mαcrϕ@ufmg

Universidade Federal de Minas Gerais
Programa de Pós-Graduação em Engenharia Elétrica
Research group MACRO - Mechatronics, Control and Robotics

HUMAN-SWARM INTERACTION WITH MULTI-ROBOT COVERAGE CONTROL IN VIRTUAL REALITY ENVIRONMENT

Lucas Coelho Figueiredo

Belo Horizonte, Brazil
2018

Lucas Coelho Figueiredo

HUMAN-SWARM INTERACTION WITH MULTI-ROBOT COVERAGE CONTROL IN VIRTUAL REALITY ENVIRONMENT

Thesis submitted to the Graduate Program in Electrical Engineering of Escola de Engenharia at the Universidade Federal de Minas Gerais, in partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

Advisor: Luciano Cunha de Araújo Pimenta

Belo Horizonte, Brazil
2018

F475h

Figueiredo, Lucas Coelho.
Human-swarm interaction with multi-robot coverage control in virtual reality environment [recurso eletrônico] /Lucas Coelho Figueiredo. - 2018.
1 recurso online (85 f. : il., color.) : pdf.

Orientador: Luciano Cunha de Araújo Pimenta.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f.81-85.

Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia Elétrica - Teses. 2. Robótica - Teses. 3. Realidade Virtual - Teses. 4. Algoritmos – Teses. 5. Robôs – Sistemas de controle – Teses. I. Pimenta, Luciano Cunha de Araújo. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica: Biblioteca Profº Mário Werneck, Escola de Engenharia da UFMG

Este trabalho é dedicado a meus amados pais e meu irmão, que foram fonte de inspiração e me deram forças nas várias vezes que pensei em desistir, e me deram suporte moral, espiritual e emocional nessa longa jornada.

Agradecimentos

Apesar de uma defesa de mestrado ser uma missão solo, eu só pude vencer essa longa batalha graças a todas as pessoas que estivam comigo. Primeiramente, gostaria de agradecer aos meus pais, Glória e Moacir, e meu irmão, Gabriel, pelo suporte incondicional. Sei que para eu viver meus sonhos (e defender meu mestrado foi um deles), vocês tiveram que abdicar de alguns de seus sonhos também, e eu sou extremamente grato por isso.

Ao meu orientador, Luciano Pimenta, por todos os ensinamentos, paciência e oportunidades oferecidas. Me sinto muito feliz de ter tomado coragem de ir pedir um estágio para você quando eu era apenas um aluno de segundo período. Desde então, estivemos sempre trabalhando juntos em vários projetos. Muita coisa legal que aconteceu comigo eu devo à você, incluindo meu trabalho atual, parte do meu intercâmbio e tudo que pude aprender enquanto trabalhamos juntos. E obrigado também por não ter desistido de mim, mesmo nos momentos que me distanciei do meu mestrado. Sou também muito grato ao Ítalo Lelis, um grande gênio disfarçado de bolsista de iniciação científica que me ajudou tanto durante esse trabalho. Saiba que você demostrou uma maturidade e capacidade de aprender muito acima de qualquer expectativa que eu tinha e foi um grande prazer poder trabalhar contigo.

Gostaria de agradecer também aos meus demais familiares, mas especialmente à minha avó Conceição, que mesmo sem entender muito o que eu estudo, fez com que isso estivesse diariamente em suas orações. Aos meus colegas de mestrado, obrigado pelo ombro amigo. Foi sempre bom ter alguém numa situação semelhante a sua para compartilhar as dores e felicidades. Aos meus colegas de trabalho na Hexagon Mining, gostaria de agradecer pela flexibilidade e por propiciar que eu continuasse meu mestrado mesmo depois de contratado. E aos meus amigos, gostaria de pedir perdão por todos os encontros que desmarquei ou não pude ir, com desculpas variando entre "estou lotado de provas" a "estou escrevendo um artigo" ou até mesmo "tenho que fazer um experimento e só posso ir lá no sábado". E obrigado a vocês pela paciência e por entender o quanto isso era importante para mim.

Finalmente, agradeço a CAPES, CNPq e FAPEMIG pelo apoio financeiro para o projeto, e à equipe do VerLab e J do DCC UFMG por gentilmente ceder espaço e equipamento para nossos experimentos.

Gratiam pro gratia

Resumo

Este trabalho apresenta a implementação de uma interface humano-enxame de robôs aplicada à tarefa de cobertura por múltiplos robôs em ambientes não-convexos. Os operadores são capazes de interagir com o grupo de robôs em ambiente de realidade virtual (VR), o que possibilita um profundo grau de controle, imersão e intuitidade. Na aplicação VR, a interação com os múltiplos robôs é abstraída em simples ações, permitindo que o usuário interaja com o grupo de robôs criando e removendo obstáculos, criando funções de distribuição de densidade e alterando a velocidade dos robôs. Simulações foram implementadas usando ROS e Unity e demonstram que os robôs reagem a mudanças no ambiente de simulação impostas pelo agente humano. Experimentos foram efetuados em um grupo de robôs e-puck utilizando um sistema de localização baseado em tags de realidade aumentada. Adicionalmente, um novo algoritmo para cálculo de diagramas de Voronoi foi proposto. Esse algoritmo, chamado de Voro*, permite cálculos locais dos diagramas, possibilitando a aplicação desses algoritmos em grandes números de robôs com um desempenho que aumenta à medida que o número de robôs também aumenta.

Abstract

This work presents an implementation of a human-swarm interaction applied in a coverage control task in non-convex environments. The operators are capable of interacting with the robot swarm in a Virtual Reality (VR) environment, that allows a deep level of control, immersivity and intuitivity. On the VR application, the multi-robot interaction is abstracted in simple actions, allowing the user to interact the robot swarm by creating and removing obstacles, creating density distribution functions and altering the speed of robots. Simulations were implemented using ROS and Unity and demonstrated that robots could react to the changes in the environment introduced by the operators. Experiments were done using a group of e-puck robots using a localization system that uses augmented reality tags. Additionally, a new algorithm for calculation of Voronoi diagrams was proposed. This algorithm, called Voro*, allows calculating local diagrams, enabling this algorithm to be applied in a large scale of robot groups with a better performance.

Contents

List of Figures

Acronyms

VR	Virtual Reality
HRI	Human-Robot Interaction
HSI	Human-Swarm Interaction

Notation

General notation

a	Italic lower case letters denote scalars
$\boldsymbol{a}$	Boldface italic lower case letters denote vectors
$\boldsymbol{A}$	Boldface italic upper case letters denote matrices

Symbols and operators

$\mathbb{N}$	Set of natural numbers
$\mathbb{I}$	Set of integer numbers
$\mathbb{R}$	Set of real numbers
$\mathbf{p}$	A position in a two-dimensional space
$\mathbf{P}$	A set of positions in a two-dimensional space
w	Robot weight
$\mathit{\Omega}$	A given representation of the environment
$\mathbf{q}$	A point in $\mathit{\Omega}$
V	A Voronoi region
$d(\mathbf{q}, \mathbf{p})$	Distance function
$g\mathbf{q}, \mathbf{p})$	Geodesic distance function
W	Weighted Voronoi region
$\mathcal{H}(P, W)$	Deployment functional
M_W	Mass of weighted Voronoi region
$\mathbf{C_{W_i}}$	Centroid of weighted Voronoi region

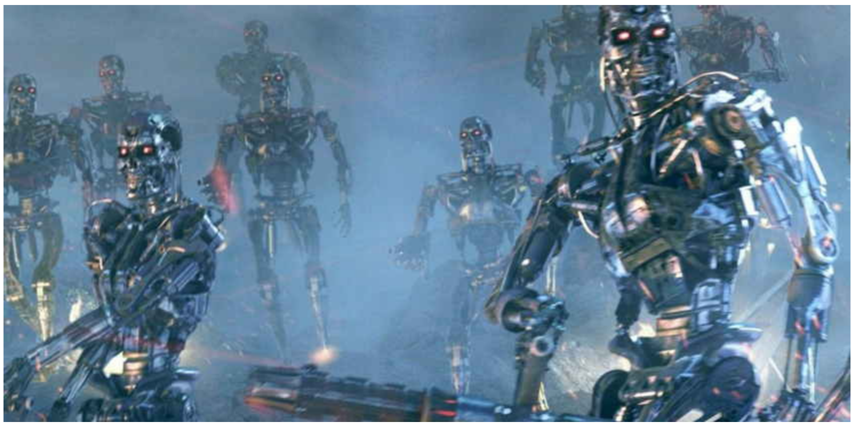
$\mathbf{u}$	Robot control input
$\mathit{\Delta}_i$	Actuation error
$h_i(x)$	Actuation health function
$\boldsymbol{K}_i$	Control gain matrix
$\mathcal{N}_\rangle$	List of robots neighbors of robot i
G	A graph
$\mathcal{V}$	Vertexes of the graph
E	Edges of the graph
$\boldsymbol{\tau}$	Tessellation map

1 Introduction

When talking about multi-robot systems, the first idea that comes to mind of the general public is a world of robotic domination. In some fiction movies that display groups of robots, like "The Matrix" (Wachowski & Wachowski (1999)), "I, Robot" (Proyas (2004)), and "The Terminator" (Cameron (1985)) robots are responsible for an apocalyptic world. Figure 1.1 shows some remarkable scenes of both movies.. Although there is a strong interest in using robots in military (IEEE Spectrum (2004a,b); SEI Insights (2004)), robot groups can be used for doing the "good" too.

(a) Robot Army in "I, Robot".

Source: 20th Century Fox



(b) Robots in "The Terminator" during a combat scene.

Source: Orion Pictures

Figure 1.1: Group of robots as displayed in fiction.

Companies like Bosch are heavily investing in groups of robots for agriculture. (Ruckelshausen et al., 2009) show the *BoniRob* (Figure 1.2), the autonomous robot developed by Deepfield Robotics, a Bosch startup, that can do various tasks in a farm. Lottes et al. (2017), researchers from the University of Bonn have developed a method for identifying weeds in farms using images from low-cost UAVs. Industry reports[1] show that Bosch is already developing robotic swarms for agriculture. Swarm robotics is an approach to

[1]https://spectrum.ieee.org/automaton/robotics/industrial-robots/bosch-deepfield-robotics-weed-control

multi-robot control in which robots are deployed in large numbers (Tan & Zheng (2013)). The main advantages of using a swarm of robots in agriculture is replacing the manual labor, which has become scarce and unreliable, because of the increasing lack of interest of people to work on farms and the constant emigration to urban areas. Also, robots and image processing can be much more accurate and fast to analyze for problems in the farm and provide actionable information for decision making.

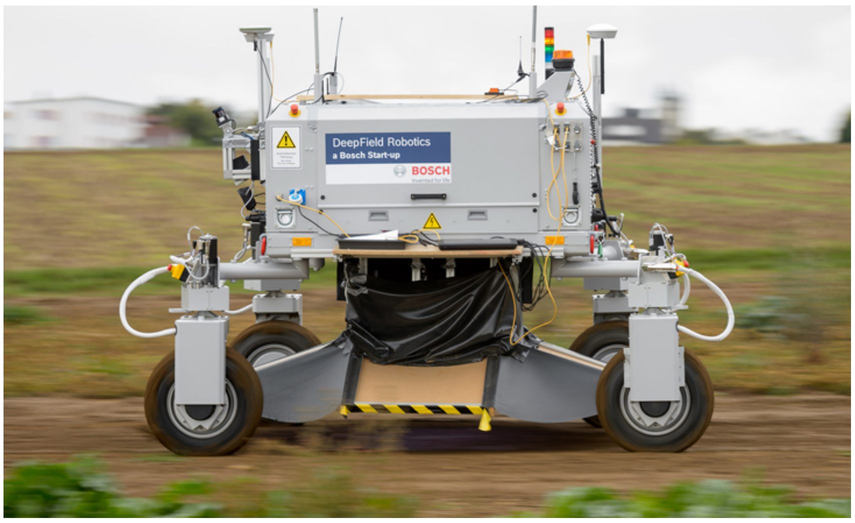


Figure 1.2: The BoniRob, a robot developed by Bosch for agriculture.

Source: Bosch

Another great example of application for robotic swarms is self-driving cars. Some companies like Tesla, Baidu, Waymo, Ford and NuTonomy are doing their first tests of a commercially available autonomous vehicle (CBS Insights (2018)). So far, the implementations of autonomous vehicles are concerned about perceiving the environment and interacting with other vehicles, especially human-driven cars. But in the future, it is expected that vehicles will drive autonomously, interacting with each other as a gigantic robotic swarm. Since the vehicles will be always exchanging information among themselves, they will keep a very short distance between one another and we won't have traffic lights anymore, greatly reducing the commute time and increasing safety. Some papers are already addressing the problem of autonomous vehicles swarms, in applications like path planning (Jann et al. (2017)), traffic optimization (Sharon et al. (2018)) and intersection management (Hausknecht et al. (2011)). Figure 1.3 shows a envision of autonomous cars.

Figure 1.3: Envision of autonomous cars, working as a collaborative robot swarm.

Source: The Motor Report Australia

Although robot swarm applications are becoming more feasible, beneficial and real, there will be undoubtedly some kind of interaction of robots with humans at some level. Humans could give orders, collaborate in the tasks or give information to the group, so real world robotics applications need to be ready to deal with human interaction. The main goal of this work is to implement an interaction between a human and a robotic swarm and analyze how can the human work together and influence the actions of the robot group. Robots can interact with humans in a variety of ways, but in this work, we have implemented such interface using a Virtual Reality (VR) headset. VR is one of the new trends in gaming, telepresence and robotics, and bring great benefits because of its immersivity, intuitivity and level of control. We will discuss VR in a deeper detail on the next chapter.

In some applications for robotic swarms, they must distribute each other in an environment to accomplish a task or sense the environment. For example CNN (2014) shows the Sea Swarm project from MIT (Figure 1.4), that developed a robot that can work in oil spills collecting residue. Instead of using very expensive ships with work crews, they have developed a swarm of robots that can distribute in an area, find out where are the largest concentrations of oil, communicate with other robots and then absorb it, in a much cheaper and faster way than using manned ships.

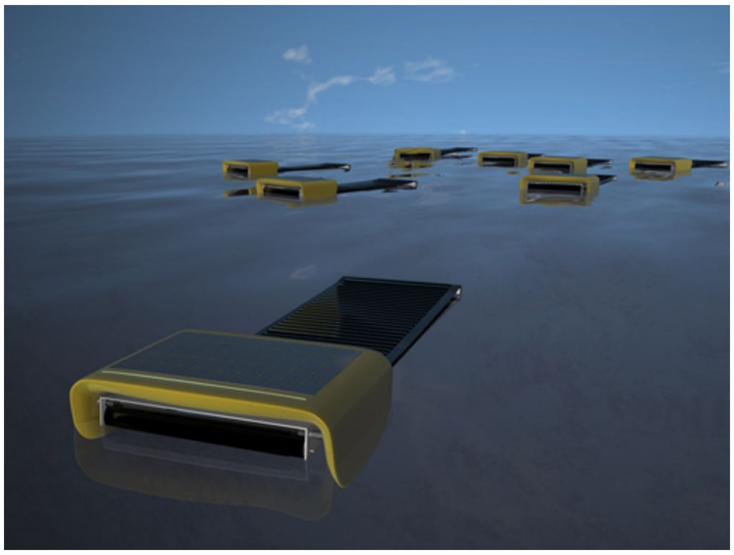


Figure 1.4: Sea Swarm project from MIT, that is building robots for absorbing oil spills from oceans

Source: Seaswarm MIT

But how could these oil cleaning robots distribute themselves in such area? In our work, the task of robot coverage control was the task chosen to the robot group to accomplish under the influence of the human agent. As we will discuss more deeply in the following section, robot coverage is a trending topic in robotics, with applications in search and rescue, sensing, and patrolling. Thus, another focus of this work is to investigate how these robots could optimally place themselves in the environment, considering relevant parameters like the areas that are more important, presence of obstacles and even the performance of each robot. We have also implemented interfaces in which humans can help robots to accomplish the coverage problem, defining the important areas, controlling the obstacles in the area, and increasing the weight of each robot on the coverage of the region. In addition, we will also discuss a new method of robot coverage that is able to locally calculate the control laws for the robots, with great benefits in performance in distributed systems.

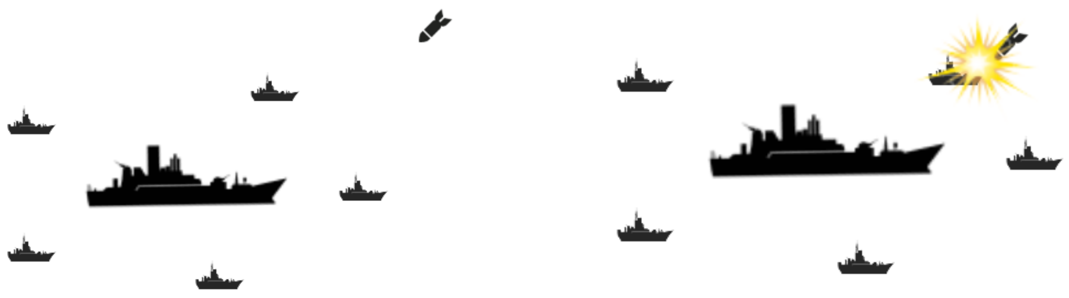
1.1 Multi-robot coverage control problem

Inspired by Cortes et al. (2004), the robot coverage problem can be defined as a special problem in multi-robot systems in which we are concerned in optimally displacing a group of robots in a certain region, based on some criteria, like the importance of an area, sensor range, robot performance and others. As we could see in the previous example, robot coverage has applications in natural disasters but also in military, environmental sensing, rescue, patrolling and others. One great example of robotic swarms in a coverage problem is the swarm of robot boats developed by the U.S. Office of Naval Research, displayed in

Figure 1.5 (IEEE Spectrum (2014)). They have developed a fleet of autonomous robot boats that can perform maneuvers autonomously to defend their mother-ship or surround enemy ships. In a situation in which a surface missile or a torpedo is fired in the direction of the mother-ship, the autonomous robot boats could physically intercept the projectile and take the damage instead. For such task, the robot fleet needs to be positioned around the mother-ship in such a way that it would minimize the required time to reach any point around the mother-ship, to safely intercept any incoming fire. Multi-robot coverage control strategies can address this problem and ensure an optimal displacement for protecting the mother-ship.

Figure 1.5: U.S. Navy fleet of autonomous boats.

Source: U.S. Navy



(a) Mother-ship with robot boats covering the area around her and an incoming projectile. (b) One of the robots intercept the projectile, saving the mother-ship.

Figure 1.6: Example of a fleet of autonomous boats protecting the mother ship using robot coverage algorithms.

In this work, we are specially interested in environments that have obstacles, which we call "non-convex" environments. Moreover, we also consider situations in which the obstacles can change over time, and thus affecting the optimal distribution of robots. This would happen for example in a fleet of ships. If we have some other ships around the mother-ship, they can be treated as obstacles by our coverage strategy.

As we previously discussed, robot-swarms in the real world will always need a interaction with a human at a certain level. A human could provide information that cannot be inferred by the robots, for example, which areas around the mother-ship are more important and thus require a denser coverage. But how could this human interact with the robots while they are accomplishing the task? The study of the relation between humans and robots swarms is a whole field of study, called Human-Swarm Interaction, which we will discuss briefly.

1.2 Human-Swarm Interaction

Before talking about Human-Swarm Interaction, we need first to understand the concept of Human-Robot Interaction (HRI). Put simply, HRI is the field of study of interactions between human and robots. In HRI, usually there is some kind of interface that provides to humans a way by which they can influence the actions of a robot. For example, a joystick, a computer, a smartphone app, or even human gestures. One of the first HRI interfaces accomplished was the "*The Master Slave Manipulator Model NE9*", built in 1960. It was a robotic arm designed for handling nuclear material, and it copied the movement of a human from an interface similar to the manipulator. In other words, by using this interface, the human could control very precisely the desired position of the manipulator actuator that was in another room. Figure 1.7 shows the robot serving a coffee to a woman. The robot is controlled by a person in the second plane of the picture. For a better understanding of the interface, please watch the video in the footnote[2].

In Human-Swarm Interaction, we are concerned about how could a human interact with an unlimited amount of robots. One great example of Human-Swarm Interaction is imagining a big operation of autonomous trucks (Figure 1.8). In this context, the autonomous trucks have to go back and forth from the mine pit, bringing ore, but all of this in an automated and independent way, with the mission of reducing overall time taken for each truck to make a round trip. Considering the existence of a system that monitors the operation of trucks, their positions and status in real time, the vehicles can have full knowledge of the position of other trucks around, thus they can cooperate with themselves to cross intersections without stopping or letting a faster vehicle over take them. But if one vehicle is teleoperated, or operated by a human agent, the robot swarm needs to interact with this human agent, either predicting when they are arriving in an intersection, or always giving them preference, for example. Another example of how human agents can interact with the swarm of autonomous trucks is giving them information about the environment, like changes in the roads, alert them about hazards in the road or manually

[2]The Master Slave Manipulator Model NE9 video: `https://www.youtube.com/watch?v=7YEMMpngZTE`

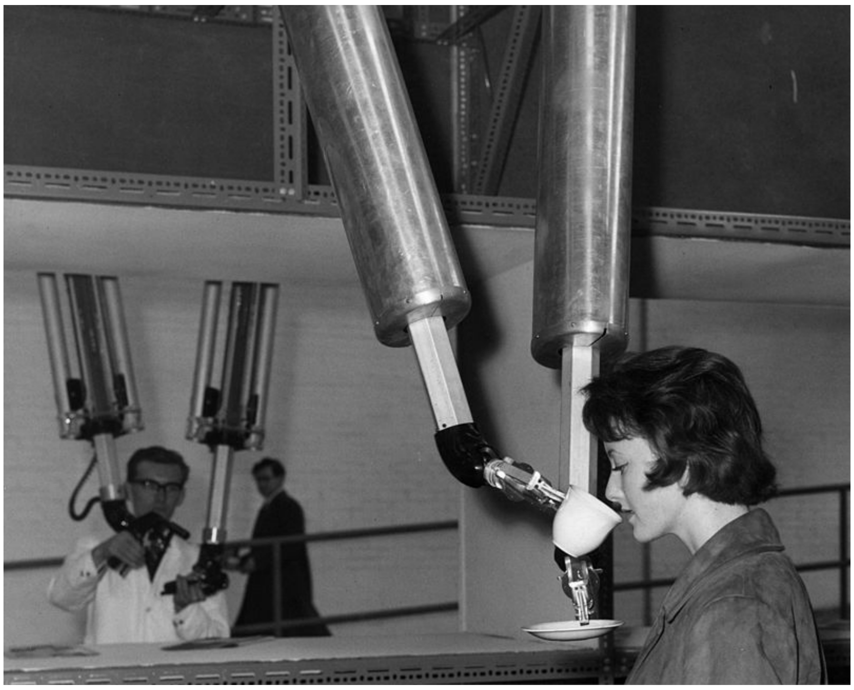


Figure 1.7: The Master Slave Manipulator Model NE9, developed for manipulation of nuclear material and one of the first human-robot interaction interfaces.

Source: British Pathe

changing the trajectory planned for a truck.

Figure 1.8: Mines of the future will be autonomous and the swarm or trucks need to cooperate with human workers.

Source: Suncor Energy

In some HRI applications, an abstraction layer that facilitates the operation of the robot is developed. Some robotic systems may have a lot of inputs (for example, each joint of a robotic manipulator), and controlling them individually is a hard task for the human and takes out the intuitivity and easiness of control. In swarm robotics, since the number of robots can be very large, controlling each robot could be an overwhelming task. Thus, creating an abstraction layer to allow the human to interact with the swarm is beneficial for the operator. In this work, we have implemented a coverage control algorithm that abstracts the control of the robot swarm, allowing the operator to interact with the robots with just a few variables. For the interface, we choose the use of an Virtual Reality headset, because it provides a very intuitive and easy to use interface.

Virtual Reality (VR) is any computer-generated scenario that simulates the experience using video, audio, vibration or any other sense stimulation. So basically, a video game or a simulator can be Virtual Reality. VR is becoming more popular nowadays because of VR headsets, which are screens with lenses that are attached to the head of a person. In the same way headphones increased the immersiveness by approaching the speakers to the ears, VR headsets are increasing the immersiveness approaching screens to the eyes. VR can be interesting when interacting with robots because it is intuitive and provides the human a high degree of immersion and control. VR systems have improved largely recently because of advancements in computer graphics and the economical interest of exploring this technology. Nowadays, we have consumer hardware, like *HTC Vive* and *Oculus Rift* that are both inexpensive, widely supported by the developer community and provide tracking cameras for getting user's motion, which gives an even deeper level of interaction.

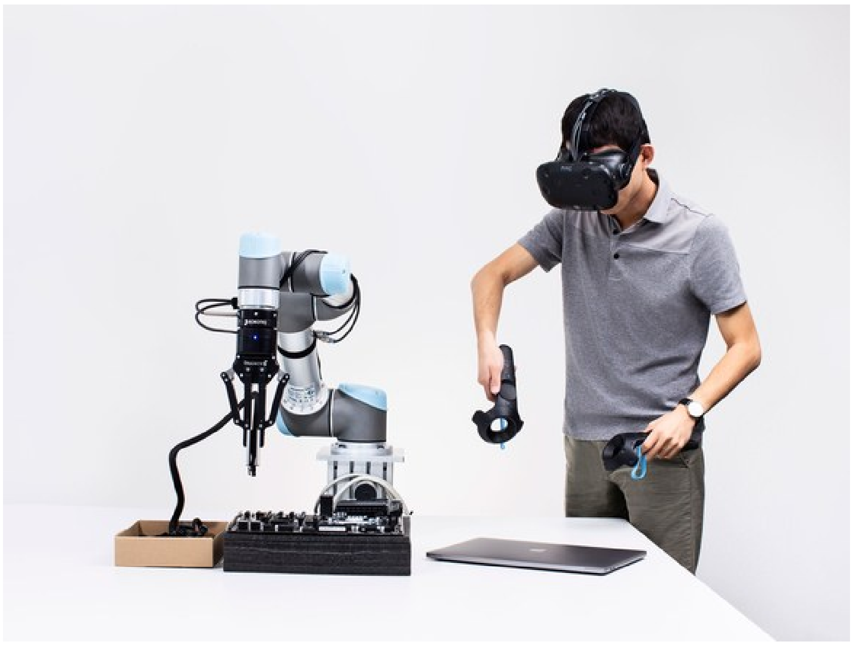


Figure 1.9: A human teleoperating a robot using a Virtual Reality headset.

Source: Quartz Media LCC

1.2.1 Objective and Contributions

This work is divided in two contributions. The first presents the implementation of a human-swarm interaction strategy with the use of a multi-robot coverage framework in non-convex environments. The interface for the interaction of the robot group with a human agent is provided by a virtual reality (VR) environment. On the virtual reality application, the control of the group of robots is abstracted to simple commands, allowing the users to interact with the swarm by creating and removing obstacles, creating density distribution functions and altering the speed of robots. Simulations were implemented using ROS and Unity and demonstrate that robots could react to the changes of the simulation environment imposed by the human agent. Real experiments were implemented using e-puck robots.

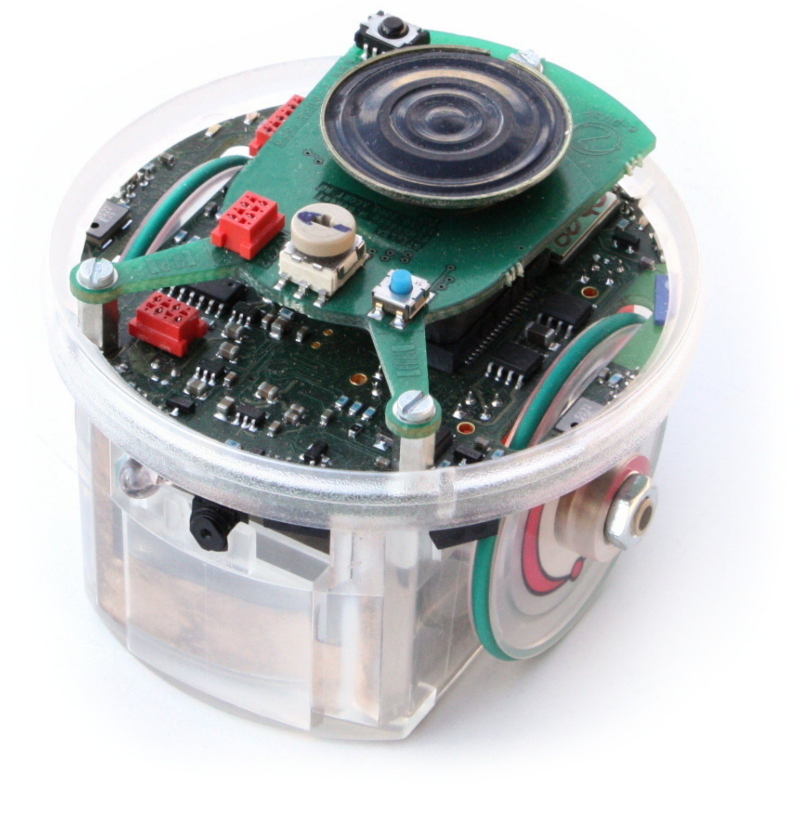


Figure 1.10: The e-puck robot, used on the experiments.

Source: Wikimedia

The second contribution presents a novel algorithm for decentralized calculation of Voronoi tessellations in non-convex environments. The new proposed algorithm allows the robots to calculate their Voronoi diagrams, required to the coverage control algorithm, without having to calculate diagrams of the surrounding robots and also without assuming anything about which robots could be their neighbors. The order of complexity of the new algorithm is also significantly lower, and inversely proportional to the number of robots, without impacts on the diagrams generated. To test the algorithm and compare its performance, simulations were implemented with large groups of robots to show that the average calculation time decreases as the robotic swarm grows, and that the algorithm is able to calculate local Voronoi diagrams without assuming which robots are their neighbors.

2
State of the Art

2.1 Human-Swarm Interaction

As defined by Kolling et al. (2016), robot swarms consist of *"multiple robots that coordinate autonomously via local control laws based on the robot's current state and nearby environment, including neighboring robots"*. Robotic swarms are useful because of their scalability and robustness to local failure of single robots. Since a robot swarm can have *a priori* unlimited number of members, the algorithms and control laws need to be as decentralized as possible, but in some cases, they have to cooperate to accomplish a task.

A number of recent projects have contributed to the development of robot hardware that can implement robot swarms and test coordination algorithms in a real world environment. One of them is the *Swarm-bot* (Mondada et al. (2004)), which is a small but robust platform developed for implementing swarm robotics strategies. One of the robot's features include a small arm and a gripper, which allows the robot to attach to other robots and perform collaborations, like building a bridge using themselves. Another great example is the *Kilobot* (Rubenstein et al. (2012)), which is a small and affordable robot which can be deployed in huge numbers, some works have even implemented a thousand-robot swarm experiment (Rubenstein et al. (2014)). However the *Kilobot* is slow and lacks in essential features such as wireless communication. Other prominent robot is the *e-puck* (Mondada et al. (2009)), which was the robot used in this work. The *e-puck* is a small differential robot with some useful features, like *Bluetooth* communication, encoder wheels, IR sensors and cameras. There are also some recent works that have built robots that are affordable, open source and can be 3D printed, like the *HeRo* robot (Rezeck et al. (2017)), which makes the deployment of a large number of robots accessible and easy.

In real world applications, robots need to be able to interact with humans at some level. Thus, it is important to understand the level of interaction of the operator with the robot swarm. In Kolling et al. (2016), one of the classifications for the HSI applications evaluates the level of attention from the human to work with the robots. In applications that robots are performing independent activities, for example search and rescue (Yamaguchi et al. (2017)), the human agent needs to give to robots individual attention to foresee their actions. These kinds of activities give an *O(n)* order of complexity to the human, since each robot needs the same operator interaction, thus increasing the complexity in a linear manner with the number of robots. A different form of control is when the operator interacts with the robots in a higher level of abstraction, influencing the robot swarm as a whole, or any other task that the number of robots is not relevant for the abstraction. As we will see in the Chapter 3, during the development of our application, we have chosen to implement an abstraction layer that would influence the robot swarm instead of individual robots, thus allowing the human agents to operate with a few variables.

Human-Swarm interactions were accomplished using different interface layers across previous works. Becker et al. (2014) have developed a web page in which users from all over the world could interact to help a simulated group of robots, accomplishing a task to move an object using the arrow keys to control where the robots should go, forming new shapes by combining them, but the user can only tell the direction in which the robotic swarm should move as a whole. So, in this case, the abstraction is the control law provided by the arrow keys, and the interface is the web page receiving control from the keyboard. In the work of Diaz-Mercado et al. (2015), the users could interact with a group of robots in a coverage problem, in a very similar solution that we are approaching in this work. Users could change the importance of the regions, and thus attracting robots to a specific region, by using a tablet to place a Gaussian density distribution function to inform for the robots the most important regions. However, they only consider convex environments and interacting with a tablet screen gives a much shallower level of interaction than VR. Nunnally et al. (2013) also deploy a swarm of robots in a coverage control problem. The method for coverage is based in vector fields, with repulsive forces to repel the robots from one another and cover the area as sparse as possible, and the interaction with the human is based in a haptic feedback interface, which is similar to a robotic arm. However, the human interacts with the robots by changing the repulsive forces around the robots and thus, it characterizes a $O(n)$ effort from the operator, which needs to interact with the robots individually instead of the group as a whole. Some other works have also used haptic interfaces in the context of HSI (Setter et al. (2015)), where a haptic interface was developed to select and control a robot which was the leader of the swarm. Kolling et al. (2013) developed user interfaces for a experiment involving a group of volunteers. One of their experiments involved solving an information foraging problem, in which robots need to explore an environment and report their findings to a base station. Moreover, robots are only able to communicate locally. Human agents collaborated with the robotic swarm by choosing the location of *beacons* using a keyboard and mouse, which influence the behavior of robots. In the work of Gioioso et al. (2014), the interface chosen for collaborating with the robots was the operator own hands. Using gestures, the human could control a swarm of aerial robots allowing them to engage different formations depending on the hand gestures. In the work of Lee et al. (2011), a haptic interface was used to control another formation of aerial vehicles. In one of their examples, they have implemented a master-slave algorithm in which the operator could select one or many robots and teleoperate them, while the other robots would keep a formation.

Because of the interest in autonomous vehicles, there is also a line of research related to human interaction with vehicles. Self-driving cars could be seen as a swarm of robots that drive the humans to their destination. Industry, academia and government have concerns in how to make these vehicles safe, efficient, but also comfortable and that could give the humans some sense of control (Kun (2018)) and could enable humans to reengage to controlling the car (Debernard et al. (2016)), in such way that the human and the self-driving car cooperate on the task of driving.

There are also many other examples on how humans could interact with robot swarms. But, as a conclusion, an important feature that must be provided by all these forms of interaction are the interfaces, that should allow the humans to avoid the activity of individually controlling an overwhelming amount of information, by creating an abstraction layer with a few and intuitive parameters to interact with.

2.2 Multi-robot coverage control problem

Multi-robot coverage has also been a largely studied problem in robotics. As previously explained, robot coverage is concerned in optimally displacing a group of robots in a certain region. An important work in this field was done by Cortes et al. (2004), in which the problem of multi-robot coverage was introduced with a solution based on Voronoi diagrams and the Lloyd's algorithm (Lloyd (1982)). The main idea is to converge to a Centroidal Voronoi Tessellation, *i.e.*, when all robots are located in their respective Voronoi region centroids, the solution is reached. One of the key points of the paper is that in the definition of the coverage algorithms, one consideration that the author makes is that the algorithms should run in a decentralized fashion, so that robots are able to calculate their control laws only depending on the positions of their neighbors and their own.

Many expansions of Cortes et al. (2004) have been proposed. Pimenta et al. (2008) used geodesic distance instead of the standard Euclidean distance to address non-convex environments. The work has also considered that robots could be heterogeneous, thus robots with a higher coverage weight value could cover a larger area. Pierson et al. (2015, 2017) developed an adaptive controller to adjust the weights of the robot based on their sensor quality and actuator performance. For example, in a coverage task in which robots need to collect images, their performance can be evaluated regarding the image quality. In the same way, the actuator performance can be evaluated analyzing the difference between the expected and performed movement of the robot. If there is a difference, it might indicate that the robot is in a difficult terrain or its actuator is not performing well, so its weight is reduced in favor of a better overall coverage.

Some other works have also addressed the problem of coverage in non-convex environments. As we discussed before, the first works used the centroid to guide the robots towards the optimal displacement, and in a non-convex region, the centroid may lie inside an obstacle, which could lead the control law invalid. Pimenta et al. (2008), as we mentioned before, used geodesic distances and a control law based on a gradient, therefore it could also deal with the non-convexity of an environment. Breitenmoser et al. (2010) addressed the problem using the same go-to-centroid controller applied in convex environments, with the addition of a TangentBug local planner (Kamon et al. (1998)) to prevent robots colliding with obstacles, such that if the centroid lies inside an obstacle, the robot would stay as close to the centroid as possible, on the border of the obstacle. Kantaros et al.

(2014) have considered non-convex environments too, but in such way that the robots have a maximum visibility distance, however there is no mention on which method is used for calculating the diagrams.

Bhattacharya et al. (2013) proposes a way for calculating Voronoi diagrams in an approach inspired by Dijkstra's algorithm for graph search (Dijkstra (1959)). Each robot is a seed of the Dijkstra's, and thus the Voronoi partitions are calculated as the wavefronts collide. The robot control laws given to robots do not depend on the centroid of the Voronoi region, but instead the control law points to the direction which maximizes the projection with the solution proposed by Pimenta et al. (2008). This method of Dijkstra's inspired coverage control is the chosen method in this work, based on their efficiency and not dependency on the centroid of the Voronoi region.

2.3 Virtual Reality

Due to the availability of virtual reality consumer electronics nowadays, there is a strong trend in applying virtual reality in the context of robot control and teleoperation. El-Shawa et al. (2017) investigated VR as a HRI for simulation in training, considering environments in which robots and humans work together and collaborate. Volunteers have contributed to their experiments, in which they could see robots doing gestures and should inform in what position they could see the robot commands more clearly, thus validating that the participants could see and interact with the robots in a safe distance even in VR environment. Nigolian et al. (2017) have implemented an interface that allowed a group of self-reconfigurable modular robots (SRMR) to be controlled by an operator. They have used Roombots modules (Spröwitz et al. (2014)), which have SRMR capabilities, and in their experiment, they have used a group of them to form structures. Users interacted with the Roombots using a VR headset and a hand tracking system, in a similar setup than we have used.

Another trending and interesting application of VR in robotics is medical training and recovery. In the works of Klamroth-Marganska et al. (2017); Choi et al. (2018), VR headsets were used to help patients to walk again with the help of robot exoskeletons. In those works, a VR application was developed to create exercises for the patients. They have also used motion tracking in the legs to give the patients visual feedback of the movement in the VR training. Both works have reported that the use of VR increased the strength gain of the patients, helping them in their recovery.

Other recent VR applications involving robotics allow users, for example, to interact with manipulators using VR and a joystick (Guerin et al. (2014)). In that work, a framework for controlling robots using VR is detailed, which enables the operator to grab the joints of a manipulator, create obstacles and plan trajectories. However, no source code of their proposed framework was released. Additionally, they have used second-generation

Oculus headsets, which does not have the motion tracking system that we have used. One of the other few works that have used HSI with VR is the one by Hönig et al. (2015). In fact, they have used mixed reality, which is the combination of VR and Augmented Reality (AR). Together, it means that not only objects are represented virtually, but they also take part in the real world too, modifying the perceiving of the actual environment for the robots and humans. In that work, they have combined simulations in Unity and the real-time experiments in physical world, and in one of their use cases they also created virtual humans that could be followed by drones, enabling safer tests. Du et al. (2016) have used a VR headset to teleoperate a robot for a SLAM mapping and compared the results with a autonomous mapping. Differently of all these previous works described in this section, we focus on head mounted VR, with motion tracking of the user and handheld wireless controllers, which allow a deeper level of immersiveness and control of the group of robots.

2.4 State of the art conclusions

As a conclusion, the three main topics discussed in this work have been heavily studied recently, however a junction of the three — interacting with a swarm of robots using a virtual reality headset, with the robots solving a coverage problem — has not been attempted before. In the next chapters, we will discuss the mathematical background, the implementation and results of the human-swarm interface developed in this work.

3
Methodology

3.1 Problem definition

In this work, for the development, we are considering a group of robots that can communicate with any other of them in a wireless network topology. The agents can randomly communicate with one another. Robots are fully aware of their own global positioning and the position of other robots. Other information, like coverage weights and speed are also exchanged. For the development part, and all the mathematical tools, we will also consider robots that are holonomic, meaning that they can move in any direction at any given time.

The problem that we are aimed to solve is to develop a multi-robot control algorithm that can receive commands from a human operator. Such human operator would influence the coverage of robots and aid them to get external information or reach a better coverage.

On this problem, we will focus more in developing a proof-of-concept instead of a product, showing that integrating a group of robots in a human interface is feasible.

3.2 Voronoi diagram

Imagine that a certain city wants to build new schools and wants to know which areas need the most. One way they could analyze this is by looking which areas are more far to the current schools. To better understand that, they could build a Voronoi diagram. According to Aurenhammer (1991), a Voronoi diagram is a partition of a geometrical space in which generating points are associated to the region of the space closest to it than any other generating points. In the previous example, the current schools would be the generating points of the diagram. Figure 3.1 shows a Voronoi Diagram, where black dots are generating points. In the Figure, take for example the point in the red area in the upper part of the Figure. This red area is closer to the black point inside it than any other black point in the Figure. Voronoi diagrams are very useful to identify and classify the positioning of resources or points of interest. Another example, given the position of police stations in a city, a Voronoi diagram would define the closest police station for each region. Also, we would see areas that are too far from police stations, thus showing areas that need better coverage or more police officers.

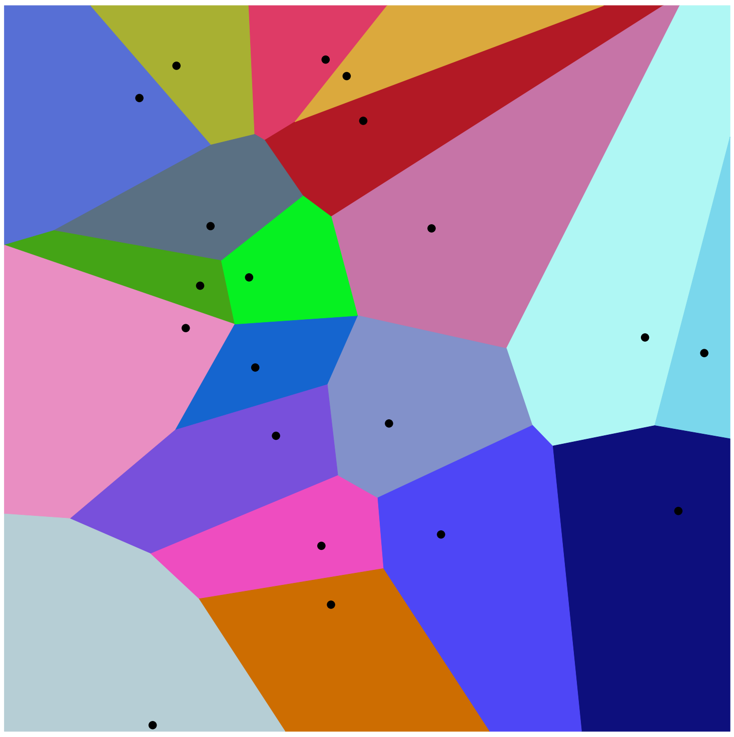


Figure 3.1: A Voronoi diagram.

Source: Wikimedia.

Let's now understand the Voronoi diagram mathematically. Let $\Omega \subset \mathbb{R}^2$ be a given representation of the environment, $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ be the set of configurations $\mathbf{p}_i = [x_i, y_i]$ of n points, where $\mathbf{p}_i \in \Omega$ and $\mathbf{q}$ a point in Ω. Let $\{V_1, \ldots, V_n\}$ be the Voronoi partitions in Ω. Thus, each cell is defined as

$$V_i = \{\mathbf{q} \in \Omega \quad | \quad d(\mathbf{q}, \mathbf{p}_i) \leq d(\mathbf{q}, \mathbf{p}_j), \quad \forall j \neq i\}, \tag{3.1}$$

where d is a function that measures distances between points in Ω and agents.

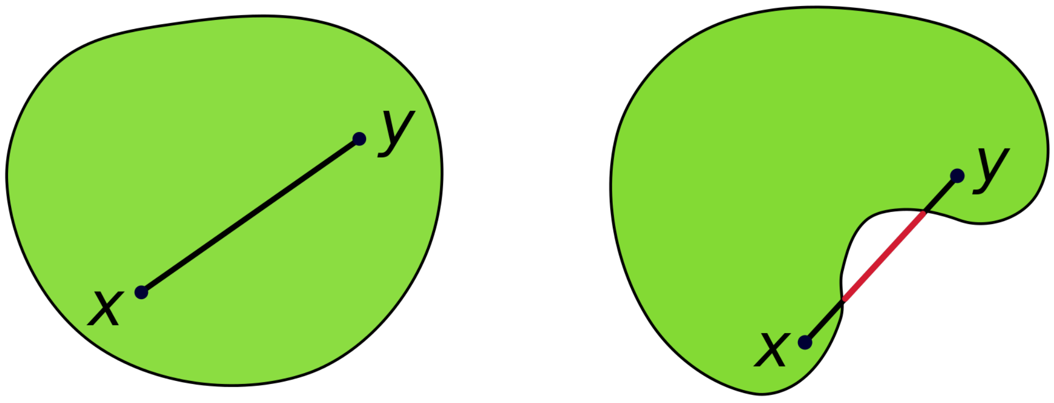
The Voronoi boundary ∂V_i is defined as:

$$\partial V_i = \cup_{j=1}^{n} l_{ij} \cup \{\partial \Omega \cap V_i\}. \tag{3.2}$$

where l_{ij} is the bisector:

$$l_{ij} = \{\mathbf{q} \in \Omega | d(\mathbf{q}, \mathbf{p}_i) = d(\mathbf{q}, \mathbf{p}_j), j \neq i\}. \tag{3.3}$$

The function d can be represented in any way that suits convenience to the problem to be solved. We consider in our problem that the space Ω can be *non-convex* (Pimenta et al. (2008); Bhattacharya et al. (2013)). A convex shape is such that any pair of points inside the set can be connected with a line from one to another, so that all points in this line are inside this shape. A non-convex shape is any shape where this property does not hold. Figure 3.2 shows a convex and a non-convex shape.



(a) A convex shape (b) A non-convex shape

Figure 3.2: Comparison between a convex and non-convex shapes.

Source: Wikimedia

3.3 Geodesic distance

The term *geodesic distance* refers to the shortest distance between two points considering the shape of the environment. Opposing to the *Euclidean distance*, the geodesic distance may not be the length of a straight line and depends on the geometry of the environment, including obstacles, curvature, and other factors. In geography, the term geodesic distance refers to the distance on the surface of earth. In graph theory, it is the shortest distance between two vertexes, considering the sum of costs of the edges connecting them. In the scope of this work, since we deal with environments with obstacles, it is crucial to deal with geodesic distances. As we will see, considering geodesic distance is not as simple as Euclidean distance, because while calculating the straight line distance between two points is as straightforward as using the Pythagorean theorem, computing the geodesic distance in our context requires the use of graph search algorithms. Figure 3.3 shows a comparison between Euclidean and geodesic distance, in a environment such that the possible routes are the ones over the black lines.

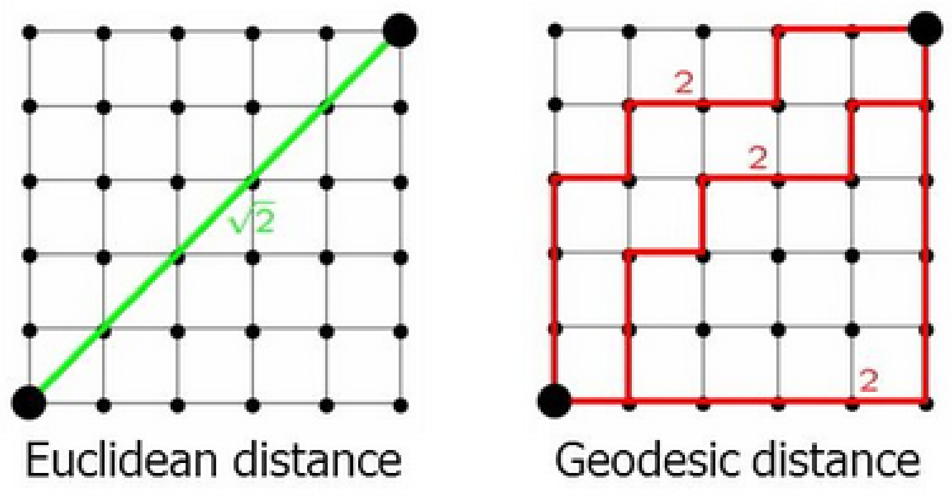


Figure 3.3: Comparison between the geodesic and Euclidean distance.

3.4 Multi-robot coverage in convex environments

Our abstraction to enable a human operator to interact with the robot swarm is based on coverage control, in which the goal is to optimally displace a group of heterogeneous robots in a given non-convex area. Inspired by the work of Cortes et al. (2004), we define an optimization-based method to reach an optimal displacement of the robot group. We will first describe the convex environment situation, since it is simpler and will give us mathematical background to understand the coverage control problem. In Section 3.4.4, we will talk about the necessary considerations to deal with non-convex environments.

Following Pimenta et al. (2008); Pierson et al. (2015, 2017), we also consider that the robots may vary in performance, thus they are heterogeneous robots, which have different weights $\mathbf{w} = \{w_1, \dots, w_n\}$. As we are dealing with robots with weights, we can define the Voronoi distance function d as:

$$d(\mathbf{q}, \mathbf{p}_i) = \|\mathbf{q} - \mathbf{p}_i\|^2 - w_i^2. \tag{3.4}$$

where $\|\mathbf{q}, \mathbf{p}_i\|$ represents the Euclidean distance between $\mathbf{q}$ and $\mathbf{p_i}$.

Let $\{W_1, \dots, W_n\}$ be the geodesic weighted Voronoi partition of Ω, with each cell defined as:

$$W_i = \{\mathbf{q} \in \Omega \mid \|\mathbf{q} - \mathbf{p}_i\|^2 - w_i^2 \le \|\mathbf{q} - \mathbf{p}_j\|^2 - w_j^2, \forall j \neq i\}. \tag{3.5}$$

As a measure of the system performance we define the *deployment functional*:

$$\mathcal{H}(P, W) = \sum_{i=1}^{n} \int_{W_i} [\|\mathbf{q} - \mathbf{p}_i\|^2 - w_i^2]\phi(\mathbf{q})d\mathbf{q}\,, \tag{3.6}$$

where the function $\phi : \Omega \to \mathbb{R}_+$ is a density distribution function which defines a weight for each point in Ω. Therefore, points with greater weight values should be better covered by the robots than points with smaller values.

We also define the mass of the Voronoi region M_{W_i} and the centroid of the Voronoi region, C_{W_i} as:

$$M_{W_i} = \int_{W_i} \phi(\mathbf{q}) d\mathbf{q} \quad \text{and} \quad \mathbf{C_{W_i}} = \frac{1}{M_{W_i}} \int_{W_i} \mathbf{q}\phi(\mathbf{q}) d\mathbf{q}\,. \tag{3.7}$$

In a coverage problem, our goal is to optimally displace the robots in the environment, which in an optimization problem perspective, translates to minimize the value of $\mathcal{H}(P, W)$. An interesting result is that the critical points of $\mathcal{H}(P, W)$ happen when robots are located in the centroids of their respective Voronoi regions, or $\mathbf{p_i} = \mathbf{C_{W_i}}$, as detailed by Pimenta et al. (2008). Thus, one way to minimize the value of $\mathcal{H}(P, W)$ and reach a critical value would be to make the robots follow the gradient of the deployment functional, which can be represented as:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{p_i}} = -2 \int_{W_i} (\mathbf{q} - \mathbf{p_i}) \phi(\mathbf{q}) d\mathbf{q}\,, \tag{3.8}$$

but using Equation (3.7), we can simplify the gradient as:

$$-\int_{W_i} (\mathbf{q} - \mathbf{p_i}) \phi(\mathbf{q}) d\mathbf{q} = -M_{W_i}(\mathbf{C_{W_i}} - \mathbf{p_i})\,, \tag{3.9}$$

Thus, we can see that as proven by Cortes et al. (2004), the critical values of the functional would be reached when all robots are in the centroid of their Voronoi cells. One way to drive the robots to their centroids is to use the Lloyd's algorithm (Lloyd (1982)), in such way that we could reach the optimal robot configuration by: 1) constructing the Voronoi diagram, 2) computing the centroids of the Voronoi regions, 3) setting the agents position to the centroid positions, and repeat from 1).

3.4.1 Robot Model

In this section, we will describe the models considered for the robot dynamics. Let's consider an integrator dynamics, and that low-level controllers are in place to cancel existing dynamics and enforce the desired control input. As done by Pierson et al. (2017), we also consider an additive actuation error, denoted by Δ_i. This error could represent an actuator problem or hard terrain, for example. Thus, the dynamics of the robot can be written as:

$$\dot{\mathbf{p}}_\mathbf{i} = k_{p_i} \mathbf{u_i} + \boldsymbol{\Delta}_i,$$

where $\mathbf{u_i}$ is the control input and k_{p_i} a proportional gain for robot i. We also assume that Δ_i has the following form:

$$\boldsymbol{\Delta_i} = \begin{bmatrix} \Delta_{i,1} & \Delta_{i,2} \\ \Delta_{i,3} & \Delta_{i,4} \end{bmatrix} \mathbf{u_i}, \tag{3.10}$$

so, $\mathbf{p_i}$ can be written as:

$$\dot{\mathbf{p}}_\mathbf{i} = \boldsymbol{K_i}\mathbf{u_i}, \quad \boldsymbol{K_i} = \begin{bmatrix} k_{p_i} + \Delta_{i,1} & \Delta_{i,2} \\ \Delta_{i,3} & k_{p_i} + \Delta_{i,4} \end{bmatrix}. \tag{3.11}$$

For simplification, we assume that K_i is a positive definite matrix. This means that given a control input u_i, the robot might deviate from its control input in 90° in each direction, as in Figure 3.4. On the Figure, the red path shows a possible path that the robot would take. In consequence, the weightings of this robot would be adjusted according to the deviance of the path comparing to u_i. Also, as we will see in Section 3.4.3, it's unlikely that we will know the values in Δ_i and thus, an estimator is used.

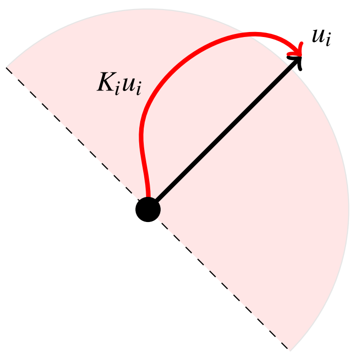


Figure 3.4: For some vector u_i, the possible directions for K_iu_i are shown in the shaded region, and a possible path is illustrated in red.

3.4.2 Control Law and Actuation Variation

In order to minimize the value of $\mathcal{H}(P, W)$, a control law $(\mathbf{u_i})$ to drive the robots needs to be chosen. We choose the desired input for the robots as the move-to-centroid controller, which implements the gradient-descent control inspired from Equation (3.9), written as:

$$\dot{\mathbf{p}}_\mathbf{i} = \mathbf{u_i} = k_{p_i}(\mathbf{C_{W_i}} - \mathbf{p_i}), \tag{3.12}$$

Replacing the with the identities found in Equation (3.11), we have

$$\dot{\mathbf{p}}_\mathbf{i} = \mathbf{u_i} + \boldsymbol{\Delta}_\mathbf{i} = \boldsymbol{K}_i(\mathbf{C_{W_i}} - \mathbf{p_i}), \tag{3.13}$$

Now, we will define a function that translates K_i to scalar, and thus giving a notion of "health" of robot i:

$$h_i = f(\boldsymbol{K_i}),$$

where h_i is the actuation performance "health" and $f(\boldsymbol{K}_i)$ is a function that translates $\boldsymbol{K}_i$ to a scalar based in its properties. The choose of the function can vary for each application, but we require that $f(\boldsymbol{K}_i)$ is bounded when K_i is bounded and continuous. In our case, $f(\boldsymbol{K}_i)$ will be the matrix norm of $\boldsymbol{K}_i$, but it could be the determinate, eigenvalues, trace or any custom function, as long as it preserves the requirement. Also, since robots do not know the true value of $\boldsymbol{K}_i$, we will define $\hat{\boldsymbol{K}}_i$ as the estimation for $\boldsymbol{K}_i$. In Section 3.4.3, we will describe such estimator.

Now, consider that we have multiple robots, and each of them can have different actuation errors. In order to deal with that, we use the following weightings adaptation law from Pierson et al. (2017):

$$\dot{w}_i = \frac{-k_w}{M_{W_i}} \sum_{j \in \mathcal{N}_i} \Big((w_i - f(\hat{\boldsymbol{K}}_i)) - (w_j - f(\hat{\boldsymbol{K}}_j)) \Big) \tag{3.14}$$

where k_w is a positive proportional gain constant. The idea of this controller is that it evaluates the K_i health of the current robot and their neighbors, and adjusts their weights. When the weight correction converges, the value of $(w_i - f(\hat{\boldsymbol{K}}_i))$ will be the same for all i, as discussed in one of our works (Pierson et al. (2017)).

3.4.3 Estimating $\hat{\boldsymbol{K}}_i$

To compute the estimated matrix $\hat{\boldsymbol{K}}_i$, we use the following online estimator:

$$\begin{aligned}
\dot{\hat{\boldsymbol{K}}}_i &= \lambda_i - \hat{\boldsymbol{K}}_i \Lambda_i \\
\dot{\lambda}_i &= \dot{\mathbf{p}}_{\mathbf{i}} (\mathbf{C}_{\mathbf{W_i}} - \mathbf{p_i})^T \\
\dot{\boldsymbol{\Lambda}}_i &= (\mathbf{C}_{\mathbf{W_i}} - \mathbf{p_i})(\mathbf{C}_{\mathbf{W_i}} - \mathbf{p_i})^T.
\end{aligned} \tag{3.15}$$

We further simplify this expression as

$$\begin{aligned}
\dot{\hat{\boldsymbol{K}}}_i &= \int_0^t \dot{\lambda}_i(\tau) d\tau - \hat{\boldsymbol{K}}_i \int_0^t \dot{\Lambda}_i(\tau) d\tau \\
&= \int_0^t \boldsymbol{K}_i (\mathbf{C}_{\mathbf{W_i}}(\tau) - \mathbf{p_i}(\tau))(C_{W_i}(\tau) - \mathbf{p_i}(\tau))^T d\tau \\
&- \hat{\boldsymbol{K}}_i \int_0^t (\mathbf{C}_{\mathbf{W_i}}(\tau) - \mathbf{p_i}(\tau))(\mathbf{C}_{\mathbf{W_i}}(\tau) - \mathbf{p_i}(\tau))^T d\tau \\
&= -\tilde{\boldsymbol{K}}_i \Lambda_i(t)
\end{aligned} \tag{3.16}$$

where $\tilde{\boldsymbol{K}}_i = (\hat{\boldsymbol{K}}_i - \boldsymbol{K}_i)$. Note that although (3.15) and (3.16) are mathematically equivalent, the robots can only directly compute (3.15) because they do not have knowledge of the true error $\tilde{\boldsymbol{K}}_i$. However, the form in (3.16) is useful for analysis.

Proofs in one of our previous works show the stability of the estimator (Pierson et al. (2017)) and weightings adjustment controller in Equation (3.14). Put simply, this estimator evaluates how aligned is the actual movement of the robot compared to the control input.

3.4.4 Non-convex environments adaptations

In Equation 3.13, we have described a control law that minimizes $\mathcal{H}(P, W)$ by driving the robots towards their Voronoi region centroids. However, in non-convex environments, some adaptations need to be done for solving the coverage control problem. In this section, we will discuss the adaptations needed, inspired by the framework developed by Pimenta et al. (2008).

Geodesic distance

In non-convex environments, the shortest traversable distance between two points may not be the Euclidean distance, because of the obstacles. In order to deal with that, we have to redefine our distance function d as

$$d(\mathbf{q}, \mathbf{p}_i) = g(\mathbf{q}, \mathbf{p}_i)^2 - w_i^2, \tag{3.17}$$

where $g(\mathbf{q}, \mathbf{p}_i)$ represents the geodesic distance between $\mathbf{q}$ and $\mathbf{p_i}$.

Since the Voronoi tessellation is defined using d, in non-convex environments it is defined as

$$W_i = \{\mathbf{q} \in \Omega \mid g(\mathbf{q}, \mathbf{p}_i)^2 - w_i^2 \leq g(\mathbf{q}, \mathbf{p}_j)^2 - w_j^2, \forall j \neq i\}. \tag{3.18}$$

This also implies changes in our deployment functional, which can be represented as:

$$\mathcal{H}(P, W) = \sum_{i=1}^{n} \int_{W_i} [g(\mathbf{q}, \mathbf{p}_i)^2 - w_i^2] \phi(\mathbf{q}) d\mathbf{q}\,, \tag{3.19}$$

Control law

In non-convex environments, we should not use a control law based in the centroid of the Voronoi regions, because the centroids might be located inside obstacles or unreachable locations. Given that, we have to use a different control law than the one proposed by Cortes et al. (2004).

Many works have addressed this problem of using Voronoi tessellations for multi-robot coverage in non-convex environments, but we will use the control law described by Pimenta et al. (2008) for this work, which can be written as

$$\mathbf{u}_i = k_{p_i} \frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} = 2k_{p_i} \int_{W_i} g(\mathbf{q}, \mathbf{p}_i)\phi(\mathbf{q})\mathbf{z}_{\mathbf{p}_i,\mathbf{q}} d\mathbf{q}, \tag{3.20}$$

where $\mathbf{z}_{\mathbf{p}_i,\mathbf{q}}$ is the unit vector along the tangent at $\mathbf{p}_i$ to the geodesic joining $\mathbf{p}_i$ and $\mathbf{q}$. In other words, $\mathbf{z}_{\mathbf{p}_i,\mathbf{q}}$ points to the direction where the geodesic path was created when calculating the distance from $\mathbf{p}_i$ to $\mathbf{q}$. The control law given in Equation (3.20) is actually very similar to the result found in Cortes et al. (2004), however, we restrict ourselves to not simplify the equation using the centroid of the Voronoi region. One way to better understand the result in Equation (3.20) is to think that it points in the direction of the gradient of the Voronoi region, thus, reducing its value and reaching a critical point.

$\hat{\boldsymbol{K}}_i$ Estimator

Since our $\hat{\boldsymbol{K}}_i$ estimator uses the control law for convex environments, we have to adapt it to use the control law in Equation (3.20).

$$\begin{aligned}
\dot{\hat{\boldsymbol{K}}}_i &= \lambda_i - \hat{\boldsymbol{K}}_i \boldsymbol{\Lambda}_\mathbf{i} \\
\dot{\lambda}_i &= \dot{\mathbf{p}}_i \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i}\right)^T \\
\dot{\boldsymbol{\Lambda}}_i &= \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i}\right)\left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i}\right)^T.
\end{aligned} \tag{3.21}$$

Such estimator works in the same way as the one for convex environments, discussed in Section 3.4.3. The estimator calculates the alignment of the movement of the robot with the control law. Now that we have discussed all the necessary adaptations for non-convex environments in our coverage control problem, we will now discuss the implementation and pseudo-algorithms.

3.5 Discrete implementation for non-convex environments

In order to implement the control algorithm, we need a way to calculate the Voronoi tessellations. In this work, we use a discrete graph search based on the work of Bhattacharya et al. (2013). The environment Ω is discretized in uniform tiling, creating a graph G with vertexes $\mathcal{V}(G)$ and edges $\mathcal{E}(G)$ and a discretized density distribution $\overline{\phi}$. The idea behind the algorithm presented by Bhattacharya et al. (2013) is to use a modified version of Dijkstra's algorithm (Dijkstra (1959)), in which each robot would be a different starting Voronoi site of the algorithm. The places where the wavefronts from different robots collide form the boundaries of the Voronoi tessellation. For this work, we modified the algorithm from Bhattacharya et al. (2013) in order to obtain the Voronoi partitions that are neighbors of one another, which is necessary for adjusting the weights as in Equation (3.14). The algorithm was implemented using Python and ROS. The source code is also available at GitHub[1].

The complete pseudo-code for computing tessellations and control commands is the **Adapted_Tessellation_and_Control** algorithm, where η is a list of graph neighbors of a specific vertex. $\mathcal{N}_i$ is the list of robot neighbors of robot i and $\mathbf{P}(u)$ is a function that returns the position of the center of the node that the two-dimensional array u is located. Lines 33, 34 and 35 were added to the original algorithm described by Bhattacharya et al. (2013) to add the determination of the Voronoi cell neighbors $\mathcal{N}_i$.

Now, we will go through the lines of the pseudo-algorithm. The inputs are the graph that describes the region Ω with the obstacles, the current robot locations, the robot weights and the density distribution function. From lines 1 to 5, we initialize some variables before we run the Dijkstra's-based algorithm. From lines 6 to 12, we initialize the robots as generating points of the Voronoi diagram. In lines 11 and 12, we initialize the neighbors of the generating points, which will be used to calculate the control integral, and will help us tell the direction that the robots should go to minimize the deployment functional. Line 13 initializes the Priority Queue, which is used by Dijkstra's Algorithm. From lines 14 to 35, we have the expansion of the graph, which will create wavefronts from the generating points until they collide, forming the Voronoi borders. Basically, the Priority Queue will always have the lowest cost node as its first element, so when the lowest cost node is requested in line 15, this element should be popped from the Priority Queue. Lines 21 and 22 calculate the mass of the Voronoi region and the control integral of the robot. From lines 24 to 27, we have the expansion of the neighbors of this node. Their geodesic distance is the geodesic distance of their generator neighbor plus the Euclidean distance between them. Then, the cost of this node from robot j is calculated. If this cost is lower than the current cost of this node, it is expanded and added to the Priority Queue. If the expansion of the node fails (the calculated cost is greater than the current cost of the node), it means that we have reached the border of the Voronoi region. Lines 33 to 35 enlists both robots as neighbors of one another. The algorithm will run from lines 14 to 35 until all nodes are expanded. When all of them are expanded, and thus, the Priority Queue is empty, we will go to lines 36 and 37. This is where the next goal of the robot is calculated, based in the control integral and the neighbors of the robot that are free of obstacles. Finally, the algorithm returns the tessellation map, the goals of each robot, the list of neighbors of each robot and the mass of the Voronoi regions.

[1]Source code will be available at `https://github.com/lucascoelhof/voronoi-hsi`

$\{\tau, \{\mathbf{p}_i'\}, \mathcal{N}_i, M_{W_i}\}$ = **Adapted_Tessellation_and_Control**$(G, \{\mathbf{p_i}\}, \{w_i\}, \overline{\phi})$
Modified from (Bhattacharya et al. (2013))
Inputs: a. Graph G
b. Agent locations $\mathbf{p}_i \in \Omega,\ i = 1, 2, \cdots, N$
c. Agent weight $w_i \in \mathbb{R},\ i = 1, 2, \cdots, N$
d. Discretized density distribution function $\overline{\phi}$
Outputs: a. The tessellation map $\boldsymbol{\tau} \in \mathbb{I}^2$
b. The next position of each robot, $\mathbf{p_i'} \in \Omega,\ i = 1, 2, \cdots, N$
c. The list of neighbors of each robot, $\mathcal{N}_i\ i = 1, 2, \cdots, N$
d. Mass of Voronoi partitions $M_{W_i},\ i = 1, 2, \cdots, N$

```
Initiate $g$: Set $g(v) := \infty,\ \forall v \in \mathcal{V}(G)$  // Geodesic distances
Initiate $\rho$: Set $\rho(v) := \infty,\ \forall v \in \mathcal{V}(G)$  // Weights
Initiate $\tau$: Set $\tau(v) := -1,\ \forall v \in \mathcal{V}(G)$  // Tessellation
          // Pointer to robot neighbor. $\eta : \mathcal{V}(G) \to \mathcal{V}(G)$
Initiate $\eta$: Set $\eta(v) := \emptyset,\ \forall v \in \mathcal{V}(G)$
for each ($i \in \{1, 2, \cdots, N\}$)
    Set $g(p_i) = 0$
    Set $\rho(p_i) = -w_i^2$
    Set $\tau(p_i) = i$
    Set $I_i := \mathbf{0}$    // The control integral. $\mathbf{I}_i, \mathbf{0} \in \mathbf{C}$
    for each ($\mathbf{q} \in \eta_G(\mathbf{p_i})$)  // For each neighbor of $\mathbf{p_i}$
        Set $\eta(\mathbf{q}) = \mathbf{q}$
Set $Q := \mathcal{V}(G)$    // Set of un-expanded vertices
while ($Q \neq \emptyset$)
    $\mathbf{q} := \arg\min_{\mathbf{q}' \in Q}\ \rho(\mathbf{q}')$  // Maintained by a heap data-structure.
    if ($g(\mathbf{q}) == \infty$)
        break
    Set $Q := Q - \mathbf{q}$    // Remove $q$ from $Q$
    Set $j := \tau(\mathbf{q})$
    Set $\mathbf{s} := \eta(\mathbf{q})$
    if ($\mathbf{s}\ != \emptyset$)    // Equivalently, $\mathbf{q} \notin \{\mathbf{p}_i\}$
        Set $\mathbf{I_j} \mathrel{+}= \overline{\phi}(\mathbf{q}) \times g(\mathbf{q}) \times (\mathbf{P}(\mathbf{s}) - \mathbf{P}(\mathbf{p_j}))$
        Set $M_{W_j} += \overline{\phi}(\mathbf{q})$
    for each ($w \in \eta_G(\mathbf{q})$)    // For each neighbor of $\mathbf{q}$
        Set $\mathbf{g}' := g(\mathbf{q}) + g(\mathbf{q}, w)$
        Set $\rho' := d(g', w_j)$
        if ($\rho' < \rho(w)$)
            Set $g(w) = g'$
            Set $\rho(w) = \rho'$
            Set $\tau(w) = j$
            if ($\mathbf{s}\ != \emptyset$)    // Equivalently, $\mathbf{q} \notin \{\mathbf{p}_i\}$
                Set $\eta(w) = s$
        else
            $m = \tau(w)$    // Gets robot id
            $\mathcal{N}_j += m$    // Adds robot m to the list of neighbors of j
for each ($i \in \{1, 2, \cdots, N\}$)
    Set $\mathbf{p}_i' := \arg\max_{\mathbf{u} \in \eta_G(\mathbf{p}_i)}\ (\mathbf{P}(\mathbf{u}) - \mathbf{P}(\mathbf{p}_i)) \cdot I_i$    // Choose action best aligned along $I_i$.
return $\{\boldsymbol{\tau}, \{\mathbf{p_i'}\}, \mathcal{N}_i, M_{W_i}\}$
```

3.5.1 Graph generation

The graph G is constructed using the occupancy grid data provided from a grayscale image, such that white areas indicate regions that are free, and the probability of an obstacle in the region increases as the color approaches black, with black indicating a 100% possibility of having an obstacle. The *map_server*[2] package reads the grayscale image file and creates an occupancy grid. The occupancy grid is a $M \times N$ matrix, in which each element represents a small discretized area from the map along a probability of that area being an obstacle, with values varying from 0% to 100%. This matrix is used to build a graph that connects the vertexes with their neighboring vertexes based on the probability of obstacle. If the neighbor has over 20% obstacle probability value, it is considered an obstacle and thus it is not connected to the graph. As we will discuss deeper in the following sections, in our VR application the operators can create obstacles, and when they do it, the occupancy grid is updated and consequently, the graph also is updated.

3.5.2 Robots and control law

To deal with the possibility of dynamically changing the environment Ω, some small changes to the Adapted Lloyd's algorithm presented by Bhattacharya et al. (2013) were implemented. But the overall algorithm still consists of iterating over the Voronoi diagram calculation and updating the positions of the robots at each iteration. The pseudo algorithm follows:

```
$\left\{\boldsymbol{\tau}^f, \{\mathbf{p_i}\}^f, \{w_i\}^f\right\}$ = Motion_Algorithm $(G, \{\mathbf{p_i}\}, \{W_i\}, \overline{\phi})$
Inputs: a. Occupancy Grid $O_G$
        b. Initial agent locations $\mathbf{p_i} \in \Omega,\ i=1,2,\cdots,N$
        c. Discretized density distribution function $\overline{\phi}$
        d. Agent weight $w_i \in \mathbb{R},\ i=1,2,\cdots,N$
Outputs: a. Final tessellation map $\boldsymbol{\tau} \in \mathbb{I}^2$
         b. Robot final position, $\mathbf{p_i'} \in \mathcal{V}(G),\ i=1,2,\cdots,N$
         c. Robot final weightings, $w_i' \in \mathbb{R}$

1 while $(t < n)$ // not converged
2     $G$ = build_graph$(O_G)$
3     Set $\{\boldsymbol{\tau}, \{\mathbf{p_i'}\}, M_{W_i}\} :=$
          Adapted_Tessellation_and_Control $(G, \{p_i\}, \{w_i\}, \overline{\phi})$
4     for each $(i \in \{1, 2, \cdots, N\})$
5         Move $i^{\text{th}}$ robot from $\mathbf{P}(\mathbf{p_i})$ to $\mathbf{P}(\mathbf{p_i'})$
6         Set $\mathbf{p_i} = \mathbf{p_i'}$    // Update robot positions
7     Adapt_Weightings$(w_i,\ M_{W_i})$
8 return $\{\boldsymbol{\tau}, \{\mathbf{p}\}, \{w\}\}$  // Latest tessellation & positions
```

Adapt_Weightings is an implementation of the Equation (3.14), **build_graph** is a function that builds the graph as described in subsection 3.5.1 when there are changes on

[2]ROS map_server package. http://wiki.ros.org/map_server

occupancy grid O_G. The estimation of t is the current elapsed time and n is a time set for convergence. It's worth also mentioning that at the robot position control level (line 5), the Estimation of $\hat{\mathbf{K}}_\mathbf{i}$ should happen, implementing Equation (3.21). The pseudo-algorithm for **Adapt_Weightings** follows:

```
$\left\{\{w_i\}^f\right\}$ = Adapt_Weightings $(G, \{\mathbf{p_i}\}, \{W_i\}, \overline{\phi})$
Inputs: a. Robot weights $w_i \in \mathbb{R},\ _{i=1,2,\cdots,N}$
        b. Agent locations $\mathbf{p_i} \in \Omega,\ _{i=1,2,\cdots,N}$
        c. Voronoi regions $W_i \in \mathbb{R},\ _{i=1,2,\cdots,N}$
        d. Robot neighbors $\mathcal{N}_i \in \mathbb{R},\ _{i=1,2,\cdots,N}$
Outputs: a. Final robot weights map $w_i^f \in \mathbb{R},\ _{i=1,2,\cdots,N}$

1  Initiate $w_i^f$: Set $w_i^f := 0,\ \forall\ i \in \{1, 2, \cdots, N\}$
1  for each $(i \in \{1, 2, \cdots, N\})$
2      for each $(j \in \{1, 2, \cdots, N\})$
3          if $j \in \mathcal{N}_i$  // If j is a neighbor of i
4              $w_i^f += \frac{-k_w}{M_{W_i}}\left((w_i - f(\hat{\boldsymbol{K}}_i)) - (w_j - f(\hat{\boldsymbol{K}}_j))\right)$
5  return $\left(\{w\}^f\right)$  // Updated weights
```

3.6 Virtual Reality environment development

In order to achieve an intuitive and flexible human interactions, we have developed the interface for robot control in the coverage abstraction using a virtual reality headset. The chosen headset, *Oculus Rift*, is a virtual reality headset developed by Oculus VR. It has a 1080x1200 resolution OLED display per eye, with a 110° field of view. Oculus has also a set of two controllers, called Oculus Touch, and a positional tracking system, *Constellation*. It uses two infrared cameras to get the position of the headset and the two controllers in six degrees of freedom. Figure 3.5 shows a user wearing the *Oculus Rift*.

3.6.1 Unity

In order to facilitate the development of the VR application, we have used a game engine called Unity. Game engines are software development environments that provide core functionality for creating games, such as a rendering engine, physics, collision detection, scripting and so on. Game engines also receive a lot of plug-ins from the community, which makes the development easier. Moreover, integrating with a VR headset in Unity is a fairly simple task. Among other game engines, Unity is known for being simpler to use than its counterparts. Also, Unity has a integration with ROS, as we will see in the next section.

Figure 3.5: User wearing the virtual reality headset.

3.6.2 ROS-C# Bridge

For this work, an interface between ROS#, Unity and Oculus was developed, which publishes in ROS the positions of the headset, hands and button states in ROS[3]. Since Oculus is only officially supported in Windows and ROS in Ubuntu, we used ROS in a Ubuntu virtual machine hosted by a Windows computer, which is running Unity to create the virtual reality application, with the help of *VRTK*[4]. *VRTK* provides tools and libraries that help on the development of VR applications in Unity. To communicate between Unity and ROS, we used ROS# (Bischoff (2018)), which is a set of libraries and tools in C# for communication between ROS and C# applications. We used *Stage*[5] simulator for robot simulation, and *rosbridge*[6] to communicate with ROS#.

Figure 3.6 shows the setup used for the simulations. The Linux VM was used only for simplification, as we could have two separate computers using exactly the same implemented software.

Virtual reality can bring some benefits to human-swarm interaction. Some of the main advantages of VR HSI are:

[3]Source code will be available at `https://github.com/lucascoelhof/oculus-ros-sharp`

[4]VRTK `https://vrtoolkit.readme.io/`

[5]Stage simulator `http://wiki.ros.org/stage`

[6]Rosbridge package.`http://wiki.ros.org/rosbridge_suite`

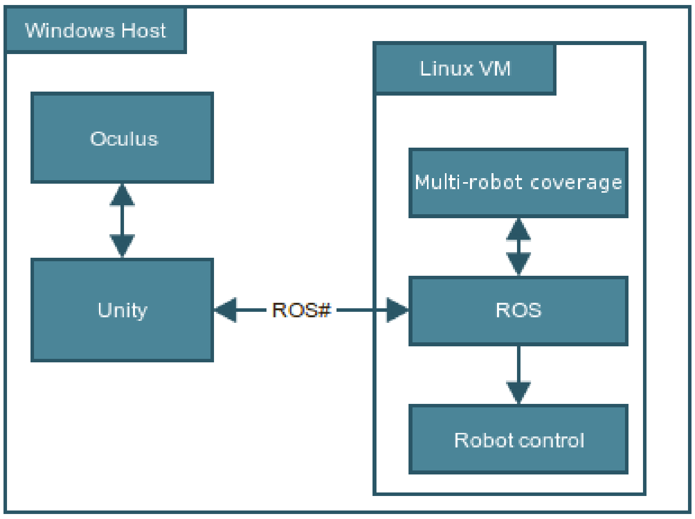


Figure 3.6: Setup for interfacing ROS and Oculus Rift.

Physical Isolation

On virtual reality, the user does not need to be close to the agents to control them, as long the agents still have communication with the human who is controlling the swarm. This greatly expands the operational flexibility because it removes the distance constraints, allowing the human to control the swarm even when the robots are in dangerous or inhospitable locations.

Robot and environment additions

Since the world is a computer graphics world, we are free to modify it and add any features that are impossible in real world. For example, for this work we implemented a flyover and teleporting feature, which is totally feasible in virtual world, but not yet fully replicable in real world for regular humans.

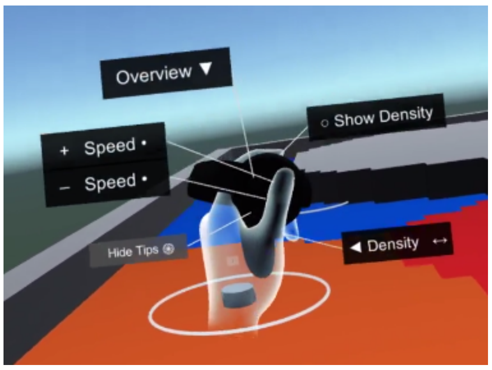
Intuitiveness and immersion

A virtual reality headset with motion tracking and handheld wireless controllers feels very intuitive to humans. It gives them the freedom to look and walk around and interact with the world with common gestures, like grabbing, pinching and pressing. Also, because of the high field of view and resolution, the users feel totally immersed on the virtual world.

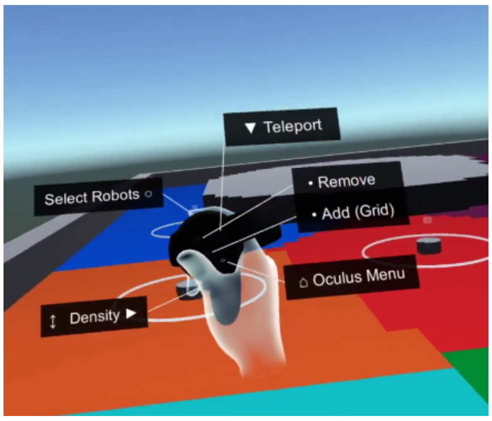
3.6.3 Features implemented on VR application

On the VR application, implemented using Unity, users can see the robots, obstacles, the Voronoi tessellation and the density distribution function. This is feasible assuming a central system responsible to broadcast the position of all robots and a known map to the

VR application. In order to interact with the robots, users can teleport, fly over, create and remove obstacles, set robot weights and create density distribution functions. Figure 3.7 shows the user interface with the commands available.



(a) Left hand menu.



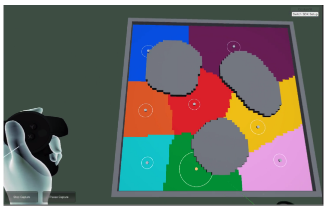
(b) Right hand menu.

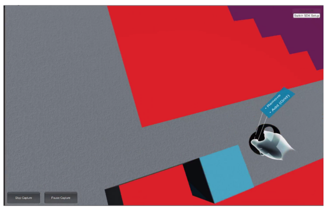
Figure 3.7: Menu of the VR application showing the features.

These are the implemented interactions between the users and the group of robots:

Obstacle creation and removal

When users press the A button, an obstacle is created in the location of their right arm. When creating this obstacle, ROS# publishes on ROS the new occupancy grid, and the Voronoi application reads it and updates the graph and the tessellation. Likewise, they can also remove obstacles using B button. Since the graphs are updated and the tessellation algorithm runs regularly, the robots can react to the new obstacles. Figure 3.8 shows the process of creating an obstacle and Figure 3.9 shows a user removing obstacles.


(a) Before.


(b) User vision.

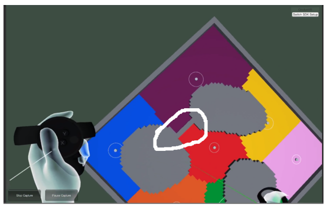
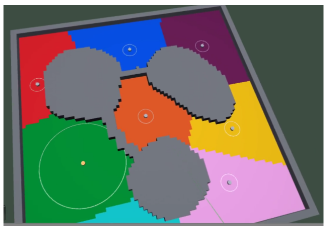
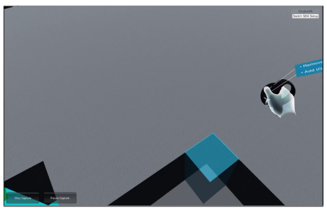

(c) After

Figure 3.8: Obstacle creation feature. Operator removed obstacles next to the light blue region.


(a) Before.


(b) User vision.

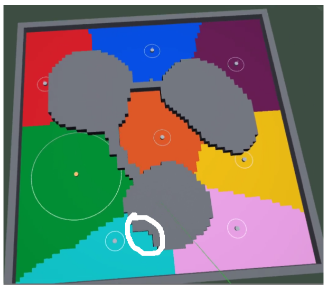

(c) After.

Figure 3.9: Obstacle removal feature.

Density function interaction

The coverage control algorithm implemented allows the use of density distribution functions, which can attract the robots. For simplicity, we modeled these density distributions as a two-dimensional Gaussian functions. When the user presses the right trigger button, a Gaussian is created. The function is modeled using the following equation,

$$f(x, y) = B + Ae^{-\left(\left(\frac{x-x_c}{\sigma}\right)^2+\left(\frac{y-y_c}{\sigma}\right)^2\right)} \tag{3.22}$$

where B is a constant, A is the height where the arm of the user is raised, x_c and y_c are the position of the right arm in relation to the tessellation plane in each respective coordinate and σ can be controlled by moving the left arm in relation to the right arm. Approaching the arms reduces the value of σ while moving them away increases σ. When the user sets the new density function, ROS# sends the Gaussian parameters to the Voronoi application, which regenerates the density distribution. This update influences the robot controllers, and will make robots converge to another distribution, changing then the Voronoi diagrams.

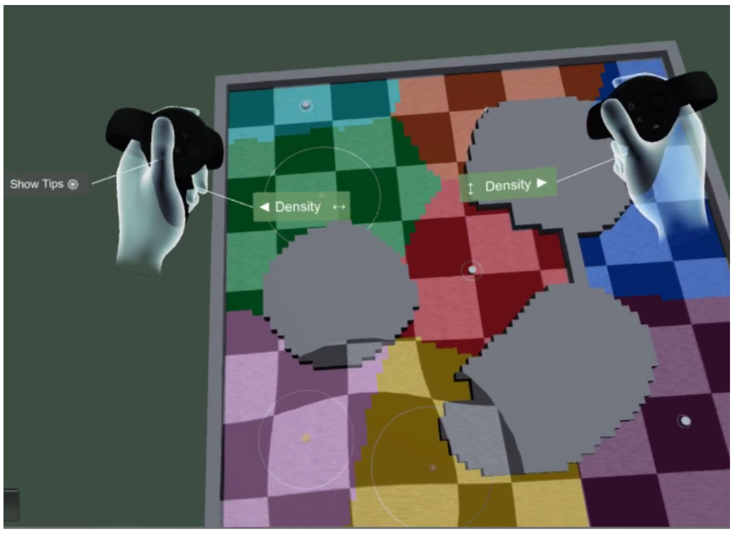


Figure 3.10: User creating a Gaussian density distribution in aerial view.

Robot gains and weight

To allow humans to manually set the speed gain of the robots, they can select a speed gain k_{p_i} and then hold the right trigger button to create a radius that is used to select a group of robots and increase their k_{p_i} gains. As shown on Equation (3.18) and Equation (3.14), this will have impact on the weight values of the robots and thus, their Voronoi partition W_i. When a value is set, we use ROS# to publish to our coverage control application the new values of k_{p_i}.

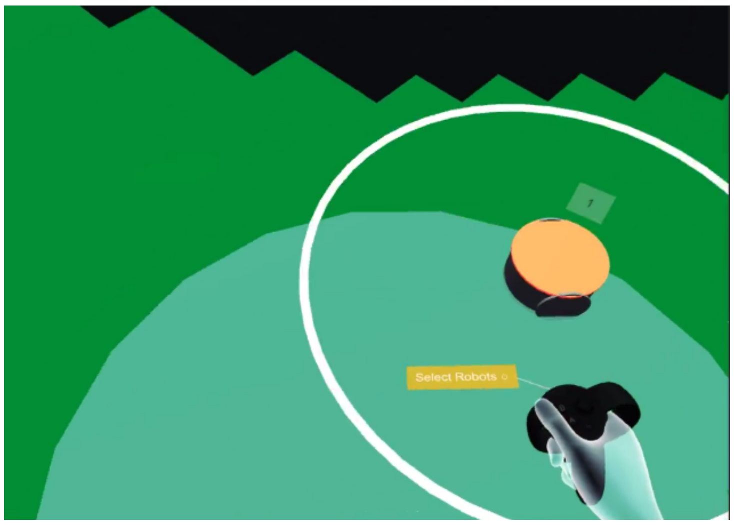


Figure 3.11: User selecting a robot.

Teleporting

Although using the *Oculus Constellation* the users can walk freely on the virtual world, it can be tiring or limiting to walk around always. To solve this problem, we implemented a teleporting feature using *VRTK*. When users press the right thumb button in *Oculus Touch*, a laser pointer shows up, and when the button is released, the user is teleported to the pointer location.

Aerial view

To better see the world and the robots and get a panoramic view of the simulation, we implemented a feature that allows users to have an aerial view of the simulated area. When users press the left right thumb button, they are teleported to a place above, with a better view. Also, using the aerial view they can produce density distributions with values much higher than standing on ground.

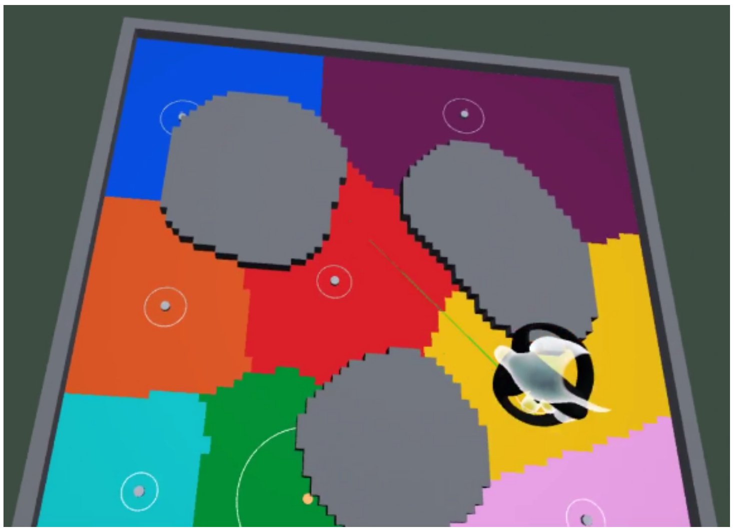


Figure 3.12: User using the teleport function. The green laser points to the position where the user will be relocated.

3.6.4 Real world implementation

Given the interface developed, it's also worth discussing what a real implementation would be like. Given that the algorithm needs the position of all robots to calculate the tessellation seen by the human operator, we would need a way to centralize this information in the computer which is dealing with the rendering for the VR headset. If the environment has obstacles that can change over time, we also need a entity which would collect such data. It could be a human operator or even sensors installed on the robots.

3.7 Voro* algorithm

Bhattacharya et al. (2013) and Section 3.5 introduced the Dijkstra-based algorithm for calculating Voronoi tessellations and also the control law of the robots. In Section 2.2, we introduced coverage control algorithms. If we consider a decentralized strategy, in which each robot has to calculate their control law for themselves, all the solutions proposed before have a common flaw: either robots need to calculate the Voronoi regions of all the other robots or they need to make considerations about who would be their Voronoi diagram neighbors before actually knowing them. This can severely cripple the performance of the algorithm and overall convergence strategy because it brings unnecessary calculations, since in order to calculate its control law, a robot only needs to calculate its own region. In this section, we will introduce a new algorithm based on the work of Bhattacharya et al. (2013) and the A* algorithm (Hart et al. (1968)), which we call Voro*. The A* is an extension of Dijkstra's algorithm that introduced a heuristic function to bias the vertex expansion in the direction of the goal. Consider that each robot needs to calculate their Voronoi diagram locally, and as shown before, each robot in the algorithm presented by

Bhattacharya et al. (2013) is a seed of the Dijkstra's algorithm. In the same way as A* improved Dijkstra's with a heuristic, we could also do that considering that the heuristic is the Euclidean distance between a point $\mathbf{q}$ in the region Ω and the position of the robot for which we are calculating the Voronoi region (which we will call $\mathbf{p}_m$).

3.7.1 Redefining the cost function d

As we previously introduced in Sections 3.2 and 3.4, the cost function d defines the boundaries of the Voronoi regions. As we want to calculate the Voronoi region of only one robot, we need some way of biasing the expansion of the graph nodes towards the position of the robot $\mathbf{p}_m$. One way we could do that is using the following function:

$$d^*(\mathbf{q}, \mathbf{p}_i, \mathbf{p}_m) = g(\mathbf{q}, \mathbf{p}_i)^2 - w_i^2 + h(\mathbf{q}, \mathbf{p}_m), \tag{3.23}$$

where $h(\mathbf{q}, \mathbf{p}_m)$ represents the heuristic function, which is given by Euclidean distance between $\mathbf{q}$ and $\mathbf{p}_m$.

Consider now the inequality that mathematically defines the Voronoi diagrams, that we introduced earlier in Section 3.2 and redefined in Section 3.4. If we substitute our new cost function into the inequality, we will have:

$$W_i = \{\mathbf{q} \in \Omega \mid d^*(\mathbf{q}, \mathbf{p}_i, \mathbf{p}_m) \le d^*(\mathbf{q}, \mathbf{p}_j, \mathbf{p}_m), \quad \forall j \ne i\}.$$
$$W_i = \{\mathbf{q} \in \Omega \mid g(\mathbf{q}, \mathbf{p}_i)^2 - w_i^2 + h(\mathbf{q}, \mathbf{p}_m) \le g(\mathbf{q}, \mathbf{p}_j)^2 - w_j^2 + h(\mathbf{q}, \mathbf{p}_m), \forall j \ne i\}.$$

We can see that our heuristic function was introduced in both sides of the inequality with the same value: a function of $\mathbf{q}$, which is the position of a point in Ω and $\mathbf{p_m}$, which is constant during the calculation of the diagram. Since they are constant and present in both sides of the inequality, canceling $h(\mathbf{q}, \mathbf{p_m})$ in both sides, we will have:

$$W_i = \{\mathbf{q} \in \Omega \mid g(\mathbf{q}, \mathbf{p}_i)^2 - w_i^2 \le g(\mathbf{q}, \mathbf{p}_j)^2 - w_j^2, \forall j \ne i\}. \tag{3.24}$$

which is the same set introduced in Section 3.4

This implicates that the introduction of the heuristic function has not changed the geometry of the Voronoi diagrams. Even though the expansion of nodes is biased, the outcomes, which are the Voronoi diagram and the control law given for robot m, are the same. Figure 3.13 shows a diagram generated by the Dijkstra's inspired and Voro*. The black areas are the nodes that were not expanded, and thus not demanded a calculation.

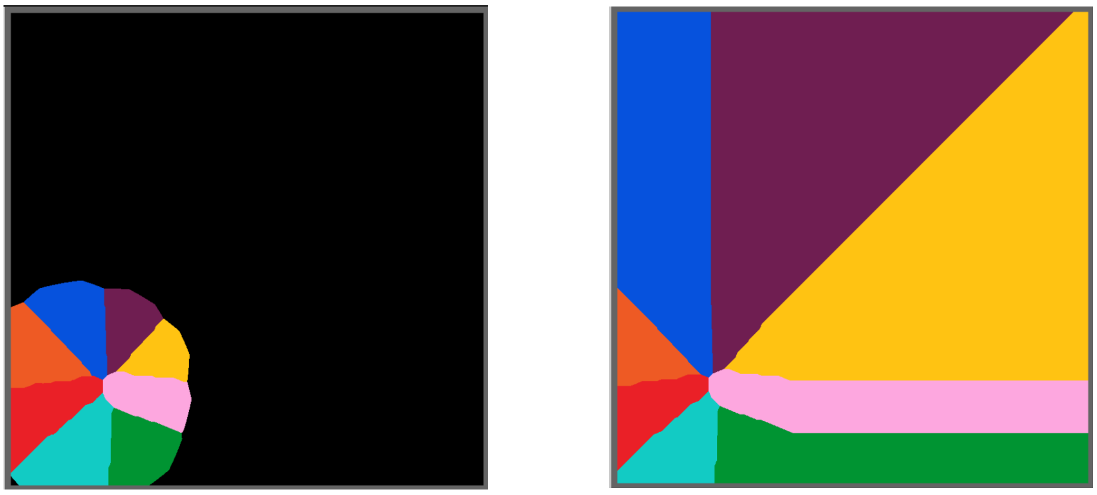

(a) Voro* algorithm. (b) Dijkstra's inspired algorithm.

Figure 3.13: Comparison between the Voro* and Dijkstra inspired Voronoi tessellation algorithm. The robot considered for the Voro* algorithm (m) is the red one. Black regions are nodes that were not expanded. Note that the Voronoi region for the red robot is the same in both diagrams

3.7.2 Changes to original Dijkstra's based algorithm

In order to transform the **Adapted_Tessellation_and_Control** into a local calculation of the Voronoi diagram considering a heuristic function, a few changes were introduced. In addition to changes to the d function, for running the algorithm in a decentralized fashion and know when there are no more nodes in the priority queue for expansion, we need to know when nodes from the robot of interest are entering and exiting the priority queue. Basically, this happens when a node from robot m is expanded or when other robot reaches a node with a lower cost than m. The pseudo-code follows in the next page. There are some substantial changes from the algorithm in Section 3.5 but the overall idea stays the same. First difference are the outputs. Since we are concerned in calculating the control law for only one robot, the outputs also now reference only robot m. Alongside with the heuristic function, we have also introduced the concept of *node candidate*, used with the ζ symbol. The node candidate is a element in the priority queue that has not been expanded yet. We will keep track of all node candidates that robot m generates, as we do in lines 13, 25, 39 and 42. Each time m gets a node candidate (lines 12 and 42), a counter is increased, and each time a node candidate of m loses a node for other robot (line 39) or the node is expanded by robot m (line 25), the counter is decreased. Keeping track of the node candidates allows us to halt the while loop as soon as we don't have any expandable nodes in the Priority Queue.

```
$\{\tau, \{\mathbf{p'_m}\}, \mathcal{N}_m, M_{W_m}\} = \mathbf{Voro*}(G, \{\mathbf{p_m}\}, \{w_i \overline{\phi}, m)$
Inputs: a. Graph $G$
        b. Agent locations $\mathbf{p_i} \in \Omega,\ i = 1, 2, \cdots, N$
        c. Agent weight $w_i \in \mathbb{R},\ i = 1, 2, \cdots, N$
        d. Discretized density distribution function $\overline{\phi}$
        e. ID of the robot $m$
Outputs: a. The tessellation map $\tau \in \mathbb{I}^2$
         b. The next position of robot m, $\mathbf{p'_m}$
         c. The list of neighbors of robot m, $\mathcal{N}_m$
         d. Mass of Voronoi partition $M_{W_m}$
1  Initiate $g$: Set $g(v) := \infty$,
                 for all $v \in \mathcal{V}(G)$   // Geodesic distances
2  Initiate $\rho$: Set $\rho(v) := \infty,\ \forall v \in \mathcal{V}(G)$   // Weights of the nodes
3  Initiate $\tau$: Set $\tau(v) := -1,\ \forall v \in \mathcal{V}(G)$  // Tessellation
4  Initiate $\zeta$: Set $\zeta(v) := \emptyset,\ \forall v \in \mathcal{V}(G)$   // Candidate for the node
5  Initiate $c$: Set $c_m := 0$,   // Counter of nodes from robot $m$ on Priority Queue
6  Initiate $\eta$: Set $\eta(v) := \emptyset,\ \forall v \in \mathcal{V}(G)$   // Pointer to robot neighbor. $\eta : \mathcal{V}(G) \to \mathcal{V}(G)$
7  Set $\mathbf{I_m} := \mathbf{0}$   // The control integral. $\mathbf{I}_m, \mathbf{0} \in \mathbf{C}$
8  for each $(i \in \{1, 2, \cdots, N\})$
9      Set $g(\mathbf{p_i}) = 0$
10     Set $\rho(\mathbf{p_i}) = -w_i^2$
11     Set $\tau(\mathbf{p_i}) = i$
12     if $(i == m)$
13         Set $c_m += 1$
14     for each $(\mathbf{q} \in \eta_G(\mathbf{p_i}))$  // For each neighbor of $\mathbf{p_i}$
15         Set $\eta(\mathbf{q}) = \mathbf{q}$
16 Set $Q := \mathcal{V}(G)$   // Set of un-expanded vertices
17 while $(Q \neq \emptyset \wedge c_m > 0)$
18     $\mathbf{q} := \arg\min_{\mathbf{q'} \in Q}\ \rho(\mathbf{q'})$  // Maintained by a heap data-structure.
19     if $(g(\mathbf{q}) == \infty)$
20         break
21     Set $Q = Q - \mathbf{q}$   // Remove $\mathbf{q}$ from $Q$
22     Set $j := \tau(\mathbf{q})$
23     Set $\mathbf{s} := \eta(\mathbf{q})$
24     if $(j == m)$
25         Set $c_m -= 1$
26     if $(\mathbf{s}\ != \emptyset \wedge j == m)$   // Equivalently, $\mathbf{q} \notin \{\mathbf{p_i}\}$
27         Set $I_m \mathrel{+}= \overline{\phi}(\mathbf{q}) \times g(\mathbf{q}) \times (\mathbf{P}(\mathbf{s}) - \mathbf{P}(\mathbf{p_m}))$
28         Set $M_{W_m} += \overline{\phi}(\mathbf{q})$
29     for each $(w \in \eta_G(\mathbf{q}))$   // For each neighbor of $q$
30         Set $g' := g(\mathbf{q}) + g(\mathbf{q}, w)$
31         Set $\rho' := d^*(g', p_m, w_j)$
32         if $(\rho' < \rho(w))$
33             Set $g(w) = g'$
34             Set $\rho(w) = \rho'$
35             Set $\tau(w) = j$
36             if $(\mathbf{s}\ != \emptyset)$   // Equivalently, $\mathbf{q} \notin \{\mathbf{p_i}\}$
37                 Set $\eta(w) = \mathbf{s}$
38             if $(\zeta(w)' == m)$
39                 Set $c_m -= 1$
40             Set $\zeta(w) = j$
41             if $(j == m)$
42                 Set $c_m += 1$
43         else if $(j == m)$
44             $k = \tau(w)$   // Gets robot id
45             $\mathcal{N}_m += k$   // Adds robot m to the list of neighbors of m
46 Set $\mathbf{p'_m} := \arg\max_{\mathbf{u} \in \eta_G(\mathbf{p_m})}\ (\mathbf{P}(\mathbf{u}) - \mathbf{P}(\mathbf{p_m})) \cdot \mathbf{I_m}$   // Choose action best aligned along $\mathbf{I_m}$.
47 return $\{\boldsymbol{\tau}, \{\mathbf{p'_m}\}, \mathcal{N}_m, M_{W_m}\}$
```

3.7.3 Order of complexity

When compared with the performance of the Dijkstra's inspired algorithm, Voro* has also its advantages. Bhattacharya et al. (2013) stated that the order of complexity of this algorithm is a function of the number of vertexes and edges:

$$O(\mathcal{E}(G)\log(\mathcal{V}(G))),$$

but given that we are in a two dimensional space, thus the maximum number of neighbors one vertex could have is eight, we can simplify the order of complexity to:

$$O(8\mathcal{V}(G)\log(\mathcal{V}(G))) = O(\mathcal{V}(G)\log(\mathcal{V}(G))).$$

Since Voro* calculates only local diagrams and the order of complexity is directly proportional to the number of the vertexes, the number of vertexes that Voro* expands is significantly lower and thus, the Voronoi region is calculated faster. In fact, as the number of the robots increase, the smaller will be the Voronoi regions and then, the order of complexity of Voro* can be inversely proportional to the number of robots. As the time of this writing, we cannot provide a mathematical proof of this complexity order, but in the Results chapter we will show in simulation that increasing the number of robots will decrease the computational cost.

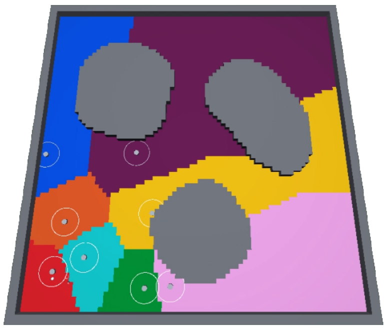
4
Experiments and Results

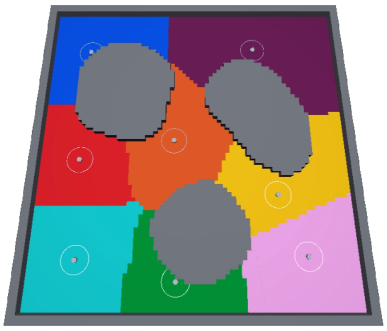
4.1 Simulations

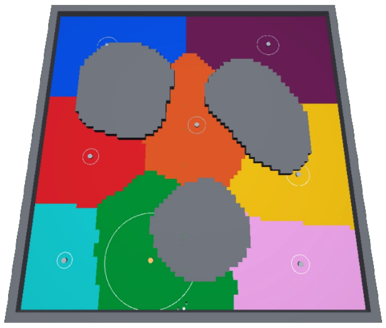
To demonstrate the HSI application and the coverage controller, we first conducted simulations[1]. Using a rectangular environment with three obstacles in the middle, the results demonstrate the interaction between the user and robots and the convergence of the algorithm even after modifications made by the human operator.

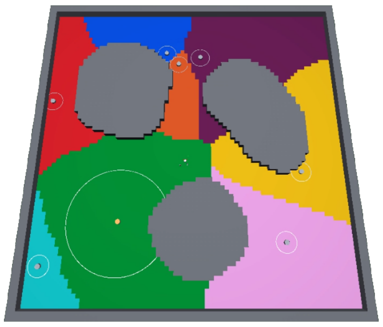
All agents start with $k_{p_i} = 1$. The white radius around the robots show their current weight. On simulation, we have used differential drive robots, so we have used a point-offset controller (Michael & Kumar (2009)) to address the nonholonomic constraints. To show the convergence of the algorithm, the human agent did not interact with the group of robots until around $t = 150$. Then, the human agent selects robot 1 (green) to increase its k_{p_i} value to 3.2. Because of the adaptive weightings controller on Equation (3.14), increasing the k_{p_i} value increases the weight of robot 1, while decreasing the weightings of other robots, as shown in Figure 4.3. Since we use a geodesic distance as in Equation (3.17), increasing the weight of the robot will increase its Voronoi partition V_1. At $t = 200$, the human agent created a Gaussian shaped density distribution function centered on the intersection of the Voronoi partitions of robots 7 (orange), 2 (blue) and 5 (purple). This changed the convergence and attracted more robots to that area. Figure 4.1d shows the convergence of robots and the increased number of robots on that area, differently from Figure 4.1c, in which the robots are more spread around. Then, the human agent creates a flat density function, which removes the importance of the previously set area. Around $t = 300$, the human agent changed the occupancy grid of the area, connecting some of the obstacles in the area. On Figure 4.1e, we can see that because of the dynamic graph generation present on the **Motion_Algorithm**, the robots could react to the changes of the obstacles and reached another optimal distribution. Figure 4.2 shows the deployment functional over time. Spikes present around $t = 200$ and $t = 250$ are because of changes on the density distribution function.

[1]Simulation video available at `https://www.youtube.com/watch?v=cpniwb6UrF8`


(a) Distribution of robots and tessellation at $t = 0$


(b) Convergence of robots to an optimal position at $t = 150$


(c) Robot 1 with increased weight and a larger Voronoi region than in Figure 4.1b


(d) Convergence showing robots attracted by a Gaussian density function.

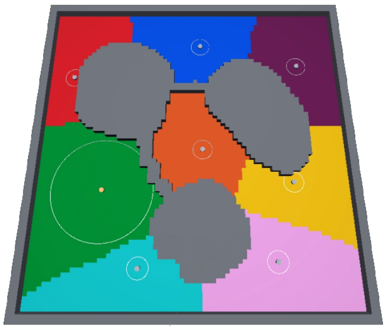

(e) Changes on the obstacles were introduced but robots were able to reach a new convergence.

Figure 4.1: Steps of the first simulation.

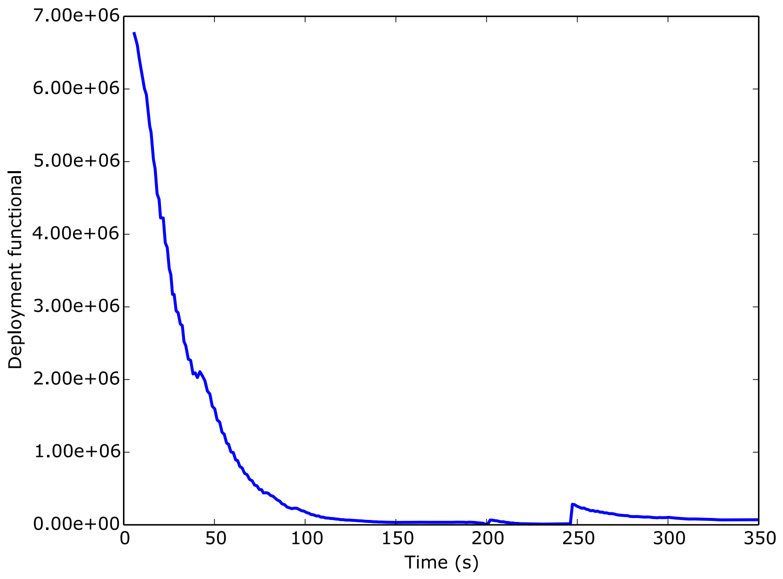


Figure 4.2: Deployment functional over the simulation.

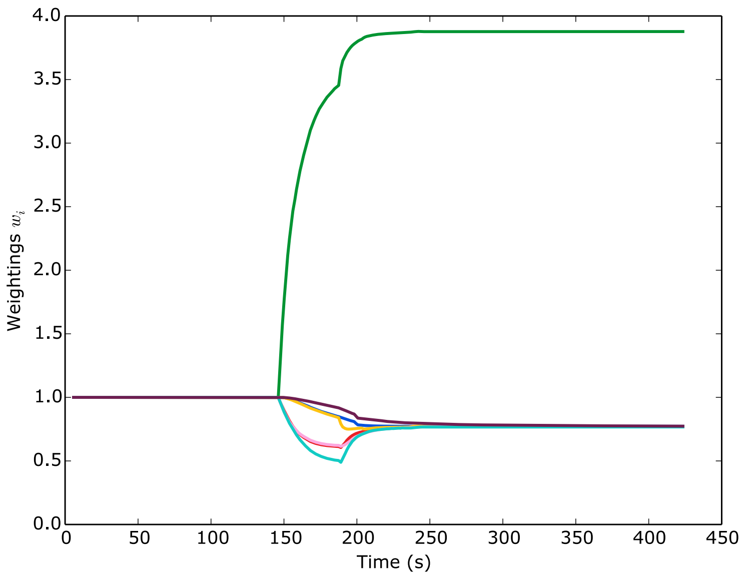
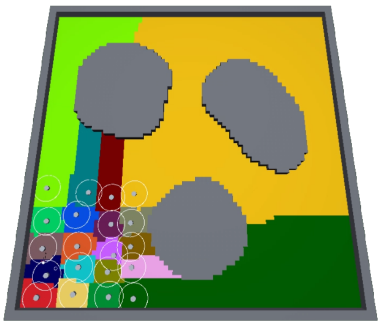
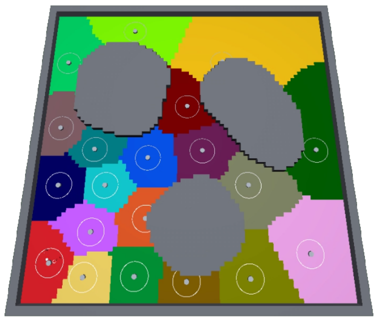
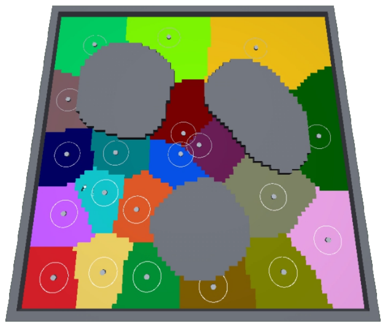


Figure 4.3: Weightings of the robots. Around $t = 150$, the human changed the gain k_{p_i} of robot 1 (green), which changed the weightings of all robots.

In a second simulation, we have evaluated the human-robot interaction using a larger group of robots. Apart from the number of robots, this simulation has the same parameters as the previous one. In Figure 4.4a, we can see the initial deployment of the robots. At $t = 228s$, the operator decides to interact with the robots by creating a Gaussian-shaped density distribution function right in the center of the environment. Note that, this time, the human interacted with the robots before the convergence, as we can see in Figure 4.4b. In Figure 4.4c, at $t = 288s$, we can see that the robot swarm has adapted its coverage to consider the density function. After that, the operator creates a flat distribution function, chooses a robot to increase its weight and creates an obstacle. Figure 4.4d shows the final deployment of the robots with the new obstacles and a robot with increased weight. At this simulation, we can see that robots seem to have converged to a local minimum, because there is a higher concentration of robots in the left bottom corner of the environment and the density distribution is flat. In this case, the human agent could, for example, help the robots by creating density functions that would disperse them from that region, allowing them to overcome the local minimum. Figure 4.5 the deployment functional over time and Figure 4.6 the weightings of the robots.


(a) Distribution of robots and tessellation at $t = 0$


(b) Deployment before applying the Gaussian density distribution function ($t = 228s$).


(c) Deployment after some time after the density function was applied ($t = 288s$).

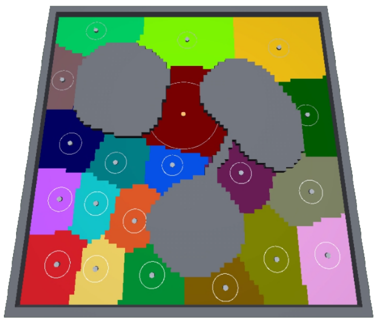

(d) Final convergence after changing the weight of a robot and introducing obstacles ($t = 380s$).

Figure 4.4: Steps of the second simulation.

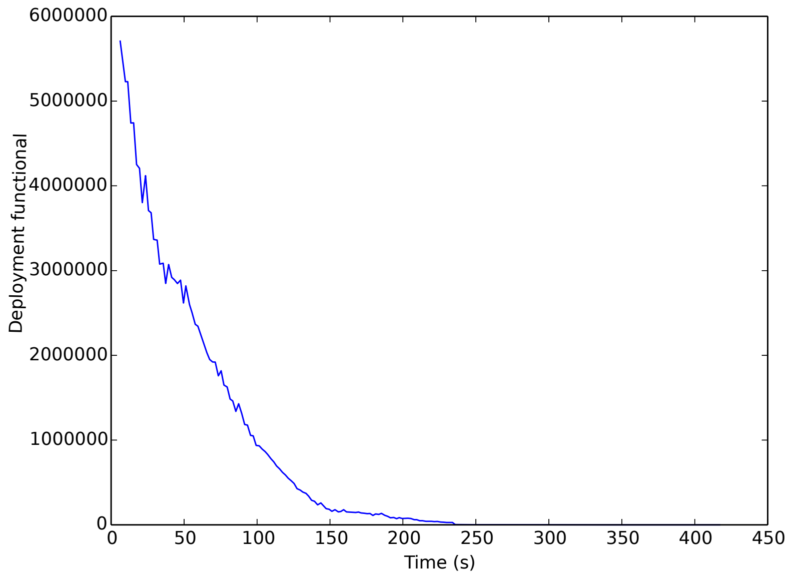


Figure 4.5: Deployment functional over the second simulation.

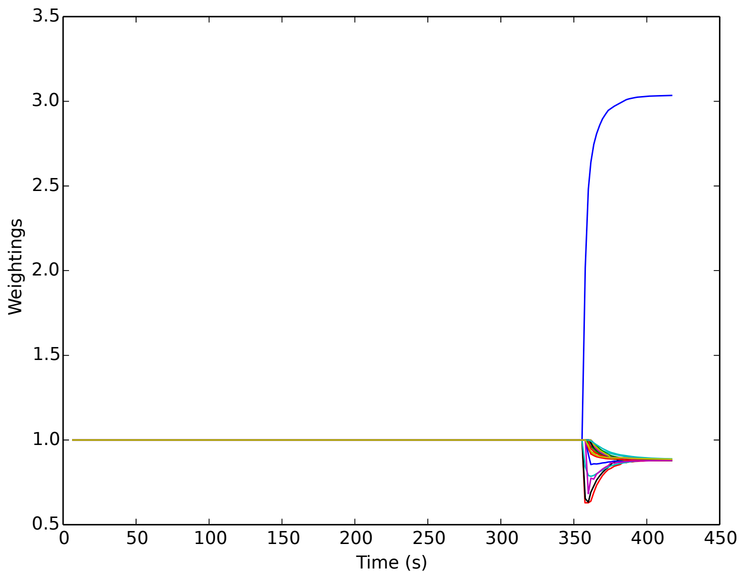


Figure 4.6: Weightings of the robots. Around $t = 350$, the operator interacts with one of the robots.

4.2 Experiments

To test the Human Swarm Interaction in a real environment, we set up experiments using a group of e-puck[2] robots (Figure 4.7). The e-puck is a small robot developed by GCtronic for research and education purposes. Among its features, it has cameras, IR sensors and

[2]e-puck robot: `http://www.e-puck.org/`

communicates via *Bluetooth* connection. We used the infrastructure from VeRLab[3] and J[4] for the experiments. For the localization, we used Augmented Reality (AR) tags on top of each e-puck robot, with a camera over the simulation area to track the position of each robot. For localization using AR tags in ROS, we used **ar_track_alvar**[5] package.

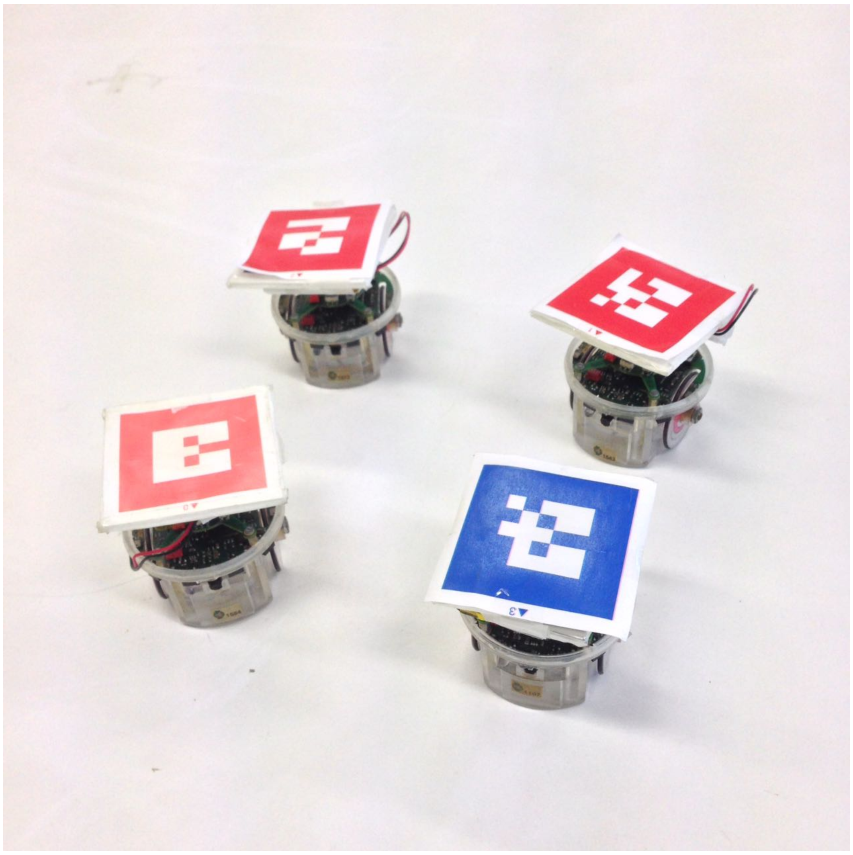


Figure 4.7: Team of e-puck robots used in the experiments.

Some changes had to be implemented to allow a smooth experiment. The e-puck robots have been in service for 9 years already at VeRLab and they are not in their best shape, especially regarding motors and battery. Their speed control is not accurate anymore: for example, giving a 0.4 forward speed makes they go forward perfectly, but sending 0.5 makes them vibrate and move in a speed lower than we could see when sending 0.4, which makes the feedback linearization controller used in the simulation unpractical, because it assumes linearity on the control response. Thus in order to deal with that, we decided to control the e-puck robots with a point-and-go position controller, in which the robot turns if the angle error is above a certain error or go forward otherwise, which is a suboptimal controlling method, but the only way that could work considering the condition of the robots.

The experiments of this section are available in a video, which is hosted online for

[3]VeRLab at UFMG: http://www.verlab.dcc.ufmg.br
[4]J at UFMG: http://www.j.dcc.ufmg.br/
[5]ar_track_alvar: http://wiki.ros.org/ar_track_alvar

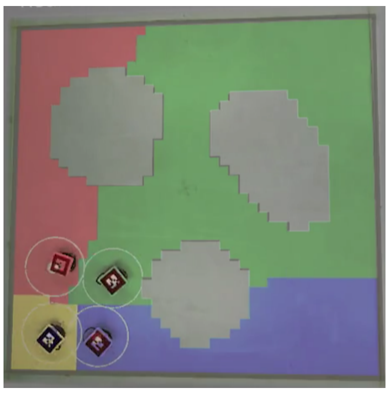
Figure 4.8: User with the VR headset during the experiments.

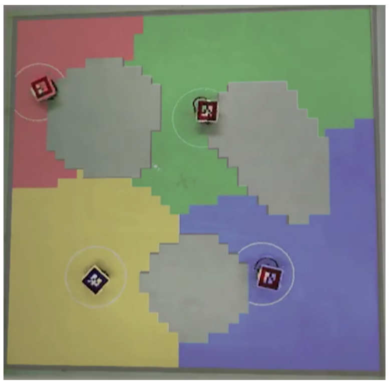
the reader convenience: https://www.youtube.com/watch?v=tMZiTOObciI. The colors of the Voronoi diagram were added later through video editing, because projecting the image over the robots during the experiment would affect the AR tag tracking.

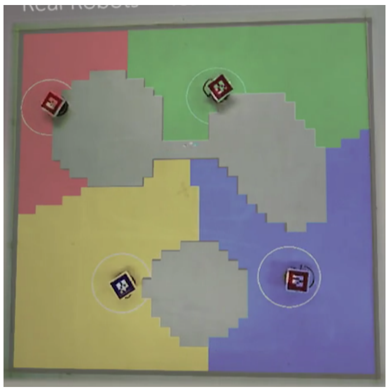
4.2.1 First experiment

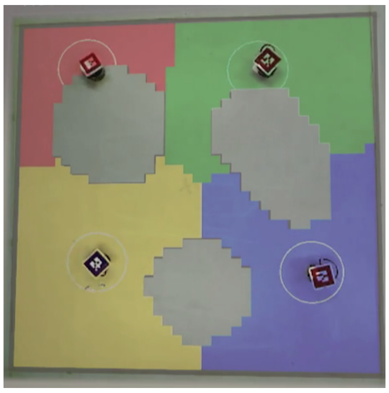
In the first experiment, robots start from the lower left corner, as in Figure 4.9a. They all start with $k_{p_i} = 1$. The human agent has the same controls as they had on the simulation results, but it let the robots converge to a optimal displacement before first interacting with them, to show the convergence of the algorithm. At $t = 135s$, the human creates obstacles, that changed the distribution of the robots. Figure 4.9b shows their displacement at that time.

At $t = 180s$, the human interacts with the yellow robot, changing its gain k_p to 3 and thus changing its weight. The weight of the robots is shown in the video as the white radius around the robot, in which the larger the circle, the greater is the value of the weight. After a while, we can see that the yellow robot not only has a larger radius, but also is covering a larger area than its neighbors. Also, because of the adaptive weight controller, the robots around the yellow robot lose weight. In Figure 4.9e, we can see the weights of each robot and how they changed after the human robot affected them. Around $t = 240s$, the human creates a Gaussian shaped density distribution function in a position centered at the map. Upon the creation of the function, a larger concentration of robots is expected in the region, which happens, but since the yellow robot has a larger weight, it dominates that region, which can be seen at Figure 4.9e. Figure 4.10 shows the $\mathcal{H}$ functional value over the experiment. Spikes were caused by the interaction with obstacles and the modification of the density distribution function.


(a) Initial robot distribution, at $t = 0s$


(b) Convergence of robots to an optimal position at $t = 135s$


(c) Human created obstacles that changed the distribution.


(d) Human removed the obstacle recently created, changing the tessellations.

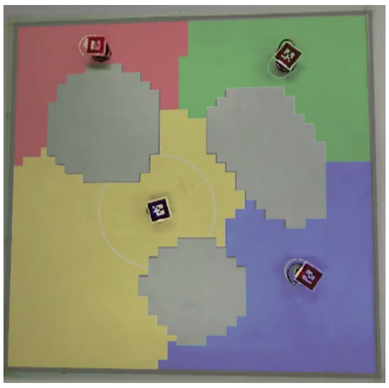

(e) Distribution of the robots around the Gaussian shaped density function placed around the center of the area, at $t = 270s$

Figure 4.9: Key steps at the first e-puck experiment.

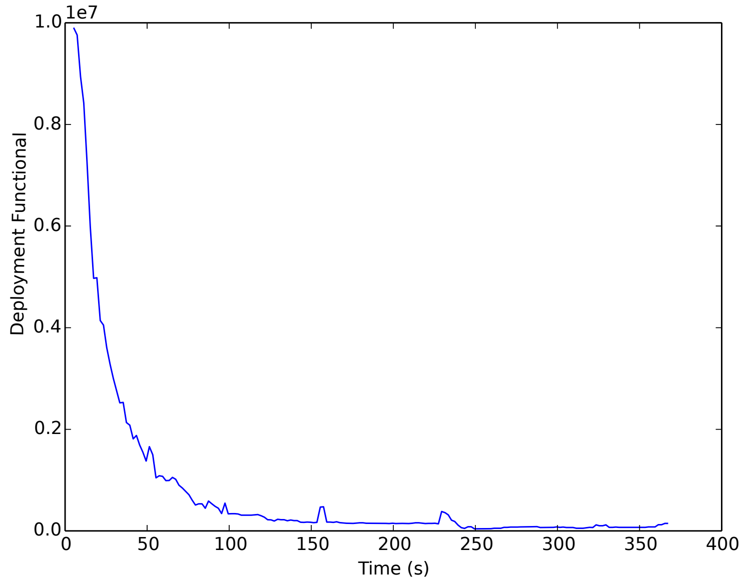


Figure 4.10: Deployment functional over the simulation.

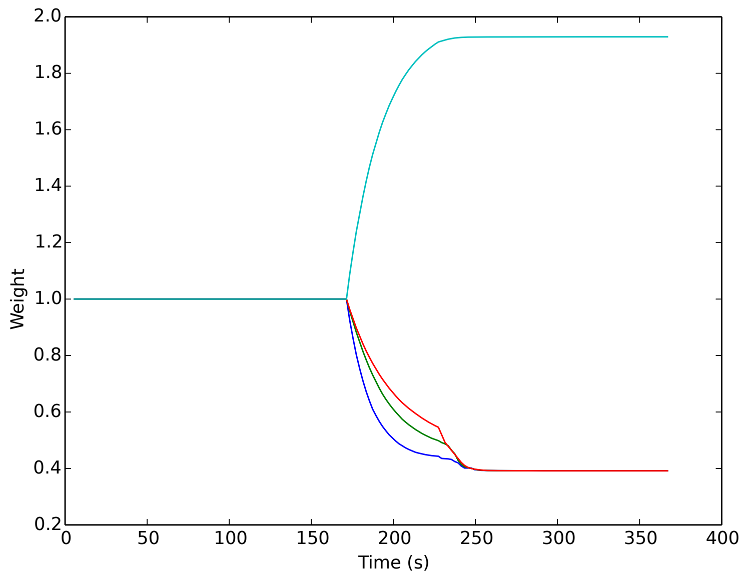
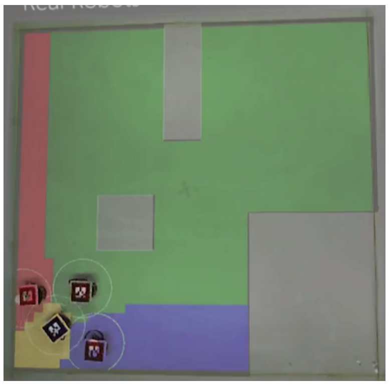


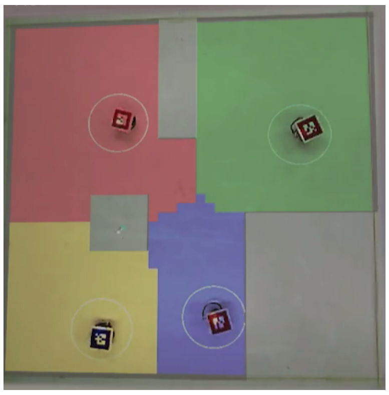
Figure 4.11: Weightings of the robots. Around $t = 150$, the human changed the gain k_{p_i} of robot 1 (green), which changed the weights of all robots.

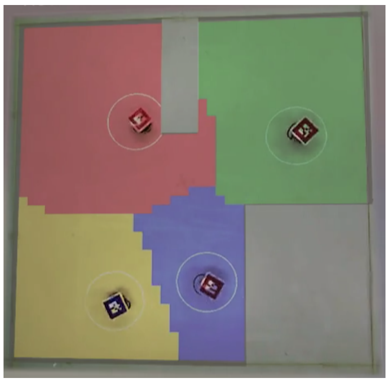
4.2.2 Second experiment

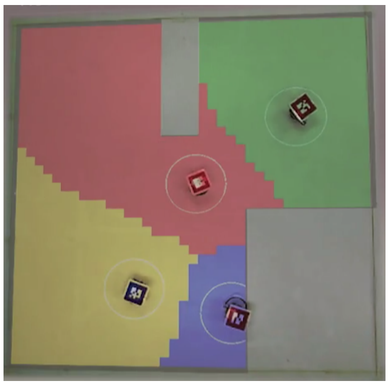
In a second experiment, we set up four robots, in the same conditions of the last experiment, but just with a different map and another sequence of interactions. Robots start from the lower left corner, as in Figure 4.12a. On $t = 115s$, they reach convergence, as in Figure 4.12b, then the human agent starts erasing an obstacle from the environment, concluding it at $t = 180s$, with final map shown in Figure 4.12c. Then the human creates a Gaussian shaped density distribution function in the center of the image, and thus the robots converge around $t = 210s$, displayed in Figure 4.12d. Then the human agent creates another Gaussian shaped density distribution function, but in the left side of the image, reaching convergence around $t = 255s$ as in Figure 4.12e. Thus, the human creates a flat distribution function, in which the robots converge around $t = 285s$ as in Figure 4.12f. At the end, the human also changes the weight of the blue robot, and the divergence of weights can be seen on Figure 4.14

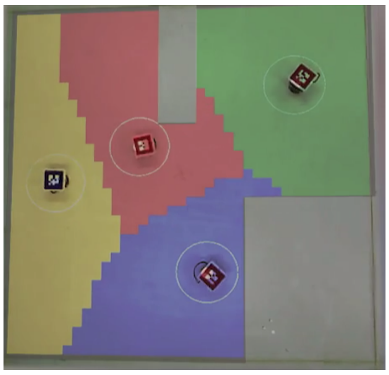
Figure 4.13 shows the $\mathcal{H}$ functional value over the experiment. This graph has a very different shape from the exponential form that we are used to see in these results. In fact, this shape happened because right at the beginning of the experiment the system suffered from localization errors, as we can see in the video. Positioning errors greatly affects the value of $\mathcal{H}$. Moreover, at the end of the experiment, we can see that the robots reduced their overall $\mathcal{H}$ functional value, thus showing that they have reached a better convergence. Spikes were caused by the changes in the density distribution. The higher steady value for $\mathcal{H}$ at the end of the experiment is because of the flat distribution created.


(a) Initial robot distribution, at $t = 0s$


(b) Convergence of robots to an optimal position at $t = 115s$


(c) Human removed an obstacle from the environment.


(d) Robots converge to a Gaussian shaped density distribution in the center of the area at $t = 210s$.


(e) Robots converge to a Gaussian shaped density distribution in the left side of the area at $t = 255s$.

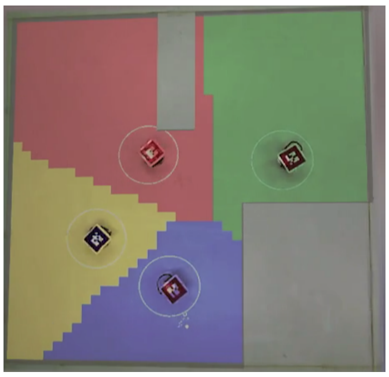

(f) Robot convergence to a flat density distribution at $t = 285s$

Figure 4.12: Key steps at the second e-puck experiment.

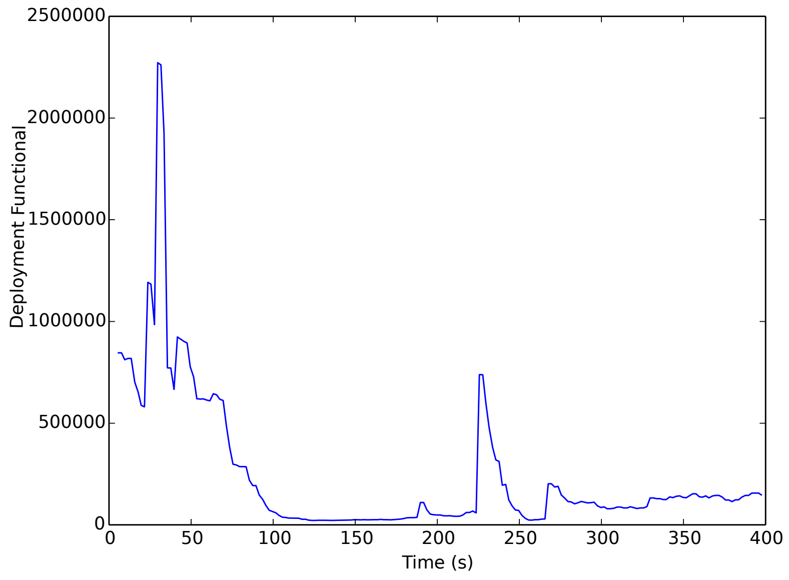


Figure 4.13: Deployment functional over the experiment.

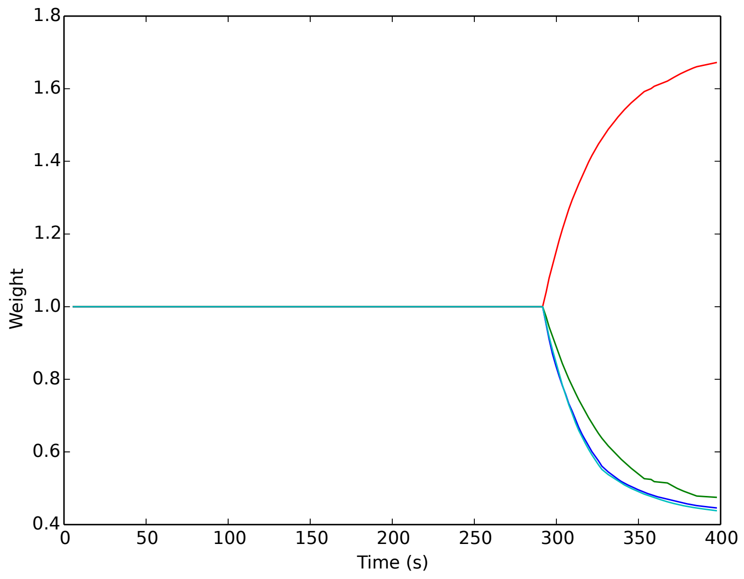


Figure 4.14: Weightings of the robots. At the end of the experiment, the human agent changed the weight of one of the robots.

4.3 Voro* simulations

In order to test the proposed algorithm, a new simulation setup was built. In order to compare the algorithms fairly and remove external interactions, the human robot interaction part was removed from this setup. First, we will show general aspects of the Voro* algorithm in a simulation, and an example of a full convergence and the diagrams seen from different robots. Secondly, we will test the Voro* algorithm with different number of robot groups to show that its order of complexity reduces when the number of robots increases. And lastly, we will compare it to the Dijkstra's-inspired algorithm.

4.3.1 Voro* behavior

Since Voro* calculates only local diagrams, the tessellations are seen differently for each robot. For exemplification of the behavior of the algorithm, we set up a simulation run using 20 robots, in a 80×80 graph and we will analyze the evolution of the algorithm over time from different robots. We have also produced a video of this simulation: `https://www.youtube.com/watch?v=8lD5Qx13bWY`.

As on the HSI simulations, we setup a simulation in which robots started from the left bottom corner. All robots have the same weight. The density distribution function is flat and equal to 1. In Figure 4.17, we can see the Voronoi diagrams built for robot 0, which has a red region. Throughout the simulation, the robot does not change much of its position, staying in the left bottom corner of the region for the whole simulation. We can see that it has only calculated a local diagram, which corresponds to a very small fraction of the whole graph. At the border of the Voronoi region, we can see some dark points, which indicate that the node has not been expanded. This happened because the red robot was a candidate for the node but it lost for another robot. When the robot loses the candidacy for a node, the counter of nodes in the priority queue reduces, and thus the counter might reach zero without the node expanding. However, as we discussed in Section 3.7.1 and we will show in simulations in the following sections, the Voronoi diagram generated for the red robot would be the same as generated by the algorithm by Bhattacharya et al. (2013).

Robot 2, in blue (Figure 4.18), had an interesting behavior. Differently from the red robot, robot 2 changed its position quite a lot, finishing the distribution with totally different neighbors from what it started. This is an interesting result because it shows that Voro* does not make any assumptions on which robots would be the neighbors of the given robots before calculating the tessellations.

Robot 10, in light green in Figure 4.19 started calculating almost the entire region. This is due to the fact that its Voronoi region expanded to one of the borders of the region, and thus allowing nodes with high cost values on the graph to be expanded. However, as the deployment converges to an optimal displacement, the area of expanded nodes drastically decreases. As a comparison, the iteration at $t = 0$ took $2.149s$ to calculate, but the last image, at $t = 441s$ took $0.176s$, improving speed by 12 times.

As we mentioned iteration times, it's also worth analyzing the iteration times throughout the simulation for each robot. Figure 4.15 shows iteration times during the simulation for all robots. We can see that some robots had an iteration time of around $2.5s$ in the beginning, and all iteration times converged to a value around $0.2s$ after $350s$. This convergence is due the fact that when the robots converge to an optimal deployment in a region that all robots have the same weight and the density distribution function is flat, robots tend to have Voronoi regions of similar sizes. Figure 4.16 shows the deployment functional value over time. We can clearly see that the robots asymptotically converged to a local minimum of the deployment functional.

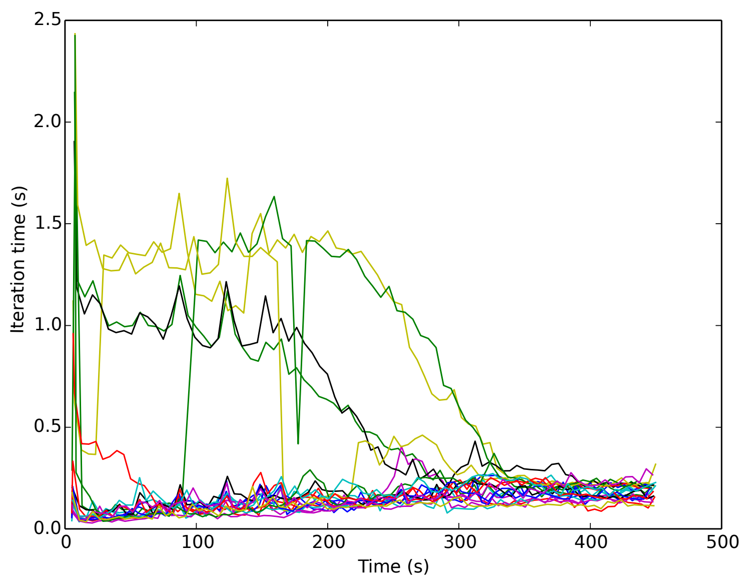


Figure 4.15: Iteration times of each robot on the simulation with 20 robots.

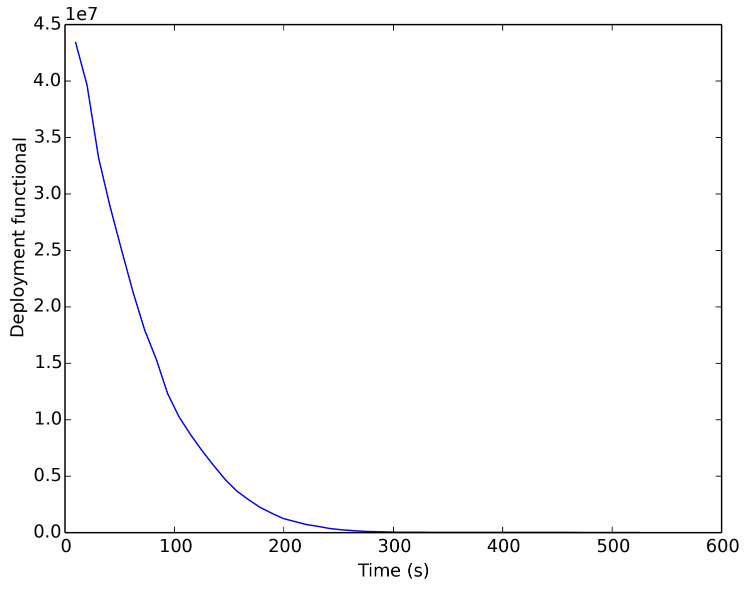
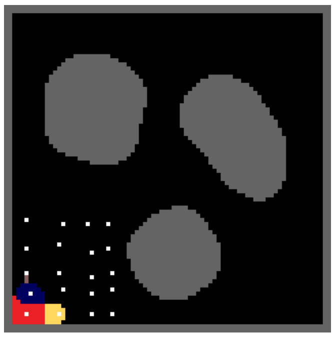
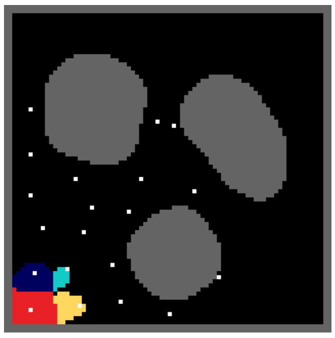
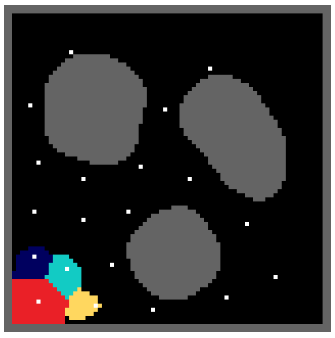


Figure 4.16: Deployment functional over time.


(a) $t = 0s$


(b) $t = 156s$


(c) $t = 238s$

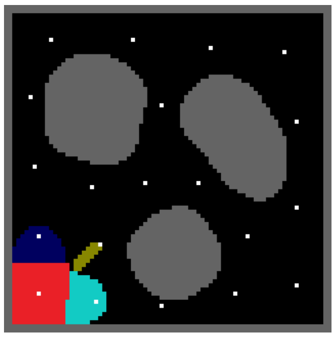
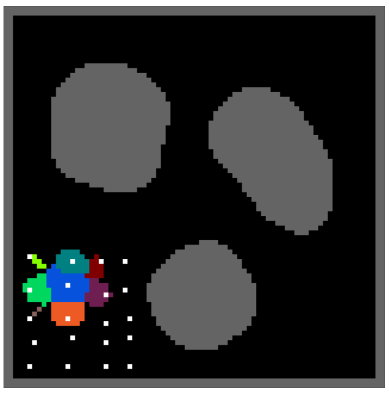
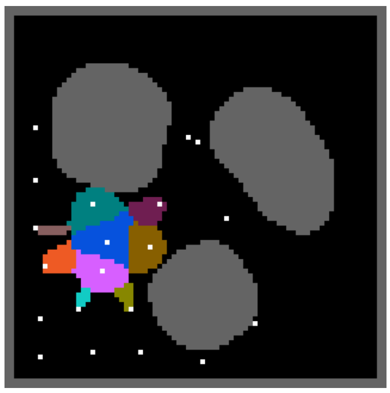
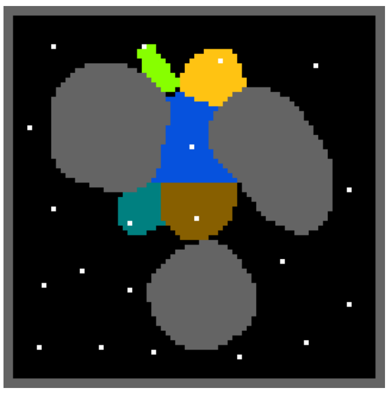

(d) $t = 442s$

Figure 4.17: Voronoi region calculated using Voro*, as seen from robot 1 (red)


(a) $t = 0s$


(b) $t = 135s$


(c) $t = 324s$

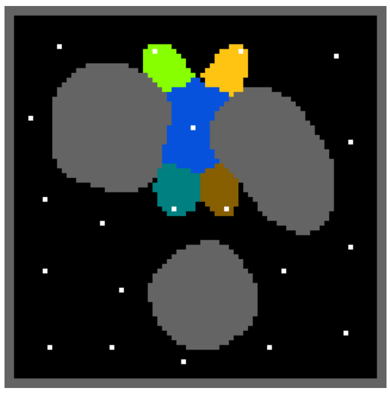

(d) $t = 438s$

Figure 4.18: Voronoi region calculated using Voro*, as seen from robot 2 (blue)

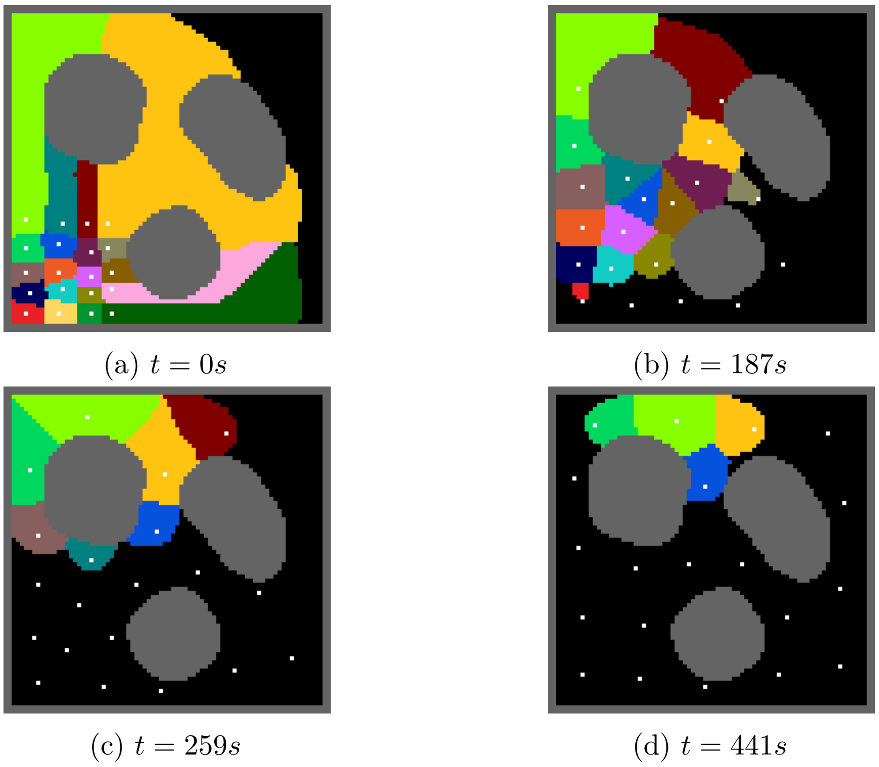

(a) $t = 0s$ (b) $t = 187s$ (c) $t = 259s$ (d) $t = 441s$

Figure 4.19: Voronoi region calculated using Voro*, as seen from robot 10 (light green)

As we mentioned before, all iteration times converged to a similar value because the density distribution was flat and all robots had the same weight. In a second experiment, we will show the behavior of the algorithm when we have heterogeneous robots and a Gaussian density distribution function.

On the second simulation, we setup 20 robots in the same map as used before. The graph used had 80×80 cells. However, robots 10 (light green), 15 (aqua) and 18 (teal) had weights equal to 2.5. All the other robots had $w = 1$. At around $t = 585s$, we have created a Gaussian shaped density distribution function centered in the middle of the region Ω.

We will discuss the results from the same robots that we have used in the previous experiment. Robot 0 (red, Figure 4.22), starts from the left bottom corner. As before, it does not change much of its position. When the operator creates the Gaussian shaped density distribution function, it attracts the robots to the center, and we see that the area of the robot increases, because the areas in the edges of the region now have a lower importance.

Robot 2 (blue, Figure 4.23) also has an interesting result. We can see that its tessellation started all black. It happened because it started next to a robot that had a larger weight than its, so this robot took all the nodes surrounding this robot. To cover this situation, we have programmed that the robot should move to the surrounding node with largest weight, making the robot go to the edge of the Voronoi region, where it can have a better chance of having its own Voronoi region. At $t = 51s$, robot 2 has its own region because both robots have moved away from one another sufficiently. At $t = 539s$ we can see that robots have converged, and so did robot 2. At $t = 936s$, we have another Voronoi diagram, which shows robot 2 has a larger area, due to the concentration of robots in the middle.

Robot 10 (light green, Figure 4.24) is one of the robots chosen to have a larger weight. We can see that in convergence ($t = 550s$) it has a region a bit larger than the other robots. His neighbors (left and right side) are the other two robots also chosen to have a larger weight. When the Gaussian shaped density distribution function is created, this robot goes to the center of it, along with the other robots. We can clearly see that the region of robot 10 is smaller than some other robots that are not close to the center. This happened because robot 10 region is more important now, and thus more robots will be attracted by this region, so Voronoi regions tend to be smaller around there. However, we can see that its region is bigger than other robots in similar positions that do not have the same weight, as the red and light blue robots that are around.

When analyzing the iteration times in Figure 4.20, we can see that they converged, as in the first experiment, but after creating the density distribution (in $t = 585s$), we can see that iteration times start to diverge. Because of the density function, there is a wider variety of Voronoi region sizes, because robots close to the center of the density distribution function will have a smaller region. Figure 4.21 shows the deployment functional during the simulation.

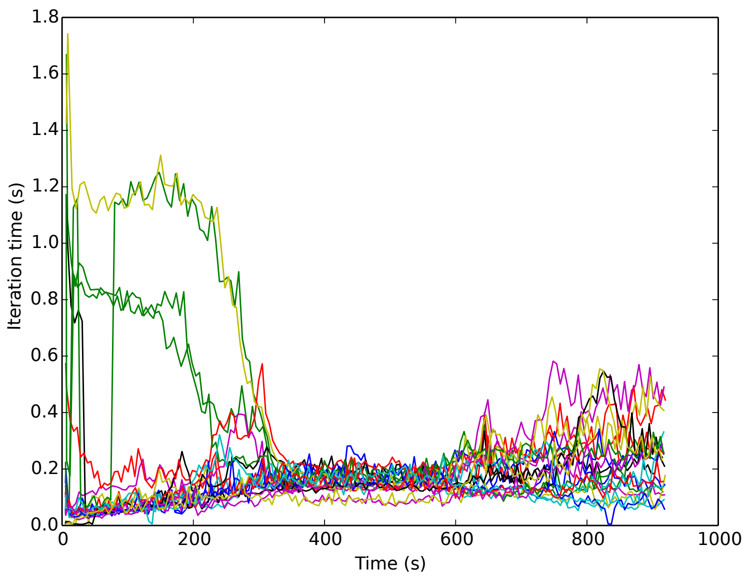


Figure 4.20: Iteration times of each robot on the simulation with 20 robots varying the density function.

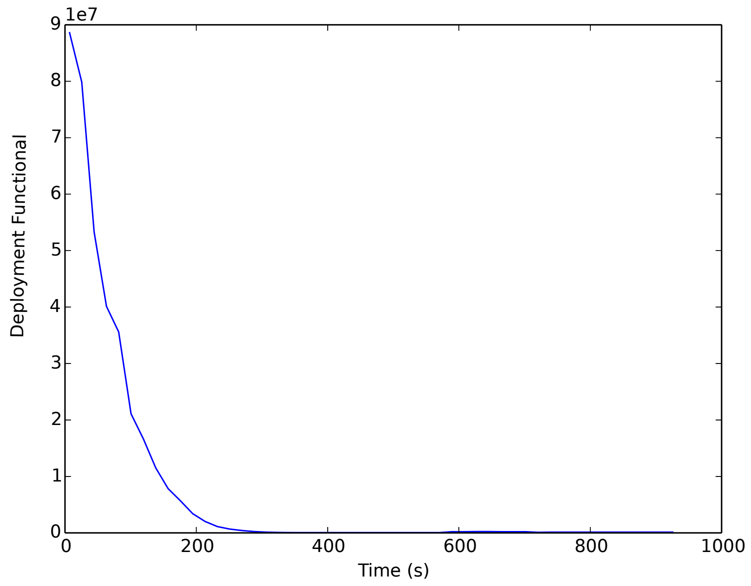
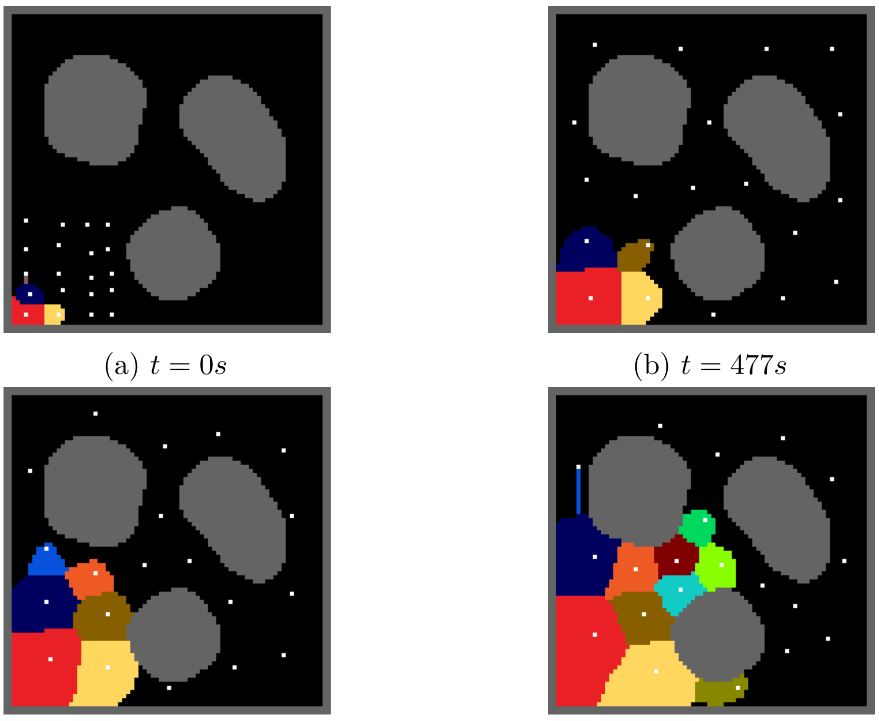


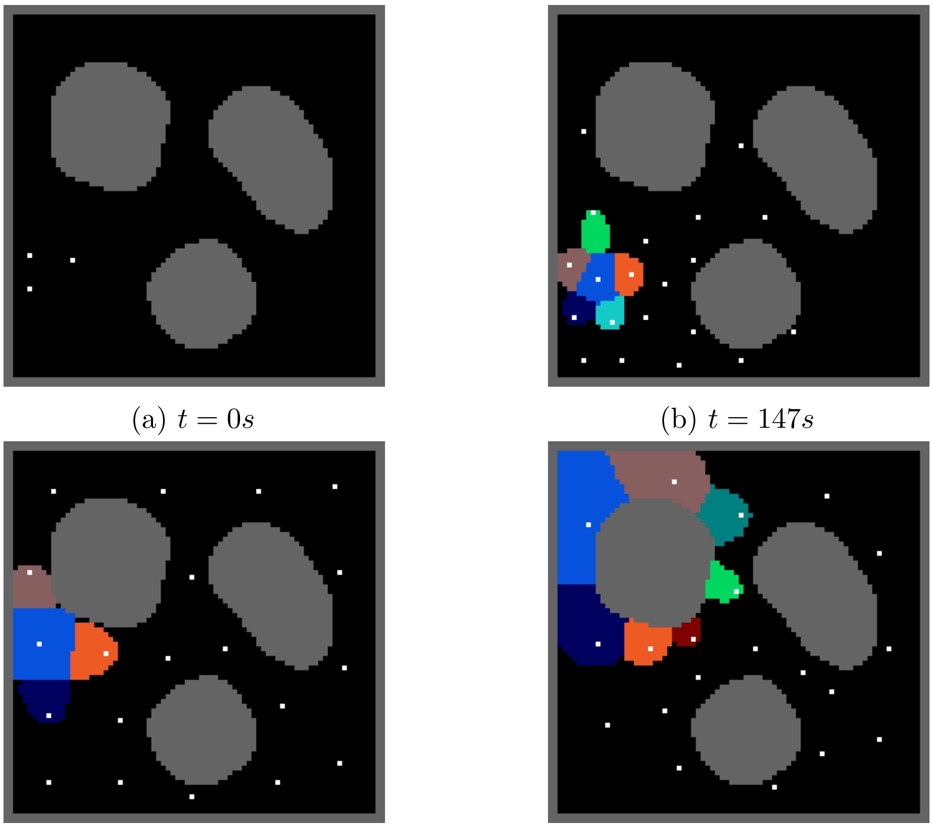
Figure 4.21: Deployment functional over time.



(a) $t = 0s$ (b) $t = 477s$

(c) $t = 657s$ (d) $t = 928s$

Figure 4.22: Voronoi region calculated using Voro* in a experiment in which we have changed the density distribution, as seen from robot 0 (red)


(a) $t = 0s$ (b) $t = 147s$

(c) $t = 539s$ (d) $t = 936s$

Figure 4.23: Voronoi region calculated using Voro*, in a experiment in which we have changed the density distribution, as seen from robot 2 (blue)

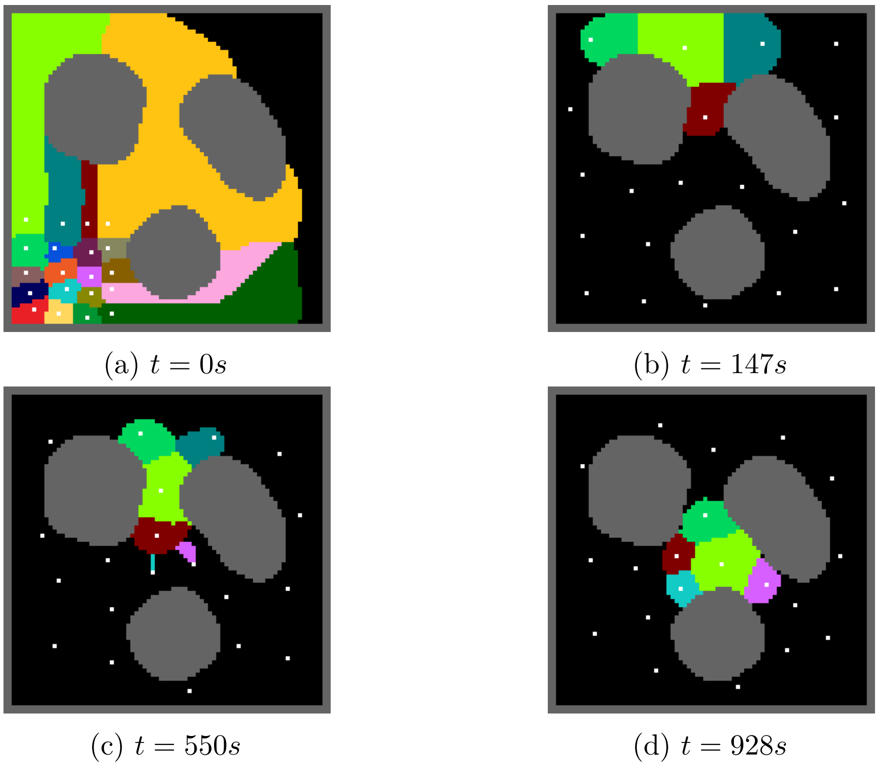

(a) $t = 0s$ (b) $t = 147s$ (c) $t = 550s$ (d) $t = 928s$

Figure 4.24: Voronoi region calculated using Voro* in a experiment in which we have changed the density distribution, as seen from robot 10 (light green)

4.3.2 Voro* in different number of robot groups

For this setup, we used again the Stage simulator as our simulation environment. We tested with 5 different groups of robots: 8, 12, 16, 20 and 24 robots. Although a even larger number of robots would be preferable, it was not possible to simulate more robots in the same computer, because with 24 robots the computer used in the simulation already reaches maximum processing power and adding more robots would artificially increase the iteration time. However, this number of robots is sufficient for showing the tendency of inversely proportional cost of the algorithm with more robots.

In the first simulation, we used the same cavern-like map used in previous experiments. We ran the simulation for around 500 seconds and two metrics were measured: the average iteration time of the Voro* algorithm throughout the whole simulation and the average time after convergence. Figure 4.25 shows the iteration time for each robot. We can see that as the robots approach convergence of the distribution, their iteration times also converge. This is due to the fact that at the beginning, there is a big difference in Voronoi diagram sizes, due to the fact that they start from a corner, but as they evenly distribute the region among themselves, the size of the diagrams is comparable. As we have briefly shown in Equation (3.7.3), the order of the complexity of Voro* is proportional to the number of vertexes, and thus, the size of the expanded nodes of the graph. Since Voro* expands the nodes in a biased fashion, the number of nodes is not only smaller, but proportional to the size of the Voronoi diagram of the current robot. The following figures show the iteration times for different group sizes. Figure 4.30 shows the average Voro* iteration times after convergence and 4.31 the average Voro* iteration times after convergence. We can see that as we increase the number of robots, the average times decrease. However, we can see that the values are tending to a stabilization. This is due to the fact that although the algorithm was running in a decentralized fashion, all robot instances were running in the same computer. Also, there is an initialization cost which was not taken into account, which has a cost proportional to the amount of robots that a given robot can communicate in the moment.

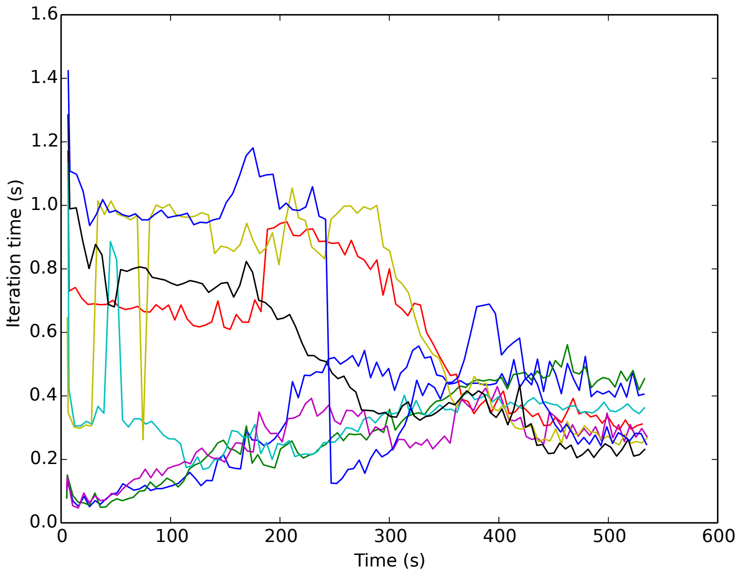


Figure 4.25: Iteration times of each robot on the simulation with 8 robots.

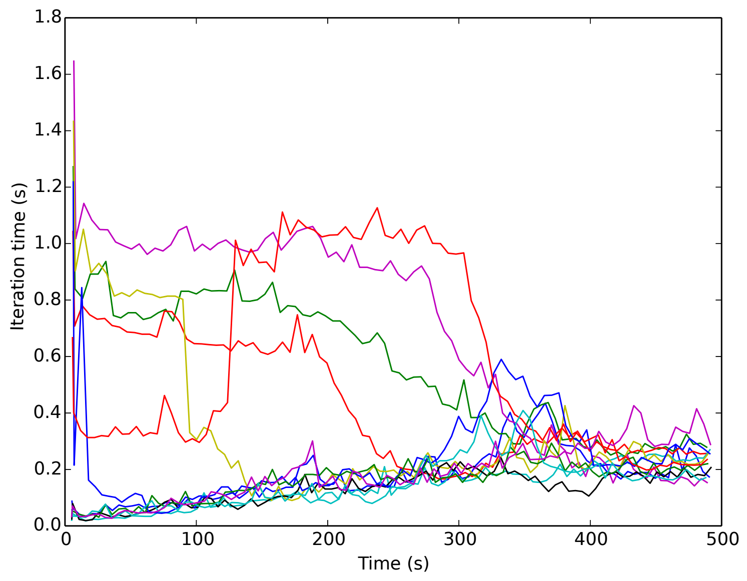


Figure 4.26: Iteration times of each robot on the simulation with 12 robots.

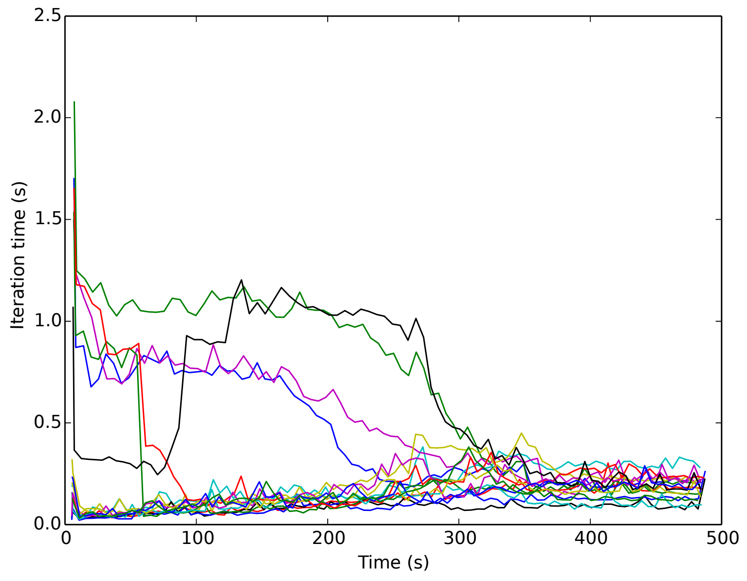


Figure 4.27: Iteration times of each robot on the simulation with 16 robots.

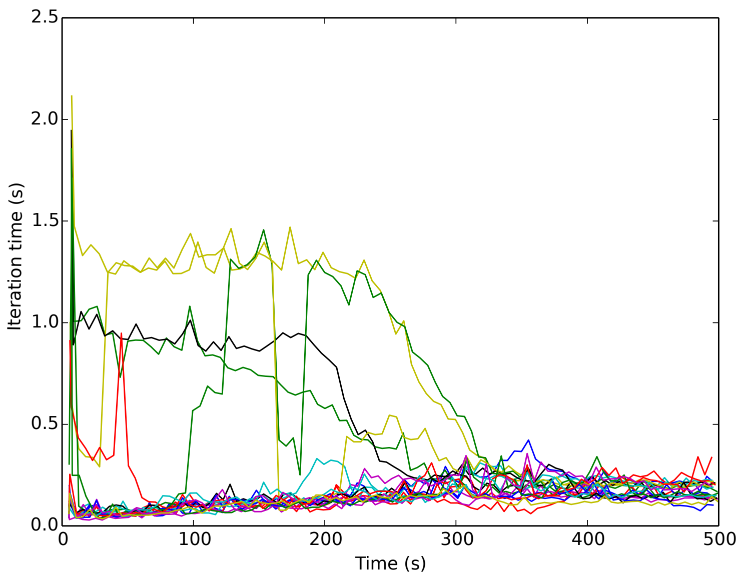


Figure 4.28: Iteration times of each robot on the simulation with 20 robots.

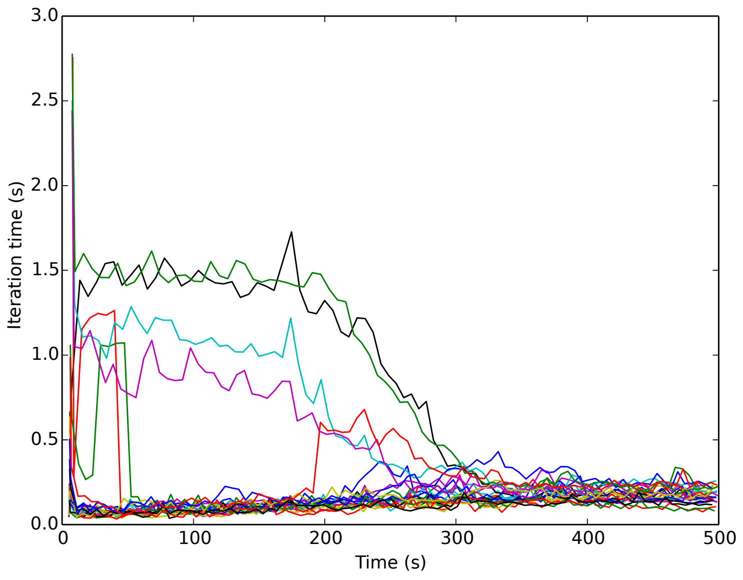


Figure 4.29: Iteration times of each robot on the simulation with 24 robots.

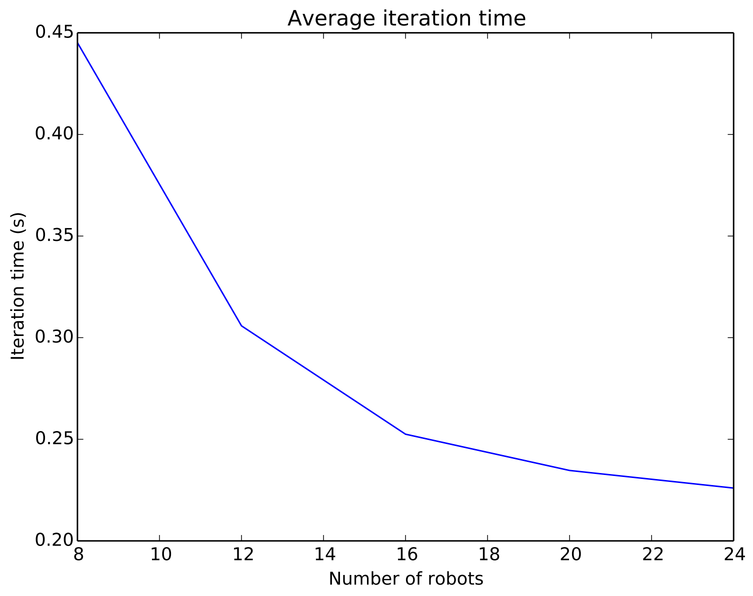


Figure 4.30: Average iteration time for the Voro* in different numbers of robot groups.

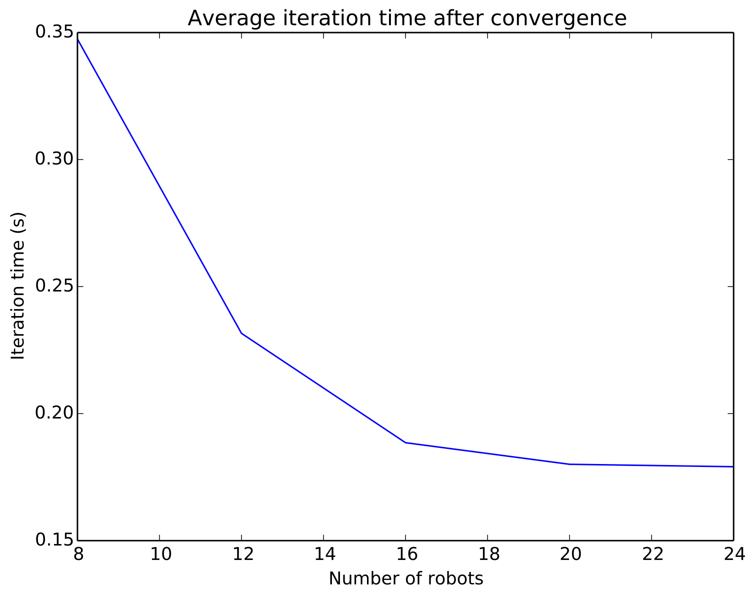
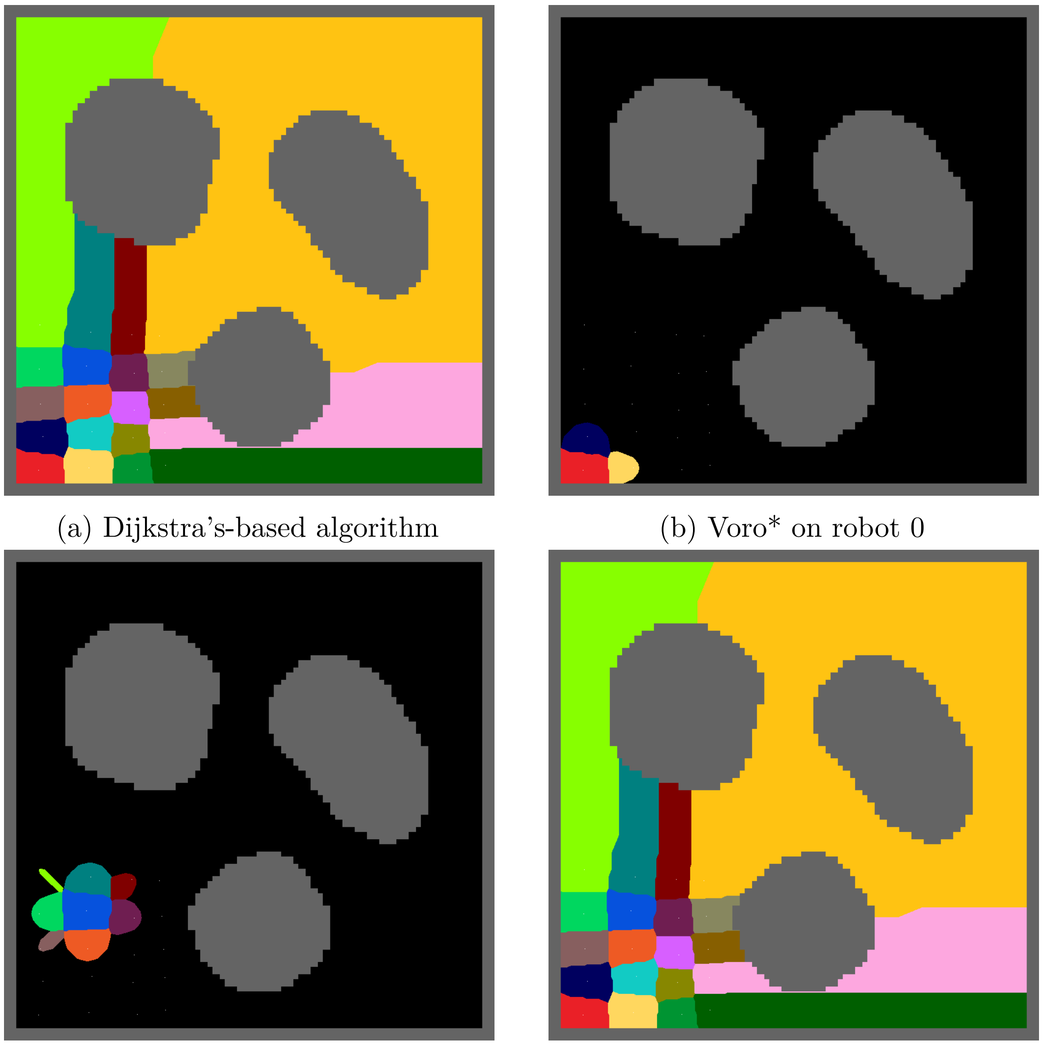


Figure 4.31: Average iteration time for Voro* after convergence in different numbers of robot groups.

4.3.3 Voro* compared with Dijkstra's-based algorithm

In this part, we will compare the performance of the Voro* algorithm with the Dijkstra's based algorithm from Bhattacharya et al. (2013) in different situations and show that the time for calculating the tessellations is smaller in Voro* and the tessellation diagrams are the same, as mathematically shown in Equation (3.7.3).

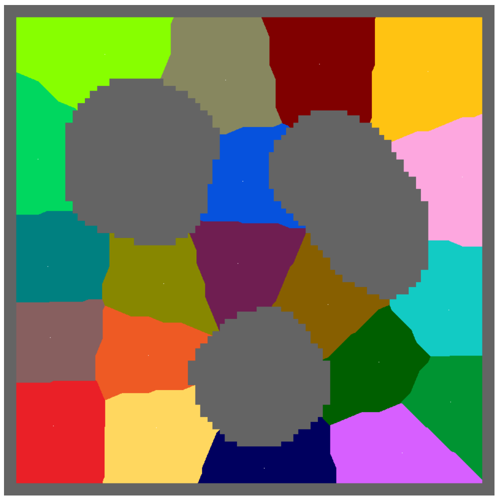
We have setup a simulation environment with 20 robots and $w = 1$ for all robots. However, to make the iteration times longer and allow a better comparison, we have used a 800×800 graph. Then, we setup the robot swarm in three different positions and ran the algorithm in the exact same positions for the Dijkstra's-based algorithm and for 3 instances of Voro*. Figure 4.32 shows the comparison. It's worth noticing that in Voro* the nodes on the graph expanded in the direction of the robots, but the formed Voronoi diagram is the same, showing empirically what we have presented in Section 3.7.1 holds. Figure 4.32a shows the diagrams of the Dijkstra's-based algorithm, that took $86.89s$ to finish. Robot 0 (red) has a much smaller region and finished in $2.94s$. Robot 2 (blue) took $6.08s$ to finish its iteration. Robot 3 (yellow) region is the perfect example of the worst-case scenario in Voro*. Because of the length of its Voronoi region, it had to calculate all the other Voronoi regions. Its iteration took longer than the Dijkstra's-based algorithm, running for $136.79s$. This longer time is due to the extra comparisons that the Voro* has to do every iteration and also because Voro* ends up inserting more nodes in the priority queue that end up unused. As our priority queue is arranged as a heap, insertions have a $O(log(N))$ complexity, where N is the size of the priority queue, thus, in a big graph as this one, Voro* does worse than the previous algorithm. Figure 4.32d has robot 3 tessellations, that show that the diagrams of all robots are exactly the same found in Figure 4.32a, also showing that the introduction of the heuristic function does not distorts the Voronoi diagrams.

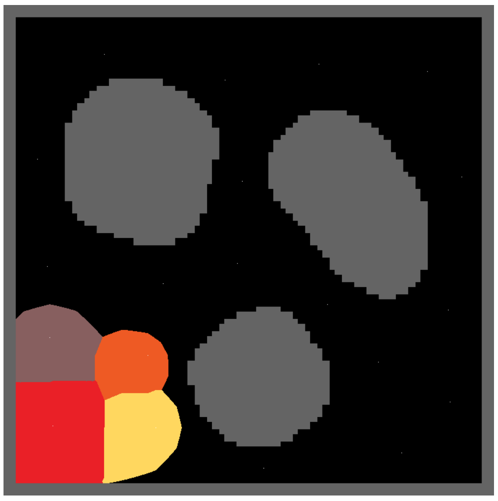

(a) Dijkstra's-based algorithm (b) Voro* on robot 0

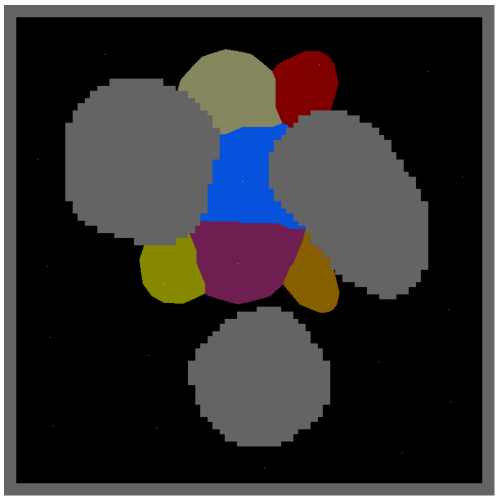
(c) Voro* on robot 2 (d) Voro* on robot 3

Figure 4.32: Comparison of the tessellations generated with Voro* when all robots are in the same corner.

Then, we setup the robots in a different deployment, close from which they would reach an optimal configuration. Figure 4.33 compare the tessellations as calculated from the algorithm from Bhattacharya et al. (2013) and Voro* as seen from the same robots as we used before. Again, it's worth noticing that the Voronoi diagrams are exactly the same when we compare them in Figure 4.33a. The algorithm in Bhattacharya et al. (2013) took 86.26s to finish, which is very close to the one found in the previous example. Robot 0 took 25.96s, robot 2 21.25s and robot 3 took 20.55s.


(a) Dijkstra's-based algorithm


(b) Voro* on robot 0


(c) Voro* on robot 2

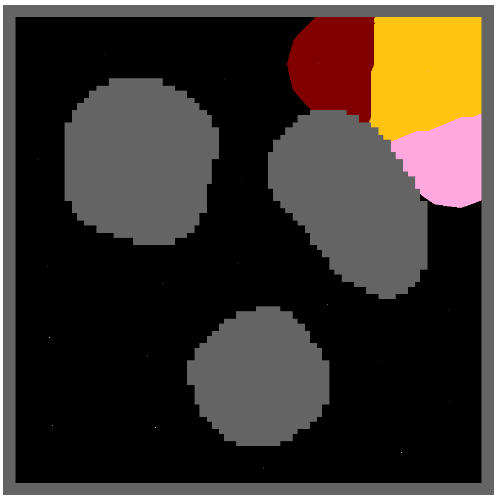

(d) Voro* on robot 3

Figure 4.33: Comparison of the tessellations generated with Voro* when all robots are close to convergence.

Thus, we can see that Voro* is more efficient when considering a decentralized strategy, in which each robot would need to calculate their control law by themselves, instead of a central computation.

5
Conclusions and Future Work

In this work, we have explored human-swarm interfaces and proposed one for interfacing with a group of robots in a coverage control abstraction. The abstraction using coverage control is interesting because it allowed operators to dramatically influence the positioning and behavior of the robotic swarm using just a few parameters. We used as an interface a Virtual Reality headset, which enabled operators to have a deep and intuitive experience while interacting with the robots. Virtual Reality also allows some commands that are much more immersible than joysticks, tablets, haptic interfaces and some other interfaces used in previous works in the field of Human-Swarm Interaction. The operators could interact with the swarm by different actions, like creating functions that could attract more robots to a specific position, interacting with the obstacles by adding or removing them, move around freely using a teleportation and fly-over feature or selecting specific robots to increase their coverage weight. Simulations and experiments with real robots proved the strategy and allowed the human agents to freely interact with the robots. We could show that interacting with a group of robots in VR can be beneficial because it gives a very deep level of immersion, but at the same time keeping the human physically isolated, which could be applied for example for teleoperating a swarm of robots in hazardous environments or simply too far away from the human operator. Also, it's worth mentioning that since we've used common libraries and abstraction layers, like Unity and VRTK, and made it available online on GitHub, one could easily adapt the current software to work with any other VR headset other than *Oculus Rift*. Also, other contributions of this work are the open software *Oculus Rift* driver for ROS. A disadvantage of VR is that it requires a high and steady throughput of data to render the information in the headset for the user.

The control law used for the robots to improve the distribution allowed them to converge to optimal displacements, as we have seen on the Results chapter, however it must be noted that a gradient-descent optimization is used, and thus, the deployment is faded to converge to a local minimum. The possibility of this happening is very related to the topology of the environment. One of the future works could investigate for example the possibility of using a sampling based control law that could help the swarm converging to a global minimum.

Although we have done experiments, it's worth noticing that the methods for acquiring the position of the robots and the map are not practical in a real world product application. To apply the developed algorithm in such conditions, another source of position is required. If the robots are outdoors, we could use GPS. If indoors, we could have a localization method based on mapping, as Monte Carlo Localization (Fox et al. (2001)). The map could be acquired before running the algorithm using SLAM (Ayache & Faugeras (1988)) or even be constructed during the execution of the coverage control algorithm, using multi-robot SLAM techniques, as developed in recent works such as Atanasov et al. (2015).

Later on, we have shown the Voro* algorithm, that calculates Voronoi diagrams locally and in some conditions greatly improves algorithms previously proposed that uses Dijkstra's inspired algorithms for graph search to build the Voronoi tessellations (Bhattacharya et al. (2013)).

In the same way we have expanded a Dijkstra's-based algorithm to an A*-based algorithm. We believe that applying sampling-based algorithms for path-planning, like RRT*, for calculating Voronoi tessellations could be possible as well. This could enable faster calculations for the diagrams. Sampling-based algorithms could also help to search for the global minimum of the deployment functional. This also can be an interesting future work that needs some deeper investigation.

It may also seem curious for the reader why the Voro* was not used in the HSI simulations and experiments. There are some reasons for it. First, and most relevant, to generate the image that the operator sees in VR, we need to compute all the regions from all robots, thus the Dijkstra's-based Voronoi algorithm is necessary. Another reason is that Voro* only makes difference when the group of robots has a substantial amount of robots. Because of hardware limitations, we have only ran experiments with 4 robots, which would make the results from Voro* identical to the former algorithm.

Another point of the Voro* algorithm is that although we claim that it is faster than previous algorithms, and we discuss about the iteration time during steady time, this metric is irrelevant in some cases. Depending on the task of the robots, after converging the algorithm could be aborted. However, in some cases that the algorithm needs to be running even after convergence, for example when the density function or obstacles can vary throughout time and the robots need to react to it, Voro* can be superior to the previous algorithms.

Some previous works in HSI have also involved different human operators and analyzed their reaction with the interface and delivered metrics about it. A statistical analysis of our interface with many different operators could also be an interesting future work.

Bibliography

Atanasov, N., Ny, J. L., Daniilidis, K., & Pappas, G. J. (2015). Decentralized active information acquisition: Theory and application to multi-robot SLAM. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 4775–4782).

Aurenhammer, F. (1991). Voronoi diagrams - a survey of fundamental geometric data structure. *ACM Computing Surveys*, 23, 345–405.

Ayache, N. & Faugeras, O. D. (1988). Building, Registrating, and Fusing Noisy Visual Maps. *Int. J. Rob. Res.*, 7(6), 45–65.

Becker, A., Ertel, C., & McLurkin, J. (2014). Crowdsourcing swarm manipulation experiments: A massive online user study with large swarms of simple robots. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 2825–2830).

Bhattacharya, S., Ghrist, R., & Kumar, V. (2013). Multi-robot coverage and exploration on Riemannian manifolds with boundaries. *Int. J. Rob. Res.*, 33(1), 113–137.

Bischoff, M. (2018). ROS#. https://github.com/siemens/ros-sharp. [Online; accessed 22. Apr. 2018].

Breitenmoser, A., Schwager, M., Metzger, J. C., Siegwart, R., & Rus, D. (2010). Voronoi coverage of non-convex environments with a group of networked robots. *2010 IEEE International Conference on Robotics and Automation*, (pp. 4982–4989).

Cameron, J. (1985). *The Terminator*.

CBS Insights (2018). 44 corporations working on autonomous vehicles. https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list. [Online; accessed 22. Apr. 2018].

Choi, W. S., Kwon, H. S., Park, W. K., Park, J. H., Lee, S. B., & Heo, S. Y. (2018). Effects of robot-mediated gait training combined with virtual reality system. *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, (pp. 1–2).

CNN (2014). MIT unveils swimming, oil-cleaning robots. http://edition.cnn.com/2010/TECH/innovation/08/26/mit.oil.robot/index.html. [Online; accessed 22. Apr. 2018].

Cortes, J., Martinez, S., Karatas, T., & Bullo, F. (2004). Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2), 243–255.

Debernard, S., Chauvin, C., Pokam, R., & Langlois, S. (2016). Designing human-machine interface for autonomous vehicles. *IFAC-PapersOnLine*, 49, 609–614.

Diaz-Mercado, Y., Lee, S. G., & Egerstedt, M. (2015). Distributed dynamic density coverage for human-swarm interactions. *2015 American Control Conference (ACC)*, (pp. 353–358).

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numer. Math.*, 1(1), 269–271.

Du, J., Sheng, W., & Liu, M. (2016). Human-guided robot 3D mapping using virtual reality technology. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 4624–4629).

El-Shawa, S., Kraemer, N., Sheikholeslami, S., Mead, R., & Croft, E. A. (2017). "Is this the real life? Is this just fantasy?": Human proxemic preferences for recognizing robot gestures in physical reality and virtual reality. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 341–348).

Fox, D., Thrun, S., Burgard, W., & Dellaert, F. (2001). Particle Filters for Mobile Robot Localization. *SpringerLink*, (pp. 401–428).

Gioioso, G., Franchi, A., Salvietti, G., Scheggi, S., & Prattichizzo, D. (2014). The flying hand: A formation of UAVs for cooperative aerial tele-manipulation. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 4335–4341).

Guerin, K. R., Riedel, S. D., Bohren, J., & Hager, G. D. (2014). Adjutant: A framework for flexible human-machine collaborative systems. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 1392–1399).

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100–107.

Hausknecht, M., Au, T.-C., & Stone, P. (2011). Autonomous intersection management: Multi-intersection optimization. *IEEE International Conference on Intelligent Robots and Systems*, (pp. 4581–4586).

Hönig, W., Milanes, C., Scaria, L., Phan, T., Bolas, M., & Ayanian, N. (2015). Mixed reality for robotics. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 5382–5387).

IEEE Spectrum (2004a). Can the U.S. Military Combat the Coming Swarm of Weaponized Drones? https://spectrum.ieee.org/tech-talk/aerospace/military/us-military-evaluates-antidrone-tech. Online; accessed 22. Apr. 2018.

IEEE Spectrum (2004b). Debating slaughterbots and the future of autonomous weapons. https://spectrum.ieee.org/automaton/robotics/military-robots/debating-slaughterbots. Online; accessed: 22 Apr. 2018.

IEEE Spectrum (2014). U.S. Navy Tests Robot Boat Swarm to Overwhelm Enemies. https://spectrum.ieee.org/automaton/robotics/military-robots/us-navy-robot-boat-swarm. [Online; accessed 22. Apr. 2018].

Jann, M., Anavatti, S., & Biswas, S. (2017). Path planning for multi-vehicle autonomous swarms in dynamic environment. (pp. 48–53).

Kamon, I., Rimon, E., & Rivlin, E. (1998). TangentBug: A Range-Sensor-Based Navigation Algorithm. *Int. J. Rob. Res.*, 17(9), 934–953.

Kantaros, Y., Thanou, M., & Tzes, A. (2014). Visibility-oriented coverage control of mobile robotic networks on non-convex regions. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 1126–1131).

Klamroth-Marganska, V., Baur, K., Nagle, A., Riener, R., & Toigo, M. (2017). Strength training of the upper extremity in virtual reality with an exoskeleton robot. *2017 International Conference on Virtual Rehabilitation (ICVR)*, (pp. 1–6).

Kolling, A., Sycara, K., Nunnally, S., & Lewis, M. (2013). Human Swarm Interaction: An Experimental Study of Two Types of Interaction with Foraging Swarms. *Journal of Human-Robot Interaction*, 2(2), 103–128.

Kolling, A., Walker, P., Chakraborty, N., Sycara, K., & Lewis, M. (2016). Human Interaction With Robot Swarms: A Survey. *IEEE Trans. Hum.-Mach. Syst.*, 46(1), 9–26.

Kun, A. L. (2018). Human-Machine Interaction for Vehicles: Review and Outlook. *HCI*, 11(4), 201–293.

Lee, D., Franchi, A., Giordano, P. R., Son, H. I., & Bülthoff, H. H. (2011). Haptic teleoperation of multiple unmanned aerial vehicles over the internet. *2011 IEEE International Conference on Robotics and Automation*, (pp. 1341–1347).

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2), 129–137.

Lottes, P., Khanna, R., Pfeifer, J., Siegwart, R., & Stachniss, C. (2017). Uav-based crop and weed classification for smart farming.

Michael, N. & Kumar, V. (2009). Planning and Control of Ensembles of Robots with Non-holonomic Constraints. *Int. J. Rob. Res.*, 28(8), 962–975.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., & Martinoli, A. (2009). The e-puck, a Robot Designed for Education in Engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1(1), 59–65.

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. W., Floreano, D., Deneubourg, J.-L., Nolfi, S., Gambardella, L. M., & Dorigo, M. (2004). Swarm-Bot: A New Distributed Robotic Concept. *Autonomous Robots*, 17(2), 193–221.

Nigolian, V., Mutlu, M., Hauser, S., Bernardino, A., & Ijspeert, A. (2017). Self-reconfigurable modular robot interface using virtual reality: Arrangement of furniture made out of roombots modules. *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, (pp. 772–778).

Nunnally, S., Walker, P., Chakraborty, N., Lewis, M., & Sycara, K. (2013). Using Coverage for Measuring the Effect of Haptic Feedback in Human Robotic Swarm Interaction. *2013 IEEE International Conference on Systems, Man, and Cybernetics*, (pp. 516–521).

Pierson, A., Figueiredo, L. C., Pimenta, L. C., & Schwager, M. (2017). Adapting to sensing and actuation variations in multi-robot coverage. *Int. J. Rob. Res.*, 36(3), 337–354.

Pierson, A., Figueiredo, L. C., Pimenta, L. C. A., & Schwager, M. (2015). Adapting to performance variations in multi-robot coverage. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 415–420).

Pimenta, L. C. A., Kumar, V., Mesquita, R. C., & Pereira, G. A. S. (2008). Sensing and coverage for a network of heterogeneous robots. *2008 47th IEEE Conference on Decision and Control*, (pp. 3947–3952).

Proyas, A. (2004). *I, Robot*. 20th Century Fox.

Rezeck, P. A. F., Azpurua, H., & Chaimowicz, L. (2017). HeRo: An open platform for robotics research and education. *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, (pp. 1–6).

Rubenstein, M., Ahler, C., & Nagpal, R. (2012). Kilobot: A low cost scalable robot system for collective behaviors. *2012 IEEE International Conference on Robotics and Automation*, (pp. 3293–3298).

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198), 795–799.

Ruckelshausen, A., Biber, P., Dorna, M., Gremmes, H., Klose, R., Linz, A., Rahe, R., Resch, R., Thiel, M., Trautz, D., & Weiss, U. (2009). Bonirob: An autonomous field robot platform for individual plant phenotyping. *Precision Agriculture*, 9, 841–847.

SEI Insights (2004). Army robotics in the military. https://insights.sei.cmu.edu/sei_blog/2017/06/army-robotics-in-the-military.html. Online; accessed: 22 Apr. 2018.

Setter, T., Kawashima, H., & Egerstedt, M. (2015). Team-level properties for haptic human-swarm interactions. *2015 American Control Conference (ACC)*, (pp. 453–458).

Sharon, G., Albert, M., Rambha, T., Boyles, S., & Stone, P. (2018). Traffic optimization for a mixture of self-interested and compliant agents. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*.

Spröwitz, A., Moeckel, R., Vespignani, M., Bonardi, S., & Ijspeert, A. J. (2014). Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot. *Rob. Auton. Syst.*, 62(7), 1016–1033.

Tan, Y. & Zheng, Z.-y. (2013). Research Advance in Swarm Robotics. *Defence Technology*, 9(1), 18–39.

Wachowski, L. & Wachowski, L. (1999). The Matrix.

Yamaguchi, S. P., Karolonek, F., Emaru, T., Kobayashi, Y., & Uhl, T. (2017). Autonomous position control of multi-unmanned aerial vehicle network designed for long range wireless data transmission. *2017 IEEE/SICE International Symposium on System Integration (SII)*, (pp. 127–132).