UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DEVELOPMENT OF A MULTI-ROBOT PATH PLANNING ALGORITHM IN DYNAMIC ENVIRONMENT

AUTHOR: ESPERANZA GARCÍA MARTÍNEZ

SUPERVISOR: DR LYUBA ALBOUL

Academic year: 2017-18

Abstract

In this dissertation a path planning algorithm for mobile robots is implemented using Artificial Potential Fields (APF) to avoid different types of dynamic and static obstacles and its performance is proven in simulation using MatLab.

Mobile robotics is a very popular area of development and research nowadays due to its interesting applications such as space exploration, air and water monitoring, autonomous vehicles or search and rescue. Swarm robotics is a new and promising branch of knowledge in this field. It is inspired by social insects and their behaviour. The idea behind this is that a large multi-robot system can perform tasks more efficiently, robustly and quickly than a single and more complex mobile robot.

Artificial Potential Fields is one of the most popular path planning methods due to its simplicity, mathematical elegance and good results. However, this method has inherent problems such as local minima. The basis of APF method is that robots are considered particles moving in the virtual potential field. Then, the obstacles are assigned a potential field that repel the robot and the goal is given a potential field that attract it.

The main aim of this project is to achieve a successful navigation for three robots in a known environment using APFs. In order to simulate a situation that could be close to a real-world application, the environment has been inspired by a warehouse where the robots are performing logistic tasks. Different types of test and simulations have been done to test the features of the method and its global performance.

Table of content

Table of figures

CHAPTER ONE:
Introduction

1.1. Background

Robotics play a key role in the functioning of our lives. They are changing the way people live and work. Robotics is a science that has achieved a vertiginous development in the last half century, from the first industrial robots that were conceived to assist or replace humans in production chains, until the most complex and sophisticated robots that exist nowadays that can perform surgeries, treat toxic waste, or perform search and rescue tasks.

Mobile robotics is a very popular area of development and research in the field due to all their interesting applications such as space exploration, air and water monitoring, or autonomous vehicles, among many others. The aim is to achieve a great degree of autonomy which means that robots must be able to get the information they need from the environment, process it, and make decisions on their own.

From one mobile robot to thousands of them, now the attention is focused on multi-robot systems, also called swarm robotics. The idea behind this is that a large multi-robot system is able to perform tasks more efficiently, robustly and quickly. Some of the potential applications are security and surveillance, exploration and mapping, or search and rescue. For example, the GUARDIANS is a research project developed to assist and safeguard a firefighter (Alboul et al.,2011). The swarm of robots provide localisation information and warnings about chemicals while it maintains the communication between them.

Figure 1: Robots assisting human navigation

Furthermore, there are robots specifically designed for research in swarms like Kilobots (K-Team,2018). As they describe it "The Kilobot is designed to make tests of collective algorithms on hundreds or thousands of robots accessible to robotics researchers". It is a low-cost and easy-to-use robotic system. Rubenstein et al. (2014) used a swarm of 1024 Kilobots to prove how they can self-organize and imitate complex shapes.

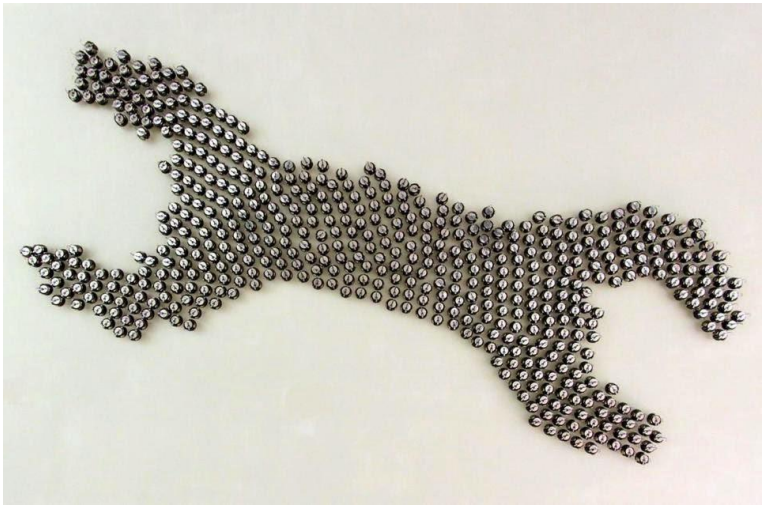


Figure 2: Swarm of Kilobots.

Another example is Robofly that is the first robotic insect that flies on its own. It has been developed in the University of Washington, and now it is in early stages of research, but it has great potential. Its creators think that in the future, many Robofly could help monitoring crops or sources of contamination, detecting leaks in pipelines and refineries, and search and rescue.

Figure 3: Robofly

1.2. Motivation

Mobile and swarm robotics are promising fields with interesting and useful applications, as it has been shown before. Mobile robots were created to fill the lack of mobility of industrial robots and its advantages, apart from mobility, are autonomy to navigate and make decisions, versatility to adapt to different environments, and wide range of robot typologies to fit different applications like humanoid, drone or wheeled robots. In the case of the swarm, it provides robustness, redundancy, and scalability to the system. Furthermore, it is low cost due to the simplicity of the

robots that conform the swam. However, as with any other area of research, both present some challenges to overcome.

To begin with, mobile robots have been developed for more than thirty years, however they still offer a wide field for development and innovation. The main goal of a mobile robot is to accomplish a robust navigation therefore it must be safe, efficient and optimal. To achieve this goal a multirobot system have tree main challenges:

- Locomotion: This issue can be divided in two parts. On the one hand, the locomotion mechanisms that enable the robot to navigate. It is referred to the physical components of the robot that enable the movement such as legs for walking, wheels to roll or helix to fly. The physical locomotion mechanism of the robot must be chosen in other to fit its application. On the other hand, there is the path planning that regards to the algorithms used to create the routes to reach the goal safely, avoiding obstacles and following the optimal path. There are many algorithms developed to accomplish this purpose but all of them can be improved or adapted.
- Perception: It concerns the selection and development of sensors because they play a key role due to through them the robot can perceive the environment. This issue also concerns how the information is interpreted by the robot.
- Navigation: It can be understood as the union of locomotion and perception towards the goal. This is about the global performance of the robot.

Next, swarm robotics is a young field therefore it is still the research stage. For this reason, the problems associated with it are more general than in the case of mobile robotics. The challenges to overcome are communication, control algorithms, formation and optimal path planning.

1.3. Objectives

The aim of this work is to implement an optimal path planning algorithm using Artificial Potential Fields (APF) to avoid different types of dynamic and static obstacles and prove its performance in simulation. The objectives of this project are the following:

- Regarding APFs:
  - To model and implement the APFs of rectangular shape static obstacles.
  - To model and implement the APFs of circular shape static obstacles.
  - To model and implement the APFs of circular shape dynamic obstacles.
  - To model and implement the APFs of circular shape static goals.
- Regarding path planning:
  - To achieve optimal path planning.
  - To avoid the obstacles.
  - To reach the goal.
  - To avoid local minima.
  - To establish priorities between the robots.
- Regarding simulation environment:
  - To develop a suitable environment for simulation where all the characteristics of the algorithm developed could be proven.
  - To be easily editable.

- Regarding simulation:
  - To simulate in 2D.

- To simulate in 3D.

1.4. Dissertation outline

This dissertation consists of five chapters and four appendices. The first chapter is the introduction and background of the project, where the motivation and aims are established. The second is an overview of the previous literature in the field focusing on path planning methods and swarm robotics. Chapter three describes the methodology employed to achieve the aims which is focused on the Artificial Potential Fields method. In chapter four the results of the simulation will be shown and finally in chapter five the conclusions reached will be explained.

On the other hand, the appendices contain the following information: appendix A contains the MatLab code, appendix B shows snap-shots of the simulation, appendix C contains information about machine vision systems, and the last one, appendix D contains scanned copies of the original forms required for the submission of this project.

CHAPTER TWO: Literature Review

2.1. Introduction

Robotics is a science that has achieved great advances in the last half century, but it still offers a wide field for development and innovation (Conde-Canaviri, n.d.). As Luo, Su, Shen, and Tsai (2003) state, nowadays intelligent robotic systems are thoroughly extended in many fields such as factory automation, surgery, space exploration or military service but it is becoming more and more common to find robot applications for daily life.

Some examples of the huge range of applications that robots perform nowadays are explained by Kaplan (2006), who remarks that robots are conceived to replace or support human beings in task that are hazardous, such as rescuing people in fires, treating toxic waste, assisting doctors in minimally invasive surgery or explore the deepest of seas; extreme, for example exploring the surface of Mars or repairing spacecrafts; or tedious, like assembling pieces or packaging and stacking food products.

Among all the possible applications shown above, in this chapter the attention will go into detail about mobile robotics. Tanwani and Calinon (2016) define mobile robots as those autonomous vehicles that are able to understand the environment surrounding them and act consequently. Ollero Baturone (2007) says that mobile robots were created in response to the necessity to extend the field of application which was mainly industrial. Robotics was restricted by the range of the robot arm, so the goal of mobile robotics is to increase autonomy and restrict human intervention.

Summarizing, the aim of this chapter is, in first place, to provide an overview of history of robotics and to define a robot and its different typologies. Then, the attention will be focused on mobile robots, concretely in the actual challenges of mobile robotics such as navigation, motion planning approaches and multi-robot systems.

2.2. History of Robots

Robotics is an old term and concept, yet it is being a long time and effort since the first robots or mechanical automatons were conceived until now (Conde-Canaviri, n.d.).

As Sánchez-Martín et al. (2007) describe, one of the first automatons date from 1300 B.C, Amenhotep ordered to build a statue of the King of Ethiopia that emitted sounds when sunrise rays illuminate it. Many years later, around 400 B.C. Archytas of Tarentum designed a wooden bird powered by steam that was able to fly 200 meters. At 62 A.D. Heron of Alexandria showed in his book *Pneumatica* the designs of toys or mechanic instruments that were able to move on their own, like singing birds, puppets, a water organ, or different types of engines, the fire engine or the *aeopile* that was the very first steam turbine (Encyclopaedia Britannica, 2018). One of the most remarkable automation inventors in history was Jacques Vaucanson (1709-1782), he did a flutist automaton, a mechanical duck that was able to complete a full digestion and many others that were able to write or draw (Encyclopaedia Britannica, 2018).

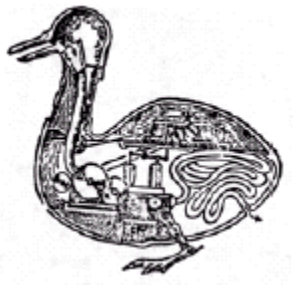


Figure 4: Robotic Duck (Hall, 1985)

During centuries, the idea of robots as a machine similar to humankind has prevailed in human cultures with the aim to create a machine capable of performing tasks with autonomy (Sánchez-Martín et al, 2007). Leonardo da Vinci in 1495 built the first humanoid robot, it was called "Mechanical knight" and it was able to imitate the human movement of jaw, arms and neck. A century later, Gianello Turriano created a doll that played the mandolin (Valero et al., 2011).

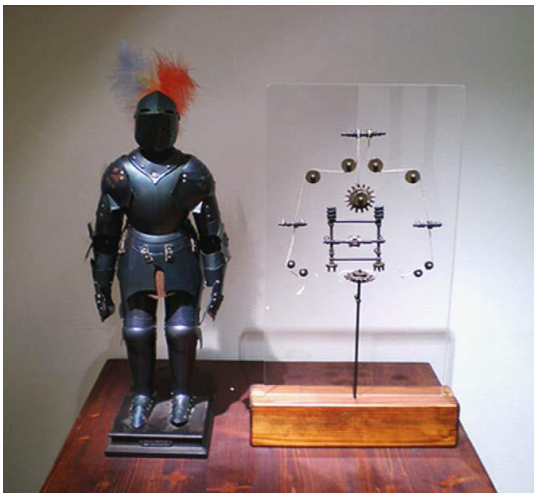


Figure 5: Mechanical knight (Valero et al, 2011)

According to Hall (1985) all the developments during the industrial revolution, such as automatic power sources, machined parts or controllers, led to the actual industrial robots. During the first part of the 20th century, thanks to all the development in the field of computer science, the first robotic arm was built by Roselund and Pollard in 1938 to perform painting tasks in a production chain.

The word "*robot*" appeared for the first time in 1921 in Karel Capek play "*Rossum's Universal Robots*". This term comes from a Czech word which means forced labour. Years later, Isaac Asimov was inspired by the work of Capek and used it in his own books where he defined the tree laws of robotics. Even though Asimov was a science-fiction writer, the laws of robotics that he established in his books are still present and used as a theoretical referent (Sánchez-Martín et al, 2007).

Although the history of Robotics is more than 2000 years old, the last two decades have shown a remarkable revolution due to the changes it has introduced in fields such as Industry, Science or Medicine (Valero et al., 2011). An example of this revolution is shown by Yates et al. (2011),

currently around the 85% of the interventions in the field of urology are performed by robots in the U.S.A. Concretely it is done by using *da Vinci Surgical System*. Lots of examples can be found in the work of Hall (1985) where he mentioned the following robot applications: automotive industry, assembly, laboratories, medicine, nuclear energy, agriculture, underwater exploration, space exploration, custom service, or arts and entertainment among others.

2.2.1. Mobile robots in history

In this section an overview of some mobile robots in history will be done, according to the work of Bermudez (2018):

- Industrial robots: Their main task was cleaning and hoover in spacious areas. The first was *RoboKent* in 1988 followed by *Roboscrub* in 1991.

Figure 6: RoboKent (Bermudez, 2018)

- Security: *Denning Sentry* was designed to patrol and detect intruders. It was equipped with TV camera, microphone and wireless transmitters.

Figure 7: Denning Sentry (Bermudez, 2018)

- Hospitals: The robot *Helpmate* was built to help the nurses. They could send it to another location just by pushing a button and the purpose of this robot was to transport food and medication.

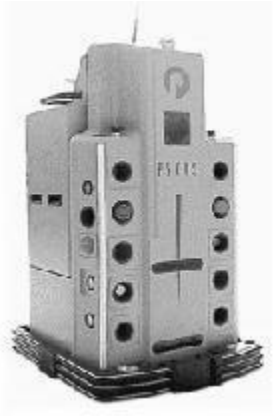


Figure 8: Helpmate (Bermudez, 2018)

- Space research: In 1997 *Sojourner* landed on Mars and for this reason it could be considered as the autonomous mobile robot more successful in history. It was programmed to explore the planet and it was controlled from the Earth, but it also was able to perform actions by itself.

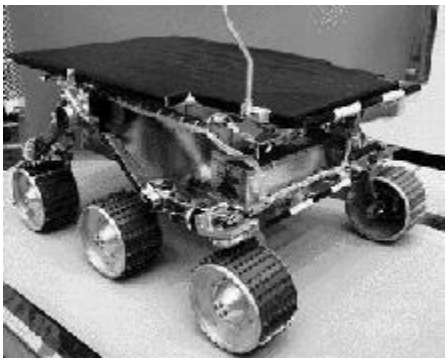


Figure 9: Sajourner (Bermudez, 2018)

- Military: *Fetch* robot was conceived to remove from the field the unexploited bombs and grenades. It performed successfully the navigation from one point to another and the ammunition research task by following a spiral trajectory.

Figure 10: Fletch (Bermudez, 2018)

- Humanoids: *Honda* started in the late eighties the development of prototypes of humanoid robots with the aim of "duplicate the complexities of human motion and genuinely help

people" (Honda, 2018). The development of ASIMO took more than two decades and now it is still in development. Honda (2018) state that ASIMO is "the most advanced humanoid robot in the world".

Figure 11: Honda P-series (Honda, 2018)

Figure 12: ASIMO (Honda, 2018)

2.3. Robot Definition

There are many definitions of a robot, from the most general as the one offered by Encyclopaedia Britannica (2018) that defines a robot as "any automatically operated machine that replaces human effort ". Until the most complete and specific such as the definition given by Latombe (2010) that said "a robot is a versatile mechanical device equipped with actuators and sensors under the control of a computing system. It operates in a workspace within the real world. This workspace is populated by physical objects and is subject to the laws of nature. The robot performs tasks by executing motions in the workspace".

Although one of the most technic definitions is provided by the International Organization of Standardization (2012), ISO from now on, that defines a robot as "an actuated mechanism programmable in two or more axes, with a degree of autonomy, moving within its environment, to perform intended tasks". The ISO also defines the differences among robot typologies:

- Industrial robots: it is "an automatically controlled, reprogrammable, multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications". Being a manipulator "a machine in which the mechanism usually consists of a series of segments, joined or sliding relative to one another, for the purpose of grasping and/or moving objects (pieces or tools) usually in several degrees of freedom".
- Mobile robots: Those that "are able to travel under its own control and it can be a mobile platform with or without manipulators".
- Cobots: robots that "are designed for direct interaction with human, the operations and the interaction occur within a defined workspace".

2.4. Challenges of mobile robotics

The greatest success to date in robotics has been achieved in the world of industrial manufacturing thanks to robot arms or manipulators (Fahimi, 2010). The success of manipulators is due to their speed and high accuracy, so they are perfect to perform repetitive tasks such as welding and painting in an assembly line. Though all their success, the lack of mobility of robot arms is their biggest disadvantage. By contrast, mobile robots are able to freely move through the plant, and its most characteristic quality is their flexibility.

Mobile robotics is an interdisciplinary field that combines knowledge of many different disciplines such as kinematics, dynamics, control, signal processing, computer vision, or computer science, among others. To achieve a robust navigation, the following problems must be solved, and they are related to the different fields that involve mobile robotics. The first challenge is locomotion; the next is perception and interpretation of the environment; and finally, navigation that combines cognition and localization (Siegwart and Nourbakhsh, 2004). The challenges presented above will be further discussed in the next sections.

2.5. Perception

Perception and interpretation of the data are two of the most important tasks that a robot has to develop because through all the information acquired by the sensors the robot is able to see and analyse the environment and perform consequently. According to Tzafestas (2013) the sensors for robots are inspired by the human sensory system that provides input signals to the brain to be processed. For a robot, those signals provide them with higher intelligence capabilities such as vision or hearing.

The interest in the field of perception and more precisely in image processing is becoming more and more popular due to it is a key tool needed to provide vision to autonomous mobile robots. This tool can be used to solve the problem of object recognition that usually it is processed by humans.

2.5.1. Machine Vision

A machine vision system objective is to recover useful information about an image from its two-dimensional projections (Ramesh and Kasturi, 1995). Images are two dimensional projections of

the tree dimensional world so there is missed information that must be recovered. The goal is to create a model of the real world from images. Machine vision provides the robot the information it needs to understand the environment and be able to navigate autonomously. There are techniques and methods to recover depth from the images and it is widely used in autonomous vehicles, airplanes, tanks and robots.

Ramesh and Kasturi (1995) also point that machine vision is usually considered as a subfield of artificial intelligence. This fact is due to many techniques from artificial intelligence are used in many aspects of computer vision. The encyclopaedia (2018) defines artificial intelligence as "the ability of a digital computer to perform tasks commonly associated with intelligent beings".

2.6. Navigation problem

As it has ben said before, navigation is the main challenge of mobile robotics because it involves all the other aspects.

Bataling et al. (2004) describes the two types of navigation problems in their work. The first one is the local navigation problem whose scale is only a few meters and the main problem is obstacle avoidance. As it has been shown in the previous sections, there are many techniques to map the environment which allow the control system to know the surroundings and using this information the control system is able to identify the obstacles and avoid them.

The second problem is global navigation and in this case the range is larger than the local. In this case the problem that must be solved is how to find the goal when it is out of the range of perception at the initial state. As it happens with the previous case there are many approaches developed to solve this problem.

2.7. Motion planning

Locomotion includes how the robot should move towards reaching its goal and the mechanisms involved in the process. Villaseñor Carrillo et al. (2010) state that efficient techniques and algorithms are required in order to implement the navigation system that is able to achieve an efficient movement in the surrounding environment. Robots must be able to perform efficiently in a real environment, build its own map, and navigate autonomously.

The locomotion mechanisms are what enable the robot to move throughout its environment (Siegwart and Nourbakhsh, 2004). There are robots that can walk, jump, run, slide, roll swim, or fly, so there are a large variety of possible choices towards locomotion, and this aspect is crucial because it must be compatible with the robot application needed.

On the other hand, path planning is the compliment to the mechanisms exposed above to achieve autonomous vehicles. Its purpose is to generate feasible routes between the initial position and the goal avoiding the obstacles placed in the environment (Gonzalez et al., 2017). Path planning is about the connectivity of the environment F and the objective is to connect two given configurations q_{init} and q_{goal}. Usually a path planning algorithm discretizes F and searches for its connectivity graph, and then it searches in the graph a suitable path (Ge and Lewis, 2006). There are numerous algorithms and the difference between them is how the connectivity graph is built, some of them will be explained below.

2.7.1. Roadmaps

The roadmap is a method that relies on rules based on the geometry of the obstacle field. As Fahimi (2010) says many motion planning problems have been solved by using this method such as motion planning for several circular or rectangular obstacles, moving obstacles, and motion planning for multiple robots.

Digani et al. (2014) define roadmap as a union of one-dimensional curves whose properties are accessibility, departability, and connectivity. In addition, the robot is restricted to move along the curves of the network.

Visibility graph is commonly used in 2D roadmaps with polygonal obstacles. It builds the connectivity graph connecting the vertices of the polygonal obstacles and the configurations q_{init} and $q_{goal.}$ The connexion between two nodes is stablished if the strait line path that connects them does not intersect the interior of an obstacle (Ge and Lewis, 2006).

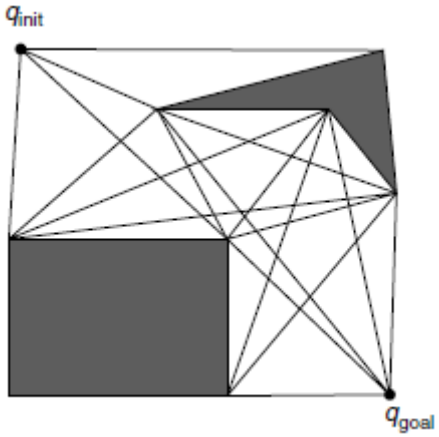


Figure 13: Visibility graph (Ge and Lewis, 2006)

Voronoi diagrams are widely used for extracting distance node information from a 2D environment. The algorithm constructs a skeleton of points with minimal distances to obstacles and walls. By following the curves in this diagram, the robot stays as far away as possible from the obstacles. Bräunl (2008) defines Voronoi diagrams as follows:

- F is the free space in the environment
- F' is the occupied space
- $b \in F'$ is basis point for $p \in F$ if b has minimal distance to p, compared with all other points in F'

$$\text{Voronoi diagram} = \{p \in F' \mid p \text{ has at least two basis points}\}$$

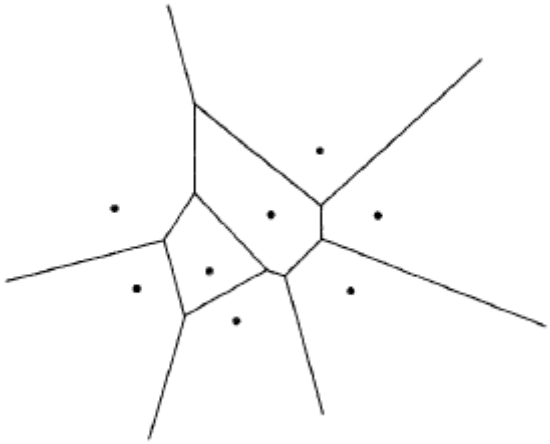


Figure 14: Voronoi diagram (Aurenhammer, 1991)

Ge and Lewis (2006) say that "in 2D polygonal configuration spaces, the visibility graph and the Voronoi diagram capture the connectivity of the space exactly". This means that a collision free path exists between two given configurations if and only if a path in the corresponding diagrams exists.

2.7.2. Cell decomposition

The cell decomposition strategy is based on the division of the free space into a finite set of regions which can be safely crossed by the robot. This method and the roadmap are based on geometry, so these types of methods are focused on connectivity ignoring optimality or computational complexity (Gonzalez et al., 2017).

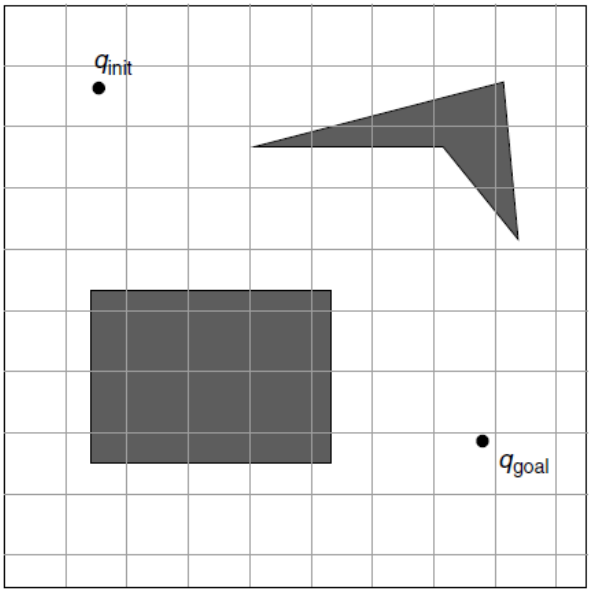


Figure 15: Cell decomposition with fixed resolution grid (Ge and Lewis, 2006)

The cell decomposition approach divides the free space into canonical regions called cells that usually are convex, and then the algorithm constructs a graph with the connectivity of the environment (Ge and Lewis, 2026). The graph obtained contains the information of each cell stored in the nodes, and each arc links two cells sharing a line segment. A minimum cost path can be found between the initial position and the goal by using a search algorithm (Gonzalez et al., 2017).

The resolution and the shape of the grid can be chosen. Depending on those parameters the result may vary due to it will find a path between q_{init} and q_{goal} only when one path exists, and the existence of the path is related directly to the resolution that must be appropriate for the environment and the obstacles that it contains. As Kloetzer et al. (2015) point, traditional formulations of this method are based on the usage of the middle points of the boundaries of the cell to build the path, but this may lead to too conservative routes and longer travelled distances.

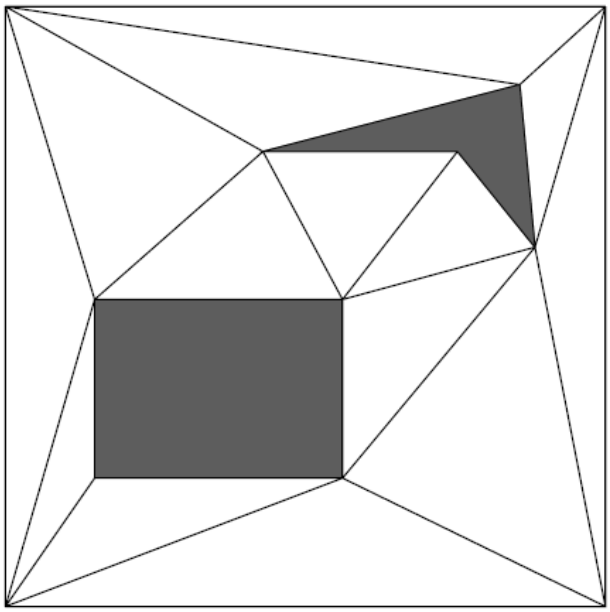


Figure 16: Cell decomposition by triangulation (Ge and Lewis, 2006)

2.7.3. Bug Algorithm

The bug algorithm aim is to find a way from the initial point to the goal in unknown environments. As Xu, Yu and Bai (2017) describe the robot method as it follows "the robot moves directly to the target position firstly, and then adds the insert points and updates the current path to avoid the obstacles, until the target position is reached". The advantages of this method are that it is simple and easy to realize.

Lumelsky and Stepanov developed this algorithm in 1987, but there are many versions of it with different characteristics, two of them are shown below.

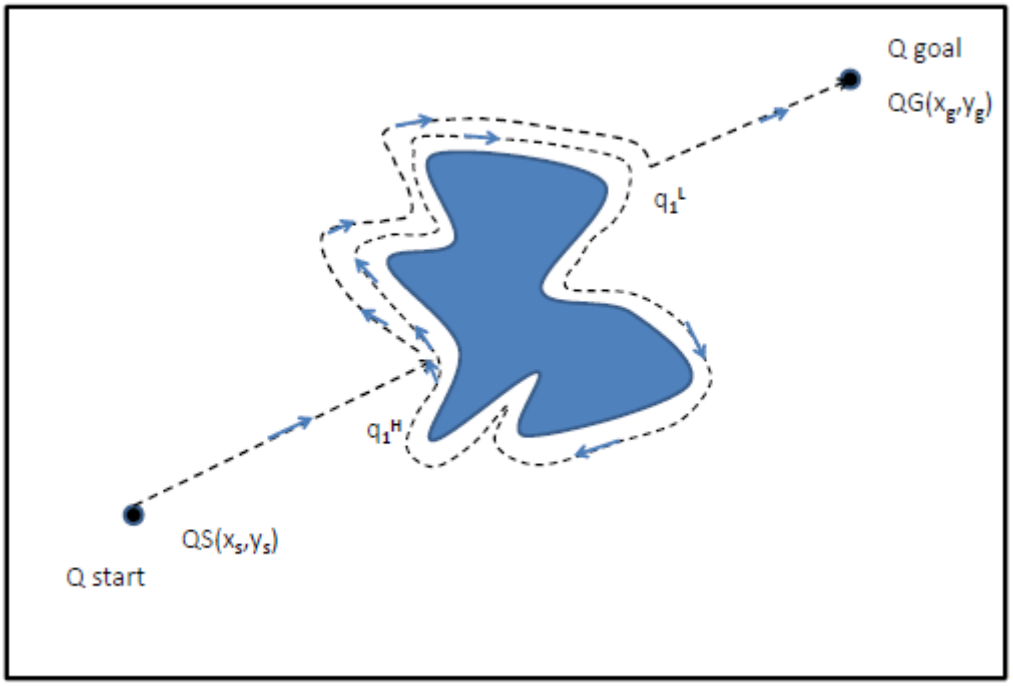


Figure 17: Bug 1 (Alboul, 2017)

The main characteristic of Bug 1 is that the robot moves around all the perimeter of the obstacle, and then it goes to the goal from the point with the shortest distance.

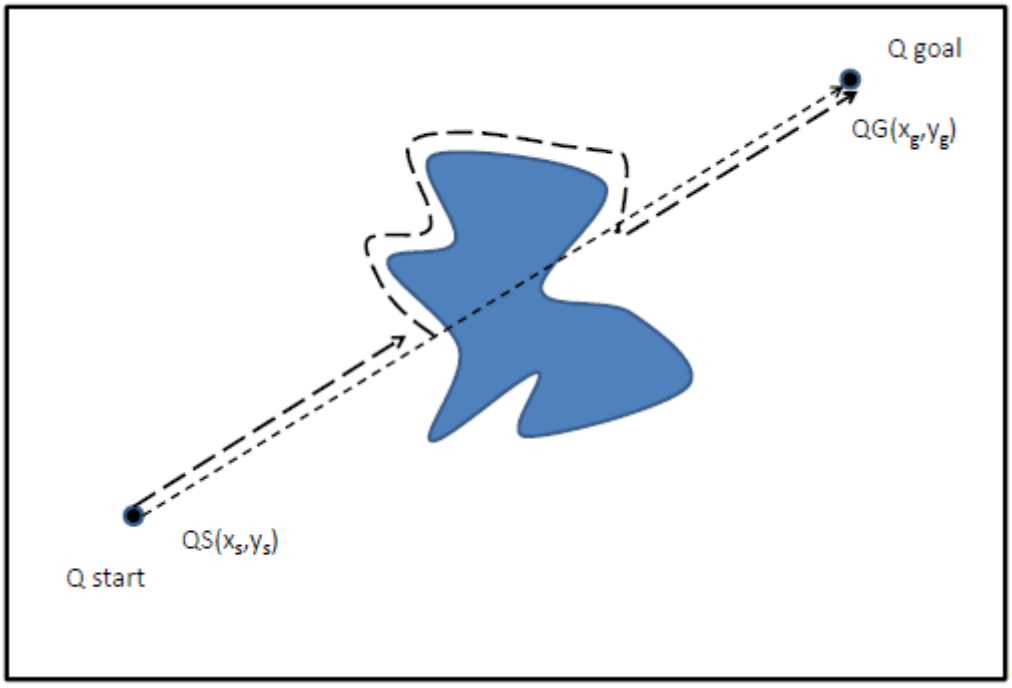


Figure 18: Bug 2 (Alboul, 2017)

In this case, Bug 2 navigates around the contour of the obstacle until it is possible to depart directly to the goal.

2.7.4. Artificial Potential fields

Artificial Potential Field (APF) is a method that is becoming more and more popular due to its simplicity and mathematical elegance (Wang et al., 2013). This method has easy implementation and provides good results. However, it has some inconveniences that difficult its application such as local minima, oscillation and Goal Non-Reachable with Obstacle Nearby (GNRON). It was introduced by Khatib in 1986, in that paper he proposed the method as a low-level control mechanism for industrial and mobile robots with the aim of improving collision avoidance in real time tasks.

The basis of this method is that robots are considered particles moving in the virtual potential field. Then, the obstacles are assigned a potential field that repel the robot and the goal is given a potential field that attract it. The resultant of all the forces applied on the robot will determine the subsequent direction and speed of travel that will guide the robot to the goal avoiding effectively any collision.

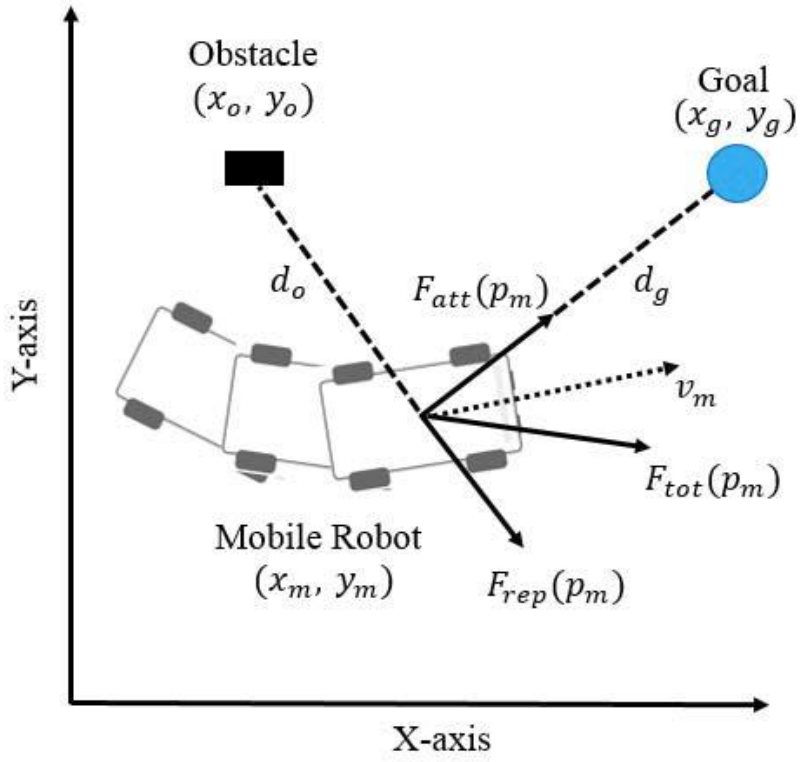


Figure 19: Force situation APF (Lee et al., 2017)

Although this method was conceived by Khatip and Arkin for single robot case in known environments, many researchers such as Zalvanos and Pappas (2007) have used it for multi-robot case or unknown environments. In their work, they model connectivity as it was an imaginary obstacle in the space and use APF to avoid collisions. Continuing the previous work, Zalvanos et al. (2009) also used APFs for dynamic networks. This approach is also used to control a large number of robots in swarm robotics as it is exposed by Bayindir and Sahin (2007).

2.7.4.1. The Traditional Artificial Potential Field Method Formulation

Lee et al. (2017) describe the method using the most common potential functions as follows, being the attractive potential of the goal *Uatt* and the repulsive of the obstacles *Urep*. The total potential field *Utot* is defined as:

$$U_{tot}(q) = U_{att}(q) + U_{rep}(q)$$

Where the position of the robot is $q = (x, y)^{\mathrm{T}}$

The most commonly used attractive potential is:

$$U_{att}(q) = \frac{1}{2}\xi\rho^{m}(q, q_{goal})$$

Where ξ is a positive scaling factor, m is 1 for attractive potential with conic shape or 2 for parabolic shape, and $\rho(q, q_{goal})$ is the distance between the actual position q and the goal q_{goal}.

$$\rho(q, q_{goal}) = \|q_{goal} - q\|$$

The attractive force is given by:

$$F_{att} = -\nabla U_{att}(q) = \xi\rho(q, q_{goal})$$

The commonly used repulsive potential function:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q, q_{obs})} - \frac{1}{\rho_0}\right)^2, & if\ \rho(q, q_{obs}) \le \rho_0 \\ 0, & if\ \rho(q, q_{obs}) > \rho_0 \end{cases}$$

Where η is a positive scaling factor, $\rho(q, q_{obs})$ is the distance between the robot and the obstacle, q_{obs} is the position of the obstacle, and ρ_0 is the distance of influence.

The repulsive force is:

$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} \eta\left(\frac{1}{\rho(q, q_{obs})} - \frac{1}{\rho_0}\right)\frac{\nabla\rho(q, q_{obs})}{\rho^2(q, q_{obs})}, & if\ \rho(q, q_{obs}) \le \rho_0 \\ 0, & if\ \rho(q, q_{obs}) > \rho_0 \end{cases}$$

Then, the resultant applied to the robot is the sum of the attractive and repulsive force:

$$F_{tot} = -\nabla U_{tot}(q) = F_{att} + F_{rep}$$

And the robot moves in the direction of the resultant:

$$\dot{q} = -\nabla(U_{att}(q) + U_{rep}(q))$$

2.7.4.2. Problems with the APF method

As it has been mentioned before, there are inherent problems in this APF method. But those problems are independent of the particular implementation. Koren and Borenstein (1991) explain them in their work:

- Trap situations due to local minima: This is the most known problem of APFs. Traps can be created by different obstacle configurations such as U-shaped obstacles. The next image shows three examples.

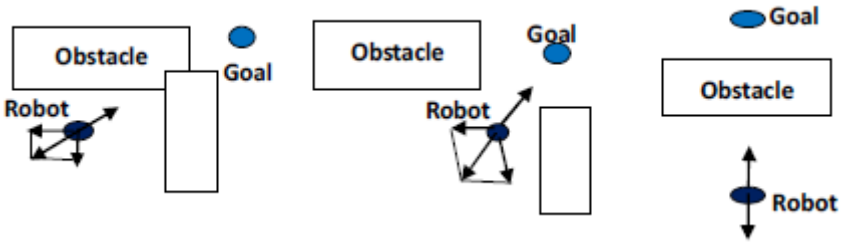


Figure 20: Obstacle configurations that create local minima (Wang et al., 2013)

- The GNRON problem: This situation occurs when the goal has obstacles nearby and the repulsive force of those is bigger than the attractive force from the goal, so the robot will never reach the target.

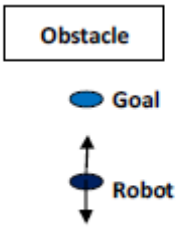


Figure 21: GNRON problem (Wang et al., 2013)

- No passage between closely spaced obstacles: When two obstacles are too close may be that the sum of the repulsive forces points in the opposite direction from the opening. Consequently, the resultant does not lead the robot between them. This problem also depends on the magnitude and direction of the attractive force of the goal as it is shown in the nest figure.

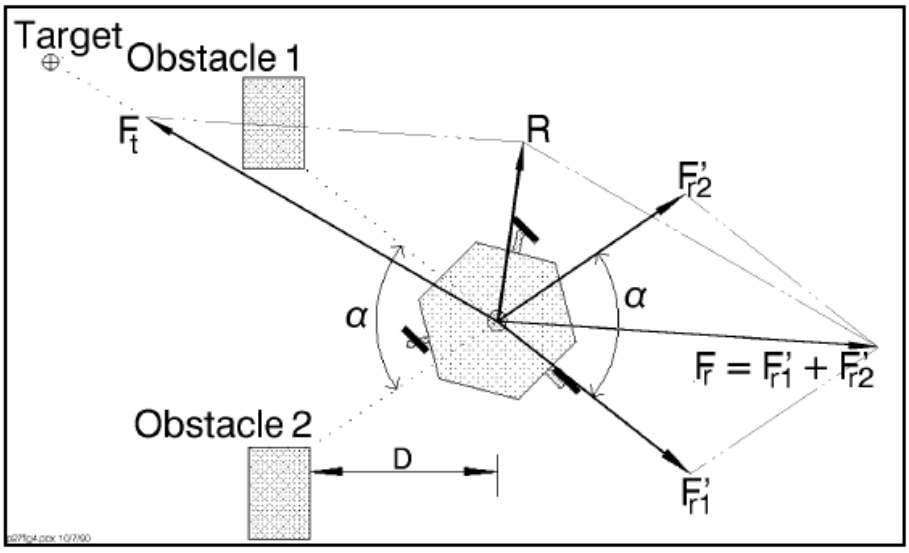


Figure 22: Example of no passage between closely spaced obstacles (Koren and Borenstein, 1991)

- Oscillations: APFs method has a high tendency to cause unstable motion in the presence of obstacles. For example, when the robot is travelling in narrow corridors where it experiences repulsive forces from opposite sides.

2.7.4.3. Social Potential Fields

Social potential field is the name that Reif and Wang (1999) gave to the method that they develop to control a very large multi-robot system from hundreds to ten thousand of robots.

For this approach, they choose distributed control due to, for that number of robots, centralized control has very high computational and communicative requirements. By using decentralized control, the scalability of the system increases owing to each robot only needs its own information to plan its navigation. Furthermore, rules were stablished for every robot to regulate the interactions with other robots and obstacles.

The robot motion is controlled by the resultant artificial force imposed by the neighbour robots and other components of the system. The force laws between robots are inverse-power laws of distances incorporating attraction and repulsion.

$$f(r) = -\frac{c_1}{r^{\sigma_1}} + \frac{c_2}{r^{\sigma_2}}$$

$$c_1, c_2 \geq 0$$

$$\sigma_1 > \sigma_2 > 0$$

The attractive term which is the positive, is dominant when the robot is going too far away, and the repulsive or negative term dominates when the robot is close to another component of the system. Collective behaviours can be achieved by the adjustment of the parameters, and those behaviours can be clustering, guarding, or patrolling.

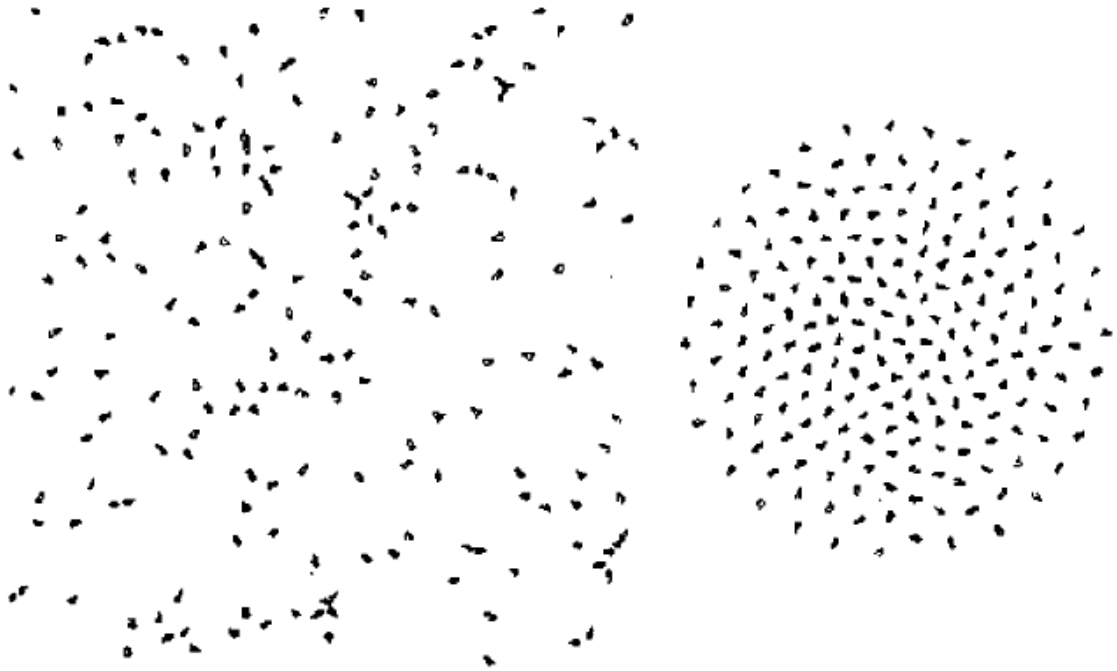


Figure 23: Cluster of robots formed with an identical force law (Reif and Wang, 1999). On the left it is shown the distribution after 3 iterations, on the right after 225 iterations.

2.8. Multi-Robot Systems: Swarm Robotics

Nowadays multi-robot systems are becoming one of the most important research areas in the field due to the high potential of its applications (Lima and Custodio, 2015). Similarities between human and robot teams can be found, for example a group of people in line is able to move a bucket from a source of water until the fire faster and more efficiently than working individually. Even though if some of the people left the line the rest can continue working and performing adequately with the pertinent penalization in velocity.

The same argument can be applied to multi-robot systems to perform tasks such as building surveillance, transport of large objects, air and underwater monitoring, forest fire detection, or search and rescue operations. As Arai et al. (2002) say a team of robots is able to perform a task in a cheaper, faster, and more reliable way than a single robot, furthermore they provide redundancy and more robustness. In order to perform the tasks such as the ones described before the robots must accomplish the following requirements: they must be highly mobile, maintain communication across a large area, estimate their own configuration, and make collective decisions.

Swarm robotics is a new approach to the coordination of large number of simple robots. It is inspired by social animals and their way of behaving because they are able to coordinate their actions to accomplish tasks that are beyond the capabilities of an individual (Bayindir and Sahin, 2007). Social insects like ants, termites, wasps and bees show three characteristics that are desired for multi robot systems that are robustness that is the degree in which a system can work under abnormal circumstances; flexibility, which means the ability to adapt; and scalability, that is the ability to expand. Swarms of robots are better in applications that require spatially distributed sensing or actuation than single and more sophisticated robots.

In the following sections some relevant aspects about swarm robotics will be introduced.

2.8.1. Biological Inspiration

Swarm behaviour observed in social insects, flocking birds, or shoals of fishes have been the inspiration of this approach. Individuals within the swarm have their own task, and all those tasks are integrated perfectly without supervision (Bonabeau et al., 1999). As a result, the swarm has a collective behaviour that is desired for multi-robot systems due to it is decentralized and self-organized.

Many examples of this collective behaviour can be found such as ants that are able to form chains of their own bodies in order to cross wide gaps and pull stiff leaf together to form a nest or create highways to bring the cut leaves from hundreds of meters away. Wasps can build complex nests differentiating among three types of groups of workers whose size is determined by the needs of the colony. Flocks of birds navigate and maintain their formation by adjusting their speed and orientation according with their immediate neighbours.

Figure 24: Example of collective behaviour performed by ants.

2.8.2. Swarm Intelligence

Behaviour-based control pretend to imitate the behaviour and characteristics of social insects or animals (Arai et al., 2002). It is also known as *swarm intelligence* because the system has a collective behaviour even though the individuals that compound it only have information about their surroundings and are governed by very simple rules that regulate the interaction with other components of the swarm. Cao et al. (1997) define swarm intelligence as "a property of systems of non-intelligent robots exhibiting collectively intelligent behaviour". Two of the most remarkable characteristics are the ability to self-organize that can be understood as the ability of the team to distribute optimally to develop a task, and its distributed architecture, which means that each robot controls itself rather than having a centralized control mechanism for the hole swarm.

The goal is to apply simple control rules of various biological societies like ants or bees, to multi-robot systems. Behaviour-based multi-robot teams can control how to disperse, aggregate, forage, flock and follow trails.

As it has been mentioned before, the application of swarm intelligence to multi robot teams has numerous advantages such as robustness, due to the inherent redundancy of the system compound by simple self-organized individuals; parallelism, the team can work in different subtasks simultaneously to complete the main task in less time; low-cost, due to the robots of the swarm are simple; and scalability (Winfield and Nembrini, 2006). On the other hand, it also presents some problems like responsiveness due to it is impossible to control a particular robot of the system, and low predictability due to their local interaction and the size of the system.

2.8.3. Formation control

Formation control can be understood as the problem of controlling the relative positions and orientations of each robot in the team, while allowing them to move as a whole. Swarm robots can form and keep various formations like line or triangle, in order to accomplish different tasks. Formation control is important in swarm robotics due to it allow individual team members to concentrate their sensors across a portion of the environment, while their partners cover the rest to ensure full coverage (Balch and Arkin, 1998).

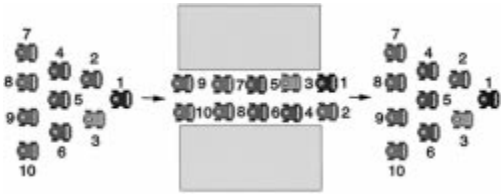


Figure 25: Formation of robots changing shapes (Desai et al., 2001)

There are three main approaches to distributed formation control as it is explained by Xu et al. (2014):

- Behaviour-based: It has predefined behaviours such as moving-to-target, keeping-formation, and avoiding-obstacles. This method is suitable for distributed systems where the robots are strongly autonomous.
- Leader-follower: A leader is designated, and the rest of the team follow it keeping a certain distance and direction.
- Virtual structure: This method considers the team as an inflexible entity where robots are points contained on it. The strict requirements of this formation limit its applications.

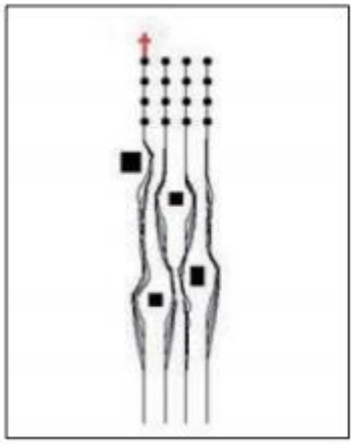


Figure 26: Formation control obstacle avoidance (Xu et al., 2014)

2.8.4. Connectivity

Communication in multi robot teams is a key issue due to it is important and necessary the continuous exchange of information between the robot team. As Zalvanos et al. (2009) point one of the main challenges involving mobile multi-robot systems is the development of distributed motion algorithms that guarantee connectivity of the overall network. However, the wireless connection between the team often fails due to environmental interference, fading, or for being out of the range. For these reasons, dynamic networks are receiving considerable attention, and algorithms are being developed to solve the problems. An example is shown in the work of Zalvanos et al. (2007) where they use potential fields for maintaining the connectivity in the network.

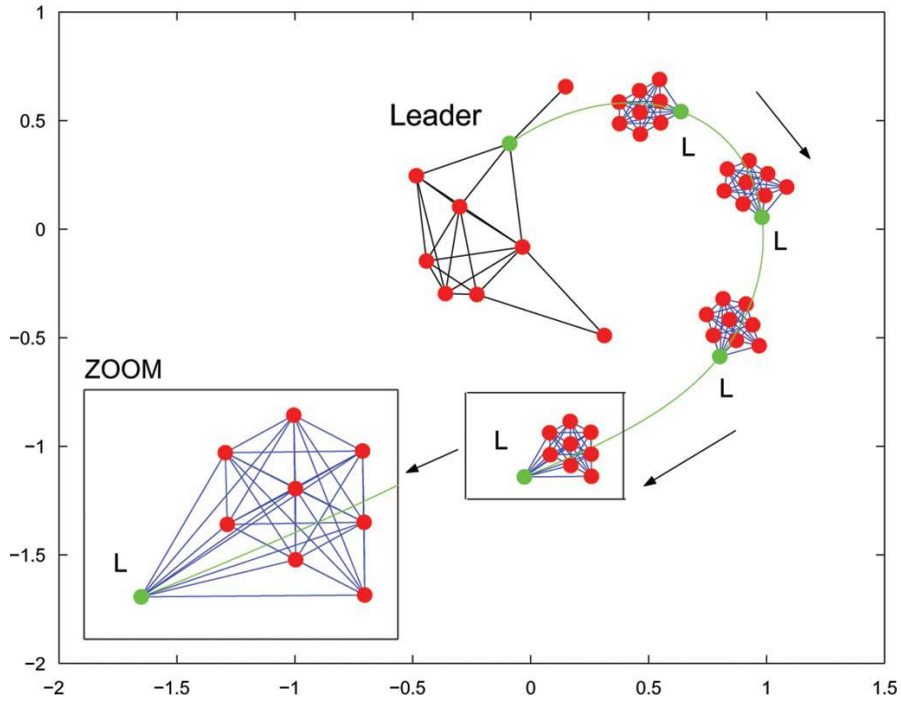


Figure 27: Communication in a team of mobile robots (Zalvanos et al., 2007)

2.9. Summary

In general terms, the aim of any mobile robot is to reach effectively his goal, which means that it must navigate safely, avoiding the obstacles and following the optimum path towards its objective. To accomplish this task the robot must deal with tree problems: perception, locomotion, and navigation.

The emphasis in this chapter has been focused on motion planning, and four of the most used algorithms have been introduced. To begin with, roadmaps and cell decomposition are two algorithms for known environments and both are based on the geometry of the field. The first one uses visibility graphs or Voronoi diagrams among others to build the connectivity graph. Whereas cell decomposition is based on the division of the space in cells following geometrical patterns. Both methods are widely used, but they need to be applied jointly with a search algorithm to find an optimum path in the graph.

On the other hand, bug algorithm is suitable for unknown environments. Its goal is to achieve the target using as little global information as possible, and for this reason it is considered one of the simplest navigation algorithms. Although, if there is no path possible the robot can be trapped in a loop around the obstacles. Since it was introduced for the first time, many versions of It have been developed in other to improve its performance.

Finally, Artificial Potential Field approach is the most popular due to its simplicity, elegance and good results. Robots are considered particles moving in the virtual potential field and they are attracted by goals and repelled by the obstacles. However, it has some inconveniences such as local minima, oscillation, and GNRON. This method was conceived for a single robot when the environment is known, but it also has been modified and adapted for multiple robots and unknown environments.

To conclude the review, the field of swarm robotics has been introduced and the key points of it have been exposed. The aim was to provide an overview of one of the most relevant fields in development nowadays in Robotics, and its possible applications.

Summarizing, among all the algorithms shown in this chapter APF is a method that has proved a good combination of simplicity and effective results, and it is backed by multiple papers. Furthermore, its capacity to adapt to the wide casuistry must be highlighted, for example its applications in the field of swarm robotics.

CHAPTER THREE:
Methodology

3.1. Introduction

In this chapter, the methodology used to achieve the objectives of this dissertation will be described. The main goal of this dissertation is to implement and simulate in MatLab an optimal path planning algorithm using Artificial Potential Fields (APF) for three mobile robots to avoid different types of static and dynamic obstacles in known environment.

To accomplish this task the APF method will be used for path planning. This algorithm has been described and compared with others in the last chapter. As it has been said before, even though this method was purposed by Khatib thirty years ago for single robot case in known environments, it is widely used nowadays due to its simplicity, elegance, and good results for mobile robot local collision avoidance.

Furthermore, the method to implement the navigation will be described as well as the software requirements, MatLab in this case, to perform the simulation and achieve a suitable solution.

3.2. Artificial Potential Fields for navigation

The simplicity of the Artificial Potential Field method relies on the existence of two types of forces that act on the robot:

- Repulsive forces: generated by obstacles
- Attractive forces: generated by goals

Robots are considered as particles moving in the virtual potential field created by the forces of the components of the environment. Each position has a potential value which determines the level of energy for that point. The direction and speed of the robot will be determined by the resultant of the forces applied on its position. The resultant force will guide the robot towards the goal avoiding effectively any collision through the optimal path (Wang et al., 2013).

The total potential field *Utot* is defined as:

$$U_{tot}(q) = U_{att}(q) + U_{rep}(q)$$

Where the position of the robot is $q = (x, y)^{\mathrm{T}}$

Uatt is the attractive potential of the goal and *Urep* is the sum of all the repulsive potentials of the obstacles. In chapter two section 2.7.4.1., the most commonly used attractive and repulsive potential functions are shown. The functions employed in the implementation of this work will be further discuss in the next sections.

A charged particle put in a potential field, it moves in the direction of decreasing the potential. Hence the applied force *Ftot* is defined as:

$$F_{tot} = -\nabla U_{tot}(q)$$

The problems of this method are explained in chapter two section 2.7.4.2. Summarizing, these problems are:

- Local minima.
- Goal non-reachable with obstacle nearby (GNROB).
- Closely spaced obstacles.

- Oscillations.

3.2.1. Potential Functions

In this section the different potential functions used for each component of the system will be exposed.

3.2.1.1. Attractive: Goals

The attractive potential function chosen for Goals is:

$$U_{att}(q) = K * \rho(q, q_{goal})$$

Being K a positive scaling factor and $\rho(q, q_{goal})$ the distance between the actual position q and the goal q_{goal}

$$\rho(q, q_{goal}) = \|q_{goal} - q\|$$

This attractive function is constant and equal to the value of the constant *K*.

Finally, the attractive force produced by goals is:

$$\overrightarrow{F_{att}} = K * \frac{q_{goal} - q}{\rho(q, q_{goal})}$$

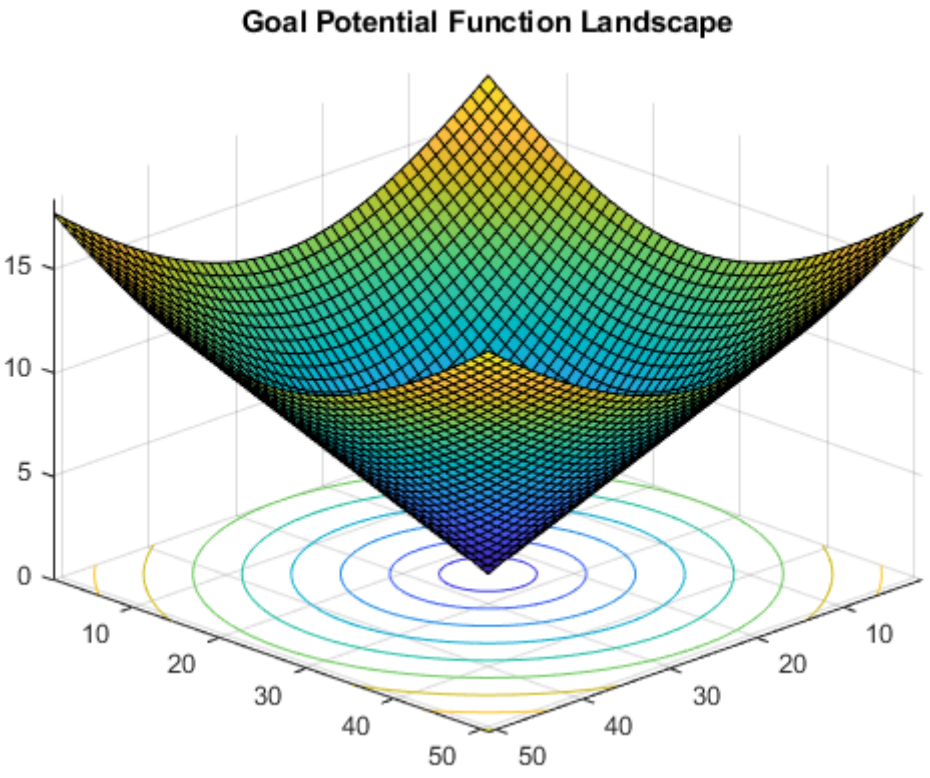


Figure 28: Goal potential function

3.2.1.2. Repulsive: Obstacles

3.2.1.2.1. Circular

The following potential function has been used for dynamic and static circular obstacles:

$$U_{rep}(q) = e^{-\frac{\rho(q,q_{obs})}{K}}$$

Where $\rho(q, q_{obs})$ is the distance between the robot and the obstacle, and q_{obs} is the position of the obstacle. Then, its attractive force is:

$$\overrightarrow{F_{rep}} = (q - q_{obs})e^{-\frac{\rho(q,q_{obs})}{K}}$$

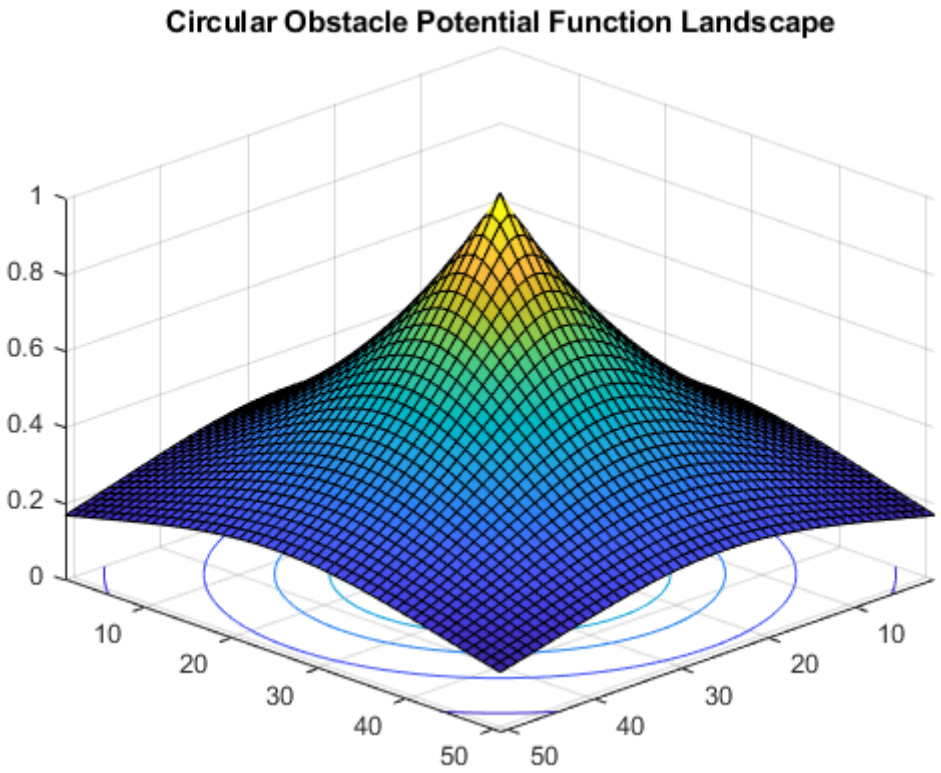


Figure 29: Circular Obstacle Potential Function

3.2.1.2.2. Rectangular

For this type of obstacles, the potential function is defined in three parts depending on the distance between the robot and the boundaries of the obstacle. An obstacle is represented in the next figure where the dark blue area is the obstacle, the light blue area is its area of influence and the white area is where the obstacle has no influence.

Figure 30: Area of influence rectangular obstacle

The area of influence is defined by two parameters, where ρ_{inv} must be grater than cero and it represents the distance from the boundaries of the obstacle where the influence of it starts, it is recommended to choose a small value. Then, ρ_{inv} is the distance from the boundaries where the influence ends.

$$\rho_{\text{inv}} > \rho_{\text{min}} > 0$$

When the distance between the robot and the obstacle $\rho(q, q_{obs})$ es greater than ρ_{inv} it has no influence, so the repulsive potential is:

$$U_{rep}(q) = 0,\ if\ \rho(q, q_{obs}) > \rho_{\text{inv}}$$

And the repulsive force is:

$$F_{rep}(q) = 0, if\ \rho(q, q_{obs}) > \rho_{\text{inv}}$$

When the robot is in the area of influence:

$$U_{rep}(q) = \frac{1}{2}\left(\frac{1}{\rho(q, q_{obs})} - \frac{1}{\rho_{\text{inv}}}\right)^2, \qquad if\ \rho_{\text{inv}} > \rho(q, q_{obs}) > \rho_{\text{min}}$$

And the repulsive force is:

$$\overrightarrow{F_{rep}} = (q - q_{obs})\frac{1}{2}\left(\frac{1}{\rho(q, q_{obs})} - \frac{1}{\rho_{\text{inv}}}\right)^2$$

Finally, if the potential function is constant in the area where is the obstacle:

$$U_{rep}(q) = \frac{1}{2}\left(\frac{1}{\rho_{\text{min}}} - \frac{1}{\rho_{\text{inv}}}\right)^2, \qquad if\ \rho(q, q_{obs}) < \rho_{\text{min}}$$

And the repulsive force is:

$$\overrightarrow{F_{rep}} = (q - q_{obs})\frac{1}{2}\left(\frac{1}{\rho_{\text{min}}} - \frac{1}{\rho_{\text{inv}}}\right)^2$$

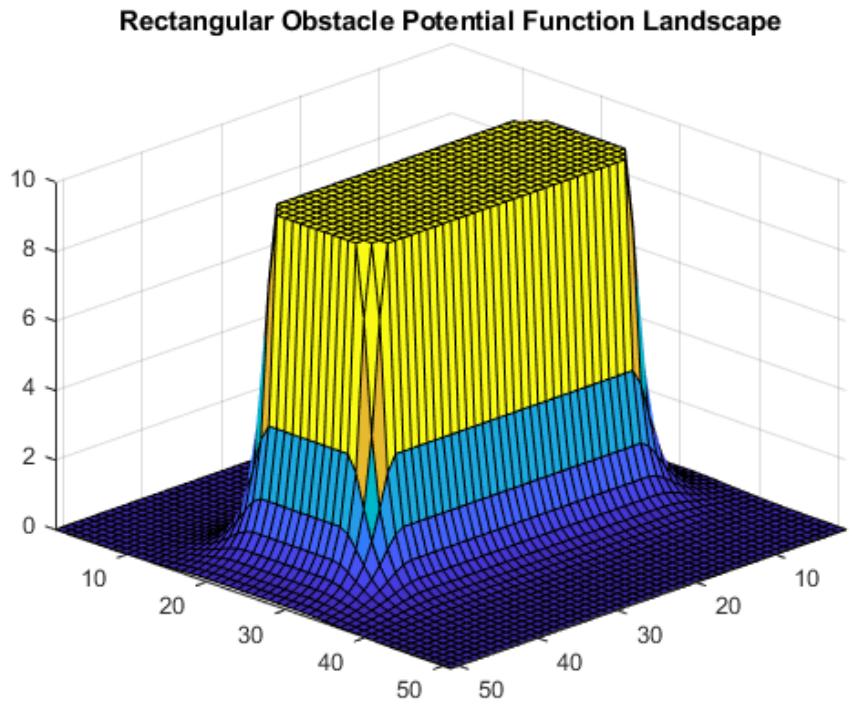


Figure 31: Rectangular Obstacle Potential Function

3.3. Navigation

The challenges of mobile robotics nowadays and its thee main problems, including navigation, have been developed in chapter two. In this section the attention will be focused on navigation.

According with Amiryan and Jamzad (2015), a motion plan can be produced from a potential field following the next steps. Starting from the initial position, then calculating the force vector in each state and the consequent new position until the goal is reached, or the robot gets trapped in local minima. In other words, the steps that must be followed to successfully complete the navigation are the following:

1. Definition of the environment and the obstacles on it.
2. Introduction of the initial position of the robots and the goals.
3. Calculation of the total APF of the environment
4. Navigation loop until the last robot reaches its objective.
    a. Check if the robot has reached the goal
    b. Calculation of the forces and the resultant on the robot.
    c. Update new position.
5. Finish navigation.

According with Koren and Borenstein (1991), the mayor disadvantage of using APFs for mobile robot navigation is the existence of local minima. This occurs whenever the attractive and repulsive forces cancel each other, so the robot gets trapped. They purpose some solutions to this problem such as the use of harmonic potential fields, but this solution is computationally expensive; an approach where a local model of the environment is done to determine a possible escape; or the simplest solution that is to move randomly. The last one has been selected to be implemented in this work.

Furthermore, in the problem proposed for this dissertation the environment contains three robots, so a system of priorities has been stablished between them in order to avoid collisions. One of the robots has no restrictions and the other two have to check its proximity with the others and if they are close they will stop until the other is out of the range of collision.

Finally, in the next diagram the navigation sequence is described including the collision avoidance and the preferences between robots.

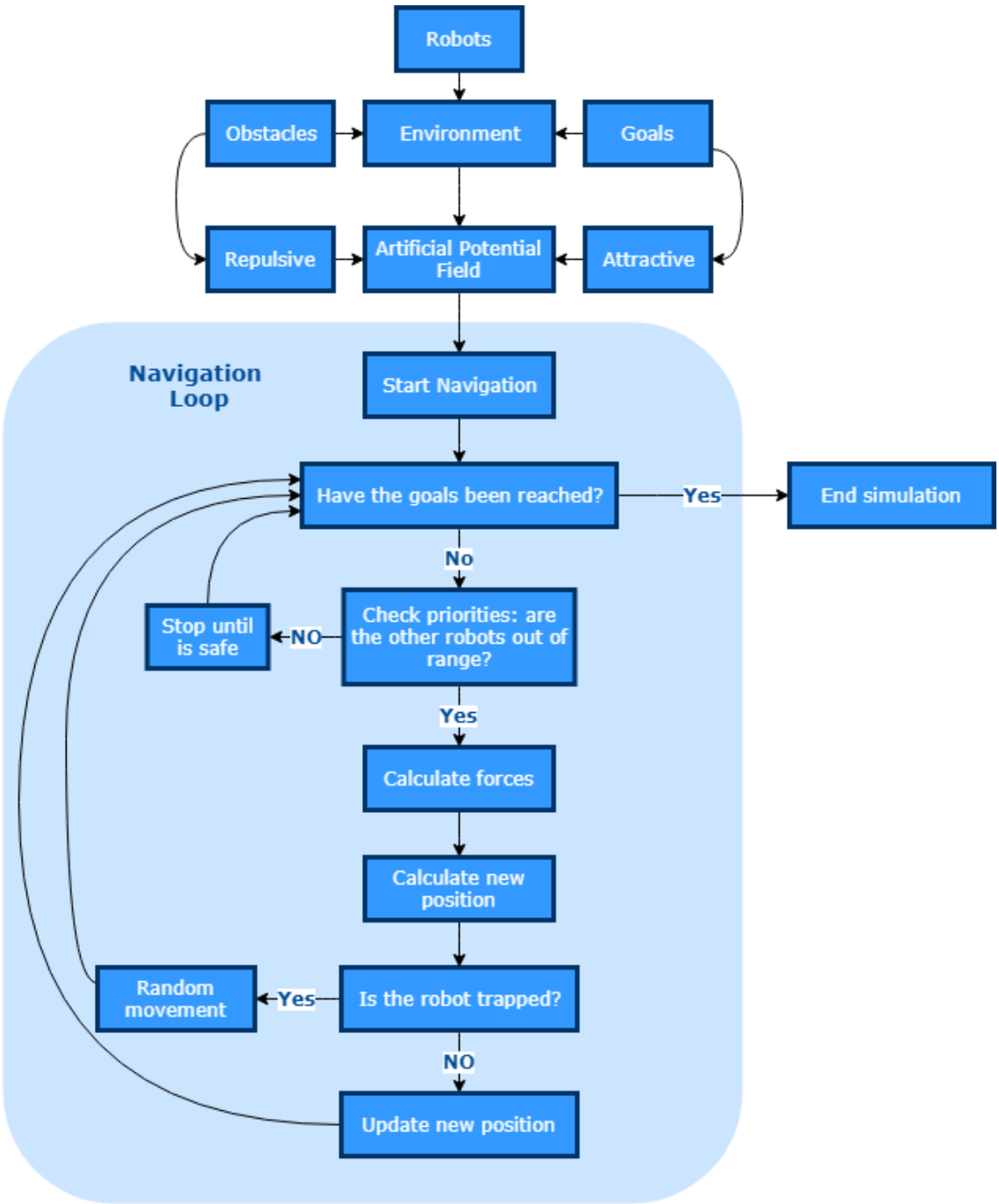


Figure 32: Simulation diagram

The inputs of the system are the initial positions and the parameters of the robots, goals and obstacles. Then the APF of the static obstacles is built and the navigation can begin. Inside the navigation loop the first thing that is checked for every robot is if it has reached its goal, if it does not then the method for collision avoidance is checked. This method consists of the establishment

of priorities between robots and the establishment of a safe range for navigation. Therefore, in each iteration the robot must check if there are robots with more priorities in the area around it defined by the safety range. If there are robots around, the robot with less priority stops its navigation until it is safe. Following with the sequence of the diagram, if it is safe to navigate the resultant of the forces applied on the robot is calculated and the new position. At this point is where the method for local minima avoidance is checked. The algorithm stores the previous position of the robot and compare it with the new one. If the robot is in the same position a counter starts. When this counter arrives until the value set as the maximum number of iterations trapped permitted a random movement is produced to release the robot from local minima. The new position of the robot is updated, and the loop begins again.

3.4. Software: MatLab

MatLab is a software developed by MathWorks that is the worldwide leader of mathematical calculation software for engineers and scientist (MathWorks, 2018). The company describes MatLab as "a high-level language for the expression of engineering and scientific ideas, and it is the computational engine that drives discovery and innovation".

This is the software chosen because it provides quick and efficient mathematical calculations, easy interface, good support, and great built-in graphics. All these qualities are important for the correct development of this project.

CHAPTER FOUR:
Simulation and results

4.1. Introduction

In this chapter the results of the project will be shown. Following the procedures and methods described in the last chapter, the navigation of three robots in known environment using APFs has been developed and tested.

To begin with, the design of the environment will be exposed. It has been inspired by a warehouse, where the robots are transporting materials while people or other robots might be moving around. It also has been designed to test the features introduced in the code such as the random movement to avoid local minima.

Then, the results of the simulations will be exposed. In first place, some preliminary simulations will be performed where the priorities between robots, or the influence of the variation of the parameters of the obstacles in the simulation, among others, are tested. Finally, the main simulation will be shown and commented.

The code of the simulation can be found in the Appendix A.

4.2. Environment design

The environment contains static and dynamic obstacles with different shapes and potential functions associated. These elements are easily editable in size, colour, number, position and coefficient of influence. On the other hand, the environment also contains the robots and their corresponding goals, but these ones do not offer that much editability. It is only possible to change their position and colour. All the details about these elements will be further developed in the next subsections.

To begin with, the dimensions of the environment must be stabilised. These boundaries are like the walls of the warehouse, so the robots must avoid collision with them too. For this reason, these elements have been modelled like rectangular obstacles.

The user only must introduce the dimensions wanted and the boundaries are printed, and their potential field calculated. In this case, the dimensions are 21x12.

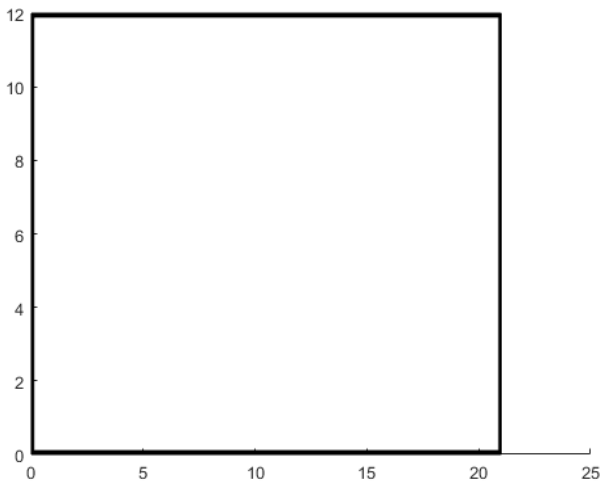


Figure 33: Environment boundaries 21x12

4.2.1. Obstacles

4.2.1.1. Rectangular

As it has been said before, these elements are modelled like the boundaries of the environment. The idea was to represent shelves containing materials in the warehouse.

To add a rectangular obstacle, the position of the left bottom corner of it must be introduced, also its width and height. In the next table can be found the information about the seven static rectangular obstacles in the environment.

Left Bottom Corner (x, y)		Width	Height
3	1,5	5	1
3	3,5	5	1
3	5,5	5	1
3	7,5	5	1
3	9,5	5	1
17	7,5	1	3
17	1,5	1	3

The next figure shows how the environment is build, in this moment it contains the boundaries plus the rectangular obstacles represented in cyan.

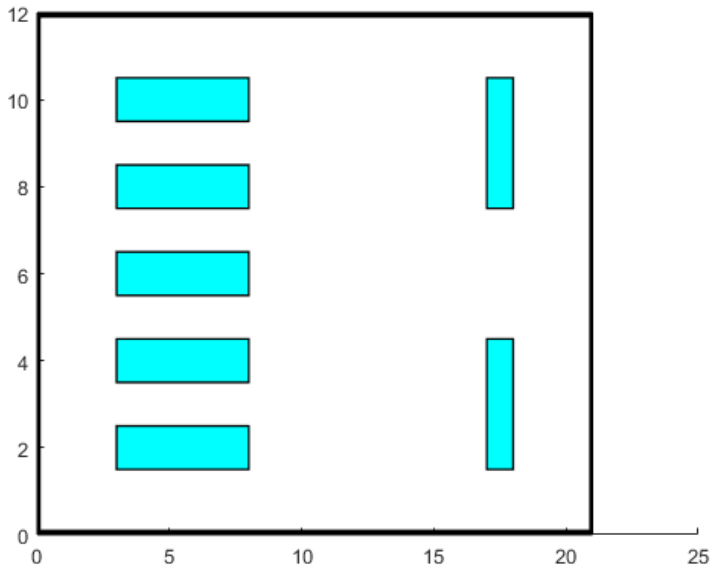


Figure 34: Boundaries and rectangular obstacles

4.2.1.2. Circular

4.2.1.2.1. Static

One of the purposes was to introduce different types of obstacles with different potential functions. The rounded obstacles are placed in the middle-right side of the environment and following with the analogy of the warehouse these elements can be considered pillars of the structure or goods stacked in the ground.

In the next table the parameters to define the circular obstacles are shown: the centre and radius of the obstacle, the number of points to draw the circle and the coefficient of influence.

Centre (x, y)		Radius	K	Points
11	6	0,5	0,35	100
11	9	0,5	0,35	100
11	3	0,5	0,35	100
14	7,5	0,5	0,35	100
14	4,5	0,5	0,35	100
14	10,5	0,5	0,35	100
14	1,5	0,5	0,35	100

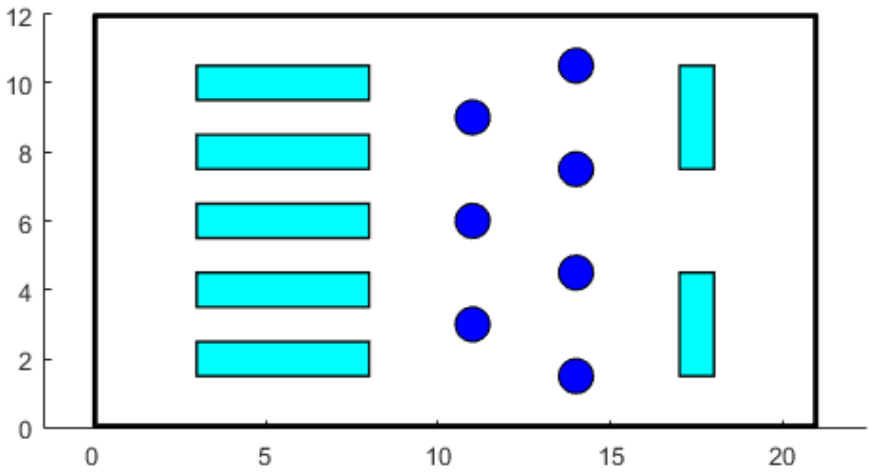


Figure 35: Boundaries and rectangular and static circular obstacles

4.2.1.2.2. Dynamic

Two circular dynamic obstacles have been introduced in the environment. One performs a linear horizontal movement between two shelves and the other is rounding one of the pillars. The properties are the same than the previous ones, and their movement is configured in the iteration loop.

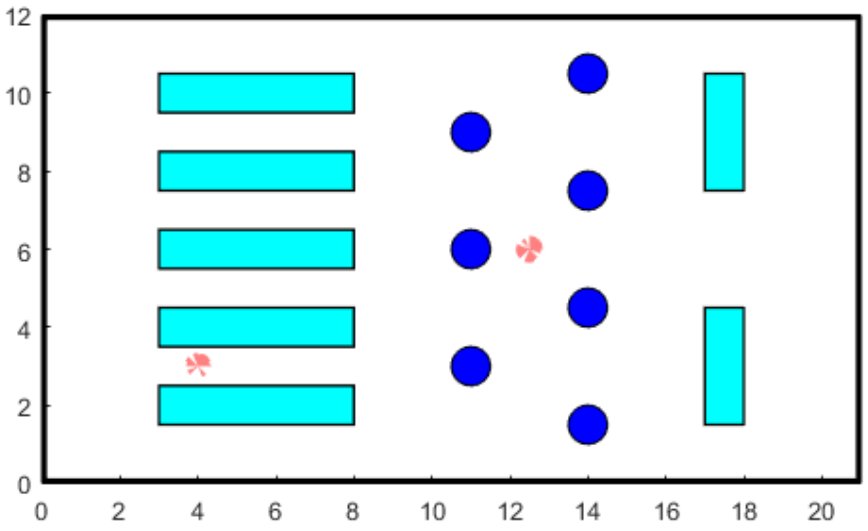


Figure 36: Boundaries and rectangular and circular obstacles.

4.2.2. Robots

The robots are represented as points situated at the left side of the environment. Their initial position is:

	Centre (x, y)	
Robot 1	1	2
Robot 2	1	6
Robot 3	1	10

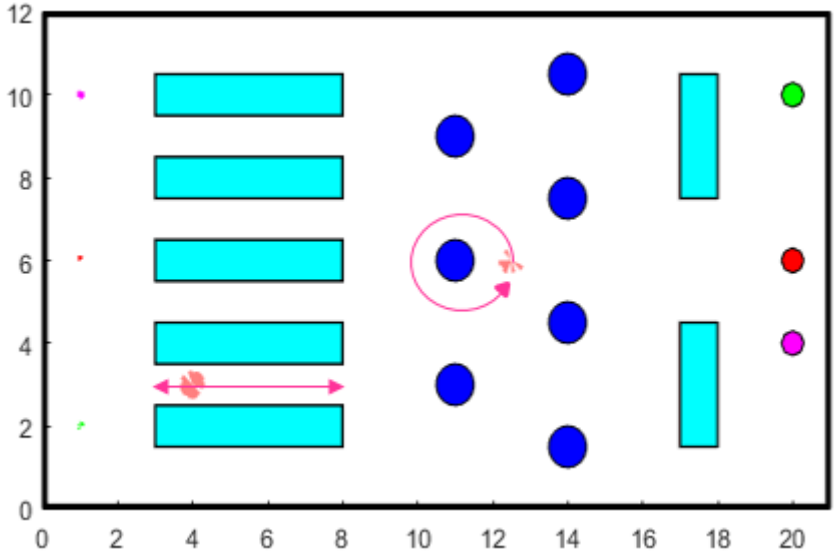


Figure 37: Boundaries, obstacles and robots

4.2.3. Goals

The definition of goals and circular obstacles is the same except for the number of points to draw the circle, in this case is inferior. The goals are matching in colour with the robot that must reach them.

	Centre (x, y)		Radius	K	Points
Goal 3	20	2	0,3	0,1	9
Goal 2	20	6	0,3	0,1	9
Goal 1	20	10	0,3	0,1	9

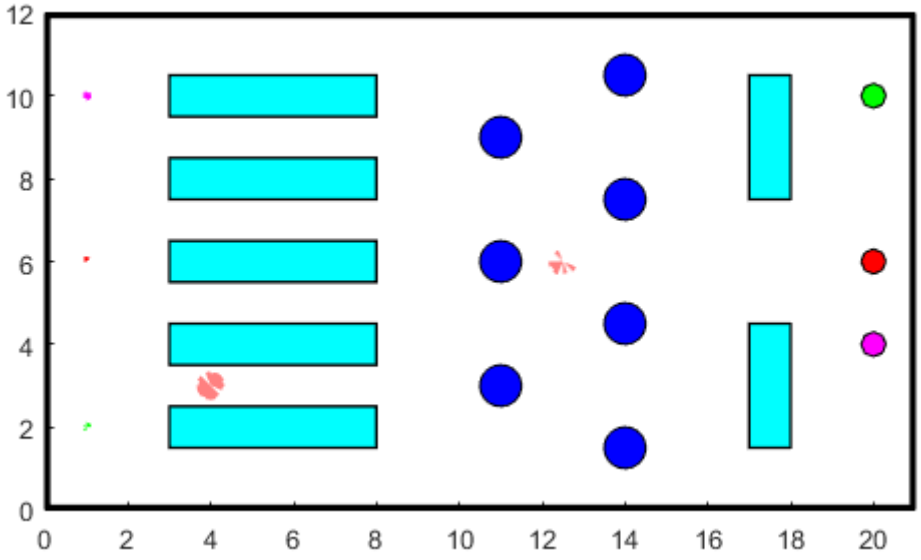


Figure 38: Complete environment. Boundaries, obstacles, robots and goals

4.2.4. Artificial Potential Field

Once the environment is defined, its artificial potential field is calculated according with all the potential functions on chapter 3 section 3.2.1, and the result are the following:

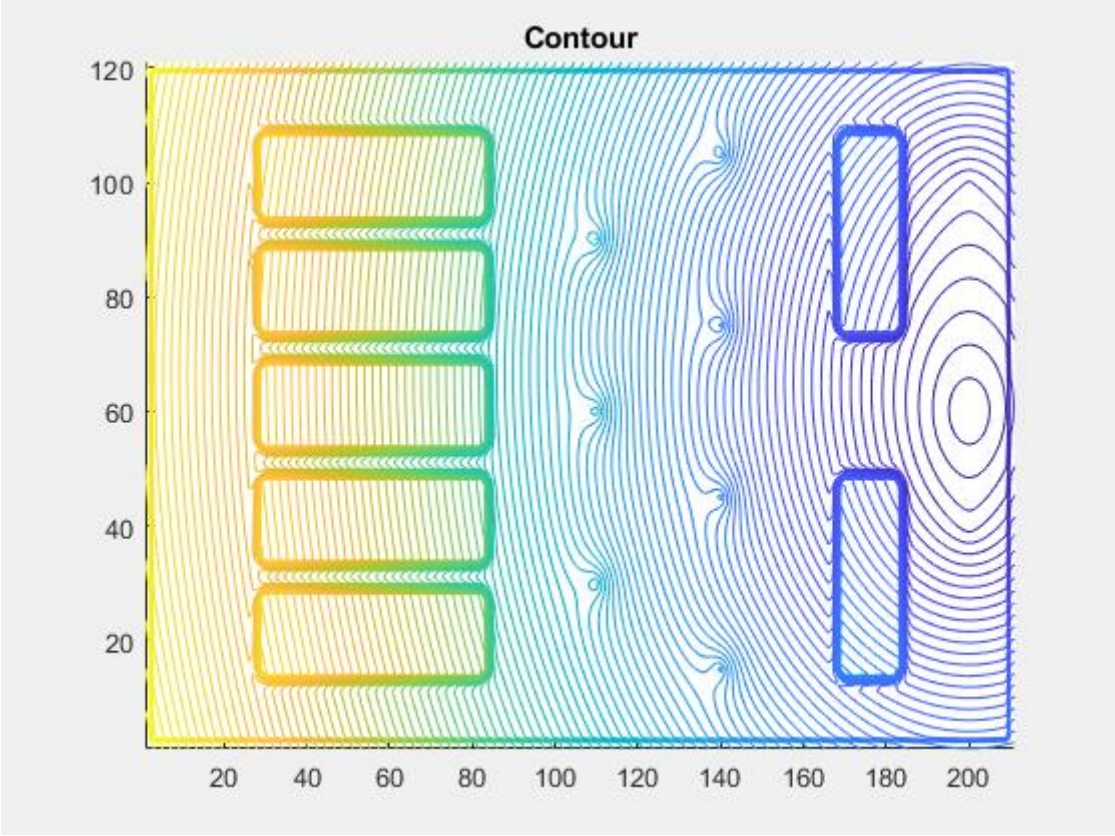


Figure 39: Contour of the environment

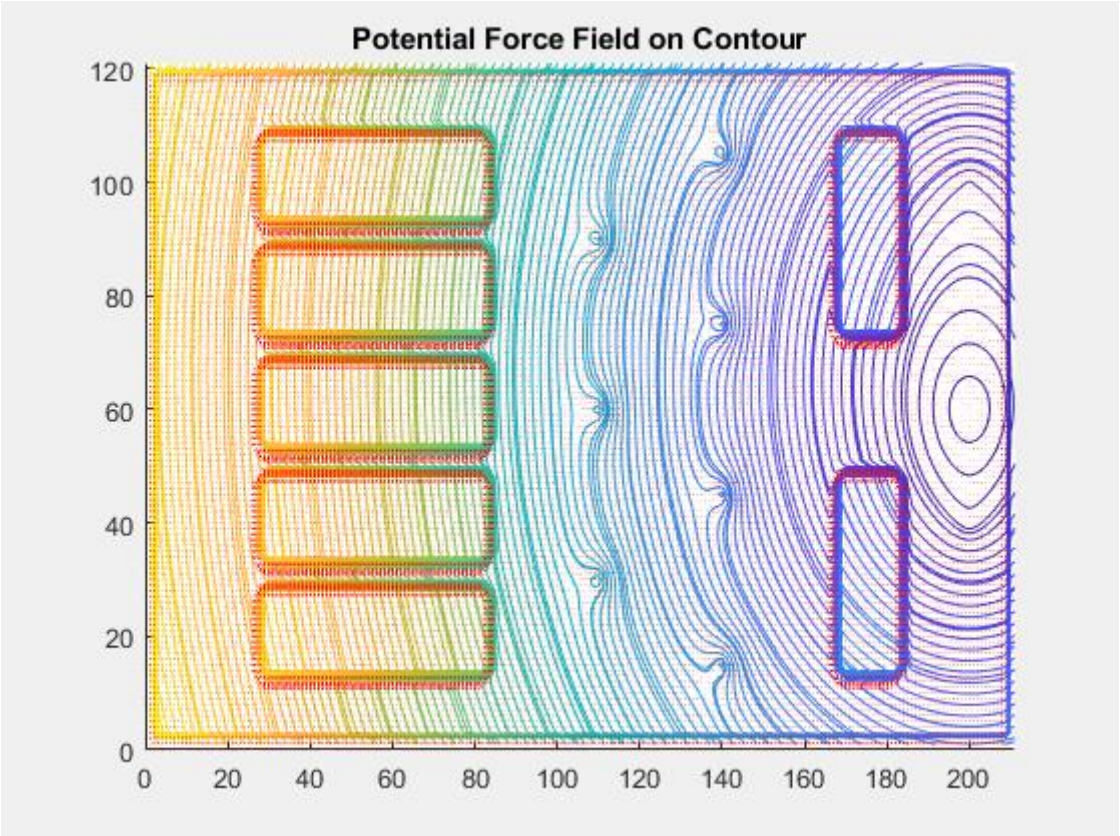


Figure 40: Contour and potential force

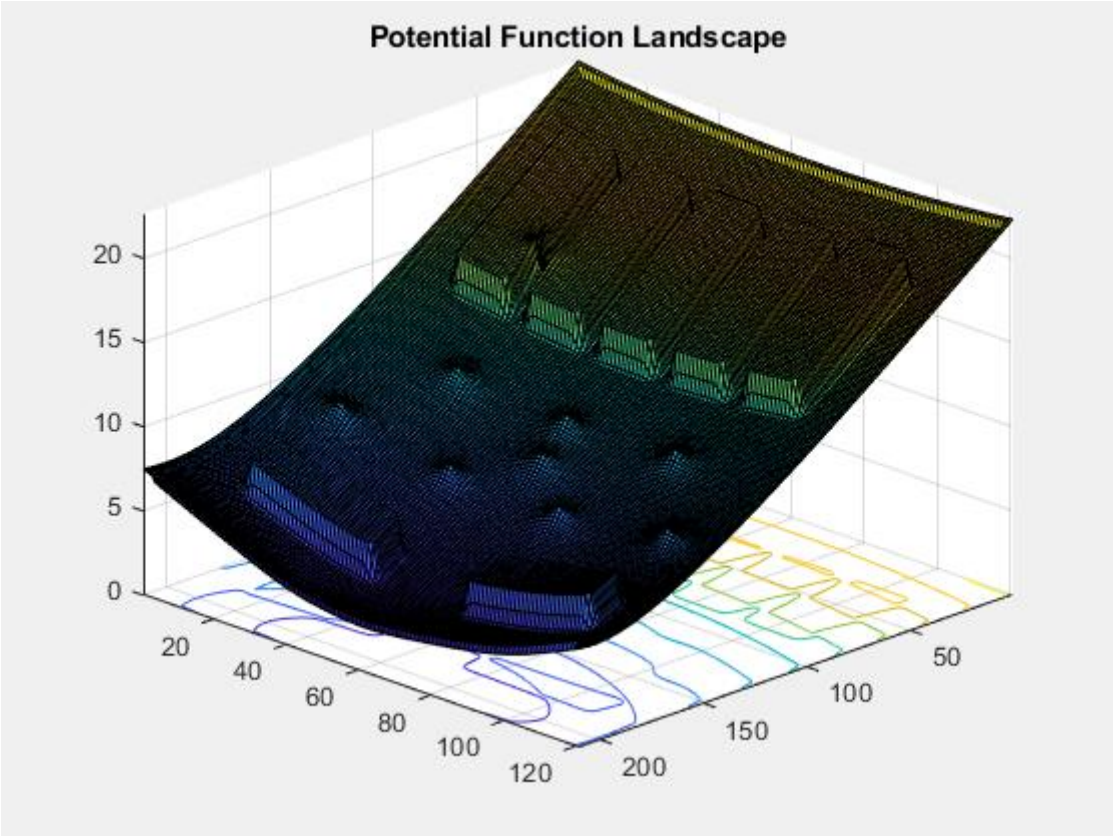


Figure 41: Potential field

4.3. Simulation

4.3.1. Test: Influence of the parameters of the potential functions

As it has been exposed before, different types of potential functions have been used and the variation of its parameters affects significantly its influence on the environment. Some examples are shown here.

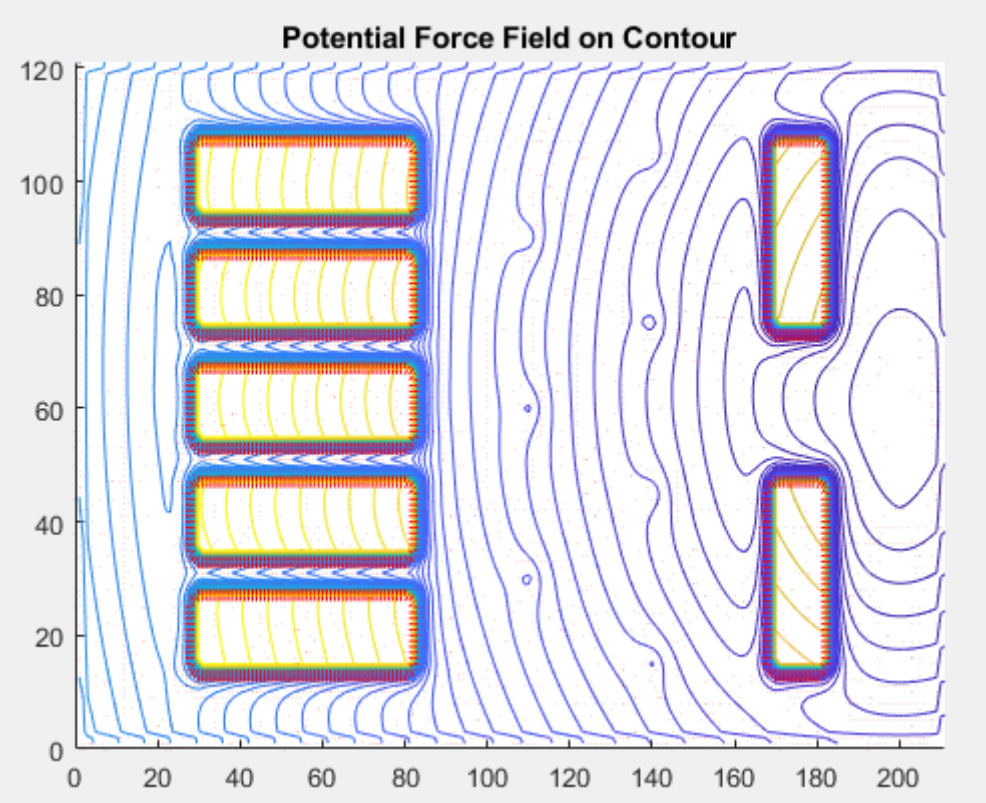


Figure 42: Contour and potential force high variation of the parameters of the rectangular obstacle.

Due to the potential functions defined for the obstacles are different there is the possibility of truncating the values in other to have all the potentials in the same range. In this test the truncation of the values has not been done to compe both situations, in Figure 40 the contour lines are closer whereas in Figure 42 the density of contour lines is concentred around the rectangular obstacles.

By comparing Figure 41with Figure 43 it is easy to appreciate this. In the first one the maximum value of the potential field for the rectangular obstacles is set in 2, however in the second figure it is not limited and values between 50 and 60 are reached.

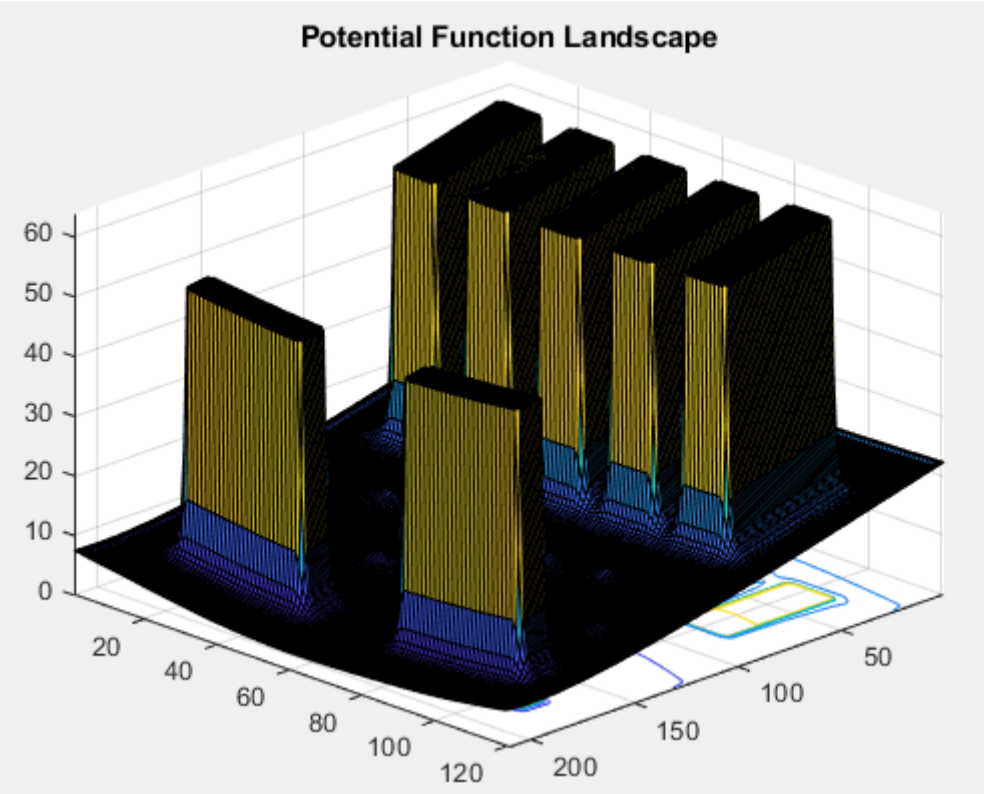


Figure 43: Potential Function. Influence of the parameters of the environment.

4.3.2. Test: Boundaries

The purpose of this test is to compare the performance of the simulation when the boundaries are considered another obstacle. The parameters of the rectangular obstacles have been modified to force the robots to cross the boundaries. In the second case the robots are trapped, and it is impossible to reach the goals.

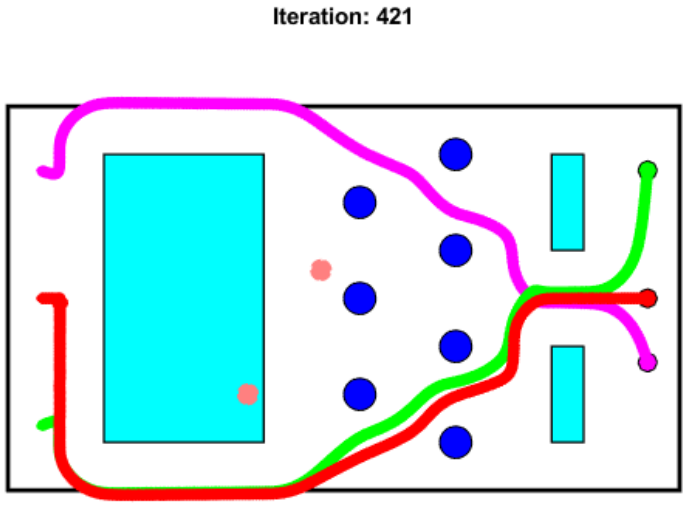


Figure 44: Environment without definition of boundaries.

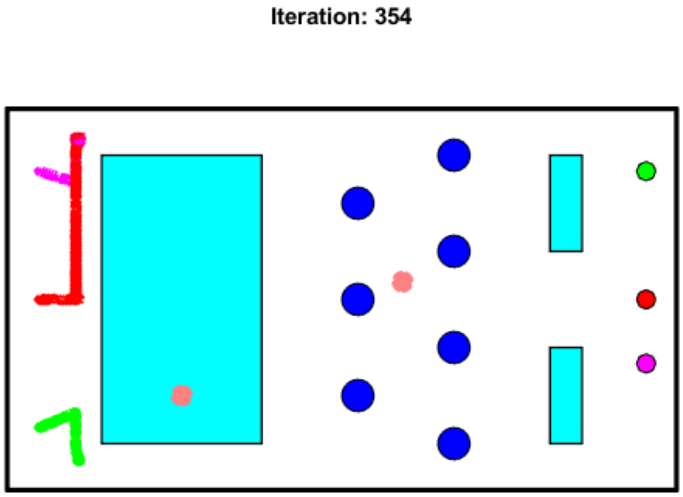


Figure 45: Comparison environments with and without boundaries

4.3.3. Test: Priorities between robots

This example has been created to test the performance of the collision avoidance between robots. In the next figures is it possible to see how the robots with less priority stop its movement until is safe to move. In both cases the same configuration of the environment and number of iteration.

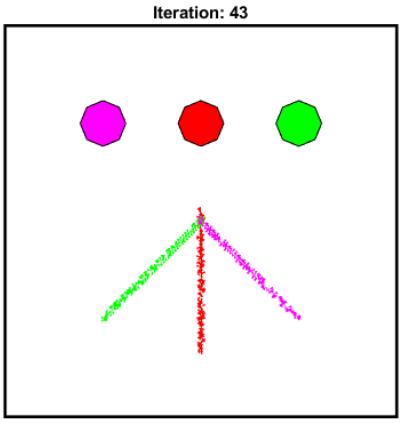


Figure 46: Without priorities

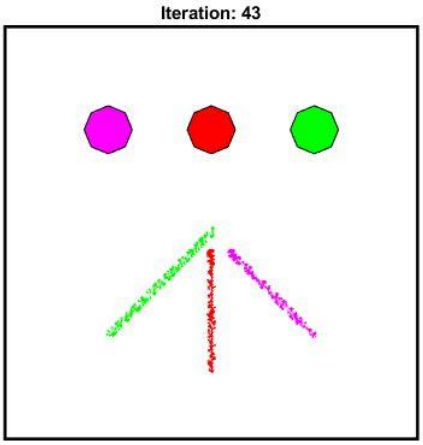


Figure 47: With priorities

4.3.4. Test: Local minima

The performance of the algorithm with and without the random movement to scape local minima are compared.

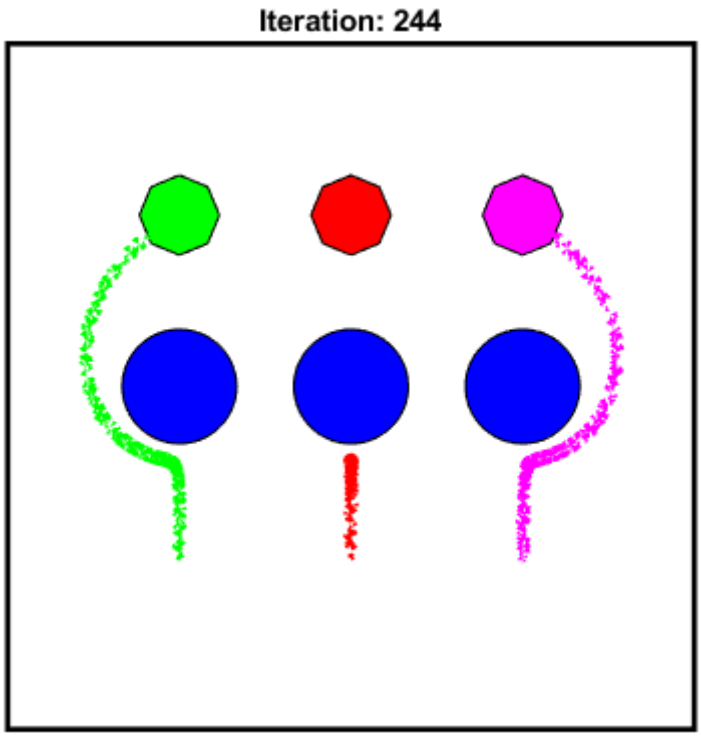


Figure 48: Red robot trapped

In Figure 48 the red robot gets trapped in local minima due to the resultant of the forces applied on it is nullify. In this case, the algorithm does not have the implementation of the local minima avoidance method, so the simulation will never stop, and the robot will remain in that position trapped. However, in Figure 49 it is shown how the robot has being able to reach its goal on iteration number 159.

The configuration of the environment in both cases is the same but with the implementation of the method, a maximum number of iterations stacked in the same position is set. If the robot is trapped more than the maximum allowed a random movement is produced to take it out of the singular point so the navigation can go on.

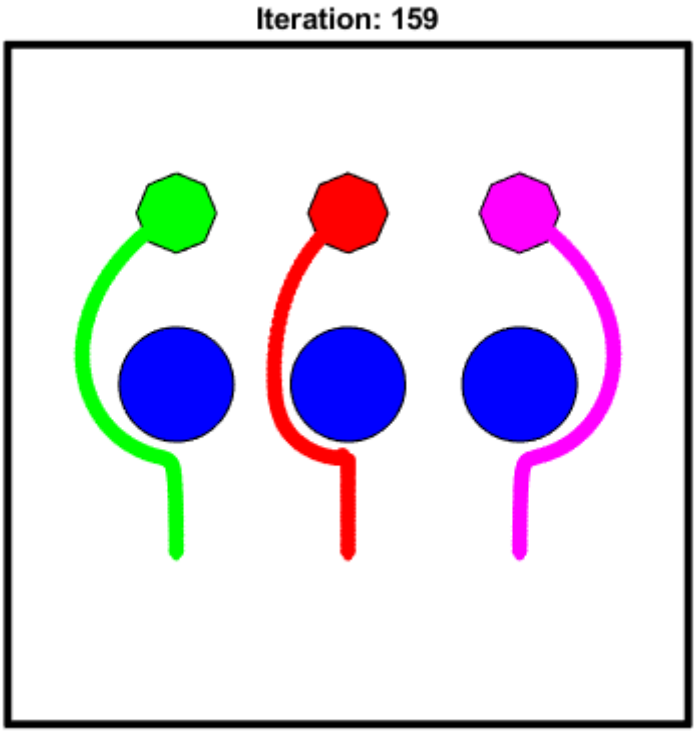


Figure 49: Red robot avoiding local minima

4.3.5. Test: GNRON

To test where this problem appears, some test has been performed using the same environment that is been designed for the main simulation. The position of the goals as has been moved into the direction of the boundaries until the goal is not reachable. The x coordinate of the goals in this case is 20.5 units and the boundaries are at x equal to 21.

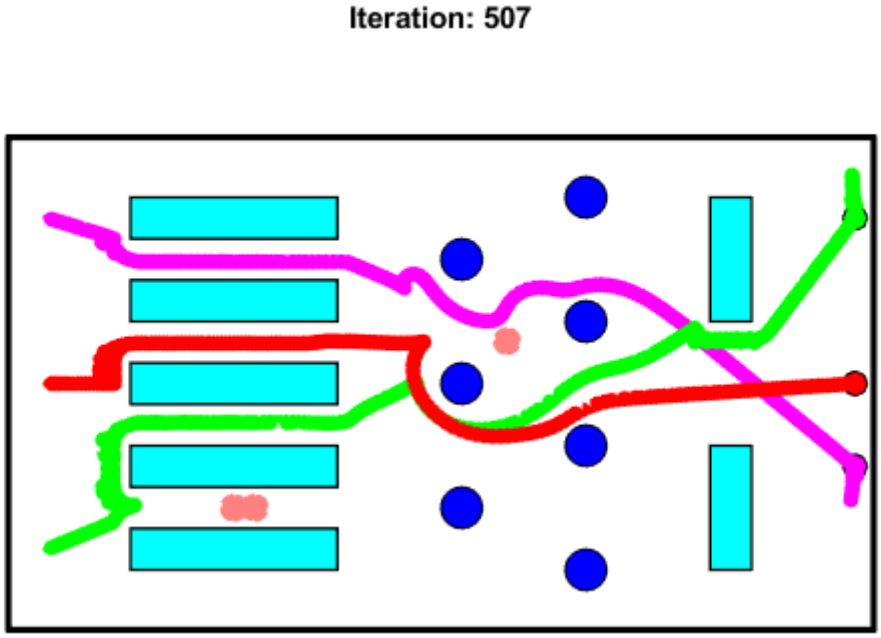


Figure 50: GNRON

It can be appreciated in the image above that the robots cannot reach the goals and they are doing random movements because they are "trapped" in the same position trying to reach it. The simulation usually ends around 400 iterations but when this problem occurs it may be stopped manually.

4.3.6. Main simulation

In this section, the results of the simulation will be commented. Due to the simulation last around 400 iterations part of the sequence can be found in the appendix B for both cases, bidimensional and three-dimensional path planning.

The result is shown in the next figure, where it can be appreciated that every robot has reached its goal without colliding with the obstacles. Later, the most relevant parts will be shown in detail.

The environment was design to challenging, that is the reason why the red robot and its goal are placed just in the symmetry axis of the figure, so it will be trapped into local minima. Furthermore, the dynamic obstacles have been placed to obstruct the path of the robots and make them change direction. It can be clearly seen in the centre of the figure where the obstacle moving in circles has produced changes in the trajectories of all the robots.

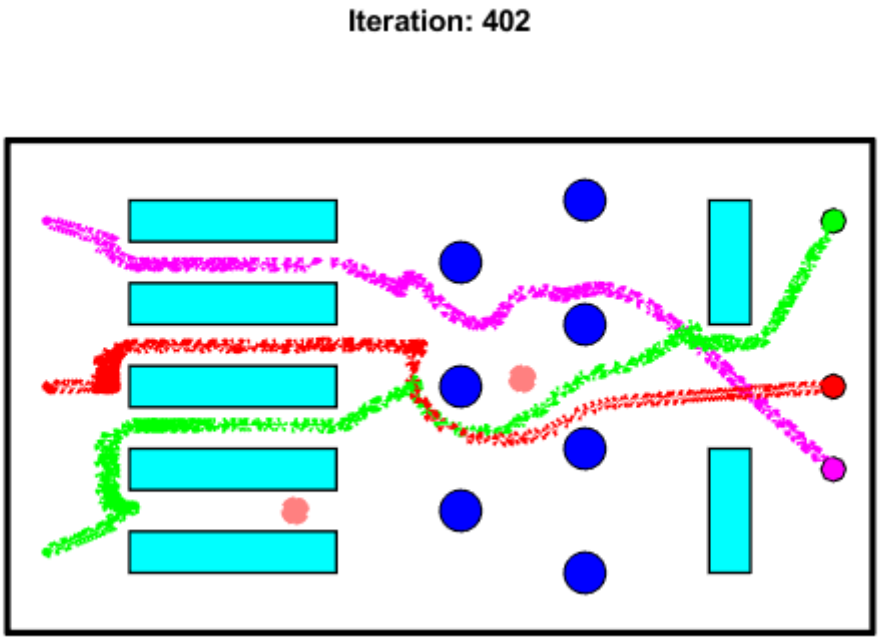
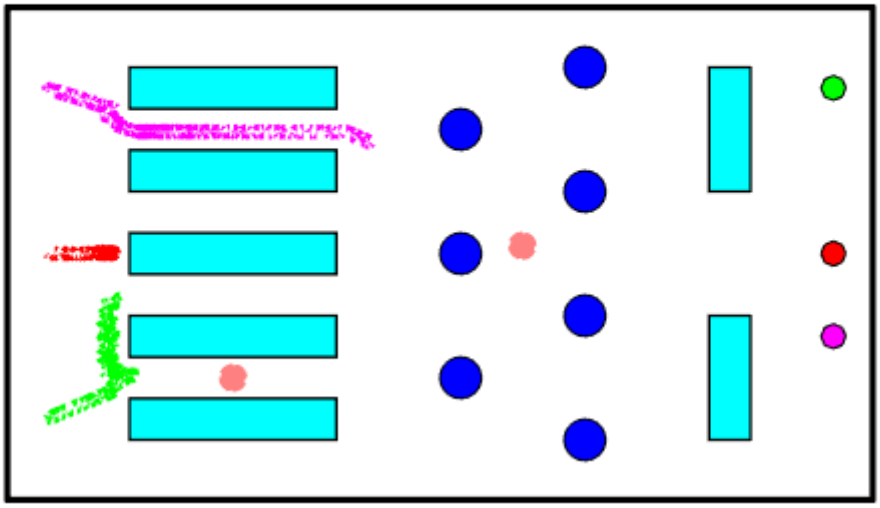


Figure 51: 2D simulation result

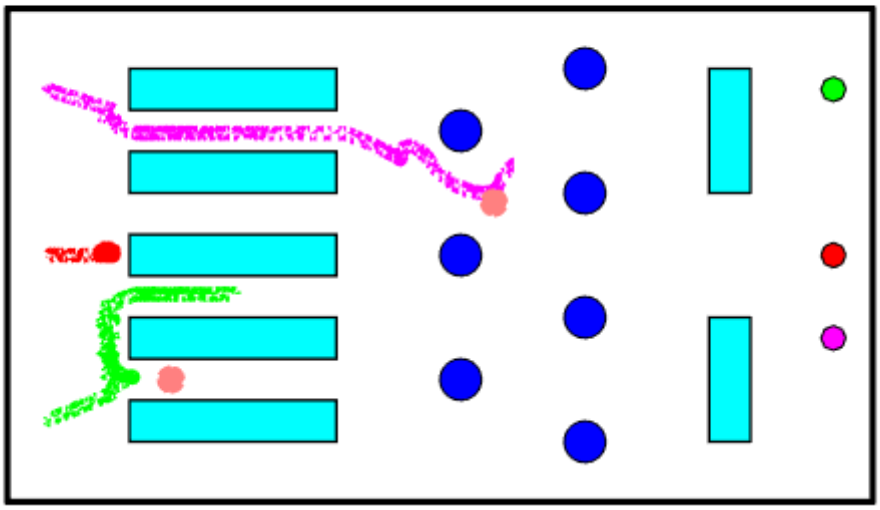
To begin with, in the next image is shown the beginning of the navigation. As it has been commented before, the red robot is trapped in local minima and it will get out of it slowly. In the loop the actual position of the robot is compared with the previous ones and if it is in the same position for more than two iterations it will move randomly. The number of iterations trapped can be edited and the maximum random movement. Regarding the green robot, it changes its direction because the moving obstacle is obstructing its path.

Iteration: 102



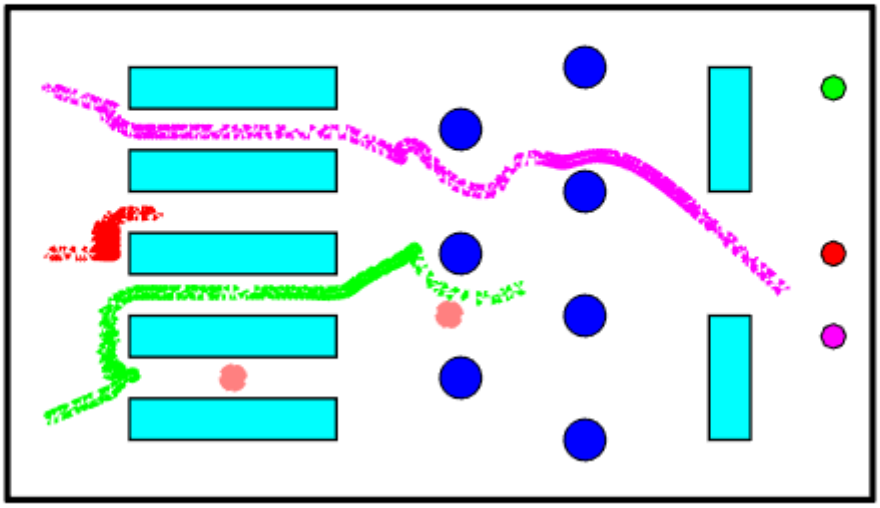
In the next figure is shown how the magenta robot has changed its trajectory twice to avoid collision with the moving obstacle that is moving in circles in anti-clockwise direction around the circle in the midle. The red robot is getting out of the trap.

Iteration: 159



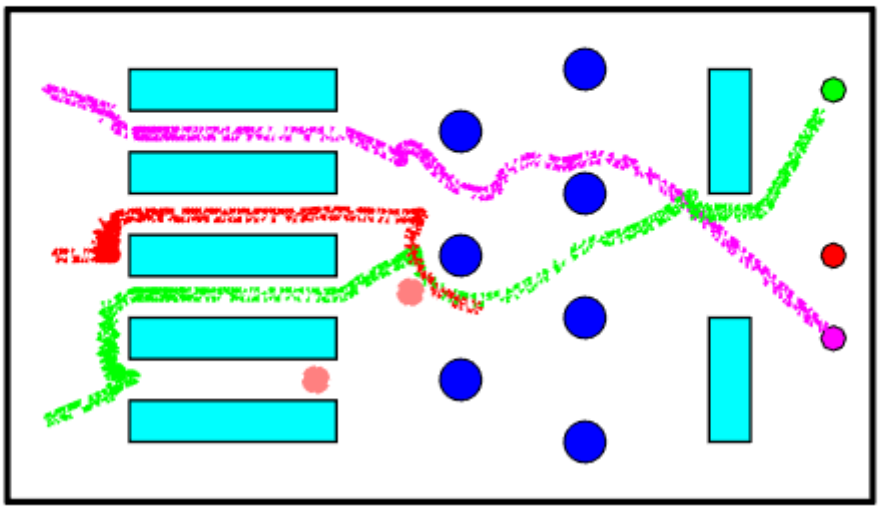
Now the green robot has reached the area where the moving obstacle is, and it was going to round the blue obstacle in clockwise direction, but the dynamic obstacle made it turn around in the other direction. The red one is out of the local minima.

Iteration: 237



In the next image is the turn of the red robot to avoid the moving obstacle and the situation is the same as the one described before. The magenta robot has reached its objective and the green one has avoided its last obstacle and is about to reach its goal.

Iteration: 331



Finally, the simulation has also been done in 3D and it is represented in the next figure:

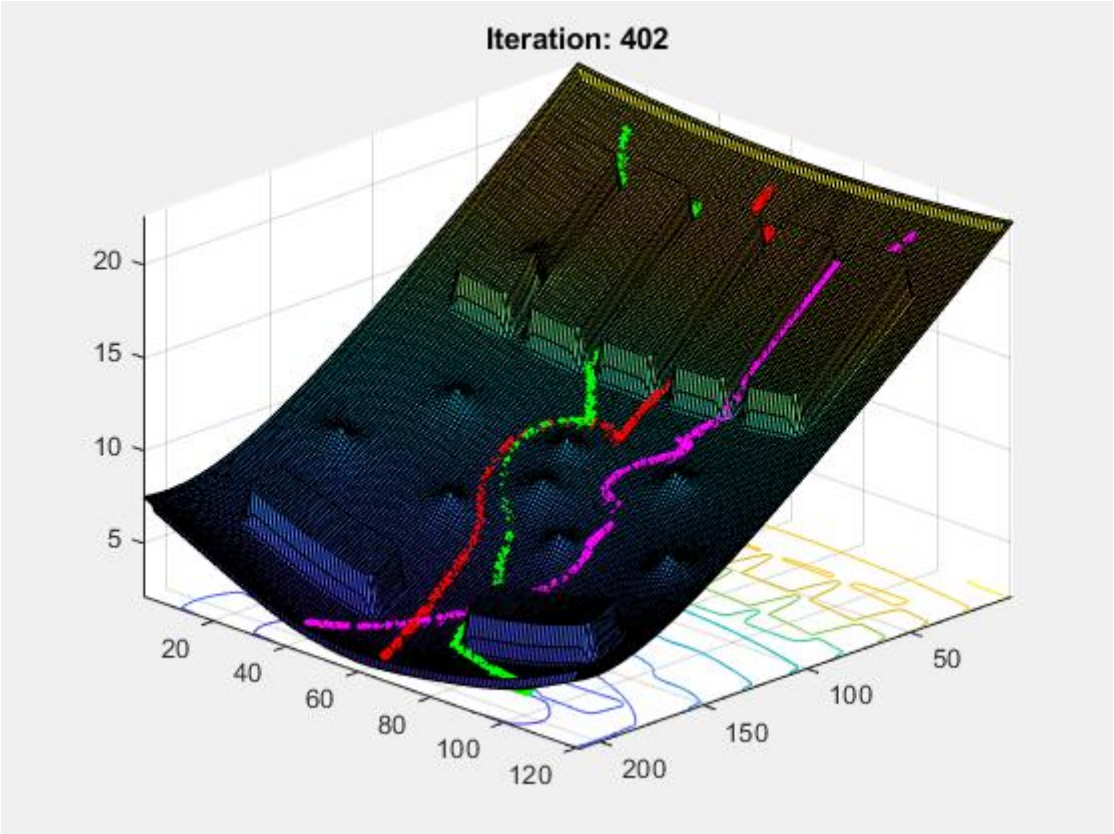


Figure 52: 3D simulation result

4.3.7. Main simulation without moving obstacles

With the aim to compare, the same simulation has been realized without the moving obstacles and there are many differences. The simulation is completed in less iterations and the robots are going direct to its goals. It is interesting that the green robot avoids going between the two lower shelves, so in the main simulation the robot goes between the second and the third because it is the optimal path not because of the moving obstacle.

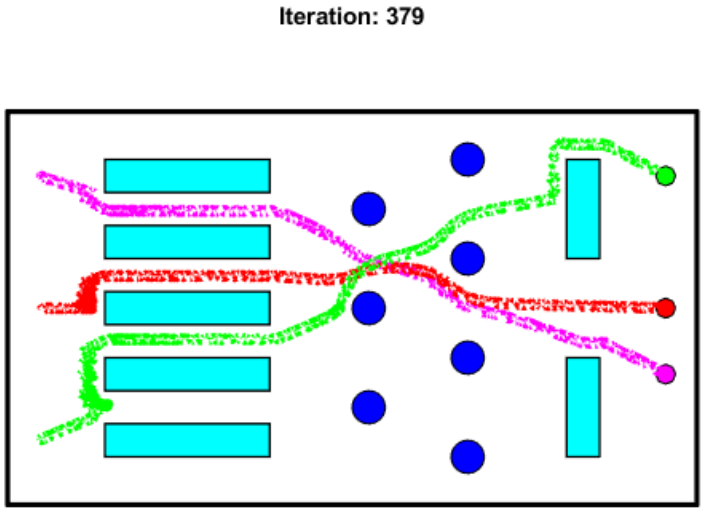


Figure 53: Main simulation without dynamic obstacles

CHAPTER FIVE: Conclusion

5.1. Conclusion

Based on the results of this project, diverse conclusions can be drawn.

To begin with, it can be considered that the main purpose of this dissertation has been achieved, due to it has been possible to successfully perform the simulation of the path planning algorithm using APFs. The results show how the tree robots move through the environment avoiding collisions with the different types of dynamic and static obstacles and reaching its goals.

Secondly, the features of the simulation have been discussed individually. These improvements have been implemented in order to solve some of the inherent problems of the APF method like local minima, or to upgrade the simulation by making it more realistic such as the system of priorities between robots or the consideration of the limits of the environment as obstacles.

Regarding local minima, it must be remark that the method implemented is the simplest of them all. For this reason, it has good performance in simple configurations, however in more complex configurations like U-shaped or L-shaped obstacles the robot will remain trapped. For example, it works appropriately when the environment is symmetric, and the robot has two optimal paths to reach the goal. Such as the red robot in the main simulation, in this situation, when the random movement is produced, the robot gets out of the crossroads and it only has one of the two paths close.

On the other hand, the upgrades introduced in this specific configuration are beneficial for the general performance of the algorithm, but in other situations can generate problems. Regarding the consideration of the boundaries has obstacles, it is clear that collision avoidance against the walls or the limits of the environment is a useful upgrade, but it can introduce another of the inherent problems of the APF method, the Goal Non-Reachable with Obstacle Nearby (GNRON). Owing to the limits are obstacles if the goal is to close, it may become unreachable.

Finally, talking about the establishment of priorities between robots, it must be remark that is beneficial when the number of robots is reduced. The main disadvantage of this method is that every robot must be programmed specifically to sense if the robots with more priority are close, and if they are close, the robot with less priority will stop its navigation. However, this procedure can lead to collision.

To conclude, it must be reminded that mobile robotics is a very promising field where lots of researches and developments are taken place nowadays. The motivation behind this project It must be highlighted, which has been to acquire and apply knowledge in the field of mobile robotics, to be able someday to contribute to its development and applications.

5.2. Future work

Based on the results and the test performed in chapter four, the problems of the algorithm developed have been identified in the conclusion, and its possible solution will be listed below as possible future work:

- To upgrade the local minima avoidance method.
- To stablish a better method for collision avoidance between robots with easiest scalability.
- To solve the problem of GNRON.

Furthermore, the next step would be to test the algorithm in the real world and compare its performance with the simulation.

References

Alboul, L., Penders, J., Saez-Pons, J., Herbrechtsmeier, S., Witkowski, U., El-Habbal, M., and Naghsh, A. (2011). *A robot swarm assisting a human fire-fighter*. Advanced Robotics, vol. 25, no. 1, pp. 93–117, 2011.

Alboul, L. (2018) *Robotics material.* Unpublished manuscript.

Amiryan, J. and Jamzad, M. (2015). *Adaptive Motion Planning with Artificial Potential Fields Using a Prior Path.* Proceedings of the 3rd RSI International Conference on Robotics and Mechatronics. October 7-9, 2015, Tehran, Iran

Arai, T., Pagello, E. and Parker L.E. (2002) *Advances in Multi-robot Systems.* IEEE Transactions on Robotics and Automation, vol. 18, no. 5, Oct 2002.

Aurenhammer, F. (1991). *Voronoi diagrams—a survey of a fundamental geometric data*

structure.

Balch, T. and Arkin, R.C. (1998). *Behavior-based formation control for multirobot teams*. IEEE Transactions on Robotics and Automation, vol. 14, no. 6, pp. 926-939, Dec 1998.

Batalin, M. A., Sukhatme, G. S. and Hattig, M. (2004). *Mobile robot navigation using a sensor network*. Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference, pp. 636-641 Vol.1.

Bayindir, L. and Sahin, E. (2007). *A Review of Studies in Swarm Robotics.* Turk J Elec Engin, vol.15, no 2.

Bermudez, G. (2018). *Robots móviles. Teoría, aplicaciones y experiencias*. Retrieved from

https://www.researchgate.net/publication/267798579_ROBOTS_MOVILES_TEORIA_APLICACIONES_Y_EXPERIENCIAS [Accessed 9 May 2018].

Bräunl, T. (2008). *Embedded robotics*. Berlin: Springer.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm intelligence: From Natural to Artificial Systems.* Oxford University Press.

Conde-Canaviri, M (n.d.). *Generaciones de La Robótica*. Retrieved from http://www.revistasbolivianas.org.bo/pdf/rits/n1/n1a32.pdf [Accessed 8 May 2018].

Digani, V., Sabattini, L., Secchi, C. and Fantuzzi, C. (2014). *An automatic approach for the generation of the roadmap for multi-AGV systems in an industrial environment*. International Conference on Intelligent Robots and Systems, Chicago, IL, 2014, pp. 1736-1741.

El País (2018). Así es el primer insecto robótico que vuela solo. [online] EL PAÍS. Available at: https://elpais.com/tecnologia/2018/05/22/actualidad/1526997567_208779.html [Accessed 30 May 2018].

Encyclopaedia Britannica (2018). *Artificial Intelligence*. Retrieved from

https://www.britannica.com/technology/artificial-intelligence

Encyclopaedia Britannica (2018). *Heron of Alexandria*. Retrieved from https://www.britannica.com/biography/Heron-of-Alexandria

Encyclopaedia Britannica (2018). *Robot*. Retrieved from

https://www.britannica.com/technology/robot-technology

Fahimi, F. (2010*). Autonomous robots. Modelling, Path Planning, and Control*. New York: Springer.

Ge, S. and Lewis, F. (2006). *Autonomous mobile robots*. Boca Raton, FL: CRC/Taylor & Francis.

Gonzalez, R., Kloetzer, M. and Mahulea, C. (2017). *Comparative study of trajectories resulted from cell decomposition path planning approaches*. 2017 21st International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, pp. 49-54.

Hall, E. and Hall, B. (1985). *Robotics, a user-friendly introduction*. New York: Holt, Rinehart, and Winston.

Honda (2018). *History of ASIMO Robotics | ASIMO Innovations by Honda*. Retrieved from http://asimo.honda.com/asimo-history [Accessed 10 May 2018].

International Organization of Standardization (2012). *Robot and robotic devices.* Retrieved from https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en [Accessed 10 May 2018].

Kaplan, L.J. (2006). *Robots and Humans*. In: Cultures of Fetishism. Palgrave Macmillan, New York.

Kloetzer, M., Mahulea, C. and Gonzalez, R. (2015). *Optimizing cell decomposition path planning for mobile robots using different metrics.* 2015 19th International Conference on System Theory, Control and Computing (ICSTCC), Cheile Gradistei, 2015, pp. 565-570.

Koren, Y. and Borenstein, J. (1991). *Potential Field Method and Their Inherent Limitations for MobileRobot Navigation*. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Sacramento, 1991, pp. 1398-1404.

Latombe, J. (2010). *Robot motion planning*. Boston: Kluwer.

Lee, D., Jeong, J., Kim, Y. H. and Park, J. B. (2017). *An improved artificial potential field method with a new point of attractive force for a mobile robot*. 2017 2nd International Conference on Robotics and Automation Engineering (ICRAE), Shanghai, 2017, pp. 63-67.

Lima, P. and Custodio, L. (2015). *Multi-robot systems*. Institute of Systems and Robotics.

Luo, R. C., Su, K. L., Shen, S. H. and Tsai, K. H. (2003). *Networked intelligent robots through the Internet: issues and opportunities*. Proceedings of the IEEE, vol. 91, no. 3, pp. 371-382, Mar 2003.

MathWorks (2018). https://es.mathworks.com

Olfati-Saber, R., Fax, J., and Murray, R.M. (2006). *Consensus and Cooperation in Networked Multi-Agent Systems.* Proceedings of the IEEE, April 2006

Ollero Baturone, A. (2007). *Robótica. Manipuladores y robots móviles*. Mexico: Alfaomega.

Ramesh, J. and Kasturi, R and Schunk, B. G. (1995) *Machine Vision*.

Reif, J. H. and Wang, H. (1999). *Social potential fields: A distributed behavioral control for autonomous robots*. English, Robotics and Autonomous Systems, vol. 27, no.3, pp. 171–194, 1999.

Rubenstein, M., Cornejo, A., and Nagpal, R. (2014). *Programmable Self-Assembly in a Thousand-Robot Swarm.*

Sánchez-Martín, F.M., Millán Rodríguez, F., Salvador Bayarri, J., Palou Redorta, J., Rodríguez Escovar, F., Esquena Fernández, S., & Villavicencio Mavrich, H. (2007). *Historia de la robótica: de Arquitas de Tarento al robot Da Vinci (Parte I)*. Actas Urológicas Españolas, Vol. 31(2), 69-76.

Sánchez-Martín, F.M., Jiménez Schlegl, P., Millán Rodríguez, F., Salvador-Bayarri, J., Monllau Font, V., Palou Redorta, J., & Villavicencio Mavrich, H. (2007). *Historia de la robótica: de Arquitas de Tarento al Robot da Vinci (Parte II)*. Actas Urológicas Españolas, Vol. 31(3), 185-196.

Siegwart, R. and Nourbakhsh, I. (2004). Introduction to autonomous mobile robots. Cambridge, Mass.: MIT Press.

Tanwani, A. K., and Calinon, S. (2016). Learning robot manipulation tasks with task-parameterized semitied hidden semi-markov model. *IEEE Robotics and Automation Letters, 1*(1), 235-242.

Tzafestas, S. G. (2013). *Introduction to mobile robot control.* Elsevier.

Valero, R., Ko, Y.H., Chauhan, S., Schatloff, O., Sivaraman, A., Coelho, R.F., … Patel, V.R. (2011). *Robotic Surgery: History and Teaching Impact.* Actas Urológicas Españolas, Vol. 35(9), 540-545.

Wang, M., Su, Z., Tu, D. and Lu, X. (2013). *A hybrid algorithm based on Artificial Potential Field and BUG for path planning of mobile robot.* Proceedings of 2013 2nd International Conference on Measurement, Information and Control, Harbin, 2013, pp. 1393-1398.

Winfield, A. F. and Nembrini, J. (2006). *Safety in numbers: Fault-tolerance in robot Swarms*. International Journal of Modelling, Identification and Control, vol. 1, no. 1, pp. 30–37, 2006.

Xu, D., Zhang, X., Zhu, Z., Chen, C., and Yang, P. (2014). *Behavior-Based Formation Control of Swarm Robots.* Mathematical Problems in Engineering, vol. 2014, Article ID 205759, 13 pages, 2014.

Xu, Q. L., Yu, T. and Bai, J. (2017). *The mobile robot path planning with motion constraints based on Bug algorithm.* 2017 Chinese Automation Congress (CAC), Jinan, 2017, pp. 2348-2352.

Yates, D. R., Vaessen, C., and Roupret, M. (2011). *From leonardo to da vinci: The history of robot-assisted surgery in urology*. BJU International, Vol. 108(11), 1708-1713.

Yu, T., Yasuda, T. and Ohkura, K. (2015). *A duration based behavior analyze approach for swarm robotics system*. 54th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), Hangzhou, 2015, pp. 276-281.

Zalvanos, M. and Pappas, G. (2007). *Potential Fields for Maintaining Connectivity*

of Mobile Networks. IEEE Transactions on Robotics, vol. 23, no. 4, August 2007.

Zavlanos, M., Kumar, V. and Pappas, G. (2009). *Maintaining Connectivity in Mobile Robot Networks*

APPENDIX A: MatLab code

1. Main: Environment.m

```
clf %clears current figure window
close all %close everything
clear all %removes all variables

%% Properties of the figure and definition of the environment
% Select the background color of the figure
figure('Color', 'w');
% Introduce the dimensions of the environment
Env= [0 0 21 12];
% Enter the distance between points
dgrid = 0.1;
x = 0:dgrid:Env(3); % Array with the x values
y = 0:dgrid:Env(4); % Array with the y values
% Matrix that contains all the coordinates of the points of the
environment
[xx,yy] = meshgrid(x,y);

%% Enviroment boundaries
% In this section the limits of the environment are drawn according to
% the dimentions introduced above and its potential field is
calculated.

% Introduce the line width of the boundaries
lw_env=0.1;

% COORD_ENV is an array with values of the coordinates of the left
corners
% of the 4 rectangles used as boundaries of the environment build
according
% to the coordinates and the line width introduced. In each row j, x-
and
% -y coordinates of jth obstacle are stored.
COORD_ENV=[Env(1) Env(2)
    Env(3)-lw_env Env(2)
    Env(1)+lw_env Env(4)-lw_env
    Env(1)+lw_env Env(2)];

% W_ENV is an array with values of the widths of boundaries. It is
% build it is built from the data introduced above. In each row j, the
% width  of jth obstacle are stored.
W_ENV=[lw_env
    lw_env
    Env(3)-2*lw_env
    Env(3)-2*lw_env];

% H_ENV is an array with values of the heights of boundaries.
% In each row j, the height  of jth obstacle are stored.
H_ENV=[Env(4)
    Env(4)
    lw_env
    lw_env];
```

```
% Definition of parameters:
% The obstacle influences  only a domain around it within the
% distances between 'roinv' and 'romin'.
roinv =0.1;
romin =0.01;
% lw Width of the boundary segments of an obstacle
lw =0.1;
% Curv is the corvature of an obstacle, in this case is 0 because the
% boundaries are defined as rectangles.
Curv = [0,0];
% strCol determines the colour of the internal domain of an obstacle.
% Currently boundaries are painted black
strCol = 'k';
%Truncate the values of the field obtained
t_env=0.5; % Limit value

%determines the size of the rows in COORD_ENV
[rC_env,cCe] = size(COORD_ENV);

UREP =zeros(numel(y),numel(x));
% creating the initial matrix of the repulsive potential,
% with the size correspinding to  the size of the matrix of points in
% the grid. Initially all values in UREP are zeros. This
% matrix will accumulate the values of the repulsive
% potentials generated by each obstacle, to obtain at the
% final step the repulsive potential of the environment

for i =1:rC_env
    %In this loop invokes the function 'potrectonstaclenew' for each
    %obstacle, which places the obstacle in the environment and
computes
    %its repulsive potential
Urep = ProtectObsTruncate(COORD_ENV(i,:),W_ENV(i,:), H_ENV(i,:),
Curv,roinv,romin,lw, strCol,x,y,xx,yy,t_env);
UREP = UREP+ Urep;
end
% zz_ENV is a matrix that contains the values of the potential field
% produced by the boundaries in the environment.
zz_ENV = UREP;

%Truncate the resultant of the sum of the potential fields of each of
the
%rectangules that conform the boundaries of the environment
[m,n]=size(zz_ENV);
for i=1:m
    for j=1:n
        if zz_ENV(i,j)>t_env
            zz_ENV(i,j)=t_env;
        end
    end
end

%% Rectangular Obstacles
% In this section rectangular obstacles are defined and placed in the
% envirnoment, its potential field is calculated.

% COORDCORN is an array with values of the coordinates of the left
corners
```

```
% of obstacles. In each row j, x- and -y coordinates of jth obstacle
are
% stored.
COORDCORN=[3 1.5
    3 3.5
    3 5.5
    3 7.5
    3 9.5
    17 7.5
    17 1.5];
% W is an array with values of the widths of obstacles. In each row j,
the
% width  of jth obstacle are stored.
W=[5
    5
    5
    5
    1
    1];
% H is an array with values of the heights of obstacles. In each row
j, the
% height  of jth obstacle are stored
H=[1
    1
    1
    1
    3
    3];

% Definition of parameters:
% The obstacle influences  only a domain around it within the
% distances between 'roinv' and 'romin'.
roinv =0.5;
romin =0.1;
% lw Width of the boundary segments of an obstacle
lw =1;
% Curv is the corvature of an obstacle, in this case is 0 because the
% boundaries are defined as rectangles.
Curv = [0,0];
% strCol determines the colour of the internal domain of an obstacle.
% Currently rectangular obstacles are painted cyan
strCol = 'c';
%Truncate the values of the field obtained
t_rec=2; % Limit value

%determines the size of the rows in COORDCORN, i.e. the number of
obstacles
[rC,cC] = size(COORDCORN);

UREP =zeros(numel(y),numel(x));
% creating the initial matrix of the repulsive potential,
% with the size correspinding to  the size of the matrix of points in
% the grid. Initially all values in UREP are zeros. This
% matrix will accumulate the values of the repulsive
% potentials generated by each obstacle, to obtain at the
% final step the repulsive potential of the environment

for i =1:rC
    %In this loop invokes the function 'potrectonstaclenew' for each
```

```
    %obstacle, which places the obstacle in the environment and
computes
    %its repulsive potential
Urep = ProtectObsTruncate(COORDCORN(i,:),W(i,:), H(i,:),
Curv,roinv,romin,lw, strCol,x,y,xx,yy,t_rec);
UREP = UREP+ Urep;
end

% zz_R is a matrix that contains the values of the potential field
% produced by all the rectangular obstacles in the environment.
zz_R = UREP;

%% Circular Obstacles
% In this section circular obstacles are defined and placed in the
% envirnoment, its potential field is calculated.

% COORD_CIR is an array with values of the coordinates of the center
% of obstacles. In each row j, x- and -y coordinates of jth obstacle
are
% stored.
COORD_CIR=[11 6
    11 9
    11 3
    14 7.5
    14 4.5
    14 10.5
    14 1.5];
% K is an array with values of the coefficient of each obstacle.
K=[0.35
    0.35
    0.35
    0.35
    0.35
    0.35];
% Rad is an array with values of the radius of each obstacle.
Rad=[0.5
    0.5
    0.5
    0.5
    0.5
    0.5];
% Points is an array that contains the number of points to make the
cirles.
Points=[100
    100
    100
    100
    100
    100];
% strCol determines the colour of the internal domain of an obstacle.
% Currently circular obstacles are painted blue
strCol = 'b';

%determines the size of the rows in COORD_CIR
[rC_Cir,cC_O] = size(COORD_CIR);

UREP =zeros(numel(y),numel(x));
```

```
% creating the initial matrix of the repulsive potential,
% with the size correspinding to  the size of the matrix of points in
% the grid. Initially all values in UREP are zeros. This
% matrix will accumulate the values of the repulsive
% potentials generated by each obstacle, to obtain at the
% final step the repulsive potential of the environment

for i =1:rC_Cir
    %In this loop invokes the function 'potrectonstaclenew' for each
    %obstacle, which places the obstacle in the environment and
computes
    %its repulsive potential
Urep = ProtectObsCircle(COORD_CIR(i,:),Rad(i,:), K(i,:),Points(i,:),
strCol,x,y);
UREP = UREP+ Urep;
end

% zz_Cir is a matrix that contains the values of the potential field
% produced by all the circular obstacles in the environment.
zz_Cir = UREP;

%% Goals
% In this section goals are defined and placed in the
% envirnoment, its potential field is calculated.

% COORD_GOAL is an array with values of the coordinates of the center
% of obstacles. In each row j, x- and -y coordinates of jth obstacle
are
% stored.
COORD_GOAL=[20 10
    20 6
    20 4];
% K_G is an array with values of the coefficient of each obstacle.
K_G=[0.1
    0.1
    0.1];
% Rad is an array with values of the radius of each obstacle.
Rad=[0.3
    0.3
    0.3];
% Points is an array that contains the number of points to make the
cirles.
Points=[9
    9
    9];

% strCol is an array that contains the colour of the internal domain
of the
% goals. Currently circular obstacles are painted blue
strCol = ['g'
    'r'
    'm'];

%determines the size of the rows in COORD_GOAL
[rC_Go,cC_Go] = size(COORD_GOAL);

UREP =zeros(numel(y),numel(x));
% creating the initial matrix of the repulsive potential,
% with the size correspinding to  the size of the matrix of points in
% the grid. Initially all values in UREP are zeros. This
```

```
% matrix will accumulate the values of the repulsive
% potentials generated by each obstacle, to obtain at the
% final step the repulsive potential of the environment

for i =1:rC_Go
    %In this loop invokes the function 'potrectonstaclenew' for each
    %goal, which places the obstacle in the environment and computes
    %its repulsive potential
Urep = ProtectObsGoals(COORD_GOAL(i,:),Rad(i,:), K(i,:),Points(i,:),
strCol(i,:),x,y);
UREP = UREP+ Urep;
end

% zz_Goal is a matrix that contains the values of the potential field
% produced by all the rectangular obstacles in the environment.
zz_Goal = UREP;

%% Total Potential Field
%Sum of the potential fields calculated before

Z=zz_Cir+zz_R+zz_ENV+zz_Goal;

%% Moving obstacles

%Linear movement
%Obstacle 4 moving
K4m=0.35;
Obs4m=[4 3];

%Draw the obstacle
 MovingObs4=plot(Obs4m(1,1), Obs4m(1,2),'o','Markersize',10,...
     'MarkerFaceColor',[1, 0.5, 0.5],'MarkerEdgeColor',[1, 0.5, 0.5]);

%Variables to contol the movement
dx4=0.5;
dx41=Obs4m;

%Circular movement
%Obstacle 5 moving
K5m=0.55;
Obs5m_ini=[11 6];
Obs5m=[Obs5m_ini(1,1)+1.5 Obs5m_ini(1,2)];

%Draw the obstacle
 MovingObs5=plot(Obs5m_ini(1,1)+1.5,
Obs5m_ini(1,2),'o','Markersize',10,...
     'MarkerFaceColor',[1, 0.5, 0.5],'MarkerEdgeColor',[1, 0.5, 0.5]);

%% Robots and complete environment in initial situation

% Coordinates of the center of each robot in its initial position
%Robot 1 green
Robot1=[1 2];
%Robot 2 red
Robot2=[1 6];
%Robot 3 magenta
Robot3=[1 10];

figure(1)
```

```
%Draw the robots in the environment
plot(Robot1(1),Robot1(2),'ro', 'MarkerSize', 3, 'MarkerFaceColor',
'g','MarkerEdgeColor','g')
plot(Robot2(1),Robot2(2),'ro', 'MarkerSize', 3, 'MarkerFaceColor',
'r')
plot(Robot3(1),Robot3(2),'ro', 'MarkerSize', 3, 'MarkerFaceColor',
'm','MarkerEdgeColor','m')

title('Enviroment')
axis off

%% Path planning 2D

%comput the current distances between:
%Robots and Goals
trg1=(abs(Robot1(1)-COORD_GOAL(1,1))+abs(Robot1(2)-COORD_GOAL(1,2)));
trg2=(abs(Robot2(1)-COORD_GOAL(2,1))+abs(Robot2(2)-COORD_GOAL(2,2)));
trg3=(abs(Robot3(1)-COORD_GOAL(3,1))+abs(Robot3(2)-COORD_GOAL(3,2)));
%between Robots
tr1=(abs(Robot1(1)-Robot2(1))+abs(Robot1(2)-Robot2(2)));
tr2=(abs(Robot1(1)-Robot3(1))+abs(Robot1(2)-Robot3(2)));
tr3=(abs(Robot3(1)-Robot2(1))+abs(Robot3(2)-Robot2(2)));

% count stores the number of iterations
count=0;

%Counter to save te previous position of the robot, this will be use
to
%identify if the robot is 'trapped'
Robot1_ant=[0 0];
Robot2_ant=[0 0];
Robot3_ant=[0 0];
%Counter no know how many iteration has been the robot trapped
Rob1=0;
Rob2=0;
Rob3=0;
%Maximum number of iterations permeted for the robot to be trapped
it=2;
%Maximum movement permited in random
dx_random=0.1;
%Save positions in each iteration to be used in path plannig 3D
Robot1_Path=[];
Robot2_Path=[];
Robot3_Path=[];
MovObs_Path=[];
MovObs5_Path=[];
%Priorities between robots:
% Robot 1 no restrictions
% Robot 2 has priority over Robot 3
% Robot 3 no priority

%check if the robots had reached the specified targets
 while trg1>0.1 || trg2>0.1 || trg3>0.1

     %accumulation matrix
     dx1=[0 0];
     dx2=[0 0];
     dx3=[0 0];

     %Compute change in robots position over time and then update it
```

```
     %Boundarys: Enviroment limits
     %Initialization of the dx produced by the boundaries on Robot
     dx_R1e=[0 0];
     for i=1:rC_env % rC_env is the number of boundaries and always is
4
         % This loop calculates the dx produced by each of the
boundaries
         % and sum all the results.
         dx_R1e=dx_R1e+ForceNewTruncate(COORD_ENV(i,:),W_ENV(i,:),
H_ENV(i,:), Robot1,roinv,romin,t_env);
     end

     %Obstacle: Rectangles
     %Initialization
     dx_R1=[0 0];
     for i=1:rC % rC is the number of rectangular obstacles in the env
         % This loop calculates de dx produced by each rectangular
obstacle
         % and sum all the results
         dx_R1=dx_R1+ForceNewTruncate(COORDCORN(i,:),W(i,:), H(i,:),
Robot1,roinv,romin,t_rec);
     end

     %Obstacle: Circles
     %Initialization
     dx_C1=[0 0];
     for i=1:rC_Cir % rC_Cir is the number of circular obstacles in
the env
         % This loop calculates de dx produced by each circular
obstacle
         % and sum all the results
         dx_C1=dx_C1+ForceNewCircleExp(COORD_CIR(i,:), K(i,:),Robot1);
     end

     %Influence of the goal on the actual position of the robot
     dx_goal1=ForceNewGoalExp(COORD_GOAL(1,:),K_G(1,:),Robot1);

     % Sum of the dx calculated above and the influence of the moving
     % obstacle
     dx1= dx1 +dx_C1+...
         (Robot1-Obs4m)*exp((-norm(Robot1-Obs4m))/K4m)+...
         (Robot1-Obs5m)*exp((-norm(Robot1-Obs5m))/K5m)+...
         dx_goal1+dx_R1e+dx_R1;

     % New position of the robot 1
     Robot1=Robot1+dx1;

     % Check if Robot 1 is trapped comparing the position obtained in
this
     % loop with the previous one stored in Robot1_ant.Due to the
position
     % contains 4 decimals it has been decided to round to two in the
     % following statement
     if round(Robot1(1),2)==round(Robot1_ant(1),2) &&
round(Robot1(2),2)==round(Robot1_ant(2),2)
         Rob1=Rob1+1; %Number of iterations the Robot is trapped
     else
         Rob1=0; %Reset the counter if the robot is not trapped
     end
```

```
     % When the robot reaches the maximum number of iterations trapped
it
     % is moved randomly.
     % The result of '-1+2*rand' is a number between -1 and 1 and it
is
     % aplied in both coordinates and multiplied by 0.1 to reduce its
value
     if Rob1==it
         Robot1=Robot1+[-1+2*rand -1+2*rand]*dx_random;
         Rob1=0;
     end

     %Update the position of Robot 2 only when Robot 1 is out of range
     if tr1>0.7

         %Boundarys: Enviroment limits
         %Initialization of the dx produced by the boundaries on Robot
         dx_R2e=[0 0];
         for i=1:rC_env % rC_env is the number of boundaries and
always is 4
             % This loop calculates the dx produced by each of the
boundaries
             % and sum all the results.
             dx_R2e=dx_R2e+ForceNewTruncate(COORD_ENV(i,:),W_ENV(i,:),
H_ENV(i,:), Robot2,roinv,romin,t_env);
         end

         %Obstacle: Rectangles
         %Initialization
         dx_R2=[0 0];
         for i=1:rC % rC is the number of rectangular obstacles in the
env
             % This loop calculates de dx produced by each rectangular
obstacle
             % and sum all the results
             dx_R2=dx_R2+ForceNewTruncate(COORDCORN(i,:),W(i,:),
H(i,:), Robot2,roinv,romin,t_rec);
         end

         %Obstacle: Circles
         %Initialization
         dx_C2=[0 0];
         for i=1:rC_Cir % rC_Cir is the number of circular obstacles
in the env
             % This loop calculates de dx produced by each circular
obstacle
             % and sum all the results
             dx_C2=dx_C2+ForceNewCircleExp(COORD_CIR(i,:),
K(i,:),Robot2);
         end

         %Influence of the goal on the actual position of the robot
         dx_goal2=ForceNewGoalExp(COORD_GOAL(2,:),K_G(2,:),Robot2);

         % Sum of the dx calculated above and the influence of the
moving
         % obstacle
         dx2= dx2 +dx_C2+...
             (Robot2-Obs4m)*exp((-norm(Robot2-Obs4m))/K4m)+...
             (Robot2-Obs5m)*exp((-norm(Robot2-Obs5m))/K5m)+...
```

```
            dx_goal2+dx_R2e+dx_R2;

        % New position of Robot 2
        Robot2=Robot2+dx2;

        % Check if Robot 2 is trapped comparing the position obtained
in this
        % loop with the previous one stored in Robot1_ant.Due to the
position
        % contains 4 decimals it has been decided to round to two in
the
        % following statement
        if round(Robot2(1),2)==round(Robot2_ant(1),2) &&
round(Robot2(2),2)==round(Robot2_ant(2),2)
            Rob2=Rob2+1; %Number of iterations the Robot is trapped
        else
            Rob2=0; %Reset the counter if the robot is not trapped
        end

        % When the robot reaches the maximum number of iterations
trapped it
        % is moved randomly.
        % The result of '-1+2*rand' is a number between -1 and 1 and
it is
        % aplied in both coordinates and multiplied by 0.1 to reduce
its value
        if Rob2==it
            Robot2=Robot2+[-1+2*rand -1+2*rand]*dx_random;
            Rob2=0;
        end
    end

    %Update the position of Robot 3 only when Robot 1 and Robot 2 are
out of range
    if tr2>0.7 && tr3>0.7

        %Boundarys: Enviroment limits
        %Initialization of the dx produced by the boundaries on Robot
        dx_R3e=[0 0];
        for i=1:rC_env % rC_env is the number of boundaries and
always is 4
            % This loop calculates the dx produced by each of the
boundaries
            % and sum all the results.
            dx_R3e=dx_R3e+ForceNewTruncate(COORD_ENV(i,:),W_ENV(i,:),
H_ENV(i,:), Robot3,roinv,romin,t_env);
        end

        %Obstacle: Rectangles
        %Initialization
        dx_R3=[0 0];
        for i=1:rC % rC is the number of rectangular obstacles in the
env
            % This loop calculates de dx produced by each rectangular
obstacle
            % and sum all the results
            dx_R3=dx_R3+ForceNewTruncate(COORDCORN(i,:),W(i,:),
H(i,:), Robot3,roinv,romin,t_rec);
        end
```

```
        %Obstacle: Circles
        %Initialization
        dx_C3=[0 0];
        for i=1:rC_Cir % rC_Cir is the number of circular obstacles
in the env
            % This loop calculates de dx produced by each circular
obstacle
            % and sum all the results
            dx_C3=dx_C3+ForceNewCircleExp(COORD_CIR(i,:),
K(i,:),Robot3);
        end

        %Influence of the goal on the actual position of the robot
        dx_goal3=ForceNewGoalExp(COORD_GOAL(3,:),K_G(3,:),Robot3);

        % Sum of the dx calculated above and the influence of the
moving
        % obstacles
        dx3= dx3 +dx_C3+...
            (Robot3-Obs4m)*exp((-norm(Robot3-Obs4m))/K4m)+...
            (Robot3-Obs5m)*exp((-norm(Robot3-Obs5m))/K5m)+...
            dx_goal3+dx_R3e+dx_R3;

        % New position of Robot
        Robot3=Robot3+dx3;

        % Check if Robot 2 is trapped comparing the position obtained
in this
        % loop with the previous one stored in Robot1_ant.Due to the
position
        % contains 4 decimals it has been decided to round to two in
the
        % following statement
        if round(Robot3(1),2)==round(Robot3_ant(1),2) &&
round(Robot3(2),2)==round(Robot3_ant(2),2)
            Rob3=Rob3+1; %Number of iterations the Robot is trapped
        else
            Rob3=0; %Reset the counter if the robot is not trapped
        end
        % When the robot reaches the maximum number of iterations
trapped it
        % is moved randomly.
        % The result of '-1+2*rand' is a number between -1 and 1 and
it is
        % aplied in both coordinates and multiplied by 0.1 to reduce
its value
        if Rob3==it
            Robot3=Robot3+[-1+2*rand -1+2*rand]*dx_random;
            Rob3=0;
        end

    end

    %Save positions of the robots
    Robot1_ant=Robot1;
    Robot2_ant=Robot2;
    Robot3_ant=Robot3;

    %Moving obsacle
    %Linear movement
```

```
      if dx4==1
        if dx41(1)<4    %(1) L-r    %(2) u-d    %lowest it goes to
             dx4=2;
        else
             dx41(1) = dx41(1)-0.5;
        end
      else
         if dx41(1)>7    %Biggest it goes to
             dx4=1;
         else
            dx41(1)=dx41(1)+0.5;
         end
      end

      %Update position of the obstacle
      Obs4m=dx41;
      %Save the position
      MovObs_Path=[MovObs_Path;Obs4m];

      %Circular movement
      Obs5m=[cos(count*pi/25) sin(count*pi/25)]*1.5+Obs5m_ini;

      %Save the position
      MovObs5_Path=[MovObs5_Path;Obs5m];

      %Recalculate distances
      %Robots and Goals
      trg1=(abs(Robot1(1)-COORD_GOAL(1,1))+abs(Robot1(2)-
COORD_GOAL(1,2)));
      trg2=(abs(Robot2(1)-COORD_GOAL(2,1))+abs(Robot2(2)-
COORD_GOAL(2,2)));
      trg3=(abs(Robot3(1)-COORD_GOAL(3,1))+abs(Robot3(2)-
COORD_GOAL(3,2)));
      %Between Robots
      tr1=(abs(Robot1(1)-Robot2(1))+abs(Robot1(2)-Robot2(2)));
      tr2=(abs(Robot1(1)-Robot3(1))+abs(Robot1(2)-Robot3(2)));
      tr3=(abs(Robot3(1)-Robot2(1))+abs(Robot3(2)-Robot2(2)));

      %update display
      count=count+1;

      %Plots and save position in each iteration
      if trg1>0.1
          plot(Robot1(1,1), Robot1(1,2), 'G.','MarkerSize',20);
          Robot1_Path=[Robot1_Path;Robot1];
      end
      if trg2>0.1
          plot(Robot2(1,1), Robot2(1,2), 'R.','MarkerSize',20);
          Robot2_Path=[Robot2_Path;Robot2];
      end
      if trg3>0.1
          plot(Robot3(1,1), Robot3(1,2), 'M.','MarkerSize',20);
          Robot3_Path=[Robot3_Path;Robot3];
      end

      %Plot new position of the moving obstacle
      MovingObs42= plot(Obs4m(1), Obs4m(2), 'o',...
        'Markersize', 10, 'MarkerFaceColor',[1, 0.5,
0.5],'MarkerEdgeColor',[1, 0.5, 0.5]);
      MovingObs52= plot(Obs5m(1), Obs5m(2), 'o',...
```

```
       'Markersize', 10, 'MarkerFaceColor',[1, 0.5,
0.5],'MarkerEdgeColor',[1, 0.5, 0.5]);
    %Delete the previous plot
    delete(MovingObs4);
    delete(MovingObs5);
    MovingObs4=MovingObs42;
    MovingObs5=MovingObs52;

     title(sprintf('Iteration: %d',count));
     refresh;
     drawnow;
     pause(0.02)
 end

 %% Plots
%Contour figure
figure(2)
title('Contour')
hold on
contour(Z,100)
pause(1)
hold on;

%combine figures
figure(3)
title('Potential Force Field on Contour')
hold on
contour(Z,100)
hold on
[px,py]=gradient(Z); %calculate gradient
quiver(xx*10,yy*10,-px*10,-py*10,'r'), hold on % plot velocity vectors
pause(1)

%3D plot
figure(4)
title('Potential Function Landscape')
hold on; grid on
view([100,100,80])
%Calculate the potential field of the moving obstacle in initial
position
Z_mov=ProtectObsMovingCircle(Obs4m,
K4m,x,y)+ProtectObsMovingCircle(Obs5m, K5m,x,y);
% Plot the potential fields obtained before and sum the new one
Moving=surfc(Z+Z_mov);

pause(1)

%% Path planning 3D
% In this secction is done the 3D plot of the path planning realized
before
% usig all the data stored.

%Sizes of the path of each robot
[path1,n] = size(Robot1_Path);
[path2,n] = size(Robot2_Path);
[path3,n] = size(Robot3_Path);

%The next loop will calculate and update the potential field produced
by the
```

```
%moving obstacle on the surface figure of the environment. Also is
made the
%plot of the path of each robot on it.
for i=1:count

    %Calculate the potential field of the moving obstacle in this
iteration
    Z_mov=ProtectObsMovingCircle(MovObs_Path(i,:),
K4m,x,y)+ProtectObsMovingCircle(MovObs5_Path(i,:), K5m,x,y);
    %Delete the previous one
    delete(Moving);
    %Print the surface with the updated potential field of the moving
    %obstacle
    Moving=surfc(Z+Z_mov);

    % Prints the position of the robots if they have not reached their
    % goals.
    if i<path1
        plot3(Robot1_Path(i,1)*10, Robot1_Path(i,2)*10,
Z(round(Robot1_Path(i,2)*10),
round(Robot1_Path(i,1)*10)),'G.','MarkerSize',20);
    end

    if i<path2
        plot3(Robot2_Path(i,1)*10, Robot2_Path(i,2)*10,
Z(round(Robot2_Path(i,2)*10),
round(Robot2_Path(i,1)*10)),'R.','MarkerSize',20);
    end

    if i<path3
        plot3(Robot3_Path(i,1)*10, Robot3_Path(i,2)*10,
Z(round(Robot3_Path(i,2)*10),
round(Robot3_Path(i,1)*10)),'M.','MarkerSize',20);
    end

    title(sprintf('Iteration: %d',i));
    refresh;
    drawnow;
    %pause(0.1)

end
```

2. Matlab Functions

2.1. Force Circular Obstacles

```
function [Frep] = ForceNewCircleExp(Coordcorn, K,Robot)

Frep=(Robot-Coordcorn)*exp((-norm(Robot-Coordcorn))/K);

End
```

2.2. Force Goals

```
function [Frep] = ForceNewGoalExp(Coordcorn, K,Robot)

Frep=K*(Coordcorn-Robot)/norm(Coordcorn-Robot);

end
```

2.3. Force Rectangular Obstacles

```
function [Frep_R] = ForceNewTruncate(Coordcorn,w, h,
Robot,roinv,romin,truncate)

% this function computes and visualises the potential function for a
% rectangular obstacle
% Parameters:
% 'Coordcorn' - an array of two numbers, Coordcorn(1) - x-coordinate
of the
% left corner of the rectangle, Coordcorn(2) - y-coordinate of the
% left corner of the rectangle.
% w  - width of the rectangle,
% We also assume that the environment where the obstacle is placed has
a rectangular form.
% and that the boundaries (straight line segments) of the
% obstacle are parallel to the boundaries of the environment.
% Env is aan array with four elements, the first two are the x- and y-
coordinates of
% the left corner of the environment.
%In this code we assume that Env(1) =0, and Env(2) =0.
% Env(3) and Env(4) - width and height of the environment

%In this code the obstacle influences only a domain around it within
the
% distances between 'roinv' and 'romin'. When the distance becomes
equal or
% smaller than 'romin', the repulsive potential becomes constant and
equal
% to (1/2)*(1/romin - 1/roinv)^2;
% [x,y] and [xx, yy} are determined in the main function, x  - an
array of
% the x-coordinates of the grid points along the OX axis,
% y - an array of
% the y-coordinates of the grid points along the OY axis,
% [xx,yy] - are the resulting arrays of 'meshgrid(x,y)
%char strCol %strCol is a character
%rectangle ('Position',[Coordcorn(1),Coordcorn(2),w,h],'Curvature',
[Curv(1),Curv(2)], 'LineWidth', lw, 'FaceColor', strCol), hold on
% the statement above draws a rectangular obstacle coloured with
green
%x1 = 5; y1 =3; x2=5; y2 =5; x3 = 7; y3 = 5; x4 =7; y4 =3; % these
values are x-coordinates and y-coordinates of corner vertices of the
obstacle
x1 = Coordcorn(1); y1 = Coordcorn(2); x2 = Coordcorn(1); y2 =
Coordcorn(2)+h;
x3 = Coordcorn(1)+w; y3 = Coordcorn(2)+h; x4 = Coordcorn(1)+w; y4 =
Coordcorn(2);

line1x =Coordcorn(1);
line2x =Coordcorn(1)+w;
line1y = Coordcorn(2);
line2y = Coordcorn(2)+h;
% four expressions above provide the equations  of the bounding lines
of the obstacle

%Check if the robot is under line1y
        if Robot(2)< y1
            %Check if the robot is between lines x1 and x2
            if ((Robot(1) >= x1) && ((Robot(1)) <= x4))
                dist1 = sqrt((Robot(2)-line1y)^2);
```

```
            %Check de distance and the consequent influence
            if (dist1 <= roinv)
                if dist1 >=romin
                    Frep = (1/2)*(1/dist1 - 1/roinv)^2;
                else
                    Frep = (1/2)*(1/romin - 1/roinv)^2;
                end
            else
                Frep=0;
            end
        %Check if the robot is left of line 1x
        elseif  Robot(1) < x1
            dist2 = sqrt((Robot(1)-x1)^2 + (Robot(2)-y1)^2);
            if (dist2 <= roinv)
                if dist2 >=romin
                    Frep = (1/2)*(1/dist2 - 1/roinv)^2;
                else
                    Frep = (1/2)*(1/romin - 1/roinv)^2;
                end
            else
                Frep =0;
            end
        elseif Robot(1) >= x4
            dist3 = sqrt((Robot(1)-x4)^2 + (Robot(2)-y4)^2);
            if (dist3 <= roinv)
                if dist3 >=romin
                    Frep = (1/2)*(1/dist3 - 1/roinv)^2;
                else
                    Frep = (1/2)*(1/romin - 1/roinv)^2;
                end
            else
                Frep =0;
            end
        end
    elseif Robot(2) > y3
        if (Robot(1) >= x2) && (Robot(1) <= x3)
            dist4 = sqrt((Robot(2)-line2y)^2);
            if (dist4 <= roinv)
                if dist4 >=romin
                    Frep = (1/2)*(1/dist4 - 1/roinv)^2;
                else
                    Frep = (1/2)*(1/romin - 1/roinv)^2;
                end
            else
                Frep =0;
            end
        elseif  Robot(1) < x2
            dist5 = sqrt((Robot(1)-x2)^2 + (Robot(2)-y2)^2);
            if (dist5 <= roinv)
                if dist5 >= romin
                    Frep = (1/2)*(1/dist5 - 1/roinv)^2;
                else
                    Frep = (1/2)*(1/romin - 1/roinv)^2;
                end
            else
                Frep =0;
            end
        elseif Robot(1) >= x3
            dist6 = sqrt((Robot(1)-x3)^2 + (Robot(2)-y3)^2);
            if (dist6 <= roinv)
                if dist6 >=romin
```

```
                        Frep = (1/2)*(1/dist6 - 1/roinv)^2;
                    else
                        Frep =(1/2)*(1/romin - 1/roinv)^2;
                    end
                else
                    Frep=0;
                end
            end
        else
            if Robot(1)< x2
                dist7 = sqrt((Robot(1)-line1x)^2);
                if dist7 <= roinv
                    if dist7 >=romin
                        Frep = (1/2)*(1/dist7 - 1/roinv)^2;
                    else
                        Frep =(1/2)*(1/romin - 1/roinv)^2;
                    end
                else
                    Frep = 0;
                end
            elseif Robot(1) > x3
                dist8 = sqrt((Robot(1)-line2x)^2);
                if dist8 <= roinv
                    if dist8 >=romin
                        Frep = (1/2)*(1/dist8 - 1/roinv)^2;
                    else
                        Frep =(1/2)*(1/romin - 1/roinv)^2;
                    end
                else
                    Frep = 0;
                end
            end
        end

        if ((Robot(1) >= x1) && (Robot(1) <= x4)&&(Robot(2)>=y1)
&&(Robot(2)<=y2))
            Frep =(1/2)*(1/romin - 1/roinv)^2;
        end

        %Truncate the results
        if Frep>truncate
            Frep=truncate;
        end

  Center=[(Coordcorn(1)+w/2) (Coordcorn(2)+h/2)];
  Frep_R = (Robot-Center)*Frep;

end
```

2.4. Potential Field Circular Obstacles

```
function [Urep] = ProtectObsCircle(Coordcorn,R, K,points, strCol,x,y)

%Print the obstacle
char strCol; %strCol is a character

theta = linspace(0, 2*pi,points);
x1 = Coordcorn(1) + R*sin(theta);
y1 = Coordcorn(2) + R*cos(theta);
```

```
plot(Coordcorn(1),Coordcorn(2), 'o', 'MarkerFaceColor', strCol,
'Markersize', 6), hold on
plot(x1,y1, strCol);
fill(x1,y1,strCol);
axis equal;

Urep =[];

for i = 1:numel(y) % i refers ro the rows in Urep, which corresponds
to values along OY-axis
    for j = 1:numel(x) %refers to the columns in Urep, which
corresponds to values along OX-axis
        Urep(i,j)=exp(-norm([j/10 i/10]-Coordcorn)/K);
    end
end

end
```

2.5. Potential Field Goals

```
function [Uatt] = ProtectObsGoals(Coordcorn,R, K,points, strCol,x,y)

%Print the obstacle
char strCol; %strCol is a character

theta = linspace(0, 2*pi,points);
x1 = Coordcorn(1) + R*sin(theta);
y1 = Coordcorn(2) + R*cos(theta);
plot(Coordcorn(1),Coordcorn(2), 'o', 'MarkerFaceColor', strCol,
'Markersize', 6), hold on
plot(x1,y1, strCol);
fill(x1,y1,strCol);
axis equal;

Uatt =[];

for i = 1:numel(y) % i refers ro the rows in Uatt, which corresponds
to values along OY-axis
    for j = 1:numel(x) %refers to the columns in Uatt, which
corresponds to values along OX-axis
        Uatt(i,j)=K*(norm(Coordcorn-[j/10 i/10]));

    end
end

end
```

2.6. Potential Field Circular Dynamic Obstacles

```
function [Urep] = ProtectObsMovingCircle(Coordcorn, K,x,y)


Urep =[];
```

```
for i = 1:numel(y) % i refers ro the rows in Urep, which corresponds
to values along OY-axis
    for j = 1:numel(x) %refers to the columns in Urep, which
corresponds to values along OX-axis
        Urep(i,j)=exp(-norm([j/10 i/10]-Coordcorn)/K);
    end
end

end
```

2.7. Potential Field Rectangular Obstacles

```
function [Urep] = ProtectObsTruncate(Coordcorn,w, h, Curv,roinv,romin,
lw, strCol,x,y,xx,yy,truncate)


char strCol; %strCol is a character
rectangle ('Position',[Coordcorn(1),Coordcorn(2),w,h],'Curvature',
[Curv(1),Curv(2)], 'LineWidth', lw, 'FaceColor', strCol), hold on
x1 = Coordcorn(1); y1 = Coordcorn(2); x2 = Coordcorn(1); y2 =
Coordcorn(2)+h;
x3 = Coordcorn(1)+w; y3 = Coordcorn(2)+h; x4 = Coordcorn(1)+w; y4 =
Coordcorn(2);

line1x =Coordcorn(1);
line2x =Coordcorn(1)+w;
line1y = Coordcorn(2);
line2y = Coordcorn(2)+h;

Urep =[];


for i = 1:numel(y) % i refers ro the rows in Urep, which corresponds
to values along OY-axis
    for j = 1:numel(x) %refers to the columns in Urep, which
corresponds to values along OX-axis
        if yy(i,j) < y1
          if ((xx(i,j) >= x1) && (xx(i,j) <= x4))
            dist1 = sqrt((yy(i,j)-line1y)^2);
        if (dist1 <= roinv)
       if dist1 >=romin
           Urep(i,j) = (1/2)*(1/dist1 - 1/roinv)^2;
       else
         Urep(i,j) = (1/2)*(1/romin - 1/roinv)^2;
       end
   else
       Urep(i,j) =0;
   end
          elseif  xx(i,j) < x1

              dist2 = sqrt((xx(i,j)-x1)^2 + (yy(i,j)-y1)^2);

   if (dist2 <= roinv)
       if dist2 >=romin
   Urep(i,j) = (1/2)*(1/dist2 - 1/roinv)^2;
       else  Urep(i,j) = (1/2)*(1/romin - 1/roinv)^2;
       end
```

```
else
    Urep(i,j) =0;
end
       elseif xx(i,j) >= x4
           dist3 = sqrt((xx(i,j)-x4)^2 + (yy(i,j)-y4)^2);
           if (dist3 <= roinv)
               if dist3 >=romin
Urep(i,j) = (1/2)*(1/dist3 - 1/roinv)^2;
               else Urep(i,j) = (1/2)*(1/romin - 1/roinv)^2;
               end
else
    Urep(i,j) =0;
           end
       end
   elseif yy(i,j) > y3
 if (xx(i,j) >= x2) && (xx(i,j) <= x3)
   dist4 = sqrt((yy(i,j)-line2y)^2);
if (dist4 <= roinv)
    if dist4 >=romin
Urep(i,j) = (1/2)*(1/dist4 - 1/roinv)^2;
    else Urep(i,j) = (1/2)*(1/romin - 1/roinv)^2;
    end
else
    Urep(i,j) =0;
end
       elseif  xx(i,j) < x2

           dist5 = sqrt((xx(i,j)-x2)^2 + (yy(i,j)-y2)^2);
if (dist5 <= roinv)
     if dist5 >= romin
Urep(i,j) = (1/2)*(1/dist5 - 1/roinv)^2;
     else Urep(i,j) = (1/2)*(1/romin - 1/roinv)^2;
     end
else
    Urep(i,j) =0;
end
       elseif xx(i,j) >= x3
           dist6 = sqrt((xx(i,j)-x3)^2 + (yy(i,j)-y3)^2);
if (dist6 <= roinv)
    if dist6 >=romin
Urep(i,j) = (1/2)*(1/dist6 - 1/roinv)^2;
    else Urep(i,j) =(1/2)*(1/romin - 1/roinv)^2;
    end
else
    Urep(i,j)=0;
end
 end
else
   if xx(i,j)< x2
       dist7 = sqrt((xx(i,j)-line1x)^2);
       if dist7 <= roinv
            if dist7 >=romin
            Urep(i,j) = (1/2)*(1/dist7 - 1/roinv)^2;
            else Urep(i,j) =(1/2)*(1/romin - 1/roinv)^2;
            end
       else Urep(i,j) = 0;
       end
   elseif xx(i,j) > x3
       dist8 = sqrt((xx(i,j)-line2x)^2);
       if dist8 <= roinv
            if dist8 >=romin
```

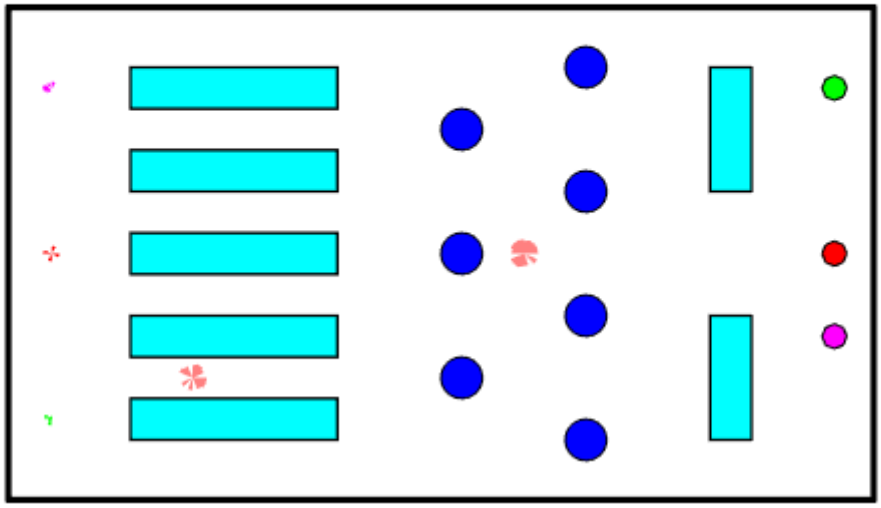
```
                Urep(i,j) = (1/2)*(1/dist8 - 1/roinv)^2;
                else Urep(i,j) =(1/2)*(1/romin - 1/roinv)^2;
                end
            else Urep(i,j) = 0;
            end
        end
    end
    if ((xx(i,j) >= x1) && (xx(i,j) <= x4)&&(yy(i,j)>=y1)
&&(yy(i,j)<=y2))
        Urep(i,j) =(1/2)*(1/romin - 1/roinv)^2;
    end

    %Truncate
    if Urep(i,j)>truncate
        Urep(i,j)=truncate;
    end
      end
end
end
```

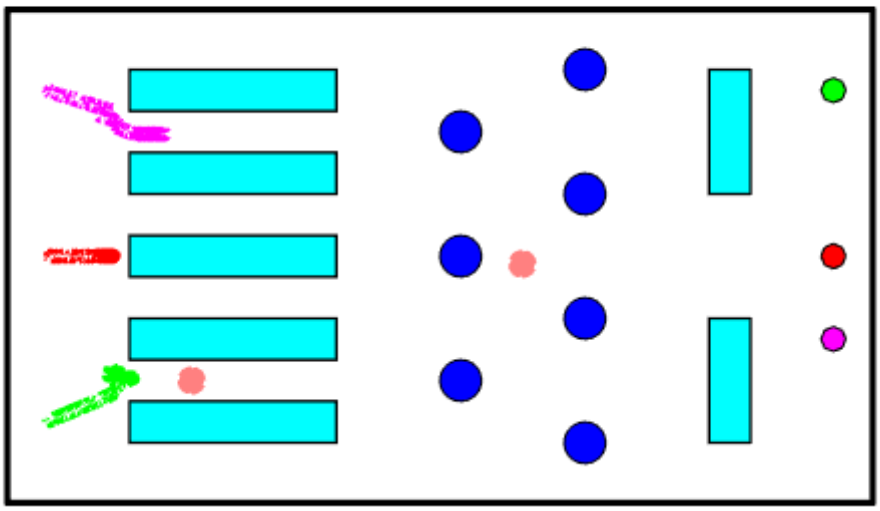
APPENDIX B: Simulation Snap-shots

1. 2D Path Planning

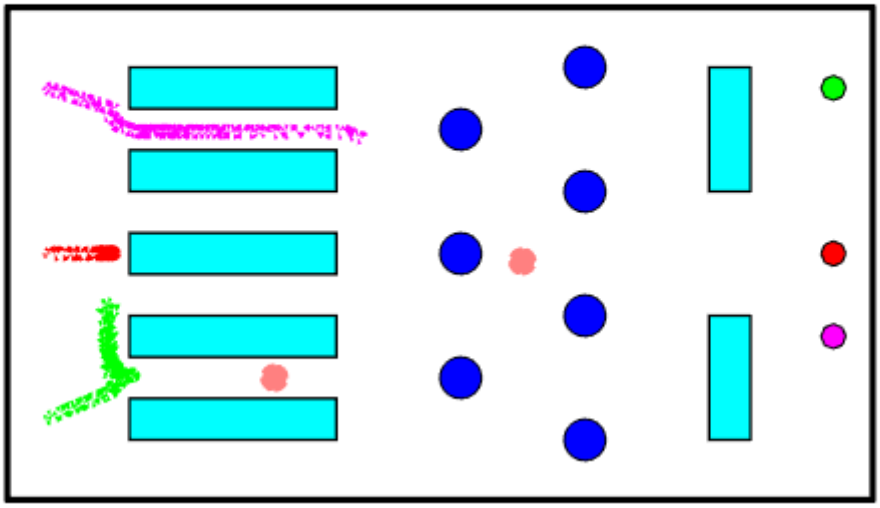
Iteration: 1



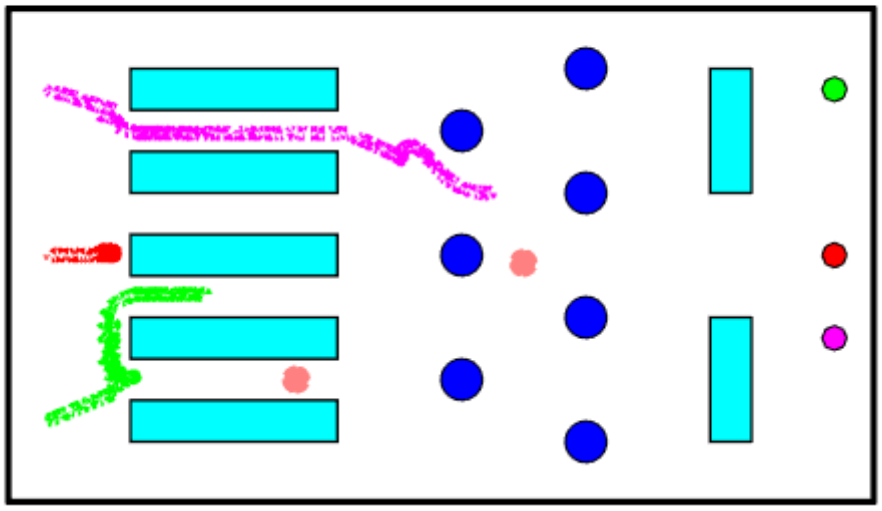
Iteration: 50



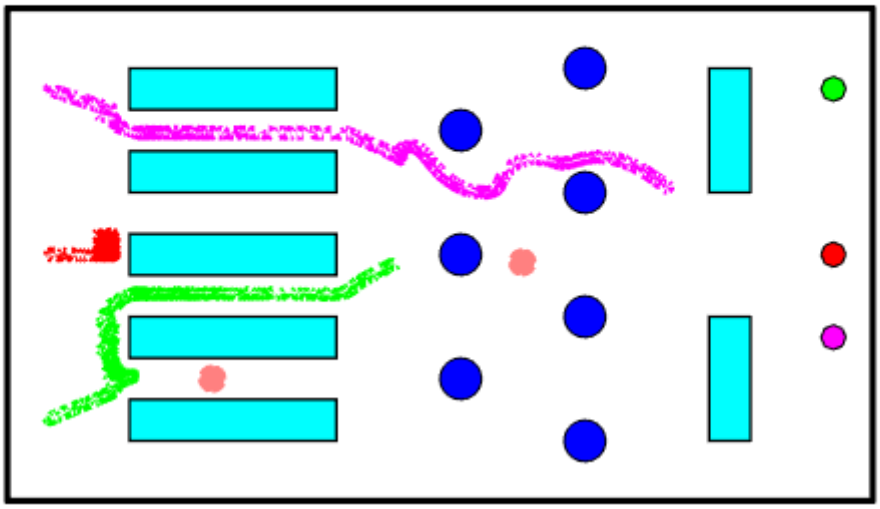
Iteration: 100



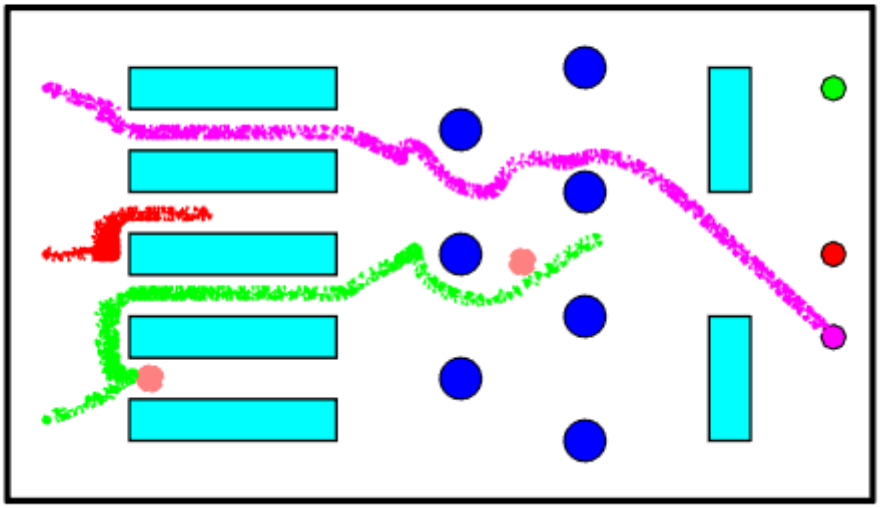
Iteration: 150



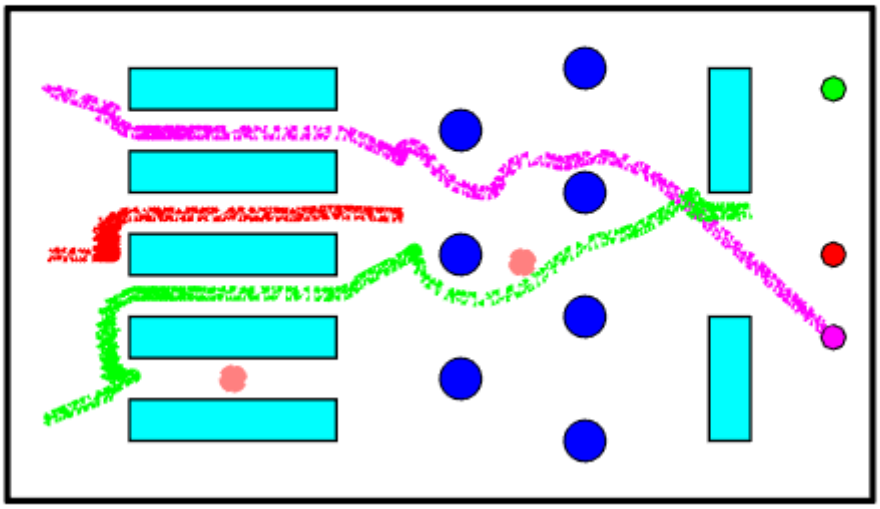
Iteration: 200



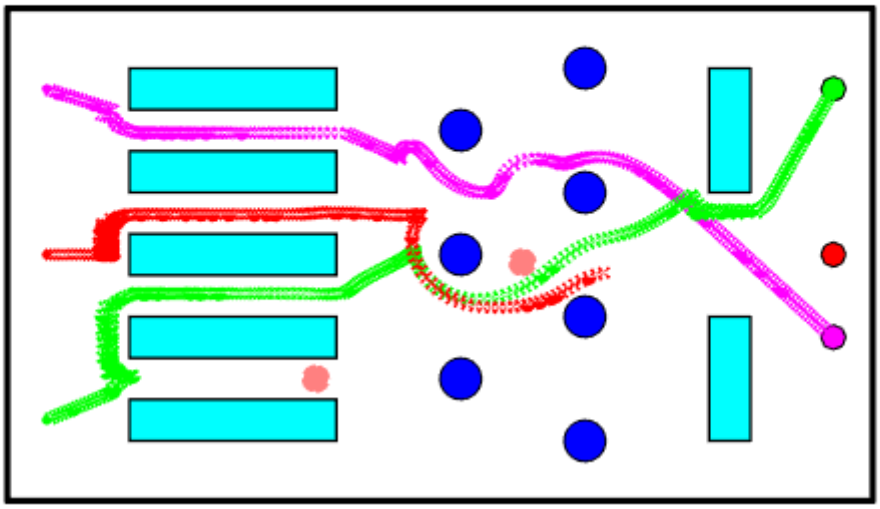
Iteration: 250

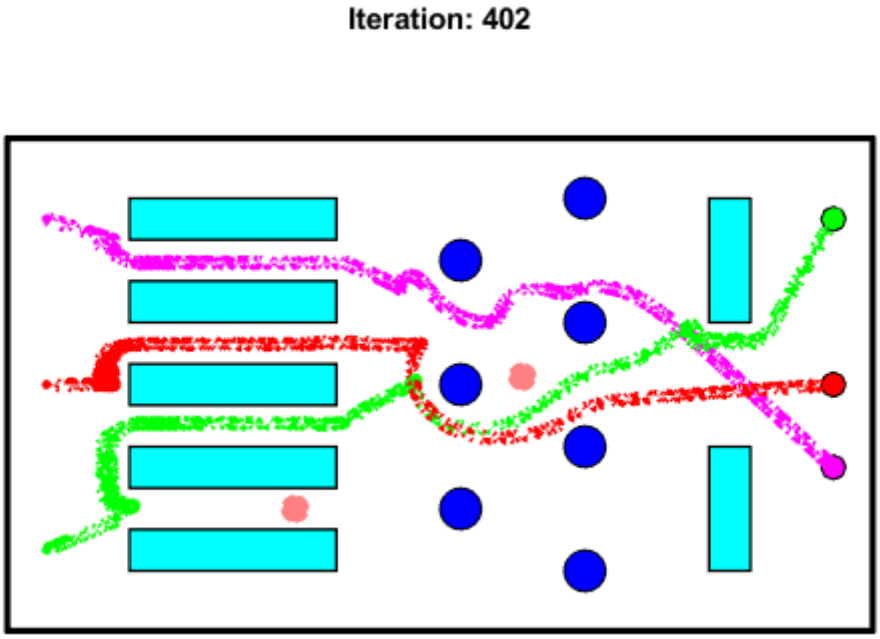


Iteration: 300

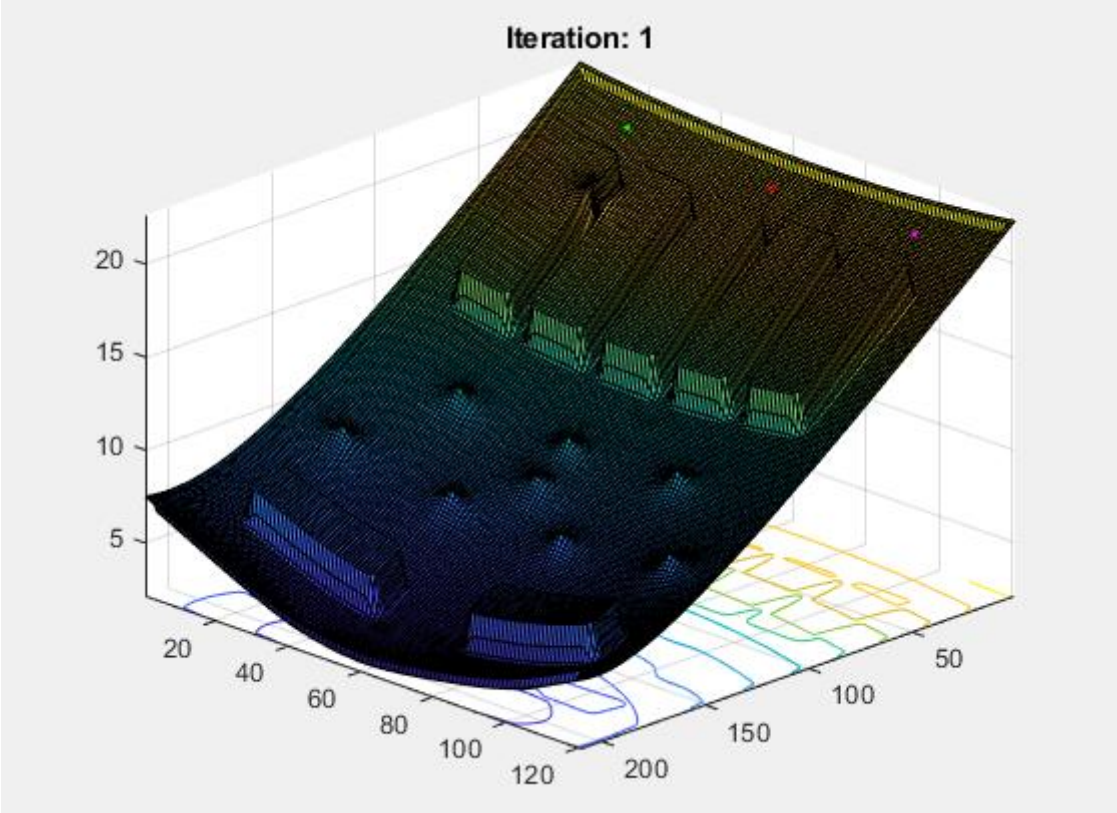


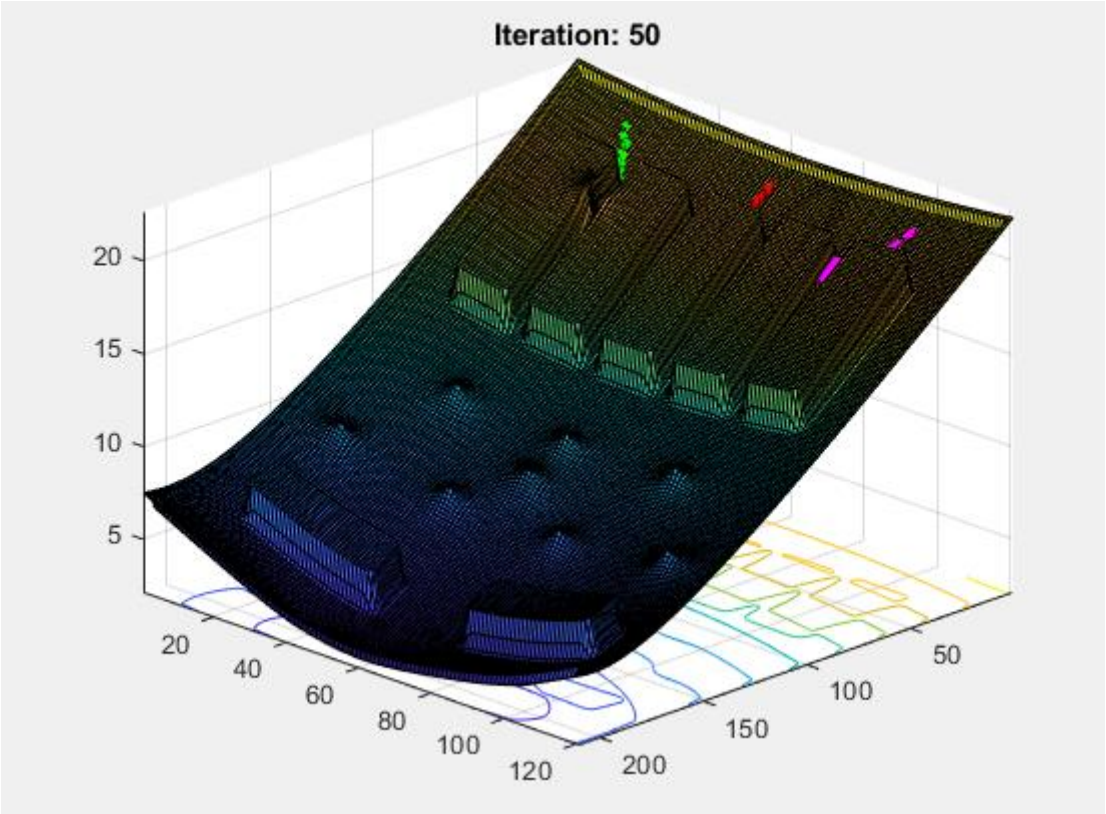
Iteration: 350

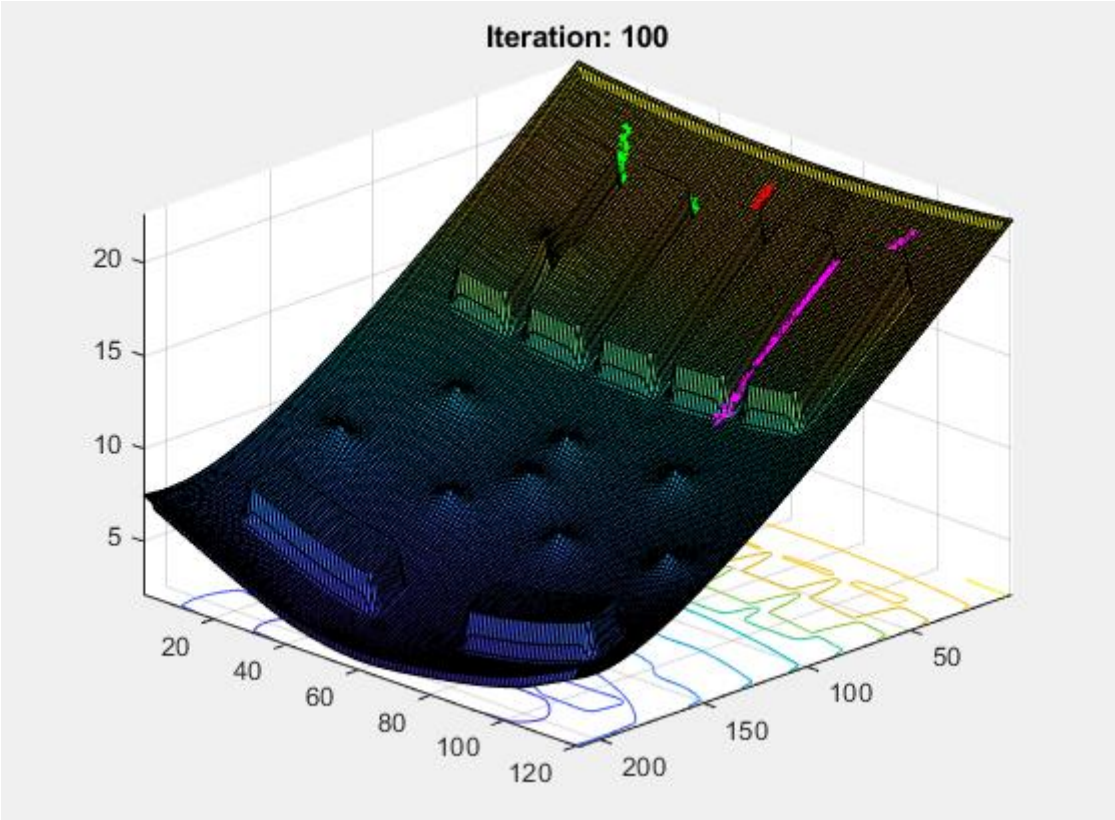


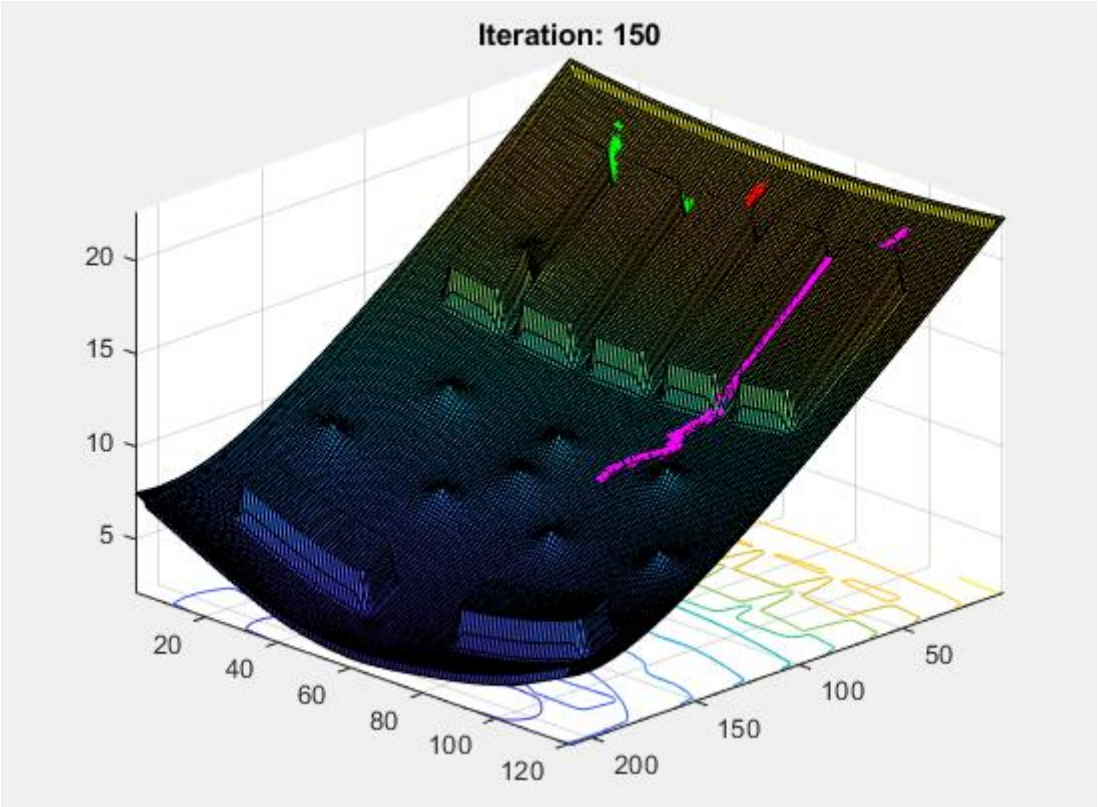


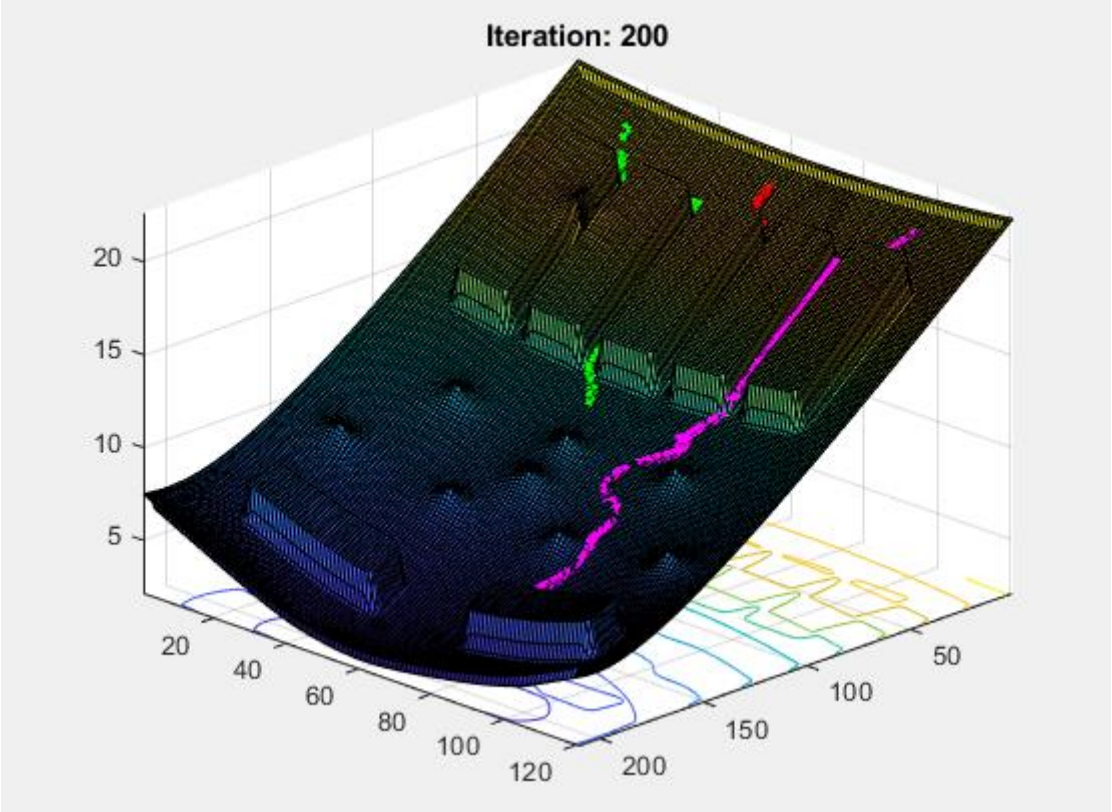
2. 3D Path Planning

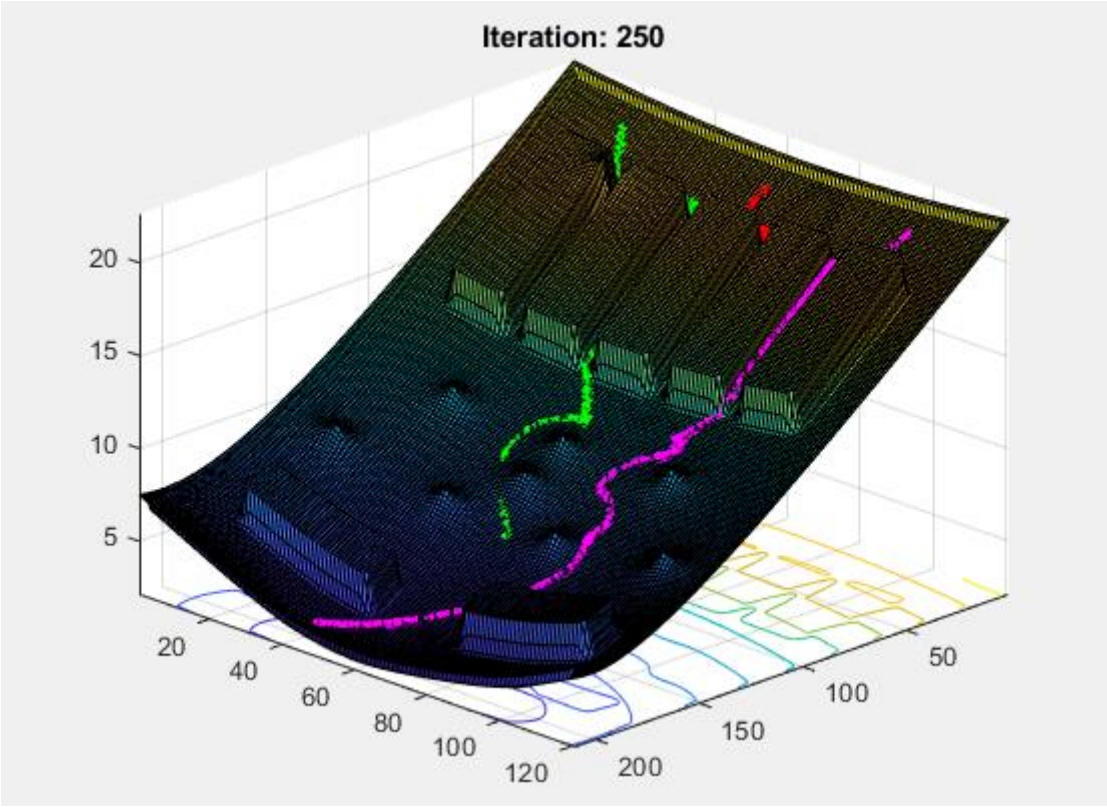


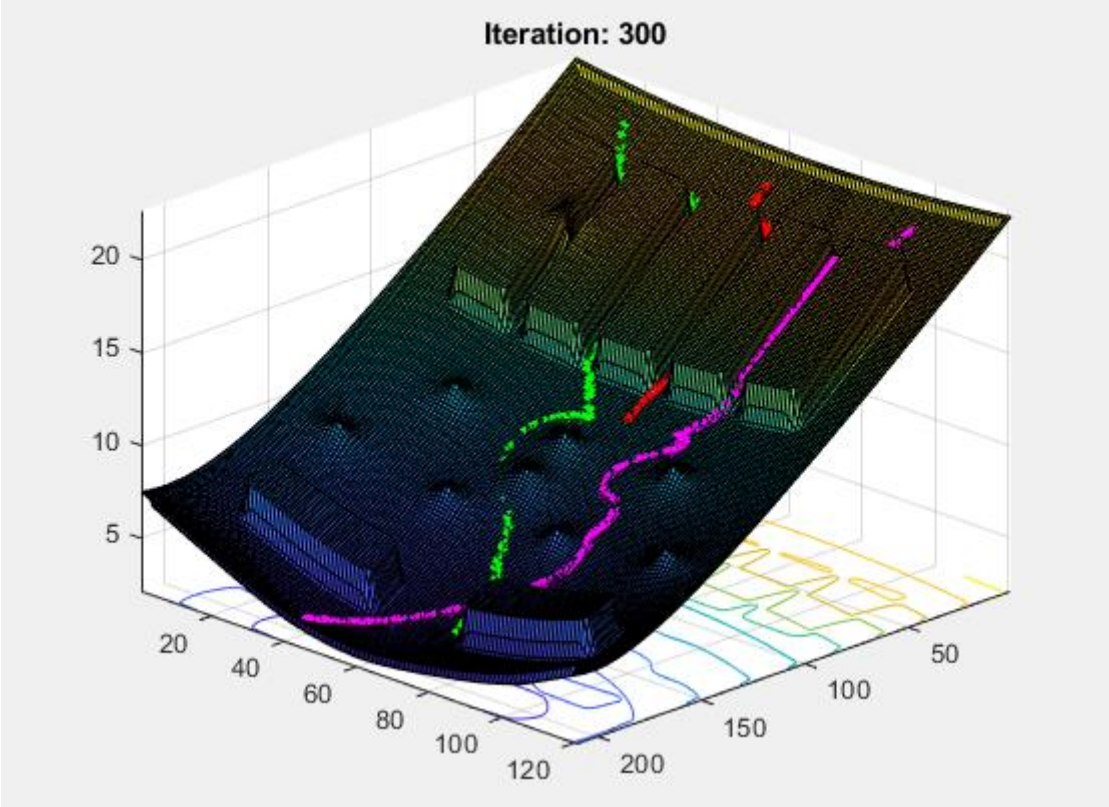

Iteration: 50
20
15
10
5
20
40
60
80
100
120
200
150
100
50

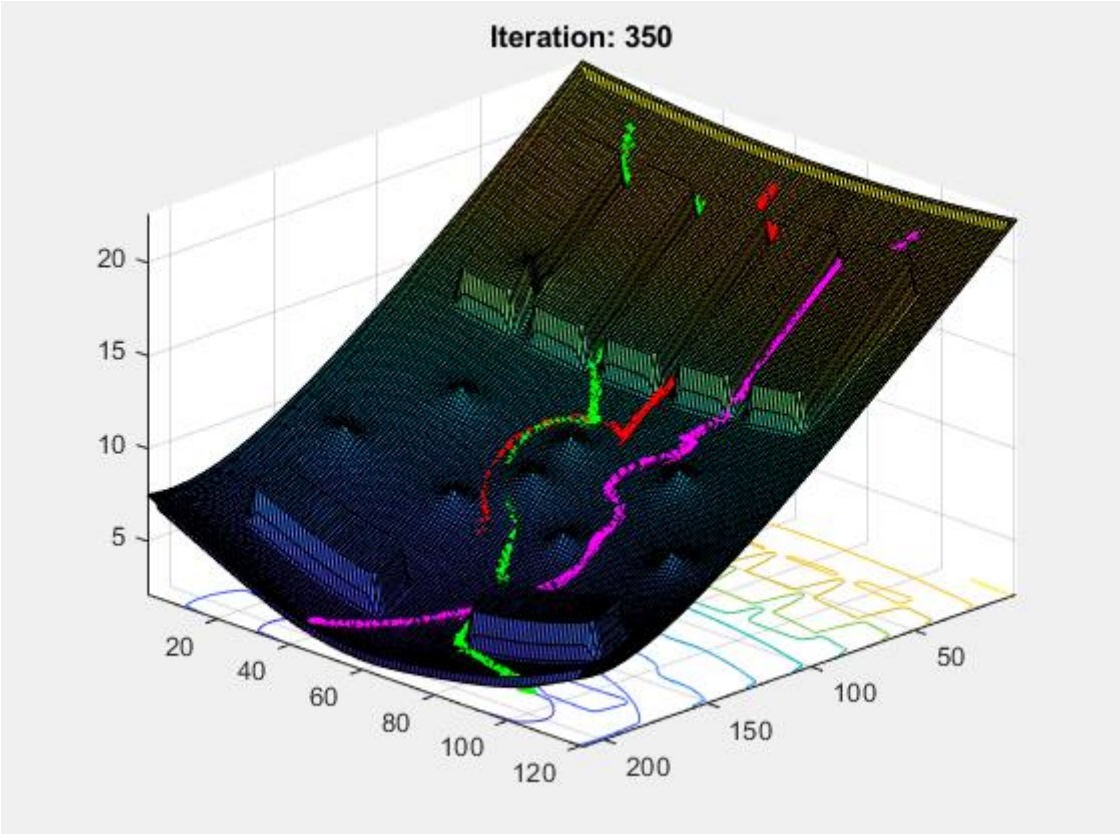

Iteration: 100
20
15
10
5
20
40
60
80
100
120
200
150
100
50

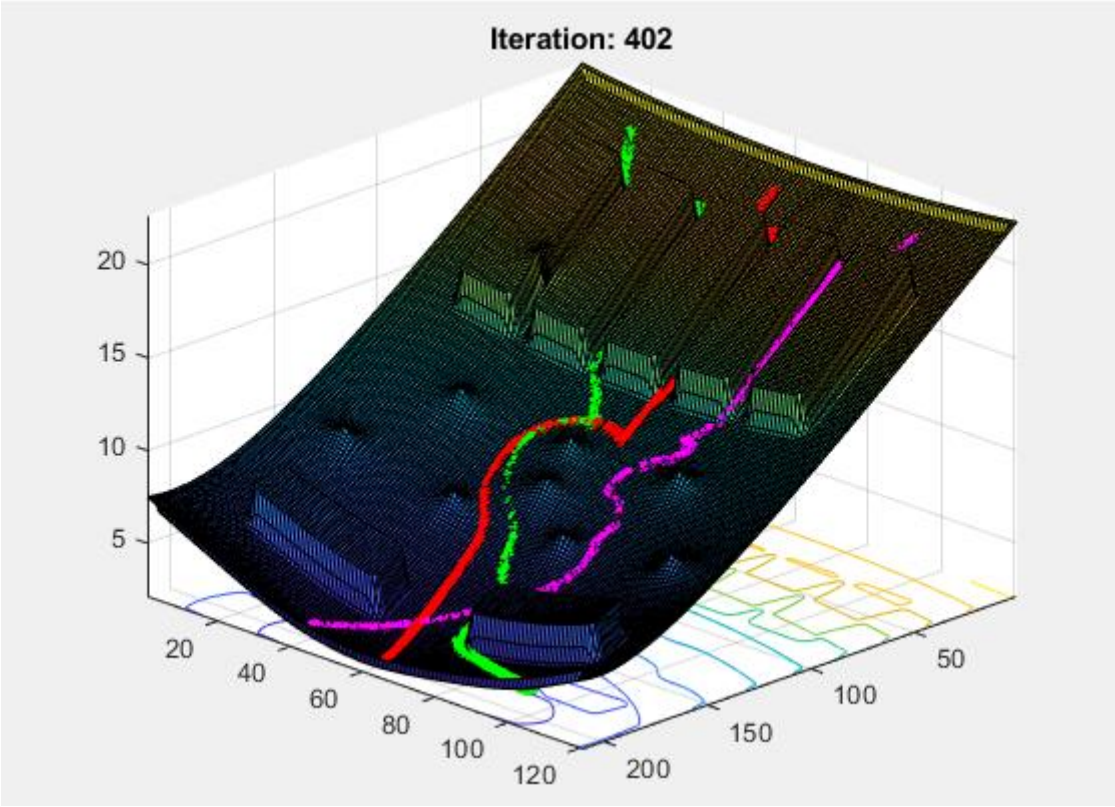

Iteration: 150
20
15
10
5
20
40
60
80
100
120
200
150
100
50


Iteration: 200
20
15
10
5
20
40
60
80
100
120
200
150
100
50


Iteration: 250
20
15
10
5
20
40
60
80
100
120
200
150
100
50


Iteration: 300
20
15
10
5
20
40
60
80
100
120
200
150
100
50


Iteration: 350
20
15
10
5
20
40
60
80
100
120
200
150
100
50


Iteration: 402
20
15
10
5
20
40
60
80
100
120
200
150
100
50

APPENDIX C: Machine Vision

Part A: Processing and analyzing an image

The image that will be used to develop the different tasks included in this part is called 'Whitby.jpg', it was acquired with a Samsung S5 on Whitby shore.

Figure 54: Original Image

Task 1

Upload the image to MATLAB and determine its properties. Determine its size. Explain the MATLAB functions used.

The properties of the image used are shown in the following figure. Its size is 5312x2988.

The functions used are 'imread' to read the image and store it in the workspace and 'imfinfo' that displays the information about the image.

```
          Filename: 'C:\Users\MI PC\Documents'
       FileModDate: '07-Apr-2018 11:07:17'
          FileSize: 5865741
            Format: 'jpg'
     FormatVersion: ''
             Width: 5312
            Height: 2988
          BitDepth: 24
         ColorType: 'truecolor'
   FormatSignature: ''
   NumberOfSamples: 3

      CodingMethod: 'Huffman'
     CodingProcess: 'Sequential'
           Comment: {}
              Make: 'samsung'
             Model: 'SM-G900F'
       Orientation: 1
       XResolution: 72
       YResolution: 72
    ResolutionUnit: 'Inch'
          Software: 'G900FXXU1CRA2'
          DateTime: '2018:04:07 11:07:16'
  YCbCrPositioning: 'Centered'
     DigitalCamera: [1×1 struct]
           GPSInfo: [1×1 struct]
     ExifThumbnail: [1×1 struct]
```

Task 2

Convert the image to an indexed image. Explain the difference between an indexed image and a true color image. Display the obtained indexed image in a colormap of your choice.

There are 4 different image types that MATLAB can support: binary, indexed, grayscale and true color.

Indexed images are formed by two matrices: The first one is the data matrix and it contains only integers, the second one is the colormap matrix. This type of image is also called pseudo-color image due to its color depends on the colormap selected to be applied with the data matrix, because color information is not carried by the last one.

On the other hand, true color images are formed by tree matrices that contains the intensity information in red, green and blue. The color of each pixel is obtained by the combination of those tree matrices, therefore the colormap is not necessary since the image itself contains all the information.

MATLAB has different predefined colormaps, in this case it has been used the default colormap called parula and colorcube.

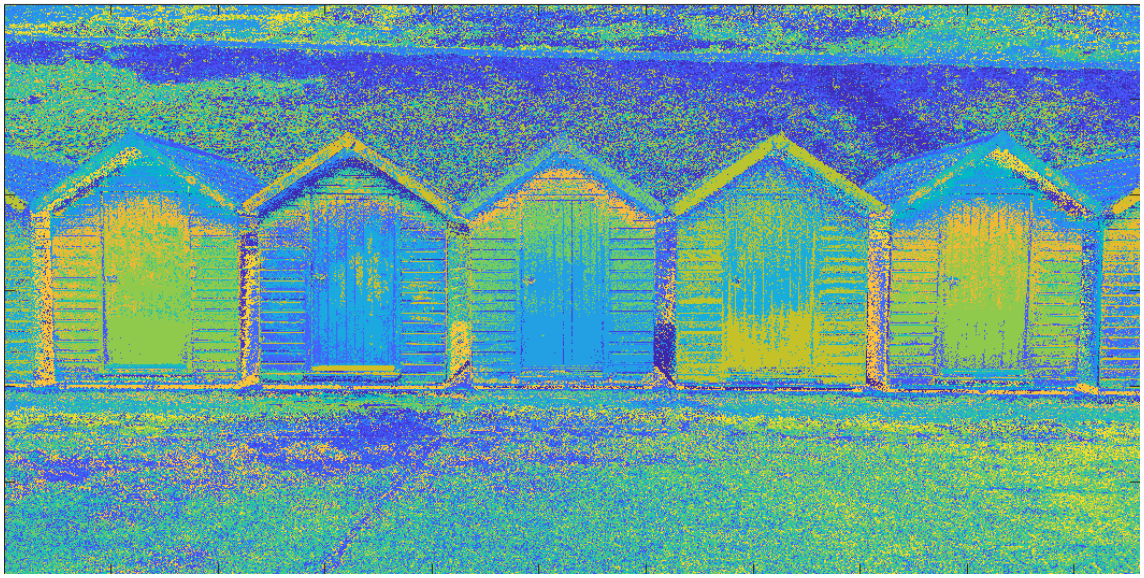


Figure 55: Default colormap parula

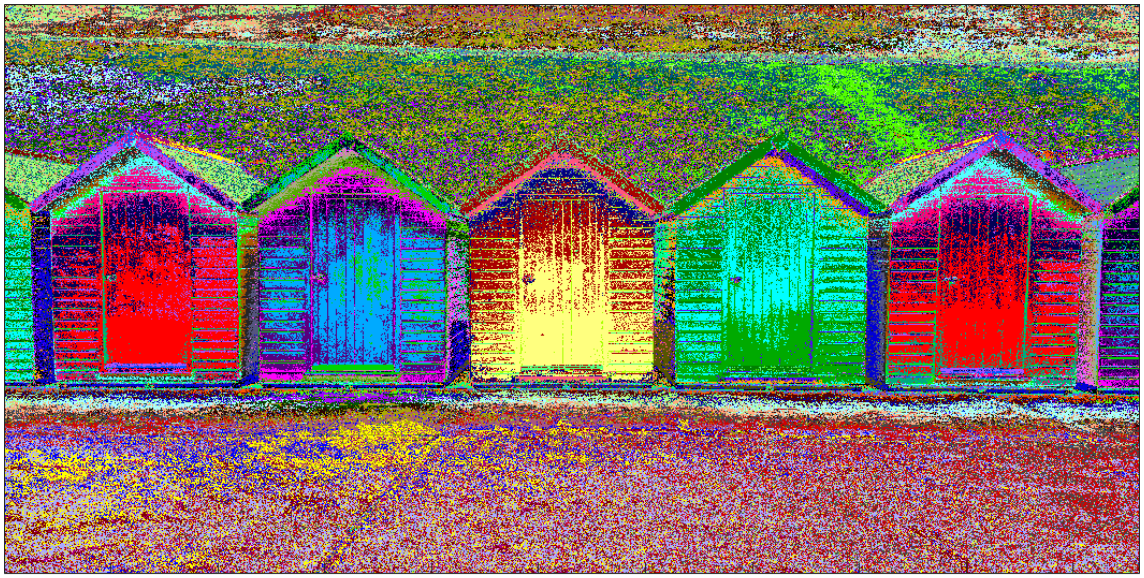


Figure 56: Colormap colorcube

Task 3

Convert the image to a gray-scale image. Determine the pixel intensity of each pixel and visualize the brightness profile in 3D. Produce the MATLAB code to perform visualization of the brightness profile in 3D.

The conversion to gray scale has been done in two different ways. The first one using MATLAB functions, in this case rgb2gray. The second one, by separating RGB channels and applying to them the coefficients of the human eye response to each channel according with:

$$I_{gray} = \alpha\, I_{red} + \beta\, I_{green} + \gamma\, I_{blue}$$

$$\alpha = 0.2989$$

$$\beta = 0.5870$$

$$\gamma = 0.1140$$

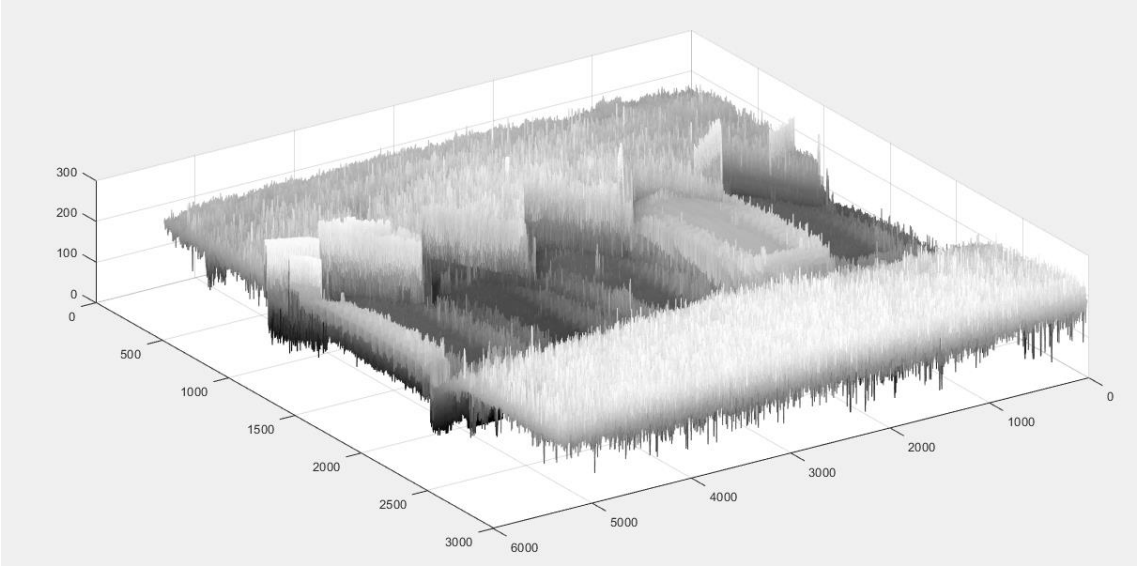


Figure 57: Brightness profile 3D

Task 4

Create MATLAB codes to produce the histograms of color channels in the true color image and the histogram of a gray-scale image.

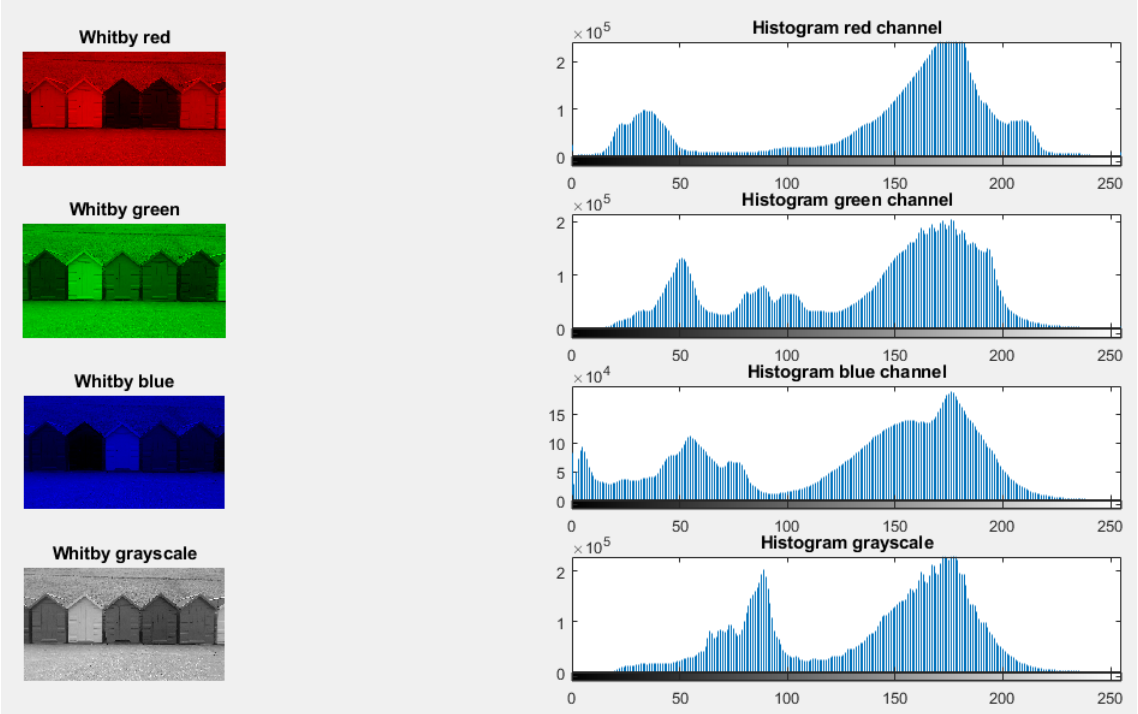


Figure 58: Histograms

A histogram is a plot of the frequency of occurrence of each color level in the image. In this case the histograms shown in the figure above represent the occurrence of red, green, blue and grayscale.

Task 5

Perform contrast enhancements of the gray-scale image by contrast stretching and by histogram equalization. Produce the corresponding MATLAB codes. Explain the effects of both enhancement procedures. Report on their attractions and possible undesirable effects.

The objective of these two methods is to adjust the intensity values of the image to cover the entire available range from 0 to 255. The technique used in each one is different.

The first one, contrast stretching, uses the lowest and the highest values of brightness in the image and sets these values as 0 and 255 and readjust the values between them. Whit this method is possible to restore the original image due to there is a one to one relationship between the values before and after. The problem of this technique is that if there is one point in the original picture whose value is in the limits, there will be no effect because the readjustment will not be effective.

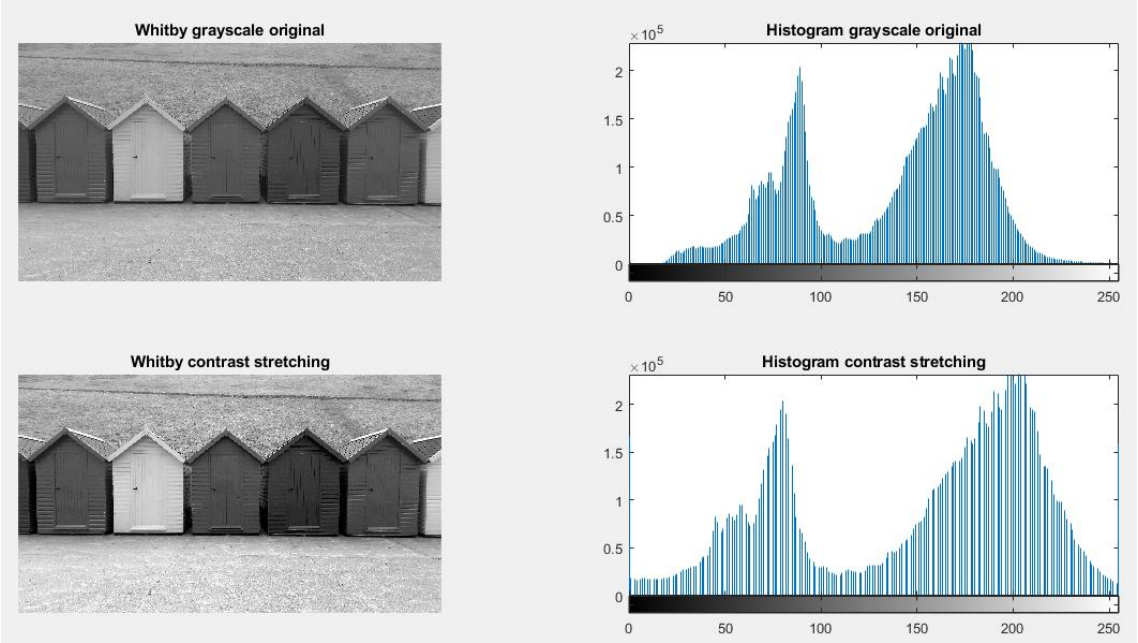


Figure 59: Original vs contrast stretching

The second method, histogram equalization, uses probability distribution to perform the readjustment. The objective is to obtain a histogram with a uniform distribution. By using this method, the contrast enhancement will always be performed but the problem is that it is impossible to restore the original image because there is no relationship between the pixel of the image before and after.

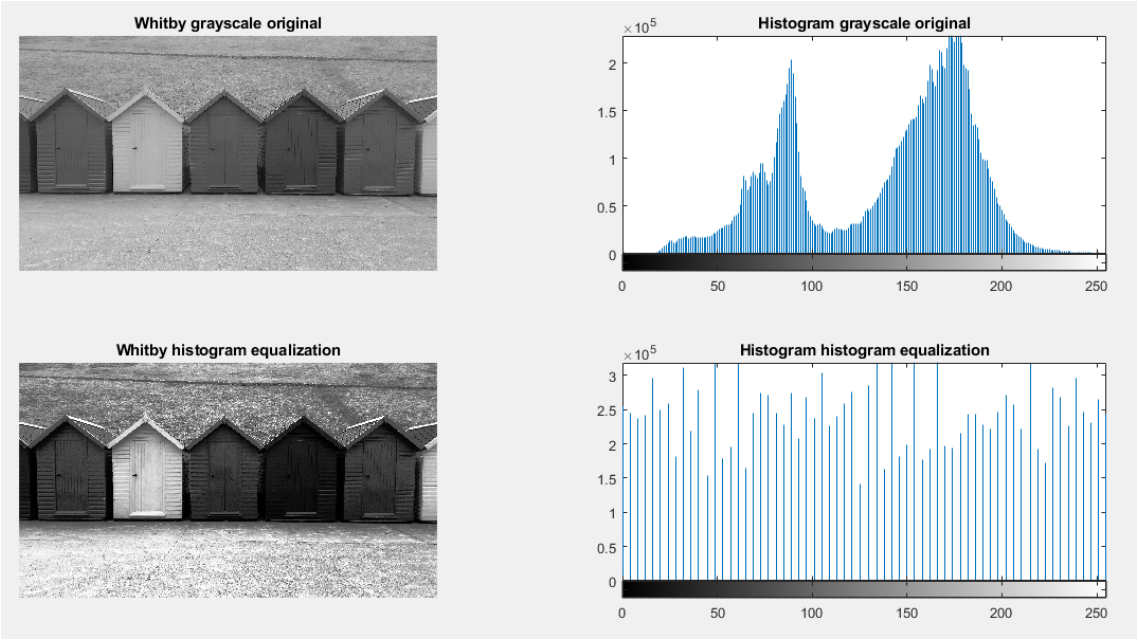


Figure 60: Original vs histogram equalization

Task 5.1

Perform histogram equalization on the true color image.

This task has been performed in two different ways. On the one hand, the RGB image has been separated into channels and histogram equalization has been performed in each one. Then, the new image has been built and displayed. On the other hand, the RGB image has been transformed to HSV and the histogram equalization has been performed only in value. The results are shown in the following figures:

Due to the similar results obtained by the two methods on the original image, the code has been checked using the image 'bot_garden2.jpg', the results are the following:

Figure 61: Histogram equalization RGB comparison on bot_garden2.jpg

As it is shown on the images above, the first method is not good in an image that contains low values of one of the main colors. In this case, the image of botanical garden has low content in blue and, because of histogram equalization is applied in each channel, the resultant image has unrealistic colors.

Task 6

Convert the gray-scale image to a binary image by using an appropriate threshold.

Figure 62: Whitby grayscale

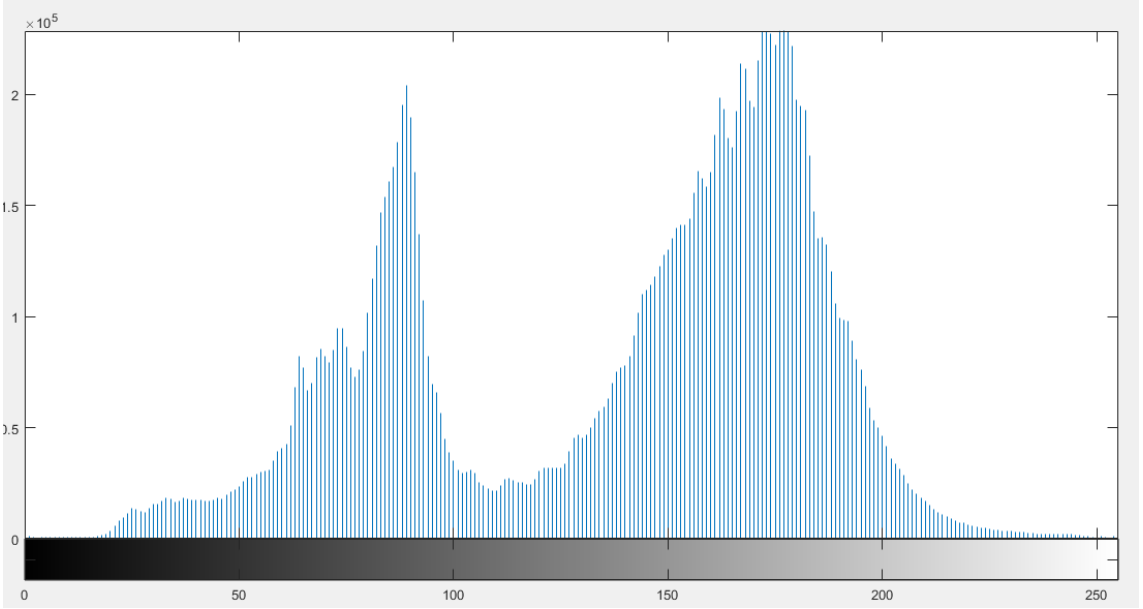


Figure 63: Histogram grayscale image

As it is shown in the histogram, the frequency of occurrence of the values of brightness is concentrated mainly in two pics. The first one, between 70 and 100, that represents the darkest areas in the image that are four of the five houses and the path in the hill. The second pick represents the brighter areas which are the remaining house, the ground and the grass in the background. According to this, has been decided to choose the value of brightness 100 to realize the binarization with the objective of separate these two areas described.

To verify if the selection of the threshold is correct, the process has been repeated, using in this case the function 'imbinarize' to compare the results. This MATLAB functions uses the method of Otsu to perform the binarization.

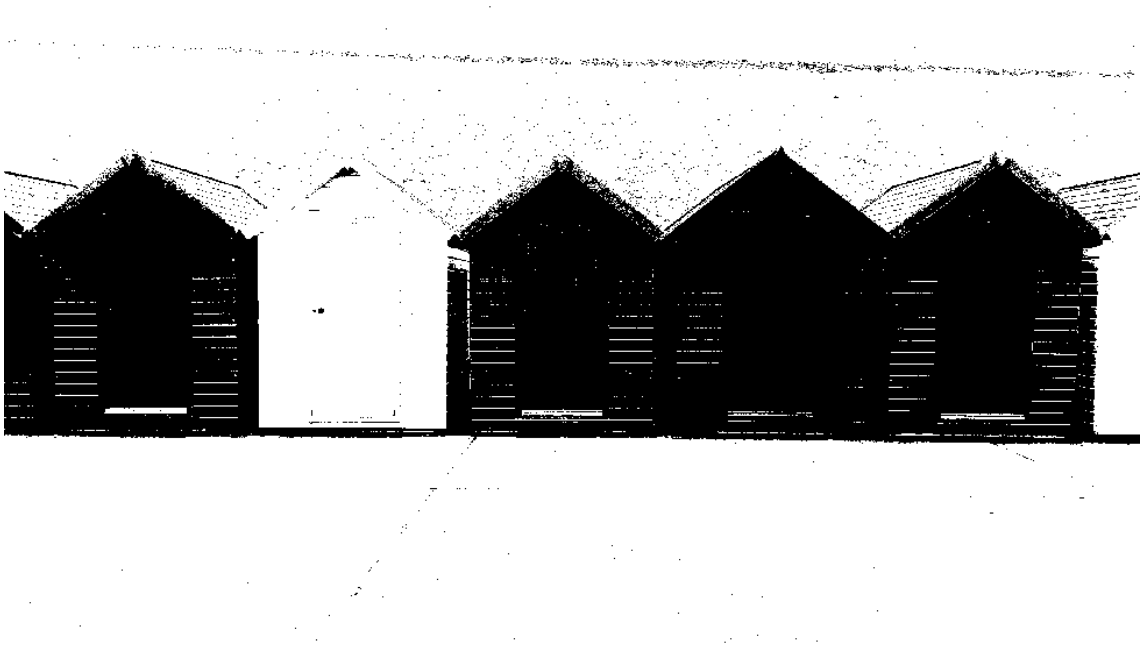


Figure 64: Manual threshold. Brightness 100.

Figure 65: Otsu's method

By comparing the results, the conclusion is that the solutions are similar, so the value selected to realize the binarization was correct. In the first case, it is possible to appreciate more details of the houses like the wood or where are the doors situated whereas in the second one there are almost no details on the facade of the constructions.

Task 7

Perform an operation of edge detection using a method of your choice. Briefly explain the key steps of the method.

The method used to perform this task is Sobel operator. It is a first order differential method through which intensity gradient is calculated in each pixel of an image. By performing this method, it is obtained the magnitude of the biggest change possible and its direction.

The gradients are computed in two directions, vertical and horizontal, by using these masks:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ for horizontal direction detection}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \text{ for vertical direction detection}$$

Due to the size of the masks, that is odd, the pixel is centered and it reduces interpolation errors furthermore larger size masks are better against noise.

The steps to perform this method are the following:

First, it is necessary to know the size of the original image to stablish the parameters of the loop. Then, in each loop iteration the gradients in horizontal and vertical direction are calculated and using these values it is possible to calculate the magnitude and direction of the edge.

Being I(x,y) the value of brightness of the image in the pixel position defined by the coordinates x and y, each gradient is calculated according to the following equations:

$$G_x\big(I(x,y)\big) = -1 * I(x-1,y+1) - 2 * I(x-1,y) - 1 * I(x-1,y-1) + 1 * I(x+1,y+1) + 2 * I(x+1,y) + 1 * I(x-1,y-1)$$

$$G_y\big(I(x,y)\big) = -1 * I(x-1,y-1) - 2 * I(x,y-1) - 1 * I(x+1,y-1) + 1 * I(x-1,y+1) + 2 * I(x,y+1) + 1 * I(x+1,y+1)$$

Using the values of the gradients obtained it is possible to calculate the magnitude of the edge:

$$E\big(I(x,y)\big) = \sqrt{G_x\big(I(x,y)\big)^2 + G_y\big(I(x,y)\big)^2}$$

And the direction, from dark to light:

$$D\big(I(x,y)\big) = arctan\left(\frac{G_y\big(I(x,y)\big)}{G_x\big(I(x,y)\big)}\right)$$

As a result of the performance of this method, the image obtained is shown below. In this case, the original image used is not suitable to perform edge detection considering the results because it is too dark which means that there are too many edges detected.

Figure 66: Edge detection Sobel method

Part B: Analysing the image of chocolate beans

Figure 67: Chocolate beans

Task 1

Threshold appropriately the image to emphasise the dark areas in it.

The procedure to perform this task is the same followed in Task 6 part A of this assignment. Summarizing, it is necessary to transform the true colour image into grayscale, obtaining Figure 68. Then, the information of the histogram of the grayscale image must be analysed to find the appropriate threshold.

According with the information shown in histogram in Figure 69, it is possible to differentiate between the darker areas, that correspond with the chocolate beans, whose values of brightness are between 20 and approximately 100 and the frequency of occurrence in this interval remains mainly constant. Next, the interval between 100 and 125 corresponds with the fold in the tissue that contains the chocolate beans. Finally, the peak between 125 and 180 is the color of the background of the image, in this case the white tissue. Keeping in mind this information has been considered suitable to stablish the value for the threshold in 100.

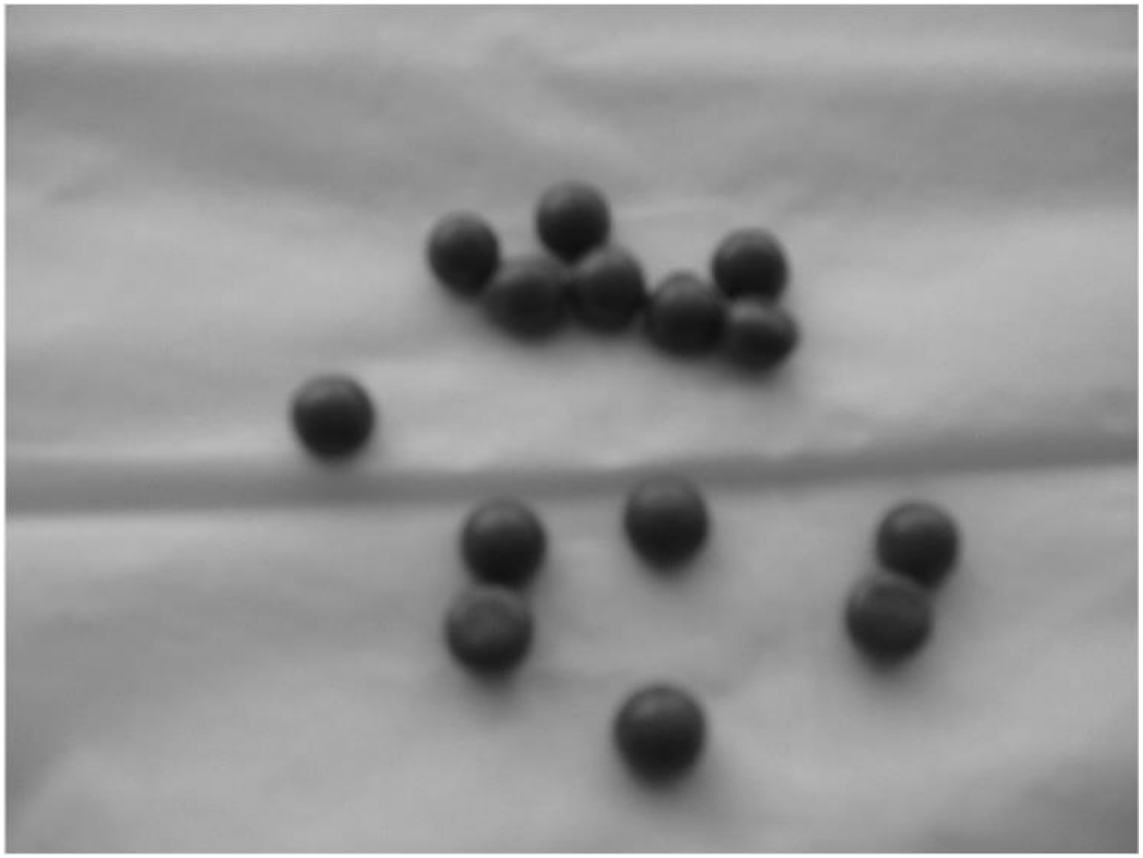


Figure 68: Chocolate beans grayscale

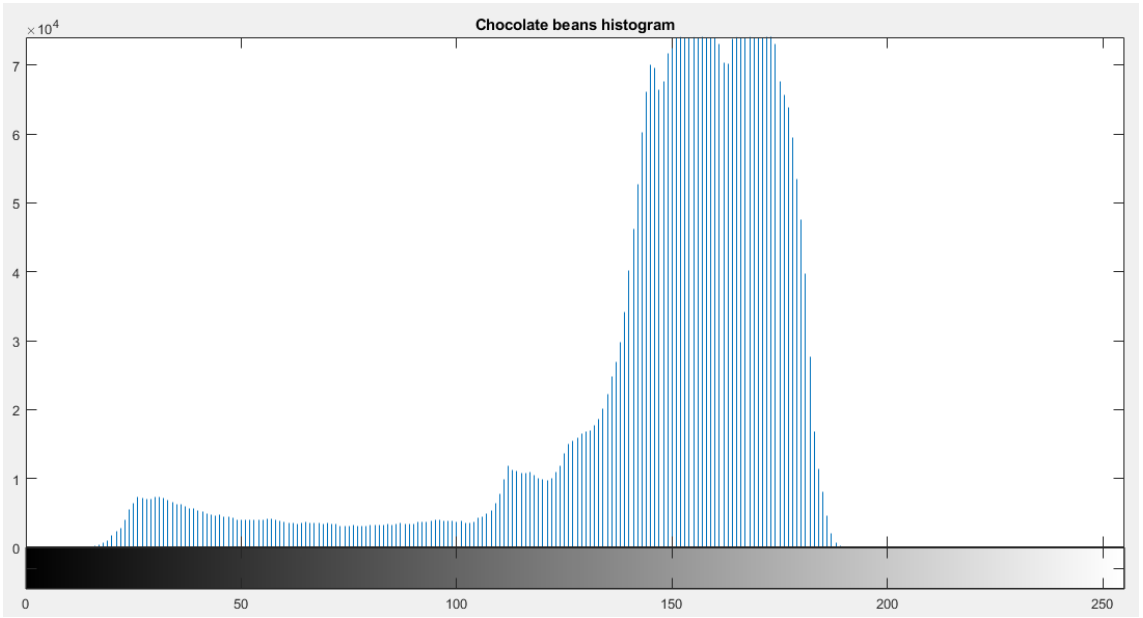


Figure 69: Chocolate beans histogram

As a result, the next image is obtained where the chocolate beans are clearly defined.

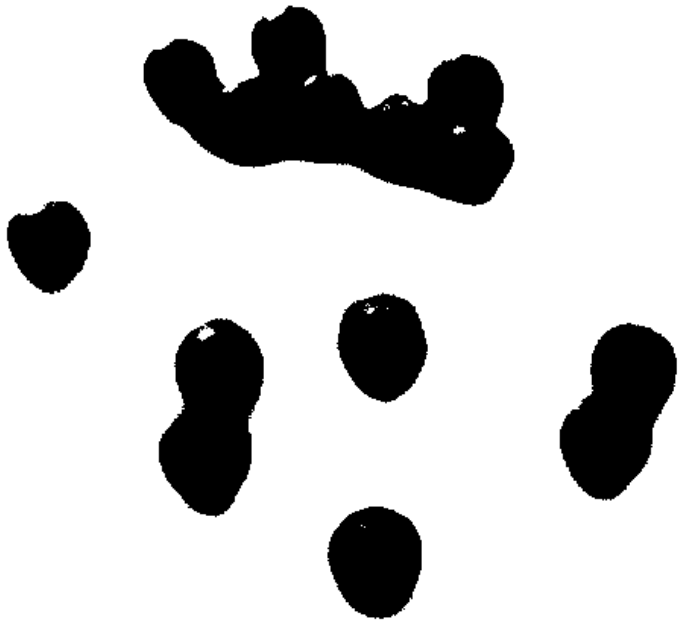


Figure 70: Chocolate beans threshold 100

Task 2

Compute approximately the area of the dark spots (in pixels) Explain the operations used.

The number of black pixels in Figure 70 is 348626. To obtain this result, a counter has been added in the loop that performs the binarization of the grayscale image to know how many pixels have been set to '0', in other words, how many pixels are whose brightness is under the value set as threshold.

Part C: Tracking a green circle in a life video

Task 1

Produce a code to tracking the green colour.

Once the parameters of the webcam attached had been set, the loop of data acquisition starts. In every iteration, a picture is taken and the green colour on it is subtracted in a different image and it is binarized. Then, by using the function *bwboundaries* with the option *'noholes'* the boundaries of all the green regions on the picture are stored and afterwards plotted.

It is important to remark that the light in the room and the quality of the camera are important to identify green colour. In this case, the quality of the webcam is not good, and it identify "light green" as it is shown in the next figure.

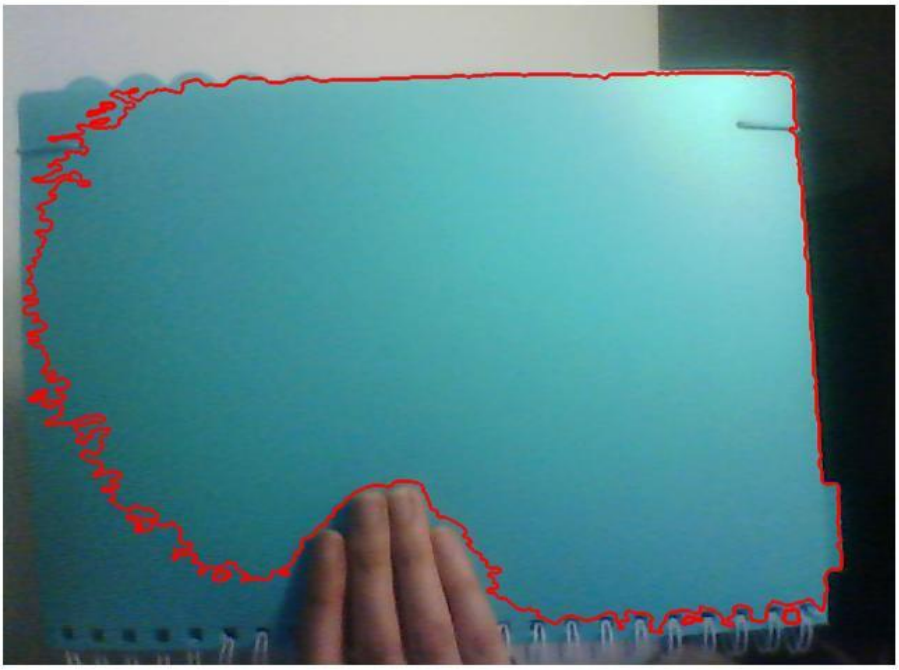


Figure 71: Green boundaries. Laptop webcam.

Task2

Be able to distinguish the circle from an object of the same green colour but of a different shape.

In addition to the code implemented in the last task, it has been necessary to use the function *regionprops* to know the area and perimeter of each green region in the picture. Using this data, the roundness of each object is calculated, and the boundaries are plotted in different colours to distinguish rounded shapes among the other shapes.

In this case, the image shown was taken with a modern webcam and there is a huge difference between the green colour in Figure 71 and the next one.

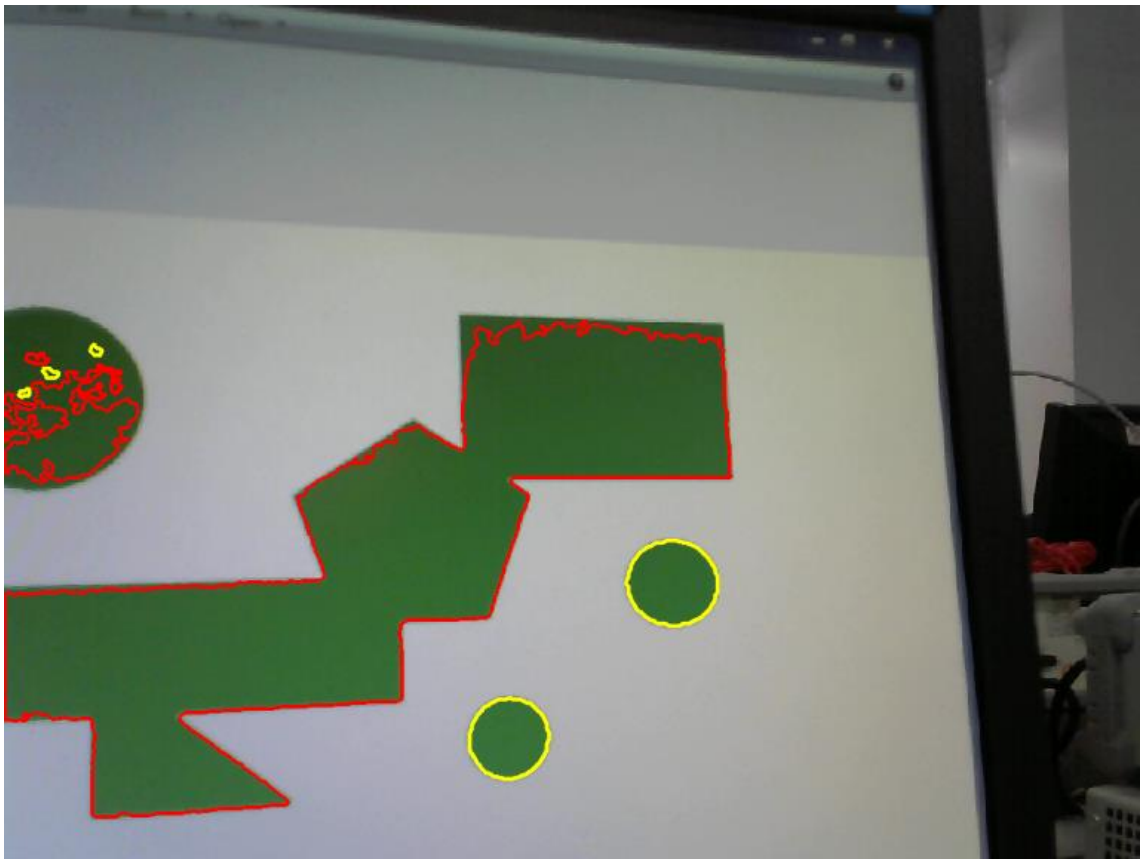


Figure 72: Boundaries and circles.

Part D: Case Study. Human Vision versus Animal Vision.

What is Vision?

Vision can be understood as the interpretation of the external world by internal coding and representation systems through the extraction of the information contained in retinal images. An accumulation of data reaches the brain from the eyes and it is processed in an efficient and quick way to provide a reliable representation of the environment. The complexity of this process contrasts with its versatility and, overall, with the simplicity whereby this information is assumed by humans and animals (Pons & Martínez, 2004).

Human interest in optical phenomena has been present since the beginnings of human history. Lindberg (1996) says that there are documents of speculations about the rainbow as old as the first written records or Egyptian ophthalmological recipes written in a papyrus from 1500 B.C., also mirrors and burning lenses of that period has been found. According to Putri (n.d.), the reason for the early interest of humankind in vision is because the eye is considered the most important organ of sense. For this reason, many theories have been enunciated over the years.

The aim of this essay is to show the evolution of knowledge in this field in order to be able to compare human with animal vision. To achieve this, first a summary of the most important ancient theories of vision will be exposed, then the characteristics and particularities of human and animal vision will be compared.

The Ancient Theories

The theories regarding the history of vision can be divided into three parts depending on who introduced them: the Greek philosophers, the Roman and the Arabs. Based on the work of Putri (n.d.) and O'Regan (n.d), a summary of the main points of the ancient theories is shown below:

- **The Greek philosophers:**
  - ***The Atomists*:** The main principle of their theories was that vision can only occur when existing contact or "touch" between the object and the eye. There are two assumptions about this, in the first one, the touch is produced by the compression of the air between the object and the organ. Whereas in the second one states that particles of the object fly into the eye.
  - ***Plato:*** He propounds that the eye emitted fire that combined with the ambient light generates a "body of vision", also known as emission theory.
  - ***Aristotle:*** He positioned himself against the two previous theories. He emphasized that the medium between the object and the eye was transparent and it was the responsibility of the movement of the colour particles. Visual perception is possible because the light is reflected by the object and enters the eye.
- **The Romans:**
  - ***Euclid and Ptolomy:*** The first one explained the mechanism of sight by the establishment of seven postulates, although his theory is only about the geometrical aspects of vision, he did not explain why things can be perceived. Then, Ptolomy extended the theory by adding the physical, physiological and psychological aspects.
  - ***Galen:*** In his theory, the soul sensed the objects, and that was possible through a visual spirit pneuma which travels along the optic nerve setting a connection between the eye and the brain.
- **The Arabs:**

- ***Al-Kindi:*** He was influenced by Aristotle, Galen and Euclid, and attempted to improve them and introduce the concepts to the mass.

The understanding of eye mechanism has been advancing from the first theories based on speculations until today.

Human Vision

The eye

The eye is the organ that detects light and it is the responsible of the sense of sight. Its main function is to transform the light energy into electric signals that are sent through the optic nerve to the brain (Wikipedia, 2018). Its anatomy is shown in the following figure:

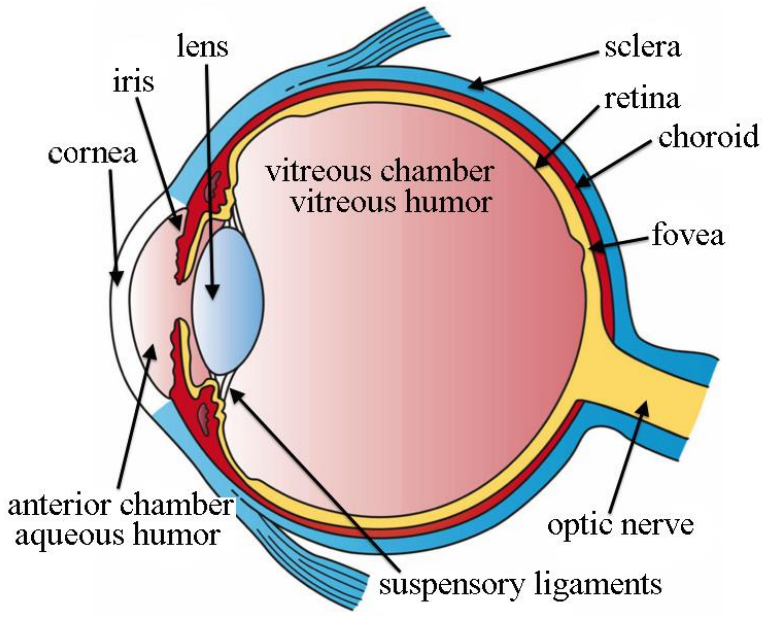


Figure 73: The structures of the eye. (Wikipedia, 2018)

The eye receives the light stimulus from the environment. The light goes through the cornea and the lens and forms an inverted image on the retina. Specialized cells transform the image into nerve impulses that are sent to the posterior region of the brain where the information is processed.

Each part of the organ has an important purpose in this process, below is shown a summary of the most important parts and its function:

- ***The Pupil and iris*** are responsible for the amount of light that enters the eye. The size of the pupil is adjusted by the iris.
- ***The Cornea and lens*** are the responsible for adapting the trajectory of the light to be projected on the retina.
- ***The Ciliary body*** contracts and expands to allow the light to be focus on the retina.
- ***The Retina*** contains the cells that catch the light.
- ***The rods and cones*** are the sensorial cells contained into the retina. Rods allow to distinguish between black, white and gray. Cones make possible color sight, there are three types, sensible to red, green and blue color.

Color vision and visual acuity

Color vision can be defined as *"the ability to distinguish objects based on the wavelengths of the light they reflect, emit or transmit"* (Wikipedia, 2018). As it was mentioned before, human eye contains rods and cones, concretely around 6 million of cones and 110 million of rods that are the responsible of human tricolor vision.

Cone type	Name	Range (nm)	Peak wavelength (nm)
S	β	400-500	420-440
M	γ	450-630	534-555
L	ρ	500-700	564-580

Visual acuity is the capacity to perceive, detect or identify objects with good light conditions, in technical words, is the resolution of the visual system. It depends on optical and neuronal factors such as the shape of the eye or the capacity of interpretation in the brain. For humans, a normal visual acuity is 20/20 (feet) or 6/6 (meters). The numerator indicates the distance between the subject and the chart and the denominator is the distance a person with 6/6 acuity would see the same. For example, a visual acuity of 6/3 means that a person has twice spatial resolution and needs half the size to discern the figure. On the other hand, 6/12 means the contrary, this person needs twice the size and has half resolution than normal 6/6.

Binocular vision

Howard and Rogers (2008) said that every animal with two eyes have binocular vision but it usually refers to *"those animals who possess a large area of binocular overlap and use it to code depth".*

According with Pons and Martínez (2004), there are four conditions that must be verified to consider binocular vision:

- Both fields of view must overlap in a wide area to obtain a large binocular field, in other words, eye orbits must be frontal.
- The eyes must move coordinately to form the images on symmetric areas of the retinas of both eyes.
- The information received in each retina must maintain a correspondence with each other during all the processing of information.
- The brain must have the capacity to merge the two neuronal images into one unique representation.

The principal utility of binocular vision is that by losing amplitude of the field of view precision in the perception of depth is won. This effect is due to the brain can obtain the distance, and consequently the depth, from the images provided by each eye. These images are taken from slightly different positions and, by its analysis, the human brain can correct the mistakes committed by one eye with the correct details obtained by the other (Wikipedia,2018).

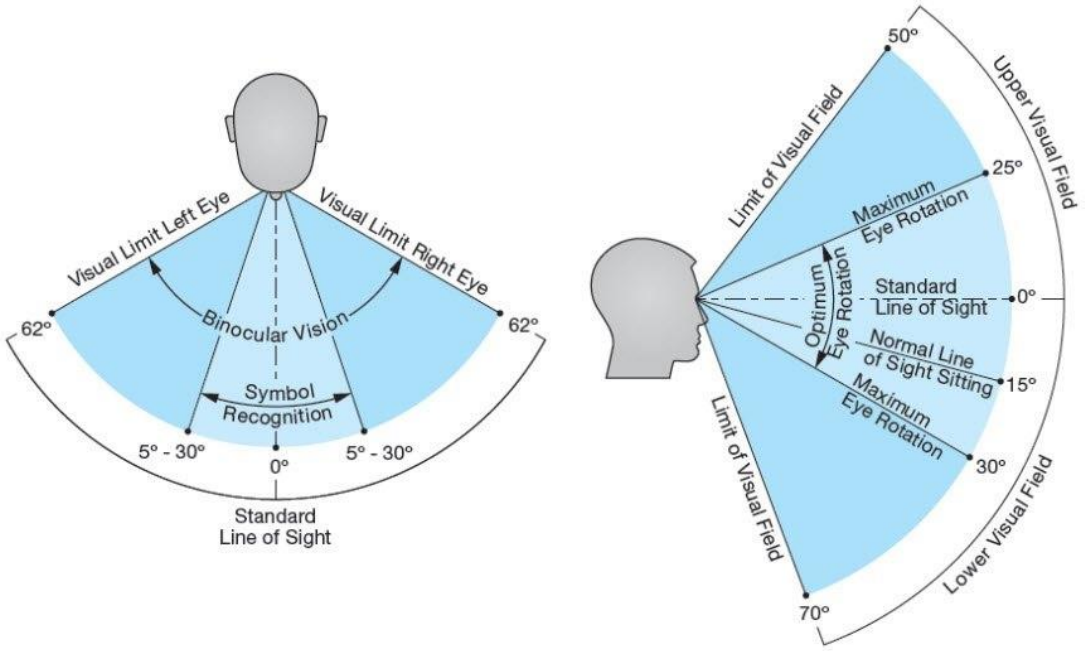


Figure 74: Human binocular field. Image from Quora (2018).

An interesting example of the perception of the space for humans is shown in the work of Pons and Martínez (2004). With only one eye humans have spatial sense, but depth perception is not as good as if both eyes are used. The example proposed by them to prove it is as easy as holding a pen in the right hand in front of your face, with the arm stretched, keeping only the right eye open and try to reach the tip of the pen with the left hand.

Summarizing, the anatomy of the eye is the responsible of adjust the image in the retina, where its cells process the information. First, cones and rods, capture the light and then other cells transform this information into electric impulses that travel through the optic nerve until the brain. Then, the information is processed, the colors are formed, and the reconstruction of distances, movements and depth distinction takes place. As Pons and Martínez (2004) stated *"the human eye is considered one of the biggest achievements of natural evolution".*

Human vision versus animal vision

The main points discussed above, anatomy, color vision, acuity and binocular vision, will be exposed and compared for animal vision.

There are anatomic differences between human eyes and animal but also, inside the animal kingdom, there are huge differences between species and the reason is evolution. All animals, including humans, have been evolving to adapt. The function of the eye is the same for all, to catch the light, transform and transport it to the brain, where it is processed.

One of the most obvious differences is the number of eyes, humans and many animals have only two but there are animals with three, four, five, six or more. For example, there are spiders with four, six or eight eyes, while bees have five eyes, or some reptiles or fishes have a third eye called parietal eye.

Figure 75: Parietal eye. Wikipedia (2018)

In addition, there are animals such as fishes, snakes or insects don't have eyelid to protect their eyes, by contrast, camels have three to protect themselves from sand storms. Talking about size, ostriches have one of the biggest eyes, even biggest that their brain (MuyInteresante, 2018).

Regarding color vision, not all animals have tricolor sight. For example, dogs have only two sets of cones so the cannot distinguish between red from green. Other animals, like birds or insects, can detect ultraviolet light to help them find nectar. In the following table extracted from Wikipedia (2018) there is a resume of the type of color vision of the different species:

State	Types of cone cells	Approx. number of colors perceived	Carriers
Monochromacy	1	100	Marine mammals, owl monkey, Australian sea lion, achromatprimates
Dichromacy	2	10,000	Most terrestrial non-primate mammals, color blind primates
Trichromacy	3	10 million	Most primates, especially great apes (such as humans), marsupials, some insects (such as honeybees)

Tetrachromacy	4	100 million	Most reptiles, amphibians, birds and insects, rarely humans
Pentachromacy	5	10 billion	Some insects (specific species of butterflies), some birds (pigeons for instance)

In addition, some animals, especially nocturnal animals have bigger number of rods which means that they have sight in darkness conditions than humans.

As it was said before, evolutionary factors are crucial in animal color perception, concretely, color perception has been improved with the purpose of finding food easily. For example, adequate light is needed for cones to function properly, for that reason nocturnal mammals have bad color vision. On the other hand, for herbivorous a good color perception is essential to recognize green leaves.

Binocular vision in animals is also highly influenced by evolution, mainly in the field of view, that depends on the position of the eyes. For example, predators like wolves or eagles have a binocular vision similar to humans because they need precision in order to hunt. On the other side, the prey like rabbits or zebras, have a wide field of view which means that they have a small binocular field. The he reason is because they need to be alert and cover the maximum field around them.

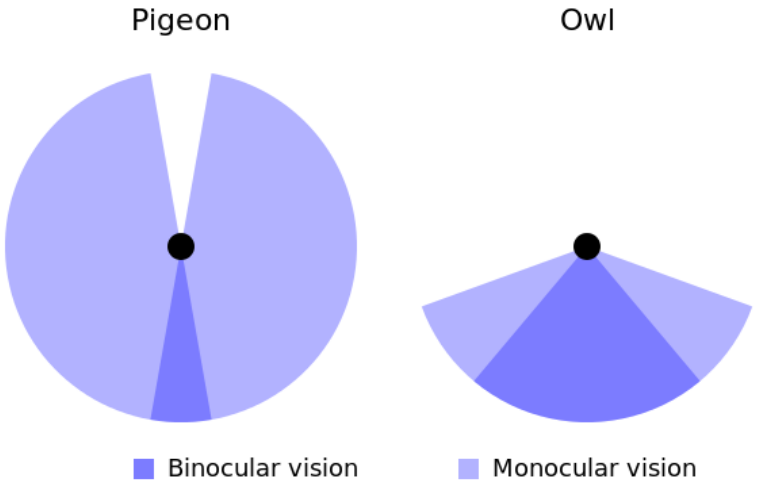


Figure 76: Differences in the field of view between a dam and a predator. Wikipedia (2018)

Finally, talking about acuity its general characteristics are like binocular vision. Predators have good acuity and can clearly identify their prey at long distances, for example, felines can distinguish a mouse up to 75 meters or eagles have acuity 20/4. On the other hand, prey usually have wide field of view, as it is said before, but low acuity. An example of that is the horse that have 20/70 or dogs with 20/75 which means that what a human can see at 70-75 feet distance is approximately what a dog or a horse can see at 20.

To sum up, it is important to remark the importance of evolutionary factor, as it has been discussed before, because depending on the environment and the habits of the animal, they have evolved

more in some attributes rather than others to survive. Humans do not have the best qualities in any of the aspects shown, due to there are lots of animals with better color sight like butterflies, better acuity and binocular sight like owls or better aptitudes to see at night like felines. In general, animals have usually one attribute highly evolved but with lack in the others. Predators like wolves have good acuity and binocular sight but lack in color vision, or herbivorous have good color vision to recognize the leaves and wide field of view but low acuity. So, it is possible to affirm that the advantage of human vision against animal vision is that humans have a good combination of attributes without any "lack" and a powerful brain.

References

Howard, I. and Rogers, B. (2008). *Binocular vision and stereopsis*. Oxford: Oxford University Press.

Lindberg, D. (1996). *Theories of vision from al-Kindi to Kepler*. Chicago: University of Chicago Press.

MuyInteresante.es. (2018). *Curiosidades sobre los ojos - Dormir con un ojo abierto*. [online] Available at: https://www.muyinteresante.es/naturaleza/fotos/curiosidades-sobre-los-ojos/dormir-con-un-ojo-abierto [Accessed 20 Apr. 2018].

O'Regan, K. (n.d.). *Ancient Visions*. [online] Available at: http://nivea.psycho.univ-paris5.fr/FeelingSupplements/AncientVisions.htm#_ednref3 [Accessed 19 Apr. 2018].

Pons Moreno, A. & Martínez Verdú, F. (2004). *Fundamentos de visión binocular* (pp.15-22). València: Publicacions de la Universitat de València.

Putri, I. (n.d.). *Ancient Theories of Vision and Al-Kindi's Critique of Euclid's Theory of Vision.*

Quora (2018). *What is the maximum human field of vision? - Quora.* [online] Available at: https://www.quora.com/What-is-the-maximum-human-field-of-vision [Accessed 20 Apr. 2018].

Wikipedia (2018). *Ojo humano*. [online] Available at: https://es.wikipedia.org/wiki/Ojo_humano [Accessed 20 Apr. 2018].

Wikipedia (2018). *Human eye*. [online] Available at: https://en.wikipedia.org/wiki/Human_eye [Accessed 20 Apr. 2018].

Wikipedia (2018). *Visión binocular*. [online] Available at: https://es.wikipedia.org/wiki/Visión_binocular [Accessed 20 Apr. 2018].

Wikipedia (2018). *Color vision*. [online] Available at: https://en.wikipedia.org/wiki/Color_vision [Accessed 20 Apr. 2018].

Wikipedia (2018). *Parietal eye*. [online] Available at: https://en.wikipedia.org/wiki/Parietal_eye [Accessed 20 Apr. 2018].

APPENDIX D: Forms

In this appendix, the different forms required for the submission of this project are attached in the following order:

- Project Specification form
- Risk Assessment
- Ethics Checklist
- Project Supervisor form