Control of Heterogeneous Robot Networks for Assistance in Search and Rescue Tasks

David Reinerio Yanguas Rojas

National University of Colombia
Engineering Faculty, Electronic and Electric Engineering Department
Bogotá, Colombia
2018

Control of Heterogeneous Robot Networks for Assistance in Search and Rescue Tasks

David Reinerio Yanguas Rojas

Thesis presented as partial requirement for apply to the tittle of:
Master in Industrial Automation

Director:
Ph.D.Eduardo Alirio Mojica Nava

Research Line:
Control and Robotics
Research Group:
PAAS - Processing, Acquisition and Analysis of Signals

National University of Colombia
Engineering Faculty, Electronic and Electric Engineering Department
Bogotá, Colombia
2018

Dedication

To my beloved mother Ana Lilia, who has been my most precious support in all my projects

Acknowledgements

I would like to thank COLCIENCIAS for the opportunity and the financial support that made this project possible through the call 706 "Convocatoria nacional jóvenes investigadores e innovadores 2015". I would like to thank my friends encouraging me in the development of this and many other projects. Last but not the least, I would like to thank my family: my mother and grand mother for supporting me throughout writing this thesis and my life in general.

Resumen

Este proyecto desarrolló una estrategia de control descentralizado para múltiples robots heterogéneos orientada a la asistencia en situaciones de búsqueda y rescate desde dos perspectivas complementarias, la asignación discreta de tareas y el control en tiempo real. Para la asignación discreta de las tareas a los robots a lo largo de la misión, presentamos un algoritmo optimizado de reasignación de tareas basado en eventos, orientado a la minimización del tiempo requerido para atender a todas las víctimas en el ambiente de misión. Este algoritmo permite asignar a cada robot una tarea apropiada considerando que los robots pueden diferir en su capacidad para completar cada tarea, así como también en sus capacidades de movimiento. Las tareas consideradas son la exploración del ambiente de misión, la búsqueda e identificación de víctimas, la entrega de suministros médicos a las víctimas incapaces de moverse y la evacuación de las víctimas capaces de moverse. Cabe destacar que, durante el desarrollo de cada tarea y la estimación de los tiempos de las mismas, los robots consideran rutas optimizadas considerando una métrica de distancia basada en el algoritmo de búsqueda en anchura (Breath first Search) llamada distancia por inundación con 8 vecinos (8NF) la cual considera movimientos netamente ortogonales y diagonales a 45 grados permitiendo una estimación de la distancia geodésica a cada punto en el mapa.

Con respecto a las leyes de control en tiempo real, estas están a cargo de la correcta ejecución de las tareas asignadas por el algoritmo de reasignación de tareas respetando las restricciones en el rango de conectividad, la evasión de colisiones y la completa ejecución de cada tarea. La exploración es desarrollada empleando una adaptación del algoritmo DisCoverage presentado por [16] el cuál empleando una teselación basada en celdas de Voronoi con el tiempo de llegada a cada punto como referencia, permite la determinación del mapa de espacios no convexos como los que se pueden encontrar en algunas situaciones de búsqueda y rescate. La evasión de obstáculos y la preservación de los enlaces se realiza a través de un enfoque de potenciales artificiales basándose en el trabajo de [37]. El seguimiento de los puntos de interés relacionados a cada tarea se realiza empleando lazos de control proporcional para cada agente identificando los puntos de ruta dentro de la línea de visión y considerando rutas optimizadas tomando la estimación brindada por la métrica de distancia por inundación 8NF. Adicionalmente se presentó un algoritmo de reconfiguración de la red heurístico que permite cambiar la topología de la red manteniendo la conectividad de la misma para cada instante de tiempo.

Este marco de trabajo completo permite a un equipo de robots autónomos brindar asistencia valiosa en ciertas situaciones de búsqueda y rescate dónde los equipos humanos sean insuficientes y/o las condiciones de la misión pueden ser peligrosas para las personas teniendo en cuenta que si bien los robots actualmente no son capaces de realizar tareas paramédicas si son capaces de realizar múltiples tareas útiles para aligerar el trabajo y el riesgo para equipos humanos en estas situaciones. El funcionamiento de estos algoritmos es presentado en simulaciones no triviales en Matlab® buscando presentar los comportamientos que emer-

gen en los robots y adicionalmente fue implementado en una versión simplificada con robots móviles tipo turtlebot y configuraciones simples de robots Bioloid.

Palabras clave: 1) Robots heterogéneos, 2) redes de robots, 3) Control en tiempo real, 4) Asignación óptima de tareas , 5) Búsqueda y rescate.

Abstract

This project develops a decentralized control strategy for multiple heterogeneous robots oriented to the assistance in search and rescue situations from two complementary perspectives, the discrete tasks allocation and the real-time control. For the discrete task allocation through the mission, we present an optimized algorithm based on events, oriented to the minimization of the time required to attend all the victims in the mission environment. This algorithm allows assign to each robot an appropriate task considering that the robots may vary in their capacity for completing each task and also may vary in their moving capabilities. The considered tasks are the mission environment exploration, the victims' search and identification, the medical supplies delivery to victims unable to move and the evacuation of victims capable to move. It is worth to mention that, through the development of each task and the estimation of its durations, the robots consider optimized routes considering a distance metric based in the breath first search algorithm called flooding distance with 8 neighbors (8NF) which considers only orthogonal and 45 degrees diagonal movements allowing an estimation of the geodesic distance to each point in the map. Regarding to the real-time control laws, they oversee the proper execution of the tasks assigned by the reallocation algorithm respecting the restrictions in the connectivity range, the obstacles avoidance and the fulfillment of each task. The exploration task is made employing an adaptation of the algorithm DisCoverage presented by [16] which employing a Voronoi cells based tessellation considering the arrival time to each point as reference, allows the determination of the map of non-convex spaces as those that may be found in search and rescue situations. The evasion of obstacles and the preservation of the robots' links is achieved employing an approach of artificial potentials based in the work of [37]. The interest points related to each task tracking is made employing proportional control loops for each agent identifying the route points within the line of sight and considering optimized routes given by the 8NF flooding distance metric. Additionally, there is presented a heuristic reconfiguration algorithm that allows to change the network topology preserving its connectivity for each instant of time.

This complete framework allows a team of autonomous robots to bring valuable assistance in certain search and rescue situations where the human teams may be insufficient, and/or the mission conditions may be harmful for the people considering that even if the robots cannot realize paramedical tasks yet, they can complete multiple useful tasks for reducing the effort and risks of the human teams in that kind of situations. The functioning of those algorithms is presented in non-trivial simulations intended to show the behaviors that emerge in the robots.

Keywords: 1) Heterogeneous Robots, 2)Robots Networks, 3) Real Time Control, 4) Optimal Task Allocation , 5) Search and Rescue.

Contents

Symbols List

Symbols with Latin Letters

Symbol	Item
A	Robots set
A_m	Area pendant to be explored
c_i	sliding modes controller constant
C	Robot's gravitational matrix
$C_{i,j}^{Del}$	Delivery cost for the i-th robot to the victim j
C_i^e	Time cost for the i-th robot to explore
$C_{i,j}^{Ev}$	Time cost for the i-th robot to evacuate the victim j
$C_{i,j}^{ID}$	Time cost for the i-th robot to identify the victim j
D	Robot's inertia matrix
d_{ij}	Distance between the i-th robot and the j-th victim
D_{max}	Maximum link distance
G	Connection graph of the robots network
H	Robot's centripetal and Coriollis accelerations matrix
H_{exp}	Exploration objective function
J	Kinematic Jacobian matrix of a robot
k_i	Integration constant of the i-th robot
K_e	Exploration priority constant
K_{ti}	Tracking proportional constant of the i-th robot
LoS	Robot's line of sight
M	Mission environment
$\bar{M}$	Estimation of the mission environment
n	Number of robots

Symbol	Item
NCF	Neighbor closest to the frontier
$N_{NF}(i)$	Number of nodes to the frontier
$\tilde{N}_v$	Estimated number of remaining victims to be found
p	Position in the environment (or map)
p_d	Position in the environment (or map) traced back to the line of sight
p_s	Safe area position
q	Joint position of a robot
q_{ei}	Position error of the i-th robot's joint
r_C	robot's communication radius
r_{cv}	robot-victim communication radius
r_d	robot's delivery radius
r_e	robot's obstacles evasion radius
$r_{i,ID}$	i-th robot's victims identification radius
r_s	robot's sensing radius
S	Known space in the mission environment
t	time [seconds]
T	Robots' tasks allocation
T_i	i-th robot tasks assigned
T_{ip}	time required for the i-th robot to reach the position p [seconds]
$T_{i,Del}$	time required for the i-th robot to deliver a supply [seconds]
$T_{i,Ev}$	time required for the i-th robot to communicate evacuation instructions [seconds]
$T_{i,ID}$	time required for the i-th robot to identify a victim [seconds]
u_i	Control signal of the i-th robot
u_{poti}	Artificial potential component of the control signal of the i-th robot
u_{ti}	Task component of the control signal of the i-th robot
V_i	i-th robot's Voronoi cell
$\bar{v_i}$	average i-th robot's velocity
$\bar{v_j}$	average j-th victim's velocity
w	Feedback linearization input
x_i	Position of the i-th robot

Symbol	Item
x_{ij}	Distance between the i-th and j-th robot
x_{io}	Distance between the i-th robot and the obstacle closest to it

Symbols with Greek Letters

Symbol	Item
δS	Exploration Frontier
$\{\phi\}$	empty matrix or array
ρ_1	collisions avoidance radius for the artificial potential
ρ_2	communication maintenance radius for the artificial potential
σ_i	sliding surface of the i-th degree of freedom of a robot
τ	Robot's generalized torques vector
τ_0	Robot's sliding modes control constant
φ_w	angular position of the wheels of the Robotino robot
φ_{sw}	angular position of the sub-wheels of the Robotino robot
ψ_{cij}	Artificial collision and connectivity potential between the i-th and j-th robots
ψ_o	Artificial obstacles evasion potential between the i-th and j-th robots
ξ_I	Position of the robot regarding to its inertial reference frame

Abbreviations

Abbreviations	Item
4NF	8 neighbors flooding
8NF	8 neighbors flooding
AAAI	American Agency of Artificial Intelligence
DARPA	Defense Advanced Research Projects Agency
HRI	Human Robot Interface
SLAM	Simultaneous Localization and Mapping

Abbreviations	Item
VRC	Virtual Robot Competition

1. Introduction

Motivated by big disasters occurred in the recent history such as the Kobe earthquake in Japan, the World Trade Center Attack in the USA, and the Fukushima's nuclear disaster, the international community has been dedicating big efforts in the development of the rescue robotics, considering the big potential of this discipline for the attention of this class of situations, where multiple situations dangerous for the humans emerge and potentially can be solved or attended by the robots. The contribution of the robots in this kind of situations is focused on the tasks that may be very dangerous for the humans, for example those situations where may be conditions such as radiation, toxins in the air or danger of fall for structural damages between many others. This introductory chapter is intended to present part of the work developed by the international community for the advance of the robotics oriented to the rescue. Initially there are presented the typical task developed by the robots in search and rescue situations. Then there are mentioned some of the main challenges faced during the execution of multi-robot collaborative missions. And finally, there are introduced some of the most famous robotics competitions where the international community presents their advances in the area as well as the new challenges to be faced1.

1.1. Traditional Tasks for the Robots in Search and Rescue Situations

In the dangerous conditions that emerge in the search and rescue situations, the robots are usually used in the early stages of the situation as a fast response method that allows the reduction in the risks taken by the rescue personal. This fast response usually consists in the exploration and mapping of the disaster area taking special attention to the victims and locations of interest. Another tasks that are also possible include carry different elements such as medical supplies or even certain obstacles in order to clear paths and even there are some research groups that are working in coordination and control schemes for robots squads in charge of realize public forces tasks, imitating the task of the police, the firefighters and the medical teams. It is worth to mention that these tasks are not necessarily exclusives and in many cases a robots squad can accomplish two or more groups of tasks.

1.1.1. Exploration and Mapping

The task of Simultaneous Localization and Mapping (SLAM) is a problem that has been widely studied from many years ago by the robotics community. The application of this tasks is particularly required by the rescue robotics since in the first stages of an emergency it is required to make (or update) maps of the disaster zone. These maps are employed later for the teams of humans and robots can realize their tasks of identification and attention of victims in an appropriate way. Considering that in the first stages of the emergency, the environment may be very unstable and/or dangerous for the humans, the robots emerge as a practical solution to generate a fast response reducing the risk for the human rescue teams. In works as the presented in [10], there are presented robots specially designed to operate in disaster conditions and to realize victim detection tasks in emergency situations, these situations require the mapping of the environment in order to report the position of the victims found to the human rescue teams. Other works as the presented in [2], introduce algorithms to improve the accuracy of the location estimation of robot teams in order to optimize those tasks, particularly the strategy taken by those authors consists in the employ of the information provided by the wireless communication network together with machine learning algorithms to realize a map of the disaster environment with an heterogeneous robots team. In recent years the development of rescue tasks in general, aims to the employ of multiple low-cost robots instead of a single or a few specialized robots in order to increase their reach and make the development of the tasks more robust regarding to the loss or the damage of some robots as presented in works as the presented in [28] and [12]. However, there are only multiple works as the presented in [10] and [21] where there are noticeable efforts made for the development of structures and algorithms for a few or even one single robot for the operation in this kind of situations.

1.1.2. Supplies Delivery

In works as [23] or in competition scenarios as the DARPA VRC (Virtual Robot Challenge), there are efforts made to the coordination of robots that allow the delivery of certain supplies (medical ones for example) to a determined location in the disaster environment in order to make easier the attention of the emergency. In particular, this work presents a robot coordinated team that consist in a flying robot in charge of exploring and determining the best routes and an land robot that has the capability of moving obstacles and deliver a first aid kit in controlled conditions.

1.1.3. Public Forces Tasks

In emergency situations there is required the action of public forces as are the police for the ordered evacuation of people, the maintenance of the order and the removal of obstacles and debris; the firefighters in charge of the fire control and the rescue of victims in difficult access

locations and of course the medical and paramedical personnel in charge of the attention and evacuation of the victims. However, depending on the magnitude of the disaster situation, it is possible that the response capabilities of these entities get overcome and that additional support may be required for the appropriate fulfillment of certain tasks. The concerning tasks related to the people guided evacuation, the obstacles removing and the fire control (up to certain magnitude) can be potentially made by robots' teams. In works as the presented in [23], there are robots with the capacity of removing debris, in works as the presented in [4], there is the possibility of employing robots for the evacuation of crowds of people in emergency situations. Also, there are multiple works aimed to the competition in the Robocup's rescue robotics branch [1], where there are control algorithms applied to the execution of coordinated tasks of obstacles removal, firefighting and victims evacuation.

1.2. Main Rescue Robotics Challenges

Considering that during the search and rescue tasks there may be a very high probability for the robots to get damaged o to lost communication with the team which can be harmful for the development of the missions. With this in mind, the work schemes that present multiple low cost robots are preferred over the ones with better individual capabilities but less numerous as presented in [14]. The work with multiple robots introduces a new challenge, that is the cooperative control of them, this challenge may present multiple typical problems as the ones presented next,

1.2.1. Formation Control

Within the most common problems of mobile cooperative robotics there is the formation control, this problem aims to place and maintain a group of robots in determined relative positions related to themselves (they can be homogeneous or heterogeneous) and in some cases, it also aims to make the robots to follow determined trajectory as in works as the presented in [33], [3] and [22] among many others. With the purpose of efficiently coordinating the movement of a robots' squad it is possible to use centralized or distributed algorithms. However, in the last years the use of the distributed ones has extended way more since this strategy results more robust against the possible damages in the agents or the communication problems. Among the multiple approaches to the formation control tasks, there are those based on graph theory, employed to define the communications and interactions between agents and in game theory, employed to define the consensus protocols and determine certain parameters of the formation. In the work presented in [14] there are summarized some common strategies for the accomplishment of this task.

- Leaders and followers approaches: In this approach there are two classes of agents, the leaders who are in charge of determine the route and the formation of the rest of the

agents and the followers who do not have direct information related to the route, but based on the information of the leaders they keep the formation and follow the desired trajectory as shown in works as the presented in [13] where there is introduced the concept of the Complex Laplacian in order to generate control laws over the agents that allow the formation control and the trajectory tracking guided by at least two leader agents.

- Virtual potentials approaches: In this approach there are introduced certain potential functions that influence the behavior of the agents altering their control signal in a way that the formation is preserved employing information of its own position, the information of their neighbors and information related to the shape of the potential function as in works as the presented in [9] where each agent determines its relative position to its neighbors and based on it evaluates certain potential functions that allow it to maintain within the desired formation without getting too close to its neighbors.

- Virtual Leaders of Virtual Structures approaches: In this approach all the agents of the formation determine its characteristics based on the movement based on a virtual leader or a virtual structure that may or not be a tangible. Those virtual elements allow all the other agents in the formation to move in a coordinated way related to them considering the information of its neighbors, their own position and the information related to the virtual structure. For example, in the work presented in [33] there is an algorithm that allows the obstacles avoidance setting them as artificial leaders, letting the other agents to move around each obstacle without losing the formation and without colliding against it. In a similar scheme, in the work presented in [25] there is a very similar approach with the difference that the virtual leader is not an obstacle, but the dummy representation of a virtual agent in the formation. This virtual agent together with the real leaders allow the adjust of the trajectories to avoid the obstacles during the navigation.

1.2.2. Tasks Allocation

The search and rescue situations include a wide variety of tasks to be done, those tasks are usually assigned a priori or on-line by a human team. However, the communication with the robots or the scarcity of the human resources lead to the necessity for this problem to be solved at least partially by an automatic algorithm. This point is especially important when the agents in the team are heterogeneous, since they may have different capability levels for the development of each task. For instance, the mobility of a quadcopter makes it ideal to develop scouting tasks while its payload is very reduced. On the other hand, a wheeled robot may have a lower mobility but may have a much larger payload. Having this kind of differences in the behaviors and the capabilities of each agent the task allocation itself is not a trivial task which can be solved employing analytical or computational methods.

In the work presented in [15] there is introduced an approach where there are three kinds of robots, the most numerous (and worst equipped) small robots, the medium size robots with an intermediate computational capacity in charge of the coordination of the smaller ones and the scarce complex robots that have even higher computational capacity that allows the employ of advance optimization techniques used to coordinate the medium and small size robots. Also in this approach, there is considered the local task allocation using local leaders, with certain sub teams that are created and modified during the advance of the mission considering only the robots that are within the communication range and that are properly responding to the group signals (otherwise it may indicate a misfunctioning robot). On the other hand, the work presented in [30] introduces the possibility of employing bioinspired techniques to do the tasks allocation. In particular, they introduce an ant colony based optimization algorithm for the assignation of tasks to heterogeneous robots groups considering the capacities of each robot to support the task development and the necessity of accomplishing that tasks in a determined task window.

1.2.3. Human Robot Interfaces (HRI) Design

Remembering that for multiple situations, the robots do not have full autonomy for the development of the search and rescue tasks, the human intervention is usually required (in a lower or higher scale depending on the situation). Then the human robot interaction processes have been widely studied in works as the presented in [36] where those interactions are studied in the particular context rescue robotics competition of the American Association of Artificial Intelligence (AAAI). There is also the work [7] where the human robot interactions are studied in a search and rescue simulation scenario during which multiple human operators control a robots team in a teleoperated way. Alternatively, there are more recent approaches as the presented in [27] where the robots team has a bigger autonomy, but the humans employing simple instructions can noticeably upgrade the performance of the team in multiple tasks as the exploration of wide areas. In this work, a mixed integer programming is combined with human instructions in order to optimize the tasks execution times considering predetermined time horizons.

1.3. Rescue Robotics Competitions

Given certain tragical events that have happened in the recent years as the Kobe Earthquake the World Trade Center terrorist attack in the USA and the Chernobyl's and Japan's nuclear disasters, the robotics community has found a big interest in the rescue robotics considering its clear applicability that those robots may have in this kind of situations (especially In situations as the last two where the humans may be strongly affected by the radiation). This interest has been reflected in the creation of some competitions in the area, those competitions aim to incentive, develop and integrate the research of multiple teams worldwide to the

disaster attention areas. Some of the most cited competences are the rescue robotics league within the Robocup competition, the European competitions ElROB and EURATHLON and the DARPA Virtual Robot Competition (VRC).

1.3.1. Robocup Rescue League

Inspired by the 6.9 degree on the scale of Richter earthquake that occurred in Japan the 17 January 1995 morning, in 1999 there was proposed a new league associate to the robots' football world cup Robocup intended to analyze multiple lessons learned from this and many other emergency situations from the perspective of the robotic with the purpose of generate an "socially significative" application as presented in [18]. This rescue league has 4 main protects that are the simulation project, the infrastructure and robotics project, the integration project and the operation project. The target of those projects is to generate a framework where the contributions presented in each one of the projects can be properly integrated. The two first competences that correspond to the algorithm and control strategies development and to the development of robots and or information systems that allow to increase significantly the performance of the robots in emergency situations in separated approaches, then the integration project aims to join those approaches and finally the operation project aims to the real-world operation of the early developments.

1.3.2. DARPA VRC

The Defense Advanced Research Program Agency DARPA presented a robotics competition during 2012 and 2015 that was intended to improve the applicability of the robots in emergency situations. This competition consisted in the design of algorithms that allow a version of their humanoid robot Atlas to be able of achieving certain tasks. Those tasks are very valuable in emergency situations and actually are mainly made by humans. This competition was intended to incentive the development of new research in this area in order to help that in the future the response to this kind of situation to be made in an agile and effective manner without risking the life of people unnecessarily. This competition consists in 10 tasks designed to test the robots capacity to realize tasks originally designed for humans and those are:

- Drive a vehicle through a determined track with obstacles.
- Walk through a determine way with debris, that can be made avoiding them or using them as support.
- Remove debris that block an entrance.
- Open a door and enter to a building.
- Climb an industrial stair and walk over an industrial platform.

- Take a tool designed for humans and use it to break a concrete panel.
- Find and close a valve close to a pipe that is filtering.
- Plug a fire hose and open the corresponding valve.

Multiple works as the presented in [21] and [20] present the experiences of teams that participated in this competition and show in an extensive manner the modeling and control schemes employed to the execution of the required tasks.

1.3.3. ELROB and EURATHLON

In the work presented in [31] there are presented the competitions ELROB and EURATHLON which are an European alternative to the search and rescue competitions and aim to civil and military applications. The land robotics European tests began in 2006 and have been done annually since then. Those tests were created as an option to present standards to the study and the development of the robotics. It is worth to mention that even if any person in the world can participate in them, they are mainly intended to the European Union since there are multiple stimulus for the European competitors. The characteristics of the competition presented in [11] explain that the event is divided in multiple scenarios which are:

- Movement – Convoying: this challenge consists in the movement of two or more vehicles through a route of approximately 3km to a camp in conditions that vary from an asphalted way to paths with plenty of earth, dust, vegetation and or water.
- Reconnoitering of Building Structure: this challenge consists in the exploration of closed ambiances under conditions of low or null visibility and that may be partially destroyed with an area of approximately 50 m x 25m. Within this area there are interest objects that must be labeled in the map build. Additionally, there are additional points for reporting other interesting parameters as the radiation level, for constantly sending the current robot location to the control station and for sending images related to the labels placed on the map.
- Transport – Mule: this challenge consists in the transport of as much as possible load through a route of about 250 meters between two camps. For the development of this tasks, the guide of a person. It is worth to mention that the environmental conditions are the same as in the first challenge.
- Search and Retrieval of Human Casualties in Outdoor Environments: this challenge consists in the collection of two simulated victims (mannequins) with vague information about its location at a distance of between 50 and 75 meters sequentially taking them to a determined point. The terrain conditions are the same than in the first competition.

Reconnaissance and Disposal of Bombs and Explosive Devices: This challenge consists in the search of potential explosive artifacts in an area of about 100 meters squared where there may be pipes, cars, rooms surveillance posts and many others. In this challenge in particular there may be traps for the robots as fake artifacts among others.

2. Exploration, Mapping and Navigation

2.1. System Modeling

2.1.1. Environment Modeling

The mission environment is considered as a connect non-convex subset of R^2 where the robots and victims can freely move as presented in Figure 2.1. It is worth to mention that convex spaces as the presented in Figure 2.2 are not common, since in search and rescue situations there may be obstacles, corridors, halls and many other types of objects that make the convexity hypothesis unachievable.

2.1.2. Robots Modeling

We consider the set A which consists in n punctual robots in R^2, we denote the position of the robot i in the time $t \geq 0$ as $x_i(t) \in R^2$ the model employed for the each one, correspond to single integrator dynamics as presented in works like [37] and [5], but considering an integration constant k_i in order to capture the variation in the speed of each robot as presented in Equation (2.1),

$$\dot{x_i}(t) = k_i u_i(t) \tag{2.1}$$

where $u_i(t)$ is the control signal associated to the robot i.
Note that if there are flying robots among the team the altitude control loop will be neglected for the team considerations, however each flying agent must have its own inner altitude controller.

Dynamics Simplification

It is worth to mention that single integrator dynamics is a very simplified model specially considering that the work is intended to be employed with heterogeneous robots, however with the appropriate control it is possible to use reduced order dynamics for each robot. For instance with techniques like the feedback linearization presented in [17], any holonomic robot, modeled by its kinetic model of the general form presented in Equation (2.2), can be seen as a double integrator with input $w(t)$ setting the generalized torques vector as presented in Equation (2.3),

$$\tau = D(q)\ddot{q}(t) + H(q, \dot{q})\dot{q} + C(q) \tag{2.2}$$

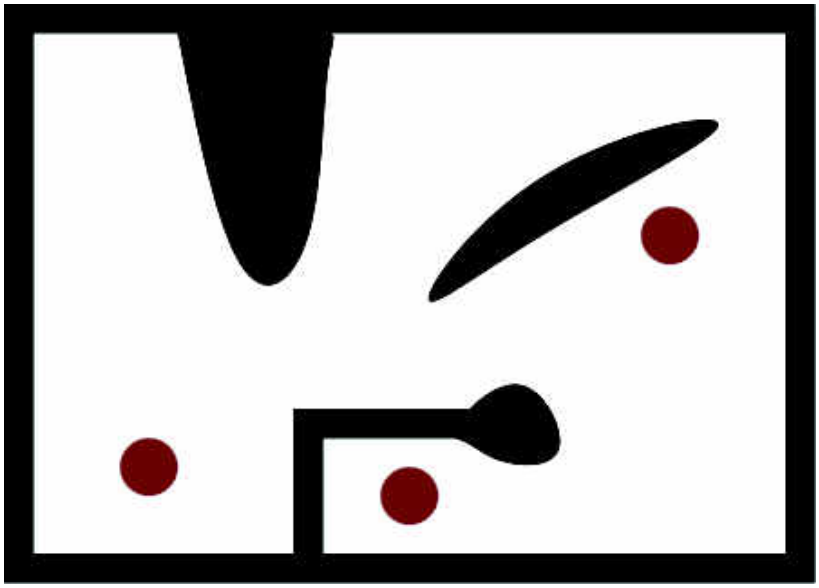


Figure 2.1.: Non Convex Space where the robots (presented as blue dots) movement is limited by obstacles, walls, halls and others. In those spaces any two points in the space can be connected through a continuous line (not necessarily straight) with all its points contained in the space

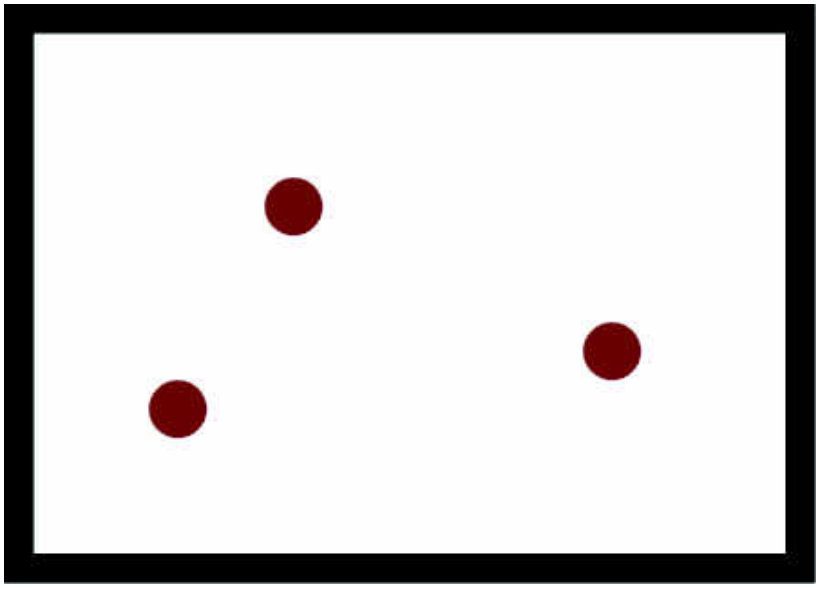


Figure 2.2.: Convex Space where the robots (presented as blue dots) can freely move. In those spaces any straight line that connects two points into the space are fully contained in the space

$$\tau = D(q)w(t) - H(q,\dot{q})\dot{q} - C(q) \tag{2.3}$$

where τ is the generalized torques vector, $q(t)$ is the desired position of each actuator, $D(q)$ is the inertias matrix of the robot, $H(q,\dot{q})$ is the centripetal and Coriollis accelerations matrix, and $C(q)$ corresponds to the gravitational forces exerted on the robot.
Once that we have double integrator dynamics as presented in Equation (2.4), we can employ different control strategies in order to impose the desired behavior.

$$\ddot{q} = w(t) \tag{2.4}$$

In particular it is possible to use sliding modes control as presented in [34] to impose first order dynamics for the system previously linearized as presented in Equation (2.5),

$$w(t) = -\tau_0 sign(\sigma_i(t)) \tag{2.5}$$

$$\sigma_i(t) = c_i q_{ei} + \dot{q_{ei}} \tag{2.6}$$

where $\tau_0 > 0$ is a control constant associated to each actuator, $\sigma_i(t)$ corresponds to the sliding surface of the controller presented in Equation (2.6), c_i is a positive constant that determines the system's time constant once the controller has arrived to the sliding surface and q_{ei} corresponds to the error signal of each actuator.
Note that it is possible to use the sliding modes control for the original system, however the feedback linearization reduces the requirements over the discontinuous term of the control signal reducing the chattering problem.
Finally if the time constant of the imposed dynamics is small enough compared to the movement speed of the robots for the purpose of the routes and movement planning the agent dynamic can be treated as a simple integrator dynamic with a speed related constant for each degree of freedom.
Another possibility to achieve single integrator dynamics is the use of kinematic models assuming that the actuators speed control loop is reliable, then the agent model will take the linearized form presented in Equation (2.7),

$$\dot{x}(t) = J(x)w(t) \tag{2.7}$$

where $J(x)$ is the Jacobian Matrix of the agent. Within that model it is possible to employ the inverse Jacobian Matrix to transform the system in a simple integrator dynamic employing input signal presented in Equation (2.8),

$$w(t) = J(x)^{-1}u(t) \tag{2.8}$$

where $J(x)^{-1}$ is the inverse (or pseudo-inverse for non square matrices) of the Jacobian matrix.

Non Holonomic Robots Considerations

Ideally all the robots involved in the mission would be holonomic and that way there would not be movement constraints, however with some considerations those kind of robots can be employed in the missions. The non-holonomic robots in general have certain turn radius required in order to being capable of changing its pose to a desired state as in the case of the standard car's parking process. With this in mind if the collision avoidance loop guarantees that the robot will always have the space for turning around, then it won't get stuck and with a determined turning delay it will be capable of achieving any position similar to an holonomic one. It is important to highlight that the non-holonomic robots can be very simple or very complex so the analysis for each type of robot must be made separately in order to introduce it to the mission. Particularly, in the kinematic model with inverse Jacobian matrix scheme the non-holonomic robots will have singularity issues. In order to deal with them, there are at least two possibilities for many cases:

- introducing slight perturbations to the agents angles allowing the system to return to its determined state, and
- dividing the task in an turn and then an forward movement, remembering that we have the proper turning space as presented in Figure 2.3.

For instance in the differential robot case, where if the desired movement goes normal to the robot orientation the model presents singularity. Then, if the angle is slightly changed, the Matrix singularity disappears or the movement can be decomposed in a rotation on its own axis and a later translation.

2.2. Real Time Challenges

From this point and on the robot model considered will be the simple integrator dynamics with speed constant associated and the modeling, linearization and inner loop control errors and fast transitory behaviors will be considered to not affect the outer loops behaviors. With this strong consideration gotten, now it is necessary to determine the desired high level behaviors for the agents in order to fulfill the mission requirements. Those main behaviors are presented as follows.

Maintenance of the Network Connectivity for Each Time Instant.

In order to accomplish a coordinated cooperative behavior, we require a full flow of information through all of the robots involved in the mission, we require that in each instant of time, all the robots could send a message to any other robot in the team (directly or through other robots). We consider radial limited wireless communication, then each pair of robots that are within a radius r_c can communicate. Also, we consider the undirected communication

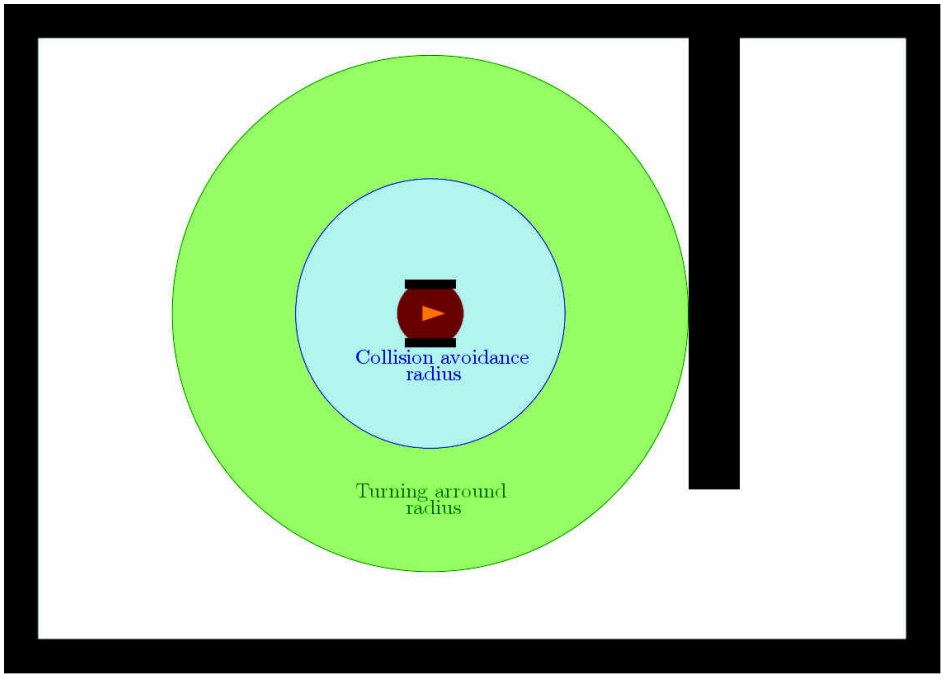


Figure 2.3.: Security radius for collisions avoidance (blue area) and turning radius (presented in green) for non-holonomic robots considered for the correct movement of the robots through the mission

graph G as the matrix that determines which robots are actually communicating (Note that if two robots are within the communication range they may or not be connected). Then in order to accomplish the communication requirements we need the communication graph G to be connected and we require the distance between robots connected to be lower than the communication range r_C.
At the beginning of the mission the robots will have an initial configuration that fulfills those requirements. Then, the problem of having a connected network in each instant of time, is reduced to maintain the network connectivity gotten from the beginning. In order to do that, it is necessary that all the links in the graph actual are preserved, this is achieved imposing that the robots that are currently linked maintain a distance between then lower that the communication radius r_c for each instant of time. Also, considering that in the present work, we consider the possibility of changing the network topology, that is to say changing the communication graph G. It is necessary to guarantee that every new communication configuration corresponds to a connected graph that respects the communication distance constrains.

Avoid Collisions Against Obstacles, Other Agents and Victims.

Since the robots are moving in an environment with plenty of obstacles and other agents, it is necessary that they consider policies that guarantee the collisions avoidance respecting

the distance to those objects, considering that since the robots are mobiles, they have an inertia and may need certain security radius to break and/or turn away. Also, it is important to consider that in most of cases the robot's sensors have a minimum distance range to be operative (for example the Lidar, infrared, and ultrasonic sensors cannot register values too close to them) so the robots must maintain a security distance r_e to any other object in the scene in order to avoid the collisions and keep their sensors in the operative range. It is important to highlight that this requirement must be evaluated dynamically since the robots have dynamic behavior, then the collision avoidance law must be capable of guarantee the robot against robot collision as well as the static obstacles avoidance on line.

Explore and Map the Disaster Environment.

Usually, the environment M is unknown at the beginning of the mission since there may not be available maps of the zone, or the actual maps are not reliable because they are outdated or because of the change in the environment caused by the emergency. Also in search and rescue situations there will be certain number of people that was unable of evacuate the environment on time for multiple reason, and they must be located, helped (generally by humans paramedical teams) and if possible evacuated. Then the first task that the robots team must accomplish is to generate an estimate of the environments' map $\bar{M}$ as similar as possible to the real one.
Since the robots have a limited sensing range r_s in every instant of time (before that the whole map is known), there will be a frontier δS between the space known S and the unknown. Then, the exploration task corresponds to the optimized tracking of this frontier through time updating the $\bar{M}$ information and detecting and updating the position of the victims in the scene in order to determine the interest points, the navigation routes and the tasks required to be done.

Identify and Classify Possible Victims

During the exploration process, the robots can find victims and report their positions to the team in order to determine the tasks related. However some of them may not have the appropriate sensors to determine if certain target is or not a real victim. For instance, robots equipped with only distance sensors are not be capable to distinguish between a mannequin and a real person. Then, once a potential victim is found any capable robot must be allocated to the task of classify it as a real victim capable to move, a real victim unable to move or a fake positive (as human shaped objects or unfortunately casualties). It is worth to mention that the robot that has identified certain possible victim may be allocated to classify it and generally the distance required to do an appropriate classification will be smaller than the required to identify a potential victim, then the classification task will demand an additional approach to the target.

Guide Capable to Move Victims to the Safe Zone.

When a victim capable to move is found, him/her must be evacuated to a safe area, for this purpose it is required that the robot approach to the victim, communicate an instruction "follow me"(for instance with a voice recording), and advance to a predetermined safe zone. During this process, the robot must maintain the distance to the victim in a range between the security range r_e and a determined distance r_{cv} that allows the victim to know the position of the robot. Additionally, the robots must maintain the victim within its line of sight since turning certain corners without considering this, can make the victim to get lost and could be required to be found again wasting valuable time. Also it is important to achieve this task through a route as short as possible since the mission environment may be dangerous and the shorter the route, the better for the safety of the people, also, taking a shorter route means a lower mission time allowing to attend a higher number of people in a determined time upgrading the right attention time probability.

Deliver Medical Supplies to Victims.

Not all the victims found may be capable to move, however depending on the situation there may be some supplies that could help the victim to be in a better state while the human help arrives (at the moment of this publication there aren't paramedical robots). For instance first aid kits may allow people to attend light injuries upgrading their survival chances, radio communicators may allow victims to report important information to the search and rescue team regarding to the person himself/herself or of another victims, and gas masks can save the life of people in situations with smoke or other dangerous gases. In this situations the robots must approach to the target victim to a determined delivery distance r_d and then drop or handle the supply to the victim (respecting the security radius as always).

Note that the first two behaviors presented must be accomplished in every instant of time, however the following ones correspond to the task assigned to the robot by the task allocation algorithm (this algorithm will be detailedly explain in the next chapter) and not necessarily all the robots will be capable of executing them (at least one of the robots of the team must can).
All of this behaviors have to be imposed on each agent through the input signal $u(t)$ that as we presented in the early section will be taken as the output's derivative. In order to do that we have considered a control signal with two main components as presented in Equation (2.9),

$$u_i(t) = u_{ti}(t) + u_{poti}(t) \tag{2.9}$$

where $u_i(t)$ is the control signal for the i robot, $u_{ti}(t)$ corresponds to a proportional controller that tracks the mission point of interest (Voronoi centroid, victim location or evacuation

point) for the robot i and $u_{poti}(i)$ is the net artificial potential for the robot i (this concept will be explained later in this chapter).
The tracking component of the control signal can take any convenient form that allows the robots to approach to the interest point required for the task. For simplicity, it was considered a proportional controller as presented in Equation (2.10),

$$u_{ti} = K_{ti}(x_d - x_i) \tag{2.10}$$

where K_{ti} is the proportional constant of the tracking control signal of the i-th robot, x_di is the desired position given by the task execution for the i-th robot and x_i is the current position of the i-th robot.

2.3. Exploration of the Disaster Environment

In a search and rescue situation, the first task required is the exploration of the disaster zone in search of task to be done, since there may not be available maps of the zone, and even if there are the disasters effects may change the real structure of the zone. This task consist in the generation of an estimation $\bar{M}$ of the that map M based on the information obtained by the robots through the time advancing in direction of the frontiers δS of the space known S in order to make that S tends to M as the time advances as presented in Figure 2.4. This process should be made as efficiently as possible since its proper implementation may improve the survival opportunities for people that may be in imminent danger or mistress.
In the real applications each robot may not know its true position in the map, however the SLAM (simultaneous Localization and Mapping) problem has been widely studied in works such as the presented in [8], then we assume that the robots have enough certainty about their position (thanks to an appropriate SLAM algorithm), We assume the robots positions is know in each time instant and we assume that they can map the scene as they advance through it.
In order to perform this exploration task automatically it is necessary to determine the appropriate task control signal $u_{ti}(t)$ for each of the robots in a way that the map is explored in an efficient way. We have adapted the algorithm DisCoverage presented in [16].

2.3.1. Heterogeneous DisCoverage

The original algorithm Discoverage is a frontier based exploration algorithm. Initially, this algorithm employs a Voronoi tessellation of the scene where each robot got assigned the points closest to it (considering that all the robots are homogeneous). The set of those closest points is called the Voronoi cell V_i of the robot i. Then each robot determines the exploration frontier δS into its own Voronoi cell based on its proper information and the information of its neighbors. And finally, With this the algorithm evaluates a weight function

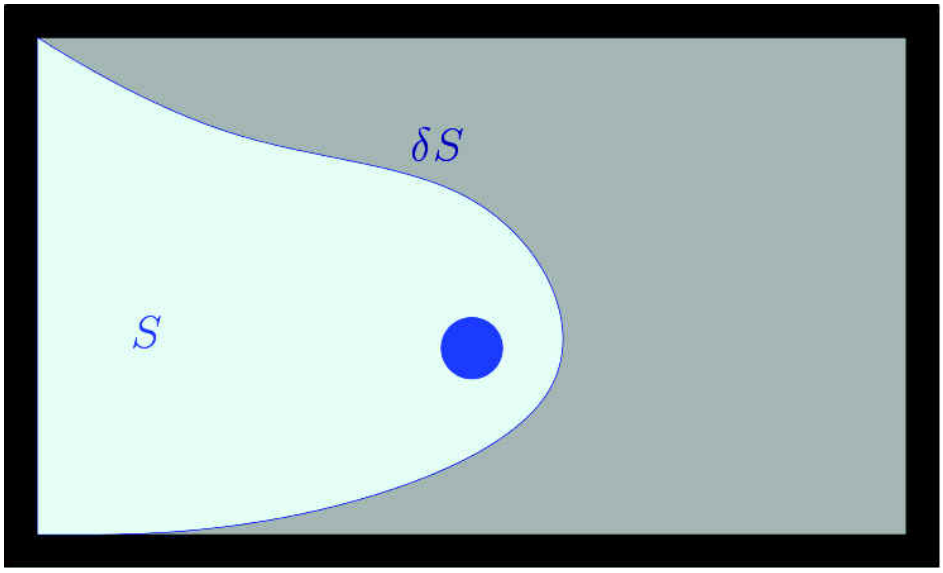


Figure 2.4.: Exploration Task definition - the whole space is M, the light cyan space corresponds to S that is the space known, the gray space corresponds to the unknown, the navy blue line correspond to the frontier of the space know δS and the blue dot represents the robot r that is tracking the frontier in order to build the estimate $\bar{M}$ of the scene

that grows near to the frontiers and use it to determines the weighted centroid of each cell x_{ci} which is tracked by each robot fulfilling the exploration task evaluated by a monotonic decreasing objective function H_{exp}.

Heterogeneous Voronoi Tessellation

Given that the robots involved in the mission have different movement speeds we should not assign the closest points into the space to each robot. Instead, we assign to each robot the points in the space that can be reached by it in the minimum time compared to its neighbors, that time is computed considering the average speed of the robots $\bar{v}_i$ as is presented in Algorithm 1.

An important change made to the original algorithm is the way the robots i that do not have frontier dS within its own Voronoi cell V_i behave. In the previous work the Voronoi centroid of the cell without frontier became its geometric cell and the robot will track it. However, that leads the robots to stay in areas of low interest since there were already explored. Considering that, we changed this behavior for tracking the neighbor NCF who has the frontier the closest (in terms of nodes of the graph $N_{NF}(i)$). This way if the robot has no frontier within its cell it will go closer to the limits of the exploration frontier and will help with the exploration process speeding up the whole process as is presented in the Algorithm 2.

Algorithm 1 Heterogeneous Voronoi Partition

function VORONOI PARTITION($Robots, scene$) ▷ where $Robots$ - robots array, $scene$ - matrix
  $MinimumTime = infinity(size(scene))$
  $VoronoiAssign = zeros(size(scene))$
  for $i \in Robots$ **do**
    for $(x, y) \in scene$ **do**
      $T_{ip} = Distance(Position(i), (x, y))/\bar{v}_i$
      if $T_{ip} < MinimumTime(x, y)$ **then**
        $MinimumTime(x, y) = T_{ip}$
        $VoronoiAssign(x, y) = i$
      end if
    end for
  end for
end function

Algorithm 2 Exploration Behavior

function EXPLORATION($Robots, scene$) ▷ where $Robots$ - robots array, $scene$ - matrix
  for $i \in Robots$ **do**
    Heterogeneous Voronoi Partition(Robots,Scene)
    if $dS \in V_i \neq \{\phi\}$ **then**
      Track Voronoi Centroid
      $N_{VF}(i) = 0$
    else
      $NCF \leftarrow$ Find $minN_{VF(j)}$ for $j \in Robot'sNeighbors$
      Track NCF
      $N_{VF}(i) = N_{VF}(NCF) + 1$
    end if
  end for
end function

In the original Discoverage algorithm, they mention the possibility of using different types of distance metrics for the distance function and use the euclidean distance as example. This metric is useful in convex spaces but fails to capture the real value of the distance into non-convex spaces that are more common in disaster scenes considering that it may be halls, walls, obstacles, and other elements that breaks the convexity of the disaster scene. In order to consider correctly this class of spaces, we propose to use the flooding distance which is based in the Breath-first search presented in [32].

2.3.2. Basic Flooding Distance

In order to capture the non-Euclidean nature of the distance metrics in the space, we propose the use of the Algorithm 3 of distance calculus based on the breath first distance.

Algorithm 3 Flooding Distance

function FLOOD($p_0, scene$) ▷ where p_0 - array, $scene$ - matrix
  $Distance = infinity(size(scene))$
  for $(x, y) \in scene$ **do**
    if scene(x,y) == obstacle **then**
      $Distance(x, y) = -1$
    end if
  end for
  $Distance(p_0) = 0$
  $DistanceBuffer = zeros(size(scene))$
  $Counter = 10$
  while $DistanceBuffer \neq Distance$ and $Counter > 0$ **do**
    $DistanceBuffer = Distance$
    $Counter = Counter - 1$
    for $(x, y) \in scene$ **do** ▷ this loop is ran twice, from top-left to bottom-down and then backwards
      for $p_n = (i \pm 1, j), (i, j \pm 1)$ **do**
        if $(Distance(i, j) \neq -1$ and $Distance(p_n) < Distance(i, j) + 1$ **then**
          $Distance(p_n) = Distance(i, j) + 1$
        end if
      end for
    end for
  end while
end function

Algorithm 3 generates an estimation of a Manhattan distance (calculated using only horizontal and vertical steps) into the non-convex space and its speed is boosted given the way

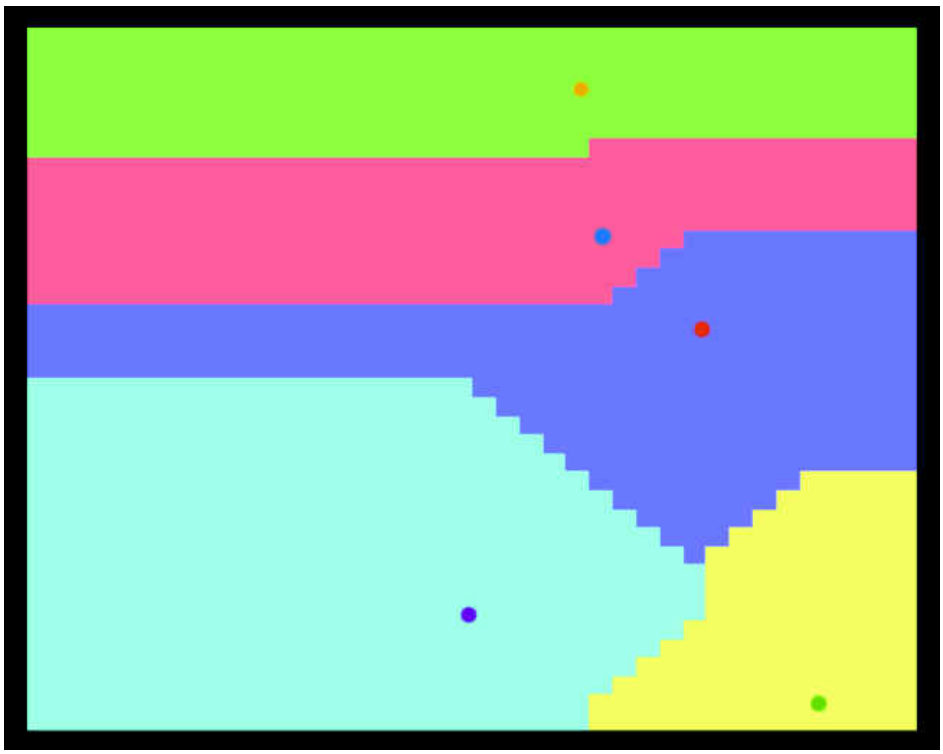


Figure 2.5.: Homogeneous Voronoi Tessellation with Flooding Distance

its calculated since the evaluation in causal and anti-casual ways speeds up the convergence compared to the single way case.
In Figure 2.5 we present an homogeneous Voronoi tessellation of an convex space where the distance behaves as a standard Voronoi tessellation with Manhattan distance, and in Figure 2.6 we present an heterogeneous Voronoi tessellation of the same space given the same seeds but with different $\bar{V}_{0i}$. In these figures we can see how in the heterogeneous case the fastest robot (the red dot) makes bigger its Voronoi cell while the slowest (the yellow dot) has its cell notability reduced.

Modifications to the Exploration Algorithm

One of the biggest changes in the algorithm is how the breadth first search is implemented, initially there was used a 4 neighbors flooding, which means that the algorithm was comparing an initial cell's value with the cells above, under, right side and left side of the initial cell, considering only movements in strictly horizontal and vertical directions. Now the algorithm is using a 8 neighbors flooding (8NF), allowing to not just going through the directions mentioned before but also through the ones located immediately on the diagonals (NE, NW, SW and SE). The improvement of this change is that the distance is way more accurate and tends to be closer to the Euclidean distance. In the algorithm implemented for flood distance, we changed the term that updates the value of the cell by comparing it with

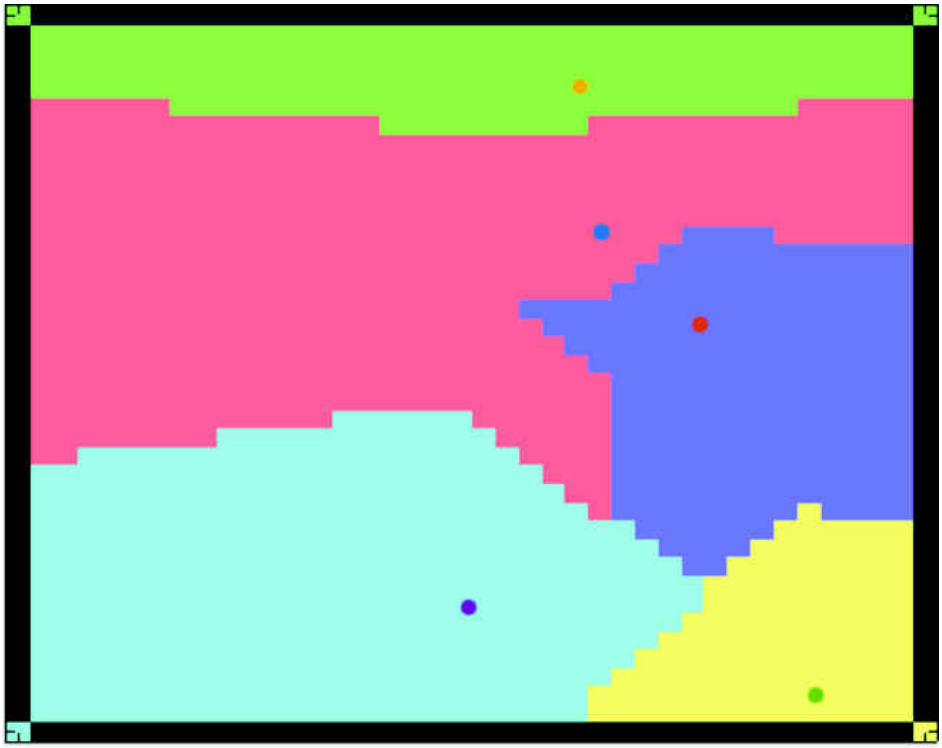


Figure 2.6.: Heterogeneous Voronoi Tessellation with Flooding Distance

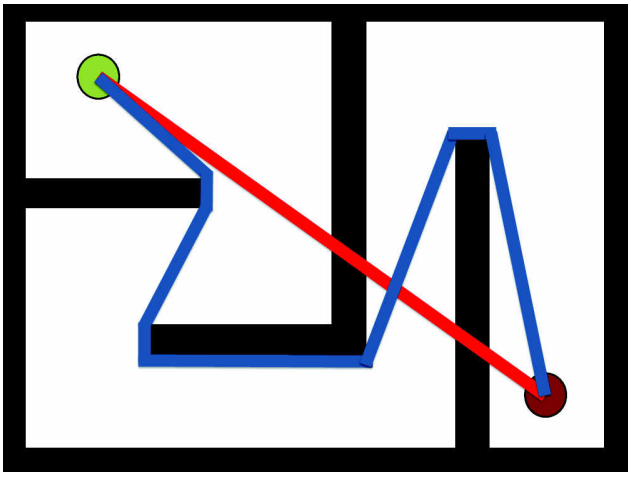


Figure 2.7.: Non euclidean space with distance metrics - Green dot: Actual position Red dot: Target position. Red line: Euclidean distance. Blue line: Strictly optimized distance in a non-convex environment.

the current value plus 1 if it is in the 4 neighbors directions or plus $\sqrt{2}$ if is in the diagonal directions (the corresponding euclidean distance for step the movement). In the next image we can realized the effect of this change (See Figure2.8) where the euclidean distance, the Manhattan distance and the presented 8NF are compared. It is worth to mention that an error can be produced when the robot is calculating in different angles than the ones represented in $n45°$, because the main calculus are performed in those angles. The maximum error presented in the calculus is when the robot sense in the angles represented in $n45° + 22{,}5°$

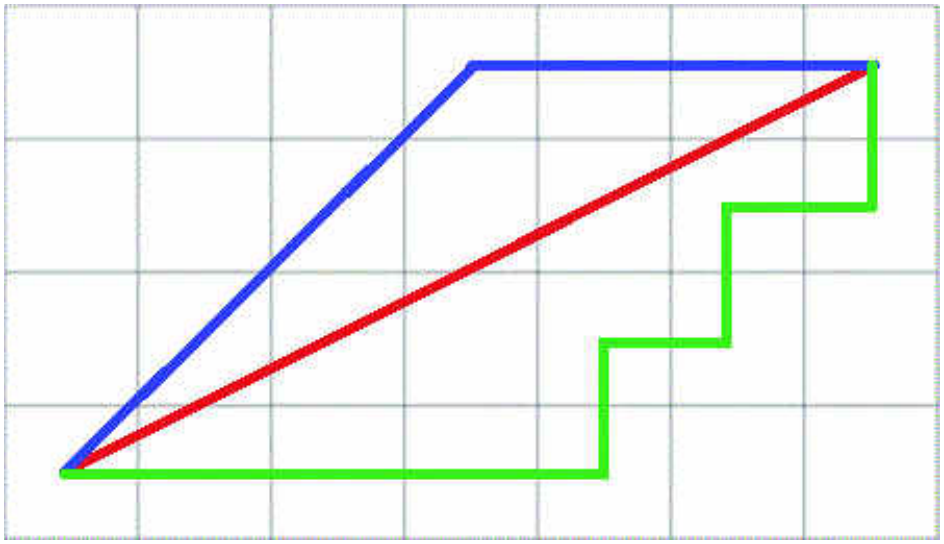
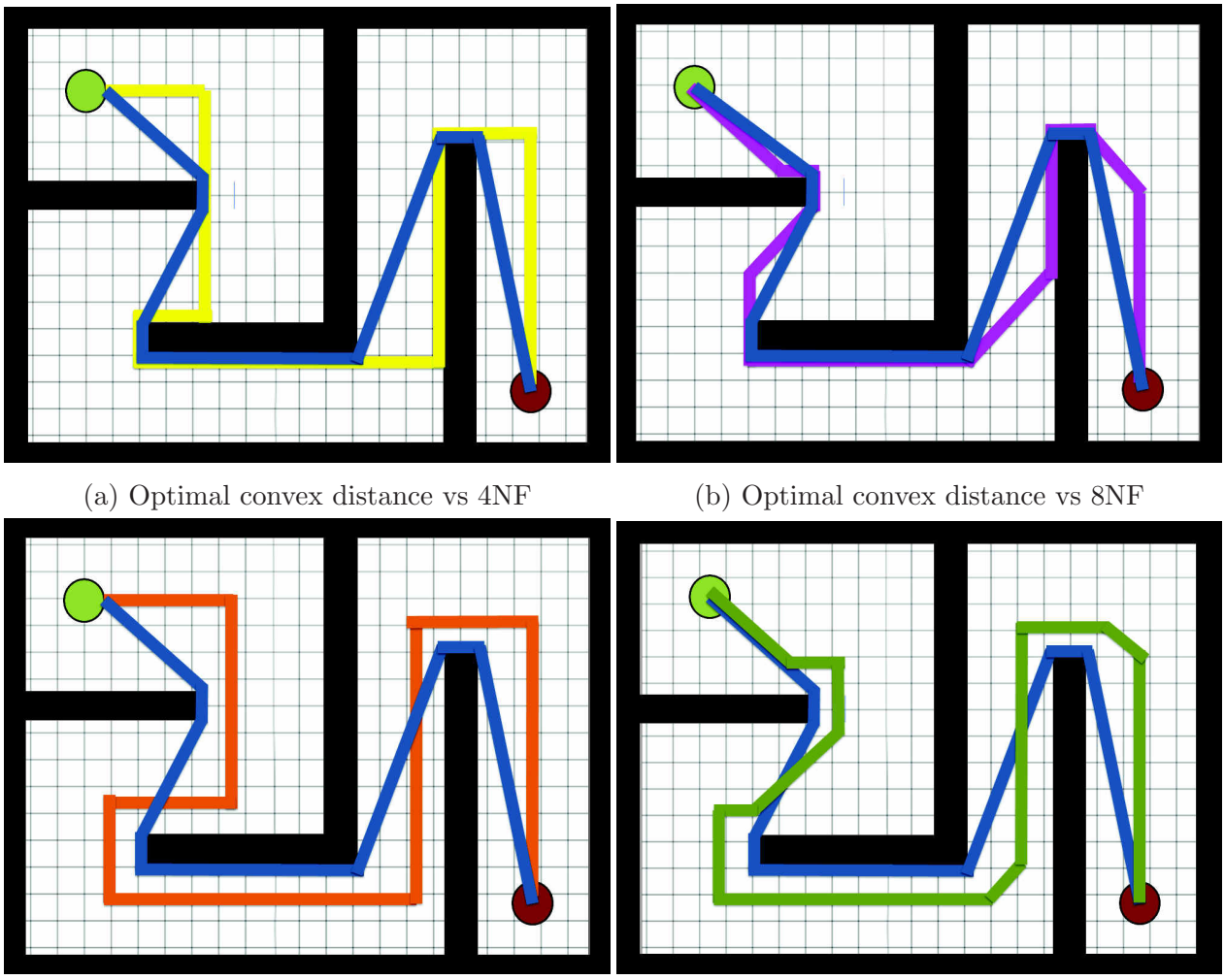


Figure 2.8.: Green line: Manhattan distance. Red line: euclidean distance. Blue line: equivalent to 8 breadth search.

In Figure2.9 it is shown the comparison between the strictly optimal trajectory and the trajectories made by the theoretical 4NF and 8NF, (2.9a and 2.9b). Then we have the strictly optimal trajectory against the real trajectories of the 4NF and 8NF (2.9c and 2.9d). The real trajectories are those where the robots also apply the net virtual potential functions to avoid collisions. We can appreciate how the robot manage to trace a trajectory that tends to the strictly optimal, when the robot applies the 8NF and can still avoid the walls by the potential functions respecting the security ranges (2.9d).

2.4. Artificial Potentials

To solve the problems of connectivity maintenance and collision avoidance we use a virtual potential functions approach. This approach must be complement the control signal corresponding to the assigned task and its components are studied in this section.



(a) Optimal convex distance vs 4NF

(b) Optimal convex distance vs 8NF

(c) Optimal convex distance vs Real 4NF

(d) Optimal convex distance vs Real 8NF

Figure 2.9.: Comparison between convex optimal distance with 4NF, 8NF, real 4NF and real 8NF

2.4.1. Connectivity Constraints

Following the ideas presented in [37] we apply a function of artificial potential in order to guarantee the maintenance of the links that are already created in the network. This function tends to infinity as the distance between robots tends to the maximum communication range in order to keep the robots connected and also when the distance between the robots becomes to small in order to avoid collisions. The artificial potential function has the next expression:

$$\psi_{cij} = \frac{1}{\rho_2^2 - ||x_{ij}||^2} - \frac{1}{||x_{ij}||^2 - \rho_1^2} \tag{2.11}$$

where psi_{cij} is the connectivity potential function between the robots i and j, ρ_2 is the communication range of the robots, x_{ij} is the distance within the robots i and j, and ρ_1

is the collisions avoidance radius of the robots. The artificial potential functions evaluated are considered later together with the mission component signals, in order to determine the actual dynamic to be imposed on each robot.

2.4.2. Network Reconfiguration

As the mission advances the initial configuration of the robots may not be appropriate since existing links could be limiting the maximum reach of the network. Based in the network connectivity analysis presented in [37] we introduce the reconfiguration Algorithm 4 that iteratively updates the network configuration based on the proximity of the robots. Each robot analyzes which of its connections can be replaced for another maintaining the full network connectivity evaluated thorough the second eigenvalue of the Laplacian matrix.
The network reconfiguration algorithm is initialized principally when the distance $||x_{ij}||$ reaches a value of $\alpha\rho_2$ for a given $\alpha \in (0, 1)$ and can be initialized periodically in order to extend the network range without employ control effort in holding the communication. Note that the algorithm can be ran using only the information of each robot and its neighbors, however the information of the deletions must be broadcasted in order to maintain updated the Laplacian of the network for each robot.

2.4.3. Obstacles Evasion

In order to guarantee the evasion of static obstacles we use a function similar to the used for the collision avoiding between robots in the previous section. The form of this function of artificial potential is:

$$\psi_o = \frac{1}{||x_{io}||^2 - \rho_1^2} \tag{2.12}$$

where $||x_{io}||$ is the distance to the nearest obstacle point detected. Note that since this is the nearest point of the obstacle this implementation guarantees that the robot will avoid all the static obstacles since it tends to infinity as the distance tends to the security range ρ_1. Is is worth to remember that the security range can be extended for non-holonomic robots in order to guarantee their minimum range for turning around in every moment of the mission.

2.4.4. Equilibrium Points Escape

The original DisCoverage algorithm presented the possibility of unstable equilibrium points in the exploration function and proposed the use of arbitrary perturbations in order to escape of these. However when the artificial potential functions are implemented the perturbation can not be completely arbitrary over the control signal since it may break the network connectivity. In order to solve these problems the arbitrary perturbation is not made directly to the control signal, instead it is made to the exploration component of this signal; this way

Algorithm 4 Network Reconfiguration

function NETWORK RECONFIGURATION($Robots, Laplacian$) ▷ where $Robots$ - robots array, $Laplacian$ - Network Laplacian matrix
  for $r \in Robots$ **do** ▷ virtually adds all the possible connections
    for $r_n \in$ Robots neighbors of r **do**
      if $distance(r, r_n) < r_C$ **then**
        $Laplacian_V \leftarrow VirtuallyConnect(r, r_n)$
      end if
    end for
  end for
  $Laplacian_B = Laplacian$
  while $Laplacian_B \neq Laplacian_V$ **do**
    $Laplacian_B = Laplacian_V$
    for $r \in Robots$ **do** ▷ virtually deletes all the non-necessary connections
      $D_{max} = 0$
      for $r_n \in$ Robots connected with r **do**
        if $distance(r, r_n) > D_{max}$ **then**
          $DeletionCandidate \leftarrow (r, r_n)$
          $D_{max} \leftarrow distance(r, r_n)$
        end if
      end for
    end for
    $Laplacian_V \leftarrow$ Virtually Delete ($DeletionCandidate$)
    if $\lambda_2(Laplacian_V) > 0$ **then**
      Virtually Delete(Deletion Candidate)
    end if
  end while
  $Laplacian \leftarrow Laplacian_V$
  update network(Laplacian)
end function

the connectivity is not lost given that the artificial potential function continue working and if it is necessary the artificial potential component opposes the effect of the perturbation that may break the link.
Also a new class of equilibrium points may appear in certain classes of situations. This class of this situations is when two or more robots try to explore in opposed directions and the network configuration does not allows them to continue its movement without lost the network connectivity; in this case an arbitrary perturbation will not be effective since these equilibrium points because they are stable equilibrium points. these non-desired equilibrium points are avoided in many situations thanks to the reconfiguration algorithm since it allows the network configuration to be changed in order to extend its maximum range, however the problem reappears when the exploration area is much bigger than the maximum network range (given by nC_r in the convex case) and in this solution is not yet covered by the algorithms proposed.

3. Tasks Definition, Allocation and Execution

Throughout the exploration process there are multiple emergent task that must be done by the rescue team. Some of them could only be made by humans like the first aid and paramedical tasks, however there are many kinds of tasks that can be realized by the robots like the ones presented in [26]. In the current work, we are going to consider mainly victims identification (using the appropriate sensors and or machine vision), the victims evacuation and the delivery of appropriate supplies (like first aid kids, air masks or radio communicators). It is worth to highlight that since the tasks are not known at mission's start, and are found as the mission advances, their allocation must be done on-line. This task allocation needs to be optimized based on determined criteria that in this case will be the time required for the team to complete all the known pending tasks considering that the robots may differ in their capabilities for accomplishing each type of task. For instance, a fast explorer robot with basic sensors may not be capable of doing an appropriate victim identification or a delivery robot may deplete its inventory.

3.1. Task Definition

Since the objective of the algorithms presented is to minimize the time for accomplish the whole mission, the duration of all the tasks that will be developed by the robots must be estimated and presented in a cost matrix in order to determine an optimized allocation. Note that not all the robots may be capable of achieving all the task so if the robot is not capable the associated time cost will be infinity or ft the robot is capable, the estimations are made as follows:

3.1.1. Exploration

The first task that each robot must do is the exploration of the scene as mentioned before, however the cost value in time of this task is not trivial and the estimation that we took is presented in Equation (3.1), in such a way that in the early stages of the mission the exploration takes a high priority and in the final stages of the mission it may be even

discarded if there are not more expected victims to be found

$$C_i^e = K_e \frac{A_m}{\bar{v_i}\tilde{N_v}} \tag{3.1}$$

where C_i^e is the cost of the exploration task for the robot i, K_e is an arbitrary constant that allows to give priority to the attention of the already found victims or to the finding of the expected missing victims and allows the conversion of the result of the expression to seconds, A_m is the area of the mission environment that has not been yet explored, $\bar{v_i}$ is the mean speed of the robot i and $\tilde{N_v}$ is the expected number of missing victims to be found. In Figure 3.1 we present a mission stage with a low K_e causing that the robots tend to attend the already found victims before of continuing the exploration process.

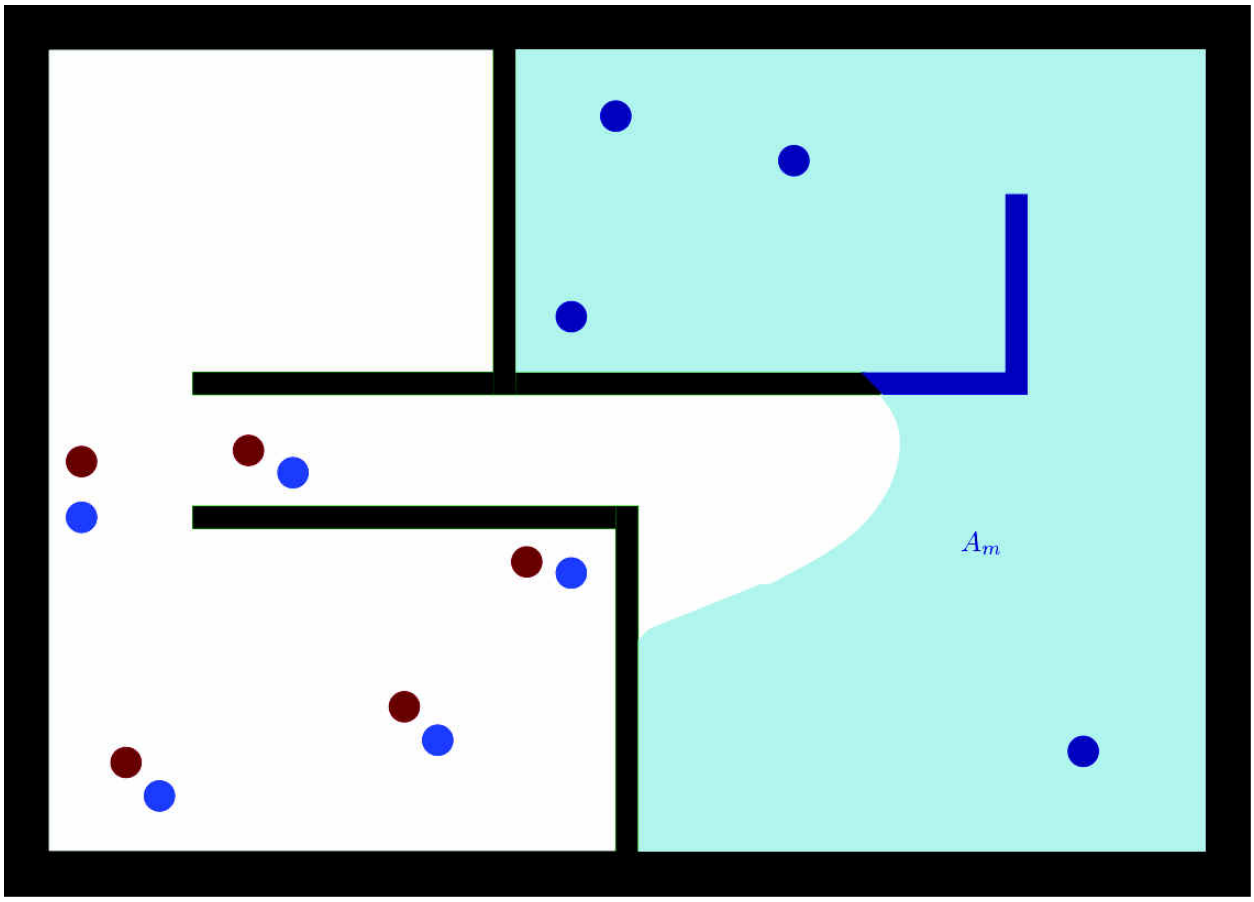


Figure 3.1.: Mission stage with low K_e where the robots (presented as red dots) decide to first attend all the already found victims (presented in blue) before exploring the rest of the missing exploration area A_m looking for the remaining victims (presented in navy blue)

On the other hand, in Figure 3.2 we present an stage where the K_e has a higher value in a way that most of the robots continue the exploration task while the others attend some found victims.

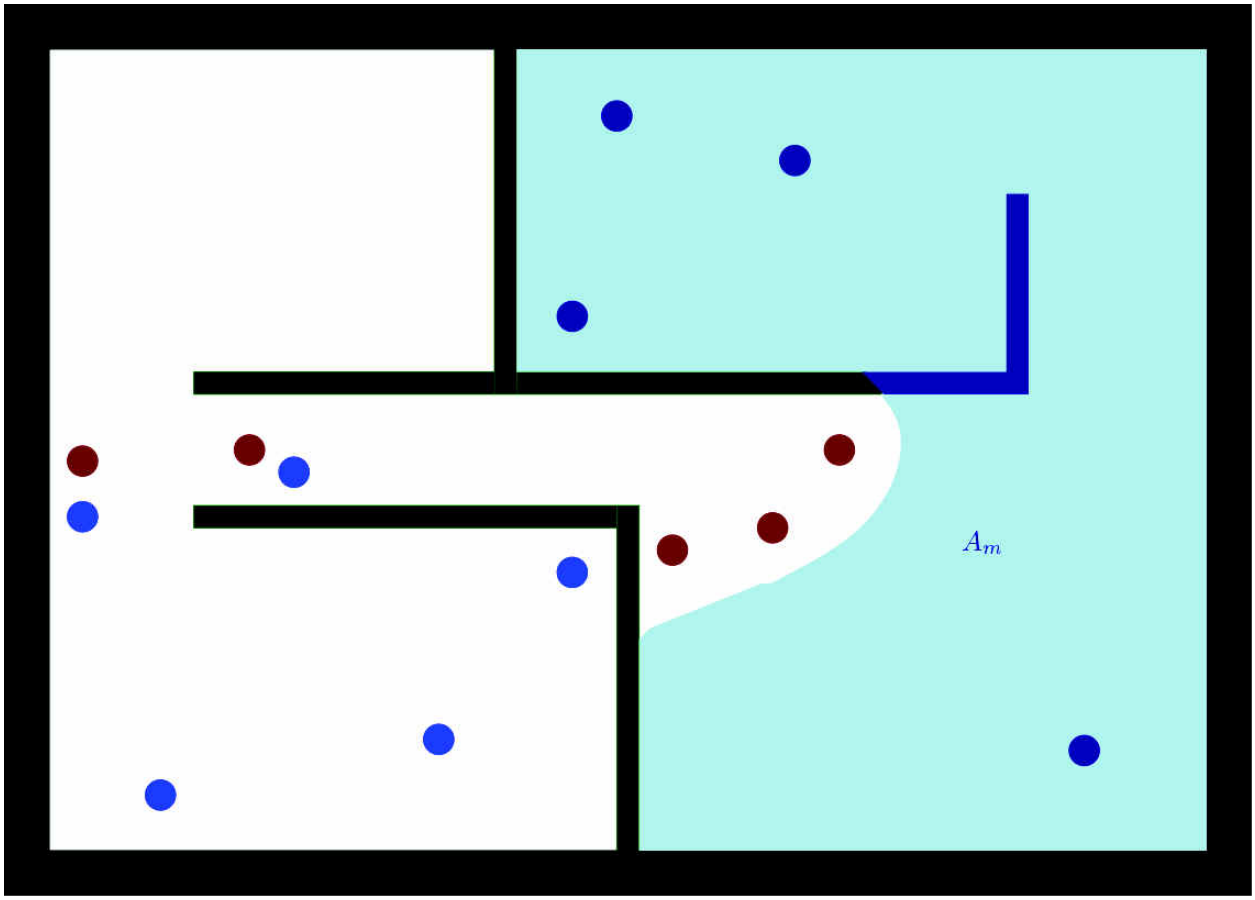


Figure 3.2.: Mission stage with higher K_e where the robots (presented as red dots) decide to first explore the missing exploration area A_m looking for the remaining victims (presented in navy blue) instead of attending the already found victims (presented in blue)

Note that the variable $\tilde{N}_v$ must be set by a upper bound of the number of people expected to be found in the disaster area, this number can be estimated considering the usual population density of the location or logistics information available before the emergency situation, for instance if a disaster occurs in a commercial center the information of its typical or maximal number of clients may be used to estimate this value. If the expected number of victims exceeds the actual number, then the robots will continue exploring until they complete the exploration the accessible map. However if the area remaining is small the priority of the task will be reduced by the form of the function.

3.1.2. Identification

Concerning to the victims we have considered four types of them.

- Potential victims that require identification.

- Victims that are capable of moving by themselves and can be evacuated.
- Victims that are alive but cannot move or require human assistance.
- Casualties or false victims that cannot be helped.

Once a victim is found the next task to be done is its classification and given that not necessarily all the robots can, there is the possibility that the robot that has detected the victim is not capable of classify the victim or it may require to be nearer to the victim in order to do the classification as presented in Figure 3.3. The cost for this task is presented in Equation (3.2), taking advantage of the distance matrix calculus with the 8 neighbors-flooding which allows the estimation of the minimum distance from each robot to the victim even if the path to it is not linear as follows

$$C_{i,j}^{ID} = T_{i,ID} + \frac{d_{i,j} - r_{i,ID}}{\bar{v}_i} \tag{3.2}$$

where $C_{i,j}^{ID}$ is the cost of the robot i for identifying the victim j, $T_{i,ID}$ is the time required for the robot i for the identification of a victim (since they may have different types of sensors this time may vary), $d_{i,v}$ is the distance between the robot i and the victim j evaluated through the distance matrix generated using the 8 neighbors-flooding, $r_{i,ID}$ is the minimum distance required for the robot i to identify a victim and $\bar{v}_i$ is the average speed of the robot i.

3.1.3. Supplies Deployment

if the victim is alive but cannot be evacuated the robots may set a transmitter for marking precisely its location for the human rescue teams or depending of the situation may deliver first aid kits, radio communicators or gas masks among others. This task requires that the robot reaches the location of the victim and executes the delivery (usually employing a clamp or a robotic arm), this is presented in Equation (3.3),

$$C_{i,j}^{Del} = T_{i,Del} + \frac{d_{i,j}}{\bar{v}_i} \tag{3.3}$$

where $C_{i,j}^{Del}$ is the cost of the robot i for delivering a supply to the victim j and $T_{i,Del}$ is the time required for the robot i to hand over the supply.
In Figure 3.4 we present the actual route that the robot may take to reach a victim destination considering the employ of the 8NF distance metric and the repulsion of the walls gotten from the artificial potentials,

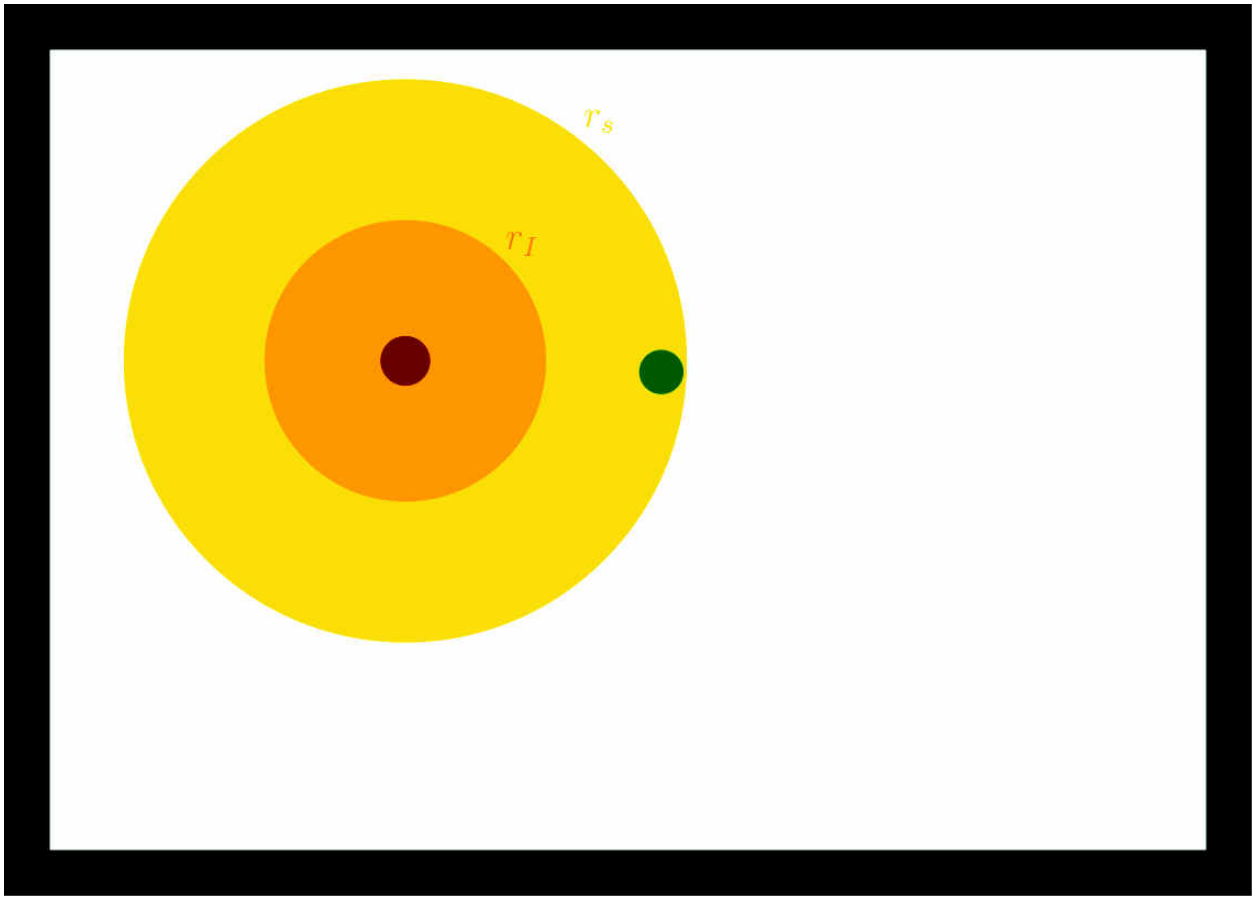


Figure 3.3.: Comparison between the detection radius $r_{i,s}$ and identification radius $r_{i,ID}$ of a robot (presented in red) related to a victim (presented in green)

3.1.4. Evacuation

If the victim is alive and can be evacuated the robot(s) may guide him/her to a safe area, so the robot must reach the location of the victim going at its own speed and then when the victim is in range they must go to the safe area going at the smallest speed between the speed of the victim and the speed of the robot as presented in Figure 3.5. This is summarized in Equation (3.4), considering that the speed of the victim depends on its state and the class of the situation

$$C_{i,j}^{Ev} = T_{i,Ev} + \frac{d_{i,j}}{\bar{v}_i} + \frac{d_{j,SA}}{min(\bar{v}_i, \bar{v}_i)} \tag{3.4}$$

where $C_{i,j}^{Ev}$ is the cost of the robot i for evacuate the victim j, $T_{r,Ev}$ is the time required for the robot i to communicate to a victim to follow it, $d_{j,SA}$ is the distance between the victim j and the safe area and $\bar{v}_i$ is the average speed of the victim j. For example the, victim could move faster if is in perfect health, has good visibility and the air is clean but may move

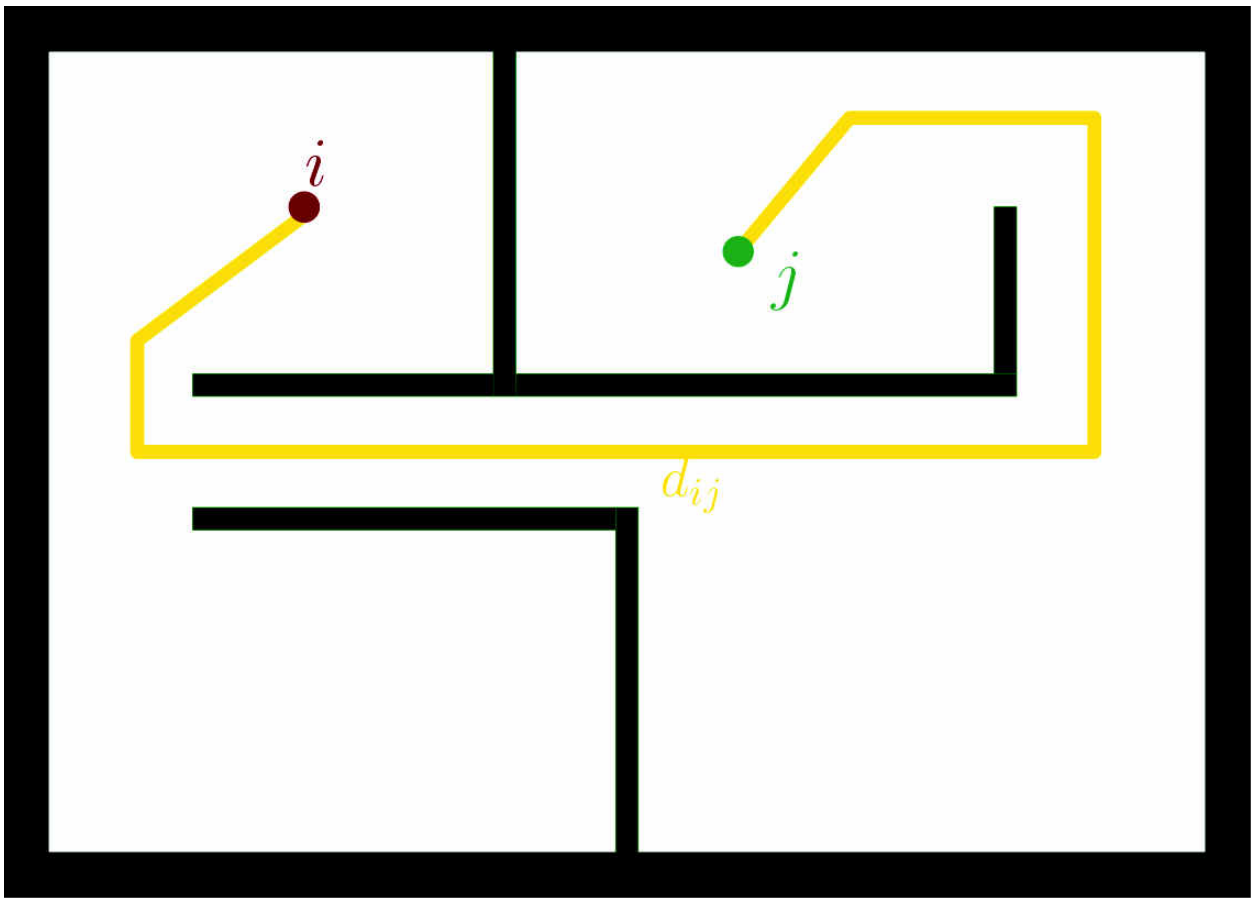


Figure 3.4.: Expected route for a robot i to reach the location of a non-mobile victim j for a delivery task

slower if is wounded, has poor visibility and/or there is smoke in the air.
The victim has to follow the robot and the robot must maintain certain distance to the victim, it is necessary that the robot maintain the victim within its line of sight and if in any moment the victim is lost or gets too far, then the robot must find it back taking as reference the last position known as presented in Algorithm 5.

3.2. Events and Task Allocation

At the start of the mission the task allocation is trivial, all the robots are set to explore the mission environment since the position of the victims is unknown and therefore there are not other possible task to be done. When a victim is found or a task is concluded, the tasks available to be done change, so the previous task allocation must be reevaluated and since generally there will be more tasks than robots it is necessary to evaluate which task allocation is optimal in the current setup. In order to solve this allocation problem we utilize the extension of the Kuhn-Munkres algorithm (commonly known as the Hungarian method)

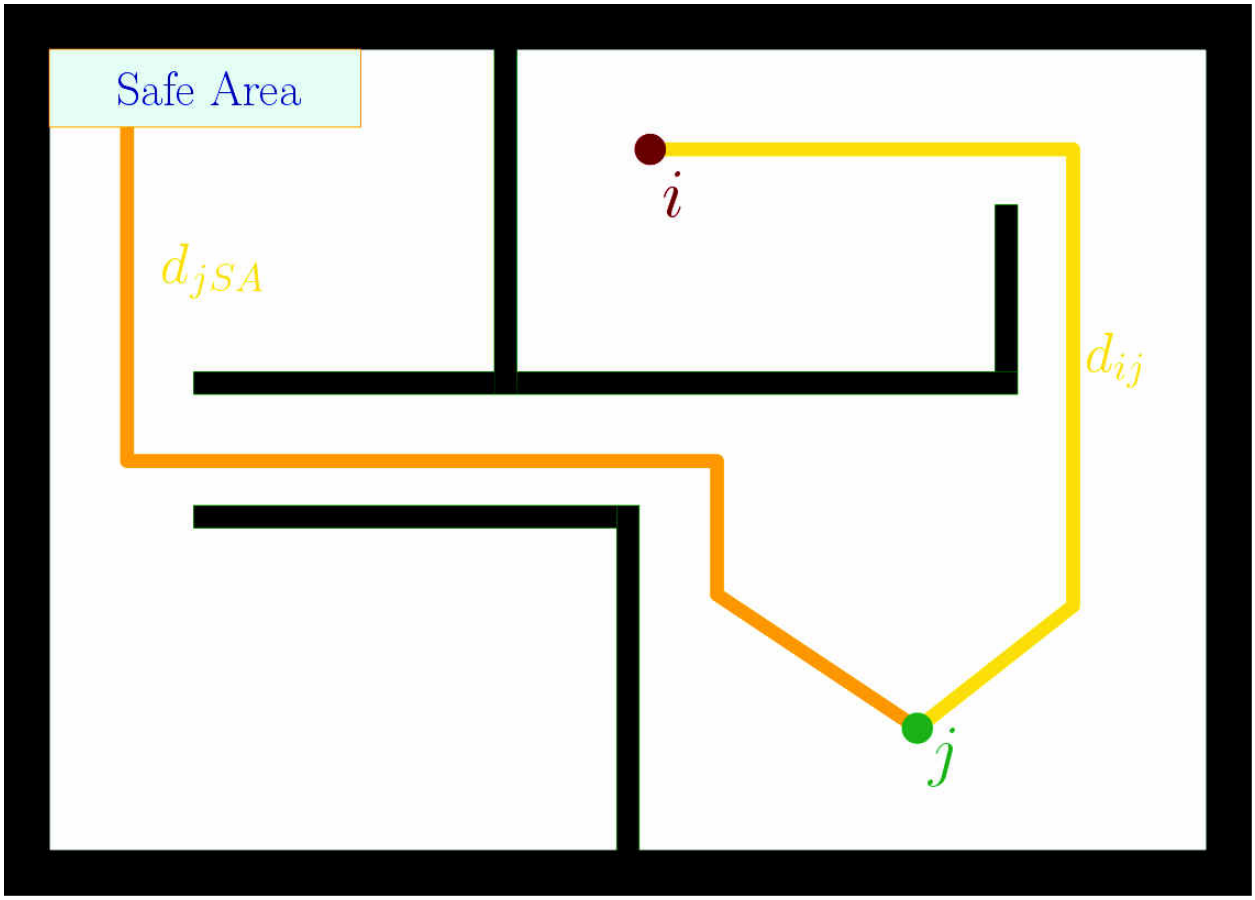


Figure 3.5.: Expected route for a robot i to reach the location of a mobile victim j and then to guide him/her to the safe area

for rectangular matrices presented in [6]. This method is a tool employed to optimally solve assignation problems with complexity $O(n^3)$ employing simple row and column operations to the cost matrix in order to evaluate which task has the minimum cost for the team. The implementation of it was realized employing the current cost matrix considering that all the robots can explore (if there is space pending to be explored) and that if there are victims that are not satisfactorily attended, the robots that can attend them will have the possibility. Note that if a robot is not capable of realizing determined tasks the correspondent cost will be infinity and the Hungarian method will not assign that task to that robot.

3.2.1. Event Triggered Reallocation

Since the mission conditions change in time we call the reallocation algorithm each time that an important event occurs. Those events considered as important are:

- The discovery of a potential victim.

Algorithm 5 Evacuate

```
1: function EVACUATE(robot, victim)          ▷ Where robot - agent, victim - agent
2:     if Distance(robot, victim) ≥ r_cv then
3:         p = position(victim)
4:         p_d = traceBack(p, LoS, Distance)
5:     else
6:         Instruct victim to follow
7:         p = p_s
8:         p_d = traceBack(p, LoS, Distance)
9:     end if
10:    track p_d
11:    if Distance(robot, p_s) ≤ r_s then
12:        Instruct victim to don't follow anymore
13:        return
14:    end if
15: end function
```

- The identification of the type of a victim.
- The successful deliver of a supply.
- The successful evacuation of a victim.
- The end of the exploration task (there is no more space to be explored or all the expected victims were found).

Since those are the uniques events considered that can change the tasks costs table and the Hungarian method guarantees the optimality of the assignation, we can be certain that in every moment of the mission, but the moments during the reallocation takes place the task allocation will be optimal.

3.2.2. The Reallocation Propagation

In order to respond to the changes in the mission environment the fastest as possible, the robot that detects the change is in charge of evaluating the task reallocation employing the Hungarian Method and send the message to all the other members of the team as presented in Algorithm 6. Note that the message will be always delivered to all the robots of the team since the communication is guaranteed by the artificial functions that preserve the existing links in the network and the network reconfiguration algorithm presented in the previous chapter.
It is important to mention that the events that trigger the reallocation, in general are expected to not be simultaneous, however if there is the case when two robots present different

Algorithm 6 Task Reallocation

```
1: function TASKS REALLOCATION(Robots, scene) ▷ Where Robots - robots array, scene - matrix
2:     Updates Tasks Cost Matrix
3:     T = Hungarian Method (Munkres)
4:     for i ∈ robots do
5:         T_i = T(i)
6:     end for
7: end function
```

task allocations based on two simultaneous events the communication protocols will give an pseudo arbitrary order to the messages and the network will accomplish the consensus based on the latest. Other possibility that could be considered is to run a simple exception algorithm which allows the restore of the consensus of the mission information and generate the correct optimal allocation. However in practical scenarios the final behavior is expected to be very similar and the first approach was considered in this work.

4. Simulations

The algorithms presented in the previous chapters are thought to be a complete framework for the robotic assistance in search and rescue situations. The present chapter is intended to show the behavior of each one of the individual algorithms by itself and finally the integration of them all in varied unknown simulation environments. All the simulations were developed in a Laptop with the following main characteristics:

- Operating System: Windows 10 Home Single Language 64-bit (10.0, Build 15063) (15063.rs2 release.170317-1834)
- System Manufacturer: ASUSTeK COMPUTER INC.
- System Model: N551ZU
- Processor: AMD FX-7600P Radeon R7, 12 Compute Cores 4C+8G (4 CPUs), 2.7GHz
- Memory: 16384MB RAM
- Video Card: AMD Radeon(TM) R7 Graphics 4GB

The integration method employed was the Euler's integration method for all the interest values as presented in Equation (4.1) considering a fixed Δt step for each simulation.

$$x(t + \Delta t) = x(t) + \frac{dx(t)}{dt}\Delta t \tag{4.1}$$

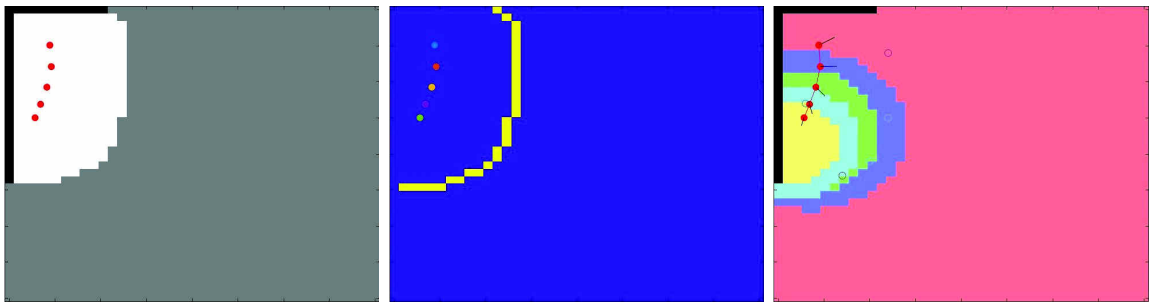
It is worth to mention that the simulations were run in Matlab, however there were not exclusive or high level functions used, so the codes may be exported with minor changes to another object oriented languages like C++, Java or Python.

4.1. Exploration

As mentioned earlier, the exploration task is the first task to be done in the mission execution. In order to realize this process in an coordinated way we employed the Algorithm DisCoverage which behavior is presented in simulation in a very simple convex space and in different non-convex spaces. For all the exploration simulations it was considered a team of 5 robots with integrator dynamics in an area of 40x40 square meters with a grid resolution of 1 square meter.

4.1.1. Heterogeneous Convex Case

In this case there is not a single obstacle into the space, then the euclidean distance direct calculus is appropriate for the development of the task. Contrary to the original DisCoverage algorithm presented in [16] the velocity of the robots is not homogeneous, in this case the velocities of the robots are 5, 6, 7, 8 and 9 meters per second and they start the top left corner of the mission environment. the communication range of the robots correspond to $\rho_2 = 30$m (chosen to be able of visualize the effects of the communication constrains), the security range for collision avoidance corresponds to $\rho_1 = 2$m, the initial distribution of robots consist in 5 robots in a straight line near to the top-left corner of the scene with simple connections with the nearest neighbors, the simulation time step was fixed to $\Delta t = 0{,}1$s and the simulations were ran from $t = 0$s up to $t = 12$s (When the exploration is over the robots return to the top left corner to an arbitrary D shaped formation). Regarding to the robots constants, a sensing range S_r of the same magnitude of its speed, and a proportional constant for the Voronoi Centroids tracking K_p of the same value.
In Figure 4.1 there are presented multiple images related to the map built by the robot team moment just after the start of the exploration. In Figure 4.1a there is the information of the known and unknown cells at the start of the mission, in Figure 4.1b is the information of the frontier δS between the know and the unknown space and in the Figure 4.1c there is the information related to the Voronoi cells and the communication links of the robots showing the initial line configuration. It is worth to highlight the shape and size of each Voronoi cell since the arrival time based tessellation changes significantly compared to the original algorithm where the cells' frontiers were straight lines and now are curved that grow noticeably as the speed of the robot grows.



(a) Actual Estimated Map Generated by the robots' team

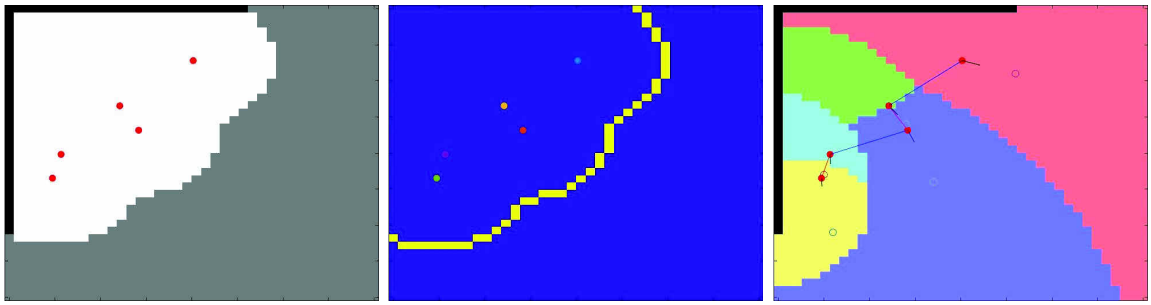
(b) Frontier δS between the known and unknown space

(c) Voronoi Cells of the robots with communication links

Figure 4.1.: Exploration in Convex Spaces - $t = 0{,}1s$ The black areas correspond to the obstacles, the white area correspond to the empty space, the gray area corresponds to the unknown area and the color circles correspond to the robots.

In Figures 4.2a and 4.2b it is possible to see how the exploration frontier is pushed back

increasing the known area and also that the faster robots get further than the slower ones as expected. It is worth to highlight the idle robot behavior that makes that the slower robots that do not have frontier within their cells are tracking the faster ones (instead of the geometric centroid of their cells) getting closer to the frontier being ready for the moment when they reach it again. On the other hand regarding to the network topology, the initial line configuration has been maintained and there are no links near to be broken since all the robots are still relatively close.



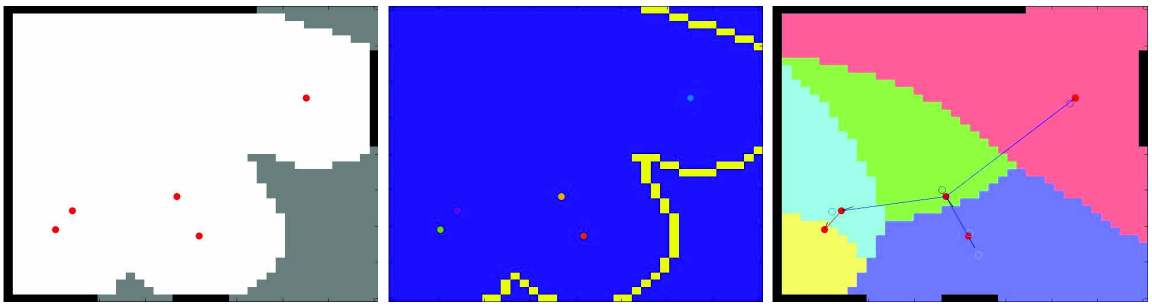
(a) Actual Estimated Map Generated by the robots' team

(b) Frontier δS between the known and unknown space

(c) Voronoi Cells of the robots with communication links

Figure 4.2.: Exploration in Convex Spaces - $t = 2{,}0s$ The black areas correspond to the obstacles, the white area correspond to the empty space, the gray area corresponds to the unknown area and the color circles correspond to the robots.

In Figure 4.3 it is possible to see how the exploration process has advanced, in particular in Figure 4.3c it is possible to see that the network topology has changed and as expected in Figure 4.3a it is possible to watch that the fastest robot which is near to the top right corner is far from the rest of the team since its velocity allows it to explore a bigger area faster than the other ones.



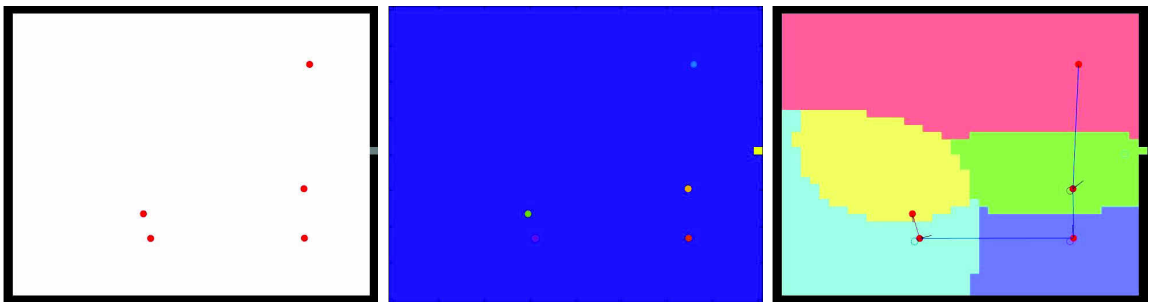
(a) Actual Estimated Map Generated by the robots' team

(b) Frontier δS between the known and unknown space

(c) Voronoi Cells of the robots with communication links

Figure 4.3.: Exploration in Convex Spaces - $t = 4{,}0s$ The black areas correspond to the obstacles, the white area correspond to the empty space, the gray area corresponds to the unknown area and the color circles correspond to the robots.

In Figure 4.4 there is presented the moment right after the end of the exploration (there was a single cell remaining to be explored). In this moment it is possible to see that the algorithm has fulfilled the exploration tasks as expected preserving the network connectivity even when the network topology has changed in different time instants. Also it is interesting to watch the shapes of the Voronoi cells and the relative position of the robots within them since most of the robots do not have frontier anymore within its own cell and then, they are tracking their neighbor who has it the closest.



(a) Actual Estimated Map Generated by the robots' team

(b) Frontier δS between the known and unknown space

(c) Voronoi Cells of the robots with communication links

Figure 4.4.: Exploration in Convex Spaces - $t = 6{,}3s$ The black areas correspond to the obstacles, the white area correspond to the empty space, the gray area corresponds to the unknown area and the color circles correspond to the robots.

4.1.2. Heterogeneous Non-Convex Case

In order to study the behavior if the DisCoverage algorithm in the non-convex spaces we present three different simulations environments, the first two correspond to pseudo-aleatory scenarios related to scenes with debris on it and the last one correspond to an structured environment that includes three walls that can be related to an urban environment where there are rooms and a hall. It is worth to remember that in this case the direct euclidean distance calculus is not effective anymore, then the distance metric employed corresponds to the 8NF distance metric.

In Figure 4.5, 4.6 and 4.7 there are presented the simulation of the the exploration tasks from 4 different perspectives. First there is presented the known map of the environment in order to show the exploration advance, second the 8NF distance matrix of an arbitrary robot in order to show the behavior of the distance metric in the different spaces, third the current exploration frontier which is tracked by the robots, and finally the Voronoi cell's of the robots with the communication links in order to show the changing nature of those and to show the nature of the exploration assignments.

In all the environments, first we have all the robots in the top left corner of each environment (we call this initial position as home position), then the team starts the exploration task tracking the frontier within its own cell or the closets neighbor in the network that has frontier at sight. Since each robot has its own velocity so, we can see that their Voronoi cell's differ in their sizes noticeably highlighting that the fastest robot is assigned to the furthest locations in the map. It result interesting to observe that the frontier is only considered as the limit between the known empty space and the unknown since it is possible that the unknown space behind the obstacle cells may also be obstacle and tracking it does not help in the exploration process.

On the other hand in the second column of Figures 4.5, 4.6, and 4.7 where the 8NF is presented as a color plot where the colder colors correspond to a closer distance and the warmer correspond to the further (note that the obstacles appear in blue also since they are noted as -1 in the algorithm). It is possible to watch that the approximation employed using only the orthogonal and diagonal movements is reasonably close to the geodesic distance and as expected the biggest variation can be found at $n45 \pm 22{,}5$ degrees. Specially in Figure 4.7 it is possible to notice that points that appear to be close are not really close since sorting the obstacle demands a longer routes to be reached. In particular it is worth to mention that when the obstacles are not fully identified the unknown area is supposed as reachable, then the distance matrix is evaluated within some obstacles as if they are hollow but when a closed area is determined all the points within it are set as unreachable.

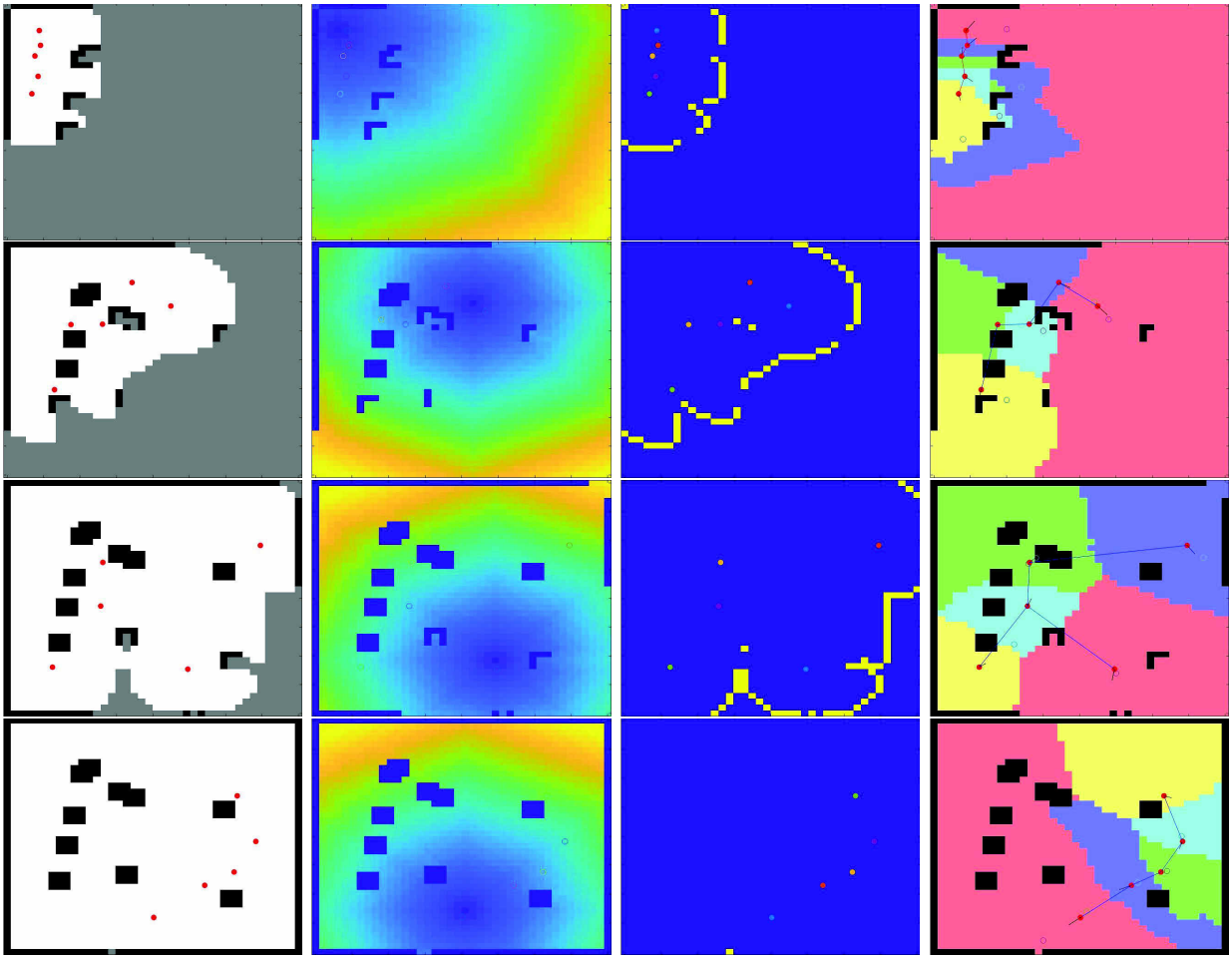


Figure 4.5.: Exploration in Non-Convex Spaces - First environment, pseudo-aleatory scene . The first column shows the actual estimation of the map, the second shows the 8NF distance metric of one of the robots of the team (the colder the color, the closer), the third shows the current exploration frontier and the last shows the current Voronoi tessellation of the environment

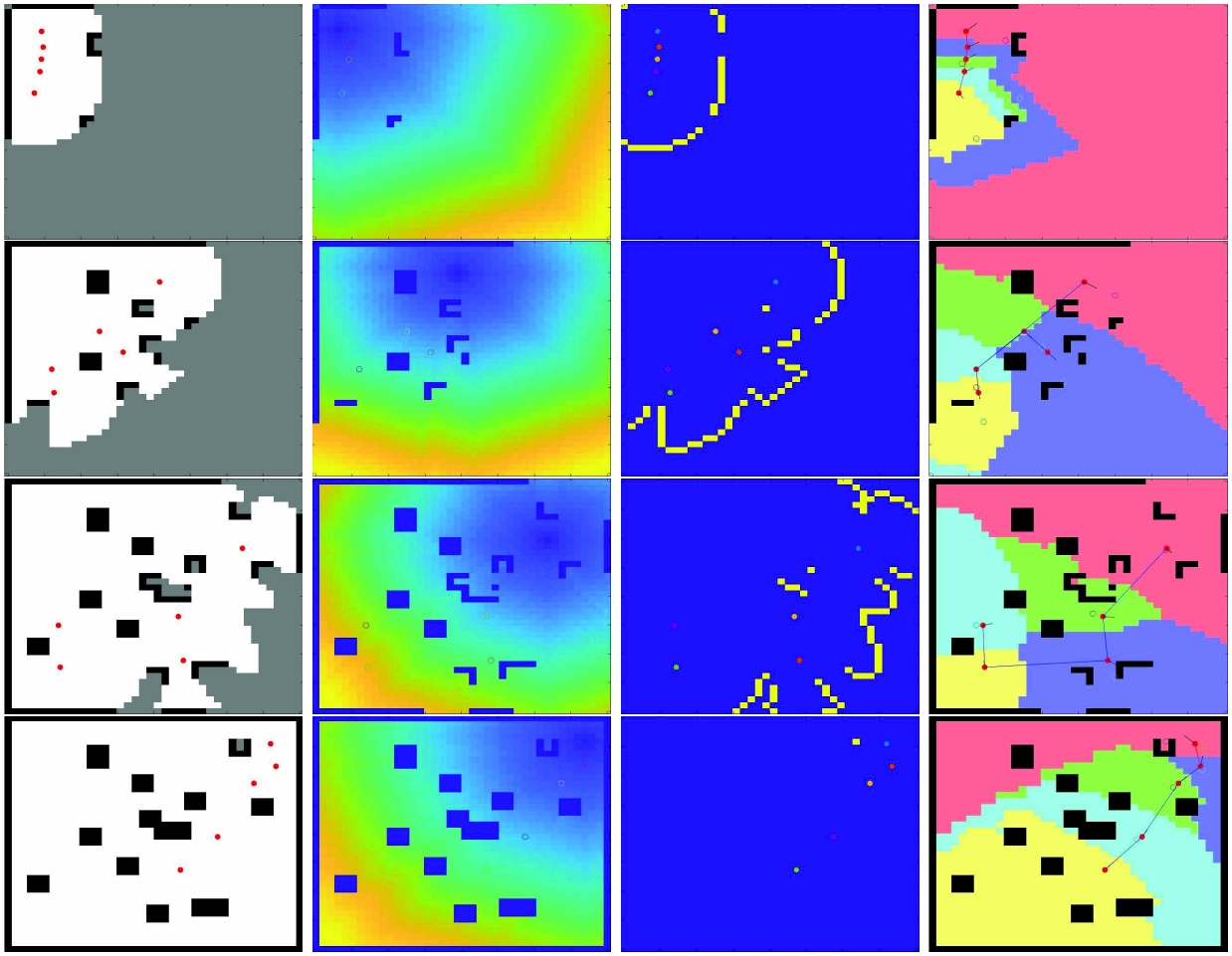


Figure 4.6.: Exploration in Non-Convex Spaces - Second environment, pseudo-aleatory scene . The first column shows the actual estimation of the map, the second shows the 8NF distance metric of one of the robots of the team (the colder the color, the closer), the third shows the current exploration frontier and the last shows the current Voronoi tessellation of the environment

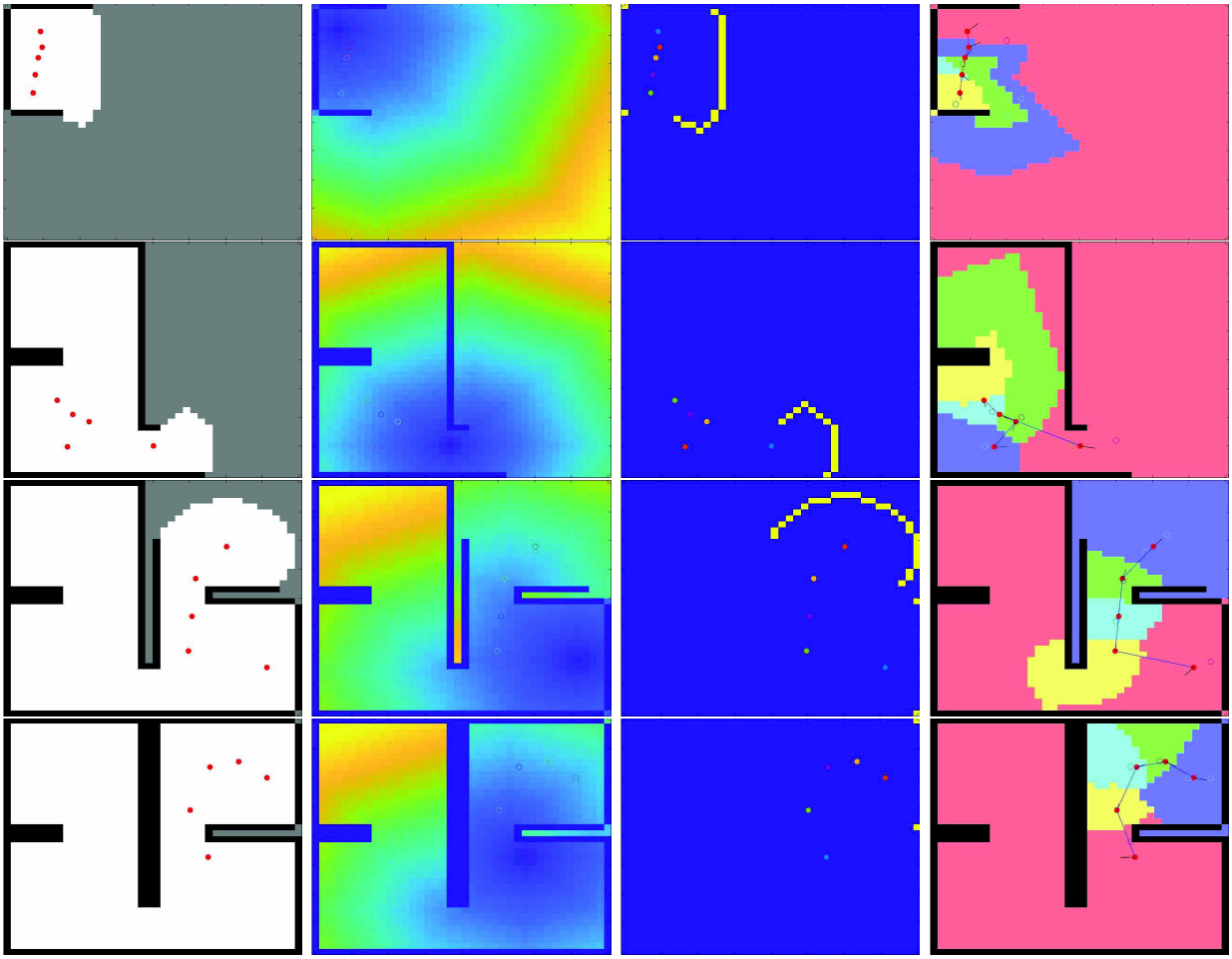


Figure 4.7.: Exploration in Non-Convex Spaces - Third Scenario, structured scene with multiple walls and rooms. The first column shows the actual estimation of the map, the second shows the 8NF distance metric of one of the robots of the team (the colder the color, the closer), the third shows the current exploration frontier and the last shows the current Voronoi tessellation of the environment

4.2. Supplies Delivery and Victims Identification and Evacuation

In order to verify the task completion there were realized multiple simulations of each one evaluating its completion. Considering that most of tasks are related to the behavior of the victims it was necessary to include a simple model of their behaviors during each one of the tasks.

4.2.1. Victims Model

The victims dynamic was modeled as single integrator dynamics identical to the employed for the robot model. The victims are initially set to stay in their location and if a robot instruct them to follow it, they will track it with a proportional control signal (respecting certain radius) if the robot is in their line of sight. Note that the line of sight is symmetric so if the victim is in the line of sight of the robot, the robot is in the line of sight of the robot. The collision avoidance potential was also included for the victims in order that they do not collide against the obstacles in the environment while they follow the robots. If a victim reaches the safe area, then continues by itself the way to the top left corner of the environment to avoid the interference of it in the movement of the rest of the robots.

4.2.2. Victims Reaching

In order to deliver a supply or to guide the evacuation of a victim is required that the robot reaches its position. In Figure 4.8 it is presented how a robot with full knowledge of the map approaches to a victim from 3 different perspectives, first the location of the robot, the victim and the current target of the robot (the projection onto the line of sight of the robot of its real goal); second the line of sight matrix that indicates if the point is visible or not to for the robot; and finally the 8NF distance matrix in order to highlight the evolution of the metric through the route. The victim is placed in a position far from it with multiple corridors. The navigation is made employing an optimized route determined tracing back the interest point based on the 8NF distance matrix with an gradient descendant approach. It is worth to highlight that even if the desired route points may be dangerously close to the obstacles, the artificial potentials ensure that the robot will not collide against them.

This particular simulation allows to see clearer the conjugation between the path planning obtained from the 8NF distance and the line of sight of the robot. It shows that the robots are capable of determine routes in non-trivial environments without requiring an additional path planning algorithm employing the matrices determined earlier for the completion of the exploration task.

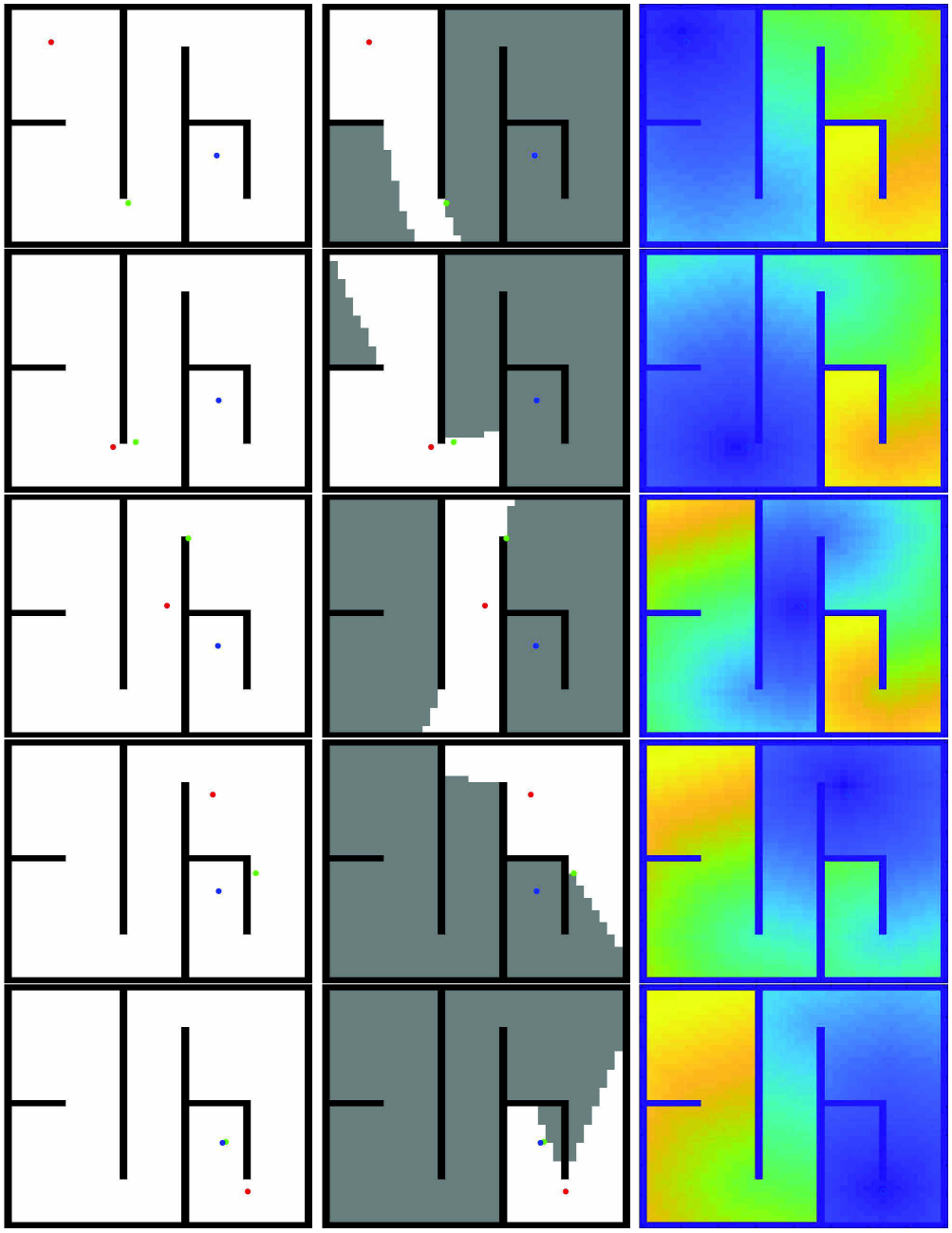


Figure 4.8.: Victim Reach Simulation - the first column correspond to the position in the general map of the robot shown in blue, the robot shown in red and the target within the line of sight of the robot presented in green. The second column shows the line of sight of the robot in white, the obstacles in black and the non visible area in gray. Finally the third column shows the 8NF distance matrix of the robot

4.2.3. Victims Evacuation

Once a victim capable to move is found, one of the robots of the team must guide him/her from its actual position to the safe area, in order to show this, the previous scenario was used again to show the return route employed by the robot maintaining certain distance in order to avoid collisions against the victim. In Figure 4.9 there are presented the same three perspectives shown in the previous tasks. It is worth to highlight the behavior of the simulated victim related to the tracking and the fact that the robot is in charge of the navigation and the maintenance of the mutual visibility. If the visibility is lost, the robot will go back for the victim but the victim will stay in its current location.

4.3. Task Reallocation

As an example of the task allocation procedure we captured an arbitrary situation shown in Figure 4.10 and present the time costs for each possible task in Table 4.1. The situation corresponds to an advanced stage in the mission when the robots have already finished their exploration tasks and are considering the distribution of the victims' attendance tasks. It is possible to notice that there are multiple tasks that have an infinity cost associated. This happens for multiple reasons:

- There are certain victims that have been already attended (the required supply was already delivered)
- There are some casualties (or fake positives) detected that do not have task related to them.
- Since the robots are heterogeneous related to their capabilities to accomplish certain tasks, it is possible that the task is pending to be done, but the robot can not fulfill it.
- since the exploration task is already completed, it does not have sense for the robots to explore

Note that in order to use the Hungarian method for the allocation the exploration task for each robot is considered as a set of independent tasks that has only sense for the corresponding robot so the cost will be infinity for the non-corresponding.
When an event occurs, the allocation is changed employing the Hungarian Method in order to maintain the optimality of the allocation. In this particular situation the correspondent result is summarized in Table 4.2. It is worth to highlight the fact that the robot 1 is unable to deliver supplies, so the time related is infinity and the Hungarian Method will never assign that type of tasks to it. On the other hand, the robots 2 and 4 are capable of fulfilling all the possible pending tasks so they may be potentially assigned to any of them. Note that the allocation generated is not necessarily trivial since the robot 2 can attend the victim 5

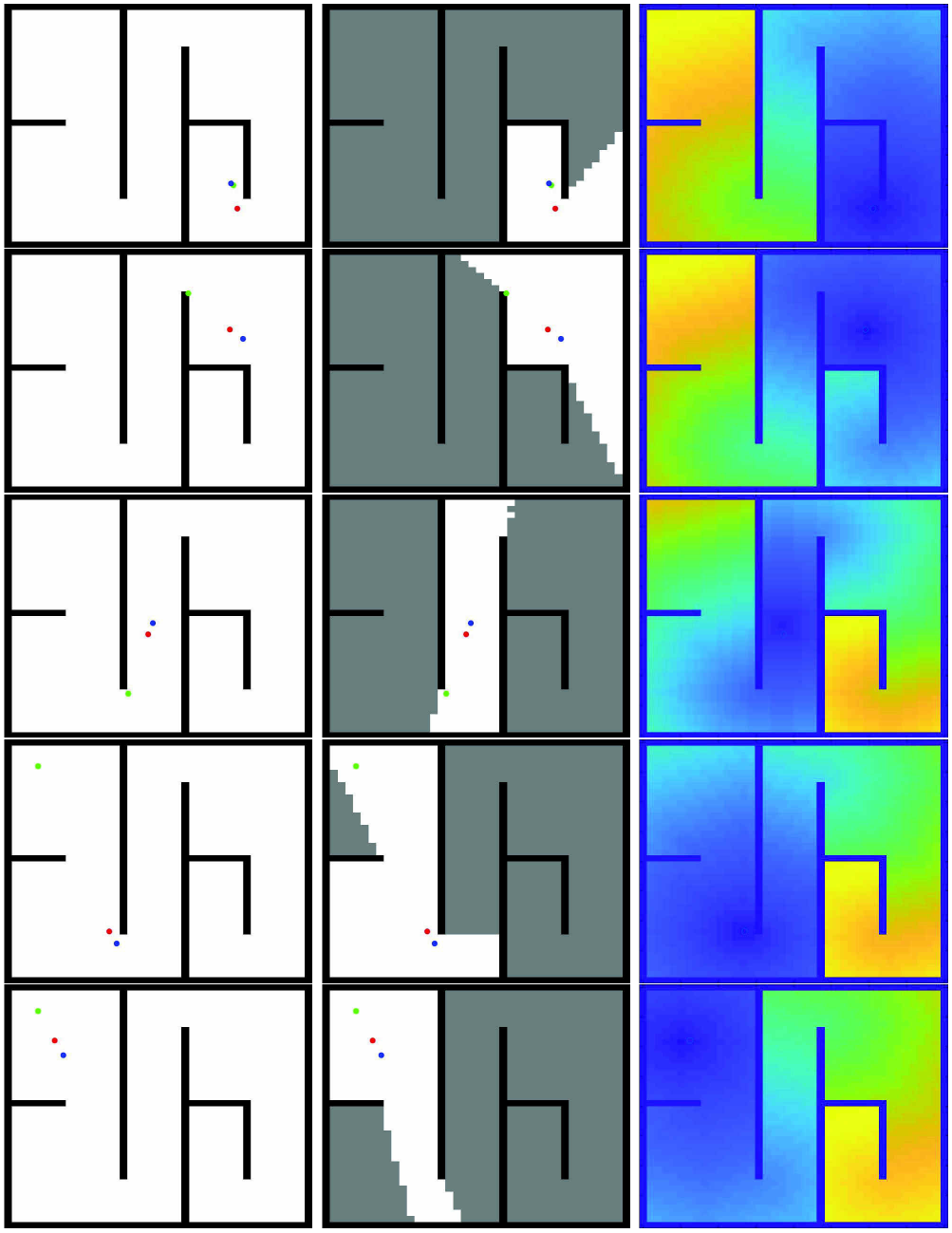


Figure 4.9.: Victim Evacuation Simulation - the first column correspond to the position in the general map of the robot shown in blue, the robot shown in red and the target within the line of sight of the robot presented in green. The second column shows the line of sight of the robot in white, the obstacles in black and the non visible area in gray. Finally the third column shows the 8NF distance matrix of the robot

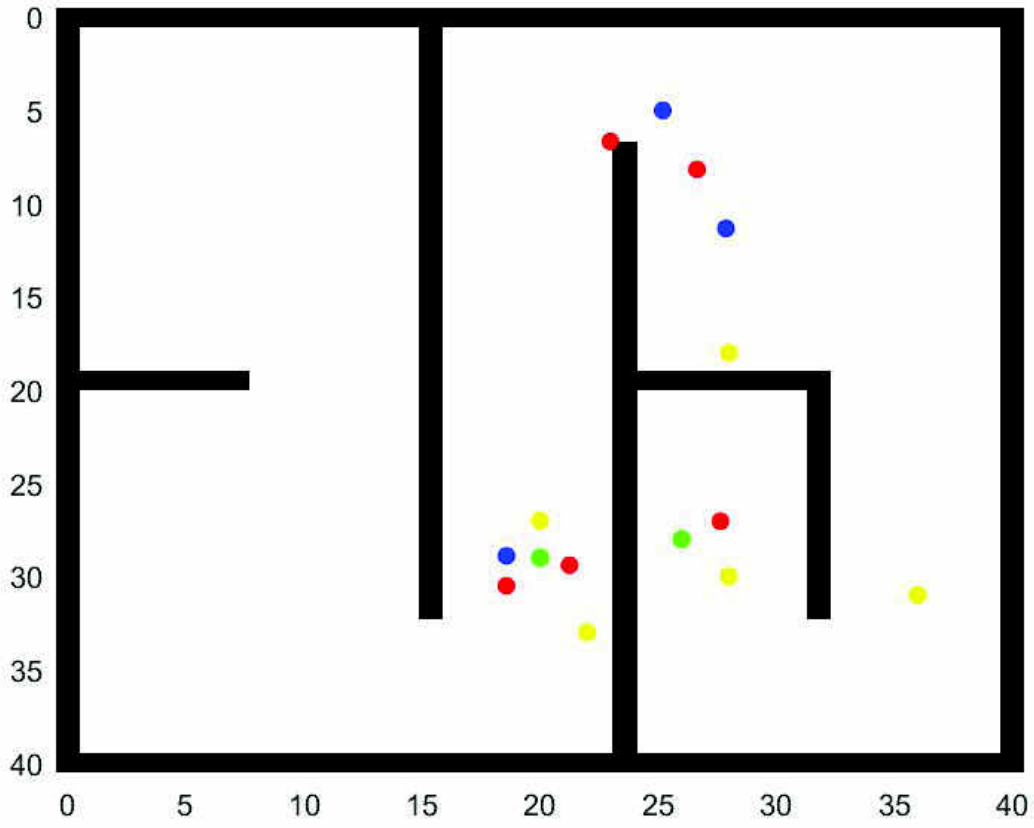


Figure 4.10.: Example Situation for the task allocation procedure - The robots are presented in red, the victims capable to move are presented in blue, the victims unable to move are presented in green and the casualties are presented in yellow.

Table 4.1.: Time Costs Table Example

Task \ Robot	1	2	3	4	5
attend victim 1	∞	∞	∞	∞	∞
attend victim 2	∞	∞	∞	∞	∞
attend victim 3	7.9205	11.0873	9.0896	26.2161	∞
attend victim 4	∞	∞	∞	∞	∞
attend victim 5	∞	0.1768	∞	10.3117	0.4828
attend victim 6	11.7903	4.2589	16.0854	35.3206	∞
attend victim 7	7.3840	10.6302	9.6120	28.1589	∞
attend victim 8	∞	7.9105	∞	0.2357	12.4083
attend victim 9	∞	∞	∞	∞	∞
attend victim 10	∞	∞	∞	∞	∞
robot 1 exploration	∞	∞	∞	∞	∞
robot 2 exploration	∞	∞	∞	∞	∞
robot 3 exploration	∞	∞	∞	∞	∞
robot 4 exploration	∞	∞	∞	∞	∞
robot 5 exploration	∞	∞	∞	∞	∞

faster than the robot 5, but the last was assigned to that task since the total time employed by the team was reduced if the robot 2 attends the victim 6 instead of the 5.

Table 4.2.: Task Allocation Example

Robot	1	2	3	4	5
Task Type	evacuation	evacuation	evacuation	supply delivery	supply delivery
Victim ID	7	6	3	8	5

4.4. Full Mission

In Figure 4.11 we present the map corresponding to a complete mission scenario where the robots have to explore the scene in the search of the potential victims, determine the state of each one, accomplish the tasks corresponding to each victim and return to a Home position.

4.4.1. Full Mission Simulation

In this simulation the initial position of the robots is the same line configuration presented in the exploration task section, however in this case there were 10 victims randomly allocated in the right half of the environment and with random status (4 victims that can move, 3 victims that cannot move and 2 casualties). It is worth to remember that the victims capable to move have simple dynamics programmed that consider the collision evasion, follow a robot if the robot calls for it and go to the top left corner (safe zone) once the robots takes then near enough. They were programmed with a speed of 5 km/h that is a typical speed for a person walking.
In Figure 4.12a we present a situation when during the exploration task there were 4 robots found and identified victims, however the task allocation algorithm has determined most of the robots to continue exploring and two of them were set to deliver supplies to the non-capable to move victims.
In Figure 4.12b we present the moment when the robots found and identified the last of the victims and did not explore anymore. Note that in this particular case the expected number of victims was set to the real number of them, if the expected number would have been greater, then the robots would have explored the entire map.
At the moment when the exploration had ended, all the non-capable to move victims had already received their medical supplies, so 4 of the robots were allocated to the evacuation task and the other was set to return home since there were not more task pending to be done. Later in Figure 4.13a we present the process of evacuation, in this moment the victims are being evacuated by the robots to the top left corner of the environment (the safe zone). Note that all the robots that are evacuating victims are close enough to their victims, this

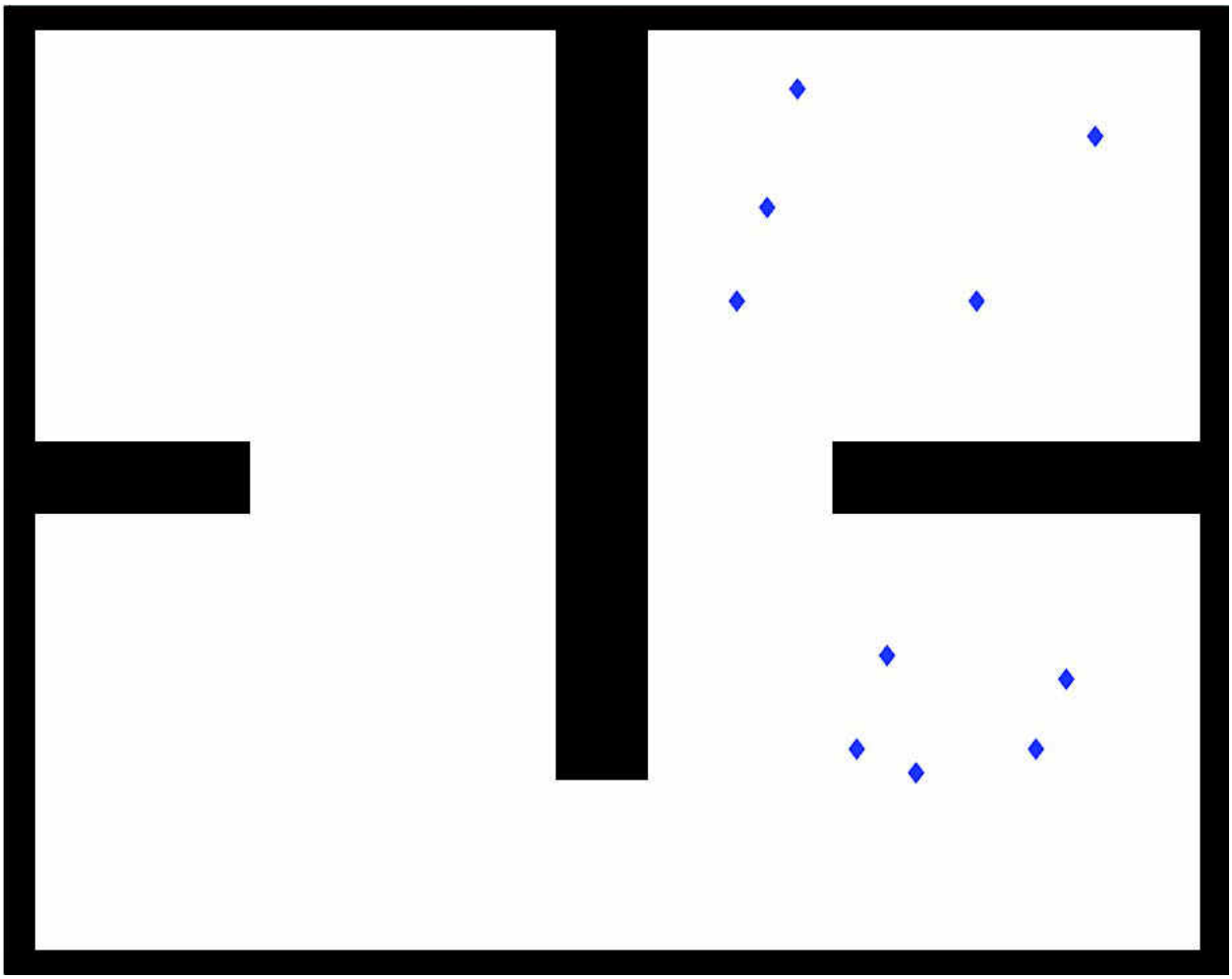
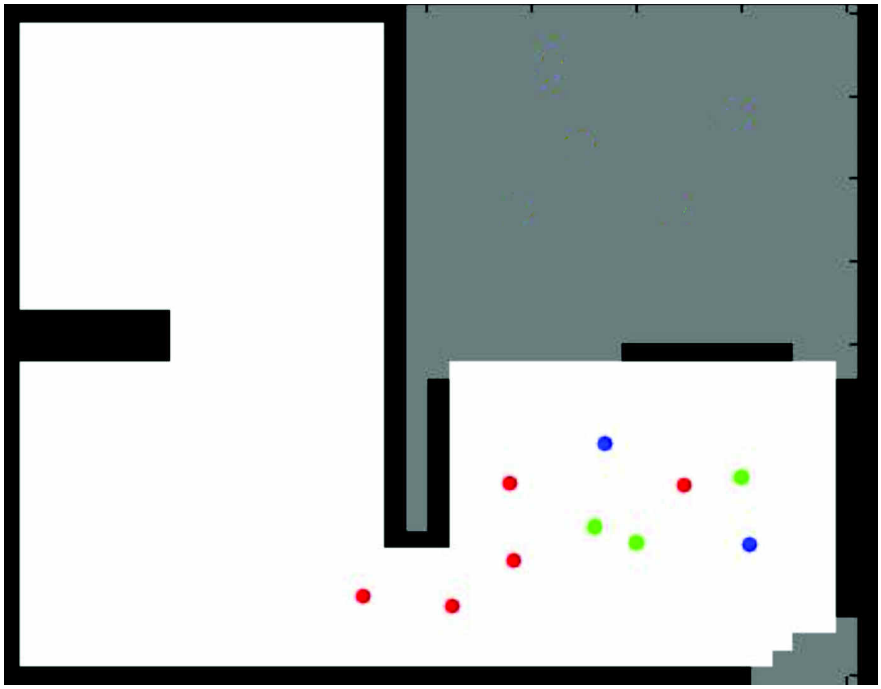
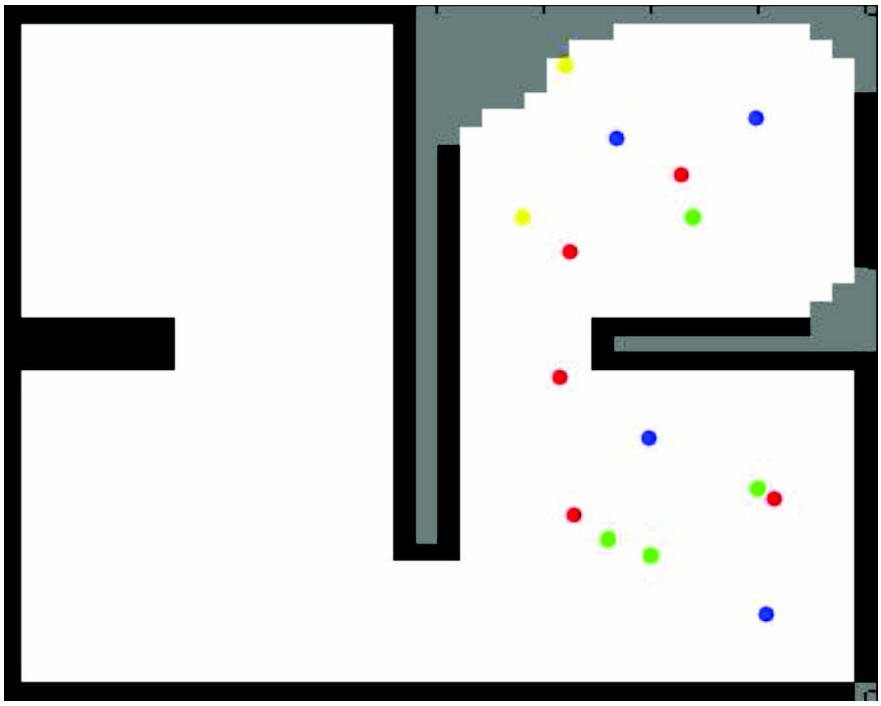


Figure 4.11.: Scene and Victims Location for the Simulations

behavior is expected and will be hold even if the victim's speed and robot's speed differ (actually in most cases this is true). Finally in Figure 4.13b we show the final state of the mission, here the robots are set to an arbitrary formation, the capable victims were allocated, the medical supplies were delivered and all the victims and casualties locations were recorded in the robots memories.

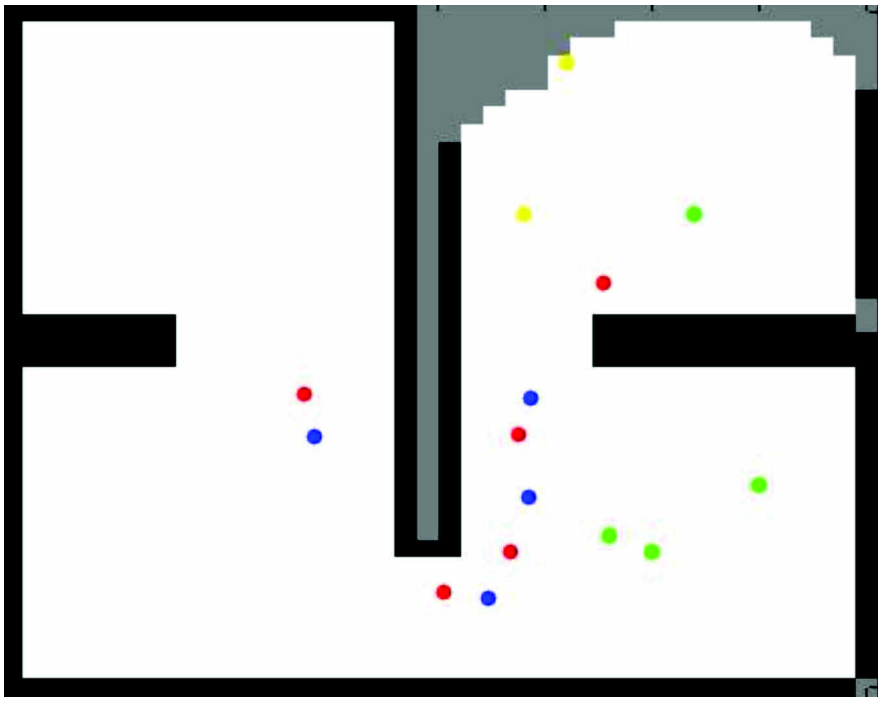


(a) First Victims Identified

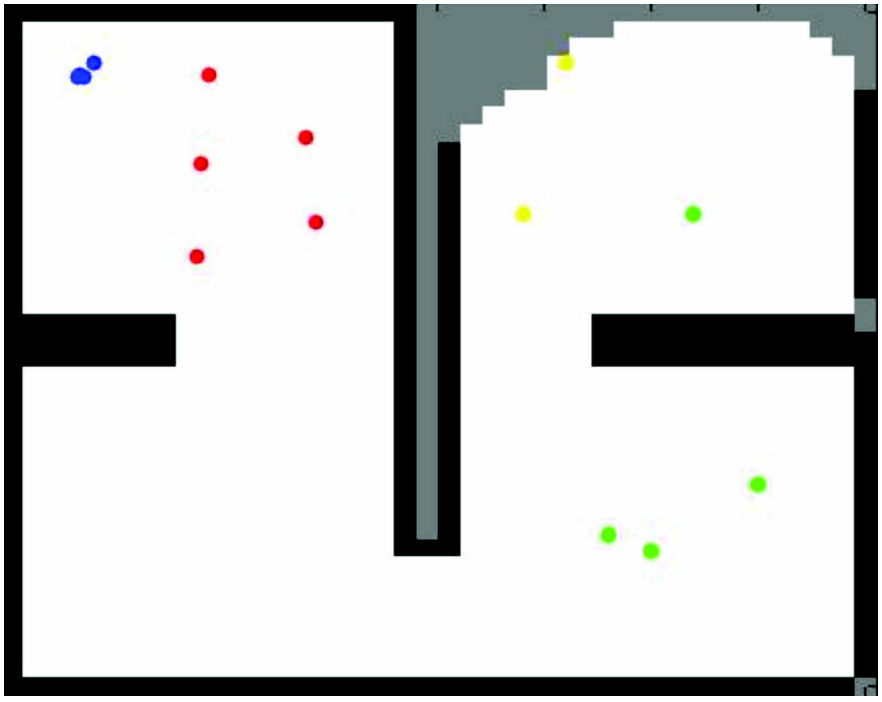


(b) End of Exploration Process

Figure 4.12.: Victims detection, classification and attention - The Blue Dots correspond to victims capable to move, the green ones correspond to the victims that are unable to move and the yellow ones to the casualties found



(a) Evacuation of capable victims



(b) End of mission

Figure 4.13.: Victims detection, classification and attention - The Blue Dots correspond to victims capable to move, the green ones correspond to the victims that are unable to move and the yellow ones to the casualties found (continuation)

5. Implementation

In order to show the applicability of the algorithms developed in real environments, there were realized some basic experiments with real robots showing the work flow and some of the considerations required to implement the models presented in this document in non-simulation environments. For this purpose, there were employed two Kobuki Turtlebots and one Festo Robotino connected employing a wireless network to perform the exploration of a non-convex environment. First, in Section 5.1 we introduce the robots characterization considered for the test, the physical environment considered and the communications network structure employed. In Section 5.2 we present the kinematic modeling of the two classes of robots considered focused on the idea of the decoupled integrator dynamics and how to achieve it in each case. In section 5.3 we present the determination of the security parameters required for the model constants in order to guarantee the success of the mission. Finally, in section 5.4 we show an exploration scenario with all the emergent issues of the implementation as well as the simplifications and adaptations employed for the experiment.

5.1. Experimental Setup

There are 3 main components of the experimental setup considered that describe the physical resources required and employed. First, the robots employed with their corresponding specifications; second, the environment where the robots are going to move and to map; and finally, the wireless communication network infrastructure.

5.1.1. Robots

We employed three robots from two different classes:

- Two differential traction Kobuki Turtlebots 2, one equipped with a Hokuyo LIDAR (Laser Imaging Detection and Ranging) sensor with an aperture angle of 240 deg and the other with a RPLIDAR Sensor with an aperture angle of 360 deg, both have bumper sensors, gyroscope, its own on-board PC with the specifications given in Table 5.1, and a single DoF IMU (Inertial Mass Unit). These robots have a maximum linear speed of $0{,}7[m/s]$ and a recommended maximum angular speed of $110[\text{deg}/s] = 11\pi/18[rad/s]$ (to maintain the gyroscope performance).

- one omni-directional traction Festo Robotino equipped with Bumper sensors, gyroscope, its own on-board PC with the specifications given in Table 5.1, infrared depth sensors, inductive sensors, and also a Hokuyo LIDAR sensor. This robots has a maximum linear speed of about 2,7$[m/s]$ and since in its documentation there are not indications related to the angular speed, it was considered a maximum angular speed of 110[deg $/s$] = $11\pi/18[rad/s]$ Similarly to the Kobukis.

Table 5.1.: PCs specifications

Specification	Kobuki	Robotino	Master
Ram Memory	1 GB	1 GB	16 GB
Processor	Intel Atom N2600 1.6GHz	Intel Atom N2600 1.6GHz	AMD FX-7600P Radeon R7 2.7GHz
Graphics Card	none	none	AMD R9 M280x 2GB
USB Ports	3 (1 USB2)	3 (1 USB2)	3(2 USB2)

5.1.2. Demonstration Scene

The environment where the exploration algorithm was tested is a closed environment with some obstacles that is presented in Figure 5.1. This corresponds to the table tennis training room of the National University of Colombia where we employed some separators in order to delimit the interest area. The separators where employed for two main reasons. First because the separators are easy to detect for the robots considering that they sensors employed only allow a 2D detection of light reflecting objects around the robot (they do not "see"glass or other refractive objects). And second, because they are easy to handle to setup the required scene. This environment was chosen because it allows to show the applicability of the algorithms since it is a non-convex environment with multiple obstacles (which is the target of this algorithms) and is a valid example of an urban area. Regarding to the environmental conditions of the room, the floor is tiled and its lightning is a mixture between natural light that comes from multiple windows and artificial light that comes from 3 roof lamps.

5.1.3. Network Structure

Since the robots control and algorithms proposed may be quite heavy for the hardware architectures of some the robots, it was decided to emulate the required network through a virtual network supported on a master desktop computer (with the specifications presented in Table 5.1) which calculates the required operations to determine the interest points required for the calculus of the control signals to be send to the robots through a wireless network.

Figure 5.1.: Demonstration environment - Table tennis training room of the National University of Colombia

The control signal is evaluated onto each robot, based on the maps known, the information of its sensors and odometry, and the interest point sent by the master. However, within the virtual network, the robots (understood as virtual agents with a physical body and virtual controller) do not consider that there is a master or that the control signal parameters Are evaluated outside but inside them, and the messages sent or received only correspond to the ones that they can exchange employing the virtual links that are available.
In order to create the robots network, it was employed the pseudo-operative system ROS (Robot Operating System) which is a software platform that runs mainly with Linux and allows to generate robots networks in a standardized way employing code instructions in multiple programming languages as c++ or Python between others [24]. This Platform also includes multiple libraries that allow to control the hardware of multiple robots including the ones chosen for this work.

5.2. Dynamics Reduction

As mentioned in Chapter 2, the dynamics of the robots are non-linear and do not correspond to simple integrators. Therefore, we have to analyze the kinematic model of each of the classes of robots to be employed in order to make valid the integrator dynamics assumption and this analysis depends on the physical architecture of each robot.

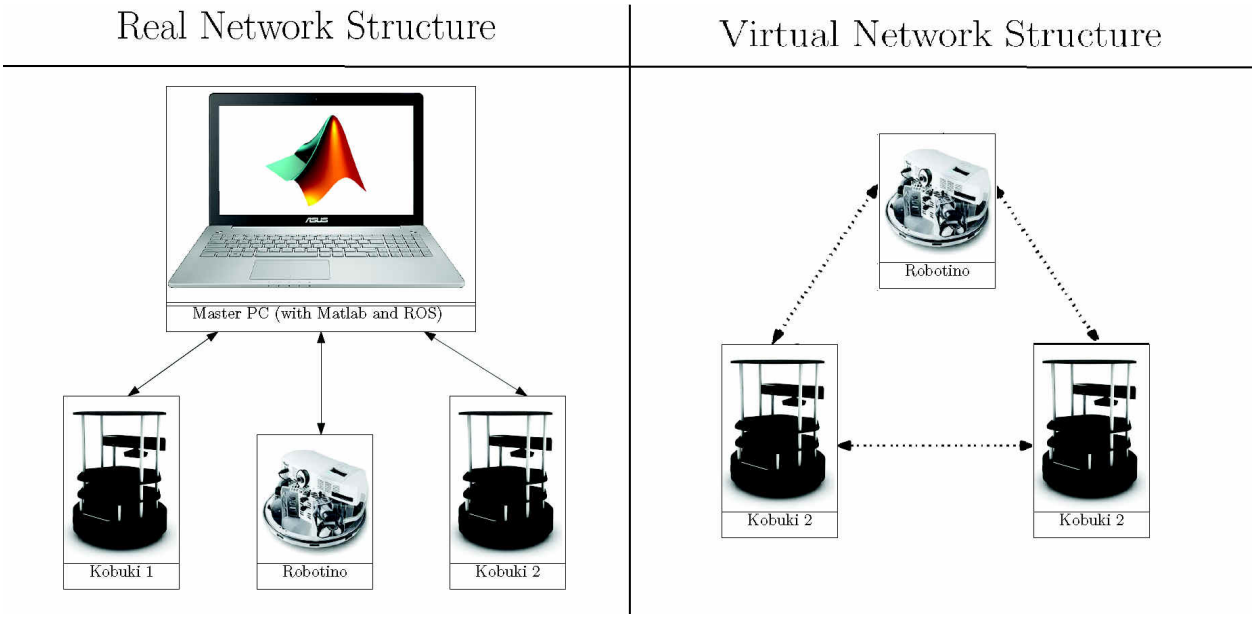


Figure 5.2.: Phisical and virtual network architectures employed - In the real network structure, the robots do not communicate between them but with a master PC that evaluates the interest point calculus, merges the individual maps information and sends this to each individual robot. In the Virtual network, the robots communicate between them using the links determined by the reconfiguration algorithm and ignore that there is a master PC.

5.2.1. Kobuki Robots

The Kobuki Turtlebots are differential robots that following the kinematic modeling presented in [29] are modeled as presented in Equation (5.1) relating the angular speed of its wheels to the position of its inertial reference frame.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix} \tag{5.1}$$

where x, y are the coordinates of the robot related to the map M, θ corresponds to the orientation of the robot (measured counterclockwise), r_w corresponds to the radius of the robot wheels, L corresponds to the distance between the robot's mass center, V corresponds to the linear velocity of the robot measured from its own inertial mass frame, and ω is the Rotational speed of the robot measured from its own inertial mass frame.
In this point, there were considered two options:

- Employ an inner kinematic proportional control loop as presented in (5.2) (also taken from [29]) tuning the control constants K_ρ, K_α and K_θ respecting $K_\alpha > K_\rho > 0$ and $K_\theta < 0$ in order to guarantee the stability and considering that the dynamic response of the system may introduce certain turning radius that must be considered in the security radius considerations.
- Decomposing the movement action into a previous rotation followed by the straight translation for each new movement desired. Note that this introduces a maximum delay of about 1.6 seconds (considering that the maximum angular speed allowed for the robot correct operation that is 110 degrees/second) which for most situations can be neglected.

$$\begin{aligned} \alpha &= atan2(y_d - y, x_d - x) \\ \begin{bmatrix} V \\ \omega \end{bmatrix} &= \begin{bmatrix} K_\rho \sqrt{(x_d - x)^2 + (y_d - y)^2)} \\ K_\alpha(\alpha - \theta) + K_\theta(\theta_d - \theta)) \end{bmatrix} \end{aligned} \tag{5.2}$$

In particular we are following the second approach since it results simpler for the implementation and the delay introduced for the pre-orientation is generally short enough to be neglected for our purposes. Note that the collision avoidance radius for this robots must be adjusted to be at least the presented in (5.3),

$$\begin{aligned} T_g &= 180[deg]/110[deg/s] = \frac{18}{11}[s] \\ \rho_1 > T_g V_i &= \frac{18}{11}[s]0{,}7[m/s] = \frac{63}{55}[m] \approx 1{,}145[m] \end{aligned} \tag{5.3}$$

where Tg is the time required for the robot to do a turn of 180 deg.

5.2.2. Festo Robotino

In order to determine the kinematic model of the Robotino Robot, we considered its wheels restrictions. The rolling constrain structure is presented in Equation (5.4) and the sliding constrain structure is presented in Equation (5.5). Note that this particular robot has 3 sweedish wheels, that have $\gamma = \pi/2$ and allow movement in the rolling and the sliding directions.

$$[sin(\alpha_i + \beta_i), -cos(\alpha_i + \beta_i), (-l)cos(\beta_i)]R(\theta_i)\dot{\xi}_I - r\dot{\varphi}_i = 0 \tag{5.4}$$

$$[cos(\alpha_i + \beta_i), sin(\alpha_i + \beta_i), (l)sin(\beta_i)]R(\theta)\dot{\xi}_I + r_{sw}\dot{\varphi_{swi}} = 0 \tag{5.5}$$

Now if we form the restrictions matrices of the robots considering the two restrictions of each one of the three wheels as presented in Equation (5.6),

$$J_1 = \begin{bmatrix} sin(\alpha_1+\beta_1) & -cos(\alpha_1+\beta_1) & (-l_1)cos(\beta_1) \\ sin(\alpha_2+\beta_2) & -cos(\alpha_2+\beta_2) & (-l_2)cos(\beta_2) \\ sin(\alpha_3+\beta_3) & -cos(\alpha_3+\beta_3) & (-l_3)cos(\beta_3) \\ cos(\alpha_1+\beta_1) & sin(\alpha_1+\beta_1) & l_1 sin(\beta_1) \\ cos(\alpha_2+\beta_2) & sin(\alpha_2+\beta_2) & l_2 sin(\beta_2) \\ cos(\alpha_3+\beta_3) & sin(\alpha_3+\beta_3) & l_3 sin(\beta_3) \end{bmatrix}$$

$$J_{2w} = \begin{bmatrix} r_{w1} & 0 & 0 \\ 0 & r_{w2} & 0 \\ 0 & 0 & r_{w3} \end{bmatrix}$$

$$J_{2sw} = \begin{bmatrix} r_{sw1} & 0 & 0 \\ 0 & r_{sw2} & 0 \\ 0 & 0 & r_{sw3} \end{bmatrix}$$

$$J_2 = \begin{bmatrix} J_{2w} & 0 \\ 0 & J_{2sw} \end{bmatrix}$$

$$J_1 R(\theta)\dot{\xi}_I = J_2 \begin{bmatrix} \dot{\varphi} \\ \dot{\varphi_{sw}} \end{bmatrix} \tag{5.6}$$

In this model, we have three control variables $\dot{\varphi_w}$ and three dependent variables $\dot{\varphi_w}$ which are mechanically set to fulfill the restriction model. In order to determine the required $\dot{\varphi_w}$ to impose a determined $\dot{\xi}_I$ we only need to consider the rolling constrains as presented in Equation (5.7),

$$\dot{\varphi_w} = \begin{bmatrix} 1/r_{w1} & 0 & 0 \\ 0 & 1/r_{w2} & 0 \\ 0 & 0 & 1/r_{w3} \end{bmatrix} \begin{bmatrix} sin(\alpha_1+\beta_1) & -cos(\alpha_1+\beta_1) & (-l_1)cos(\beta_1) \\ sin(\alpha_2+\beta_2) & -cos(\alpha_2+\beta_2) & (-l_2)cos(\beta_2) \\ sin(\alpha_3+\beta_3) & -cos(\alpha_3+\beta_3) & (-l_3)cos(\beta_3) \end{bmatrix} R(\theta)\dot{\xi}_I \tag{5.7}$$

Noting that this is an holonomic model. Therefore, we can impose the desired behavior directly by the inversion of the Jacobian matrix achieving the desired integrator decoupled dynamics. On the other hand, it is worth to mention that the velocity of the rollers of the Swedish wheels $\dot{\varphi_{sw}}$ can be obtained from the restriction model considering the imposed $\dot{\xi}_I$ as presented in Equation (5.8),

$$\dot{\varphi_{sw}} = \begin{bmatrix} 1/r_{sw1} & 0 & 0 \\ 0 & 1/r_{sw2} & 0 \\ 0 & 0 & 1/r_{sw3} \end{bmatrix} \begin{bmatrix} cos(\alpha_1+\beta_1) & sin(\alpha_1+\beta_1) & l_1 sin(\beta_1) \\ cos(\alpha_2+\beta_2) & sin(\alpha_2+\beta_2) & l_2 sin(\beta_2) \\ cos(\alpha_3+\beta_3) & sin(\alpha_3+\beta_3) & l_3 sin(\beta_3) \end{bmatrix} R(\theta)\dot{\xi}_I \tag{5.8}$$

5.3. Characterization and Security Parameters

In the previous chapters there were multiple constants employed for the algorithms that must be determined for its proper functioning. The first group of those constants correspond to the modeling of the robots considered for the algorithms.

5.3.1. Robots Characterization

In Table 5.2 we present the values of the constants related to the hardware characteristics of the robots related to their kinematics, their sensing and their communications. Particularly it is worth to highlight that the communication range considered in the virtual network ρ_2 was chosen arbitrarily within the communication range of the physical network (several tenths of meters) in order to show the effects of the artificial potentials for connectivity during the experiment.

Table 5.2.: Robot Modeling Constants

Constant	Value	Comment
$\bar{v_1}$	0.7 [m/s]	Characteristic velocity of the robot 1 (Kobuki 1)
$\bar{v_2}$	2.7 [m/s]	Characteristic velocity of the robot 2 (Robotino)
$\bar{v_3}$	0.7 [m/s]	Characteristic velocity of the robot 3 (Kobuki 2)
S_r	6 [m]	Sensing range of the robots (LIDAR effective range)
ρ_2	5 [m]	Communication range of the robots (Virtual Network)

5.3.2. Security Parameters and Control Constants

In order to guarantee the integrity of the robots during the mission it is necessary to tune some control parameters to be appropriate for the mission conditions. Particularly, the para-

meter ρ_1 that corresponds to the security radius considered by the robots to avoid collisions must be set based on the minimum distance where the robots can sense objects is 6 centimeters (with a tolerance of 3 centimeters) for the Hokuyo LIDAR sensors employed both in the Kobuki robots as well as in the Robotino. Also, it is important to note that the position of the robot corresponds to its mass centroid and that the robot has a corresponding perimeter of 351.5mm, therefore the security radius must fulfill $\rho_1 > 0{,}20575$ meters (the minimum sensing radius plus the worst case tolerance of the sensor).
On the other hand, it is important to determine the constants required for the control signal calculus. Initially we determine the proportional control constant k_{ti} required for the interest point tracking. Since the maximum speed of the robots is limited, it is required to consider that the control signal must be constrained between 0 [m/s] and the corresponding max speed of the corresponding robot $\bar{v_i}$ (5.9). Considering that the maximum distance between a robot and its interest point is within its line of sight (that is limited by its sensing range) (5.10), the maximum value of the tracking component of the control signal is bounded as presented in Equation (5.11). The resulting values of this calculus are summarized in Table 5.3 rounded down arbitrarily using two decimal positions.

$$|u_{ti}| \leq \bar{v_i} \tag{5.9}$$

$$|u_{ti}| = |K_{ti}(x_d - x_i)| \leq K_{ti}S_r \tag{5.10}$$

$$K_{ti} \leq \frac{\bar{v_i}}{S_r} \tag{5.11}$$

Table 5.3.: Robot Proportional Control Constants

Constant	Value	Comment
K_{t1}	$0{,}23[s^{-1}]$	Proportional constant of the robot 1 (Kobuki 1)
K_{t2}	$0{,}9[s^{-1}]$	Proportional constant of the robot 2 (Robotino)
K_{t3}	$0{,}23[s^{-1}]$	Proportional constant of the robot 3 (Kobuki 2)

It is worth to mention that the control signal is not only composed by the tracking component (which is bounded) but also by the artificial potentials components which are unbounded. Therefore, the control signal may get saturated in the real robots. If this happens it is necessary to process the components of the control signal in a way that allows the preservation of the connectivity. For this purpose, the saturation of the control signal is considered in its norm maintaining the direction of the vector. This way, the effects of the artificial potentials are capable of surpass the effect of the tracking component while the control signal is maintained in its operational rank as presented in Equation (5.12),

Figure 5.3.: Initial deployment of the robots

$$u_{iR} = min(|u_i|, \bar{v}_i)\hat{u}_i \tag{5.12}$$

where u_{iR} is the real control signal applied on the robot, $min(a, b)$ correspond to the minimum value between the scalars a and b and $\hat{u}_i$ is an unitary vector that points to the same direction of the original control signal u_i.

5.4. Exploration

In this case, the robots were deployed in line formation, all of them facing approximately to the same direction in an arbitrary position of the environment as presented in Figure 5.3 and they were let to create the map of it following the interest points determined by the algorithms. Initially, the robots know their position, the position of their partners and an first estimation of the corner where they are deployed determined by a preliminary turn as presented in Figure 5.4.
As mentioned in section 5.1.2 the calculus of the interest points required for the robots is made in the master PC where the interest point is evaluated periodically based on the information sent by the robots. However, the calculus of the control signal must be made into the robot since the obstacles avoidance requires to be made on-line as fast possible. With this in mind, the interest point x_d that is tracked by the proportional controller is

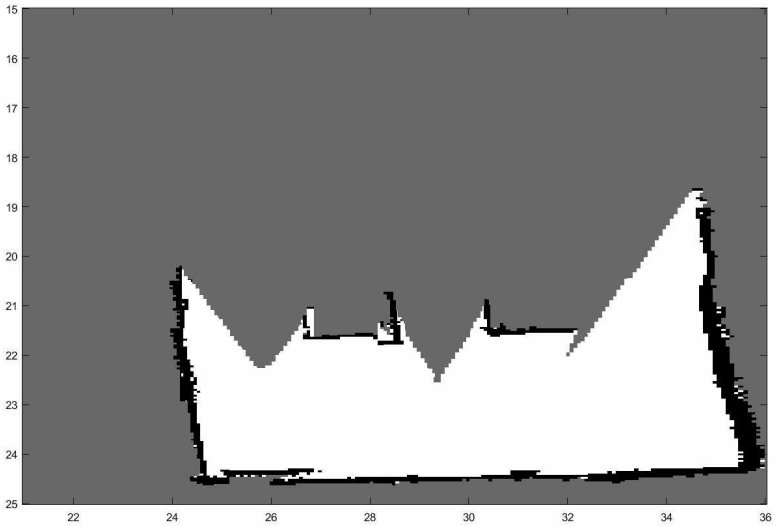


Figure 5.4.: Initial Map known by the robots

evaluated in the master and send to the internal controller of the robot that computes the overall control signal.

In this situation, contrary to the simulation behavior, it is possible that the robot reaches the interest point before that a new reference is set. However, that is not a problem since in this situation, the robot simply will stay in the position while a new reference is set and if the reference correspond to an obstacle or a position close to an obstacle the artificial potentials component of the control signal will maintain it safe.

5.4.1. Simultaneous Localization and Mapping

In order to fulfill the Simultaneous Localization and Mapping task, we employed the package Hector SLAM developed as one of the ROS packages presented in [19]. This package employs a LIDAR based SLAM algorithm for grid mapping that allows the implementation within the robots hardwares with a reduced error and low computational cost. This algorithm does not employ the information provided by the odometry of the robot but only their laser depth sensors (provided by the Hokuyo sensors in this case) in order to estimate the current position of the robot with certain probability distribution allowing the recurrent task of knowing what is the pose of the robot with respect to the map and creating the map based on the current pose of the robot. The algorithm uses the laser scan information to determine the position of obstacles around the robot and when it moves, the algorithm searches for coincidences of the objects seen allowing to generate a good estimate of the map and the robot position.

5.4.2. Map Re-sampling and Merging

The grid occupancy matrix provided by the Hector SLAM package was configured to employ pixels that correspond to squares with a side of 0.05[m] in order to allow an appropriate execution of the algorithm. However, this resolution is much finer than the employed in the simulation scenarios. Therefore, it was necessary to resize the resultant matrix in order to reduce the time required for the evaluation of the flooding distance calculus and the distance to the frontier which are the most demanding functions of the algorithms proposed. This resizing was made assigning to each cell the maximum value within it remembering that the value of the map matrix is set to 0 to the unknown cells, 1 for the cells free to navigate, and 2 for the cells that corresponds to obstacles. This way if there is a single obstacle within the cell, it will not be considered to be navigated. Note that if there is noise in the map it is reduced as the robots approach to it.
On the other hand, it was necessary to integrate the maps provided by the three robots into a single common matrix. This was made in the virtual maps of the robots considering that the robots where initially placed to known distances and orientations to each other and that the positions of the robots should be cleaned from the occupancy map when detected by other robot. With those considerations and trusting in the SLAM algorithm, the origin coordinate frame of each robot within its own map was set as presented in table 5.4 in order to have a consistent reference frame to the general map allowing the maps to be directly overlapped with tolerable errors. Note that this procedure may be made employing coincidence finding algorithms as presented in [35] but this procedures are beyond the scope of the present work.

Table 5.4.: Initial relative positions of the robots

Robot	Position [m,m]	Orientation
Kobuki 1	(2,0)	$\pi/2$
Kobuki 2	(0,0)	$\pi/2$
Robotino	(-2,0)	$\pi/2$

5.4.3. Results

With the previously described setup, the exploration process of the proposed scene was achieved. In Figure 5.5 it is presented the Position of the robots after 20 evaluations and tracking of the interest points. Each time the robot reached the interest point, it stopped and turned around in order to improve the performance of the SLAM. As expected, the robots took separate ways to achieve the exploration in an organized way and the Robotino which is the fastest of the robots has advanced more than the Kobuki robots.
Next, in Figure 5.6 we present the final position of the robots after the exploration procedure.

Figure 5.5.: Position of the robots after 20 evaluations and tracking of the interest points.

Figure 5.6.: Final position of the robots after the exploration procedure

It is worth to highlight that for simplicity reasons, the idle robot behavior (follow the closest neighbor) was not employed and when the Kobuki 1 had no more frontier to explore, it rested in their last Voronoi centroid evaluated while the Kobuki 2 and the Robotino finished the exploration. Also, it is worth to mention that even when the reconfiguration algorithm was evaluated during each interest point evaluation step, the network topology maintained unchanged since the line configuration was the appropriate configuration during the whole process.

Later, in Figure 5.7 it is presented the final virtual map created by the robots team autonomously after filling the unaccessible areas with black employing the flooding algorithm to the raw matrix. As expected, the resultant map has certain errors due to the uncertainties of the SLAM algorithm and the procedure employed for the map merging. However, the shape of the room was properly identified showing the two main obstacles that corresponds to the bases of the two tables in the environment.

Finally, in Figure 5.8 there are presented the robots returned to their initial positions after employing the recently elaborated map following optimized routes determined by the flooding distance algorithm concluding the experiment and demonstrating that the proposed exploration algorithms functioned in a real world scenario.

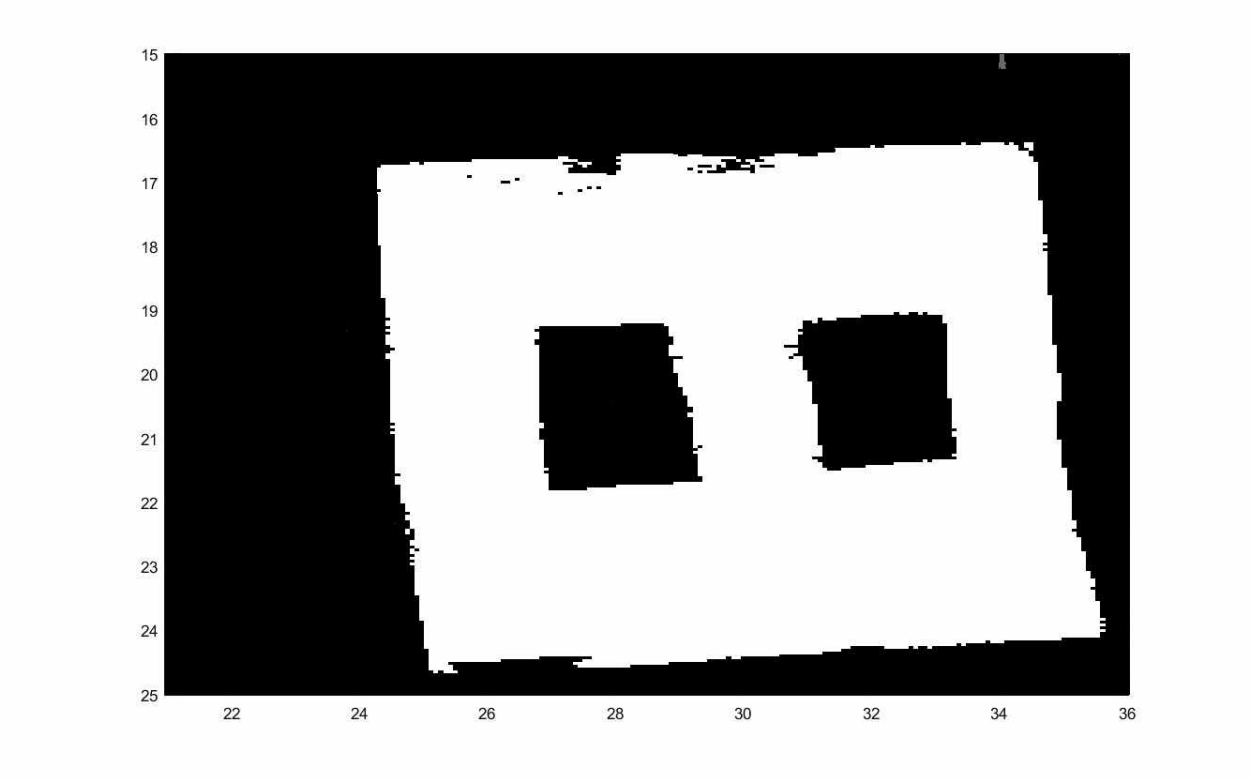


Figure 5.7.: Final map generated by the robots

Figure 5.8.: Comparison between the real map of the environment and the map generated by the robots team

6. Conclusions and Future Work

6.1. Conclusions

The present work introduced an hierarchic control scheme for an heterogeneous robots team, intended to bring useful assistance for human rescue teams in a search and rescue situations from two complementary perspectives, the discrete events based task reallocation and the real time control.

The discrete events based task reallocation algorithm, allows the robot team to determine in an optimized way which task must be made by each robot during the mission development triggering the reallocation order each time the situation changed (when an event happens). The task reallocation procedure employs the Kuhn-Munkres algorithm for multi-agent, multi-task allocations problems employing estimations of the time required for each task.

The real time control perspective was intended to determine control laws to be imposed on the robots team to fulfill certain requirements: maintain the network connectivity for each time instant; avoid collisions against obstacles, other agents and victims; realize an optimized exploration of a non-convex environments and; navigate through the environment in order to accomplish the task assigned to each robot in order to fulfill the whole mission. The collisions avoidance as well as the connectivity maintenance conditions were achieved employing an artificial potentials based approach that guarantees those objectives for each instant of time. On the other hand the exploration algorithm developed, the heterogeneous Discoverage algorithm allows the robot team to determine the interest points to realized the exploration process in an efficient and coordinate manner. This algorithm employs a Voronoi Tessellation of the environment that considers the arrival time of each robot to the point instead of the distance as parameter for the tessellation helping to capture the heterogeneous nature of the robots movement capabilities (The faster robots got bigger areas to be explored while the slower ones got smaller areas).

As a tool for the navigation in non-convex environments, it was implemented the 8 neighbors flooding (8NF) distance calculus algorithm. This algorithm allows to generate estimations of the geodesic distance between any position and every other point within a non-convex space considering only horizontal, vertical and diagonal (at 45 degrees) movements. The resultant matrix of this algorithm allowed not only to realize the tessellation of the non-convex environment required to implement the heterogeneous DisCoverage algorithm, but also allowed to trace back interest point to the line of sight of the robots employing simple gradient descendant algorithms and allowed to construct estimations of the required time

for a robot to reach any position in the map which is fundamental to the estimation of the time cost of each task and consequently to the appropriate reallocation of the tasks.
All the algorithms proposed were extensively evaluated through different simulation scenarios that show that the goals of the project were fully achieved. It is worth to highlight that the algorithms showed to be effective in multiple classes of environments from the simplest cases as in the convex exploration as the most complicated full mission examples where all the desired behaviors were presented at once. In particular it is interesting to observe the behavior of the flooding distance matrix through many of the simulations since even when it is just an approximation to the real geodesic distance, at fist sight its contour plot sees very similar to the real euclidean case. Naturally it has certain error but it is not representative for the mission execution.
The experimental evaluation of the algorithms showed that the algorithms proposed can be employed in real robots with certain considerations. It is important to note that the SLAM is a key element in the functioning of the algorithms and the performance of the algorithms strongly depends on its quality. However with an approach that only employs the information of LIDAR sensors, it was possible to realize an approximate mapping of a non-convex scenario as expected and it was possible to trace back interest points navigating employing this map.

6.1.1. Future work

This work presented multiple tools and algorithms to be employed as a valuable assistance for the human teams in search and rescue situations and there are multiple ways to improve and extend this project. An interesting idea that was not considered in the current project is the possibility of having different types of terrain where some robots may be better than the others to move on or even where only certain robots were capable to move. For instance, an environment with too many debris pieces may be really hard to navigate for a wheeled robot, but on the other hand a quadrotor will not have any problem to move with debris in the floor.
An important issue that may be considered in next related projects is the use of more varied sensor models, since the radial sensitivity is not very common for single robots, so extending the present algorithms to angle limited sensors as single cameras or to sensors that lose accuracy with the distance may be an interesting next step. Within this perspective it can be interesting to evaluate the methods for the identification and classification of the victim since in this work they were assumed to be given, however this is not a trivial task and there are multiple approaches that can be implemented. Particularly it will be interesting (and quite complex) to implement the fusion of multiple robots information to do the classification when the information of the sensors of the individuals may not be enough.
Another gap that can be worked as extension to this work is the requirement of the connectivity of the network for the task reallocations procedure since it limits the overall size of

the environments that can be covered by the robots team. The use of high level behaviors of rendezvous may be a possible option to achieve this, also it is possible to consider multiple robot squads assigned to a bigger area that with certain initial information and certain eventual communication allows the fulfillment of broader missions.
Regarding to the task definitions, reevaluate the tasks models considering the possibility of interconnection between the tasks can be an interesting approach. For instance the debris removal task (that was not considered in this project) can be possibly made by a single or by multiple robots where the second case is very challenging since it requires the evaluation of an non deterministic polynomial hard allocation problem and also the coordination of multiple agents in a single task. Also it is possible to consider the sequence of certain tasks, for instance, the identification task and the evacuation task may be considered as sequential tasks that may be allocated to a single robot as a single task or to be decomposed as presented in this work depending on the situation. Particularly, the model of the evacuation task is one of the most susceptible to be upgraded, since in the approach presented, the evacuation is made individually, but with certain considerations it may be possible for a single robot or for a certain sub set of the robots team to guide the evacuation of multiple persons at once. Finally it is worth to highlight the requirement of model the human behaviors in higher detail since the human robot interaction is very valuable in this kind of situations. Also it can be very useful to consider humans as part of the rescue team since there are multiple tasks that are currently impossible for the robots but no for the humans as the paramedical tasks or the evacuation of injured people, so instead of consider the robots mission as a previous stage for the human team to enter to the area, it can be possible to generate cooperation schemes where the robots realize scout tasks and cover the most dangerous areas while the humans realize their normal work in safer conditions.

A. Annex: Matlab Simulation Codes

For the simulation and implementation of the current thesis there were required multiple codes in different platforms. Those are organized as follows:

A.1. Matlab Codes

Most of the work developed in this work was made employing the software Matlab.

1. Main Code: Wraps the whole simulation behavior defining the scenes, the time evolution of the models and emulates the sensing and movement of the virtual robots.

2. Implement: Wraps the communications and processing employed with the ROS packages employed for the implementation. This code has the same structure of the Main Code, but is limited only to the exploration and return home behaviors.

3. Robot Class: Wraps all the methods and attributes required for the real and virtual robots to work, also includes most of the algorithm calculus.

4. Victim Class: Since the simulation required victims with determined behaviors, this class was created to generate agents that fulfill those requirements and made the victim's related behaviors possible in simulation.

5. Hungarian Task Reallocation and Munkres: Function that receiving as input the current state known by a robot, determines the optimized task allocation employing the Hungarian method presented in [6] (Kuhn-Munkres Algorithm) and formats the result to be propagated to the robots.

6. Line Of Sight: Function that determines the visibility of the points within the environment checking if a straight line that connects the position of interest and the evaluated point is continuous (does not intersect any obstacle).

7. Network Update: Realizes the network reconfiguration virtually adding and deleting the required links and returning the new connected configuration to be set in the communication network.

```
%% Adds all the posible conections
Nr = size(Laplacian,1);
AddCan=zeros(1,Nr);
it = 0;
itMax = 10;
while it<itMax
    it = it + 1;
    Conver = Nr;
    for i=1:1:Nr;
        Dmin = robot{r}.Cr;
        for j = 1:1:Nr
            if(i~=j && Laplacian(i,j) ~= -1)
                Dan = sqrt((robot{i}.Pos(1)-robot{j}.Pos(1))^2+...
                           (robot{i}.Pos(2)-robot{j}.Pos(2))^2 );
                if(Dan < Dmin)
                    AddCan(i) = j;
                    Dmin = Dan;
                end
            end
        end
    end
    for i = 1:1:Nr
        if(AddCan(i)~=0)
            if(Laplacian(i,AddCan(i))~=-1)
                Laplacian(i,i) = Laplacian(i,i)+1;
                Laplacian(AddCan(i),AddCan(i)) = Laplacian(AddCan(i),AddCan(i))+1;
                Laplacian(i,AddCan(i)) = -1;
                Laplacian(AddCan(i),i) = -1;
                % disp(['se agrega la conexión (' num2str(i) ','...
                %        num2str(AddCan(i)) ')'])
            else
                % disp('la conexión candidata ya había sido creada')
                Conver = Conver - 1;
            end
        else
            Conver = Conver - 1;
        end
    end
    if(Conver == 0)
        % disp('convergió')
        break
    end
end
%% Delete the non necesary conections
DelCan=zeros(1,Nr);
it = 0;
itMax = 10;
```

```
48 while it<itMax
49     it = it + 1;
50     Conver = Nr;
51     for i=1:1:Nr;
52         Dmax = 0;
53         for j = 1:1:Nr
54             if(i~=j && Laplacian(i,j) ~= 0)
55                 Dan = sqrt((robot{i}.Pos(1)-robot{j}.Pos(1))^2+...
56                            (robot{i}.Pos(2)-robot{j}.Pos(2))^2 );
57                 if(Dan>Dmax)
58                     DelCan(i) = j;
59                     Dmax = Dan;
60                 end
61             end
62         end
63     end
64     for i = 1:1:Nr
65         Lt = Laplacian;
66         if(Laplacian(i,DelCan(i))~=0)
67             Lt(i,i) = Lt(i,i)-1;
68             Lt(DelCan(i),DelCan(i)) = Lt(DelCan(i),DelCan(i))-1;
69             Lt(i,DelCan(i)) = 0;
70             Lt(DelCan(i),i) = 0;
71             lambdat = sort(eig(Lt));
72             if(lambdat(2)>0.01)
73                 % disp(['grafo conexo ^_^ se borra la conexión (' num2str(i) ','...
74                 %       num2str(DelCan(i)) ')'])
75                 Laplacian=Lt;
76             else
77                 % disp('grafo no conexo u_u , se conservan las conexiones')
78                 Conver = Conver-1;
79             end
80         else
81             % disp('la conexión candidata ya había sido borrada')
82         end
83     end
84     if(Conver == 0)
85         % disp('convergió')
86         break
87     end
88 end
89
```

```
 1 function [ LineOfSight ] = lineOfSight( scene, Mres, pos )
 2 %lineOfSight determies the visibility for each point from a determined
 3 %position in a given scene
 4 %   determies the visibility for each point from a determined
 5 %   position in a given scene using radial rays to determine if the point
 6 %   can or can not be reached linearly from the robot perspective
 7 LineOfSight = zeros(size(scene));
 8 for i=1:1:size(scene,1)
 9     for j=1:1:size(scene,2)
10         if(scene(i,j)~=1)
11             LineOfSight(i,j)=-1;
12         end
13     end
14 end
15 P0 = pos'/Mres;
16 if(P0(1) == 0)
17     P0(1) = 1;
18 end
19 if(P0(2) == 0)
20     P0(2) = 1;
21 end
22 %% line Of Sight Calculus
23 for x = 1:1:size(scene,1)
24     for y = 1:1:size(scene,2)
25         dP = P0 - [x,y];
26         if(norm(dP)==0)
27             % evaluating itself
28             LineOfSight(x,y) = 1;
29         elseif(LineOfSight(x,y) ~= 1)
30             % general evaluation
31             continua = 1;
32             rho = 0:1/(abs(dP(1))+abs(dP(2))):1;
33             cur = [P0(1).*(1-rho) + x*(rho);...
34                    P0(2).*(1-rho) + y*(rho)];
35             for i = 1:1:size(rho,2)
36                 PCur = [max(round(cur(1,i)),1) , max(round(cur(2,i)),1)];
37                 if(scene(PCur(1),PCur(2)) == 1 && continua == 1)
38                     LineOfSight(PCur(1),PCur(2)) = 1;
39                 elseif(continua == 1)
40                     LineOfSight(PCur(1),PCur(2)) = 1;
41                     % continua = 0;
42                     break
43                 else
44                     LineOfSight(PCur(1),PCur(2)) = -1;
45                 end
46             end
47         end
```

```
48     end
49 end
50 end
```

```
 1 classdef Robot
 2     properties
 3         % Características y estado del robot
 4         Pos = [0;0];          % Posición
 5         Vel = [0;0];          % Velocidad actual
 6         V0 = 1;               % Velocidad promedio [m/s]
 7         Sr = 1;               % Radio de Sensado [m]
 8         Cr = inf;             % Radio de Comunicación [m]
 9         ID = 0;               % ID del robot en la misión
10         % Características del mapa
11         MW = 1;               % Ancho del mapa [m]
12         MH = 1;               % Alto del mapa [m]
13         Mres = 1;             % Resolución del mapa [m]
14         Flood = [];           % Distancia por inundación
15         % Exploración del Mapa
16         Map = [];             % Mapa propio de la escena
17         maxVor = inf;         % Máximo tiempo de llegada a un punto posible
18         minTime = [];         % Mínimo tiempo de llegada a un punto
19         LoS = [];             % Determina si la celda es visible o no
20         VAssign = [];         % Asignación de Celda de Voronoi
21         dS = [];              % Frontera de exploración
22         dF = [];              % Distancia a la Frontera
23         phi = [];             % Función de potencial artificial
24         HDis = 0;             % Función de desempeño de la exploración
25         Vc = [0;0];           % Centroide de la Celda de Voronoi Propia
26         Nvf = 0;              % Número de vecinos hasta la frontera
27         % Evacuación de Víctimas
28         VictimID = [];        % Identificador de víctima encontrada
29         VictimTar = 0;        % ID de la víctima objetivo actual
30         VictimPos = [];       % Última posición conocida de las víctimas halladas
31         VictimStatus = [];    % Estado de las víctimas (0-ND, 1-Mov, 2-inMov, 3-↙
Falsa)
32         HomePos = [];         % Posición de Home para evacuación y retorno
33         % Funciónes de potencial de atracción y repulsión
34         PotNet = [0;0];
35         % Parámetros del Controlador
36         Kp = 10;              % Constante proporcional para seguimiento de posición
37         Kar = 10;             % Constante proporcional para atracción/repulsión
38         PerSign = 0;          % Signo para la perturbación
39         PerDir = [0;0];       % Dirección de la perturbación
40         PerCon = 0;           % Contador para mantener Perturbación
41         % Lista de Tareas
42         TaskList = [];        % Lista de Tareas
43         CurrentTask = 1;      % Posición actual en la lista de tareas
44         Tc = [1 1 1 1];       % Capacidad para realizar tareas
45         % Conectivdad
46         NeighborsList = [];   % Lista de Vecinos
```

```
47          Nn = 0;                 % Número de Vecinos
48          Laplacian = [];         % Laplaciano del grafo
49          RecNet = 0;             % Comando de reconfiguración
50      end
51      methods
52      % Constructor
53          function r = Robot(ID, MW, MH, Mres, pos, V0, Sr, Cr, Kp, Tc)
54              % Propiedades y estado del robot
55              r.ID = ID;
56              r.V0 = V0;
57              r.Sr = Sr;
58              r.Cr = Cr;
59              r.Vc = pos;
60              r.Kp = Kp;
61              r.Pos = pos;
62              r.Vel = zeros(2,1);
63              % Mapa
64              r.MW = MW;
65              r.MH = MH;
66              r.Mres = Mres;
67              r.Map = zeros(MW/Mres,MH/Mres);
68              r.Flood = MW/Mres*MH/Mres*ones(size(r.Map));
69              % Exploración
70              r.maxVor = max(MW,MH)*sqrt(2);
71              r.minTime = r.maxVor/r.V0 * ones(size(r.Map));
72              r.VAssign = zeros(size(r.Map));
73              r.dS = zeros(MW/Mres,MH/Mres);
74              r.Nvf = 0;
75              % Tareas
76              r.TaskList = [];
77              r.CurrentTask = 1;
78              r.Tc = Tc;
79          end
80      %% Cálculo de la distancia
81          % distancia por inundación
82          function r = flood(r)
83              r.Flood = r.MW/r.Mres * r.MH/r.Mres * ones(size(r.Map));
84              for i=1:1:size(r.Map,1)
85                  for j=1:1:size(r.Map,2)
86                      if(r.Map(i,j) == 2)
87                      % if(r.Map(i,j) ~= 1)
88                          r.Flood(i,j) = -1;
89                      end
90                  end
91              end
92              Px = max(1,round((r.Pos(1)/r.Mres)));
93              Py = max(1,round((r.Pos(2)/r.Mres)));
```

```
 94              r.Flood(Px,Py) = 1;
 95              dPile = zeros(size(r.Flood));
 96              cicle = 1;
 97              MaxCicles = r.V0/r.Mres;
 98              while(sum(sum(abs(r.Flood-dPile))) ~=0 && cicle < MaxCicles)
 99                  dPile = r.Flood;
100                  cicle = cicle + 1;
101                  %up-down, left-right cicle
102                  for i=1:1:size(r.Map,1)
103                      for j = 1:1:size(r.Map,2)
104                          if(i-1>0)
105                              if(r.Flood(i,j) > 0 && r.Flood(i-1,j) > r.Flood(i,j) &&↙
r.Flood(i-1,j) ~= -1)
106                                  r.Flood(i-1,j) = r.Flood(i,j) + 1;
107                              end
108                          end
109                          if(i+1<=size(r.Flood,1))
110                              if(r.Flood(i,j)>0&&r.Flood(i+1,j)>r.Flood(i,j)&&r.Flood↙
(i+1,j)~=-1)
111                                  r.Flood(i+1,j) = r.Flood(i,j) + 1;
112                              end
113                          end
114                          if(j-1>0)
115                              if(r.Flood(i,j)>0&&r.Flood(i,j-1)>r.Flood(i,j)&&r.Flood↙
(i,j-1)~=-1)
116                                  r.Flood(i,j-1) = r.Flood(i,j) + 1;
117                              end
118                          end
119                          if(j+1<=size(r.Flood,2))
120                              if(r.Flood(i,j)>0&&r.Flood(i,j+1)>r.Flood(i,j)&&r.Flood↙
(i,j+1)~=-1)
121                                  r.Flood(i,j+1) = r.Flood(i,j) + 1;
122                              end
123                          end
124                      end
125                  end
126                  %down-up, right-left cicle
127                  for i=size(r.Map,1):-1:1
128                      for j = size(r.Map,2):-1:1
129                          if(i-1>0)
130                              if(r.Flood(i,j)>0&&r.Flood(i-1,j)>r.Flood(i,j)&&r.Flood↙
(i-1,j)~=-1)
131                                  r.Flood(i-1,j) = r.Flood(i,j) + 1;
132                              end
133                          end
134                          if(i+1<=size(r.Flood,1))
135                              if(r.Flood(i,j)>0&&r.Flood(i+1,j)>r.Flood(i,j)&&r.Flood↙
```

```
(i+1,j)~=-1)
136                                 r.Flood(i+1,j) = r.Flood(i,j) + 1;
137                             end
138                         end
139                         if(j-1>0)
140                             if(r.Flood(i,j)>0&&r.Flood(i,j-1)>r.Flood(i,j)&&r.Flood↙
(i,j-1)~=-1)
141                                 r.Flood(i,j-1) = r.Flood(i,j) + 1;
142                             end
143                         end
144                         if(j+1<=size(r.Flood,2))
145                             if(r.Flood(i,j)>0&&r.Flood(i,j+1)>r.Flood(i,j)&&r.Flood↙
(i,j+1)~=-1)
146                                 r.Flood(i,j+1) = r.Flood(i,j) + 1;
147                             end
148                         end
149                     end
150                 end
151             end
152         end
153         % distancia por inundación euclidea (muy costoso)
154         function r = euclideanFlood(r)
155             r.Flood = r.MW/r.Mres * r.MH/r.Mres * ones(size(r.Map));
156             for i=1:1:size(r.Map,1)
157                 for j=1:1:size(r.Map,2)
158                     if(r.Map(i,j) == 2)
159                     % if(r.Map(i,j) ~= 1)
160                         r.Flood(i,j) = -1;
161                     end
162                 end
163             end
164             Px = max(1,round((r.Pos(1)/r.Mres)));
165             Py = max(1,round((r.Pos(2)/r.Mres)));
166             r.Flood(Px,Py) = 0;
167             dPile = ones(size(r.Flood));
168             cicle = 1;
169             MaxCicles = r.V0/r.Mres;
170             while(sum(sum(abs(r.Flood-dPile))) ~=0 && cicle < MaxCicles)
171                 dPile = r.Flood;
172                 cicle = cicle + 1;
173                 %up-down, left-right cicle
174                 for i=2:1:size(r.Map,1)-1
175                     for j = 2:1:size(r.Map,2)-1
176                         for x = -1:1:1
177                             for y = -1:1:1
178                                 if(r.Flood(i,j) ~= -1)
179                                     canD = r.Flood(i,j) + sqrt(x^2+y^2);
```

```
180                                     if(r.Flood(i+x,y+j) > canD)
181                                         r.Flood(i+x,y+j) = canD;
182                                     end
183                                 end
184                             end
185                         end
186                     end
187                 end
188                 %down-up, right-left cicle
189                 for i=size(r.Map,1)-1:-1:2
190                     for j = size(r.Map,2)-1:-1:2
191                         for x = -1:1:1
192                             for y = -1:1:1
193                                 if(r.Flood(i,j) ~= -1)
194                                     canD = r.Flood(i,j) + sqrt(x^2+y^2);
195                                     if(r.Flood(i+x,y+j) > canD)
196                                         r.Flood(i+x,y+j) = canD;
197                                     end
198                                 end
199                             end
200                         end
201                     end
202                 end
203             end
204         end
205         % distancia Euclidea
206         function r = eucDis(r)
207             r.Flood = zeros(size(r.Map));
208             for i=1:1:size(r.Map,1)
209                 for j=1:1:size(r.Map,2)
210                     r.Flood(i,j) = sqrt((r.Pos(1)-i*r.Mres)^2 + ...
211                                         (r.Pos(2)-j*r.Mres)^2);
212                 end
213             end
214         end
215     %% funciones utilitarias
216         function pV = goTo(r,p0)
217             p = round(p0./r.Mres);
218             counter = 0;
219             while(r.LoS(p(1),p(2)) ~= 1 && counter < 100) % p is not visible
220                 counter = counter + 1;
221                 CurD = r.Flood(p(1),p(2));
222                 for i = -1:1:1
223                     for j = -1:1:1
224                         if(r.Flood(p(1)+i,p(2)+j) < CurD && r.Flood(p(1)+i,p(2)+j)↙
~= -1)
225                             CurD = r.Flood(p(1)+i,p(2)+j);
```

```
                            pCan = [p(1) + i; p(2)+j];
                        end
                    end
                end
                p = pCan;
            end
            pV = p*r.Mres;
        end
    %% Funciones de Exploración
        % Actualización del Mapa Basado en Mapas Recibidos
        function r = updateMap(r,Map)
            for i=1:1:size(Map,1)
                for j=1:1:size(Map,2)
                    if(Map(i,j) ~= 0)
                        r.Map(i,j) = Map(i,j);
                        if(Map(i,j)~=1)
                            r.Flood(i,j)=-1;
                        end
                    end
                end
            end
        end
        % Partición en celdas de Voronoi
        function r = initializeVoronoi(r)
            r.minTime = r.Flood/r.V0;
            r.VAssign = r.ID*ones(size(r.Map));
            for i=1:1:size(r.Map,1)
                for j=1:1:size(r.Map,2)
                    if(r.Map(i,j) == 2)
                        r.VAssign(i,j) = 0;
                        r.minTime(i,j) = 0;
                    end
                end
            end
        end
        function r = updateVoronoi(r,rn)
            for i = 1:1:r.MW/r.Mres %for each x pos
                for j = 1:1:r.MH/r.Mres % for each y pos
                    Trv = rn.Flood(i,j)/rn.V0; % Flooding Distance
                    if(Trv < r.minTime(i,j) && r.VAssign(i,j) ~= 0)
                        r.minTime(i,j) = Trv;
                        r.VAssign(i,j) = rn.ID;
                    end
                end
            end
        end
        % Cálculo de la función de desempeño de exploración
```

```
273         function r = updateVc(r)
274             % Frontier definition (non optimized)
275             r.HDis = 0;
276             r.dS = zeros(size(r.Map));
277             % Center of the map
278             for i = 2:1:r.MW/r.Mres-1
279                 for j = 2:1:r.MH/r.Mres-1
280                     if(r.VAssign(i,j) == r.ID)
281                         if(r.Map(i,j) == 0 && ...
282                                 ((r.Map(i+1,j) ~= 0 && r.Map(i+1,j) ~= 2)||...
283                                  (r.Map(i-1,j) ~= 0 && r.Map(i-1,j) ~= 2)||...
284                                  (r.Map(i,j+1) ~= 0 && r.Map(i,j+1) ~= 2)||...
285                                  (r.Map(i,j-1) ~= 0 && r.Map(i,j-1) ~= 2)))
286                             r.dS(i,j) = 1;
287                         end
288                     end
289                 end
290             end
291             % Left Boarder
292             for i = 1:1:r.MW/r.Mres
293                 if(r.VAssign(i,1) == r.ID)
294                     if(r.Map(i,1) == 0 && r.Map(i,2) ~= 0)
295                         r.dS(i,1) = 1;
296                     end
297                 end
298             end
299             % Right Boarder
300             for i = 1:1:r.MW/r.Mres
301                 if(r.VAssign(i,r.MH/r.Mres) == r.ID)
302                     if(r.Map(i,r.MH/r.Mres) == 0 && r.Map(i,r.MH/r.Mres-1) ~= 0)
303                         r.dS(i,r.MH/r.Mres) = 1;
304                     end
305                 end
306             end
307             % Top Boarder
308             for j = 1:1:r.MH/r.Mres
309                 if(r.VAssign(1,j) == r.ID)
310                     if(r.Map(1,j) == 0 && r.Map(2,j) ~= 0)
311                         r.dS(1,j) = 1;
312                     end
313                 end
314             end
315             % Bottom Boarder
316             for j = 1:1:r.MH/r.Mres
317                 if(r.VAssign(r.MW/r.Mres,j) == r.ID)
318                     if(r.Map(r.MW/r.Mres,j) == 0 && r.Map(r.MW/r.Mres-1,j) ~= 0)
319                         r.dS(r.MW/r.Mres,j) = 1;
```

```
320                     end
321                 end
322             end
323             %
324             if(sum(sum(r.dS)) == 0)
325                 % there is no Frontier at sight
326                 r.Vc = r.Pos;
327                 r.Nvf = inf;
328             else
329                 % distance to Frontier computing (Brute Force Computing)
330                 r.Nvf = 0;
331                 r.dF = zeros(r.MW/r.Mres,r.MH/r.Mres);
332                 for i = 1:1:r.MW/r.Mres % for each point in the VCell
333                     for j = 1:1:r.MH/r.Mres
334                         if ( r.VAssign(i,j) == r.ID && r.Map(i,j) == 1)
335                             % if ( r.VAssign(i,j) == r.ID && r.Map(i,j) ~= 0)
336                             for x = 1:1:r.MW/r.Mres %for each point in the Frontier
337                                 for y = 1:1:r.MH/r.Mres
338                                     if(r.dS(x,y)==1)
339                                         % Drf = abs(i*r.Mres-x*r.Mres) +...
340                                         %       abs(j*r.Mres-y*r.Mres); %↙
Mannhattan
341                                         Drf = sqrt((i*r.Mres-x*r.Mres)^2 +...
342                                             (j*r.Mres-y*r.Mres)^2); % Euclidean
343                                         % Drf = r.Flood(i,j);
344                                         if(r.dF(i,j) > Drf || r.dF(i,j) == 0)
345                                             r.dF(i,j) = Drf;
346                                         end
347                                     end
348                                 end
349                             end
350                         end
351                     end
352                 end
353                 % phi function computing
354                 r.phi = zeros(r.MW/r.Mres,r.MH/r.Mres);
355                 for i = 1:1:r.MW/r.Mres %for each x pos (can be reduced to the↙
sensed area)
356                     for j = 1:1:r.MH/r.Mres % for each y pos (can be reduced to the↙
sensed area)
357                         if(r.VAssign(i,j) == r.ID && r.Map(i,j) ~= 0)
358                             r.phi(i,j) = r.V0*exp(-1/r.V0*r.dF(i,j)^2);
359                             % Drp = abs(i*r.Mres-r.Pos(1)) +...
360                             %       abs(j*r.Mres-r.Pos(2)); % Euclidean
361                             Drp = sqrt((i*r.Mres-r.Pos(1))^2 +...
362                                 (j*r.Mres-r.Pos(2))^2); % Euclidean
363                             r.HDis = r.HDis + Drp^2*r.phi(i,j)*r.Mres^2;
```

```
364                     end
365                 end
366             end
367             % Voronoi Centroid Calculus
368             r.Vc = [0;0];
369             TotalMass = 0;
370             for i = 1:1:r.MW/r.Mres % for each x pos (can be reduced to the
371                 % Voronoi Cell area)
372                 for j = 1:1:r.MH/r.Mres % for each y pos (can be reduced to
373                     % the Voronoi Cell area)
374                     if(r.VAssign(i,j)==r.ID)
375                         r.Vc = r.Vc + r.phi(i,j)*[(i*r.Mres);(j*r.Mres)];
376                         TotalMass = TotalMass + r.phi(i,j);
377                     end
378                 end
379             end
380             %disp(['robot ' num2str(r.ID) ' TM = ' num2str(TotalMass)])
381             r.Vc = r.Vc./TotalMass;
382         end
383     end
384     % Cálculo del centroide de la celda de Voronoi (por optimizar)
385     function r = visibleVc(r)
386         r.Vc = r.goTo(r.Vc);
387     end
388 %% Funciones de detección de Obstáculos
389     % función de detección de obstáculos
390     function r = detectObstacle(r)
391         Obs = inf;
392         % Detect Obstacle in range
393         for i = 1:1:r.MW/r.Mres % for each x pos (can be reduced to the
394                                 % Sensed area)
395             for j = 1:1:r.MH/r.Mres % for each y pos (can be reduced to
396                                     % Sensed area)
397                 DObs = sqrt((r.Pos(1)-i*r.Mres).^2+(r.Pos(2)-j*r.Mres).^2);
398                 if(DObs <= Obs && DObs <= r.Sr && r.Map(i,j) == 2)
399                     Obs = DObs;
400                     ObsPos = [i*r.Mres;j*r.Mres];
401                 end
402             end
403         end
404         % If an obstacle was detected adds
405         % the potential generated by it
406         if(Obs ~= inf && Obs > r.Mres)
407             PotNorm = -(r.V0)/(Obs-r.Mres)^3;
408             direction = (r.Pos-ObsPos)/norm((r.Pos-ObsPos));
409             r.PotNet = r.PotNet + PotNorm*direction;
410         end
```

```
411         end
412     %% Funciones de mantenimiento de la comunicación
413         function r = updateNeighbors(r)
414             for i=1:1:size(r.Laplacian,1)
415                 if(r.Laplacian(r.ID,i)==-1)
416                     r.NeighborsList = [r.NeighborsList i];
417                 end
418             end
419         end
420         function r = conectToRobot(r,rn)
421             if(r.Laplacian(r.ID,rn.ID)==-1)
422                 Drn = sqrt((r.Pos(1) - rn.Pos(1))^2+...
423                            (r.Pos(2) - rn.Pos(2))^2);
424                 if(Drn/r.Cr>0.95)
425                     % disp(['El robot ' num2str(r.ID) ' y el robot ' ...
426                     %     num2str(rn.ID) ' están por desconectarse' ])
427                     r.RecNet = 1;
428                 else
429                     r.RecNet = 0;
430                 end
431                 if(Drn < r.Cr)
432                     PotNorm = r.V0/(r.Cr^2-Drn^2);
433                     r.PotNet = r.PotNet + PotNorm*(r.Pos-rn.Pos);
434                 elseif(r.ID < rn.ID)
435                     disp(['El robot ' num2str(r.ID) ' y el robot ' num2str(rn.ID) '↙
se desconectaron' ])
436                 end
437                 if(Drn > r.Mres)
438                     PotNorm = -r.V0/(Drn-r.Mres)^3;
439                     r.PotNet = r.PotNet + PotNorm*(r.Pos-rn.Pos);
440                 elseif(r.ID < rn.ID)
441                     disp(['El robot ' num2str(r.ID) ' y el robot ' num2str(rn.ID) '↙
se chocaron' ])
442                 end
443             end
444         end
445         function r = conectToVictim(r,victim)
446             Drv = norm(r.Pos-victim.Pos);
447             SrMin = min(r.Sr,victim.Sr);
448             PotDir = (r.Pos-victim.Pos)/Drv;
449             if(Drv<SrMin)
450                 % determines the type of the victim if possible
451                 if(r.Tc(3) == 1 && r.VictimStatus(victim.ID) ~= 4)
452                     r.VictimStatus(victim.ID) = victim.Type;
453                 elseif(r.Tc(3) == 0 && r.VictimStatus(victim.ID) ~= 4)
454                     r.VictimStatus(victim.ID) = 3;
455                 end
```

```
456                 if(r.VictimStatus(victim.ID) == 2)
457                     % if the victim type is false there will not be attention
458                     r.VictimStatus(victim.ID) = 4;
459                 end
460                 % colltion avoiding potential
461                 PotNorm = -1/Drv^2;
462                 % tracking potential if needed for delivery or evacuation
463                 if(victim.Type == 0 && r.VictimTar == victim.ID)
464                     PotNorm = Drv-victim.Cr + r.V0/(SrMin^2-Drv^2)-1/Drv^3;
465                 end
466                 r.PotNet = r.PotNet + PotNorm*PotDir;
467             end
468         end
469         function r = cleanRobotPotential(r)
470             r.PotNet = [0;0];
471         end
472         function r = updateNetwork(r)
473             it = 0;
474             itMax = 10;
475             Na = size(r.Laplacian,1);
476             Conver = Na;
477             while(it<itMax && Conver ~= 0)
478                 it = it+1;
479                 Conver = Na;
480                 for i = 1:1:Na
481                     Lt = r.Laplacian;
482                     % Addition
483                     Dmin = inf;
484                     AddCan = 0;
485                     for j = 1:1:Na
486                         if(i~=j && L(i,j) == 0)
487                             Dan = sqrt((P(1,i)-P(1,j))^2+(P(2,i)-P(2,j))^2);
488                             if(Dan<Dmin)
489                                 AddCan = j;
490                                 Dmin = Dan;
491                             end
492                         end
493                     end
494                     Lt(i,i) = Lt(i,i)+1;
495                     Lt(AddCan,AddCan) = Lt(AddCan,AddCan)+1;
496                     Lt(i,AddCan) = -1;
497                     Lt(AddCan,i) = -1;
498                     % Supression
499                     Dmax = 0;
500                     DelCan = 0;
501                     for j = 1:1:Na
502                         if(i~=j && Lt(i,j) == -1)
```

```
503                             Dan = sqrt((P(1,i)-P(1,j))^2+(P(2,i)-P(2,j))^2);
504                             if(Dan>Dmax)
505                                 DelCan = j;
506                                 Dmax = Dan;
507                             end
508                         end
509                     end
510                     Lt(i,i) = Lt(i,i)-1;
511                     Lt(DelCan,DelCan) = Lt(DelCan,DelCan)-1;
512                     Lt(i,DelCan) = 0;
513                     Lt(DelCan,i) = 0;
514                     % Reeplace the candidates if they are valid and different
515                     if(AddCan~=0 && DelCan~=0 && AddCan ~= DelCan)
516                         % Verifies if the change is valid
517                         lambdat = sort(eig(Lt));
518                         if(lambdat(2)>0.1)
519                             disp(['grafo conexo ^_^ se reemplaza ' ...
520                                 '(' num2str(i) ',' num2str(DelCan) ')'...
521                                 ' por (' num2str(i) ',' num2str(AddCan) ')'])
522                             L=Lt;
523                         else
524                             disp('no se modificaron las conexiones')
525                             Conver = Conver-1;
526                         end
527                     else
528                         disp('no se modificaron las conexiones')
529                         Conver = Conver-1;
530                     end
531                 end
532                 if(Conver == 0)
533                     disp(['Convergió en la iteración ' num2str(it)])
534                     break
535                 end
536             end
537         end % not available yet
538     %% Señal de control
539         % determinación de la señal de control
540         function [r,u] = controlSignal(r,dt)
541             uExp = - r.Kp*(r.Pos - r.Vc);
542             uPot = - r.Kar*r.PotNet;
543             % detects if the robot is stuck
544             if((norm(r.Pos - r.Vc)<r.Mres/sqrt(2) ||...
545                 norm(uExp+uPot)<r.Mres/sqrt(2)      ) &&...
546                 r.PerCon == 0 && r.Tc(2) == 1)
547                 r.PerCon = 1/dt;
548                 r.PerSign = (-1).^randi([1,2],2,1);
549                 r.PerDir = rand(2,1);
```

```
550                 % disp([' Robot ' num2str(r.ID) ' atorado'])
551             end
552             if(r.PerCon>0 && r.CurrentTask ~= 2)
553                 Per = r.PerSign.*round(r.V0/sqrt(2)).*r.PerDir;
554                 uExp = Per;
555                 r.PerCon = r.PerCon - 1;
556                 % disp([' Robot ' num2str(r.ID) ' perturbado Contador = ' num2str↙
(r.PerCon)])
557             end
558             u = uExp + uPot;
559             uNorm = norm(u);
560             % saturates the robot input
561             if(uNorm > r.V0)
562                 u = r.V0 * u/uNorm;
563             end
564             r.Vel = u;
565         end
566     %% Dinámica de Movimiento
567         % Dinámica de Movimiento según entrada
568         function r = move(r, u, dt)
569             newPos = r.Pos + u .* dt;
570             r.Pos = newPos;
571         end
572     end
573 end
```

```
  1 function [assignment,cost] = munkres(costMat)
  2 % MUNKRES   Munkres (Hungarian) Algorithm for Linear Assignment Problem.
  3 %
  4 % [ASSIGN,COST] = munkres(COSTMAT) returns the optimal column indices,
  5 % ASSIGN assigned to each row and the minimum COST based on the assignment
  6 % problem represented by the COSTMAT, where the (i,j)th element represents the cost↙
to assign the jth
  7 % job to the ith worker.
  8 %
  9 % Partial assignment: This code can identify a partial assignment if a full
 10 % assignment is not feasible. For a partial assignment, there are some
 11 % zero elements in the returning assignment vector, which indicate
 12 % un-assigned tasks. The cost returned only contains the cost of partially
 13 % assigned tasks.
 14
 15 % This is vectorized implementation of the algorithm. It is the fastest
 16 % among all Matlab implementations of the algorithm.
 17
 18 % Examples
 19 % Example 1: a 5 x 5 example
 20 %{
 21 [assignment,cost] = munkres(magic(5));
 22 disp(assignment); % 3 2 1 5 4
 23 disp(cost); %15
 24 %}
 25 % Example 2: 400 x 400 random data
 26 %{
 27 n=400;
 28 A=rand(n);
 29 tic
 30 [a,b]=munkres(A);
 31 toc                 % about 2 seconds
 32 %}
 33 % Example 3: rectangular assignment with inf costs
 34 %{
 35 A=rand(10,7);
 36 A(A>0.7)=Inf;
 37 [a,b]=munkres(A);
 38 %}
 39 % Example 4: an example of partial assignment
 40 %{
 41 A = [1 3 Inf; Inf Inf 5; Inf Inf 0.5];
 42 [a,b]=munkres(A)
 43 %}
 44 % a = [1 0 3]
 45 % b = 1.5
 46 % Reference:
```

```
47 % "Munkres' Assignment Algorithm, Modified for Rectangular Matrices",
48 % http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html
49
50 % version 2.3 by Yi Cao at Cranfield University on 11th September 2011
51
52 assignment = zeros(1,size(costMat,1));
53 cost = 0;
54
55 validMat = costMat == costMat & costMat < Inf;
56 bigM = 10^(ceil(log10(sum(costMat(validMat))))+1);
57 costMat(~validMat) = bigM;
58
59 % costMat(costMat~=costMat)=Inf;
60 % validMat = costMat<Inf;
61 validCol = any(validMat,1);
62 validRow = any(validMat,2);
63
64 nRows = sum(validRow);
65 nCols = sum(validCol);
66 n = max(nRows,nCols);
67 if ~n
68     return
69 end
70
71 maxv=10*max(costMat(validMat));
72
73 dMat = zeros(n) + maxv;
74 dMat(1:nRows,1:nCols) = costMat(validRow,validCol);
75
76 %*************************************************
77 % Munkres' Assignment Algorithm starts here
78 %*************************************************
79
80 %**************************************************************************
81 %   STEP 1: Subtract the row minimum from each row.
82 %**************************************************************************
83 minR = min(dMat,[],2);
84 minC = min(bsxfun(@minus, dMat, minR));
85
86 %**************************************************************************
87 %   STEP 2: Find a zero of dMat. If there are no starred zeros in its
88 %           column or row start the zero. Repeat for each zero
89 %**************************************************************************
90 zP = dMat == bsxfun(@plus, minC, minR);
91
92 starZ = zeros(n,1);
93 while any(zP(:))
```

```
 94      [r,c]=find(zP,1);
 95      starZ(r)=c;
 96      zP(r,:)=false;
 97      zP(:,c)=false;
 98  end
 99
100  while 1
101  %************************************************************************
102  %   STEP 3: Cover each column with a starred zero. If all the columns are
103  %           covered then the matching is maximum
104  %************************************************************************
105      if all(starZ>0)
106          break
107      end
108      coverColumn = false(1,n);
109      coverColumn(starZ(starZ>0))=true;
110      coverRow = false(n,1);
111      primeZ = zeros(n,1);
112      [rIdx, cIdx] = find(dMat(~coverRow,~coverColumn)==bsxfun(@plus,minR(~coverRow),↙
minC(~coverColumn)));
113      while 1
114          %************************************************************************
115          %   STEP 4: Find a noncovered zero and prime it.  If there is no starred
116          %           zero in the row containing this primed zero, Go to Step 5.
117          %           Otherwise, cover this row and uncover the column containing
118          %           the starred zero. Continue in this manner until there are no
119          %           uncovered zeros left. Save the smallest uncovered value and
120          %           Go to Step 6.
121          %************************************************************************
122          cR = find(~coverRow);
123          cC = find(~coverColumn);
124          rIdx = cR(rIdx);
125          cIdx = cC(cIdx);
126          Step = 6;
127          while ~isempty(cIdx)
128              uZr = rIdx(1);
129              uZc = cIdx(1);
130              primeZ(uZr) = uZc;
131              stz = starZ(uZr);
132              if ~stz
133                  Step = 5;
134                  break;
135              end
136              coverRow(uZr) = true;
137              coverColumn(stz) = false;
138              z = rIdx==uZr;
139              rIdx(z) = [];
```

```
140             cIdx(z) = [];
141             cR = find(~coverRow);
142             z = dMat(~coverRow,stz) == minR(~coverRow) + minC(stz);
143             rIdx = [rIdx(:);cR(z)];
144             cIdx = [cIdx(:);stz(ones(sum(z),1))];
145         end
146         if Step == 6
147             %↙
**********************************************************************
148             % STEP 6: Add the minimum uncovered value to every element of each↙
covered
149             %         row, and subtract it from every element of each uncovered↙
column.
150             %         Return to Step 4 without altering any stars, primes, or↙
covered lines.
151             %↙
**********************************************************************
152             [minval,rIdx,cIdx]=outerplus(dMat(~coverRow,~coverColumn),minR↙
(~coverRow),minC(~coverColumn));
153             minC(~coverColumn) = minC(~coverColumn) + minval;
154             minR(coverRow) = minR(coverRow) - minval;
155         else
156             break
157         end
158     end
159     %**********************************************************************
160     % STEP 5:
161     %  Construct a series of alternating primed and starred zeros as
162     %  follows:
163     %  Let Z0 represent the uncovered primed zero found in Step 4.
164     %  Let Z1 denote the starred zero in the column of Z0 (if any).
165     %  Let Z2 denote the primed zero in the row of Z1 (there will always
166     %  be one).  Continue until the series terminates at a primed zero
167     %  that has no starred zero in its column.  Unstar each starred
168     %  zero of the series, star each primed zero of the series, erase
169     %  all primes and uncover every line in the matrix.  Return to Step 3.
170     %**********************************************************************
171     rowZ1 = find(starZ==uZc);
172     starZ(uZr)=uZc;
173     while rowZ1>0
174         starZ(rowZ1)=0;
175         uZc = primeZ(rowZ1);
176         uZr = rowZ1;
177         rowZ1 = find(starZ==uZc);
178         starZ(uZr)=uZc;
179     end
180 end
```

```
181
182 % Cost of assignment
183 rowIdx = find(validRow);
184 colIdx = find(validCol);
185 starZ = starZ(1:nRows);
186 vIdx = starZ <= nCols;
187 assignment(rowIdx(vIdx)) = colIdx(starZ(vIdx));
188 pass = assignment(assignment>0);
189 pass(~diag(validMat(assignment>0,pass))) = 0;
190 assignment(assignment>0) = pass;
191 cost = trace(costMat(assignment>0,assignment(assignment>0)));
192
193 function [minval,rIdx,cIdx]=outerplus(M,x,y)
194 ny=size(M,2);
195 minval=inf;
196 for c=1:ny
197     M(:,c)=M(:,c)-(x+y(c));
198     minval = min(minval,min(M(:,c)));
199 end
200 [rIdx,cIdx]=find(M==minval);
201
```

```
 1 function [ DMap ] = displayMap( Map )
 2 %UNTITLED2 Summary of this function goes here
 3 %   Detailed explanation goes here
 4 DMap = zeros(size(Map,1),size(Map,2),3);
 5 for i=1:1:size(Map,1)
 6     for j=1:1:size(Map,2)
 7         if(Map(i,j)==0)
 8             DMap(i,j,:) = [0.5,0.5,0.5];
 9         elseif(Map(i,j) == 1)
10             DMap(i,j,:) = [1,1,1];
11         elseif(Map(i,j) == 2)
12             DMap(i,j,:) = [0,0,0];
13         end
14     end
15 end
16
17 end
18
19
```

```
 1 function [ DVor ,Alpha ] = displayVor( Vor )
 2 %UNTITLED2 Summary of this function goes here
 3 %   Detailed explanation goes here
 4 DVor = zeros(size(Vor,1),size(Vor,2),3);
 5 Alpha = 0.5*ones(size(Vor,1),size(Vor,2));
 6 for i=1:1:size(Vor,1)
 7     for j=1:1:size(Vor,2)
 8         if(Vor(i,j) == 0)
 9             DVor(i,j,:) = [0,0,0];
10             Alpha(i,j) = 1;
11         elseif(Vor(i,j) == 1)
12             DVor(i,j,:) = [1,0,0];
13         elseif(Vor(i,j) == 2)
14             DVor(i,j,:) = [0,0,1];
15         elseif(Vor(i,j) == 3)
16             DVor(i,j,:) = [0,1,0];
17         elseif(Vor(i,j) == 4)
18             DVor(i,j,:) = [0,1,1];
19         elseif(Vor(i,j) == 5)
20             DVor(i,j,:) = [1,1,0];
21         end
22     end
23 end
24 end
25
26
```

```
 1 classdef Victim
 2     properties
 3         % Características y estado de la víctima
 4         Pos = [0;0];          % Posición
 5         Vel = [0;0];          % Velocidad actual
 6         V0 = 1;               % Velocidad promedio [m/s]
 7         Sr = 1;               % Radio de Sensado [m]
 8         Cr = 1;               % Radio de comunicación [m]
 9         Type = 0;             % Tipo de la víctima (0->movil, 1->inmovil, 2->falsa)
10         Saw = 0;              % Bandera que indica si ya ha sido identificada
11         ID = 0;               % ID de la victima en la misión
12         % Funciónes de potencial de atracción y repulsión
13         PotNet = [0;0];
14         Robot = 0;
15         % Parámetros del Controlador
16         Kar = 1;            % Constante proporcional para atracción/repulsión
17     end
18     methods
19     %% Constructor
20         function victim = Victim(ID, Type, pos, V0, Sr, Cr, Kar)
21             % Propiedades y estado del robot
22             victim.ID = ID;
23             victim.Type = Type;
24             victim.V0 = V0;
25             victim.Sr = Sr;
26             victim.Cr = Cr;
27             victim.Kar = Kar;
28             victim.Pos = pos;
29         end
30     %% Evasión de Obstáculos
31     % función de detección de obstáculos
32         function victim = detectObstacle(victim,Map,MW,MH,Mres)
33             Obs = inf;
34             % Detect Obstacle in range
35             for i = 1:1:MW/Mres % for each x pos (can be reduced to the
36                                 % Sensed area)
37                 for j = 1:1:MH/Mres % for each y pos (can be reduced to
38                                     % Sensed area)
39                     DObs = sqrt((victim.Pos(1)-i*Mres).^2+(victim.Pos(2)-j*Mres).↙
^2);
40                     if(DObs <= Obs && DObs <= victim.Sr && Map(i,j) == 2)
41                         Obs = DObs;
42                         ObsPos = [i*Mres;j*Mres];
43                     end
44                 end
45             end
46             % If an obstacle was detected adds
```

```
47              % the potential generated by it
48              if(Obs ~= inf && Obs > Mres)
49                  PotNorm = -1/(Obs-Mres)^3;
50                  direction = (victim.Pos-ObsPos)/norm((victim.Pos-ObsPos));
51                  victim.PotNet = victim.PotNet + PotNorm*direction;
52              end
53          end
54      %% Funciones de mantenimiento de la comunicación
55          function victim = conectToRobot(victim,rn)
56              Drv2 = (rn.Pos(1)-victim.Pos(1))^2+(rn.Pos(2)-victim.Pos(2))^2;
57              PotNorm = -1/Drv2;
58              % PotNorm = 0;
59              PotDir = (victim.Pos-rn.Pos)/sqrt(Drv2);
60              if (rn.VictimTar == victim.ID)
61                  PotNorm = sqrt(Drv2)-victim.Cr;
62                  victim.Robot = rn.ID;
63              end
64              victim.PotNet = victim.PotNet + PotNorm*PotDir;
65          end
66          function victim = conectToVictim(victim,vn)
67              Dvv2 = (vn.Pos(1)-victim.Pos(1))^2+(vn.Pos(2)-victim.Pos(2))^2;
68              if(Dvv2 ~= 0)
69                  PotNorm = -1/Dvv2;
70              else
71                  PotNorm = 0;
72              end
73              victim.PotNet = victim.PotNet + PotNorm*(victim.Pos-vn.Pos)/sqrt(Dvv2);
74          end
75          function victim = cleanVictimPotential(victim)
76              victim.PotNet = [0;0];
77          end
78      %% Señal de control
79          % determinación de la señal de control
80          function [victim,u] = controlSignal(victim)
81              u = - victim.Kar*victim.PotNet;
82              uNorm = norm(u);
83              % saturates the victim input
84              if(uNorm > victim.V0)
85                  u = victim.V0 * u/uNorm;
86              end
87              victim.Vel = u;
88              if(victim.Type ~= 0)
89                  u = 0;
90              end
91          end
92      % Dinámica de Movimiento
93          % Dinámica de Movimiento según entrada
```

```
94          function victim = move(victim, u, dt)
95              newPos = victim.Pos + u .* dt;
96              victim.Pos = newPos;
97          end
98      end
99  end
```

```
 1 function TA = HRT( robot )
 2 %HRT Hungarian Reallocate Tasks
 3 %   Using a centralized algorithm determines the optimal task allocation
 4 %   for the current stage of the mission
 5 %% Memory Preallocation
 6 Nr = size(robot,1);
 7 TA = zeros(2,Nr);
 8 Nv = size(robot{1}.VictimStatus,1);
 9 Crt = zeros(Nv+Nr,Nr); % Capacity of the robot i for executing the task j
10 Tt = Inf*ones(Nv+Nr,Nr); % Time required for the robot i to complete the task j
11 %% Determines the cost asociated to each task (its completion time)
12 for r = 1:1:Nr
13     for v = 1:1:Nv
14         Crt(v,r) = ((robot{r}.Tc(3) == 1 && robot{r}.VictimStatus(v) == 3) || ...
15                     (robot{r}.Tc(4) == 1 && robot{r}.VictimStatus(v) == 1) || ...
16                     (robot{r}.Tc(5) == 1 && robot{r}.VictimStatus(v) == 0)   )...
17                     && robot{r}.VictimID(v) == 1;
18         if(Crt(v,r) == 1)
19             % Evaluates the minimum distance to the victim and compares it
20             % tho its speed
21             Pv = robot{r}.VictimPos(v,:)';
22             PVicR = [round(Pv(1)/robot{1}.Mres),...
23                      round(Pv(2)/robot{1}.Mres)];
24             Vr = robot{r}.V0;
25             Tt(v,r) = robot{r}.Flood(PVicR(1),PVicR(2))/Vr;
26             % if the task is evacuation considers the returning time
27             if(robot{r}.Tc(5) == 1 && robot{r}.VictimStatus(v) == 0)
28                 Pr = robot{r}.HomePos';
29                 PRetR = [round(Pr(1)/robot{1}.Mres),...
30                          round(Pr(2)/robot{1}.Mres)];
31                 Tt(v,r) = 2*Tt(v,r) + robot{r}.Flood(PRetR(1),PRetR(2))/Vr;
32             end
33         end
34     end
35 end
36 %% Determines the estimated time of exploring for finding a victim
37 for r = 1:1:Nr
38     for rn = Nv+1:1:Nv+Nr
39         if(r==rn-Nv)
40             Nmv = sum(robot{r}.VictimID == 0); % Number of Victims not found
41             % Area Based
42             Am = sum(sum(robot{r}.Map == 0))/robot{r}.Mres^2; % Area pending for↙
exploration
43             Vs = pi*robot{r}.Sr*robot{r}.V0; % estimation of the exploration speed↙
[m^2/s]
44             % Fm = sum(sum(robot{r}.dS))/robot{r}.Mres^2; % Frontier "Area" in cell
45             Tt(rn,r) = Am/(Vs*Nmv);
```

```
46              if(isnan(Tt(Nv+1,r)))
47                  Tt(rn,r) = Inf; % there is nothing else to explore
48              end
49          else
50              Tt(rn,r) = Inf; % no robot can explore for other
51          end
52      end
53  end
54  %% Solve the allocation Problem
55  Art = munkres(Tt);
56  %% Changes the allocation's format to mission's format
57  for r = 1:1:Nr
58      task = find(Art == r);
59      if(isempty(task)) % there is no task allocated then it returns home
60          TA(2,r) = 0;
61          TA(1,r) = 2;
62      elseif(task<=Nv)
63          TA(2,r) = task;
64          if(robot{r}.VictimStatus(TA(2,r)) == 0) % evacuate
65              TA(1,r) = 3;
66          elseif(robot{r}.VictimStatus(TA(2,r)) == 1) % deliver Kit
67              TA(1,r) = 4;
68          elseif(robot{r}.VictimStatus(TA(2,r)) == 3) % identify
69              TA(1,r) = 4;
70          end
71      else % explore
72          TA(2,r) = 0;
73          TA(1,r) = 1;
74      end
75  end
76  end
```

```
%% Simulation Parameters
dt = .125; % [s]
Tf = 30; % [s]
%% Scene Definition
MW = 40;%m
MH = 40;%m
Mres = 1; % [m/celda]
% creates the scene
scene = ones(MW/Mres,MH/Mres);
% External Boarders
for i = 1:1:size(scene,1)
    scene(i,1) = 2;
    scene(i,size(scene,2)) = 2;
end
for j = 1:1:size(scene,2)
    scene(1,j) = 2;
    scene(size(scene,1),j) = 2;
end
% Internal Walls
for i = 1:1:round(.8*size(scene,1))
    for j = round(0.475*MH/Mres):1:round(0.525*MH/Mres)
        scene(i,j) = 2;
    end
end
for i = round(0.475*MW/Mres):1:round(0.525*MW/Mres)
    for j = 1:1:.2*size(scene,1)
        scene(i,j) = 2;
    end
end
for i = round(0.475*MW/Mres):1:round(0.525*MW/Mres)
    for j = round(.7*size(scene,1)):1:size(scene,1)
        scene(i,j) = 2;
    end
end
% obstacles randomly allocated
% Nobs = 10;
% Pobs = [0;0];
% for i = 1:1:Nobs
%     Pobs(1) = randi([2,size(scene,1)-1]);
%     Pobs(2) = randi([2,size(scene,2)-1]);
%     for x = -1:1:1
%         for y = -1:1:1
%             scene(Pobs(1)+x,Pobs(2)+y) = 2;
%         end
%     end
% end
%% defines victims randomly allocated (3 = victim non detected)
```

```
 48 Nv = 10;
 49 victim = cell(Nv,1);
 50 Sr = 20;
 51 Cr = 2;
 52 % Type = 0*ones(1,Nv); % Tipo de la víctima (0->movil, 1->inmovil, 2->falsa,
 53                        % 3->desconocido,4->atendida)
 54 Type = randi([0,2],1,Nv); % Tipo de la víctima (0->movil, 1->inmovil, 2->falsa,
 55                           % 3->desconocido,4->atendida)
 56 V0 = 10 * (Type == 0);
 57 Kar = 20;
 58 for i=1:1:Nv
 59     pos = [randi([1,MW/Mres]);randi([0.5*MH/Mres,MH/Mres])];
 60     % reallocates the victims placed into non accesible areas
 61     while(scene(pos(1)-1,pos(2)-1)==2 || scene(pos(1)  ,pos(2)-1)==2 || scene(pos↙
(1)+1,pos(2)-1)==2 || ...
 62           scene(pos(1)-1,pos(2)  )==2 || scene(pos(1)  ,pos(2)  )==2 || scene(pos↙
(1)+1,pos(2)  )==2 || ...
 63           scene(pos(1)-1,pos(2)+1)==2 || scene(pos(1)  ,pos(2)+1)==2 || scene(pos↙
(1)+1,pos(2)+1)==2 )
 64         pos = [randi(MW/Mres);randi([0.5*MH/Mres,MH/Mres])];
 65     end
 66     victim{i} = Victim(i, Type(i), pos*Mres, V0(i), Sr, Cr, Kar);
 67     %scene(pos(1),pos(2)) = 3;
 68 end
 69 h = figure(1);
 70 DScene = displayMap(scene);
 71 image([0 MW], [0 MH], DScene,'CDataMapping','scaled')
 72 hold on
 73 for vic=1:1:Nv
 74     scatter(victim{vic}.Pos(2),victim{vic}.Pos(1),'filled','db')
 75 end
 76 saveas(h,'Mapa.jpg')
 77 saveas(h,'Mapa')
 78 hold off
 79 %% Robots definition
 80 % defines the tester robots
 81 Nr = 5;
 82 robot = cell(Nr,1);
 83 Cr = 30; % entre 30 y 45 m de alcance para Wi-Fi - Dato tomado de
 84          % http://pyme.lavoztx.com/
 85          % qu-distancia-todava-funciona-un-router-inalmbrico-7195.html
 86 % V0a = 8*ones(Nr,1); % Homogeneous
 87 V0a = 1*[9 8 7 6 5 4 3]; % Heterogeneous
 88 Tca = [1 1 1 1 1;... % Task capabilities
 89        1 1 1 1 1;... % column 1 explore and Map
 90        1 1 1 1 1;... % column 2 Identify posible Victim (and mark)
 91        1 1 1 1 1;... % column 3 determine Victim's Type
```

```
 92        1 1 1 1 1];   % column 4 carry First Aid Kit(s)
 93                      % column 5 victim evacuation
 94 for i=1:1:Nr
 95     V0 = V0a(i);
 96     Sr = 9;
 97     Kp = V0;
 98     pos = [2*(i)+4 ;3];
 99     Tc = Tca(i,:);
100     robot{i} = Robot(i, MW, MH, Mres, pos, V0, Sr, Cr, Kp, Tc);
101     robot{i}.TaskList = [1 2 3 4]; % everyone explores and then returns home
102 end
103 % Hand Defined Adjacency for line config.
104 A = [0 1 0 0 0;...
105      1 0 1 0 0;...
106      0 1 0 1 0;...
107      0 0 1 0 1;...
108      0 0 0 1 0];
109 % Hand Defined Adjacency for full config.
110 % A = [0 1 1 1 1;...
111 %      1 0 1 1 1;...
112 %      1 1 0 1 1;...
113 %      1 1 1 0 1;...
114 %      1 1 1 1 0];
115 D = diag(sum(A,2));
116 Laplacian = D-A;
117 Nn = [Laplacian(1,1) Laplacian(2,2) Laplacian(3,3)...
118       Laplacian(4,4) Laplacian(5,5)];
119 HomePos = [4 9; 8 14; 9 9; 12 14; 14 9]';
120 for i=1:1:Nr
121     robot{i}.Laplacian = Laplacian;
122     robot{i}.Nn = Nn(i);
123     robot{i} = robot{i}.updateNeighbors();
124     robot{i}.VictimID = zeros(Nv,1);
125     robot{i}.VictimPos = zeros(Nv,2);
126     robot{i}.VictimStatus = zeros(Nv,1);
127     robot{i}.HomePos = HomePos(:,i);
128 end
129 %% Time evolution
130 PosRobHist = zeros(2,Nr,round(Tf/0.5)+1);
131 PosVicHist = zeros(2,Nv,round(Tf/0.5)+1);
132 for i=1:1:Nr
133     PosRobHist(:,i,1) = robot{i}.Pos;
134 end
135 for i=1:1:Nv
136     PosVicHist(:,i,1) = victim{i}.Pos;
137 end
138 % Create Video writers
```

```
139 MinTimeVideo = VideoWriter('MinTime.avi');
140 MapVideo = VideoWriter('Map.avi');
141 FrontierVideo = VideoWriter('Frontier.avi');
142 VoronoiCenterVideo = VideoWriter('VoronoiCenter.avi');
143 % Edit Frame Rates
144 MinTimeVideo.FrameRate = 10;
145 MapVideo.FrameRate = 10;
146 VoronoiCenterVideo.FrameRate = 10;
147 % open Video Writers
148 open(MinTimeVideo);
149 open(MapVideo);
150 open(FrontierVideo);
151 open(VoronoiCenterVideo);
152 for t = 0:dt:Tf
153     disp(['tiempo = ' num2str(t) ' segundos - ' num2str(100*t/Tf) '% Completado'])
154     %% Robot's Potentials cleaning
155     for r=1:1:Nr
156         robot{r} = robot{r}.cleanRobotPotential();
157     end
158     %% Distance Calculus
159     for r = 1:1:Nr
160         % robot{r} = robot{r}.flood();
161         robot{r} = robot{r}.euclideanFlood(); % cost warning also note there is an↙
error of 8% at 22.5°
162         %robot{r} = robot{r}.eucDis();
163     end
164     %% Map Updating (non optimized)
165     % individual robot scene sensing
166     for r = 1:1:Nr %for each robor
167         % Line Of Sight Calculus
168         LoS = lineOfSight(scene, Mres,robot{r}.Pos);
169         % LoS = ones(size(scene));
170         robot{r}.LoS = LoS;
171         if(robot{r}.Tc == 1) % the robot can map
172             for i = 1:1:MW/Mres %for each x pos (can be reduced to the sensed area)
173                 for j = 1:1:MH/Mres % for each y pos (can be reduced to the sensed↙
area)
174                     Drp = sqrt((robot{r}.Pos(1)-(i-1)*Mres)^2+...
175                         (robot{r}.Pos(2)-(j-1)*Mres)^2); % Euclidean distance
176                     if(Drp < robot{r}.Sr && LoS(i,j) == 1)
177                         robot{r}.Map(i,j) = scene(i,j);
178                     end
179                     % the non accesible areas are set to -1
180                     if(robot{r}.Flood(i,j) == MW/Mres * MH/Mres)
181                         robot{r}.Map(i,j) = scene(i,j);
182                     end
183                 end
```

```
184             end
185         end
186     end
187     % Map broadcasting (non optimized)
188     for i = 1:1:Nr %for each robor
189         for j = 1:1:Nr %for each robor
190             if(i~=j)
191                 robot{i} = robot{i}.updateMap(robot{j}.Map);
192             end
193         end
194     end
195     %% Victims Behavior
196     for i=1:1:Nv
197         victim{i} = victim{i}.cleanVictimPotential();
198         % avoids victim-victim collitions
199 %         for j=1:1:Nv
200 %             if(i~=j)
201 %                 victim{i} = victim{i}.conectToVictim(victim{j});
202 %             end
203 %         end
204         % avoids victim-robot collitions and sets follow signals if
205         % necessary
206         for j=1:1:Nr
207             Drv = sqrt((robot{j}.Pos(1)-victim{i}.Pos(1))^2+...
208                        (robot{j}.Pos(2)-victim{i}.Pos(2))^2);
209             PVicR = [round(victim{i}.Pos(1)/Mres),...
210                      round(victim{i}.Pos(2)/Mres)];
211             if(Drv < victim{i}.Sr  && robot{j}.LoS(PVicR(1),PVicR(2)) == 1)
212                 victim{i} = victim{i}.conectToRobot(robot{j});
213             end
214             if(Drv < robot{j}.Sr && robot{j}.LoS(PVicR(1),PVicR(2)) == 1)
215                 % collition avoiding and tracking if needed
216                 robot{j} = robot{j}.conectToVictim(victim{i});
217                 % Victim information updated and broadcasted
218                 for k = 1:1:Nr
219                     robot{k}.VictimPos(i,:) = victim{i}.Pos;
220                     if(robot{k}.VictimStatus(i) ~= 4)
221                         robot{k}.VictimStatus(i) = robot{j}.VictimStatus(i);
222                     end
223                 end
224                 if(robot{j}.VictimID(i) == 0) % 1st time detected)
225                     % task reallocation
226                     disp(['el robot ' num2str(j) ' detectó a la víctima ' num2str↙
(i)...
227                           ' de tipo ' num2str(robot{j}.VictimStatus(i))]);
228                     for k = 1:1:Nr
229                         robot{k}.VictimStatus(i) = robot{j}.VictimStatus(i); %↙
```

```
Victim Attention Complete
                    end
                    in = HRT(robot);
                    disp('Nueva asignación de Tareas')
                    disp(in)
                    % al the robots are notified
                    for k = 1:1:Nr
                        robot{k}.VictimID(i) = 1;
                        robot{k}.VictimTar = in(2,k);
                        robot{k}.CurrentTask = in(1,k);
                    end
                end
            end
        end
        victim{i} = victim{i}.detectObstacle(scene,MW,MH,Mres);
        [victim{i},u] = victim{i}.controlSignal;
        if(victim{i}.Type == 0 && robot{1}.VictimStatus(i) == 4)
            % already evacuated and continues to corner by him/herself
            u = u - victim{i}.Pos;
        end
        victim{i} = victim{i}.move(u,dt);
    end
    %% Tasks Allocation and Execution
    HDiscoverage = 0;
    VoronoiCentroids = zeros(2,Nr);
    for r = 1:1:Nr
    % Events Identification
        % Check for end of exploration
        if(all(all(robot{r}.Map>0)==1) && robot{r}.Tc(1) == 1)
            if(r == 1 && robot{r}.Tc(1) == 1)
                % task reallocation
                in = HRT(robot);
                disp('Los robots terminaron de explorar ^_^')
                disp('nueva asignación de Tareas')
                disp(in)
                % al the robots are notified
                for k = 1:1:Nr
                    robot{k}.VictimTar = in(2,k);
                    robot{k}.CurrentTask = in(1,k);
                end
            end
            robot{r}.Tc(1) = 0; % there is nothing to be explored
        end
    % Task Execution
    %% Exploración
        if (robot{r}.TaskList(robot{r}.CurrentTask) == 1) % Exploration
            robot{r} = robot{r}.initializeVoronoi();
```

```
276             for rn = 1:1:Nr
277                 if(r~=rn)
278                     robot{r} = robot{r}.updateVoronoi(robot{rn});
279                 end
280             end
281             % Voronoi Cells Centroid Copmutation
282             robot{r} = robot{r}.updateVc();
283             if(robot{r}.Nvf ~= 0) % sets the robot to follow a neighbor
284                                   % if there is no visible frontier
285                 rn = robot{r}.NeighborsList(1);
286                 robot{r}.Vc = robot{rn}.Pos;
287                 robot{r}.Nvf = robot{rn}.Nvf +1;
288                 for i = 2:1:robot{r}.Nn
289                     rn = robot{r}.NeighborsList(i);
290                     if(robot{rn}.Nvf+1 < robot{r}.Nvf)
291                         robot{r}.Vc = robot{rn}.Pos;
292                         robot{r}.Nvf = robot{r}.Nvf+1;
293                     end
294                 end
295             end
296             robot{r} = robot{r}.visibleVc();
297             VoronoiCentroids(:,r) = robot{r}.Vc;
298         end
299         %% Idle State and Home returning
300         if (robot{r}.TaskList(robot{r}.CurrentTask) == 2) % Idle State
301             robot{r}.Vc = robot{r}.Pos;
302             VoronoiCentroids(:,r) = robot{r}.Pos;
303             if(sum(robot{r}.VictimID == 0) == 0) % Return Home
304                 robot{r}.Vc = robot{r}.goTo(robot{r}.HomePos);
305                 VoronoiCentroids(:,r) = robot{r}.HomePos;
306             end
307             dBuff = norm(robot{r}.Pos-robot{r}.HomePos);
308 %             if(dBuff<1)
309 %                 disp(['home alcanzado por el robot ' num2str(r)])
310 %             end
311         end
312         %% Evacuación
313         if(robot{r}.TaskList(robot{r}.CurrentTask) == 3)
314             Vic = victim{robot{r}.VictimTar};
315             dBuff = norm(robot{r}.Pos-Vic.Pos);
316             if(dBuff > 4*Mres)
317                 robot{r}.Vc = robot{r}.goTo(Vic.Pos);
318                 VoronoiCentroids(:,r) = Vic.Pos;
319             else
320                 robot{r}.Vc = robot{r}.goTo(robot{r}.HomePos);
321                 VoronoiCentroids(:,r) = robot{r}.HomePos;
322                 PVicR = [round(Vic.Pos(2)/Mres),...
```

```
323                         round(Vic.Pos(1)/Mres)];
324                 if(norm(robot{r}.Pos-robot{r}.HomePos) < 4*Mres &&...
325                         robot{j}.LoS(PVicR(1),PVicR(2)) == 1)
326                     % task reallocation
327                     disp(['la víctima ' num2str(Vic.ID) ' fue evacuada por el robot↙
'...
328                           num2str(r)]);
329                     for k = 1:1:Nr
330                         robot{k}.VictimStatus(Vic.ID) = 4; % Victim Attention↙
Complete
331                     end
332                     in = HRT(robot);
333                     disp('Nueva asignación de Tareas')
334                     disp(in)
335                     % al the robots are notified
336                     for k = 1:1:Nr
337                         robot{k}.VictimTar = in(2,k);
338                         robot{k}.CurrentTask = in(1,k);
339                     end
340                 end
341             end
342         end
343         %% Entrega de suministros e identificación de Víctimas
344         if (robot{r}.TaskList(robot{r}.CurrentTask) == 4 ||...
345             robot{r}.TaskList(robot{r}.CurrentTask) == 5)
346             Vic = victim{robot{r}.VictimTar};
347             robot{r}.Vc = robot{r}.goTo(Vic.Pos);
348             VoronoiCentroids(:,r) = Vic.Pos;
349             dBuff = norm(robot{r}.Pos-Vic.Pos);
350             PVicR = [round(Vic.Pos(2)/Mres),...
351                      round(Vic.Pos(1)/Mres)];
352             if(dBuff < 4*Mres &&...
353                robot{j}.LoS(PVicR(1),PVicR(2)) == 1)
354                 % task reallocation
355                     if(robot{r}.VictimStatus(Vic.ID) == 3)
356                         disp(['Víctima ' num2str(Vic.ID) ' de tipo ' num2str(Vic.↙
Type)...
357                               ' identificada por el robot ' num2str(r)]);
358                         for k = 1:1:Nr
359                             robot{k}.VictimStatus(Vic.ID) = Vic.Type; % Victim↙
Identified
360                         end
361                         in = HRT(robot);
362                         disp('Nueva asignación de Tareas')
363                         disp(in)
364                         % all the robots are notified
365                         for k = 1:1:Nr
```

```
366                             robot{k}.VictimTar = in(2,k);
367                             robot{k}.CurrentTask = in(1,k);
368                         end
369                     else
370                         disp(['Suministro entregado a la víctima ' num2str(Vic.ID)↙
...
371                               ' por el robot ' num2str(r)]);
372                         for k = 1:1:Nr
373                             robot{k}.VictimStatus(Vic.ID) = 4; % Victim Attention↙
Complete
374                         end
375                         in = HRT(robot);
376                         disp('Nueva asignación de Tareas')
377                         disp(in)
378                         % all the robots are notified
379                         for k = 1:1:Nr
380                             robot{k}.VictimTar = in(2,k);
381                             robot{k}.CurrentTask = in(1,k);
382                         end
383                     end
384             end
385         end
386     end
387     %% Robots Movement
388     for r = 1:1:Nr
389         % AR potentials approach for obstacles evasion and conectivity hold
390         updateNetwork;
391         for rn = 1:1:Nr
392             if(r~=rn)
393                 robot{r} = robot{r}.conectToRobot(robot{rn});
394             end
395         end
396         robot{r} = robot{r}.detectObstacle();
397         % Control Signal
398         [robot{r},u] = robot{r}.controlSignal(dt);
399         robot{r} = robot{r}.move(u,dt);
400         PosRobHist(:,r,round(t/dt)+2) = robot{r}.Pos;
401     end
402     for r=1:1:Nr
403         robot{r}.Laplacian = Laplacian;
404     end
405     for i=1:1:Nv
406         PosVicHist(:,i,round(t/dt)+2) = victim{i}.Pos;
407     end
408     %% Videos
409     % Min Time Display
410     image([0 MW], [0 MH], robot{1}.Flood,'CDataMapping','scaled');
```

```
411     hold on
412     for r=1:1:Nr
413         scatter(robot{r}.Pos(2),robot{r}.Pos(1))
414     end
415     frame = getframe;
416     writeVideo(MinTimeVideo,frame);
417     hold off
418 %     %Fontier
419     image([0 MW], [0 MH],robot{1}.dS + robot{2}.dS + robot{3}.dS + robot{4}.dS +↙
robot{5}.dS,'CDataMapping','scaled')
420     hold on
421     for r=1:1:Nr
422         scatter(robot{r}.Pos(2),robot{r}.Pos(1),'filled')
423     end
424     frame = getframe;
425     writeVideo(FrontierVideo,frame);
426     hold off
427 % Map Display
428     DMap = displayMap(robot{1}.Map);
429     image([0 MW], [0 MH], DMap,'CDataMapping','scaled')
430     hold on
431     for r=1:1:Nr
432         scatter(robot{r}.Pos(2),robot{r}.Pos(1),'filled','r')
433     end
434     sColors = ['b','g','y'];
435     for vic=1:1:Nv
436         sColor = sColors(victim{vic}.Type+1);
437         scatter(victim{vic}.Pos(2),victim{vic}.Pos(1),'filled',sColor)
438     end
439     frame = getframe;
440     writeVideo(MapVideo,frame);
441     hold off
442 % Voronoi Centers
443     [DVor,Alpha] = displayVor(robot{3}.VAssign);
444     im = image([0 MW], [0 MH], DVor,'CDataMapping','scaled');
445     im.AlphaData = Alpha;
446     hold on
447     for r=1:1:Nr
448         scatter(robot{r}.Pos(2),robot{r}.Pos(1),'filled','r')
449         DVel = 0.25*robot{r}.Vel;
450         quiver(robot{r}.Pos(2),robot{r}.Pos(1),DVel(2),DVel(1),'k')
451     end
452     for vic=1:1:Nv
453         sColor = sColors(victim{vic}.Type+1);
454         scatter(victim{vic}.Pos(2),victim{vic}.Pos(1),'filled',sColor)
455         DVel = 0.25*victim{vic}.Vel;
456         quiver(victim{vic}.Pos(2),victim{vic}.Pos(1),DVel(2),DVel(1),'b')
```

```
457     end
458     for r=1:1:Nr
459         scatter(VoronoiCentroids(2,r),VoronoiCentroids(1,r))
460     end
461     % Links
462     for r = 1:1:Nr
463         for rn = 1:1:Nr
464             if(r~=rn && r<rn && Laplacian(r,rn)==-1)
465                 if(norm(robot{r}.Pos-robot{rn}.Pos) <= 0.8 * robot{r}.Cr &&...
466                    norm(robot{r}.Pos-robot{rn}.Pos) >= 4*Mres)
467                     plot([robot{r}.Pos(2),robot{rn}.Pos(2)],...
468                          [robot{r}.Pos(1),robot{rn}.Pos(1)],'b')
469                 else
470                     plot([robot{r}.Pos(2),robot{rn}.Pos(2)],...
471                          [robot{r}.Pos(1),robot{rn}.Pos(1)],'r')
472                 end
473             end
474         end
475     end
476     frame = getframe;
477     writeVideo(VoronoiCenterVideo,frame);
478     hold off
479 end
480 close(MinTimeVideo);
481 close(MapVideo);
482 close(FrontierVideo);
483 close(VoronoiCenterVideo);
484 %% Robots Travel Plot
485 image([0 MW], [0 MH], scene,'CDataMapping','scaled')
486 hold on
487 for ID=1:1:Nr
488     scatter(PosRobHist(2,ID,:),PosRobHist(1,ID,:),'filled')
489 end
490 for ID=1:1:Nv
491     scatter(PosVicHist(2,ID,:),PosVicHist(1,ID,:))
492 end
493 hold off
494 saveas(h,'RobotTravel.jpg')
495 saveas(h,'RobotTravel')
```

References

[1] Amigoni, Francesco ; Visser, Arnoud ; Tsushima, Masatoshi: RoboCup 2012 rescue simulation league winners. En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* Vol. 7500 LNAI, 2013, p. 20–35

[2] Balaguer, Benjamin ; Erinc, Gorkem ; Carpin, Stefano: Combining classification and regression for WiFi localization of heterogeneous robot teams in unknown environments. En: *IEEE International Conference on Intelligent Robots and Systems*, 2012, p. 3496–3503

[3] Bechlioulis, Charalampos P. ; Kyriakopoulos, Kostas J.: Robust model-free formation control with prescribed performance for nonlinear multi-agent systems. En: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015. – ISBN 978–1–4799–6923–4, p. 1268–1273

[4] Boukas, Evangelos ; Kostavelis, Ioannis ; Gasteratos, Antonios ; Sirakoulis, Georgios C.: Robot Guided Crowd Evacuation. En: *IEEE Transactions on Automation Science and Engineering* 12 (2015), Nr. 2, p. 739–751

[5] Bullo, F. ; Cortés, J. ; Martínez, S.: *Distributed Control of Robotic Networks*. First Edition. Princeton University Press, 2009 (Applied Mathematics Series). – Electronically available at http://coordinationbook.info. – ISBN 978–0–691–14195–4

[6] Burgeois, F. ; Lasalle, J.-C.: An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices. En: *Communications of the ACM* 14(12) (1971), p. 802 – 804. – ISSN 00010782

[7] Burke, Jennifer ; Murphy, Robin ; Coovert, Michael ; Riddle, Dawn: Moonlight in Miami: Field Study of Human-Robot Interaction in the Context of an Urban Search and Rescue Disaster Response Training Exercise. En: *Human-Computer Interaction* 19 (2004), Nr. 1, p. 85–116

[8] Carrillo, Henry ; Latif, Yasir ; Rodriguez-Arevalo, Maria L. ; Neira, Jose ; Castellanos, Jose A.: On the monotonicity of optimality criteria during exploration in active SLAM. En: *Proceedings - IEEE International Conference on Robotics and*

Automation Vol. 2015-June, Institute of Electrical and Electronics Engineers Inc., 2015, p. 1476–1483

[9] Cheng, J ; Cheng, Winston ; Nagpal, R: Robust and self-repairing formation control for swarms of mobile agents. En: *AAAI Proceedings of the 20th national conference on Artificial intelligence - Pages 59-64* I (2005), p. 59–64

[10] Chuengsatiansup, Kamol ; Sajjapongse, Krittanai ; Kruapraditsiri, Phartara ; Chanma, Chanin ; Termthanasombat, Nawarat ; Suttasupa, Yuttana ; Sattaratnamai, Sukhum ; Pongkaew, Ekkaluk ; Udsatid, Pakorn ; Hattha, Borirak ; Wibulpolprasert, Panut ; Usaphapanus, Prasit ; Tulyanon, Nantinee ; Wongsaisuwan, Manop ; Wannasuphoprasit, Witaya ; Chongstitvatana, Prabhas: Plasma-RX: Autonomous rescue robots. En: *2008 IEEE International Conference on Robotics and Biomimetics, ROBIO 2008*, 2008, p. 1986–1990

[11] England, University W. *Eurathlon Web Site*. 2017

[12] Eoh, Gyuho ; Choi, Jeong S. ; Lee, Beom H.: Faulty robot rescue by multi-robot cooperation. En: *Robotica* 31 (2013), p. 1239–1249

[13] Ghommam, J. ; Mehrjerdi, H. ; Saad, M.: Robust formation control without velocity measurement of the leader robot. En: *Control Engineering Practice* 21 (2013), Nr. 8, p. 1143–1156

[14] Guanghua, Wang ; Deyi, Li ; Wenyan, Gan ; Peng, Jia: Study on Formation Control of Multi-Robot Systems. En: *2013 Third International Conference on Intelligent System Design and Engineering Applications* (2013), p. 1335–1339. ISBN 978–1–4673–4893–5

[15] Gunn, Tyler ; Anderson, John: Dynamic heterogeneous team formation for robotic urban search and rescue. En: *Journal of Computer and System Sciences* 81 (2015), Nr. 3, p. 553–567

[16] Haumann, D. ; Willert, V. ; Listmann, K. D.: DisCoverage: From coverage to distributed multi-robot exploration. En: *IFAC Proceedings Volumes (IFAC-PapersOnline)* Vol. 4, 2013, p. 328–335

[17] Khalil, Hassan K.: *Nonlinear systems.* Upper Saddle River, New Jersey Prentice Hall, 2002. – ISBN 0130673897

[18] Kitano, Hiroaki ; Tadokoro, Satoshi: RoboCup Rescue: A Grand Challenge for Multiagent Systems. En: *AI Magazine* 22 (2001), Nr. 1, p. 39–52

[19] Kohlbrecher, S. ; Meyer, J. ; von Stryk, O. ; Klingauf, U.: A Flexible and Scalable SLAM System with Full 3D Motion Estimation. En: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)* IEEE, 2011

[20] Koolen, Twan ; Smith, Jesper ; Thomas, Gray ; Bertrand, Sylvain ; Carff, John ; Mertins, Nathan ; Stephen, Douglas ; Abeles, Peter ; Englsberger, Johannes ; McCrory, Stephen ; Van Egmond, Jeff ; Griffioen, Maarten ; Floyd, Marshall ; Kobus, Samantha ; Manor, Nolan ; Alsheikh, Sami ; Duran, Daniel ; Bunch, Larry ; Morphis, Eric ; Colasanto, Luca ; Hoang, Khai Long H. ; Layton, Brooke ; Neuhaus, Peter ; Johnson, Matthew ; Pratt, Jerry: Summary of Team IHMC's virtual robotics challenge entry. En: *IEEE-RAS International Conference on Humanoid Robots* Vol. 2015-Febru, IEEE Computer Society, 2015, p. 307–314

[21] Levi, Nir ; Kovelman, Gregory ; Geynis, Amit ; Sintov, Avishai ; Shapiro, Amir: The DARPA virtual robotics challenge experience. En: *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, 2013

[22] Lin, Z. ; Wang, L. ; Han, Z. ; Fu, M.: Distributed Formation Control of Multi-Agent Systems Using Complex Laplacian. En: *IEEE Transactions on Automatic Control* 59 (2014), Nr. 7, p. 1765–1777. – ISSN 0018–9286

[23] Mueggler, Elias ; Faessler, Matthias ; Fontana, Flavio ; Scaramuzza, Davide: Aerial-guided navigation of a ground robot among movable obstacles. En: *12th IEEE International Symposium on Safety, Security and Rescue Robotics, SSRR 2014 - Symposium Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2015

[24] O'Kane, Jason M.: *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 10 2013. – ISBN 9781492143239

[25] Park, B.S. ; Park, J.B. ; Choi, Y.H.: Robust adaptive formation control and collision avoidance for electrically driven non-holonomic mobile robots. En: *IET Control Theory & Applications* 5 (2011), Nr. 3, p. 514. – ISBN 1751–8644

[26] Pellenz, Johannes ; Jacoff, Adam ; Kimura, Tetsuya ; Mihankhah, Ehsan ; Sheh, Raymond ; Suthakorn, Jackrit: RoboCup rescue robot league. En: *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* Vol. 8992, Springer Verlag, 2015, p. 673–685

[27] Petersen, Karen ; Kleiner, Alexander ; Von Stryk, Oskar: Fast task-sequence allocation for heterogeneous robot teams with a human in the loop. En: *IEEE International Conference on Intelligent Robots and Systems*, 2013, p. 1648–1655

[28] Reich, Joshua ; Sklar, Elizabeth: Robot-sensor networks for search and rescue. En: *In Proc. IEEE Int'l Workshop on Safety, Security and Rescue Robotics* (2006)

[29] Roland Siegwart, Illah R. N. ; Scaramuzza., Davide: *Introduction to autonomous mobile robots*. Massachusetts Institute of Technology, 2011. – ISBN 978–0–262–01535–6

[30] dos Santos, Fernando ; Bazzan, Ana L C.: Towards efficient multiagent task allocation in the RoboCup Rescue: A biologically-inspired approach. En: *Autonomous Agents and Multi-Agent Systems* 22 (2011), Nr. 3, p. 465–486

[31] Schneider, Frank E. ; Wildermuth, Dennis ; Wolf, Hans-Ludwig: ELROB and EURATHLON: Improving search & rescue robotics through real-world robot competitions. En: *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*, IEEE, 2015. – ISBN 978–1–4799–7043–8, p. 118–123

[32] In: Skiena, Steven S.: *Sorting and Searching*. London : Springer London, 2008, p. 103–144. – ISBN 978–1–84800–070–4

[33] Soorki, M. N. ; Talebi, H. A. ; Nikravesh, S. K Y.: A robust leader-obstacle formation control. En: *Proceedings of the IEEE International Conference on Control Applications*, 2011, p. 489–494

[34] Utkin, V. ; Guldner, J. ; Shijun, M. ; Fagerberg, Jan (Ed.) ; Mowery, David C. (Ed.) ; Nelson, Richard R. (Ed.): *Advanced Robotics*. Taylor & Francis, 1999 (Automation and Control Engineering). – 271–301 p.. – ISBN 9780748401161

[35] Velásquez Hernándes, Carlos A.: *Desarrollo de Algoritmo de Mezclado de Mapas por Ocupación de Celdas Aplicado a la Navegación y Exploración Colaborativa de Entornos Internos Desconocidos*, Universidad Nacional de Colombia, Tesis de Grado, 2017

[36] Yanco, Holly A. ; Drury, Jill L.: Rescuing interfaces: A multi-year study of human-robot interaction at the AAAI Robot Rescue Competition. En: *Autonomous Robots* Vol. 22, 2007, p. 333–352

[37] Zavlanos, Michael M. ; Egerstedt, Magnus B. ; Pappas, George J.: Graph-theoretic connectivity control of mobile robot networks. En: *Proceedings of the IEEE* Vol. 99, 2011, p. 1525–1540