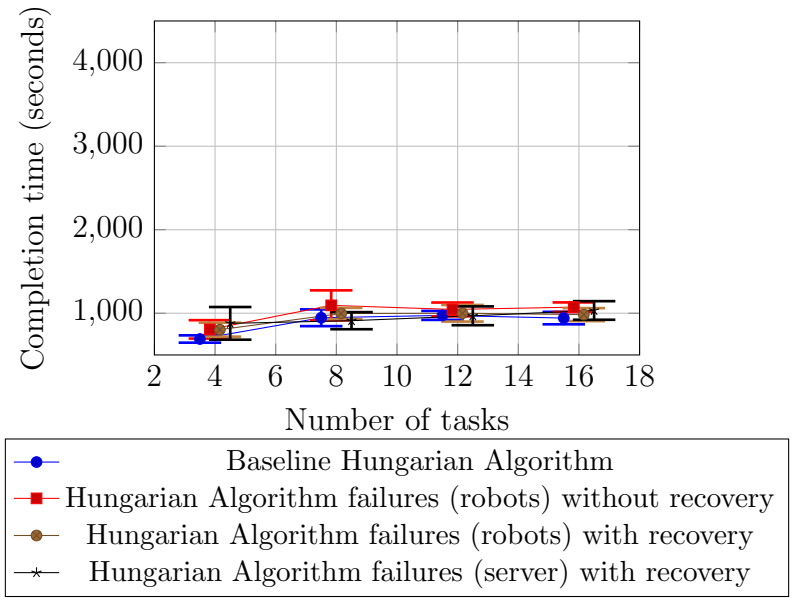


Figure 6.16: Comparison of completion time metric for 9 robots without and with failures and server with failures, Hungarian Algorithm

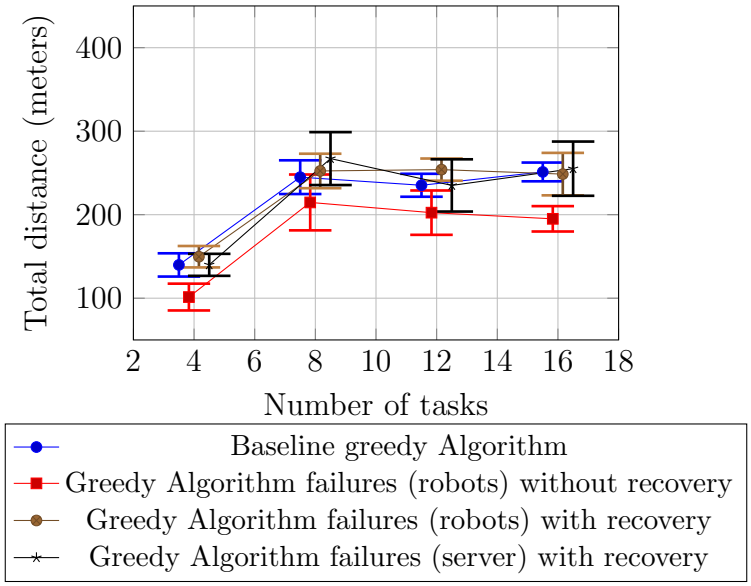


Figure 6.17: Comparison of total distance metric for 9 robots without and with failures and server with failures, greedy Algorithm

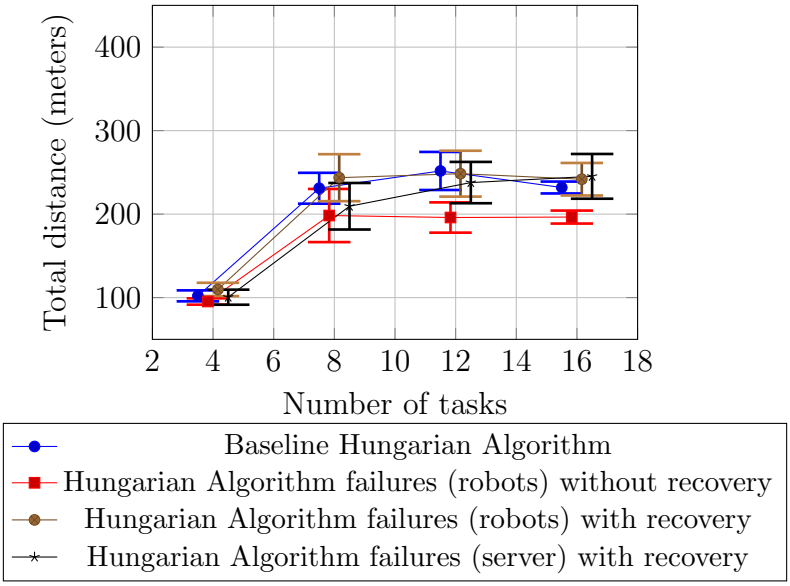


Figure 6.18: Comparison of total distance metric for 9 robots without and with failures and server with failures, Hungarian Algorithm

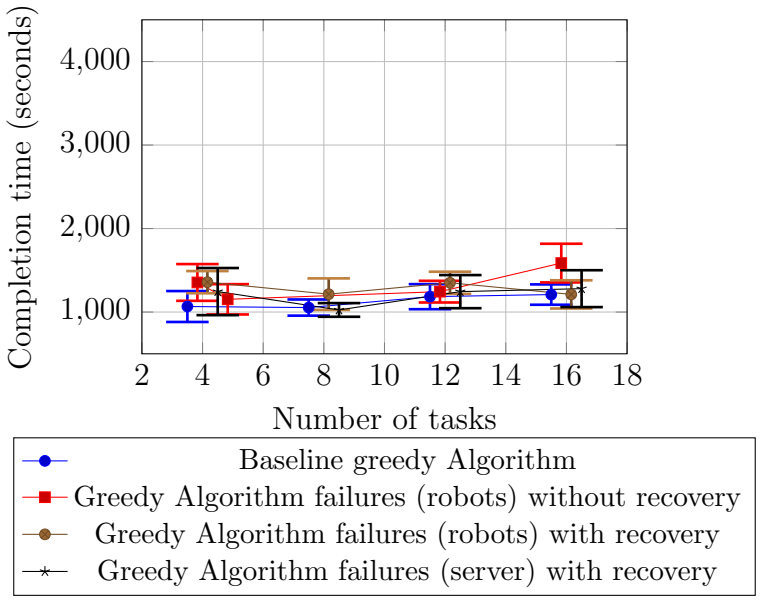


Figure 6.19: Comparison of completion time metric for 12 robots without and with failures and server with failures, greedy Algorithm

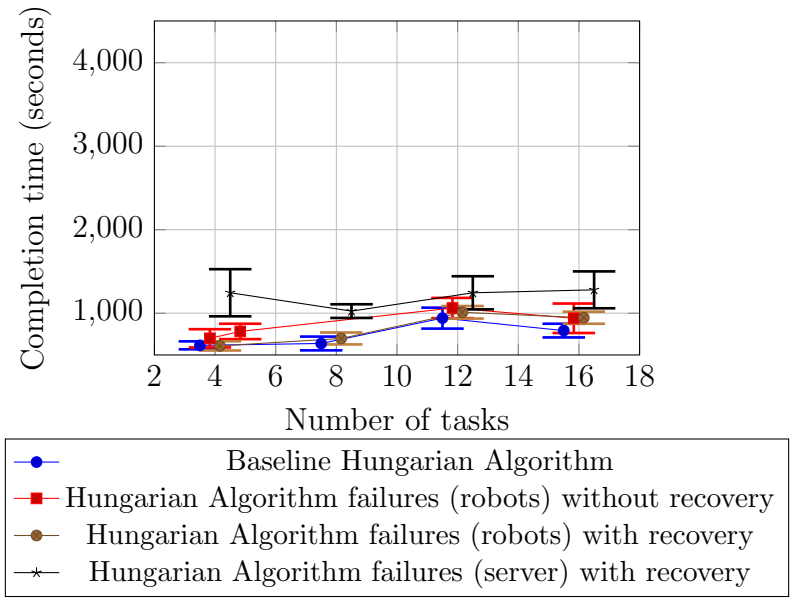


Figure 6.20: Comparison of completion time metric for 12 robots without and with failures and server with failures, Hungarian Algorithm

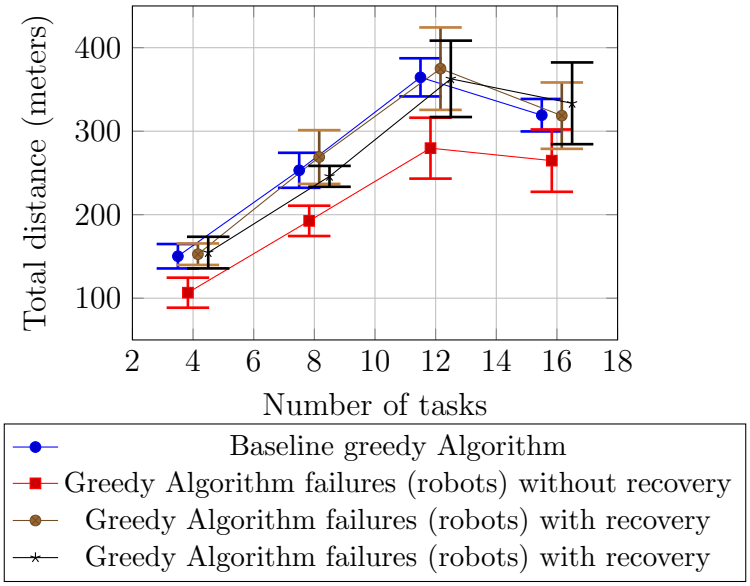


Figure 6.21: Comparison of total distance metric for 12 robots without and with failures and server with failures, greedy Algorithm

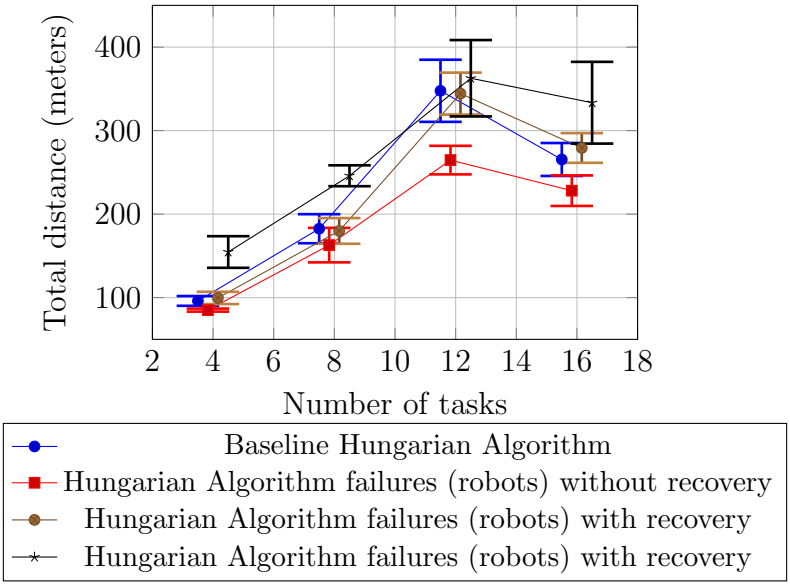


Figure 6.22: Comparison of total distance metric for 12 robots without and with failures and server with failures, Hungarian Algorithm

It can be observed that in all the figures corresponding to completion time for 6 and more robots (Figures 6.11, 6.15 and 6.19, greedy), the time increases above the baseline in almost all combinations of robots and tasks for any case of failure, however, it can be seen that there are some cases (12 robots and 16 tasks, for example) where the time lasted in the simulation of failures of robots with recovery almost overlaps the baseline. A similar situation happens with the failure of server with recovery for 3 robots where the failure of server with recovery (black line) also overlaps with the baseline (blue line). We claim that the shaking calls system helps the greedy task allocation Algorithm to find a better allocation during run-time by reassigning, for example, a task to a closer robot than the robot that failed and was supposed to serve this task, resulting in a faster time for the completion time metric and a shorter distance for the total distance metric.

On the other side, the reported distance for the case of failures of robots without recovery in all combinations of robots and tasks (red line) is always less than the baseline (blue line) since the robots that fail will not complete all their due distance for achieving all the tasks (all the tasks were served for 6, 9 and 12 robots). However, the distance for the case of failures of robots with recovery almost overlaps to the baseline because the task allocation Algorithm works similarly in both cases.

Note a special case for 3 robots for completion time (Figure 6.7) the maximum run time is reached because the number of robots equals the number of demands per task, meaning that in some point when one robot (one third of the number of robots) fails and does not recover. It is worth to remind that a robot is able to serve a demand once (*c.f.* Section 4.1, item 5), and 3 demands are required for closing a task, if a robot fails during run time is able to serve some of the assigned tasks but not all. In most cases, the robots that remained operative were able to complete half (or less) the tasks only.

Another special case is the reported distance for 3 robots in Figure (6.9) for failures without recovery (red line) is less than the baseline (blue line) since the robot

that fails will not travel all its due distance for achieving all the tasks.

Note that the metrics for the case of failure of server with recovery seems to behave, in almost all cases, the same as the failures of robots. This means that the task allocation Algorithm works similarly in the robots and in the server.

The graphics for the metrics idle time and avoidance time can be consulted in Appendix B.

6.3 Dealing with failures

Tables 6.23 and 6.24 show Gantt charts of how our mechanism (shaking calls) reassigns tasks when failures happen with and without recovery. With respect to these tables it is worth to remark:

- The Gantt charts are processor-oriented, or robot-oriented in this case: the tasks are the colored bars, meanwhile the idle time is represented by a black bar and the square pattern bar represents when a robot is in the state dead.
- All environments with the exception of the one represented in Table 6.23 (a) are "finishable", meaning that there exists enough robots for attending all demands (and all tasks) in the environment even though failures happen.
- The case of the environment represented in Table 6.23 (b) is "non-finishable" because a minimum of 3 demands are required for closing the tasks. However, one out of 3 robots fails without recovery during operation time and the other robots are not able to attend more than one demand per task, leaving two open tasks. This situation happens in all environments containing only 3 robots and failure without recovery. For this reason the stop criterion is the maximum run time.

- In the case of failure of the server with recovery (for example in Table 6.23(d)) the colored square pattern indicates the period of failure of the server. Meanwhile, self-allocation among robots might happen in order to continue operations in the MRS.

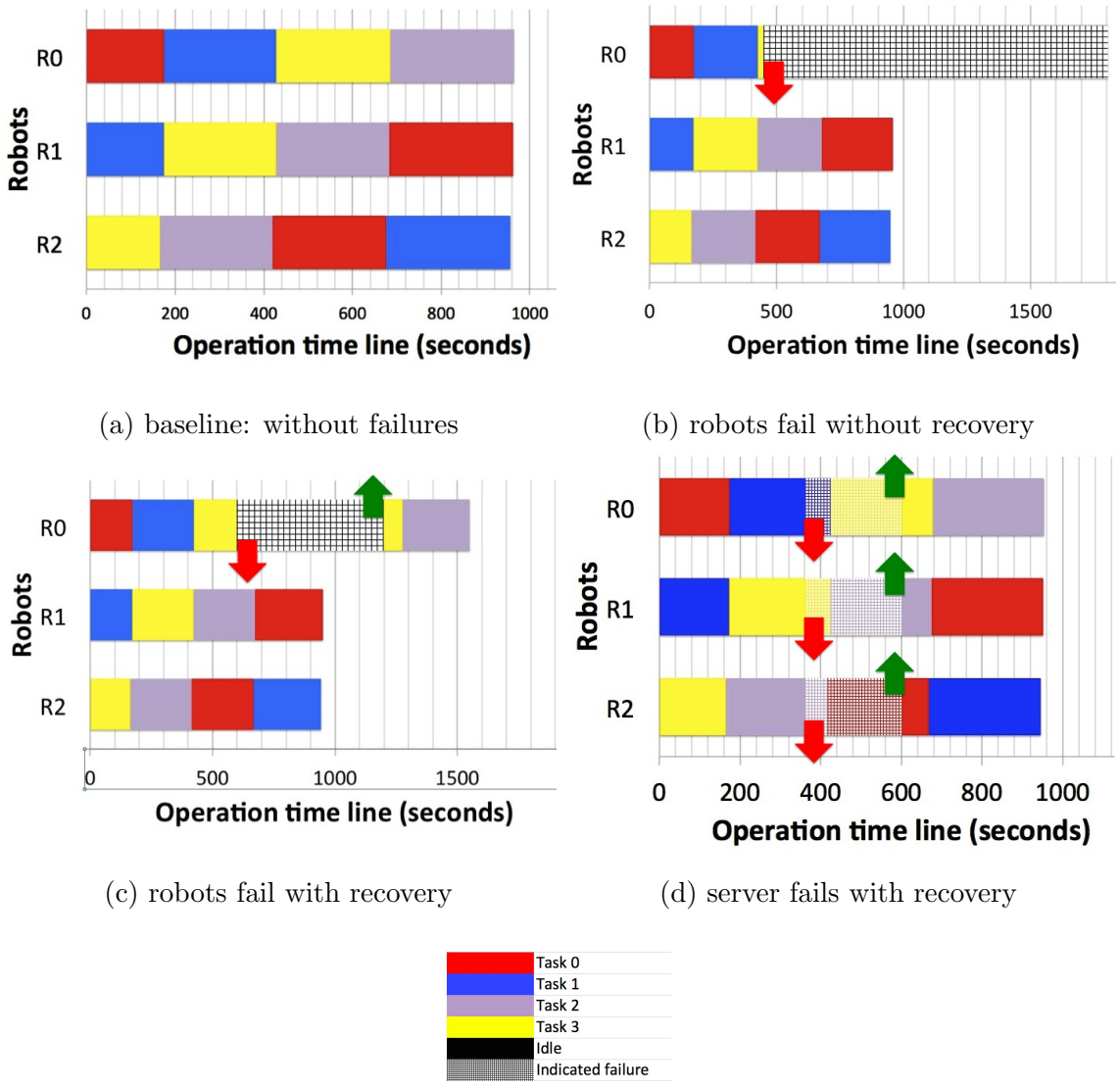
  A detailed report of all simulations with failures is available in Appendix C.

6.4 Summary

- We presented our experiments layout and basis in Section 6.1 for a first batch of randomly-generated simulated environments and a new tasks' distribution of controlled environments. The randomly-generated environments were not used for all the cases of failure; however, it gave us a key feature for the controlled environments: the maximum run time required as a stop criterion for our system.

- Two Algorithms were used for testing purposes: our proposed greedy Algorithm and an adaptation of the Hungarian Algorithm (see Subsection 2.3.2 for a detailed explanation of this Algorithm). The Hungarian Algorithm gave overall more efficient results than the greedy Algorithm. Our purpose of making this comparison was to show that the task allocation algorithm is independent of the shaking calls mechanism (see Subsection 5.2.2). **Using both Algorithms robots were able to finish all tasks in all cases of failures, except in the case of failure of robots without recovery where the number of robots equals (or is less than) the number of demands per task.**

- Our graphics illustrate how our mechanism does not look for making the more efficient allocation but to be resilient to failures. Note that there is in general a tendency for all graphics of metrics completion time and total distance with both Algorithms. Even more, the Algorithms used for this work produce similar



(a) baseline: without failures

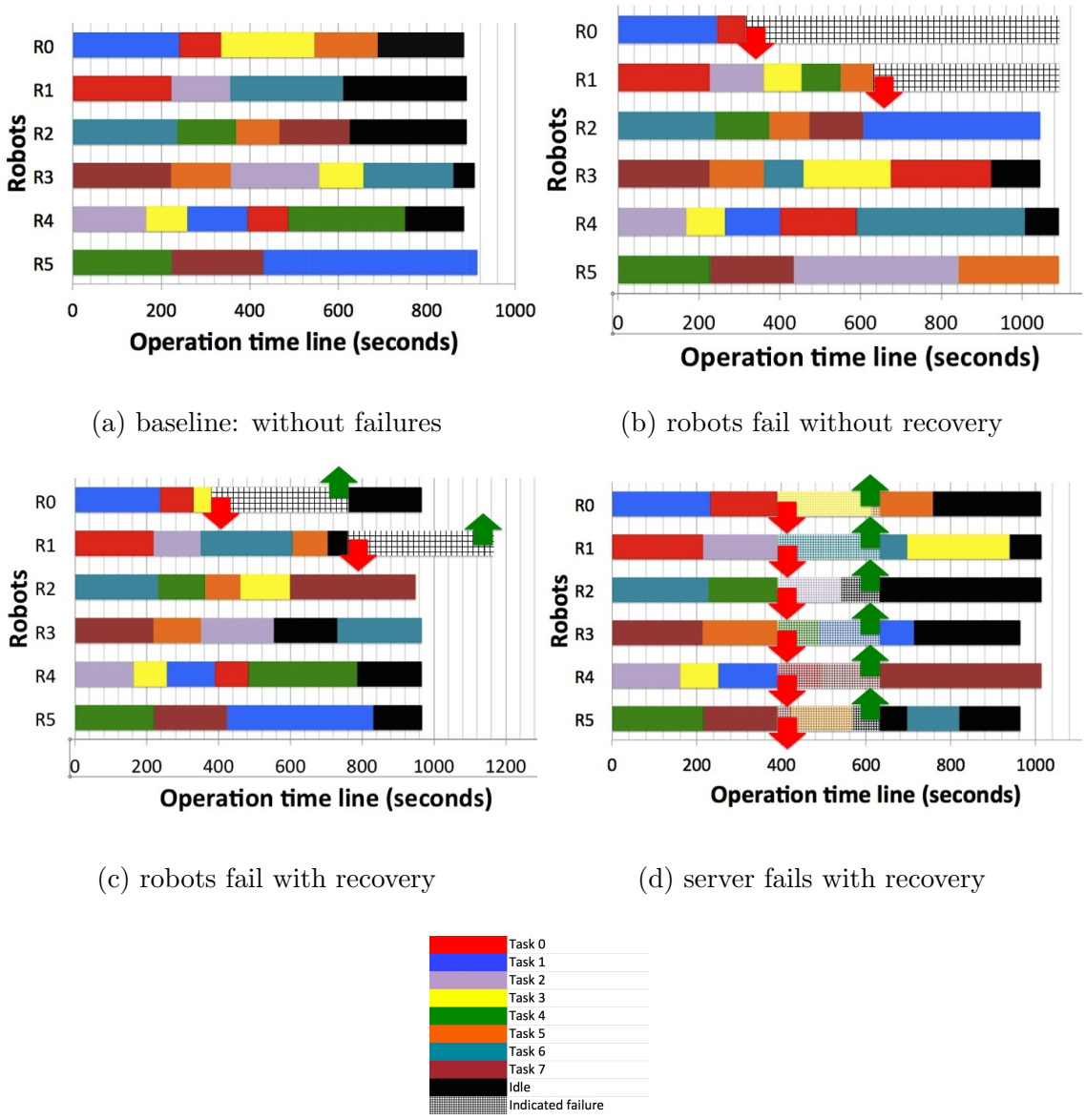
(b) robots fail without recovery

(c) robots fail with recovery

(d) server fails with recovery

Figure 6.23: Comparison of Gantt charts of environment 3-4a: without failures (a); failures of robots without recovery (b); failures of robots with recovery (c); and failure of server with recovery (d). Red arrows represent start of the corresponding failure, whereas green arrows represent recovery

allocations since the environments were the same for every single combination and case of failure; however, the Hungarian Algorithm is more efficient in terms



(a) baseline: without failures

(b) robots fail without recovery

(c) robots fail with recovery

(d) server fails with recovery

Figure 6.24: Comparison of Gantt charts of environment 6-8d: without failures (a); failures of robots without recovery (b); failures of robots with recovery (c); and failure of server with recovery (d). Red arrows represent start of the corresponding failure, whereas green arrows represent recovery

of the mentioned metrics by reducing time and distance.

- Even though, our task allocation Algorithm is inefficient compared with the Hungarian Algorithm, the best scores for completion time and total distance achieved using our greedy Algorithm for all tasks-robots combinations without failures was for the scenario with 9 robots with 12 tasks with 915 seconds and 235 meters, respectively.

- Note that by increasing the number of tasks the metrics did not grew up too far, this is due to the design of our environments where the tasks were located within similar distances from each other (2.5 meters in parallel with axes distance and about 3.5 meters in diagonal) taking the original 4 tasks environments' location as basis. The robots, however, had to avoid collisions whenever more than two robots encountered each other, resulting in an increase of completion time as well as avoidance time.

- The Gantt charts (Figures 6.23 and 6.24) included show examples of how **shaking calls reassigns tasks in case of failure in combination with the task allocation Algorithm**, which is, in fact, the main goal of this research work.

Chapter 7

Conclusions, contributions and future work

We present in this chapter the pertinent conclusions derived from this research work in Section 7.1, we highlight contributions in Section 7.3 and give recommendations for future research work in Section 7.4.

7.1 Conclusions

We remind the motivation and objectives of this research. Our main motivation was to propose a mechanism to deal with communications fails among the members of a robot team that can enrich MRTA algorithms.

We have proposed a **mechanism for fault tolerance in the communication field** as a complementary counterpart to a very simple greedy Algorithm in a Multi-Robot Task Allocation scenario. Our work gives a simple yet powerful solution to real world applications where all kind of interference and intermittent radio signals happen. Along with the fault tolerance mechanism, an *ad-hoc* communication protocol was designed in order to fulfill the afore mentioned mechanism.

We think communication schemes should be studied more in depth in the fu-

ture since any communication failure may lead a Multi-Robot System to misbehave. Very few references of related work take failures or limited communication into account, focusing only in the optimality of the task allocation when designing the task assignment Algorithm.

In some environments we found that a failure somehow helps the MRS to finish earlier its tasks with respect to the time invested by the baseline Algorithm. This situation illustrates a resilience advantage of our work against single task allocation algorithms. Even though our system will invest longer time than others to accomplish tasks if no failures occur, our mechanism for fault tolerance in the cases of failure treated in this research ensures that all tasks will be served if the number of operative robots remains greater or equal than the number of demands per task required for serving a task and if the maximum run time is not reached.

Our fault tolerance mechanism is based on a periodic handshake among robots, being enough for keeping the MRS communicated and replan task assignments due to unexpected failures or recoveries during run-time. We stated the dynamic nature of task allocation in the beginning of this document, this feature has been illustrated for the unexpected changes in the environment that require that tasks are assigned dynamically.

We highlight the functional independence between the task assignment Algorithm and the fault tolerance mechanism by depending only on a shared data structure when necessary, making possible an entire change of the task allocation Algorithm if desired in order to achieve a more efficient task assignment scheme as we have shown with the use of the Hungarian Algorithm adapted for our purposes to work as a Time-extended Assignment task allocation Algorithm.

We have experimentally demonstrated that a simple mechanism introduced in this research called **Shaking Calls**, is able to detect communication and robot fails in a MRS using explicit communication, and it contributes to the achievement of the global goal of the MRS in the context of a Multi-Robot Task Allocation problem.

7.2 Limitations

The mechanism (shaking calls) works under the circumstances we have defined previously. We summarize below the limitations of our work.

- We have proven that our mechanism works with combinations of tasks and robots proposed in Subsection 6.1.4. However, we can not ensure the generalization of this mechanism by increasing the number of robots or tasks due to the hardware limitations of the machine which runs the simulations.
- We state that shaking calls is functional on simulated robots and we assume that this mechanism is able to operate in physical robots by including the use of network communication protocols at the exchange of messages (for example TCP or UDP). Also, in the simulated environment the transmitter and receiver worked with an infinite range or coverage. Real-world scenarios present a problem and limitation in range due to metallic structures, electromagnetic noise and interference.
- The mechanism is able to deal with single cases of failure.
- The path planner requires at any time the reading from the GPS and compass sensors for localizing the robot. All GPS and compass sensors were assumed accurate for every single robot in all simulated environments. Note that the planner function should be replaced if this mechanism was to be used into real-world applications.
- We have shown that the shaking calls mechanism is independent of the task allocation Algorithm. However, it is worth to remark that both Algorithms used in combination with our mechanism shaking calls utilized the same data structures introduced in Subsection 4.3.1.

7.3 Contributions

Below we enumerate the main contributions of this research.

1. A **fully functional Algorithm** for task allocation along with a **communication failure tolerant mechanism** for a Multi-Robot System in a dynamic environment, both interoperable but functionally independent from each other.
2. A **communication protocol** for coordinating the Multi-Robot System as a vital part of the communication failure tolerant mechanism that ensures passing messages among robots through *handshaking*.
3. A simulated **solution** to radio interference, intermittent communications and robots' unexpected failures for a Multi-Robot System application in a real-world scenario.

7.4 Future work

We recommend the next points as possible avenues of future research work.

1. We propose the use of a cluster of computers for simulating a larger (say 1600 m^2) environment and increase the number of concurrent simulated robots in operations (say above 100 robots). We advise and endorse the use of Cyberbotics Webots© as a research tool which is capable of simulating Multi-Robot Systems of both homogeneous and heterogeneous sets of robots. According to the manufacturer, the software can handle unlimited number of concurrent simulated robots, however, the computer on which simulations were run might lack of capabilities for simulating the proposed environment and robots.

2. A second extension is the implementation of this research work using physical robots. The cost of acquiring these physical robots is higher than the use of a simulator, however, we suggest tests in a real-world scenario and simulating communication failure as well as robot's failures.

3. We suggest the design of a sort of predictive Algorithm for the task allocation: idle robots in the environment could be dispatched and park near the task. Thus, when tasks are available again, the idle robots should serve faster those tasks.

4. We have studied single cases of failure, however, a real-world application is not so simple. Extensions to the communications failure recovery scheme studied in this research are suitable for dealing with more complicated cases of failures such as combined cases of failures of robots and server in the same environment and in a concurrent time.

5. Finally, we advice the decentralization of the task allocation function by relegating to robots the self-assignment Algorithm. Nevertheless, we think a centralized scheme with shacking calls mechanism as well as data storage containing all required environment's data is recommended because of the application of the proposed MRS, in this case, mine detection.

Appendix A

Calculation of maximum run time for 3, 6, 9 and 12 robots

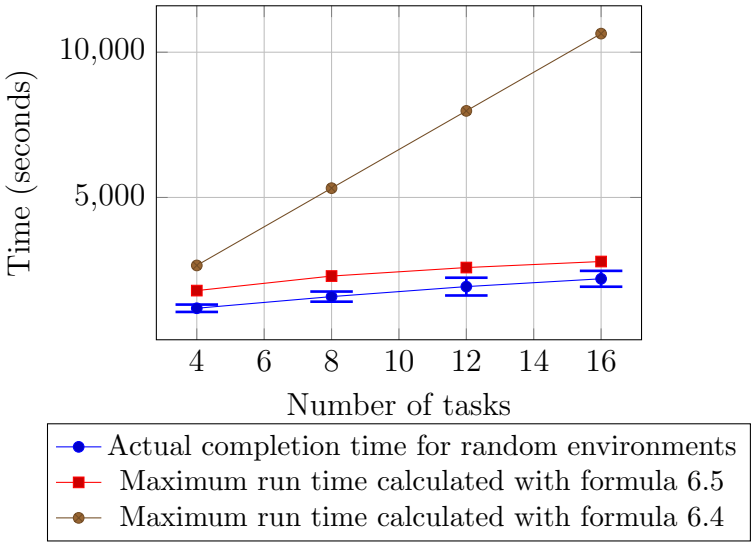


Figure A.1: Calculation of maximum run time for 3 robots with equations 6.4 and 6.5 compared with simulation results of random environments, greedy Algorithm

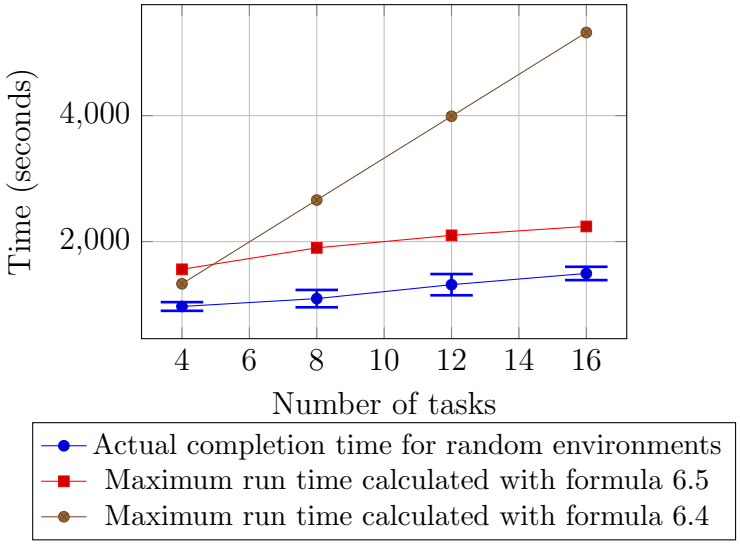


Figure A.2: Calculation of maximum run time for 6 robots with equations 6.4 and 6.5 compared with simulation results of random environments, greedy Algorithm

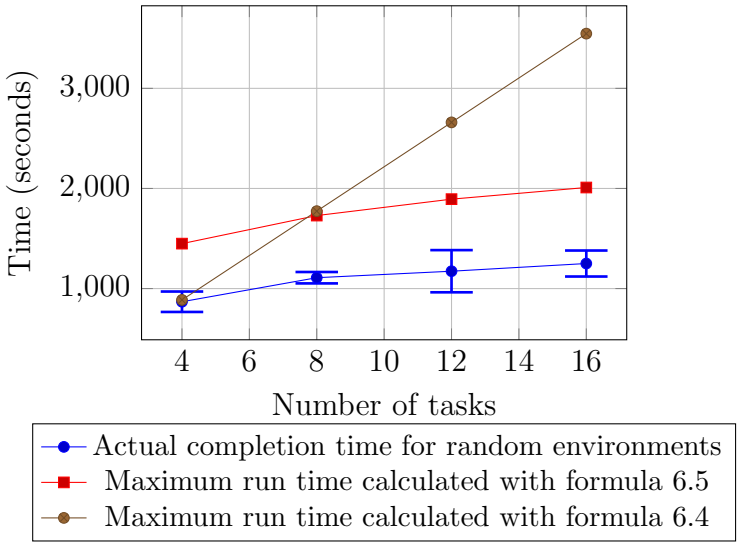


Figure A.3: Calculation of maximum run time for 9 robots with equations 6.4 and 6.5 compared with simulation results of random environments, greedy Algorithm

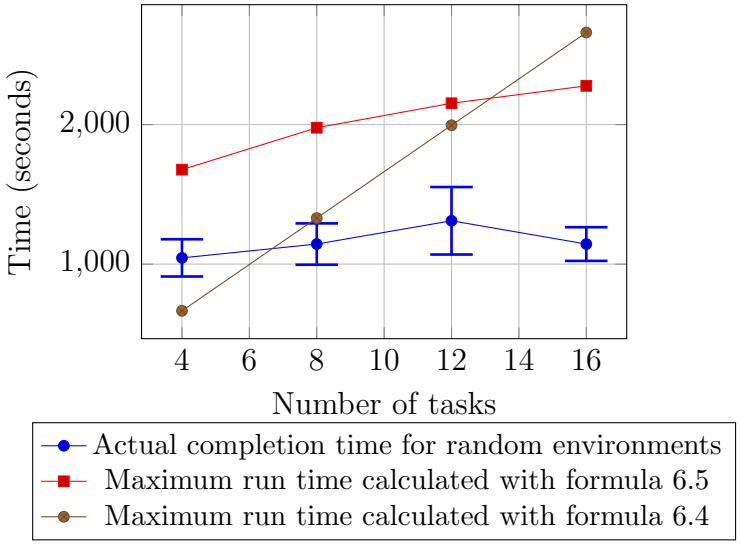


Figure A.4: Calculation of maximum run time for 12 robots with equations 6.4 and 6.5 compared with simulation results of random environments, greedy Algorithm

Appendix B

Graphics for metrics of idle time and avoidance time for 3, 6, 9 and 12 robots with greedy and Hungarian Algorithms

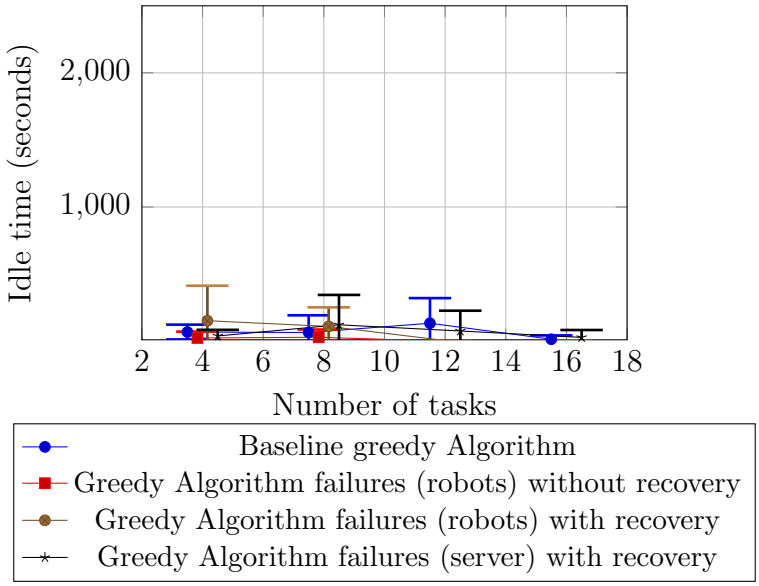


Figure B.1: Comparison of idle time metric for 3 robots without and with failures and server with failures, greedy Algorithm

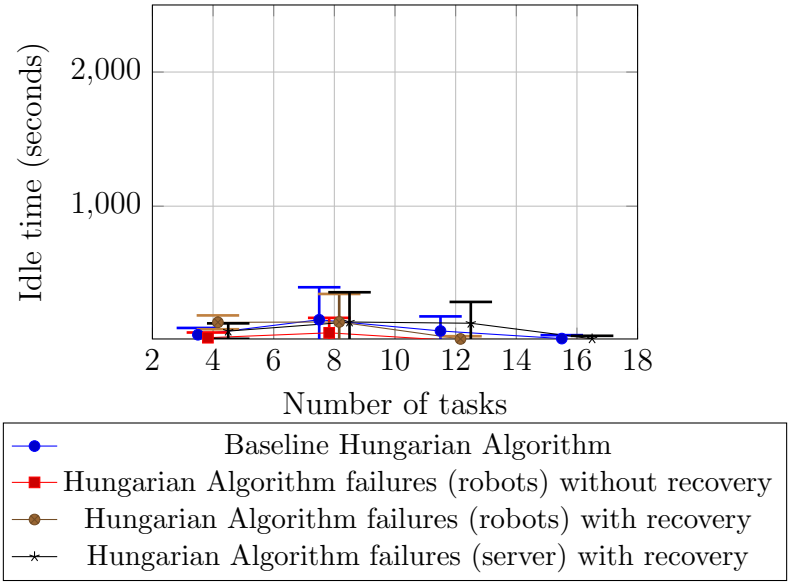


Figure B.2: Comparison of idle time metric for 3 robots without and with failures and server with failures, Hungarian Algorithm

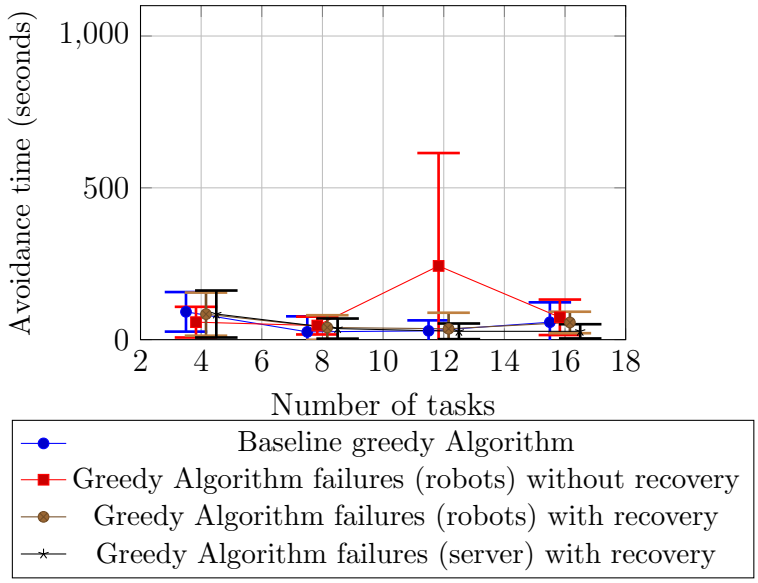


Figure B.3: Comparison of avoidance time metric for 3 robots without and with failures and server with failures, greedy Algorithm

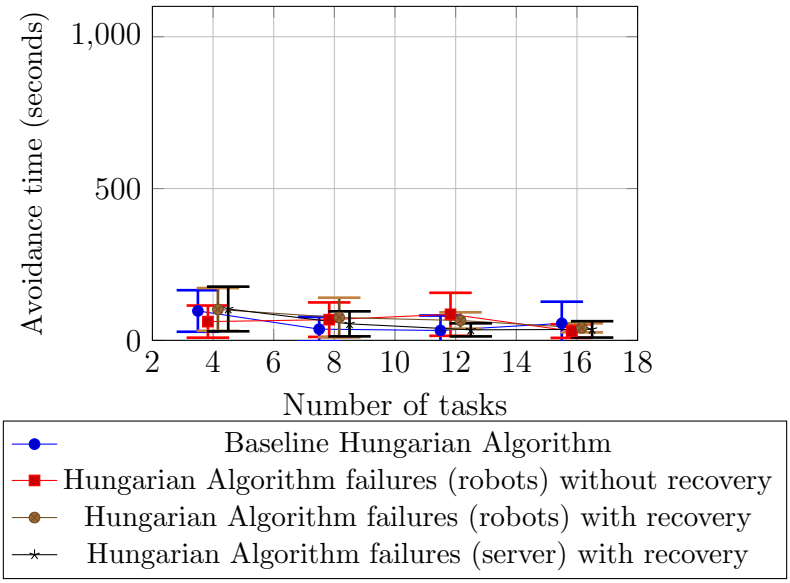


Figure B.4: Comparison of avoidance time metric for 3 robots without and with failures and server with failures, Hungarian Algorithm

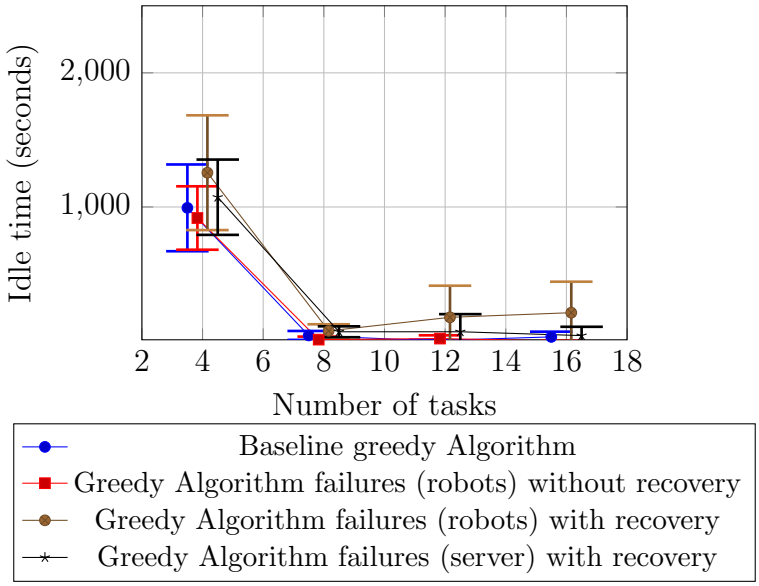


Figure B.5: Comparison of idle time metric for 6 robots without and with failures and server with failures, greedy Algorithm

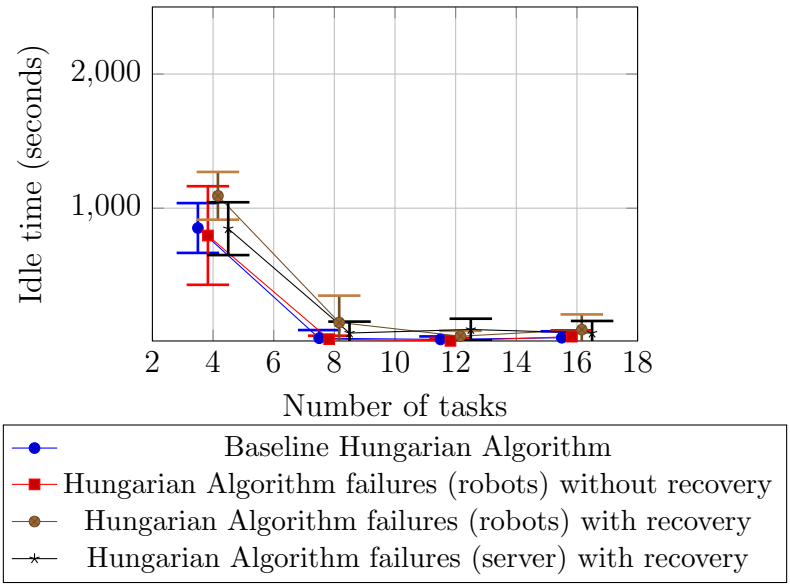


Figure B.6: Comparison of idle time metric for 6 robots without and with failures and server with failures, Hungarian Algorithm

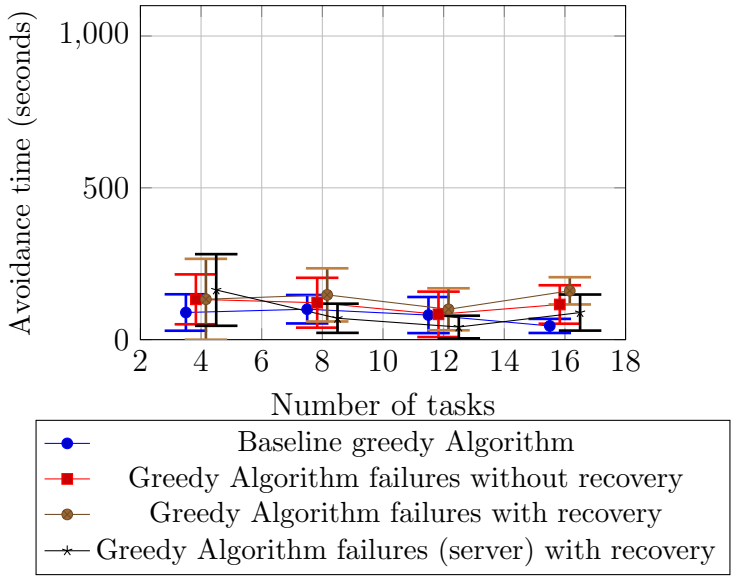


Figure B.7: Comparison of avoidance time metric for 6 robots without and with failures and server with failures, greedy Algorithm

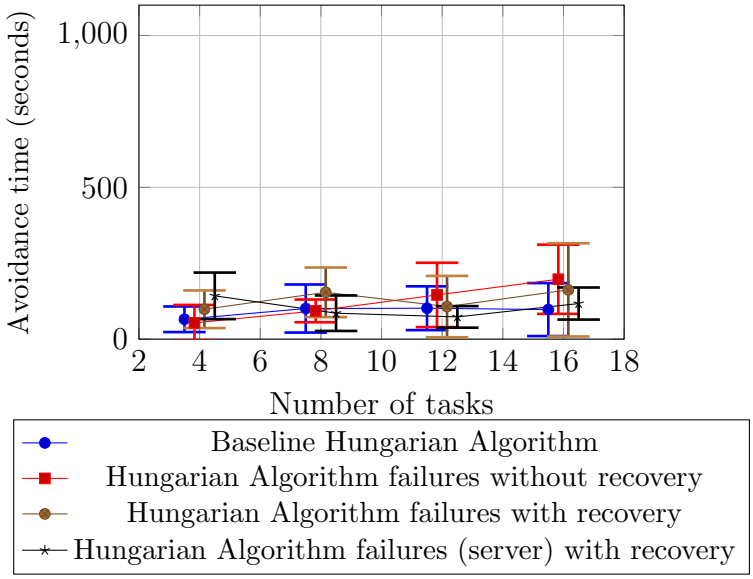


Figure B.8: Comparison of avoidance time metric for 6 robots without and with failures and server with failures, Hungarian Algorithm

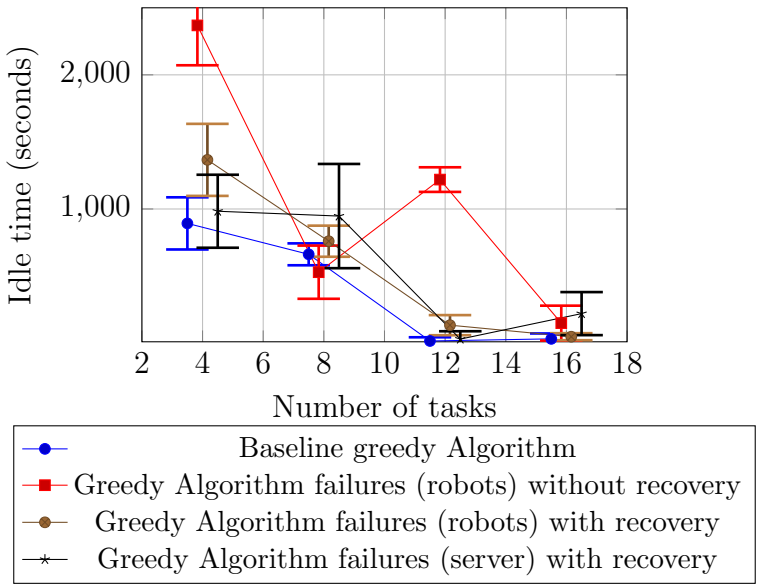


Figure B.9: Comparison of idle time metric for 9 robots without and with failures and server with failures, greedy Algorithm

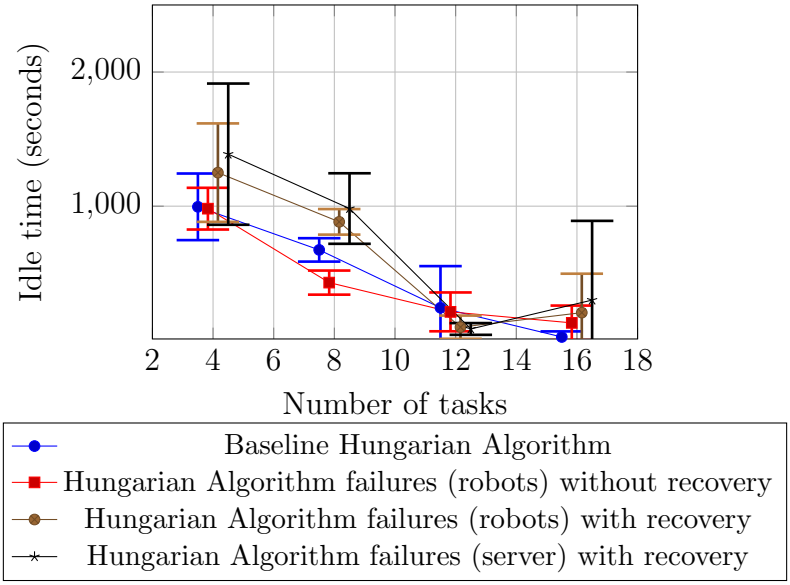


Figure B.10: Comparison of idle time metric for 9 robots without and with failures and server with failures, Hungarian Algorithm

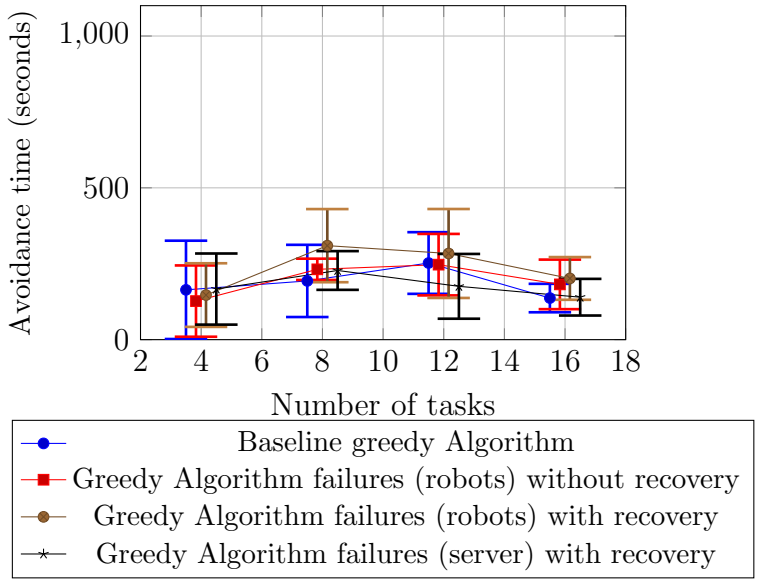


Figure B.11: Comparison of avoidance time metric for 9 robots without and with failures and server with failures, greedy Algorithm

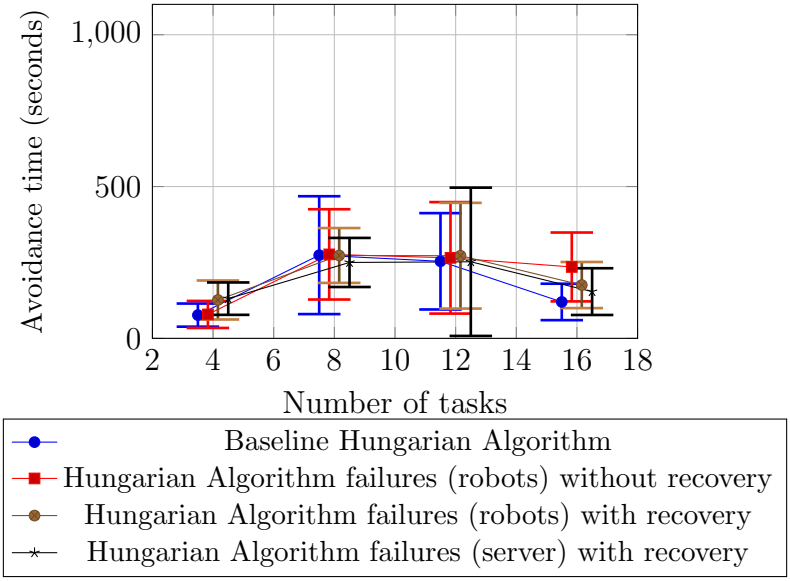


Figure B.12: Comparison of avoidance time metric for 9 robots without and with failures and server with failures, Hungarian Algorithm

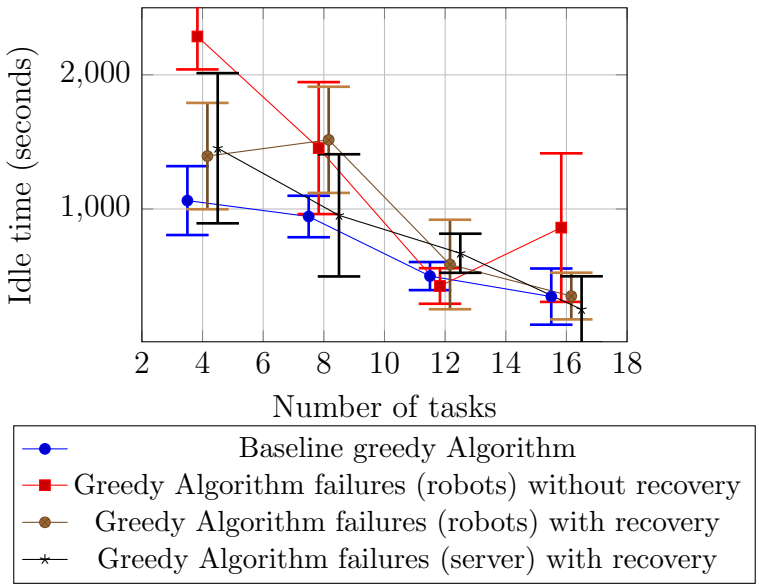


Figure B.13: Comparison of idle time metric for 12 robots without and with failures and server with failures, greedy Algorithm

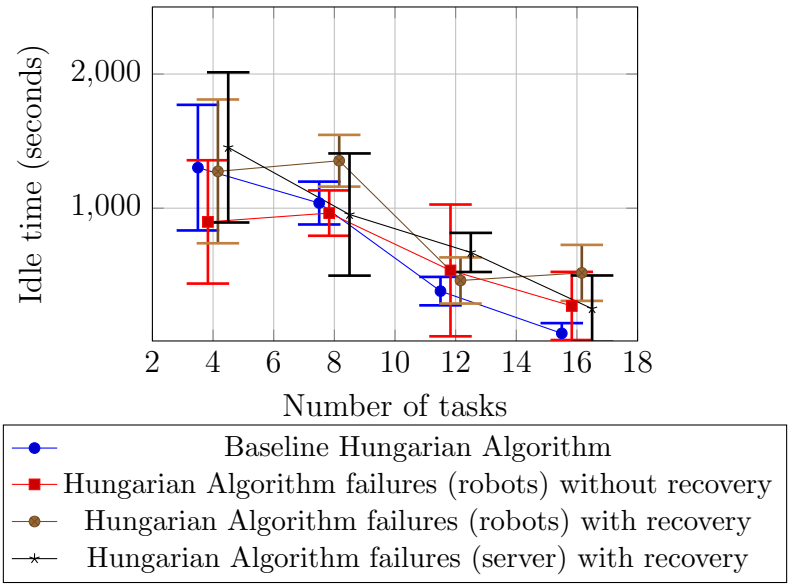


Figure B.14: Comparison of idle time metric for 12 robots without and with failures and server with failures, Hungarian Algorithm

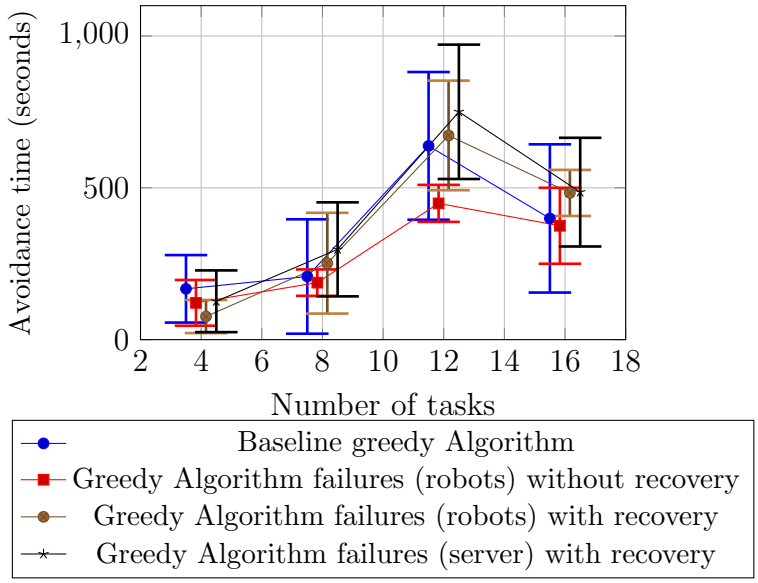


Figure B.15: Comparison of avoidance time metric for 12 robots without and with failures and server with failures, greedy Algorithm

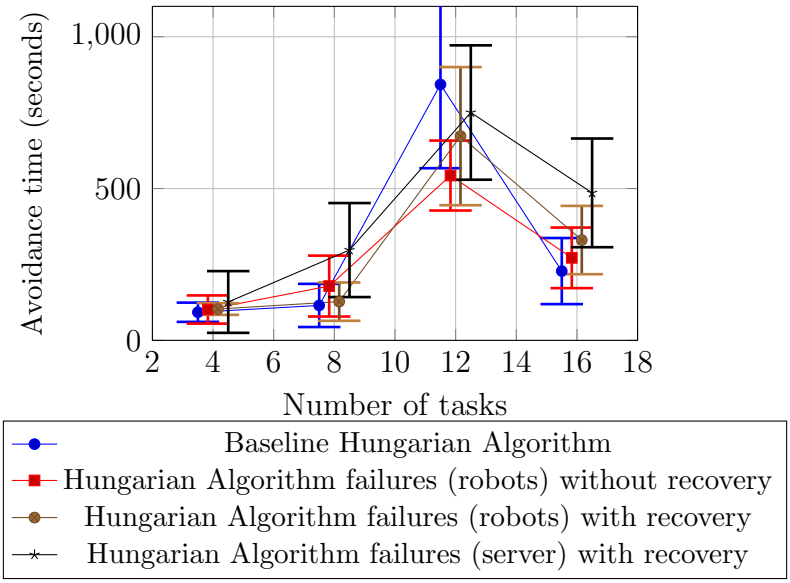


Figure B.16: Comparison of avoidance time metric for 12 robots without and with failures and server with failures, Hungarian Algorithm

Appendix C

Results for controlled environments for greedy and Hungarian Algorithms

Table C.1: Detailed report: 3 robots without failures, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	113	0	0	965
b	111	112	35	1145
c	119	1	139	1022
d	95	106	108	945
e	160	120	176	1718
Average	120	67	91	1159
Standard deviation	21	55	65	288
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	153	0	0	1525
b	136	0	0	1317
c	148	0	128	1349
d	152	321	0	1456
e	163	0	0	1725
Average	150	64	25	1474
Standard deviation	8	128	51	145
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	166	0	0	1460
b	162	30	93	1507
c	140	143	35	1292
d	175	493	0	1800
e	187	0	17	1926
Average	166	133	29	1597
Standard deviation	15	187	34	232
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	190.	0	0	1706
b	174	0	0	1609
c	174	0	17	1598
d	174	71	159	1556
e	170	0	111	1733
Average	176	14	57	1640
Standard deviation	6	28	65	67

Table C.2: Detailed report: 3 robots without failures, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	114	0	0	895
b	112	105	32	1072
c	119	0	138	1008
d	107	0	136	1075
e	161	102	178	1700
Average	122	41	97	1150
Standard deviation	20	51	68	283
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	156	0	0	1424
b	163	142	37	1606
c	148	0	110	1318
d	149	625	38	1543
e	163	0	0	1703
Average	156	153	37	1519
Standard deviation	7	242	40	135
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	166	0	0	1442
b	164	33	127	1498
c	141	30	0	1368
d	175	285	0	1752
e	175	0	36	1591
Average	164	70	33	1530
Standard deviation	12	109	49	133
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	190	0	0	1661
b	169	0	0	1573
c	172	0	0	1612
d	173	64	171	1522
e	170	0	109	1706
Average	175	13	56	1615
Standard deviation	8	26	71	65

Table C.3: Detailed report: 6 robots without failures, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	137	1042	56	1034
b	118	1057	0	775
c	130	705	101	990
d	134	1539	181	984
e	148	625	109	1043
Average	134	993	89	965
Standard deviation	9	323	60	97
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	233	42	35	1191
b	184	78	104	925
c	185	81	177	922
d	178	0	74	909
e	221	11	111	1187
Average	200	42	100	1026
Standard deviation	22	33	46	132
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	207	0	40	976
b	188	0	0	962
c	195	0	157	1145
d	168	0	68	829
e	230	15	140	1248
Average	198	3	81	1032
Standard deviation	20	6	59	147
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	192	0	34	1027
b	230	27	35	1122
c	214	106	19	1117
d	208	24	87	1123
e	263	0	51	1323
Average	221	31	45	1142
Standard deviation	24	39	23	97

Table C.4: Detailed report: 6 robots without failures, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	127	683	0	834
b	118	1050	54	793
c	106	1041	56	743
d	110	593	90	841
e	131	894	127	1059
Average	118	852	65	854
Standard deviation	10	186	42	108
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	186	0	0	957
b	179	0	36	1007
c	185	0	180	900
d	187	0	203	1005
e	215	152	85	1046
Average	190	30	101	983
Standard deviation	13	61	79	50
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	191	8	16	957
b	204	62	16	891
c	195	10	166	807
d	183	28	182	883
e	226	0	129	1113
Average	200	22	102	930
Standard deviation	15	22	72	103
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	240	0	54	1190
b	201	123	111	942
c	202	0	0	1091
d	232	47	257	1117
e	266	0	65	1307
Average	228	34	97	1129
Standard deviation	24	48	87	120

Table C.5: Detailed report: 9 robots without failures, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	143	1177	89	1047
b	121	1055	0	774
c	125	703	94	1019
d	153	844	470	1099
e	155	684	167	1165
Average	139	892	164	1020
Standard deviation	13	194	161	133
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	262	611	89	1144
b	212	546	144	836
c	250	660	418	952
d	232	785	111	1029
e	266	708	205	1269
Average	244	662	193	1046
Standard deviation	20	81	118	150
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	249	0	206	890
b	240	3	184	887
c	221	0	290	858
d	216	71	433	976
e	249	6	149	967
Average	235	16	252	915
Standard deviation	13	27	101	47
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	241	0	95	904
b	256	0	77	994
c	243	95	176	1154
d	243	0	201	921
e	270	64	136	1140
Average	251	31	137	1022
Standard deviation	11	40	46	106

Table C.6: Detailed report: 9 robots without failures, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	115	1413	105	745
b	96	713	36	635
c	97	889	35	644
d	102	1126	75	724
e	101	835	132	707
Average	102	995	77	691
Standard deviation	7	248	38	44
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	237	564	0	996
b	213	582	232	921
c	246	752	606	1073
d	206	779	290	771
e	253	693	242	967
Average	231	674	274	946
Standard deviation	18	87	194	100
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	247	36	75	928
b	272	818	231	1036
c	217	18	248	902
d	241	18	547	972
e	281	324	168	1031
Average	252	243	254	974
Standard deviation	23	310	159	54
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	239	0	93	890
b	240	17	198	960
c	231	107	133	1080
d	220	0	155	899
e	230	0	21	880
Average	232	25	120	942
Standard deviation	7	42	60	75

Table C.7: Detailed report: 12 robots without failures, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	162	1313	75	1023
b	125	1047	0	789
c	160	1353	249	1369
d	141	949	293	1094
e	160	651	220	1056
Average	150	1062	167	1066
Standard deviation	14	256	111	185
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	263	1150	112	960
b	238	880	99	1058
c	266	741	584	1205
d	220	852	135	941
e	277	1097	110	1104
Average	253	944	208	1053
Standard deviation	20	154	188	96
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	382	316	464	1121
b	341	488	629	1032
c	369	566	1057	1418
d	335	628	697	1051
e	393	501	344	1298
Average	364	499	638	1184
Standard deviation	22	104	243	150
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	352	619	345	1205
b	310	582	751	1431
c	294	151	187	1215
d	327	153	604	1097
e	311	229	109	1098
Average	319	346	399	1209
Standard deviation	19	209	244	121

Table C.8: Detailed report: 12 robots without failures, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	99	2048	135	596
b	87	839	105	550
c	92	919	82	589
d	98	1647	102	668
e	104	1058	39	673
Average	96	1302	93	615
Standard deviation	6	468	32	48
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	190	1104	105	717
b	196	730	184	680
c	163	1177	196	567
d	161	1133	90	513
e	203	1047	0	711
Average	183	1038	115	638
Standard deviation	17	160	71	82
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	376	244	670	959
b	305	289	643	814
c	386	391	1330	1135
d	300	446	603	796
e	372	538	968	1001
Average	348	382	843	941
Standard deviation	37	106	276	125
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	227	0	171	655
b	268	75	89	746
c	277	50	241	824
d	273	0	419	879
e	282	208	221	856
Average	265	67	228	792
Standard deviation	20	76	109	82

Table C.9: Detailed report: 3 robots with failures without recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	75	0	0	1805
b	69	0	0	1805
c	78	0	89	1805
d	62	0	71	1805
e	98	118	128	1805
Average	76	23	57	1805
Standard deviation	11	47	50	0
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	96	0	39	2305
b	98	0	78	2305
c	95	0	37	2305
d	105	144	78	2305
e	97	0	0	2305
Average	98	28	46	2305
Standard deviation	3	57	29	0
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	120	0	137	2598
b	106	0	36	2597
c	96	0	60	2598
d	149	0	981	2598
e	132	0	0	2598
Average	121	0	242	2597
Standard deviation	18	0	371	0
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	130	0	58	2806
b	119	0	0	2806
c	133	0	179	2805
d	113	0	56	2806
e	125	0	75	2806
Average	124	0	73	2805
Standard deviation	7	0	58	0

Table C.10: Detailed report: 3 robots with failures without recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	80	0	0	1795
b	69	0	0	1794
c	80	0	114	1795
d	63	0	74	1795
e	98	97	121	1794
Average	78	19	62	1795
Standard deviation	12	39	53	0
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	113	0	114	2295
b	118	0	154	2294
c	95	0	35	2294
d	98	279	38	2294
e	97	0	0	2294
Average	104	56	68	2294
Standard deviation	9	112	57	0
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	132	0	198	2587
b	107	0	39	2587
c	99	0	59	2587
d	116	0	133	2587
e	118	0	0	2587
Average	114	0	86	2587
Standard deviation	11	0	71	0
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	130	0	58	2795
b	120	0	0	2795
c	122	0	20	2795
d	113	0	55	2795
e	118	0	20	2795
Average	121	0	31	2795
Standard deviation	6	0	22	0

Table C.11: Detailed report: 6 robots with failures without recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	110	742	36	942
b	110	1241	38	1248
c	116	1052	228	1346
d	108	985	210	1103
e	121	569	152	988
Average	113	917	132	1125
Standard deviation	4	236	82	152
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	156	0	0	1197
b	150	56	61	1364
c	151	0	235	1169
d	154	0	159	1092
e	172	0	153	1130
Average	157	11	121	1190
Standard deviation	7	22	82	93
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	165	0	42	1181
b	148	0	0	1029
c	154	36	139	1305
d	126	59	201	997
e	170	0	35	1150
Average	153	19	83	1132
Standard deviation	15	24	74	110
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	202	0	82	1363
b	179	0	124	1367
c	160	3	230	1192
d	188	0	102	1482
e	196	0	41	1662
Average	185	0	115	1413
Standard deviation	14	1	63	155

Table C.12: Detailed report: 6 robots with failures without recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	114	506	0	892
b	106	519	0	1012
c	84	1103	18	884
d	111	1366	135	1176
e	114	487	115	1085
Average	106	796	54	1010
Standard deviation	11	368	59	112
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	147	0	55	1155
b	144	67	49	952
c	136	38	92	923
d	129	11	144	883
e	188	0	125	1349
Average	149	23	93	1052
Standard deviation	20	26	37	176
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	176	0	101	1305
b	163	0	81	1130
c	160	0	94	1159
d	188	55	357	1295
e	178	0	95	1279
Average	173	11	146	1234
Standard deviation	10	22	106	74
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	199	0	170	1324
b	189	18	390	1336
c	136	119	80	1018
d	170	74	95	1186
e	205	0	251	1486
Average	180	42	197	1270
Standard deviation	25	47	114	158

Table C.13: Detailed report: 9 robots with failures without recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	91	2465	114	1461
b	94	1877	56	1046
c	92	2392	74	1459
d	133	2309	356	1459
e	95	2801	35	1459
Average	101	2368	127	1376
Standard deviation	16	297	117	165
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	228	344	239	1334
b	187	445	197	1216
c	195	411	269	1125
d	187	905	185	1176
e	274	539	268	1552
Average	214	528	231	1280
Standard deviation	33	198	35	152
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	225	80	177	1169
b	179	281	213	1020
c	168	303	172	967
d	199	293	445	1126
e	238	139	228	1174
Average	202	219	247	1091
Standard deviation	26	91	101	83
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	174	0	37	952
b	191	181	279	1090
c	221	324	235	1297
d	192	241	185	1175
e	195	0	174	1011
Average	195	149	182	1105
Standard deviation	15	130	81	121

Table C.14: Detailed report: 9 robots with failures without recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	99	1039	45	773
b	98	726	57	655
c	89	1179	104	960
d	93	900	154	901
e	98	1063	34	743
Average	95	981	79	806
Standard deviation	4	155	45	110
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	196	371	16	989
b	190	586	308	1061
c	206	319	476	1177
d	150	444	265	856
e	250	431	318	1387
Average	198	430	277	1094
Standard deviation	32	90	149	180
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	213	240	134	1088
b	179	474	68	926
c	169	57	346	994
d	208	111	583	1170
e	210	178	195	1048
Average	196	212	265	1045
Standard deviation	18	145	184	83
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	188	0	37	1004
b	188	230	270	1091
c	207	325	342	1149
d	196	95	190	1009
e	204	0	337	1109
Average	196	130	235	1072
Standard deviation	8	129	113	57

Table C.15: Detailed report: 12 robots with failures without recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	131	2186	218	1565
b	84	2297	34	1031
c	88	2450	92	1370
d	108	1887	203	1193
e	119	2611	59	1613
Average	106	2286	121	1354
Standard deviation	17	245	75	220
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	206	2160	209	1383
b	185	724	185	963
c	184	1310	235	1032
d	167	1270	203	1022
e	219	1806	107	1362
Average	192	1454	187	1152
Standard deviation	18	491	43	181
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	313	201	389	1286
b	254	390	390	1137
c	239	535	517	1117
d	258	581	527	1206
e	332	423	421	1475
Average	279	426	448	1244
Standard deviation	36	132	60	129
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	337	471	555	1637
b	234	1942	225	1847
c	242	503	249	1301
d	251	816	461	1333
e	258	574	385	1814
Average	264	861	375	1586
Standard deviation	37	553	125	231

Table C.16: Detailed report: 12 robots with failures without recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	88	1796	86	865
b	84	740	118	531
c	83	485	179	716
d	86	752	87	735
e	84	715	38	649
Average	85	898	102	699
Standard deviation	2	460	46	109
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	163	1092	97	788
b	156	800	88	788
c	145	955	262	614
d	149	763	334	827
e	202	1207	113	892
Average	163	963	179	782
Standard deviation	21	169	100	92
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	285	273	400	1260
b	238	223	404	996
c	265	1513	650	1105
d	256	288	632	894
e	280	383	628	1055
Average	265	536	543	1062
Standard deviation	17	491	115	121
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	247	28	313	977
b	247	224	124	719
c	217	102	187	845
d	199	753	348	1249
e	230	248	387	907
Average	228	271	272	939
Standard deviation	18	254	100	177

Table C.17: Detailed report: 3 robots with failures with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	113	0	0	1551
b	99	669	0	1403
c	116	0	124	1539
d	101	0	122	1532
e	145	92	174	1805
Average	115	152	84	1566
Standard deviation	16	260	71	130
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	144	0	0	2036
b	140	0	58	1887
c	142	0	108	1971
d	149	200	0	2008
e	165	348	34	2068
Average	148	109	40	1994
Standard deviation	8	142	40	62
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	167	0	20	2285
b	157	0	0	2194
c	138	0	0	2020
d	173	0	140	2289
e	190	0	19	2328
Average	165	0	35	2223
Standard deviation	17	0	52	110
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	188	0	0	2512
b	175	0	96	2351
c	176	0	39	2260
d	167	0	57	2304
e	170	0	91	2296
Average	175	0	56	2344
Standard deviation	7	0	35	88

Table C.18: Detailed report: 3 robots with failures with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	115	50	0	1283
b	110	189	34	1297
c	117	181	144	1293
d	98	146	166	1109
e	136	104	165	1383
Average	115	134	102	1273
Standard deviation	12	52	71	90
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	170	0	0	1904
b	155	145	159	1627
c	139	0	51	1441
d	134	539	20	1514
e	167	0	146	1789
Average	153	137	75	1655
Standard deviation	14	209	65	171
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	163	0	70	1847
b	160	0	39	1857
c	158	0	110	1860
d	152	0	36	1617
e	175	51	72	1793
Average	162	10	65	1795
Standard deviation	8	20	27	92
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	192	0	56	2048
b	184	0	40	1975
c	178	0	32	1879
d	159	0	59	1742
e	188	0	19	1992
Average	180	0	41	1927
Standard deviation	12	0	15	107

Table C.19: Detailed report: 6 robots with failures with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	139	757	0	1051
b	119	975	36	1033
c	142	1046	184	1474
d	152	1916	369	1321
e	141	1584	76	1132
Average	139	1255	133	1202
Standard deviation	10	427	133	169
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	232	133	36	1550
b	181	82	89	1395
c	191	0	234	1428
d	183	45	113	1166
e	222	122	265	1529
Average	202	76	147	1413
Standard deviation	21	49	87	136
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	202	0	40	1328
b	191	0	0	1306
c	187	588	173	1377
d	177	0	168	941
e	221	299	119	1490
Average	195	177	100	1288
Standard deviation	14	235	69	184
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	257	351	150	1527
b	213	20	206	1136
c	237	600	178	1538
d	223	0	191	1402
e	255	89	79	1439
Average	237	212	160	1408
Standard deviation	17	230	44	145

Table C.20: Detailed report: 6 robots with failures with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	149	1093	32	1027
b	120	946	32	804
c	127	1351	88	970
d	118	857	175	940
e	134	1214	165	1014
Average	130	1092	98	951
Standard deviation	11	178	62	80
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	184	47	17	1035
b	196	13	224	1053
c	201	107	237	1229
d	201	543	110	1141
e	236	29	183	1369
Average	204	148	154	1165
Standard deviation	17	200	82	123
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	193	50	73	987
b	189	119	73	961
c	180	32	18	958
d	219	27	305	1083
e	227	0	67	1079
Average	202	46	107	1014
Standard deviation	18	40	101	56
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	254	238	107	1240
b	220	4	120	1044
c	201	0	49	1115
d	236	229	465	1237
e	235	0	70	1313
Average	229	94	162	1190
Standard deviation	18	114	154	97

Table C.21: Detailed report: 9 robots with failures with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	135	986	0	1124
b	142	1125	56	1159
c	140	1698	186	1429
d	161	1457	290	1307
e	168	1565	202	1334
Average	149	1366	146	1270
Standard deviation	12	268	104	113
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	259	670	145	1254
b	222	956	186	1184
c	254	619	446	1120
d	240	789	382	1154
e	284	765	389	1202
Average	252	759	309	1182
Standard deviation	20	115	120	45
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	252	87	79	1183
b	245	67	264	952
c	240	233	207	1155
d	252	217	517	1351
e	278	65	352	1160
Average	253	133	283	1160
Standard deviation	13	75	146	126
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	274	20	297	1298
b	228	58	82	1224
c	241	53	206	1174
d	217	89	235	998
e	281	19	188	1379
Average	248	47	201	1214
Standard deviation	25	26	70	128

Table C.22: Detailed report: 9 robots with failures with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	114	1800	140	845
b	97	732	39	656
c	104	1369	72	900
d	115	1371	219	773
e	119	978	162	841
Average	110	1250	126	803
Standard deviation	8	367	64	84
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	278	926	300	1117
b	234	755	367	969
c	211	791	362	982
d	220	1010	193	911
e	275	934	143	1004
Average	244	883	273	997
Standard deviation	28	95	90	68
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	270	36	202	985
b	223	118	179	933
c	211	46	122	858
d	255	30	611	1122
e	283	259	247	1099
Average	248	98	272	999
Standard deviation	27	87	174	100
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	276	54	214	1021
b	216	98	57	853
c	238	13	151	961
d	237	82	288	1089
e	243	784	168	992
Average	242	206	176	983
Standard deviation	19	290	76	78

Table C.23: Detailed report: 12 robots with failures with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	160	1507	36	1453
b	138	715	0	1316
c	153	1445	132	1398
d	139	1356	142	1500
e	171	1949	71	1122
Average	152	1394	76	1357
Standard deviation	12	396	54	132
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	276	981	41	1130
b	230	1200	72	1121
c	240	1584	340	1051
d	276	2115	464	1187
e	321	1698	342	1582
Average	269	1515	251	1214
Standard deviation	32	395	165	188
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	409	294	427	1371
b	354	916	656	1583
c	329	595	592	1221
d	326	988	980	1222
e	453	138	708	1355
Average	374	586	672	1350
Standard deviation	49	333	180	132
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	395	262	582	1459
b	303	684	398	1292
c	313	178	557	1182
d	282	317	404	941
e	298	314	475	1183
Average	318	351	483	1211
Standard deviation	39	173	75	168

Table C.24: Detailed report: 12 robots with failures with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	102	2199	88	638
b	91	833	107	520
c	91	990	123	623
d	109	1553	123	580
e	105	797	75	683
Average	100	1274	103	609
Standard deviation	7	536	19	55
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	182	1373	113	781
b	208	1449	230	760
c	166	1067	38	596
d	166	1639	153	635
e	178	1240	104	716
Average	180	1354	128	698
Standard deviation	15	193	63	71
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	329	229	479	927
b	306	628	443	938
c	354	594	823	1097
d	353	275	1042	991
e	380	579	576	1101
Average	344	461	673	1011
Standard deviation	25	172	227	75
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	256	692	257	886
b	288	203	195	840
c	278	798	528	986
d	267	445	351	979
e	308	450	321	1039
Average	279	518	330	946
Standard deviation	18	209	112	72

Table C.25: Detailed report: 3 robots, server fails with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	114	0	0	954
b	111	112	34	1131
c	120	8	194	1094
d	107	0	160	1091
e	161	88	163	1706
Average	123	42	110	1195
Standard deviation	19	48	78	262
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	153	0	0	1487
b	151	42	34	1365
c	147	0	74	1325
d	150	566	74	1564
e	163	0	0	1709
Average	153	122	36	1490
Standard deviation	6	223	33	139
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	163	0	0	1441
b	143	0	0	1300
c	149	0	34	1439
d	158	378	69	1532
e	175	0	34	1578
Average	158	76	27	1458
Standard deviation	11	151	26	95
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	193	0	33	1718
b	166	0	0	1443
c	173	0	0	1638
d	180	138	51	1659
e	169	0	53	1646
Average	176	28	27	1621
Standard deviation	10	55	23	93

Table C.26: Detailed report: 3 robots, server fails with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	128	0	0	1269
b	112	111	34	1135
c	119	0	137	1020
d	106	138	154	1069
e	161	94	192	1670
Average	125	69	103	1233
Standard deviation	20	58	74	234
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	145	0	0	1286
b	150	0	36	1410
c	149	0	111	1327
d	151	572	93	1572
e	157	110	32	1360
Average	150	136	54	1391
Standard deviation	4	222	41	99
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	164	0	0	1389
b	168	268	37	1857
c	149	0	34	1439
d	158	368	68	1525
e	175	0	34	1584
Average	163	127	35	1559
Standard deviation	9	159	22	164
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	193	0	34	1704
b	166	0	0	1469
c	166	0	17	1511
d	170	56	77	1454
e	169	0	53	1676
Average	173	11	36	1563
Standard deviation	10	22	27	106

Table C.27: Detailed report: 6 robots, server fails with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	134	1046	51	1118
b	120	958	16	945
c	146	1338	144	988
d	132	1366	251	1345
e	155	827	111	1272
Average	137	1107	115	1134
Standard deviation	12	212	82	155
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	233	54	18	1202
b	282	178	263	1334
c	258	63	319	1217
d	250	379	207	1367
e	287	483	56	1598
Average	262	231	173	1344
Standard deviation	20	172	117	142
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	62	0	0	1091
b	193	17	18	1023
c	187	0	17	1140
d	184	0	90	949
e	220	332	83	1226
Average	169	70	42	1086
Standard deviation	55	131	37	95
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	253	16	133	1238
b	213	13	0	1094
c	226	172	145	1411
d	204	5	132	1119
e	227	0	36	1251
Average	225	41	89	1223
Standard deviation	16	66	59	113

Table C.28: Detailed report: 6 robots, server fails with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	129	755	95	954
b	108	733	197	914
c	102	1232	16	795
d	109	693	224	888
e	121	823	181	994
Average	114	847	143	909
Standard deviation	10	197	77	67
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	186	0	0	969
b	176	1	57	936
c	183	155	70	950
d	184	0	136	1196
e	235	190	164	1294
Average	193	69	85	1069
Standard deviation	21	85	58	147
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	190	185	18	1051
b	209	0	77	1097
c	176	0	72	990
d	210	126	130	1260
e	227	167	69	1315
Average	203	96	73	1143
Standard deviation	18	80	36	124
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	239	123	81	1181
b	207	0	53	1061
c	212	0	94	1230
d	229	0	166	1104
e	251	222	192	1509
Average	227	69	117	1217
Standard deviation	17	90	53	157

Table C.29: Detailed report: 9 robots, server fails with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	209	3007	133	1224
b	170	2808	161	1209
c	196	2889	402	1478
d	171	2777	331	982
e	232	3124	164	1486
Average	196	2921	238	1276
Standard deviation	24	129	108	189
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	316	1205	142	1294
b	277	1372	192	1242
c	224	235	255	947
d	242	976	332	1007
e	277	948	218	1444
Average	267	947	228	1187
Standard deviation	32	389	64	185
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	268	3	107	1134
b	224	0	80	837
c	230	0	377	1027
d	185	147	127	824
e	268	0	187	1081
Average	235	30	176	981
Standard deviation	31	59	107	127
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	236	0	176	902
b	318	358	215	1392
c	248	351	36	1384
d	225	342	121	1139
e	248	49	151	1029
Average	255	220	140	1169
Standard deviation	32	160	60	194

Table C.30: Detailed report: 9 robots, server fails with recovery, Hungarian Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	111	1796	194	948
b	95	610	166	767
c	109	2087	138	1224
d	87	1411	119	794
e	101	1035	37	660
Average	101	1388	131	879
Standard deviation	9	526	53	196
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	212	1337	117	982
b	186	961	264	869
c	208	702	208	790
d	181	692	325	840
e	260	1220	336	1072
Average	209	982	250	911
Standard deviation	28	263	81	102
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	270	45	95	1048
b	214	64	95	884
c	214	162	328	814
d	265	49	699	1130
e	226	103	44	969
Average	238	85	252	969
Standard deviation	25	44	244	113
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	248	0	93	1127
b	290	1481	191	1180
c	215	18	139	866
d	220	0	282	974
e	254	0	65	1018
Average	245	300	154	1033
Standard deviation	27	591	77	111

Table C.31: Detailed report: 12 robots, server fails with recovery, greedy Algorithm

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	155	1702	39	1105
b	121	628	70	811
c	161	956	185	1223
d	156	1961	299	1465
e	180	2018	39	1621
Average	155	1453	126	1245
Standard deviation	19	560	102	282
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	256	231	132	1018
b	223	623	182	873
c	248	1147	470	1049
d	245	1341	499	1077
e	258	1421	204	1109
Average	246	953	297	1025
Standard deviation	13	455	155	82
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	387	830	381	1174
b	322	609	637	1057
c	336	733	1019	1570
d	327	416	871	1055
e	442	762	844	1366
Average	363	670	750	1244
Standard deviation	46	146	221	198
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	428	741	791	1698
b	299	128	371	1098
c	299	176	598	1096
d	332	228	552	1277
e	308	0	201	1231
Average	333	255	503	1280
Standard deviation	49	255	201	221

Appendix D

Results for random environments for greedy Algorithm

Table D.1: Detailed report: 3 robots without failures, random environments

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	100	217	34	1057
b	105	317	50	1273
c	98	250	67	1078
d	109	230	0	1133
e	134	108	87	1390
Average	109	224	48	1186
Standard deviation	13	68	30	127
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	175	78	17	1793
b	187	248	139	1663
c	156	166	17	1456
d	126	288	17	1318
e	186	0	0	1705
Average	166	156	38	1587
Standard deviation	23	106	51	174
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	201	108	84	1901
b	236	0	17	2120
c	214	947	85	2346
d	208	63	17	1865
e	162	0	17	1426
Average	204	224	44	1932
Standard deviation	24	364	33	306
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	253	39	34	2301
b	268	147	77	2510
c	213	86	34	1865
d	214	0	0	1888
e	254	0	0	2438
Average	240	54	29	2200
Standard deviation	23	56	28	273

Table D.2: Detailed report: 6 robots without failures, random environments

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	123	548	17	854
b	121	707	41	954
c	97	845	241	976
d	141	661	75	1066
e	151	621	151	999
Average	127	676	105	970
Standard deviation	19	99	82	69
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	224	29	34	1303
b	196	268	160	1063
c	172	264	52	1025
d	189	70	187	900
e	196	22	68	1183
Average	195	131	100	1095
Standard deviation	17	112	61	138
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	228	190	54	1327
b	259	0	96	1296
c	234	414	37	1624
d	242	33	35	1201
e	222	38	104	1133
Average	237	135	65	1316
Standard deviation	13	154	29	169
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	290	118	86	1664
b	307	349	156	1571
c	242	458	236	1420
d	299	86	123	1385
e	250	187	34	1432
Average	277	240	127	1494
Standard deviation	26	142	68	106

Table D.3: Detailed report: 9 robots without failures, random environments

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	96	822	80	738
b	108	679	95	854
c	112	597	143	794
d	122	1040	120	927
e	131	668	196	1029
Average	114	761	127	868
Standard deviation	12	157	41	102
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	239	553	155	1089
b	240	782	172	1145
c	231	565	335	1054
d	231	698	287	1201
e	244	494	21	1054
Average	237	618	194	1109
Standard deviation	5	106	110	57
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	276	36	183	1152
b	298	285	170	1269
c	253	677	466	1454
d	271	111	188	1184
e	216	15	36	808
Average	263	225	209	1173
Standard deviation	27	245	140	211
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	296	0	152	1139
b	283	276	96	1211
c	258	64	159	1258
d	285	96	480	1148
e	275	342	75	1495
Average	279	156	192	1250
Standard deviation	13	131	147	130

Table D.4: Detailed report: 12 robots without failures, random environments

4 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	119	468	38	795
b	138	925	192	1046
c	117	874	73	1150
d	154	1310	254	1168
e	152	701	176	1064
Average	136	856	147	1045
Standard deviation	16	278	80	133
8 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	304	1428	42	1330
b	294	963	393	1188
c	222	1117	249	888
d	260	648	362	1217
e	271	986	231	1096
Average	270	1028	255	1144
Standard deviation	29	252	124	148
12 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	427	766	481	1522
b	449	414	434	1627
c	363	691	557	1281
d	394	621	293	1168
e	335	362	502	955
Average	394	571	453	1311
Standard deviation	41	157	89	242
16 tasks				
Environment	TOTAL DISTANCE (meters)	IDLE TIME (seconds)	AVOIDANCE TIME (seconds)	COMPLETION TIME (seconds)
a	336	58	458	967
b	378	177	225	1068
c	366	400	207	1308
d	388	485	430	1138
e	358	182	354	1238
Average	365	260	335	1144
Standard deviation	18	157	103	121

Bibliography

Botelho, S. C. and Alami, R. (1999). M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 2:1234–1239.

Brassard, G. and Bratley, P. (1996). *Fundamentals of algorithmics*. Prentice-Hall, Inc.

Bullo, F., Frazzoli, E., Pavone, M., Savla, K., and Smith, S. L. (2011). Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE*, 99(9):1482–1504.

Cao, Y. U., Fukunaga, A. S., and Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27.

Dahl, T. S., Matarić, M., and Sukhatme, G. S. (2009). Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems*, 57(6):674–687.

Dasgupta, P. (2011). Multi-robot task allocation for performing cooperative foraging tasks in an initially unknown environment. *Innovations in Defence Support Systems-2*, pages 5–20.

Dias, M. B., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE, Special Issue on Multi-Robot Systems*, 94(7):1257–1270.

Geppert, A., Kradolfer, M., and Tombros, D. (1998). Market-based workflow management. *International Journal of Cooperative Information Systems*, 7(04):297–314.

Gerkey, B. P. and Matarić, M. J. (2002). Sold!: Auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, 18(5):758–768.

Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954.

Kalra, N. and Martinoli, A. (2006). Comparative study of market-based and threshold-based task allocation. *Distributed autonomous robotic systems 7*, pages 91–101.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Lagoudakis, M. G., Markakis, E., Kempe, D., Keskinocak, P., Kleywegt, A. J., Koenig, S., Tovey, C. A., Meyerson, A., and Jain, S. (2005). Auction-based multi-robot routing. *Robotics: Science and Systems*, 5.

Lenagh, W. H. (2013). Multi-robot task allocation: a spatial queuing approach. Master's thesis, University of Nebraska at Omaha.

Matarić, M. J., Sukhatme, G. S., and Østergaard, E. H. (2003). Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14(2-3):255–263.

Miller, D., Dasgupta, P., and Judkins, T. (2007). Distributed task selection in multi-agent based swarms using heuristic strategies. *Swarm Robotics*, pages 158–172.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38.

Munoz-Meléndez, A., Dasgupta, P., and Lenagh, W. (2012). A stochastic queueing model for multi-robot task allocation. *Proceedings of the International Cconference on Informatics in Control, Automation and Robotics(1)*, pages 256–261.

Seow, K. T., Dang, N. H., and Lee, D.-H. (2010). A collaborative multiagent taxi-dispatch system. *Automation Science and Engineering, IEEE Transactions on*, 7(3):607–616.

Trojanek, P., Zielinski, C., and Szynkiewicz, W. (2007). Definition and composition of individual robot behaviours in cooperative box pushing. *IEEE Conference Number*, 12459:1137.

Viguria, A., Maza, I., and Ollero, A. (2007). Set: An algorithm for distributed multirobot task allocation with dynamic negotiation based on task subsets. *Robotics and Automation, 2007 IEEE International Conference on*, pages 3339–3344.