MASTER´S THESIS

Màster Universitari en Automàtica i Robòtica

VISION BASED TRAJECTORY PLANNING FOR ROBOTIC ASSISTED FETAL SURGERY TREATINT TTTS

Report and Appendices

Author:	Javier Asensio Baeza
Director:	Alícia Casals Gelpí
Co-Director:	Narcís Sayols Baixeras
Call Date:	Junio 2019

Abstract

Medical Robots is the field focused on improving and making easier the work of medical personnel in certain interventions with the help of robotic systems. Because of this, although some of them are already in use, there is a large number of R&D projects, especially in soft tissues. In this case, this project deals with the improvement of fetal surgery, more specifically in the treatment of Twin-to-Twin Transfusion Syndrome.

TTTS is a syndrome that affects pregnancy in twins. When it occurs, during the development of fetuses, there is an interconnection between the blood vessels of both in points called anastomosis. These connections cause the exchange of blood flow between both fetuses and, if it is not treated by fetal surgery, results in the death of both twins.

A teleoperated robotic system is being developed in the ESAII laboratory to provide help and assistance to the surgeon during these surgeries. In this project an automation of the robotic system is implemented. It is done by means of using the work environment information collected and Computer Vision tools. The objective is to create an automatic movement of the robot through the fastest and safest path from one point to other over the placenta´s surface.

This report details everything about the project development. It is also described the main topics related to the project as the global robotic system, designed and made in the ESAII laboratory of the UPC; the fetal surgery and the current state of the art around this type of medical robots.

Acknowledgements

Thanks to my parents, Jose Antonio and Mª Luisa, for helping me to get opportunities, for supporting me always on my way and for showing their happiness when I achieve my goals.

Thanks to my girlfriend, Anna, for supporting me day and night, for helping me in everything and for making me immensely happy.

Thanks to all my fellows and friends of the ESAII laboratory team for receiving me so great, for making me feel like one more of them and for helping and supporting me throughout this project.

Thanks to Rut, Tomàs, Alice, Narcís, Albert and Alícia for their help, company and good times.

Thanks to Albert, Narcís and Alícia for dedicating their time and effort in supporting and helping me to make this project as better as possible.

Thanks to all the UPC teachers for teaching me, helping me and making me enjoy even more the robotics world

Thanks to all my friends and my family for always being there.

.

Index

Figures Index

Tables Index

1 Introduction

Computer vision is an engineering branch that is becoming increasingly important in different fields, such as minimally invasive surgery, MIS. Despite the great advantages of MIS compared to open surgery, including reduced trauma, pain and post-operative recovery time, MIS still has many limitations. One of the most used tools to overcome some of these limitations is Computer Vision, RMIS. Real-time image analysis provides additional information to the surgeons and to robotic control system. Image analysis can be used for guiding tools in robot surgery, stabilizing the region of interest, detecting collisions between tools and organs or delicate tissues, locating the tool over a point of interest (POI), navigating over collision-free regions, etc. In this case, computer vision is used to assist the surgeon in fetal surgery, specifically in Feto-fetal transfusion syndrome or Twin-to-Twin transfusion syndrome, TTTS.

1.1 Feto-fetal transfusion syndrome

TTTS is a dysfunction that can be observed in monochorial placentae and it may create an unbalanced transfer of blood between twins. Both circulatory systems are connected by some points called anastomosis points. According to different studies, vascular anastomosis appears in the 85-100% of cases that presents a monochorial placentae.

There are three different types of vascular communications, the most common is the arterio-arterial type, but there exist also the veno-venous and the arterio-venous anastomosis, which is probably the one with greater pathological significance.

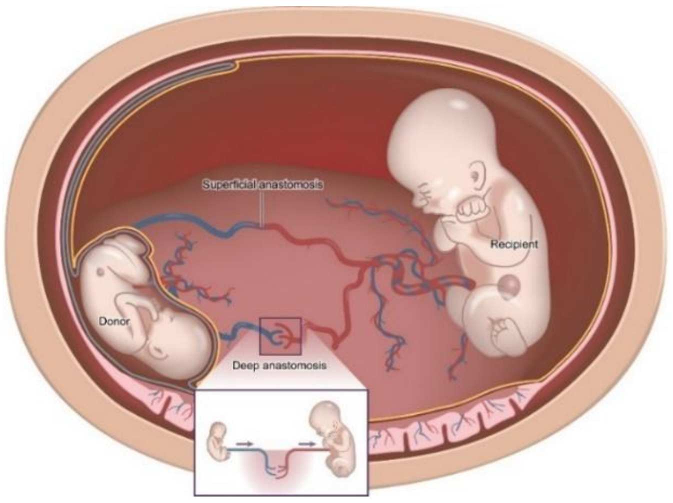


Figure 1. Feto-fetal transfusion syndrome illustration

During pregnancy, the donor fetus suffers hypovolemia and hypertension, while the receptor suffers hypervolemia and also hypertension since it is receiving substances from the vessels of the other fetus (**¡Error! No se encuentra el origen de la referencia.**). Feto-fetal transfusion syndrome is one of the main causes of mortality in monozygotic twin pregnancies. (P. Galea, 1982)(Rogelio Cruz-Martínez, 2014)

1.2 Current Surgery

Nowadays, the fetuses are considered as patients. With the evolution of surgical techniques, especially in MIS, it is possible to perform surgeries to the fetuses with TTTS among other disfunctions.

Since MIS requires indirect vision during surgery (Figure 2), the tool that allow such visualization is inserted through an access point using a trocar. In this case, the tool used is the fetoscope.

Fetoscope is a surgical tool with optical fiber that transmits the image to an external camera that allows surgeons to visualize images of the inside of the uterus, including the surface of the placenta (Figure 3, b), so that surgeons can explore the region and perform the surgery. This device consists of a fibre optic channel to transmit the image to the camera, another fibre channel to transmit light to the interior of the work space and several auxiliary channels, including the fibre optic channel that transmits the laser light spot used to coagulate the anastomosis points.

The anastomosis points are the places where the connection between the circulatory systems of both twins takes place, producing the exchange of blood flow and therefore the TTTS. The current treatment to solve TTTS is the cauterization or coagulation of these anastomosis by applying a laser spot during the fetoscopy (Figure 3, image c and d).

At present, the process is done betweem the 15th and the 26th week of pregnancy, but some tests have demonstrated that is also possible until the 29th week. Results in the last 10 years show between 75% to 88% probability of saving at least one of the fetuses; and a 60% probability of saving both twins. (Rogelio Cruz-Martínez, 2014) .

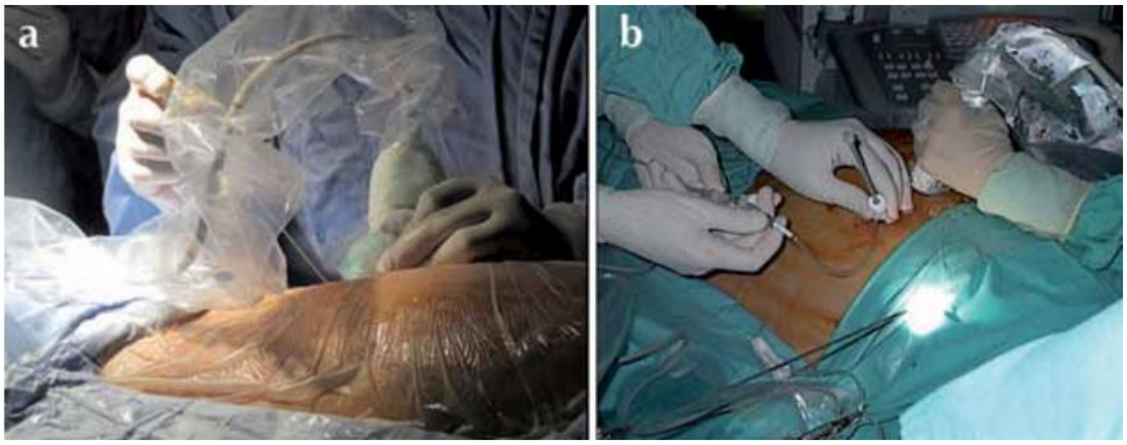


Figure 2. MIS in fetal surgery

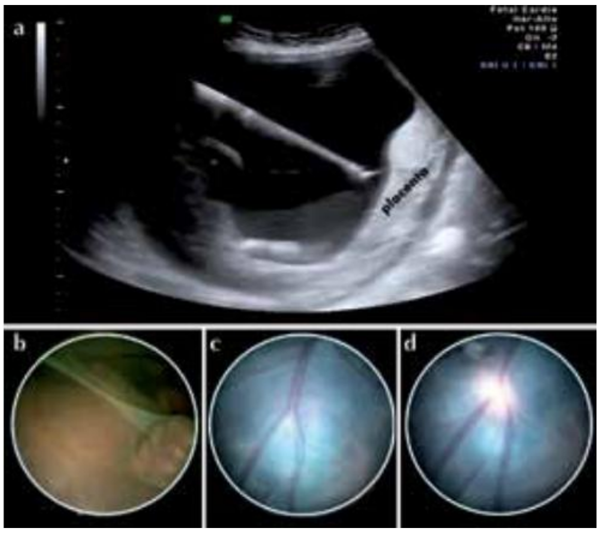


Figure 3. Ultrasound image during the use of the laser to coagulate.

2 Teleoperated Robotic System for TTTS Surgical Assistance

This work is part of a research project that aims to develop a robotized teleoperation system for fetoscopic surgery in order to improve surgeon´s performance and patient’s safety to treat TTTS.

The teleoperated system consists of a single master and a single slave robotic arm with a fetoscope and a laser based coagulation device as end-effector. In this manner, the surgeon controls the robotic arm through a haptic device as a Phantom Omni, pedals and an advanced interactive user interface. The system includes virtual aids like filters to avoid the movements due to the human tremor; or motion scale factors. The system also modifies the motion scale of the movements made by the surgeon with the phantom, in a range more convenient for the precise movement of robot over the placenta, increasing the human accuracy. All this is done in order to facilitate the work of the surgeon, increase patient safety, reduce surgery time, besides making this work more comfortable, trying to reduce his/her mental and physical workload.

2.1 Control structure

The global control system consists of three main parts, the Master, the Slave and the central teleoperation control system.

2.1.1 Central teleoperation control system

This module is in charge of computing robot kinematics. It receives the orders coming from the Master module and makes that the robot and the fetoscope move as it has been sent. These movements are restricted by the presence of a pseudo-fixed point (fulcrum). It performs a high-level control of the robot and it takes into account the environment, thus controlling the fulfillment of the restrictions. For instance, computing a safety distance between fetoscope and placenta, which is defined by the surgeon. The module also controls the system of trocar compensation (avoids undesired changes on trocar depth inside uterus) and the laser coagulation system.

This research work, which consists of the automation of safety and optimized path generation, is part of the main control. Surgeons can use this tool to navigate through these points, as the robot automatically guides the fetoscope safely and efficiently.

For this purpose, optimized paths are created using the trajectory map of previous fetoscope movements. These trajectories are generated during the manual phase, while locating the equator zone (zone of maximum probability of finding the anastomosis points) and the POI.

The trajectory map is treated and processed by computer vision tools. In this way, the final result is obtained, which is the trajectory between two points. The objective is generating a trajectory from an initial to a desired point minimizing distance and maximizing safety.

In terms of security, only those trajectories whereby the surgeon has guided the fetoscope are considered. Also, in case of wide regions, where the fetoscope can move freely within these zones (Figure 4), the central points of these trajectories are chosen for more security. Regarding to the economy of movement, Dijkstra algorithm, modified to fulfil the needs of the objectives, is applied to obtain the costless path. In this case, only distance affects the cost of the paths. Once the fastest and safest path has been found, the trajectory is sent to the robot. Finally, the robot proceeds with the movement, but always under the supervision of a surgeon and a technician.

In addition, the information provided by this module is also useful for certain visual interpretation of the working environment either to track a point, compensate the movements of the system, etc.

Concerning communications, this system is in charge of the communication between master and slave modules and monitors their status (Figure 5).

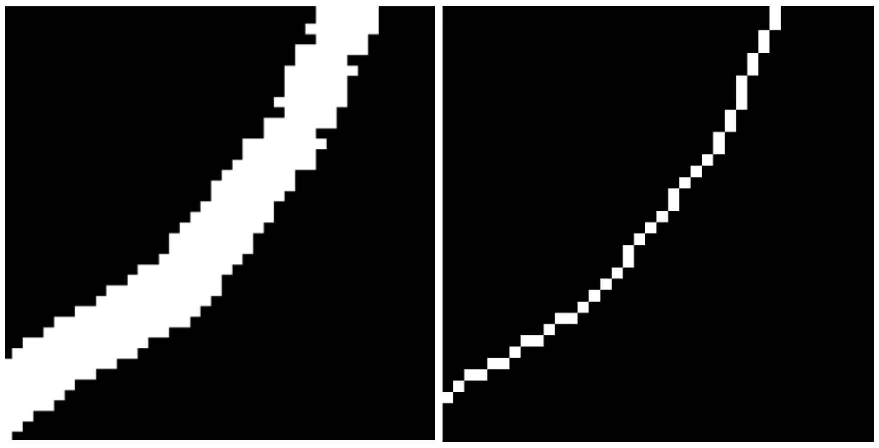


Figure 4. Compared wide region with the final chosen central points

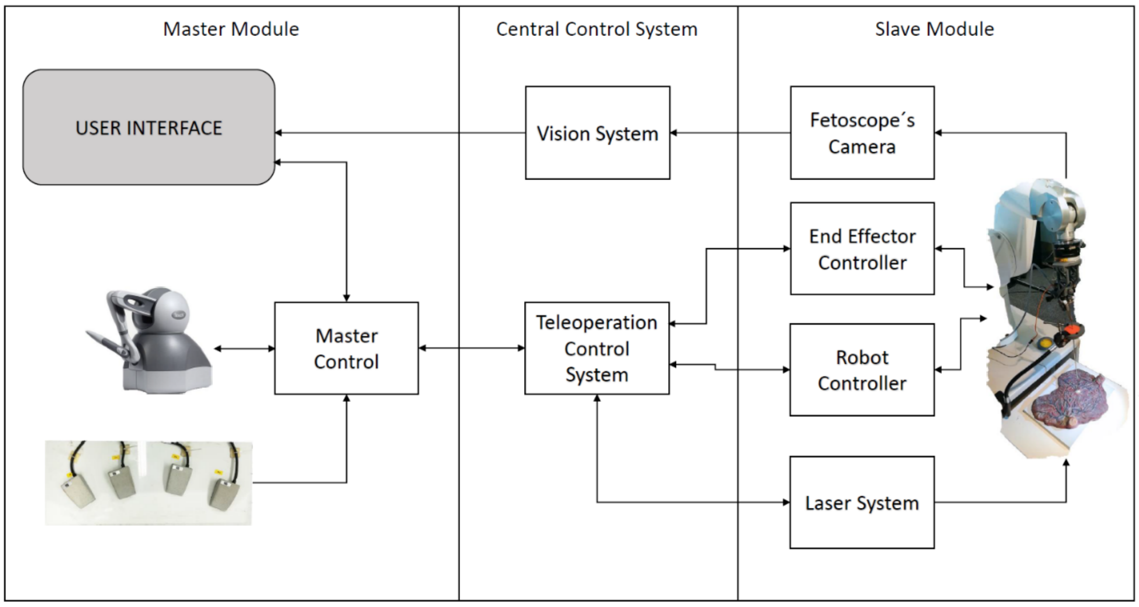


Figure 5. Global architecture

2.1.2 Master

Master is the module through which the interaction between the user and the system takes place. This is the module that guides the robot, activates the laser, provides the user with certain information through the User Interface (UI), etc. Therefore, it is the module for commanding the system.

For its function, the module is provided with different components:

- **Haptic device**: The haptic device is an arm with encoders on the joints to obtain the end-effector pose. The user grasps the end-effector (pencil-like) to interact with the device (e.g: describing a trajectory). They are used in various applications such as 3D modelling, research, training and testing, etc. In this case, the device is controlled by the surgeon and the desired movements are transmited to the fetoscope. It also has buttons, which in this case have been used to activate or deactivate the movement of the robot, providing safety and manoeuvrability. (Figure 6)

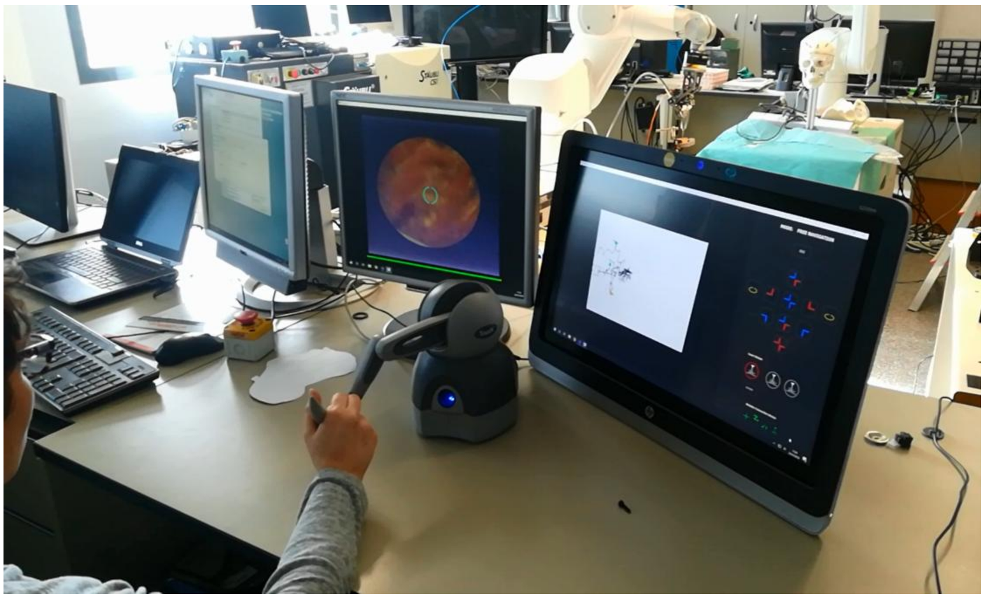


Figure 6. Haptic (Phantom) device used for the teleoperated movement of the fetoscope

- **Pedals**: The right foot has the emergency pedal. In the event of a dangerous or emergency situation, the surgeon may press this pedal, causing the robot to move in the Z tool axis (out of the mother's abdomen) and then remain immobilized until the surgeon sends the information that everything is correct again. The left foot is placed above the coagulation pedal. This pedal is used to activate the laser. This is only possible when the tip of the laser is outside the cannula. (Figure 7)

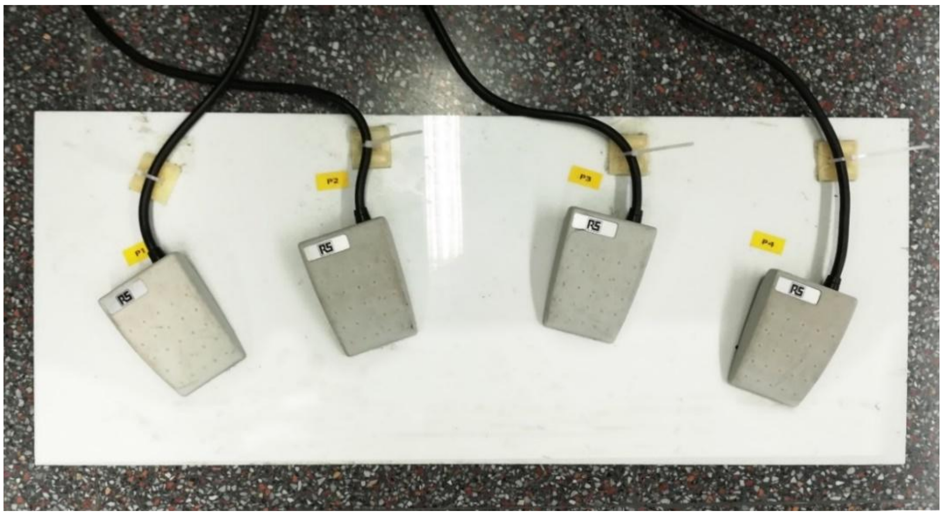


Figure 7. Pedals used by the surgeon. From left to right: Coagulation – NA – NA – Emergency

- **User interface**: The purpose of the user interface is to provide the surgeon with all the necessary data and information about the system and surgery in real time. The interface shows the images collected by the fetoscope camera (Figure 8), the status of the connections with the modules, the state and position of the laser, the map of trajectories and the points marked during the location phase (Figure 9). The interface also includes data input such as modifying the position of the laser or marking those POI desired by the surgeon.

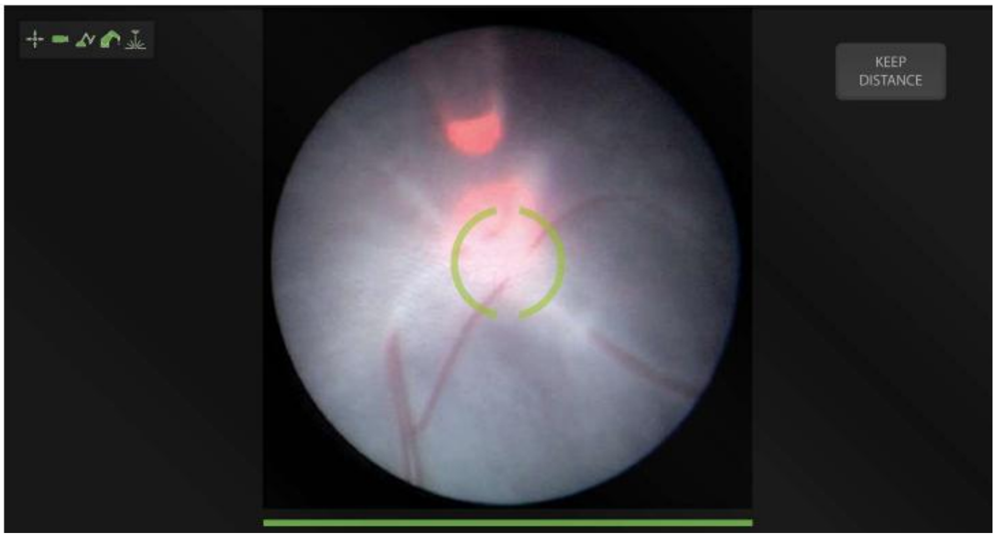


Figure 8. Visualization of images sent by fetoscope camera

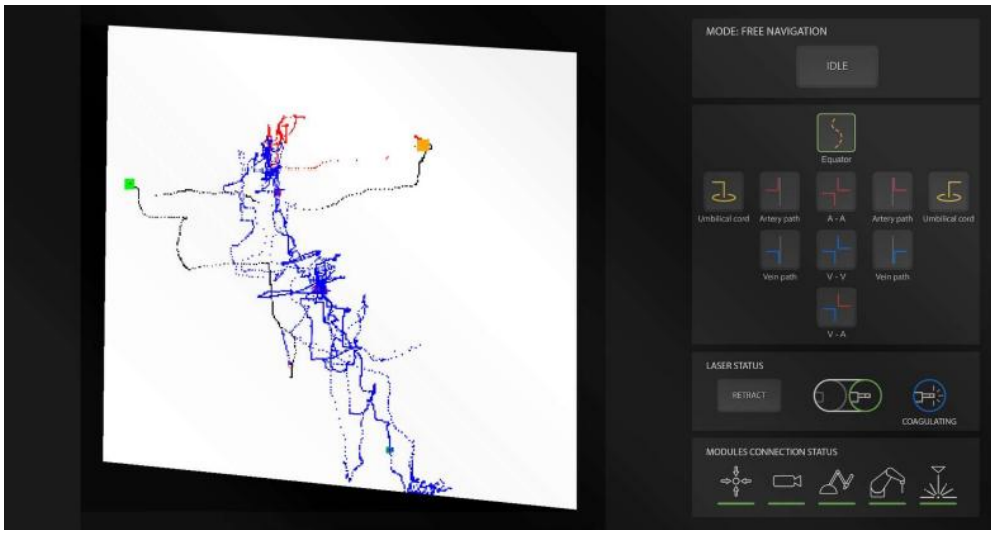


Figure 9. Interactive map and information and markers from UI.

In terms of communcation, the Master module sends the fetoscope movement orders. The module also sends the actions made by the user using the pedals and the interactive UI (e.g laser tool actions or emergency stop).

It receives the robot position and through this, the trajectory map is generated and shown in the UI. The information about the status of the central control and slave modules is sent to the Master module, which is shown in the UI. The module also receives force vectors to generate in the haptic device through some motors. This allows to represent tactile force on the user´s hand, which come from the forces exerted on the end effector of the robot, such as collisions. These forces make teleoperation more immersive.

2.1.3 Slave

It is the module that interacts with the patient/work environment, executing the instructions generated by the master.

This module counts also with some components:

- **Robot**: A robot with six degrees of freedom (Figure 10) is the component responsible of carrying out all the movements sent from the central control system.

Depending on the task to perform, the movements come from the surgeons controlling the Master device or from an automated process sent by the central system. The fetoscope is assembled as the end-effector of the robot. (Figure 11)

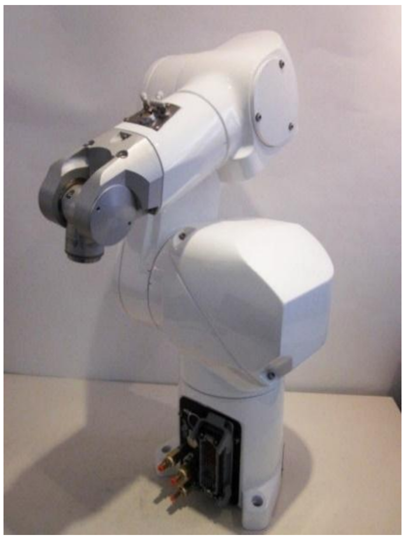


Figure 10. The robot used to test the global system.

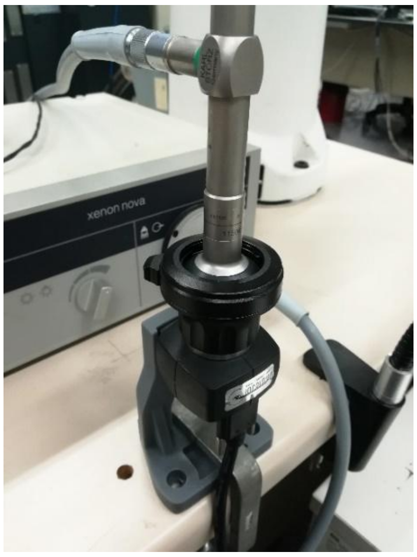


Figure 11. The fetoscope camera that sends the images to the user interface

- **Laser:** The laser (Figure 8 and Figure 12) is the tool used for the coagulation of the anastomosis points. It is inserted through one of the fetoscope´s ducts. An automatic system is available for the extraction and retraction of the laser.

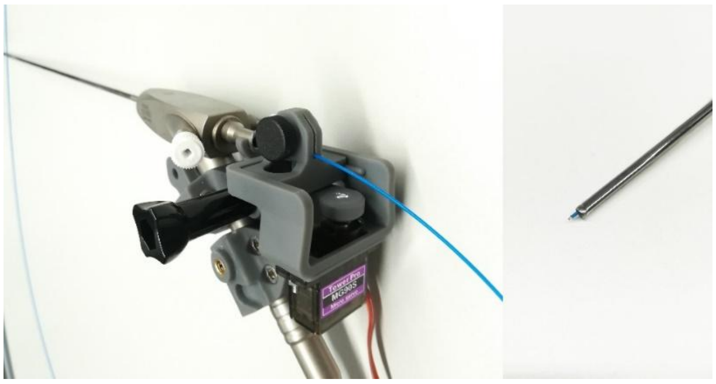


Figure 12. The laser tool to produce the coagulation of the anastomosis points and the extraction/retraction actuator

- **Compensator system for the abdominal region:** The trocar is the device used as an access channel to the workspace. It consists of a head with a membrane that prevents internal liquid from flowing out and a flexible tube through which the fetoscope is guided. To create the access route, a puncture is made from the outside and all the layers of tissue are crossed to access the uterus.

  There is friction between the tube and the fetoscope. This can cause the trocar to come out of the uterus and consequently, a new puncture is required. Therefore, dramatically increases the risk of rupture of the amniotic sac membranes.

  To solve this situation, a compensation system (Figure 13) has been implemented. This system is automated, compensating the movement of the fetoscope to keep the trocar fixed. If the fetoscope goes into the abdomen, the system performs the opposite movement (outward) and vice versa.

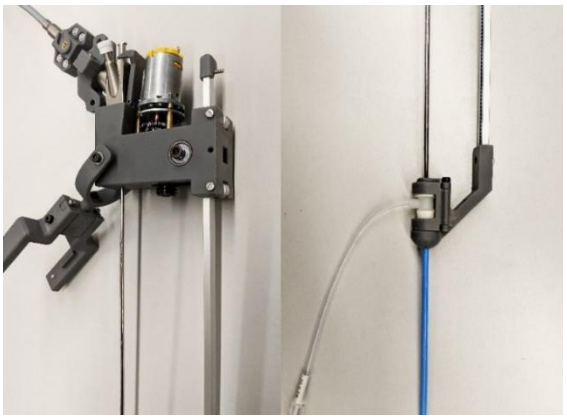


Figure 13. The compensator system for the trocar.

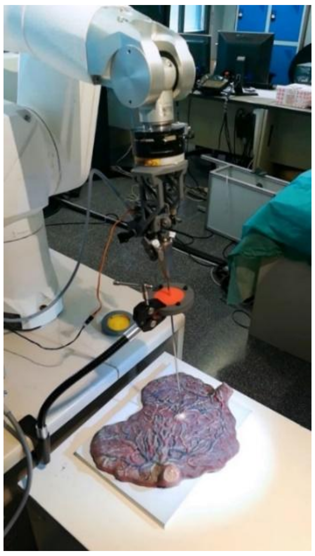


Figure 14. The whole Slave module.

Regarding communication, Slave module receives orders from the central control module: trocar compensation, robot movements, laser actions, etc. This module also sends information to the other modules: the location of the fetoscope, the laser status, the image coming from the fetoscope´s camera, etc.

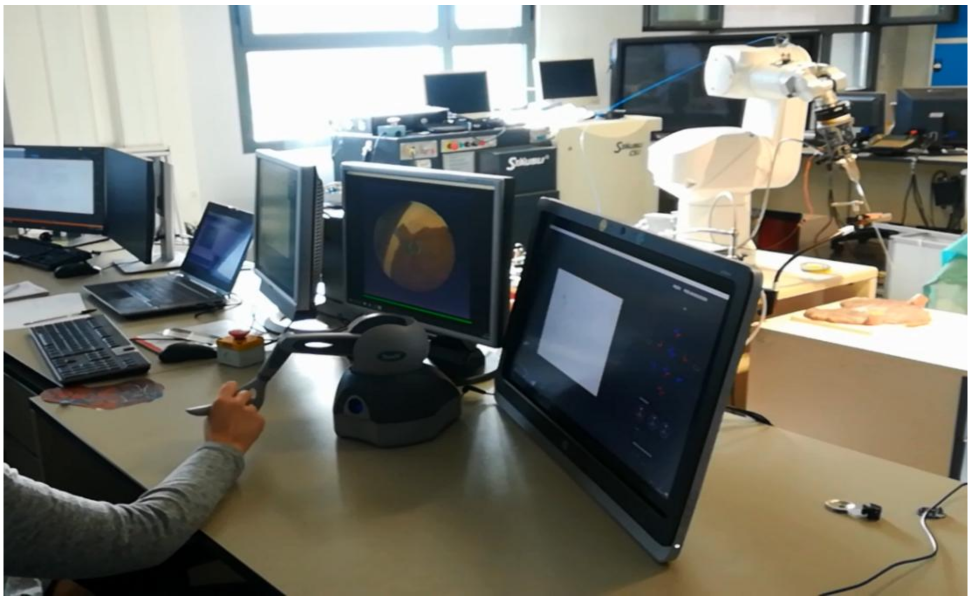


Figure 15. Foreground: Master station. Behind it, Slave system and placenta´s reproduction (silicone rubber)

3 Main Problem and Objectives

Fetal surgery presents several problems, such as limited accessibility and vision of the work environment, limited time due to anesthetic exposure to the mother and fetuses, risk of membranes rupture, etc.

The aim of this project is to address these problems by developing safe (collision-free) minimum distance paths between POI over the placenta´s surface for the surgical teleoperation platform. Safe paths are considered those that the surgeons describe during the exploration, while manipulating the fetoscope. The minimum distance or the fastest paths are found by the application of Dijkstra algorithm, which is modified in terms of fulfilling TTTS´s specific requirements. Both conditions (safety and economy of movement) must be accomplished at the same time, in order to determine the final path, which connects the initial and the goal points. This solution reduces the execution time of the surgery, increases precision and maximizes patient safety.

First, the main phases of surgery are introduced, corresponding to the phases and objectives of this project:

- **Phase 1: Exploration phase.** The surgeon explores the placenta´s surface and looks for the placenta´s equator, umbilical cords connections and POIs.
- **Phase 2: Location phase.** The anastomoses are located.
- **Phase 3: Coagulation phase.** The anastomoses are coagulated using the laser tool.
- **Phase 4: Revision phase.** It is checked the correct coagulation of the anastomoses.

Now, the phases of the algorithm execution are described and related with the surgery phases.

3.1 Map creation

This phase occurs during the phase 1 and phase 2 of the surgery. During the exploration and location phases, the UI generates a map with the trajectories of the robot´s movements.

The surgeon can choose what type of element or trajectory is manually focusing or following, e.g. blood vessel, free trajectory, equator zone or if he or she locates an anastomosis. Each anastamosis can be marked by its type to differentiate them.

All this finally generates an interactive map with all the information collected during the navigation. The map shows all the points that the robot has passed through during the exploration and the points that have been marked. This map is the main base of the algorithm execution.

3.2 Modification and use of the trajectory map

The objective of this phase is to modify/adapt the generated map for its use in the automatic trajectory generation using the Dijkstra algorithm. Once the map is obtained, it is necessary to perform certain modifications for using the map.

The map is generated by sampling the trajectories done by the robot during the manually exploration. Due to the sampling, the generated map contains sets of unconnected points. In order to solve the sampling gaps, it is created trajectories between the points to connect them. This is done by Computer Vision tools, specifically with morphological operations applied on the map.

Finally, it is also possible to connect two trajectories that are close enough (without reducing safety) in order to optimize even more the final path.

3.3 Automatic movement phase

In this phase, the surgeon is able to decide to which destination/goal point wants the robot to go. This phase occurs during the phase 3 and phase 4 of the surgery.

The objective of this phase is to perform an automatic movement of the robot, in a safety way, from one point to other; and thus, releasing the surgeons to lead the robot manually and reducing their workload.

It is considered that the paths followed by the surgeon´s movements, in the exploration and location phase, are safe. Among these possibilities, the robot must automatically choose the fastest or least cost path, i.e. the one with the shortest distance. To do this, the Dijkstra algorithm is used to find the path and then the robot receives a succession of points to follow this path.

Finally, the robot reaches the point. Once reached, the surgeons can launch the laser and coagulate the anastomosis if they are in the coagulation phase; or check the coagulation of the anastomosis if they are in the revision phase.

The robot can be stopped at any time by the surgeon, who has at his / her disposal both the emergency pedal and the emergency button that act immediately stopping and moving to a safety pose the robot and giving the possibility of returning to the navigation / exploration phase.

4 State of the Art

In the case of automation in medical robots, two main cases must be differentiated: rigid tissues and soft tissues. In rigid tissues, the application of pre-defined trajectories during surgery is easier than in soft tissues. Because of this, there are already existing systems used in surgery of rigid tissues; while in soft tissues, most of the systems are nowadays in research and development phase.

4.1 Rigid tissues

In rigid tissues, as orthopedic surgery, the automation is feasible, because they are difficult to deform and they remain practically always unchanged. This is the case of the skull, the hip or the spinal column.

One of the first methods used was fixing the robot in a well defined pose with respect the workspace. Examples like RoboDoc shown below have been implemented for some time and they are still in use.

Also, the 3D reconstruction is a common method used in this type of surgery. It is possible to make this 3D reconstruction of the surgical POI, and even to plan and simulate the surgery previously. The robot uses this planning in order to perform the surgery. This method can be used also in surgeries which need the human-robot co-manipulation where, for instance, the movement of the robot is passive and manually guided by the surgeon (Figure 17). It is possible to implement virtual fixtures like protection zones or surfaces, or even areas where the robot moves at a slower speed for safety reasons (Figure 16). In case the human makes some forces and surpass the speed limit or he / she tries to go to restricted areas, the robot can impose speed limit or even stop itself despite the human makes any force.

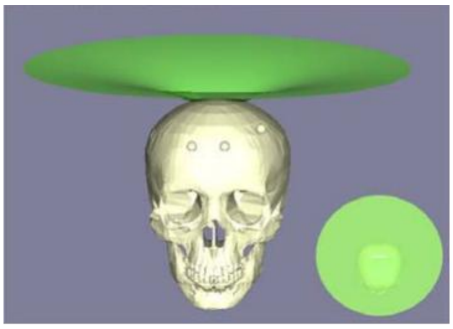


Figure 16. 3D reconstruction and virtual fixtures

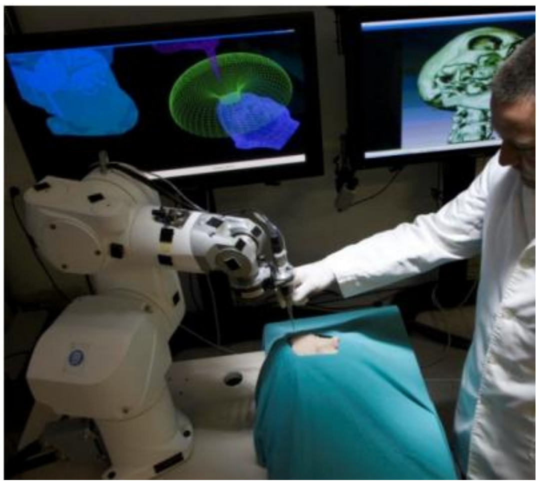


Figure 17. Manual guidance of the robot by the surgeon (Co-manipulation)

Stereotaxis refers to the possibility of fixing and immobilizing a specific part of the human body. The registration uses two types of data: the first type is taken during a preoperative by a scanner (Figure 18) while the second type is taken during the surgery. Comparing both data models and with the help of some reference points (Figure 19), the robot is able to perform pre-planned surgery automatically.

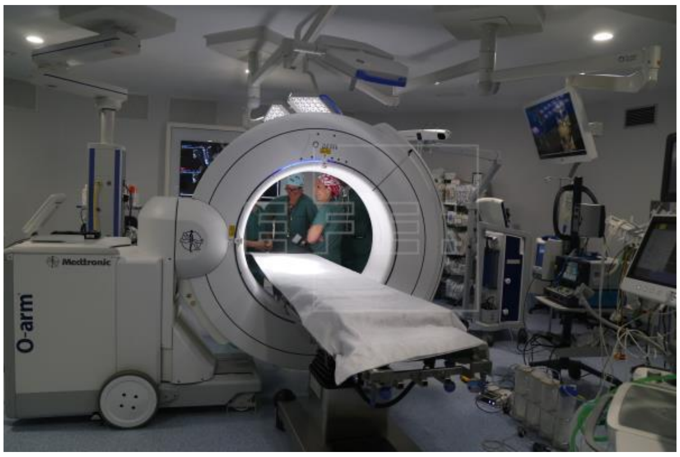


Figure 18. Scanner to take images

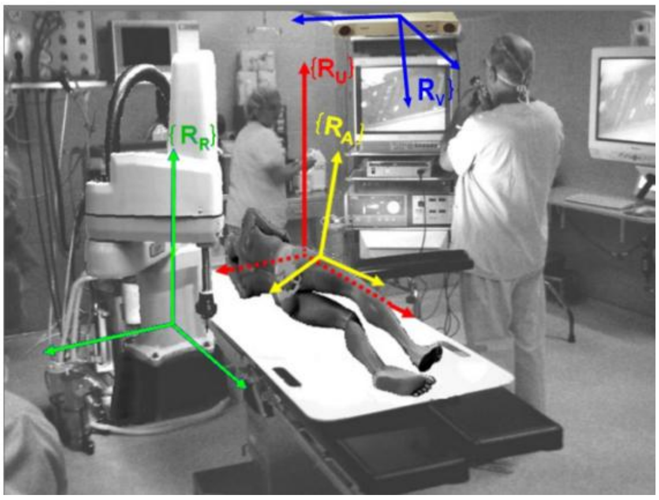


Figure 19. Reference systems during the surgery

Some examples of the current state of the automation on orthopedic surgery are:

- **RoboDoc**: Developed by Paul et al. Initially this system was used for a total hip replacement, although it was later modified and was also used in total knee replacements. It is now less used because of the controversy surrounding long-term clinical effectiveness.

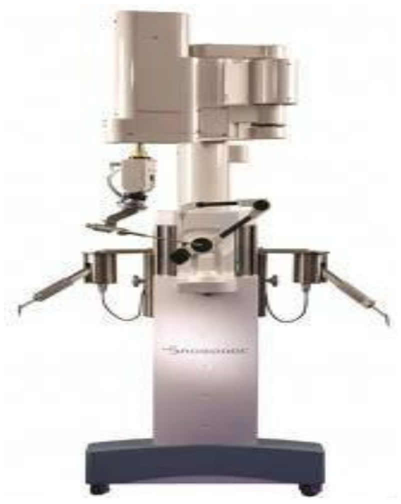
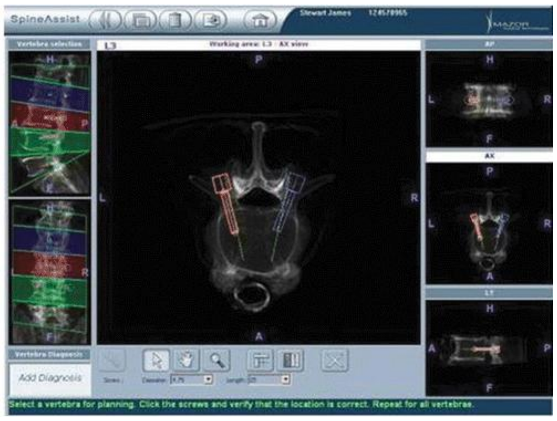


Figure 20. RoboDocc surgical system

- **SpineAssist:** Image-based guidance system which helps the surgeons, both in open and minimal invasive surgery, to screw insertion with an accuracy range of 1 mm.

During the preoperative it is done an image acquisition of the spine and vertebrae. With these images, the surgeon plans the surgery and the trajectories through which the screw will be inserted (Figure 21).



Select a vertebra for planning. Click the screws and verify that the location is correct. Repeat for all vertebrae.

Figure 21. Visual trajectory planning

During surgery, the robot is attached to the bone. Depending on the type of surgery, if it is open surgery SpineAssist is attached directly to the vertebra (Figure 22 left); if it is minimal invasive, another device called minimally invasive Hover-T frame is attached to the iliac crest. (Figure 22 right).

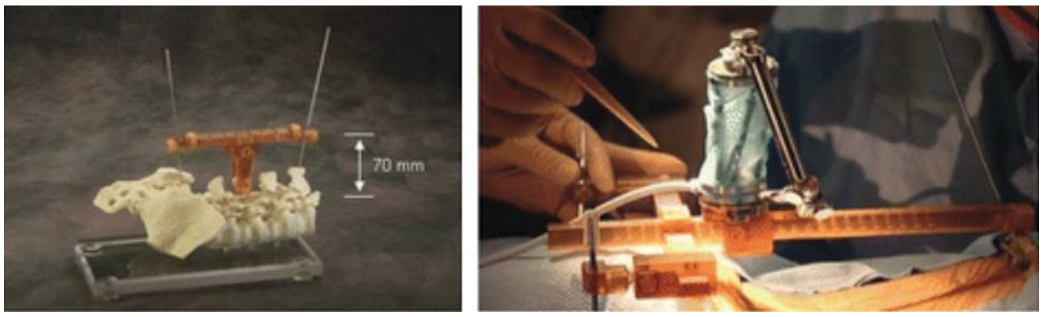


Figure 22. Attachments in open or minimal invasive surgery

Once the SpineAssist is attached, there is a second image acquisition. These images are matched with the preoperative images under the supervision of the surgeon, who verifies and approves the result.

At the last step, the robot moves automatically until reaching the correct position to match the trajectories of the cannulas with the trajectories decided in the preoperative. Finally, the surgeon drills and inserts the crews through these cannulas (Figure 23).

Figure 23. Drilling

- **Navigators:** Navigators are essential systems in most medical robotic systems. Their main role is played during the pre-operative phase, where a registration takes place.

  The objective is to help the surgeon during the surgery, transforming and comparing the data obtained during the preoperative with those of the current surgery. The navigation provides a global reference system on the patient's anatomy in order to determine the relative position of each key component. During surgery, these positions help the surgeon to study and plan in advance the procedure he / she wants to follow.

  One example using navigators is the projection of virtual images over the patient. Technology based on augmented reality merges the obtained images with the real through a semi-transparent screen. This produces an illusion to the surgeon so that it seems that they can see both the inside and outside of the patient's body part. The head of the surgeon is tracked so that, with their relative position, the transformation of the images can be produced, gaining even more immersion for the surgeon.

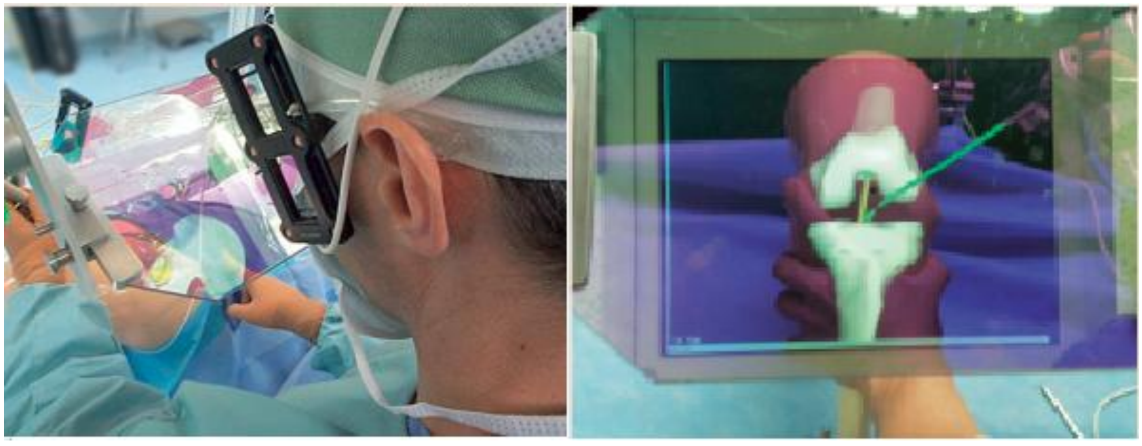


Figure 24. Augmented reality surgery

4.2 Soft tissues

In contrast to rigid tissues, soft tissues present a greater difficulty when implementing robotic and automatic assistance in surgery.

In MIS, there is not a fixed and precise relation between pre-operative data and real surgical scenario, due to the air insufflations inside the organs, the soft tissues are deformed and their shape change, producing several problems for automatic surgery such as shape changes, locations changes, … which produces a lack of precision.

Problems are not only due to the interaction between surgical tool and tissues. It should be taken into account that during surgery, the patient is sedated or anesthetized. Therefore, apart from the patient's own breathing, spasms and abrupt movements may also occur, which also causes changes in the tissues and may even lead to collisions with the surgical tool.

For this reason, pre-planned or assisted robotic in soft tissues surgery is more difficult to implement and they are most of them still in research and development phase.

Some examples about researches are:

- **Robotic suture assistance:** It is based on robotic assistance during the suture of tissues and organs. During the experiments, two main phases are differentiated: interactive auto-guidance and gesture recognition.

  In the interactive auto-guidance phase, the trajectories during the suturing are calculated. This is done by the use of Artificial Potential Fields algorithm, which is based in creating virtual attractive and repulsive forces to guide the robot to the goal point (attractive forces) and avoid collisions with obstacles or undesired regions (repulsive forces).

  In gesture recognition, the robot is able to adapt its forces to the forces made by the human, assisting them during the collaborative suturing. For instance, during the needle insertion, the robot applies a constant force while during the knotting; the robot applies variable forces to make correctly the knots.

- **Optimal planning for minimally invasive surgical robots:** In this case, the trajectory planning previous to the intervention is divided in two steps:

The first step is called port placement and it refers to the incision site for minimally invasive access (Figure 25). This step relies on patient´s modeling that is transformed into an optimization problem, where visibility and dexterity during the surgery serve as cost functions to determine the optimal access point for this surgery.

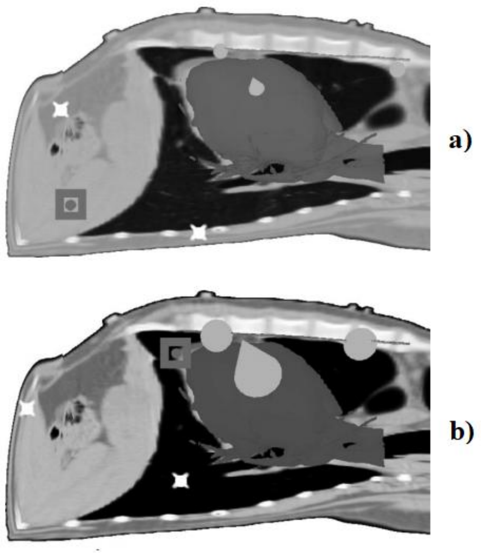


Figure 25. Choice of the targets in port placement. Crossed point represent surgical instruments and squares represent endoscope port placement

The second step refers to the robot position planning. Its objective is guaranteeing a collision free operation during the intervention, by assigning flexibility to the robot to choose the path at the beginning of the surgery, but always ensuring the fixed position of the minimally invasive access route and port.

Figure 25 b) shows the suggested ports placement which leaded to successful surgery in a dog. However, Figure 25 a) shows suggested port placement but with targets too small that do not reflect the needed area to operate and finally it resulted in internal collisions between the dog´s hearth and the surgical tools.

4.2.1 Placental MIS

It is a special case where, despite the fact that there are soft materials, the work environment configuration and distribution allows for a certain planning. The fetuses and the placenta are the main obstacles in the work environment. The placenta is an organ that has a surface with a constant or regular curvature. On the other hand, fetuses make very little movement and furthermore, it is allowed certain collisions between the tool´s body (never with the tip) and the limbs of the fetuses in order so that these limbs can be moved without danger.

During the surgery, it is used the fetoscope manually to explore and perform the necessary interventions. In the followed trajectories using the fetoscope, it is avoided any obstacle. As the environment is reasonably under control, these trajectories can be used as collision-free vias for implementing robotic systems and move their end-effectors over these trajectories. Therefore, in this particular environment with deformable elements it is possible to apply path planning, but taking into account all the restrictions; in specific, the restriction of only moving through those collision-free trajectories.

5 Project Structure

In this section, it is described the structure and main steps followed during the development of the project. The time frame of the project is from February 11th 2019 to June 22th 2019. There is a main organization structure, which is minimally updated during the project due to some different problems that could not be foreseen before starting the project.

- **Planning the problem:** This project is based on implementing a robotic system for assisting in the TTTS surgery. One of the main needs is the automatic generation of trajectories. For a fast code development it is used MATLAB R2017a in order to obtain results in a quick manner. Then, there is the possibility of further programming and integration in control module, using C++. During the project, access to the teleoperation platform is available for saving and using real trajectories for tests.

- **Map generation:** First it is planned how to extract the necessary information for the algorithm computation. This information needs to be treatable and mathematically interpretable, to later allow the application of algorithms for path planning. The information is represented by graphs, where nodes are the initial and final points of the automatic movement, the terminal points, the path intersections points and the POIs. The edges are those paths connecting the nodes.

  At the beginning, simple and synthetic maps are created. These maps are used to plan and extract the necessary information. It is verified that the way in which the information is represented in a graph is correct. Then, Dijkstra algorithm is chosen as fast path algorithm, and it is checked that could be a feasible solution. The complexity of synthetic maps increases along the project until reaching the point where real maps are used. These maps have some drawbacks that should be solved before using them. The maps are a set of discrete points where some of them are not connected. These points are connected by using Computer Vision tools, specifically, morphological operations as erosion, dilation, etc. After applying this solution, it is checked that the graphs obtained from these real maps become useful for the objective of the project.

- **Trajectory optimization:** The surgeons, using the fetoscope during the surgery, perform trajectories which are not optimal. These manual movements define intersected trajectories or regions where these paths are close. From here, among these trajectories, it has to be found the optimal path. Then, a second optimization is applied, where the paths which are closer than a safety distance, defined by the surgeon, can be connected. In this way, it can be generated even more optimal paths.

- **Final results and tests on real maps:** Validated the algorithm for simple paths, it is collected the data from real trajectories where it is applied the algorithm developed in this project. The results are satisfactory and the algorithm can find efficient, safe and fast paths.

- **Written report:** During the project is being documented all the important parts when results are obtained. At the end of the project, a report is written where all this documentation is collected, explaining step by step the context and development of this project as well as the current state of similar projects.

6 Project Development

The objective during this development is to create and algorithm which finds optimal paths. However, this kind of algorithms has the problem that they can require long computation times. Therefore, in this specific case, it is studied and developed an algorithm that looks for the optimal paths, but also accomplishing certain computation time ranges in order to be compatible with the performance and the time spent in the surgery.

It is done a development process, where initially it is proposed simple and easy maps to check the algorithm operation. During the development, the algorithm evolves in order to optimize its computation time and, at the same time, it is created and inserted more complex maps.

When using certain maps and algorithms, sometimes the computation time arises excessively, and this forces to reformulate the algorithms and the solution in order to achieve more efficiency. This was complicated to foresee a priori, therefore this issues are discovered and fixed during the development of the project.

All the development is done in Matlab. Matlab is a good tool for rapid solution development for mathematical problems, but, in contrary, it is not too optimal in calculation and computation times. The use of this tool makes that the computation times are not optimal at all, but finally, this algorithm will be implemented in C++, where the computation and calculation times are more optimal. Therefore, using C++ the time results will improve significantly.

6.1 First approaches

This section describes the approaches that emerged as the project progressed in the complexity of the maps until using real maps.

6.1.1 Easy virtual scenarios implementations

The goal in this phase is to test the implementation of the automation, including the Dijkstra algorithm, over easy generated maps or scenarios. The first thing that was done was the definition of the main elements and the creation of virtual easy scenarios using Matlab.

At this point, one of the first and main objectives is how to represent the data obtained from the map. The trajectory map and the points that form it are used and transformed into a graph. This graph is formed by nodes and edges.

Nodes are those trajectory points that can be reached from more than two different paths or, in other words, the points with more than two outgoing paths.

On the other hand, edges are those paths which connects the nodes. If the robot is in any other point that is not a node, it has one-direction path to continue as it comes from already visited points and it must go through the non-visited points. Figure 26 shows both examples.

In addition to the points defined in Figure 26, there two more node types:

- **Terminal points:** Those points that constitute a termination, that is, that coming from a visited point and arriving at this terminal point, there is no possibility to continue anywhere.

- **The initial and final points:** These are the point where the robot is at the moment when the automatic movement begins (initial point), and the desired point to reach by this automatic movement (final point).

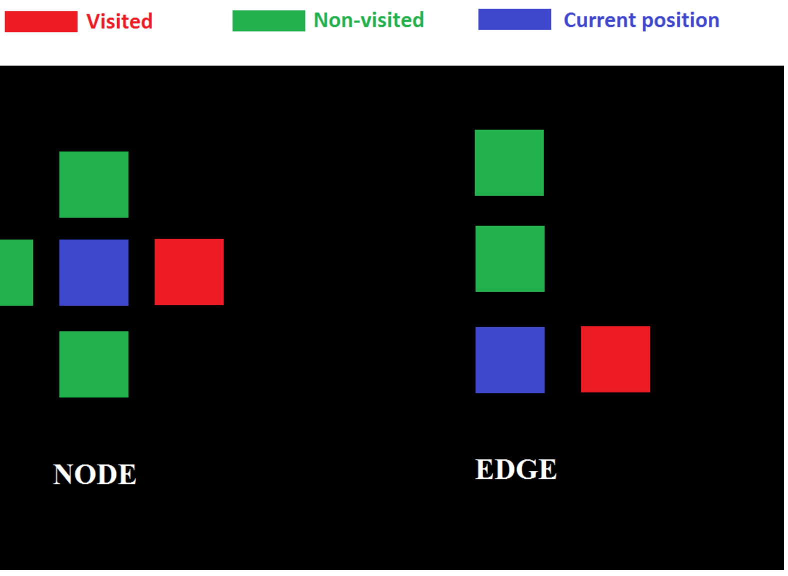


Figure 26. Graphical definition of nodes and edges.

Once defined nodes and edges in this particular problem, it is needed to know the distances between each node and its connected nodes. These distances correspond to the edges costs. The responsible of taking and saving these costs is the cost matrix.

The cost matrix is used as input for the Dijkstra algorithm. This matrix has the dimensions equivalent to the number of nodes in rows and columns. Each element in this matrix represents important information about the nodes concerned, which are those indicated by the column number and the row number:

The element of the cost matrix is a zero value, when both nodes (row node and column node) are indirectly connected or non-connected. Indirectly connection means that they are connected, but it does not exist an edge that connects both nodes. Non-connected means that: there is no possibility of reaching one node from the other through the allowed paths. Zero value is also obtained when the row and the column refers to the same node.

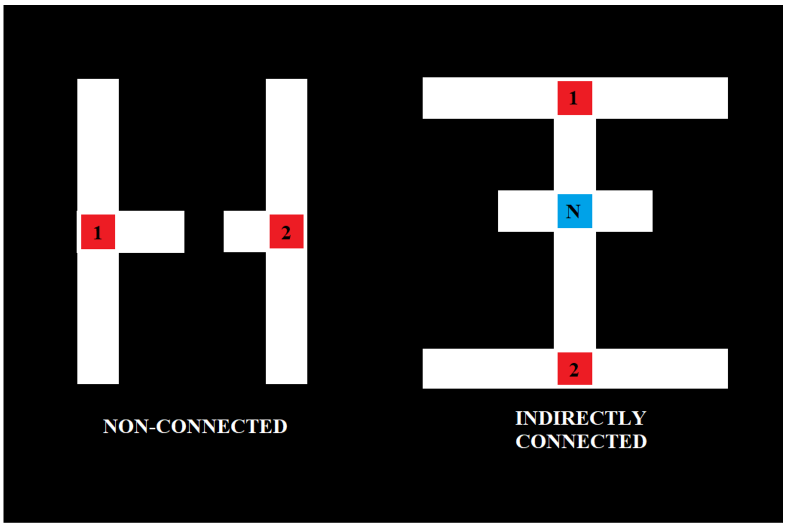


Figure 27. Non-connected and indirectly connected nodes examples

A non-zero value of an element of the cost matrix indicates that both nodes (row and column) are directly connected. In this case, the value reflects the edge cost which separates both nodes.

In the case of this project, as the edge costs that connect nodes are the same in any direction, the cost matrix is diagonal symmetric. Table 1 shows an example of a cost matrix, based on this project.

NODES	NODE 1	NODE 2	NODE 3	NODE 4	NODE 5
NODE 1	0	0	55,66	0	0
NODE 2	0	0	157,62	1	1
NODE 3	55,66	157,62	0	0	0
NODE 4	0	1	0	0	0
NODE 5	0	1	0	0	0

Table 1. Cost matrix example

In the case of synthetic easy scenarios, binary matrices are used. Pixels with value 1 indicate the points of the path the robot has gone through during the exploration phase, and pixels with 0 value are the points where the robot has not passed through.

For a better understanding let´s use a binary symmetric matrix of 50x50 dimensions (Figure 28). In this simple case, the basic concepts, as the graph extraction and fast path planning, are applied to see how they work, how fast they are, and what are the most probable and critical problems that can be found.

Finally, map and data extraction is done, Dijkstra algorithm is applied and tests are performed. This map and other similarly easy without symmetry are used to check if the objectives are achieved and, therefore, the fastest and safest path is obtained.

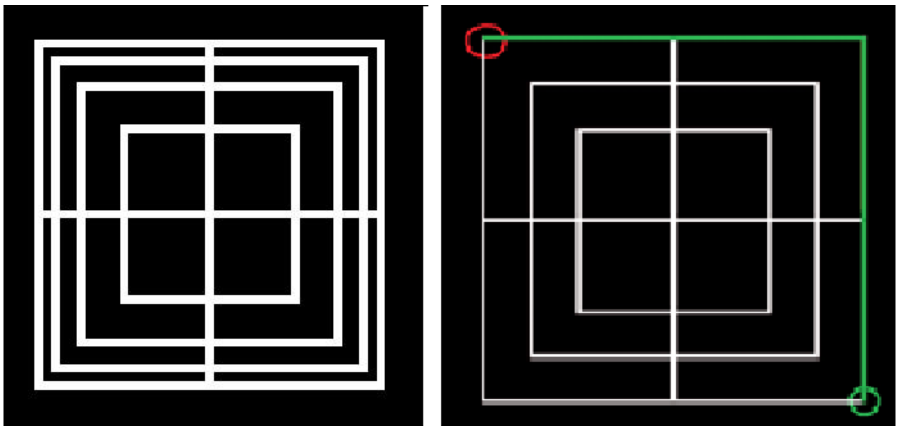


Figure 28. The first and easiest map developed.

Functions used during this phase: *map_creation*, *calc_saltos*, *num_nodos*, *tracking* and *dijkstra.*

6.1.2 Optimization in more complex virtual scenarios

The next step during the development is to apply and optimize the algorithm in complex scenarios. In order to achieve this objective, it is necessary to create more complex maps (Figure 29). Once applied the algorithm on the complex maps, it is demonstrated that the first approaches were not valid and the easier maps allowed testing less solutions.

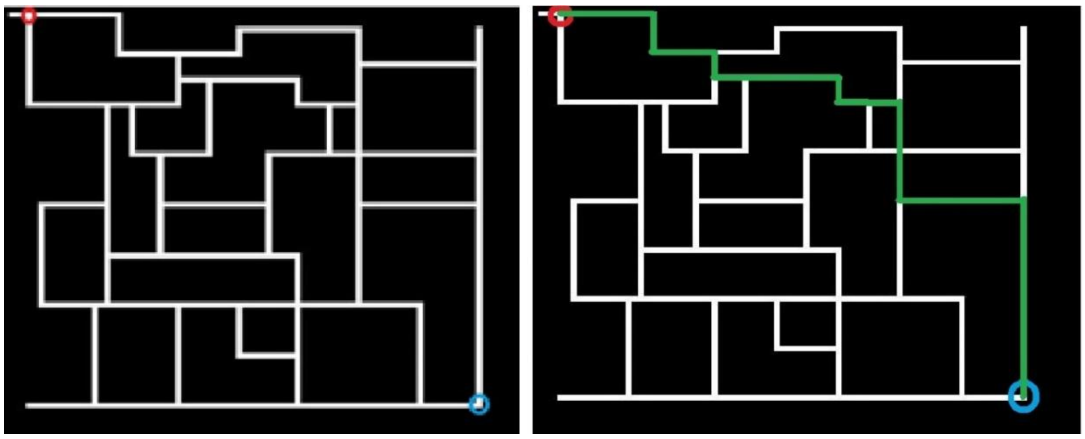


Figure 29. Complex map used for the development of the code.

In this phase, one of the key points of the project is also developed, the creation of intermediate paths that allows to connect other initial paths in a safety manner. The objective is to locate certain map regions surrounded by initial trajectories. These regions are small enough in order to consider them as safe zones (visited) in the final solution.

The dimensions of these zones can be modified in agreement with the surgeons ensuring patient´s safety. For this purpose, a variable limit is defined, which can be changed at any time, defining the regions allowed to be considered as safe or visited.

Once the limit is set, all pixels forming the zones are safe and considered as visited, so the Dijkstra algorithm can take them into account for the calculation of the fastest path.

First Figure 30.1 shows the original map and Figure 29.2 those regions, which have been filled, considered safe and visited. Then, Figure 30.3, shows how some intermediate paths appear inside these areas. Finally, as it can be seen in Figure 30.4, the robot can move freely through these zones without any restrictions and the optimal path can pass through any pixel inside these zones.

However, there is a reason for keeping the visual intermediate paths instead of the full region. Although it does not affect the movement of the robot over these areas, these intermediate paths are used in the nodes identification phase. In this manner, Dijkstra algorithm is able to use more nodes to compute easier the final path.

Comparing results in the same map but with and without the application of the intermediate zones, it can be observed that in Figure 29.2 the fastest path varies with respect to Figure 30.4. This is due to the fact that, having the possibility of the intermediate zones, the cost is lower through them and therefore the new path is chosen.

Due to the possible movement of the robot through these zones, it is necessary to take into account these areas when applying the Dijkstra algorithm. For this reason, the nodes definition and the edges calculation are updated.

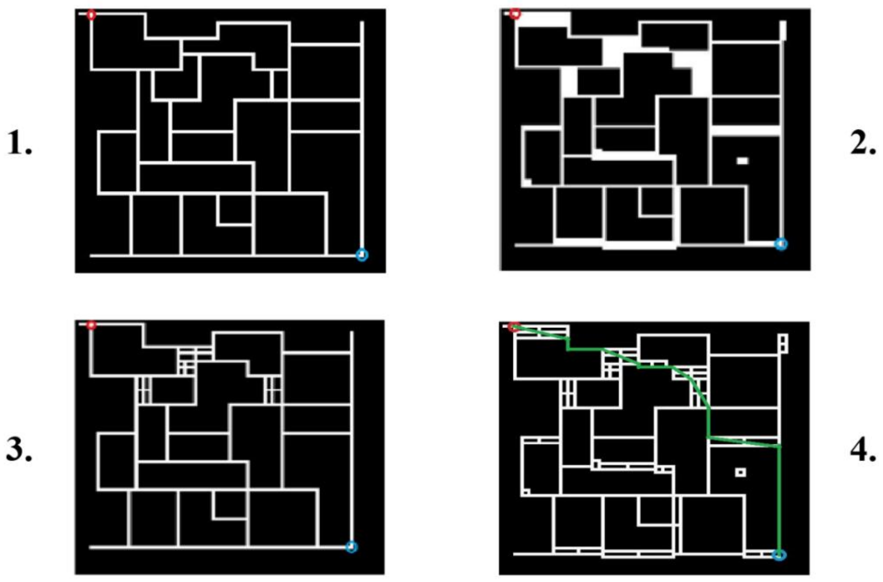


Figure 30. Procedure to get the intermediate paths from the gaps limited by their dimensions.

Functions used during this phase: *map_creation, calc_saltos, num_nodos, tracking, dijkstra, map_form, path_connection* and *imclose*.

6.1.3 Real exploration scenario

Once the complexity is high enough and everything works correctly, it is time to proceed to the last step: the implementation of developed solution in real maps (Figure 31), obtained from a real exploration using the robotic system.

First tests over these maps failed due to the computation time. This is produced for different reasons:

- The dimensions of the map in this case are much larger than those that have been virtually performed in Matlab. As the dimension increase, the size of the paths and the number of points is higher also, so the computation slows down.

- Problems appear due to sampling during mapping (Figure 32). The sampling frequency is low in comparison to the movement of the robot. Therefore, incomplete paths are found, which should be solved due to the intermediate zones seen above or similar methods. Fulfilling certain objectives requires that these unconnected points must be joined. However, many of these points do not connect through intermediate paths, as they exceed the limit. Increasing the limit to produce their link, extra-connections are produced in areas that are not convenient and this does not ensure the patient´s safety.

- As the map is larger and more intricate and due to many more intermediate paths between nearby areas, the number of nodes is too large. Therefore the information collected is excessive and the total computation for the calculation of edges costs is too slow.

All this is tried to solve in the most effective and general possible way, but after a long time and many tests, it is decided that the initial proposed solution is difficult to implement in the real maps.

Finally, there is a reformulation of the code. It aims to reach the same solution but in an easier and more general way, making some previous implementations disappear and some new ones come into development. In order to do this, real exploration maps are used directly and all the configurations and methods are tested using these maps.

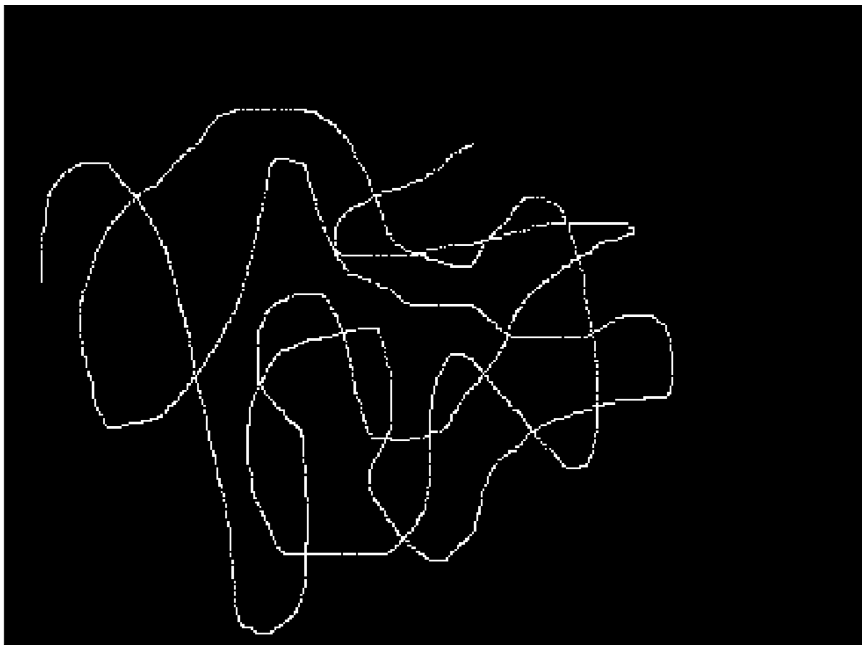


Figure 31. Initial map from a real exploration.

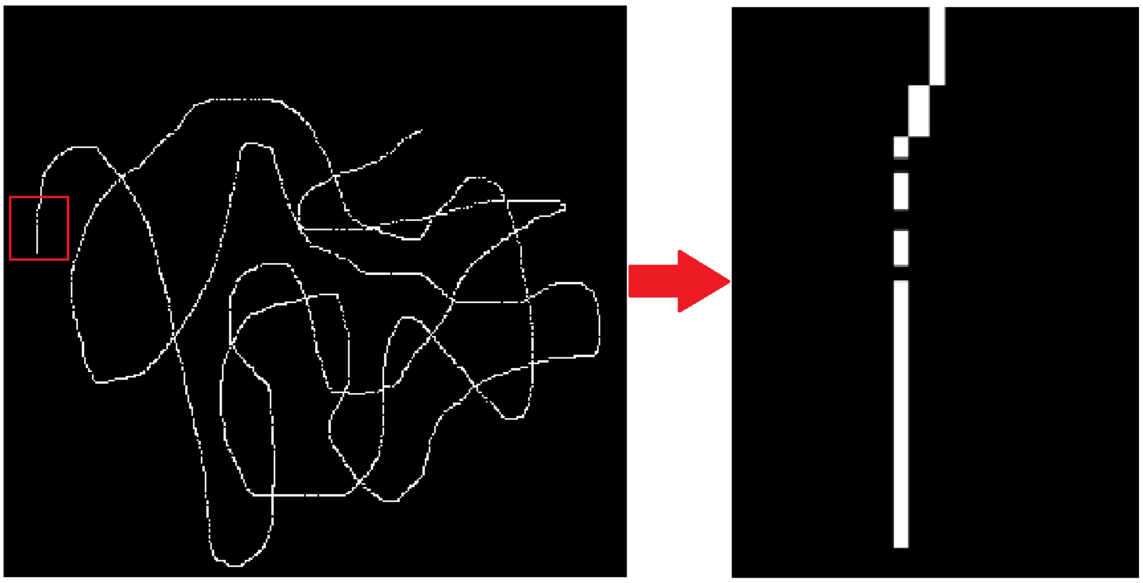


Figure 32. Errors that occur during the sampling.

Functions used during this phase: *map_creation, calc_saltos, num_nodos, tracking, dijkstra, map_form, path_connection, imclose* and *imerode*.

6.2 Other approaches

6.2.1 Geodesic distance

Geodesic distance is the minimum distance between two points using the paths of the graphs. In other words, only the pixels with a value of 1 are valid points that can be used for the distance computation. The regions where the image is true or 1 represent valid regions that can be navigated. The regions where the image is false or 0 represent restricted regions that cannot be navigated. For each pixel of value true or 1, geodesic distance function, *bwdistgeodesic* (Appendices, Matlab functions) gives to those pixels the shortest restricted distance to the seed (goal point), only in case that it is possible to connect them by regions of value 1.

The seed is the point from where the distances are calculated or the true pixel whose distance is zero. This means that every pixel takes the value of the shortest distance from the pixel itself to the seed.

For the edges costs calculation, each node is defined as a seed at some point, and restricted distances of all the other direct connected nodes indicate edges costs.

Geodesic distance is used firstly as a better alternative for distance computation than the previous method (tracking). Before that, the distance computation needed to navigate all the trajectories point by point. In contrast, geodesic distance method performs the calculation of the distance between each pixel and the desired point in one execution.

However, during the use of this function, three main problems are identified:

- **Connected Nodes:** Geodesic distance allows to know the restricted distance from a particular node to any other pixel true on the map. Therefore, the distance between nodes can be known, or in other words, the edges costs. But the problem is due to the absence of information using the geodesic distance function.

  This function performs the geodesic distance from every point to the desired point (the seed), which has distance value 0. The problem occurs when the information received by the outputs of the function do not allow to know what are the nodes that are directly or indirectly connected to the seed or desired node. In other words, the geodesic distance function gives the distance of every point that can be connected to the seed by paths of value 1, but there is no possibility to know which points are directly connected and which are not. This produces that the cost matrix is generated in a wrong way, including distance values in some places where the value should be zero, as those nodes are not directly connected. Seeing this wrong cost matrix, it could be assumed that some nodes are directly connected with other nodes, when in fact, it is not true. Finally, this would lead to a wrong Dijkstra algorithm computation and to a wrong solution.

In this case, all the elements of the cost matrix are different from zero, no matter if they are directly or indirectly connected. For Dijkstra algorithm this means that all nodes are directly connected to each other, and this is not true. In Figure 33, it can be seen this case in nodes 2 (direct connected) and 4 (indirect connected).

This creates an incorrect cost matrix, and thus, incorrect information given to the Dijkstra algorithm.

This problem was solved using *bwdistgeodesic* in a sequentially way. The solution consists of taking the value of the smallest geodesic distance as this ensures that the node at that distance is direct connected. This is because if they weren't connected, the geodesic distance couldn't be calculated and; as there is not any other node closer to the seed (minimum geodesic distance) there is nothing that makes the connection indirect.

The function *bwdistgeodesic* is applied as many times as outgoing paths exist from the desired node. To do this, the outgoing paths are eliminated sequentially leaving only one unaltered. Then, it is calculated the geodesic distance from all the other nodes, and the minimum of them is the node directly connected through that path. The loop continues until make this with all the outgoing paths.

At the end, the distances indicate the cost of the edges, and also ensure that the nodes at those distances are directly connected to the initial node (Figure 35).

In the following images the colours represent:

- **Green:** Initial node or seed.
- **Red:** Nodes surrounding the initial node or seed.
- **Yellow:** Path followed to obtain the edge cost
- **White:** Unvisited points

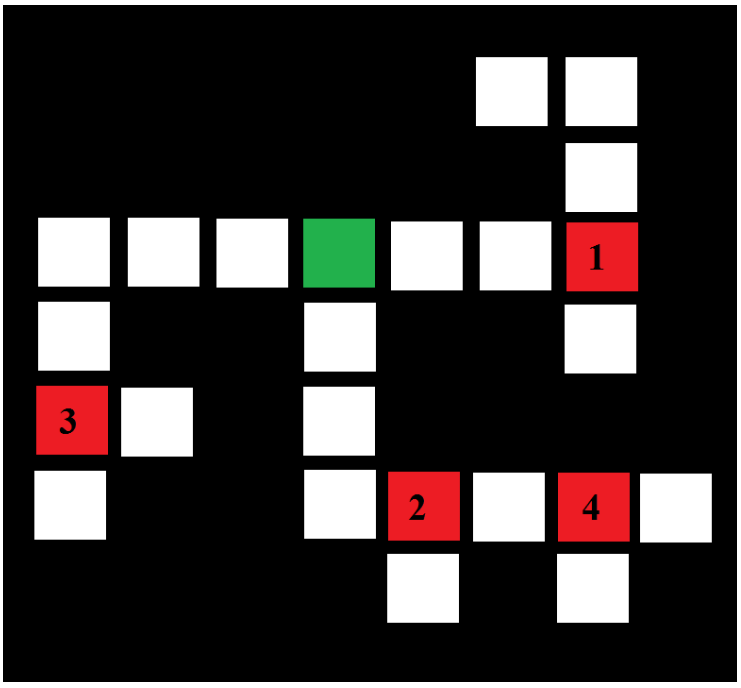


Figure 33. An example of direct and indirect connected nodes

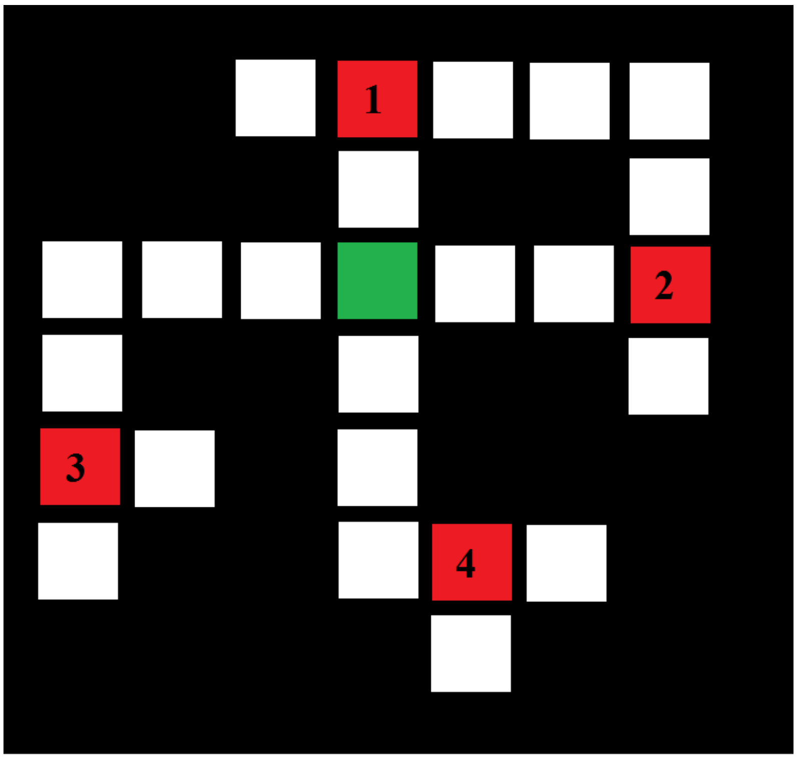


Figure 34. Initial map using geodesic distance

Node 1: direct connection

Node 2: indirect connection

Node 3: not connected

Node 4: not connected

Min. Geo. Dist = 2

Node 1: indirect connection

Node 2: direct connection

Node 3: not connected

Node 4: not connected

Min. Geo. Dist = 3

Node 1: not connected

Node 2: not connected

Node 3: not connected

Node 4: direct connection

Min. Geo. Dist = 4

Node 1: not connected

Node 2: not connected

Node 3: direct connection

Node 4: not connected

Min. Geo. Dist = 5

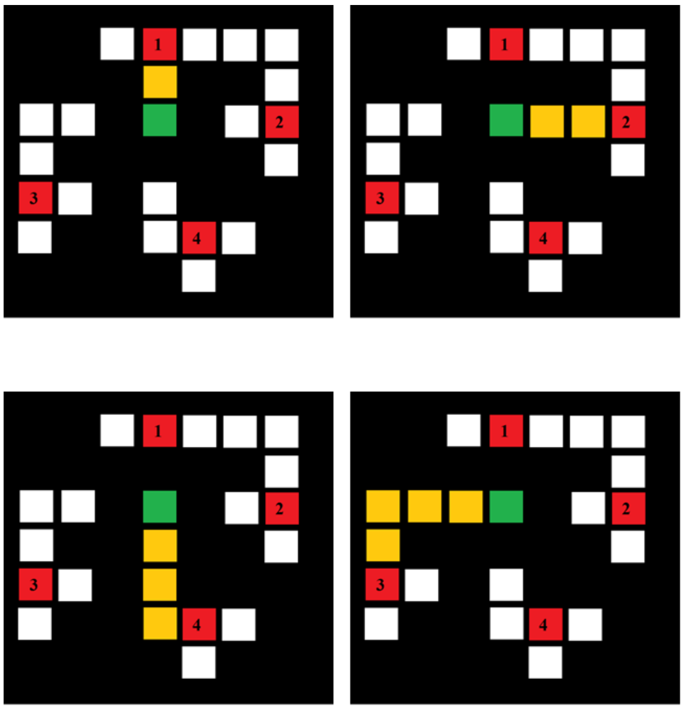


Figure 35. An example of how is cost matrix calculated with geodesic distance

- **Computation time:** Although *bwdistgeodesic* is a function that seems to perform the desired operation in less computational time than *tracking* (point-by-point previous method), the times that is needed to use this function increase during the development of the solution. First, it is necessary to implement this function for each node. Later, not only one time in each node, but the function must be executed as many times as outgoing paths the node has. Therefore, despite its less computational time, the function is called many times, which makes negligible the time difference compare with *tracking*.

- **Path points:** There is a last problem, which has to do with the lack of information again.

  The main goal of this project is to provide the robot with the necessary information in order that the robot moves from one point to other through the fastest and safest path. For this, it is not only necessary to calculate that path, but also provide the robot with the succession of points that form that path.

  Using geodesic distance, it is obtained the edges costs and thereby, the costs matrix to compute the Dijkstra algorithm. However, in order to know the points that form the path calculated by Dijkstra, geodesic distance is not sufficient.

  Hence paths must be followed point by point even if *bwdistgeodesic* is used. This means that the computation time of the code, is not even the same, but is longer than using *tracking*.

Due, especially to the last two problems, *bwdistgeodesic* is discarded. Its use implies large computational costs.

6.3 Final approach

After all the previous proposals, the final result is reached. This solution consists of previously used and new methods. The objective is to perform the automatic movement of the robot through the safest and costless path using the trajectories made during the exploration phase, manually guided by the surgeon. Now it is described the steps followed to reach this final solution

6.3.1 Security margins and filling of not sampled points

In this phase the necessary operations are made on the map to solve the sampling problems. The objective is to modify the map, using Computer Vision and keeping the original trajectories as unaltered as possible. In this way, it is possible to obtain the same real map extracted from the UI, but without gaps and without alterations that complicate the subsequent search for the safest and costless path.

At the first moment, morphological operations were used to solve the sampling problems. Using functions that produce erosion, dilation and morphological closure; it was tried to find the less expensive and best way to perform the desired map. After some tests, some problems were found using these types of function. The main problem was that it is needed to define the shape with which this operation is performed. Once applied these morphological operations, using different defined shapes in Matlab, the result was not enough good in order to the subsequent operations. Below it is shown some examples.

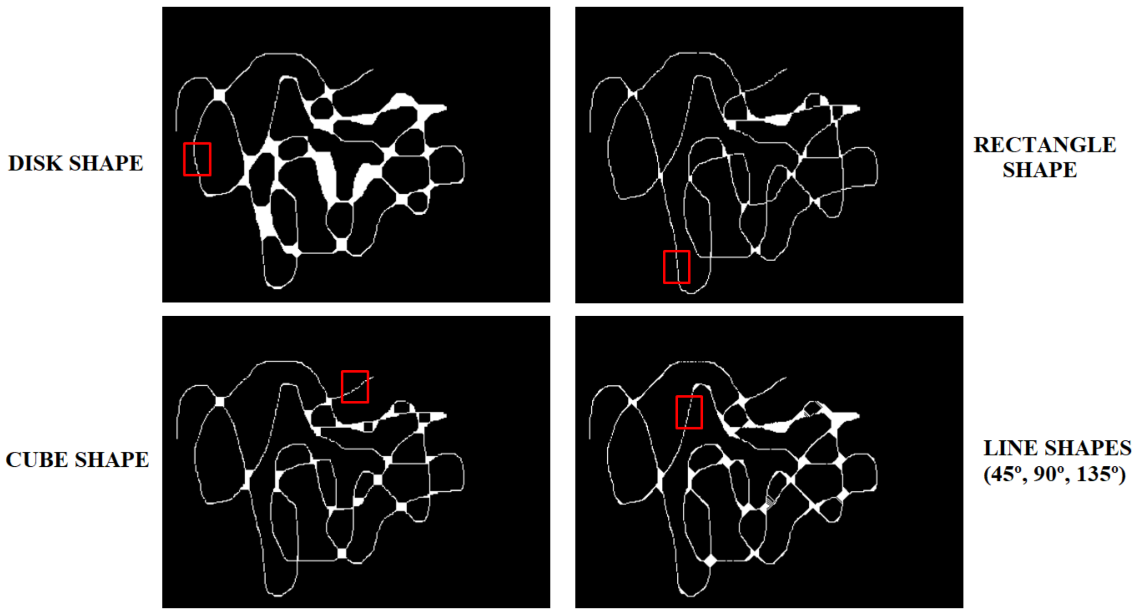


Figure 36. Different morphological operations applied on the map

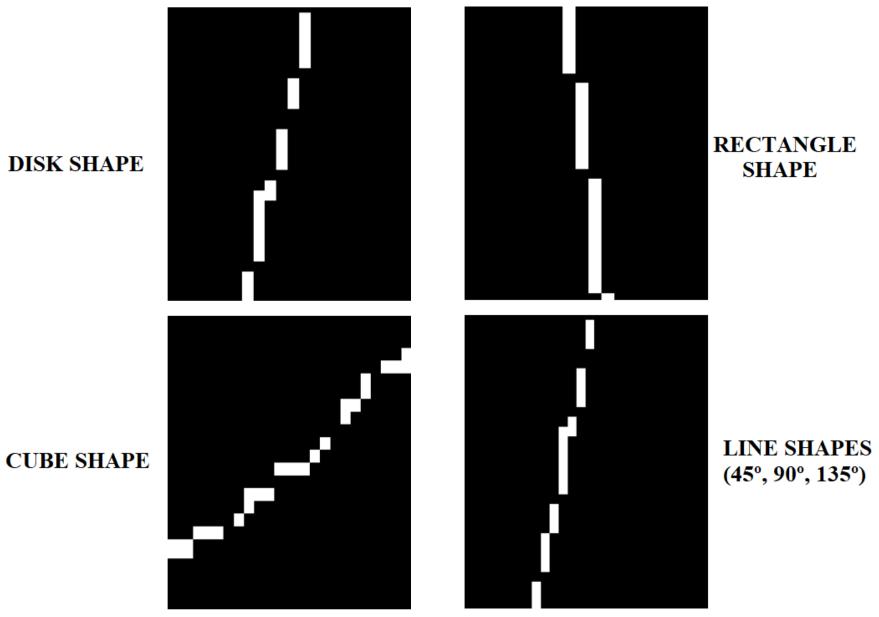
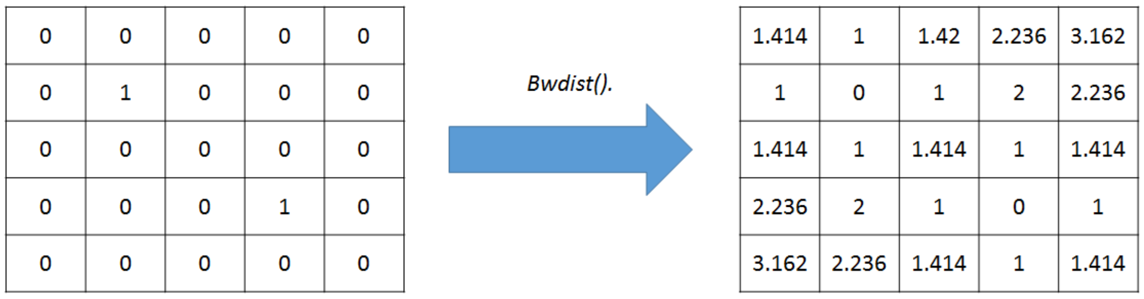


Figure 37. Zoom in errors marked in Figure 36

As it can be seen, the sampling errors are not solved at all. During this phase some different shapes for morphological operations were applied in order to reach to a final and good result but unsuccessfully.

Finally, another morphological operation is applied. It is based on the function *bwdist*. This function assigns to each pixel the Euclidean distance to the closest non-zero value pixel. In this way, every previous pixel with value 0 or false obtains a Euclidean distance value, while the pixels with value 1 or true, obtains the value of 0 (Figure 38).

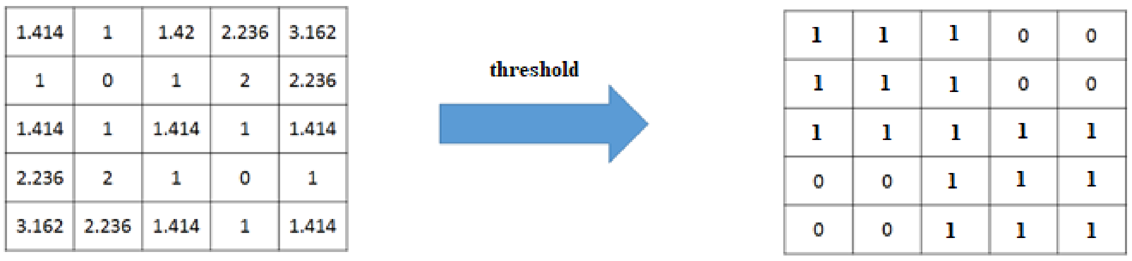


0	0	0	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	1	0
0	0	0	0	0

1.414	1	1.42	2.236	3.162
1	0	1	2	2.236
1.414	1	1.414	1	1.414
2.236	2	1	0	1
3.162	2.236	1.414	1	1.414

Figure 38. Euclidean distance to closest non-zero pixel

Once this function is applied, it is defined a threshold. The objective of this threshold is similar to the limit of intermediate paths method seen above. After applying the *bwdist* function, the threshold is used in order to assign value 1 to all the pixels under this limit, and all the other pixels return to zero value. This method makes that the trajectories on the map grow in all directions and the sampling gaps that are smaller than the limit are filled (Figure 42). In Figure 39 and Figure 40 it is shown the previous example with a threshold of 1.5 and how its application works.



1.414	1	1.42	2.236	3.162
1	0	1	2	2.236
1.414	1	1.414	1	1.414
2.236	2	1	0	1
3.162	2.236	1.414	1	1.414

1	1	1	0	0
1	1	1	0	0
1	1	1	1	1
0	0	1	1	1
0	0	1	1	1

Figure 39. Performance of the threshold method

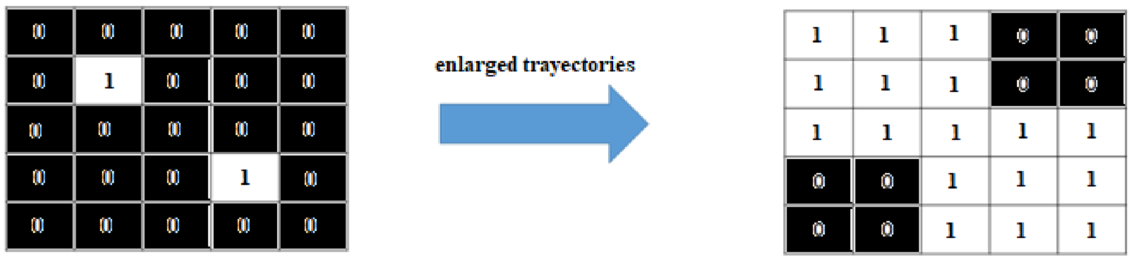


Figure 40. Visual example of enlarged trajectories

In order to ensure the highest possible level of safety, the threshold can be changed at any moment to avoid covering dangerous areas or places where is possible to find an obstacle. For this reason, security margins established by the surgeon must be taken into account while defining the threshold´s value.

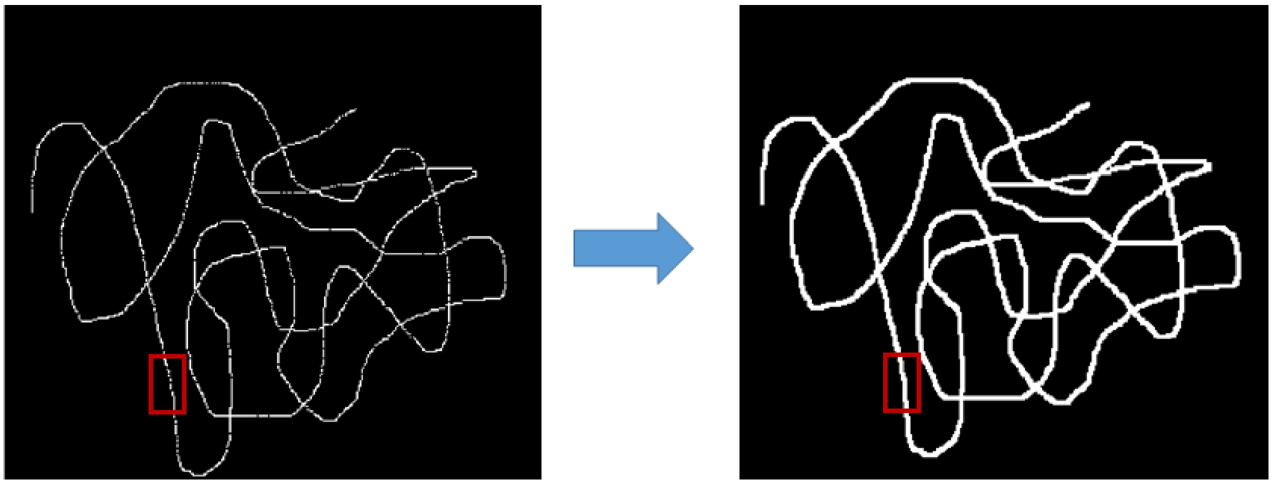


Figure 41. Comparison between the real initial map and the enlarged trajectory map

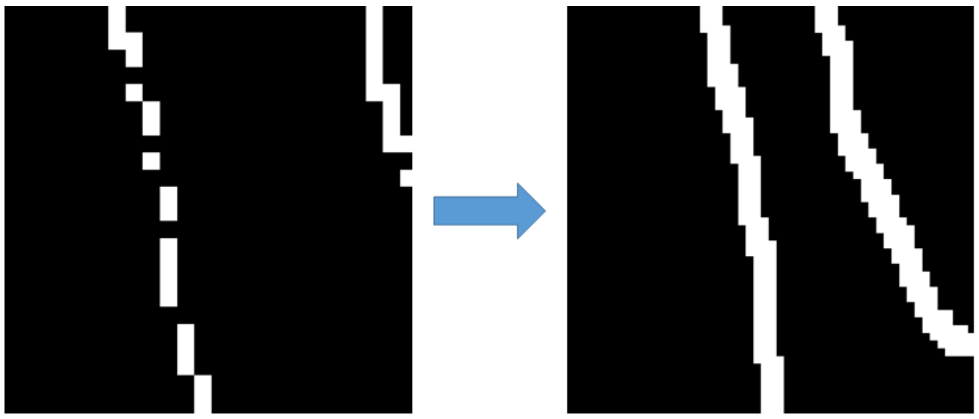


Figure 42. Comparing zoom in on the initial map and enlarged trajectories map

As it is shown in the enlarged trajectories allow filling the gaps during the sampling.

6.3.2 Trajectories thinning and central path

Once the trayectories have been enlarged and all the gaps due to the sampling are filled, the trajectories must be tracked and the edges' costs calculated. However, tracking and calculating costs is not simple when trajectories are enlarged. Different ways to do it are evaluated and tested. Some of them are not useful and others are difficult to implement.

Then, it is decided to modify more the map, in order to achieve easy to track trajectories. The objective is to slim trajectories up to one pixel wide for a facilitated tracking of these. To do this, the Fast Parallel Algorithm for Thinning Digital Patterns by Zhang and Suen is applied [15].

This algorithm performs, through some iteration, a thinning of different digital patterns. The final result is the "*skeleton*" of that patter of unitary thickness. End points and pixel connectivity are preserved. More in deep, this algorithm consists of two subiterations: one aims at deleting the south and east contour points while the other aims at deleting north and west boundary points. Below it is shown the example over the real map (Figure 43).

In this case, the thinning algorithm makes on the map that, after its application, only the central path of the enlarged trajectories remains. As the enlargement is symmetrical as it is applied a distance threshold for the pixels around, this unitary central path is the same than the trajectories in the initial map. The only difference is that this unitary central path is complete and without gaps.

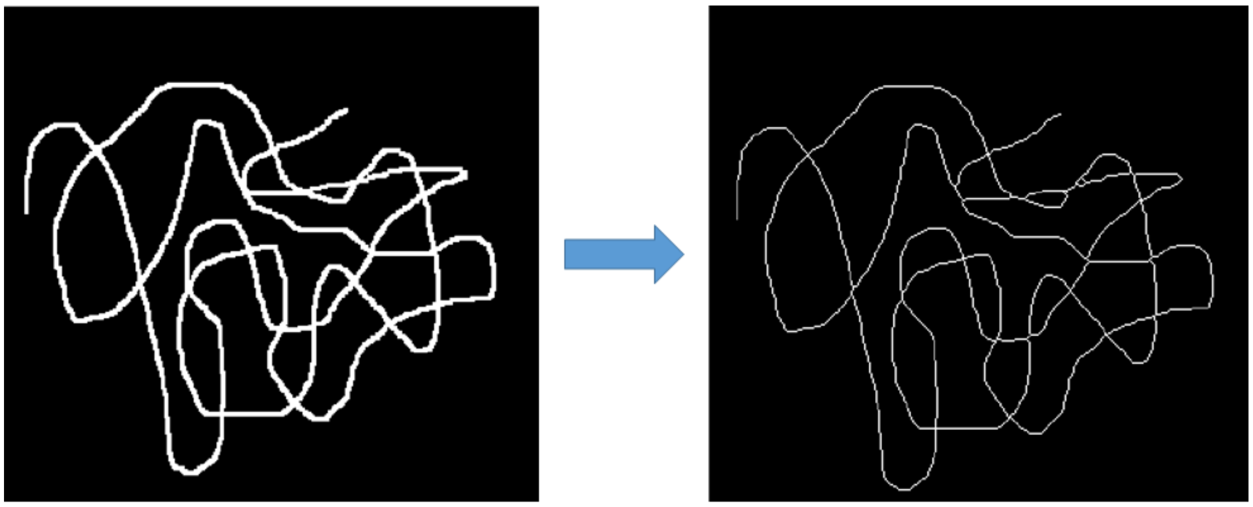


Figure 43. Comparison between enlarged trajectories map and map after applying thinning algorithm

As it can be seen in Figure 43, applying the Fast Thinning Algorithm, it is obtained the same wideness that in the initial map, but this time, without gaps or failures due to the sampling. This is the last morphological operation and modification over the map, as the final result is good enough. Now the main objectives of this project are aimed at. These objectives consist of obtaining the edges cost for the later application of the Dijkstra algorithm, in order to know the costless path between any two points within the trajectories of value 1 or true.

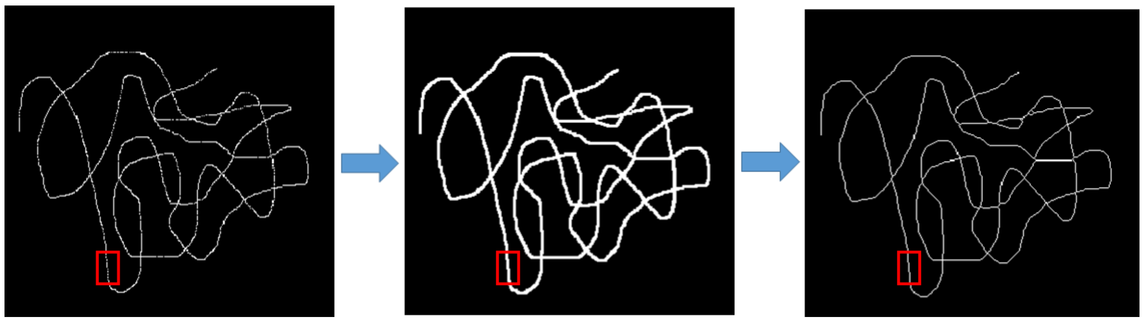


Figure 44. Progression of the map during the modifications applied

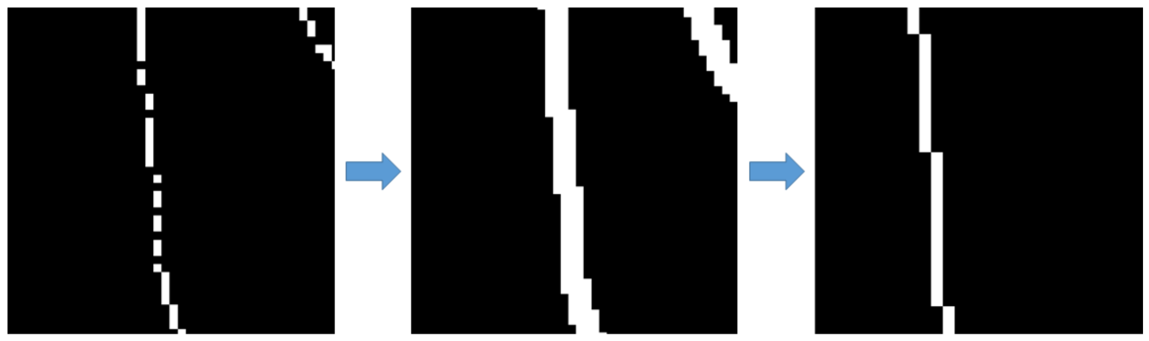


Figure 45. Progression of the map (zoom in) during the modifications applied

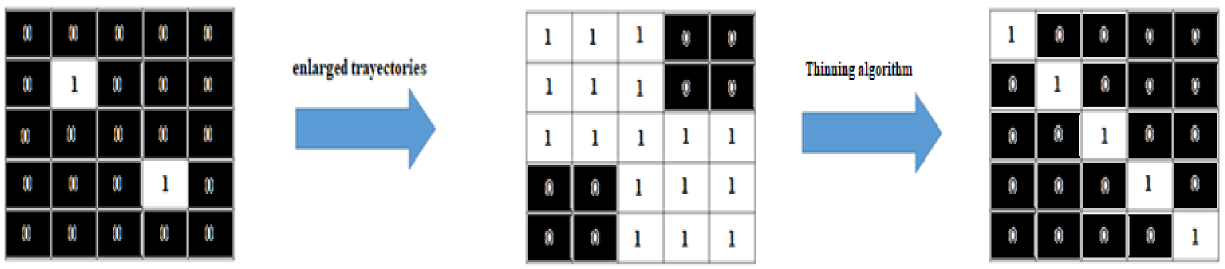


Figure 46. Visual example of the map´s progression

6.3.3 Trajectory tracking and edges costs

In this phase the objective is to track the trajectories, obtaining the edges costs and the succession of points between nodes for the computation of the final path. In order to do this, the trajectories are tracked point by point. As it has discussed above, geodesic distance method seemed to be a good alternative to avoid point-by-point tracking, but due to some problems, geodesic distance method is discarded.

As the map during this phase is complete and with unitary width, it is easier to follow the trajectories. The method used of this purpose is the following:

- First, the nodes are found and determined. To do this, *calc_saltos* function is used (Appendices, Functions developed) in order to find those points with more than two possibilities and defined them as nodes. (Figure 47)

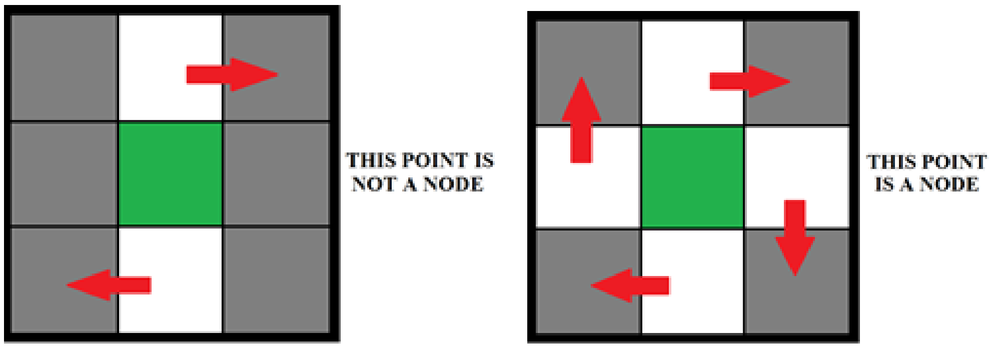


Figure 47. Node definition examples

- Once the nodes are found, the trajectories following process starts. The function used is *tracking*. This function takes all the nodes and their position. The function starts with an initial node and finishes finding other directly connected node.

  The function takes the initial node, and calculates the number of all the outgoing paths from that node. After that, the trajectory following begins (Figure 48).

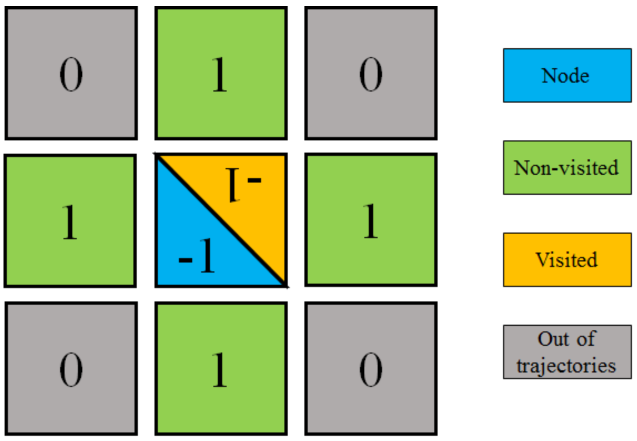


Figure 48. Node with non-visited outgoing paths

The method looks for the values of pixels around that are equal to 1. For this, a matrix of 3x3 is defined with the central point as the current visited pixel and the rest are its neighbors. To avoid coming back during the tracking of the trajectories, the central pixel takes the value of -1, specifying that it is already visited. Once is located a neighbor of value 1, the matrix moves and the central pixel become this located neighbor. In the case that the central pixel is not a node, one of its neighbors has the value of 1 and the other has the value of -1. This indicates that only one direction is possible (Figure 49).

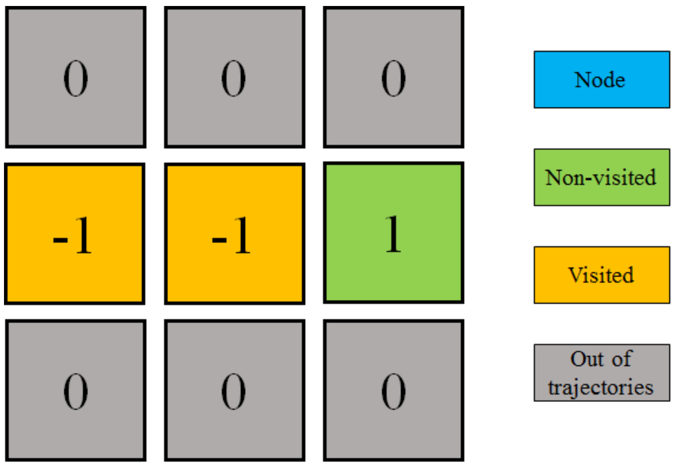


Figure 49. Only one direction is possible if the central pixel is not a node

This is repeated in the same way until reaching a node, which is the direct connected node through that path. This directly connected node is not marked as visited, in order to allow to reach it through other paths. The nodes are only marked as visited when they become the initial node (Figure 50).

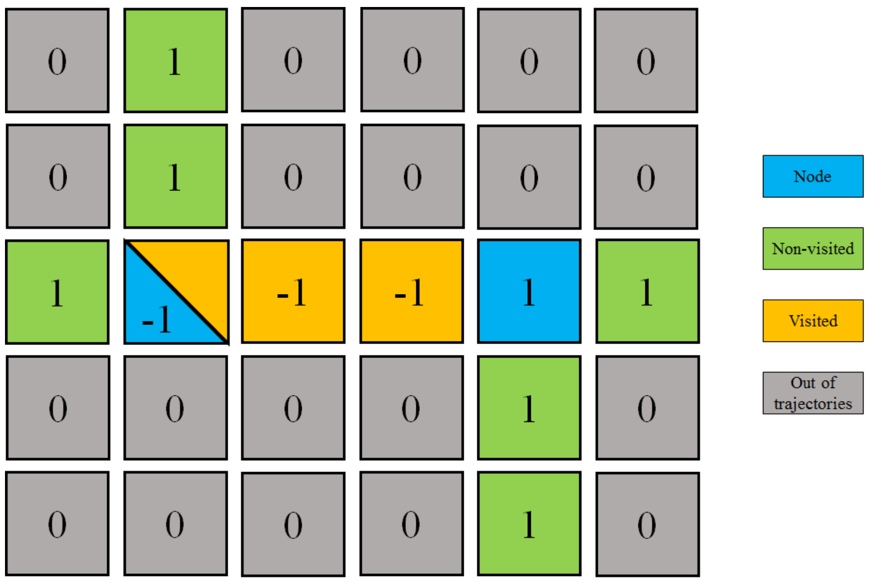


Figure 50. Example of looking for the directly connected node. Node found is not marked as visited

This complete process is repeated, coming to the initial node again and checking the outgoing paths. When the central pixel is the initial node, and every neighbor has the value of -1 or 0, that means that every outgoing path has been covered (Figure 51) and the matrix moves to the next node, which becomes the initial node

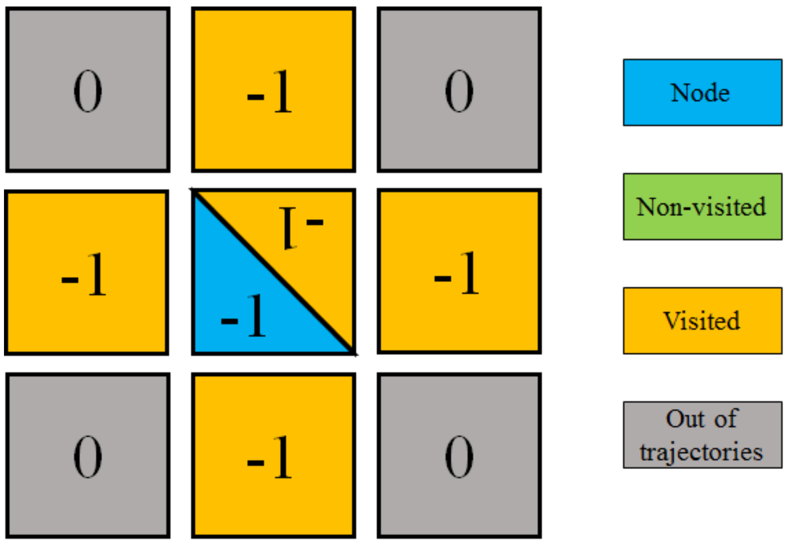


Figure 51. All the outgoing paths have been visited

All nodes are initial nodes in their respective iteration, following the numerical order. The number of outgoing paths is the number of iterations done using a unique initial node. The total number of iterations is the sum of all outgoing paths of every node.

- During the tracking of the trajectories, it is obtained the edges costs between nodes.

The cost is calculated and increases during the trajectory. The cost is the Euclidean distance measured along the trajectory. When the matrix displacement is vertical or horizontal, the cost increases in 1 unit; while if the displacement is diagonal, the cost increases in 1.42. When other node is reached and the central pixel becomes the initial node again, the cost is initialized to 0. For instance, in Figure 50, the edges cost that connects both nodes is 3.

- While tracking the trajectories, it is also computed the succession of points that compose the path between nodes. These successions of points, at the end, compose the final trajectory found by Dijkstra algorithm. All the points are given to the robot to follow this final trajectory.

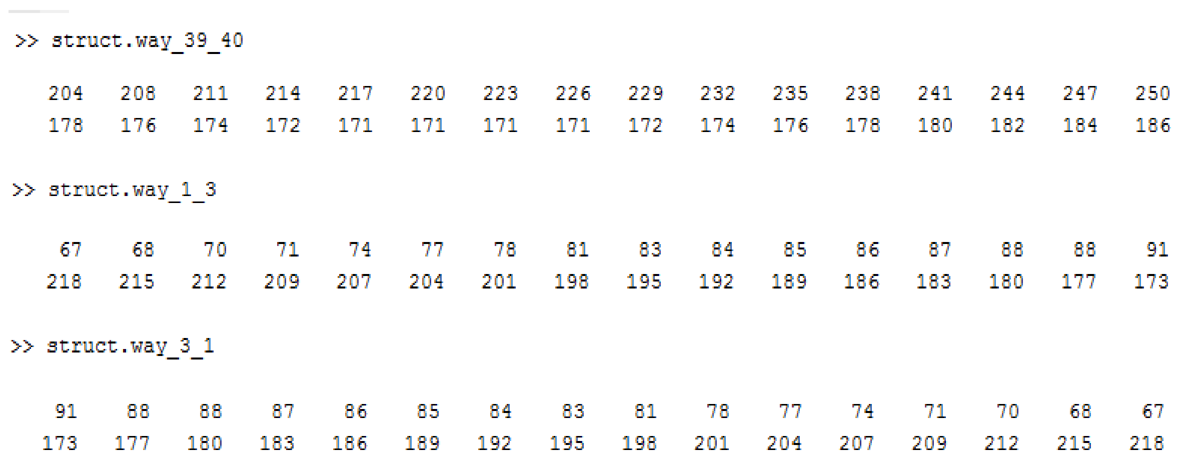
To do this, it is defined an array where points are saved, and a counter that establishes when a trajectory point is saved.

Then, the first point saved is the initial node. Once the tracking begins, for each movement of the matrix the counter increases in one unit. When the counter reaches its limit, it is initialized to zero and the position of the current central pixel of the matrix is saved in the array. The counter limit can be varied according to requirements and precision. The process continues until finding a node, which is also saved in the array.

Both initial and final points, that are both nodes, are always saved in the array independently of the counter value.

Finally, the array is saved in a structure variable, where each array takes the name *way_X_Y*, being *X* and *Y* the numbers of the connected nodes. *X* is the initial node and Y is the final node. The array and the counter are initialized to zero before starting the tracking of any other trajectory.

In Figure 52, it can be seen three arrays of points. It can be observed that, in the last two cases, the array is the same but in opposite directions, as they are the same nodes. The only difference is which is the initial node.



```
>> struct.way_39_40

   204   208   211   214   217   220   223   226   229   232   235   238   241   244   247   250
   178   176   174   172   171   171   171   171   172   174   176   178   180   182   184   186

>> struct.way_1_3

    67    68    70    71    74    77    78    81    83    84    85    86    87    88    88    91
   218   215   212   209   207   204   201   198   195   192   189   186   183   180   177   173

>> struct.way_3_1

    91    88    88    87    86    85    84    83    81    78    77    74    71    70    68    67
   173   177   180   183   186   189   192   195   198   201   204   207   209   212   215   218
```

Figure 52. Example of arrays cointaining points between nodes

6.3.4 Fast path algorithms (Dijkstra)

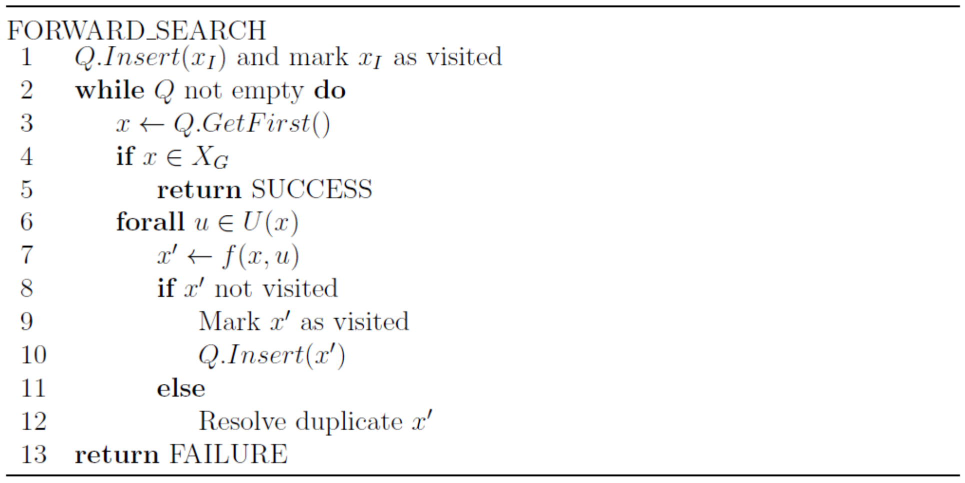
After all the trajectories have been covered, the edges costs are calculated and the path points are saved, the fast path algorithm is applied. In this case, the algorithm used is Dijkstra.

Dijksrta algorithm is a tool used to find optimal and feasible plans in order to get a single solution which provides the costless path in a graph from the initial to the goal point. It is a special form of dynamic programming.

The way to work of this algorithm is as follows. It is defined a priority queue *Q*, which is sorted to a function called *cost-to-come*. For each node, there is *C*(x)* which defines the *optimal cost-to-come* from the initial to the current node. This is obtained summing all the edges costs from the initial to the current node over all the possible existing paths. Then, *C*(x)* is the least cumulative cost of all of them. If it is not known that it is the *optimal cost-*

to-come, it remains as *C(x)*. The *cost-to-come* is incrementally computed, that means that although a path has been found, if the current node *x* is still in *Q*, it is possible that the new path, which is being calculated, could be the really optimal path. In that case, the *cost-to-come* in the current node should be substituted by this new computed one.

Once all the possible paths have been checked from the initial to the current node and all the possible edges leaving have been explored, the current node *x* becomes dead (the node is removed from the queue). For the next node *(x')* from *Q*, the value must be optimal, as the previous nodes are dead, and the only way to have a lower total cost is going through another node in *Q*, but these already have higher costs, so it is known that *x* cannot be reached with a lower cost. Then the edges leaving *x* are explored and the induction continuous until get the final *optimal cost-to-come* from the initial to the goal or final node. (LaValle and M., S. (2006). *Planning algorithms*. Illinois.)



```
FORWARD_SEARCH
Q.Insert($x_I$) and mark $x_I$ as visited
while $Q$ not empty do
    $x \leftarrow Q.GetFirst()$
    if $x \in X_G$
        return SUCCESS
    forall $u \in U(x)$
        $x' \leftarrow f(x, u)$
        if $x'$ not visited
            Mark $x'$ as visited
            $Q.Insert(x')$
        else
            Resolve duplicate $x'$
return FAILURE
```

Figure 53. Pseudo-code definition of the Forward Search algorithm.

In Figure 53, pseudo-code of the previous explained method called Forward Search algorithm is shown. The set of alive nodes is stored in the priority queue (*Q*). Assuming that *Q* is a common FIFO (First-in-First-out) queue, whichever node has been waiting is chosen when *Q.GetFirst()* is called. At the first moment, *Q* contains the initial node x_I. Then a **while** loop is executed and it does not end until *Q* is empty. This happens when the complete graph has been explored without finding the final node, and a FAILURE returns.

In each **while** iteration, the highest ranked element x, is removed from Q. If x is the X_G then SUCCESS returns. Alternatively, if x is not the final node, the algorithm tries to apply every possible action $u \in U(x)$. For the next node $x' = f(x,u)$, the algorithm must determine if x' is found for the first time. If it is unvisited, it is added to Q. Otherwise, it must be dead or already in Q so it is not considered.

Finally, after the Dijkstra algorithm is executed, some output information is obtained. These outputs are:

- The series of nodes ordered that compose the path (Figure 57). As it has shown in the examples above, the path goes through the edges and nodes until it reaches the goal node. The nodes are numbered, so the succession of nodes in fact is a succession of numbers, each one corresponding to its respective node.
- The final cost. This is the total cost of the path, which is the sum of all the edges costs that constitute the final path (Figure 57).

Now it is shown an example of how Dijkstra algorithm works. The example is based on these project problems and objectives, this means, over the placenta´s surface (Figure 54). In these examples, it can be seen the graph used to compute the Dijkstra algorithm overlayed on the placenta. Nodes are white circles and they represent the POI. Edges are light blue lines, which represent the trajectories manually done in the exploration phase. The equator´s placenta zone is delimited by both yellow lines. As previously mentioned, this place is the zone with most probability of finding anastomosis. All these elements have been tracked and found during the exploration phase

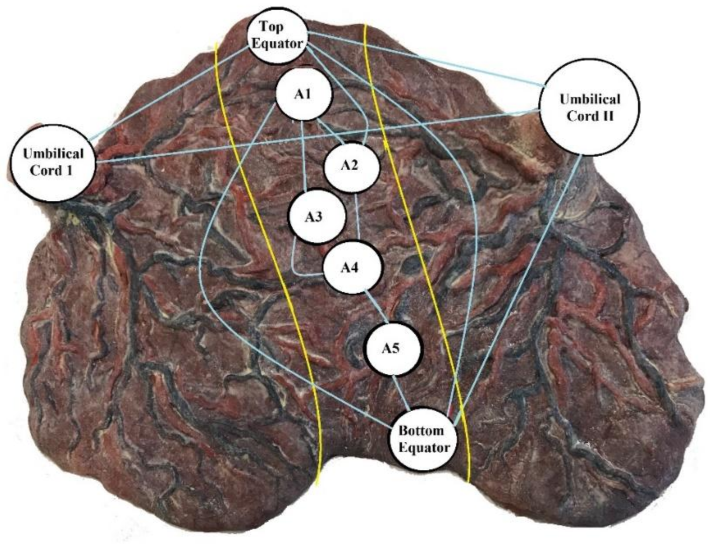


Figure 54. Graph of placenta´s surface.

Before the visual examples, in order to execute the Dijkstra algorithm, it is necessary some information. A part from the trajectories map, the cost matrix is essential for this computation. In this case, an approximated cost matrix is applied:

Nodes	UC1 (1)	UC2 (2)	TOP.E (3)	BOT.E (4)	A1 (5)	A2 (6)	A3 (7)	A4 (8)	A5 (9)
UC1 (1)	0	300	150	0	0	0	0	0	0
UC2 (2)	300	0	165	250	0	0	0	0	0
TOP.E (3)	150	165	0	315	0	85	0	0	0
BOT.E (4)	0	250	315	0	280	0	0	0	40
A1 (5)	0	0	0	280	0	35	70	0	0
A2 (6)	0	0	85	0	35	0	0	50	0
A3 (7)	0	0	0	0	70	0	0	65	0
A4 (8)	0	0	0	0	0	50	65	0	55
A5 (9)	0	0	0	40	0	0	0	55	0

Table 2. Cost matrix example 1

This cost matrix is used in the following examples to show how the Dijkstra algorithm computes the final path cost.

- **1st example (Figure 55):** In this case, the purpose is to move the fetoscope from Umbilical Cord I to Anastomosis 5. The objective is to find the fastest path to reach the last point from the first one. This can only be done through the trajectories that have already been followed and drawn (light blue lines). In this specific example, the path found is: Umbilical Cord I – Top Equator – Anastomosis 2 – Anastomosis 4 – Anastomosis 5.

  Using the cost matrix seen above, the outputs are:

  - **Path nodes:** <u>1 – 3 – 6 – 8 – 9</u>
  - **Total cost of the path:** 150 + 85 + 50 + 55 = **<u>340</u>**

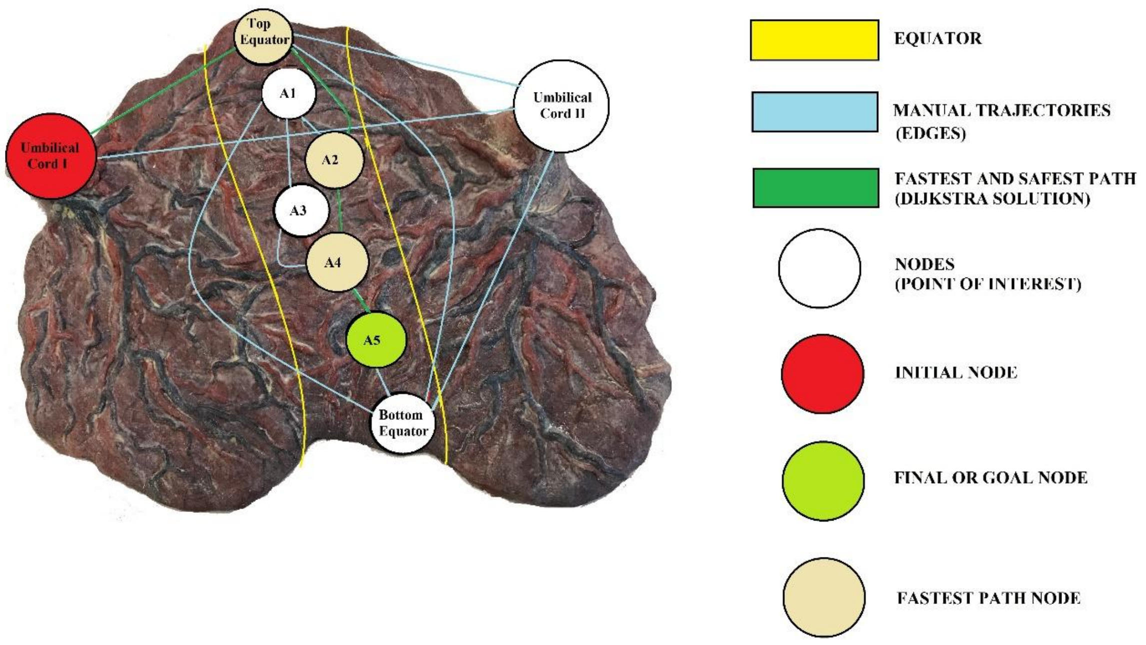


Figure 55. First example of Dijkstra algorithm working on a placenta´s surface

- **2nd example (Figure 56):** With the same conditions as in the previous example, here the objective is to find the fastest path from the Umbilical Cord II node to the Anastomosis 1 node. In this case, interconnected path nodes have been included, to see a more in deep example. This makes necessary the inclusion of these nodes in the cost matrix.

Nodes	UC1 (1)	UC2 (2)	TOP.E (3)	BOT.E (4)	A1 (5)	A2 (6)	A3 (7)	A4 (8)	A5 (9)	N1 (10)	N2 (11)	N3 (12)	N4 (13)
UC1 (1)	0	0	150	0	0	0	0	0	0	0	0	135	0
UC2 (2)	0	0	165	0	0	0	0	0	0	130	0	0	0
TOP.E (3)	150	165	0	0	0	0	0	0	0	120	0	100	0
BOT.E (4)	0	0	0	0	0	0	0	0	40	290	0	250	0
A1 (5)	0	0	0	0	0	0	0	0	0	0	20	40	25
A2 (6)	0	0	0	0	0	0	0	50	0	0	20	0	5
A3 (7)	.0	0	0	0	0	0	0	0	0	0	35	0	0
A4 (8)	0	0	0	0	0	50	0	0	55	0	0	0	0
A5 (9)	0	0	0	40	0	0	0	55	0	0	0	0	0
N1 (10)	0	130	120	290	0	0	0	0	0	0	0	0	30
N2 (11)	0	0	0	0	20	20	35	0	0	0	0	3	5
N3 (12)	135	0	100	250	40	0	0	0	0	0	3	0	0
N4 (13)	0	0	0	0	25	5	0	0	0	30	5	0	0

Table 3. Cost matrix example 2

Applying the Dijkstra algorithm over the created graph the result is: Umbilical Cord II – Node 1 – Node 4 – Anastomosis 1.

Doing in the same way than before, the outputs in this case are:

- **Path nodes:** <u>2 – 10 – 13 – 5</u>
- **Total cost of the path:** 130 + 30 + 25 = <u>**185**</u>

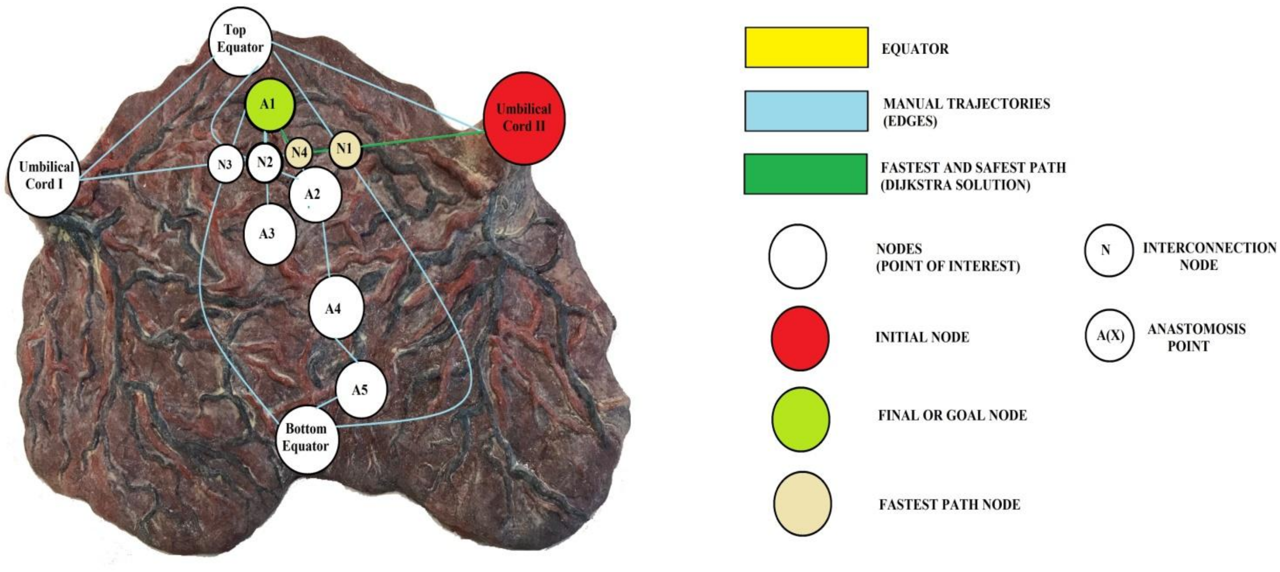


Figure 56. Second example of Dijkstra algorithm application over the placenta´s surface

With all the elements and outputs generated during the Dijkstra algorithm execution, the final path can be composed and

6.3.5 Final path points

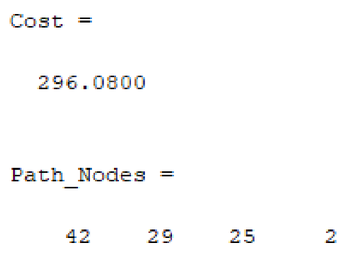
The main objective of this last phase is to collect and adjust all the information properly to give it to the robot and generate the automatic movement of the fetoscope.

Once it is obtained the sequence of the final path nodes, it is possible to calculate and perform the total final path point-by-point. As it has been seen in *Trajectory tracking and edges costs* section, during the trajectory tracking, the succession of points that compose the edges are collected and ordered in arrays, which are labeled with their respective variable name.

Now it is the time to use these arrays. As Dijkstra algorithm execution gives the succession of nodes, and henceforth, the edges that compose the path; it is used each points array that corresponds at each edge used by the final path. Ordering and joining them, it is how the final path is obtained.

Below it is shown an example of all this information obtained after the execution of the code:

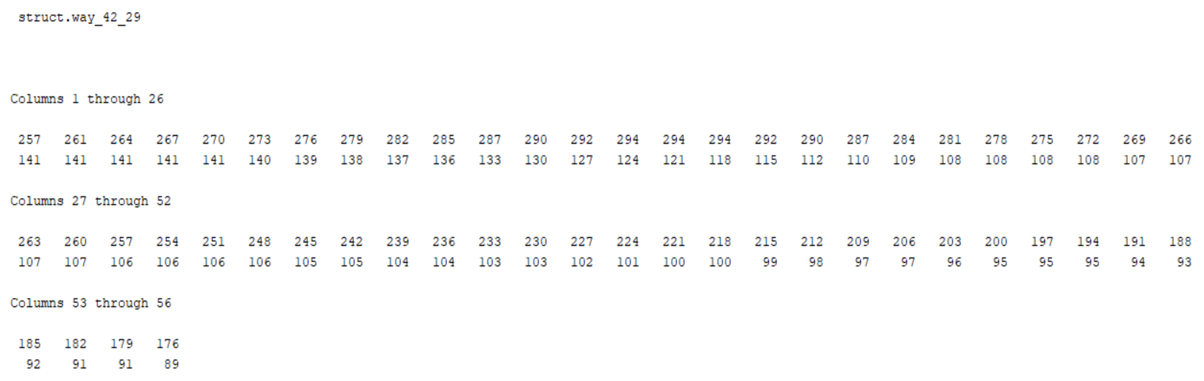
First, after Dijkstra algorithm execution, the nodes and the cost are obtained.



```
Cost =

  296.0800


Path_Nodes =

    42    29    25     2
```

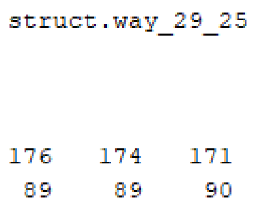
Figure 57. Cost and nodes of the final path

After that, it is possible to know the edges, as it is known that they are connecting the adjacent nodes. The list of the nodes, which appears as output of the code, is ordered. In this manner, knowing the edges, it can be called to the points that form these edges. The first, for instance, it is the edge 42-29, which is called *way_42_29* (Figure 58).



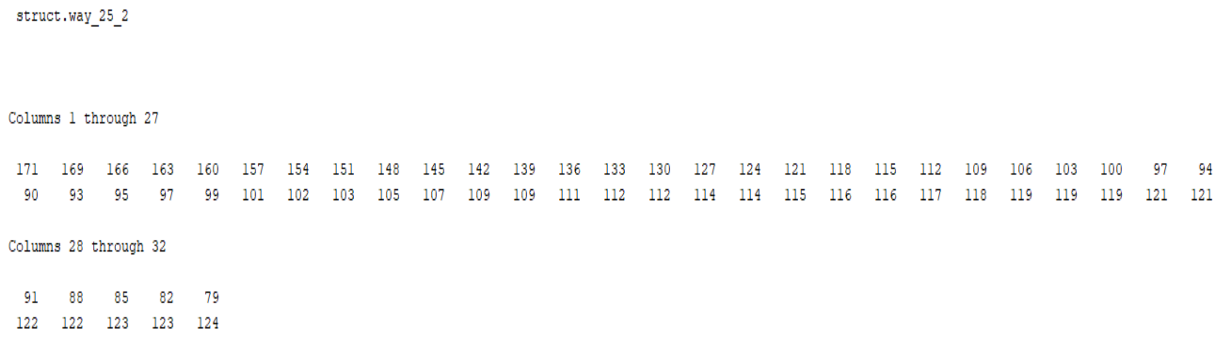
```
 struct.way_42_29


Columns 1 through 26

 257   261   264   267   270   273   276   279   282   285   287   290   292   294   294   294   292   290   287   284   281   278   275   272   269   266
 141   141   141   141   141   140   139   138   137   136   133   130   127   124   121   118   115   112   110   109   108   108   108   108   107   107

Columns 27 through 52

 263   260   257   254   251   248   245   242   239   236   233   230   227   224   221   218   215   212   209   206   203   200   197   194   191   188
 107   107   106   106   106   106   105   105   104   104   103   103   102   101   100   100    99    98    97    97    96    95    95    95    94    93

Columns 53 through 56

 185   182   179   176
  92    91    91    89
```

Figure 58. Points that form the edge from the node 42 to the node 29



```
struct.way_29_25


176   174   171
 89    89    90
```

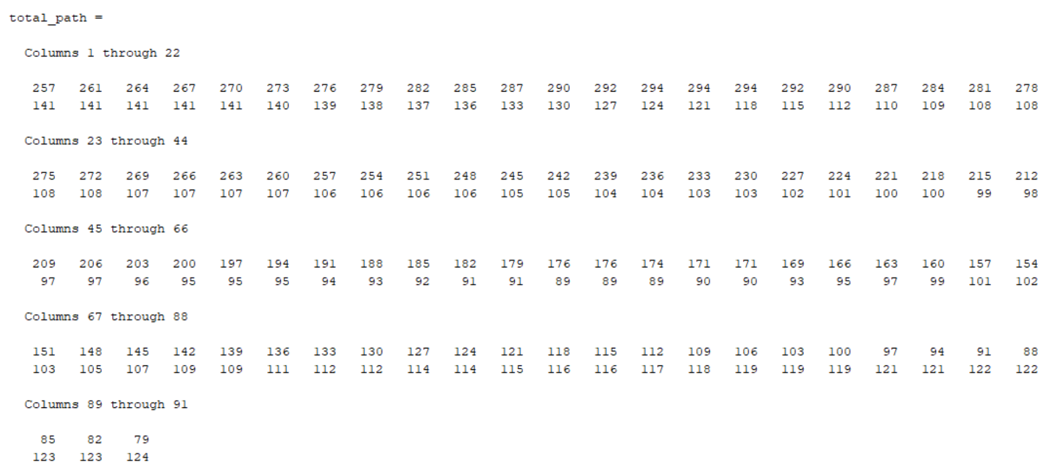
Figure 59. Points that form the edge from the node 29 to the node 25



```
 struct.way_25_2


Columns 1 through 27

 171   169   166   163   160   157   154   151   148   145   142   139   136   133   130   127   124   121   118   115   112   109   106   103   100    97    94
  90    93    95    97    99   101   102   103   105   107   109   109   111   112   112   114   114   115   116   116   117   118   119   119   119   121   121

Columns 28 through 32

  91    88    85    82    79
 122   122   123   123   124
```

Figure 60. Points that form the edge from the node 25 to the node 2

Finally, after taking all the points from their respective edges, it can be formed the total path point by point (Figure 61).



```
total_path =

  Columns 1 through 22

   257   261   264   267   270   273   276   279   282   285   287   290   292   294   294   294   292   290   287   284   281   278
   141   141   141   141   141   140   139   138   137   136   133   130   127   124   121   118   115   112   110   109   108   108

  Columns 23 through 44

   275   272   269   266   263   260   257   254   251   248   245   242   239   236   233   230   227   224   221   218   215   212
   108   108   107   107   107   107   106   106   106   106   105   105   104   104   103   103   102   101   100   100    99    98

  Columns 45 through 66

   209   206   203   200   197   194   191   188   185   182   179   176   176   174   171   171   169   166   163   160   157   154
    97    97    96    95    95    95    94    93    92    91    91    89    89    89    90    90    93    95    97    99   101   102

  Columns 67 through 88

   151   148   145   142   139   136   133   130   127   124   121   118   115   112   109   106   103   100    97    94    91    88
   103   105   107   109   109   111   112   112   114   114   115   116   116   117   118   119   119   119   121   121   122   122

  Columns 89 through 91

    85    82    79
   123   123   124
```

Figure 61. Total path point by point

It is necessary to take into account that, during the trajectory tracking phase, the points have been taken and saved every 3 points. That is why the location of the points do not correspond exactly to the previous or to the posterior. This way of sampling or saving the points can be changed at any moment, but in this example, for simplicity, it is shown in this manner.

To conclude, as it can be seen, the information goes from higher to lower level, starting from the cost, following with nodes, and finally ending with all the points. Although for the interpretation of the results, cost and nodes would be enough; the robot need the points to follow the trajectory ordered. That is why these points path are also calculated.

7 Tests and experiments

In this section, the results of the tests and experiments done during the development of the projects are shown.

The objective of whole project is to perform the automatic movement of the robot, but also accomplishing some conditions as the computation time limit. In this case, the limit agreed for the total computation time of the code was about 1 second. Therefore, even a solution is found, the optimization of the code is needed in order to reach a computation time short enough.

Although 1 second is not a short computation time, the objective is implementing this code at the end using C++. It is estimated that the code would run ten times faster. This means that, if the computation time is 1 second in Matlab; in the final implementation would be about 0.1 seconds, which is short enough.

The tests shown in this section are ordered following the project development. The first easiest approaches are avoided, as they are not orientative due to their difficulty level and their symmetry.

It is necessary to clarify two terms: Self Time and Total Time. These terms appear on the computation time statistics and they are the main elements to know where the most spent time is.

- Self-Time refers to the time spent during the execution of a particular function, but this time does not count the time spent by the Child Functions inside. Child Functions are those functions that are inside a function and are called during the execution of this function. As these statistics show the time spent by each function, they use the Self Time term in order to see how much time is used by a function purely, without depending on the time spent by other functions, and thus, know which function actually has a long computation.

- Total Time refers to the total time spent by a function since it is called. Here the Child Functions and everything inside is taken into account.

In the statistics, it can be seen a blue band representing the Total Time spent during the function. This band is divided in blue light region and dark blue region:

- **Dark blue region:** This part of the band indicates the Self-Time, which is the purely used time by the own function without taking into account the child functions within.
- **Light blue region:** This part of the band indicates the time spent by all the child functions inside the own function. It is the sum of the Total Time used by these child functions.

Dijkstra_working function is always the main code, so its Total Time reflects the total computation time of the code.

7.1 More complex virtual scenarios test

Here it is still used scenarios virtually done in Matlab.

In this phase, it is developed and tested the intermediate path method. Due to this, two main different maps exist. These maps are the final result of the code and the implementation of all the methods.

- **First test:** In this case (Figure 62) only the intermediate paths and Dijkstra algorithm are applied. It can be seen the intermediate paths over zones that accomplish the conditions for it. The problem of intermediate paths method, is the time spent during the execution (Figure 63).

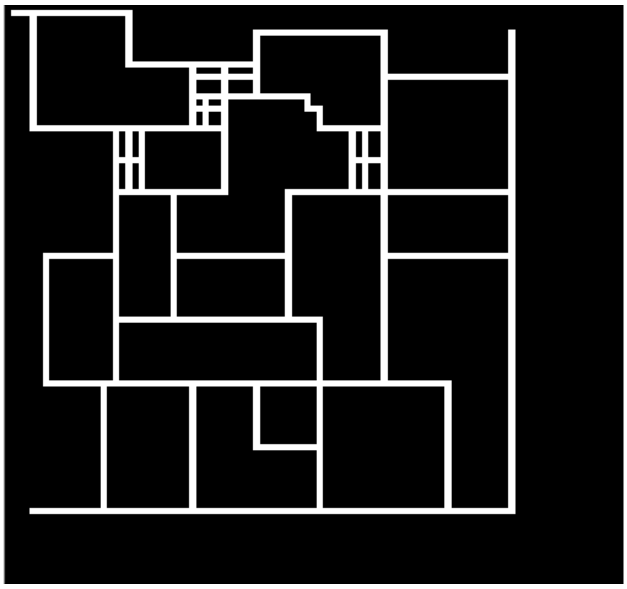
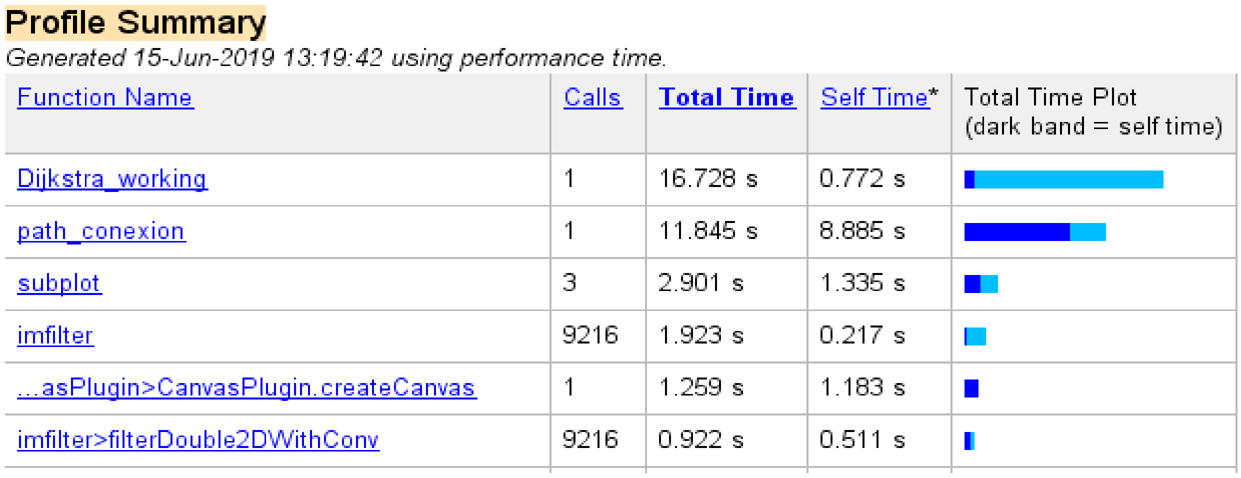


Figure 62. First test map with difficult virtual scenarios



Profile Summary

Generated 15-Jun-2019 13:19:42 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Dijkstra_working	1	16.728 s	0.772 s	
path_conexion	1	11.845 s	8.885 s	
subplot	3	2.901 s	1.335 s	
imfilter	9216	1.923 s	0.217 s	
...asPlugin>CanvasPlugin.createCanvas	1	1.259 s	1.183 s	
imfilter>filterDouble2DWithConv	9216	0.922 s	0.511 s	

Figure 63. Computation time of the first test map

As it can be seen, the Total Time spent by the main code is almost 17 seconds, which is a large computation time. Looking at the statistics, omitting the *subplot* function as it is used just as aid, the most of the time spent is in *path_conexion* function, which its Self-Time is 9 seconds.

This is the first test using the more complex maps, and although the test works successfully, the code needs to be optimized to reduce the computation time.

- **Second test:** In this case, the intermediate paths method is still applied, but also some others. For instance, it is taken into account that the trajectories can be wider than one pixel, and the *tracking* or *track_traj* function is not prepared for this. So in this phase finally, it is decided to work with those trajectories that are wider than 1 pixel and fix them to perform the *tracking* function. This is done because in the intermediate paths method is done something similar, and this maintains the consistency of the code. The map is used for the second and the third test in this phase (Figure 64).

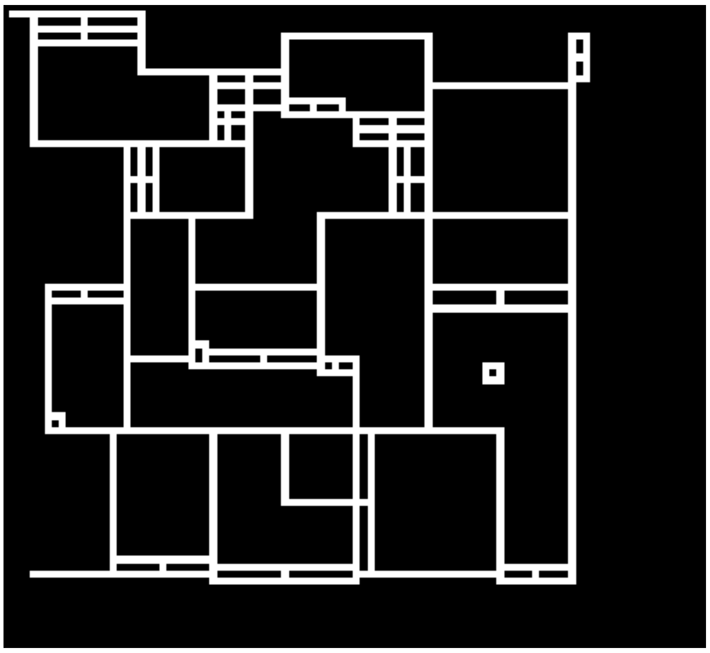
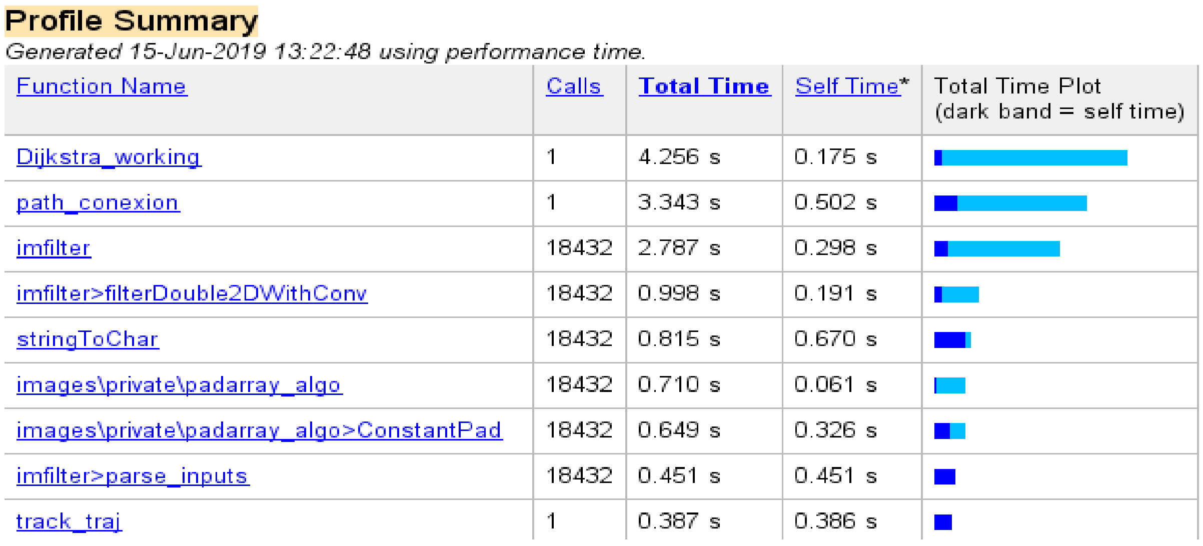


Figure 64. Second map used. Difficult virtual scenario with trajecrtories wider than 1 pixel

As it is shown in the map, the intermediate paths are still there and other modifications are also in this map. These modifications are due to the wider trajectories.


Profile Summary

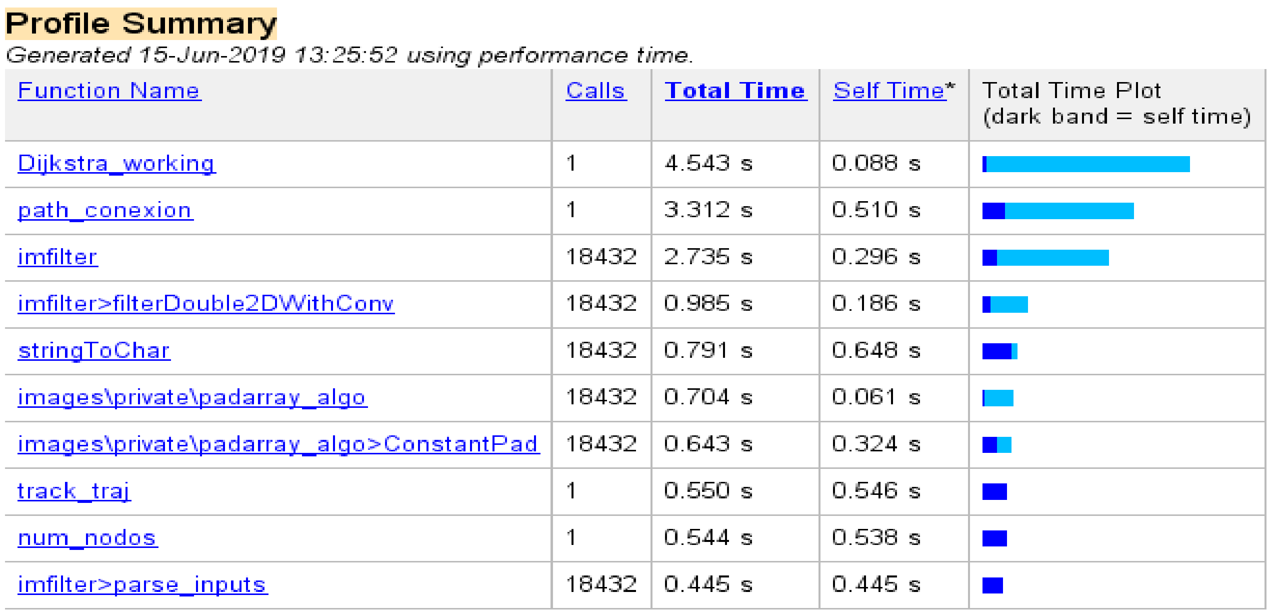
Generated 15-Jun-2019 13:22:48 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Dijkstra_working	1	4.256 s	0.175 s	
path_conexion	1	3.343 s	0.502 s	
imfilter	18432	2.787 s	0.298 s	
imfilter>filterDouble2DWithConv	18432	0.998 s	0.191 s	
stringToChar	18432	0.815 s	0.670 s	
images\private\padarray_algo	18432	0.710 s	0.061 s	
images\private\padarray_algo>ConstantPad	18432	0.649 s	0.326 s	
imfilter>parse_inputs	18432	0.451 s	0.451 s	
track_traj	1	0.387 s	0.386 s	

Figure 65. Computation time of the second test

In this case, the computation time is significantly shorter than the previous test. Although is not under the limit, there are more modifications than before and the total computation time is four times less. The code still needs optimization but there is a progression.

- **Third test:** In this test is also used the map for the second (Figure 64). In this case, it is added to the code some information collection, as *mat_info*. This information is about every node and saves the number of the node, the numbers of the directly connected nodes and the costs of these connections.



Profile Summary

Generated 15-Jun-2019 13:25:52 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Dijkstra_working	1	4.543 s	0.088 s	
path_conexion	1	3.312 s	0.510 s	
imfilter	18432	2.735 s	0.296 s	
imfilter>filterDouble2DWithConv	18432	0.985 s	0.186 s	
stringToChar	18432	0.791 s	0.648 s	
images\private\padarray_algo	18432	0.704 s	0.061 s	
images\private\padarray_algo>ConstantPad	18432	0.643 s	0.324 s	
track_traj	1	0.550 s	0.546 s	
num_nodos	1	0.544 s	0.538 s	
imfilter>parse_inputs	18432	0.445 s	0.445 s	

Figure 66. Computation time of test 3.

Even the time spent is a bit more, taking into account the added collection of information; the result is not bad at all. However, the optimization of the code is still needed.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

7.2 Real exploration map tests

During this phase, the objective is that everything develop before works in a real exploration map, and the computation time do not exceed the computation time expected.

First, the main code of the *More complex virtual scenarios test* is applied over the map, obtaining the following results.

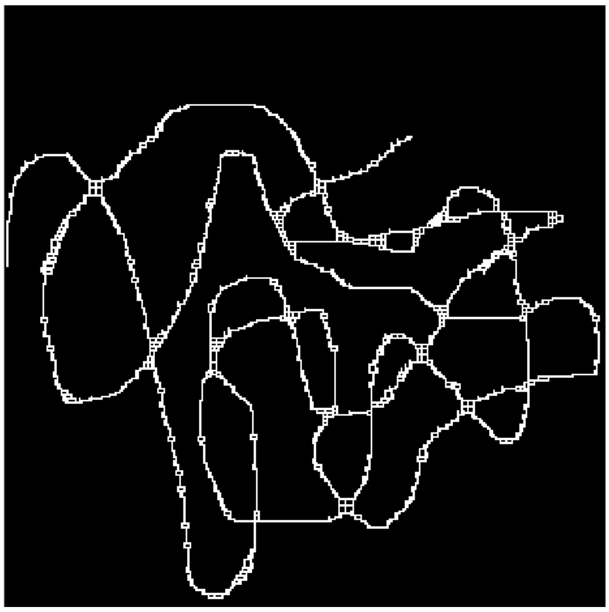
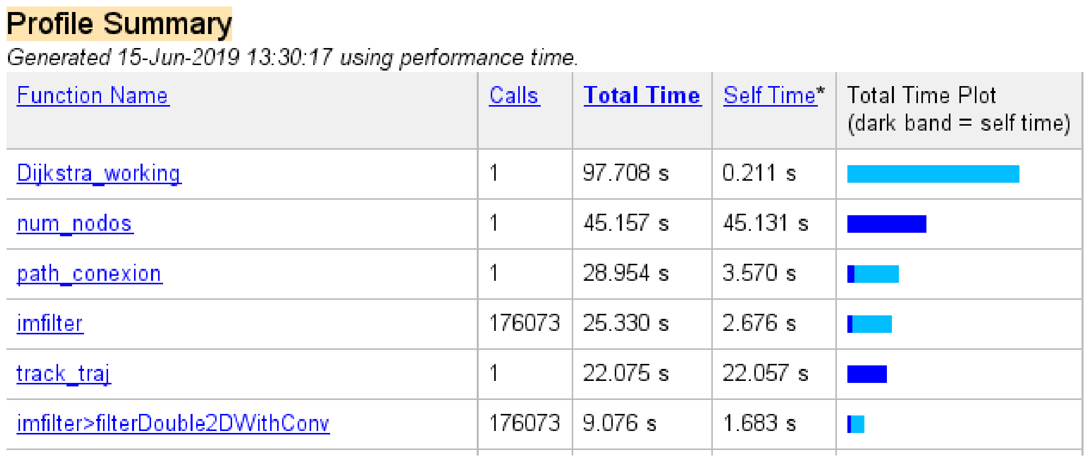


Figure 67. Real exploration map obtained.

As it can be seen, the trajectories of the map obtained are intricate. The methods applied with virtual scenarios perform a large number of intricate paths.



Profile Summary

Generated 15-Jun-2019 13:30:17 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Dijkstra_working	1	97.708 s	0.211 s	
num_nodos	1	45.157 s	45.131 s	
path_conexion	1	28.954 s	3.570 s	
imfilter	176073	25.330 s	2.676 s	
track_traj	1	22.075 s	22.057 s	
imfilter>filterDouble2DWithConv	176073	9.076 s	1.683 s	

Figure 68. Computation time of the first test with a real map

As expected, the time spent to execute the main code is too long. As seen above, here is the point when a reformulation occurs, as although optimizing the current code, it is difficult to achieve the goal time.

7.3 Geodesic distance test

The first method used as alternative is using the geodesic distance. The application of this function makes these results. The obtained map in this case is less complicated and intricate (Figure 69).

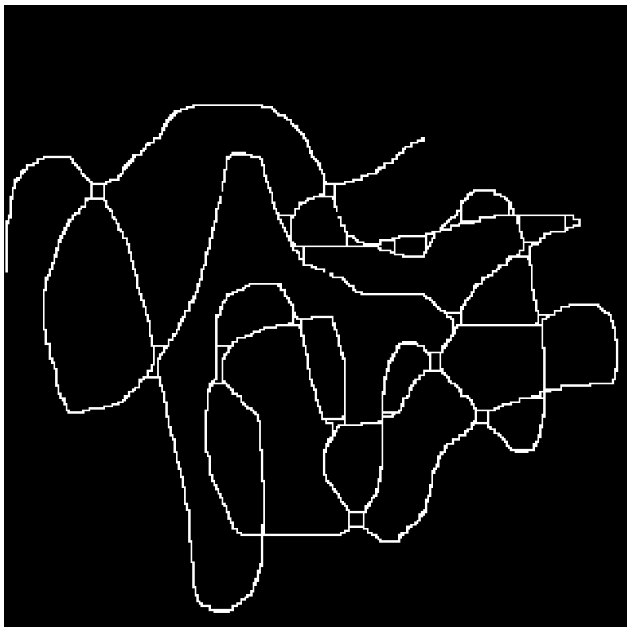
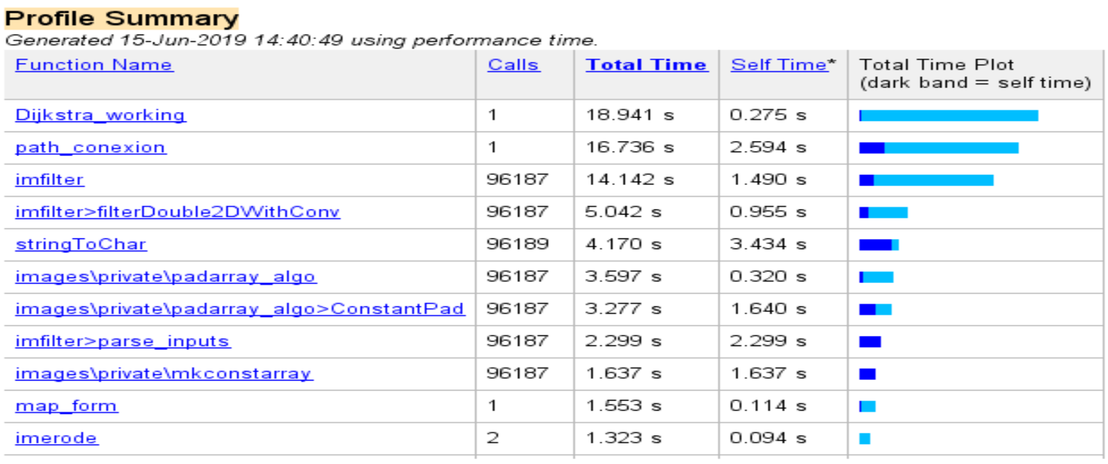


Figure 69. Real exploration map in geodesic distance test



Profile Summary

Generated 15-Jun-2019 14:40:49 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Dijkstra_working	1	18.941 s	0.275 s	
path_conexion	1	16.736 s	2.594 s	
imfilter	96187	14.142 s	1.490 s	
imfilter>filterDouble2DWithConv	96187	5.042 s	0.955 s	
stringToChar	96189	4.170 s	3.434 s	
images\private\padarray_algo	96187	3.597 s	0.320 s	
images\private\padarray_algo>ConstantPad	96187	3.277 s	1.640 s	
imfilter>parse_inputs	96187	2.299 s	2.299 s	
images\private\mkconstarray	96187	1.637 s	1.637 s	
map_form	1	1.553 s	0.114 s	
imerode	2	1.323 s	0.094 s	

Figure 70. Computation time applying geodesic distance

However, despite the problems and the lack of information, the computation time is still long. It can be optimized, but due to the problems found, the method is finally discarded.

8 Final results

Finally, after some approaches and test, the final and definitive solution is found. The result accomplishes every objective and purpose of the project.

The resultant map is the following (Figure 71):

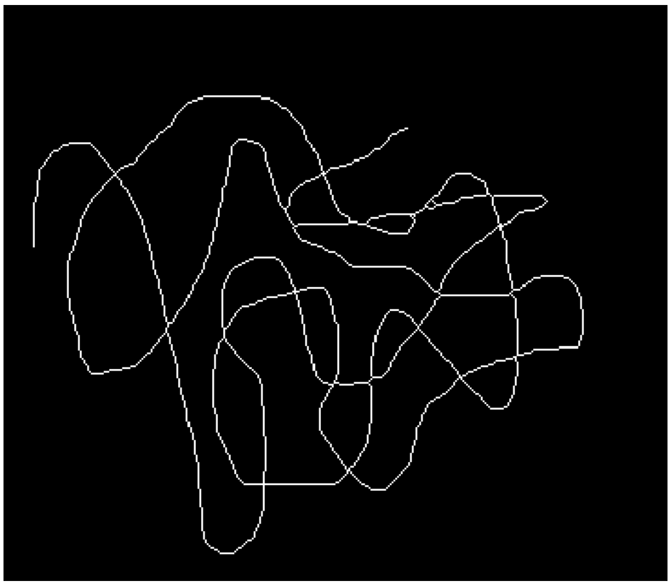
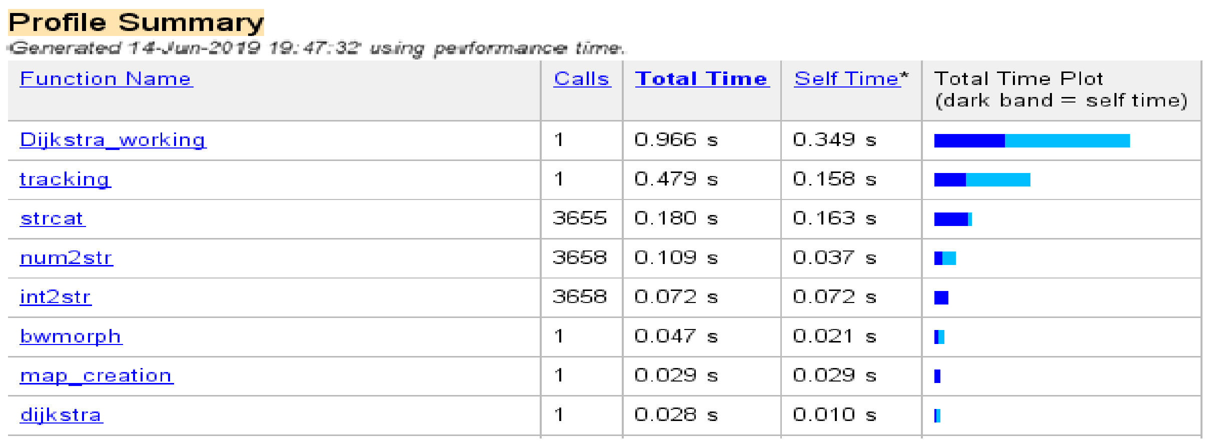


Figure 71. Final result map

Compared to the others, this map is the least intricate and complex. This allows to perform other operations, as *tracking*, in an easier way. The required computation time is also reduced although it needs to be optimized.

Once the optimization is enough to achieve the computation time goal, the statistics show the following (Figure 72):



Profile Summary

Generated 14-Jun-2019 19:47:32 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Dijkstra_working	1	0.966 s	0.349 s	
tracking	1	0.479 s	0.158 s	
strcat	3655	0.180 s	0.163 s	
num2str	3658	0.109 s	0.037 s	
int2str	3658	0.072 s	0.072 s	
bwmorph	1	0.047 s	0.021 s	
map_creation	1	0.029 s	0.029 s	
dijkstra	1	0.028 s	0.010 s	

Figure 72. Statistics of computation time of the final solution

It can be seen that the total computation time of the main code is even less than 1 second. This has been possible due to the reformulations and new adjustments applied during the whole development of the project. However, this does not mean that the code cannot be more optimized.

For instance, the function *map_creation* is a function used to help during the project development, making easy to create a map or to change the used map. In the implementation on the system, this function it is not used, as the map is already created, so it is a not-real time spent.

There are other operations, like *strcat* or *num2str* that are used for saving the edges point in a variable. These functions spent large amounts of time due to the number of calls that they receive. This could be optimized changing the method in which the edges points are saved, for example, saving the points in a 3D matrix, similar to cost matrix.

Applying all these changes, and taking into account the reduction of time using other languages like C++ when implementing the code on the system, the time spent could be less than 0.1 seconds for the code´s execution.

8.1 Final examples

Now it is show some map examples, where the final path is also shown. The initial and goal points have been changed in each case. The green path is the final path that Dijkstra algorithm has computed. The red point refers to the initial point and the blue point to the goal or final point.

8.1.1 Map 1

1. Initial point: [79,124]. Goal point: [257,141]. Total cost: 296.08

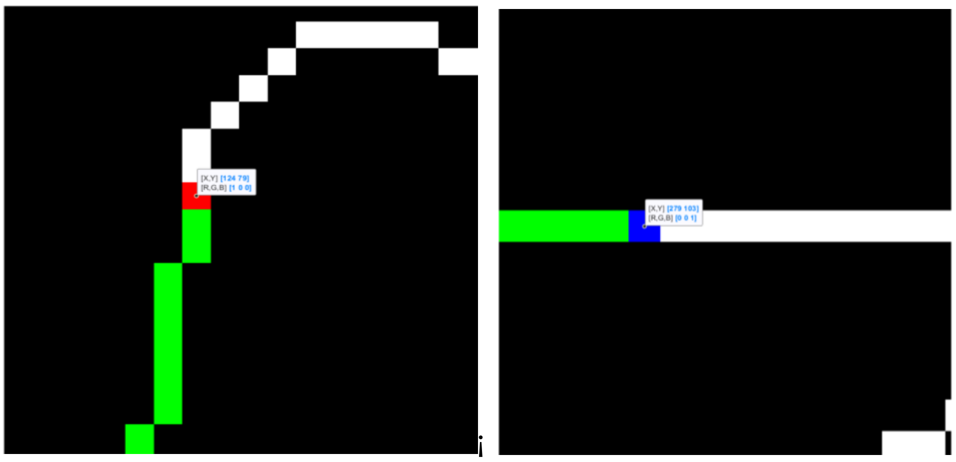


Figure 73. Initial and goal points (case 1)

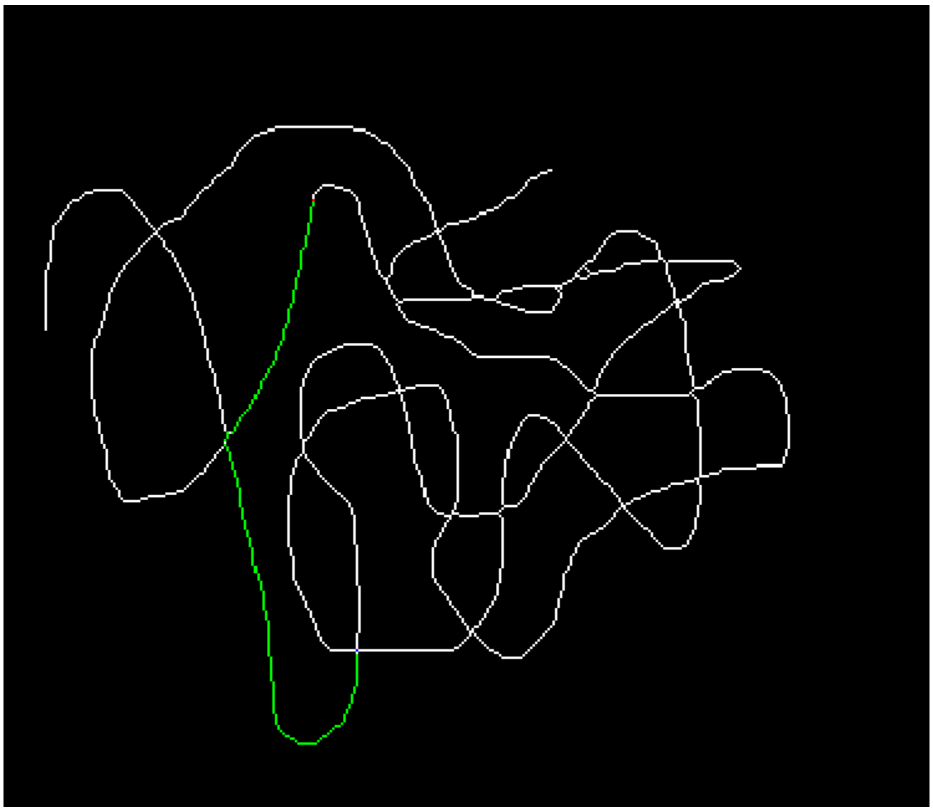


Figure 74. Final result map (case 1)

2. Initial point: [130,18]. Goal point: [103,279]. Total cost: 390.08

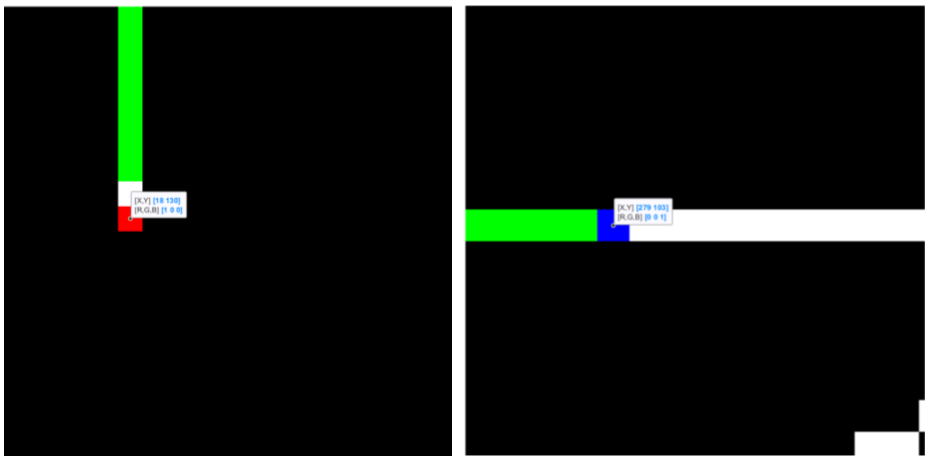


Figure 75. Initial and goal points (case 2)

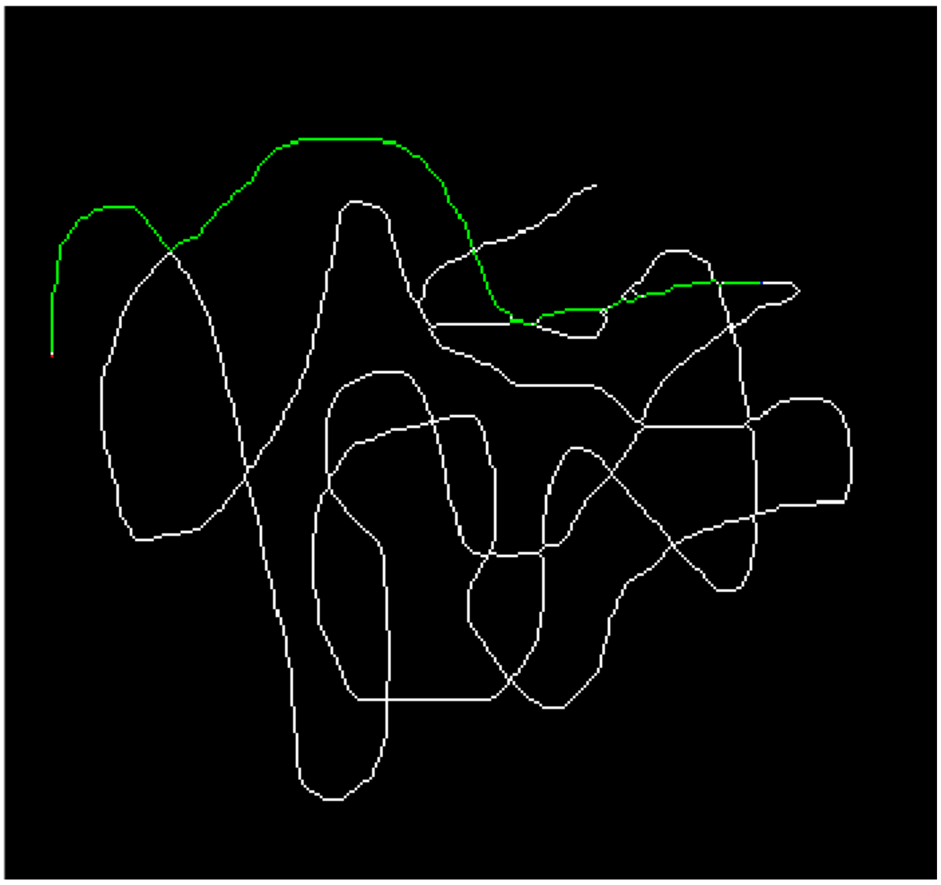


Figure 76. Final result map (case 2)

3. Initial point: [170,312]. Goal point: [212,114]. Total cost: 299.80

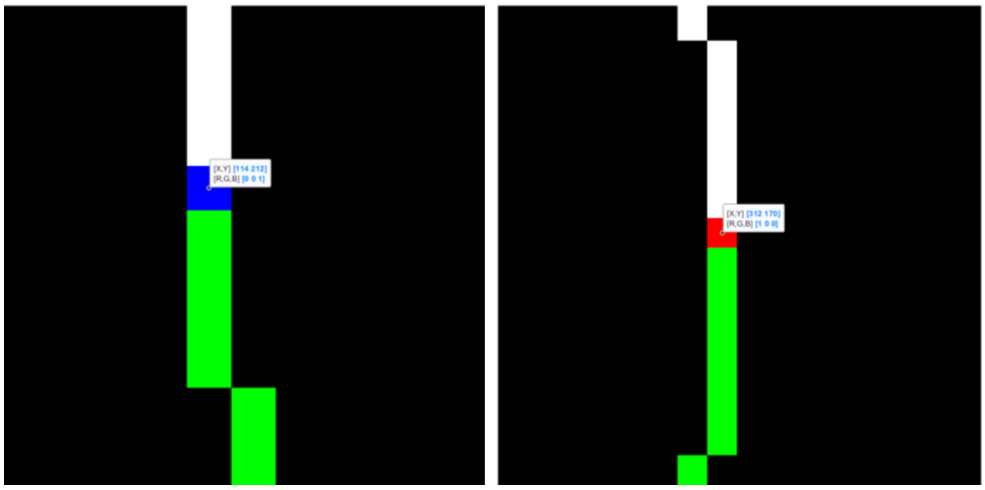


Figure 77. Initial and goal points (case 3)

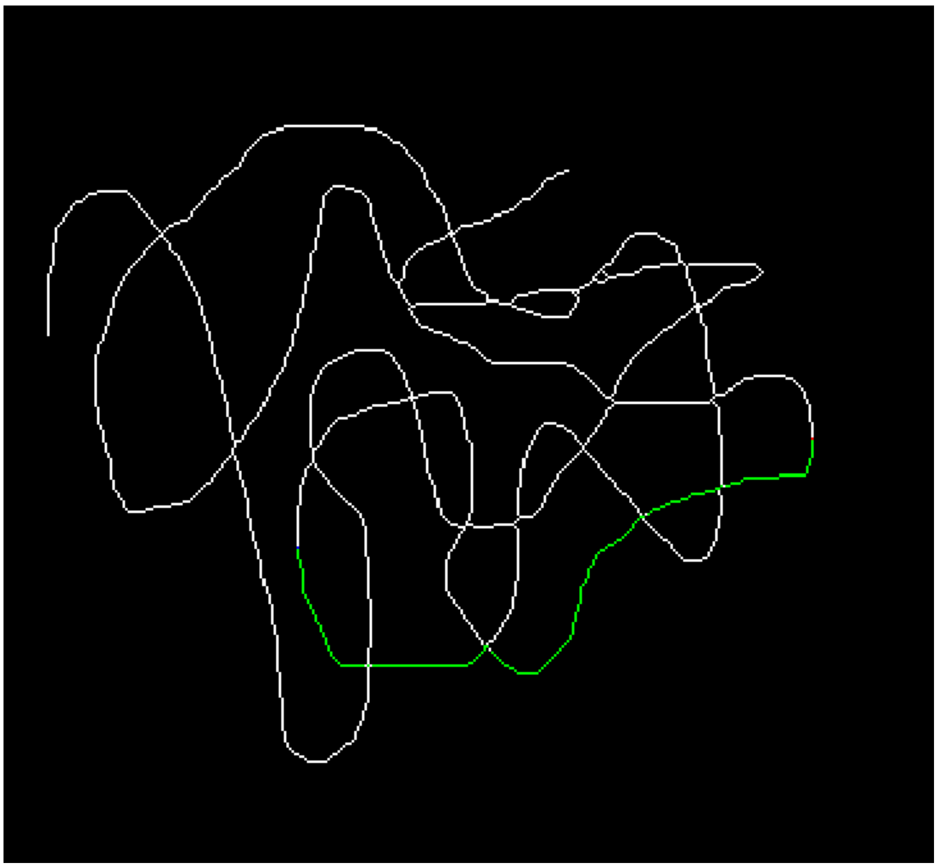


Figure 78. Final result map (case 3)

8.1.2 Map 2

The initial map:

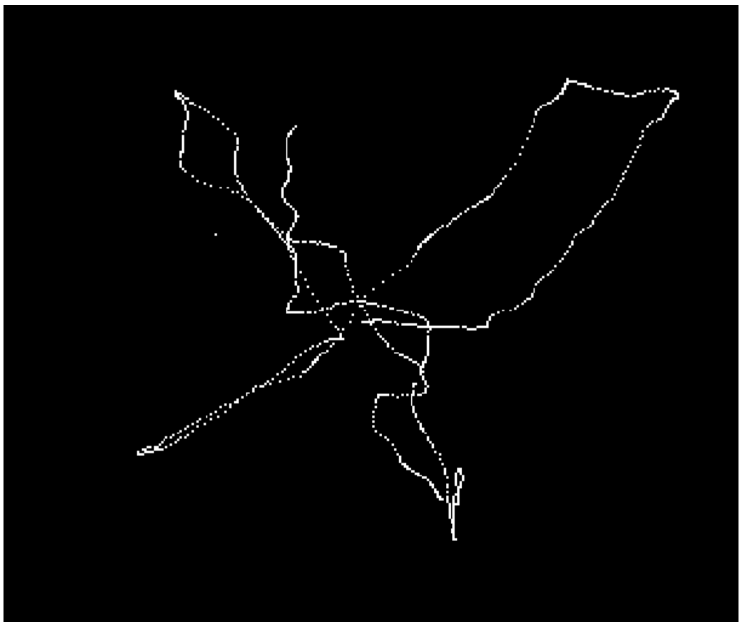


Figure 79. Initial map 2 (sampling errors)

Map modified:

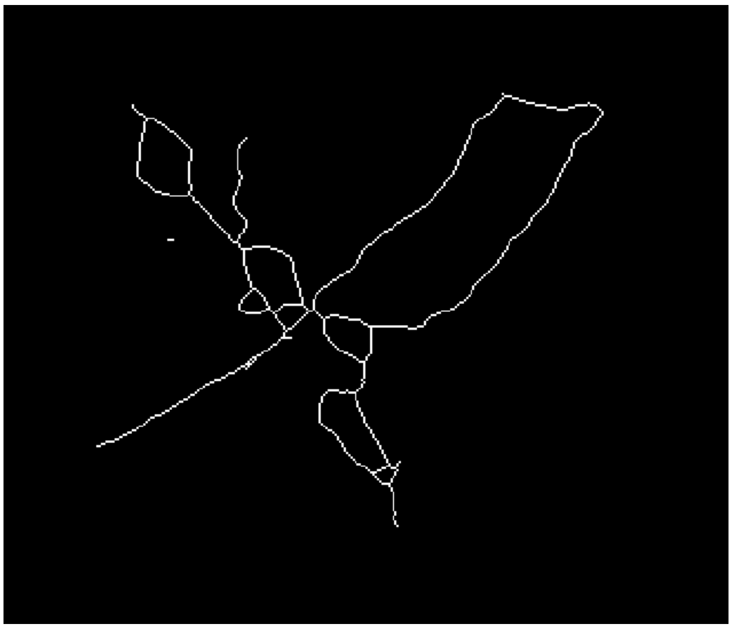


Figure 80. Map 2 after morphological operations and modifications

Initial point: [99,113]. Goal point: [203,197]. Total Cost: 195.08.

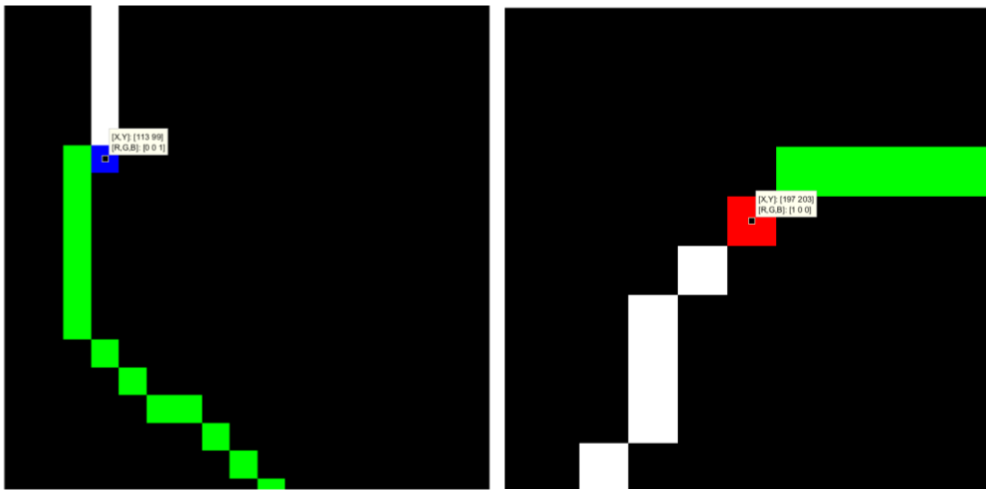


Figure 81. Initial (left) and goal (right) points

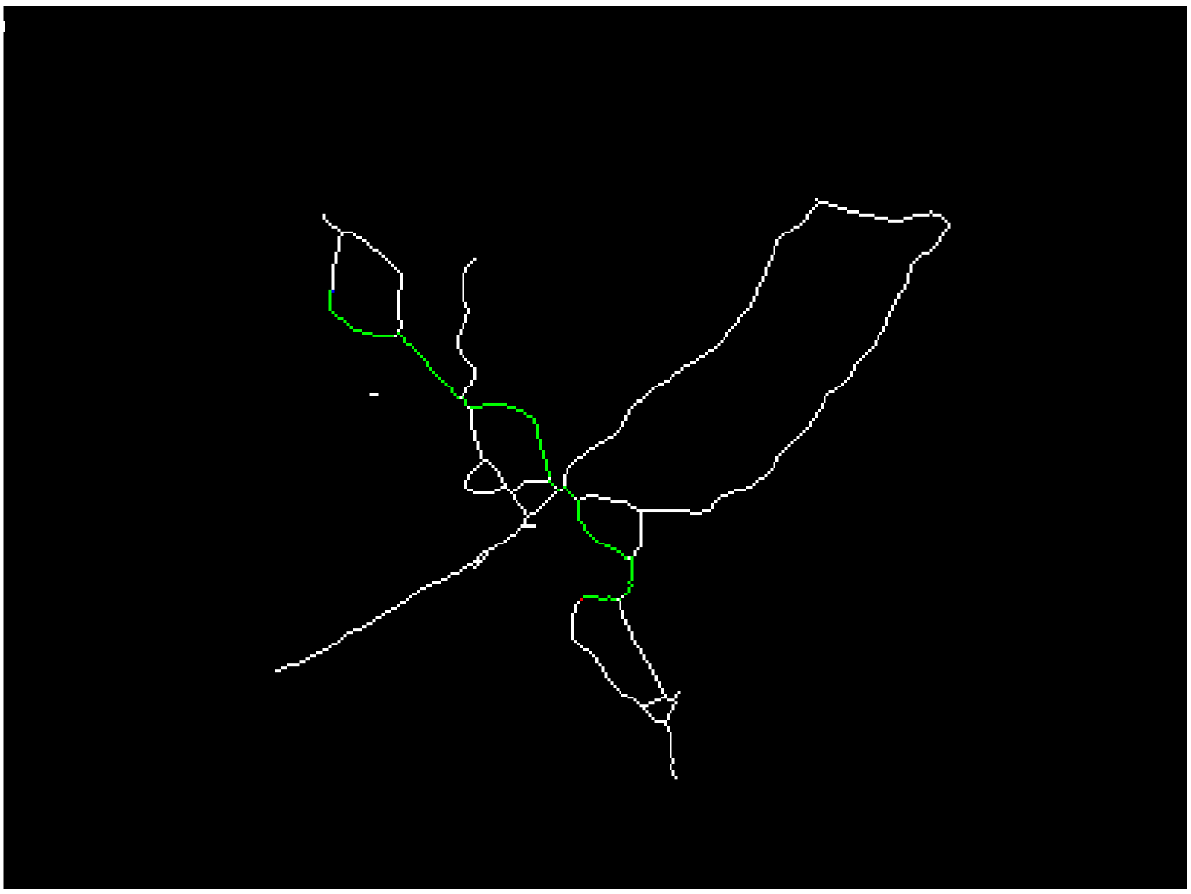


Figure 82. Final result in map 2

8.1.3 Map 3

Initial map:

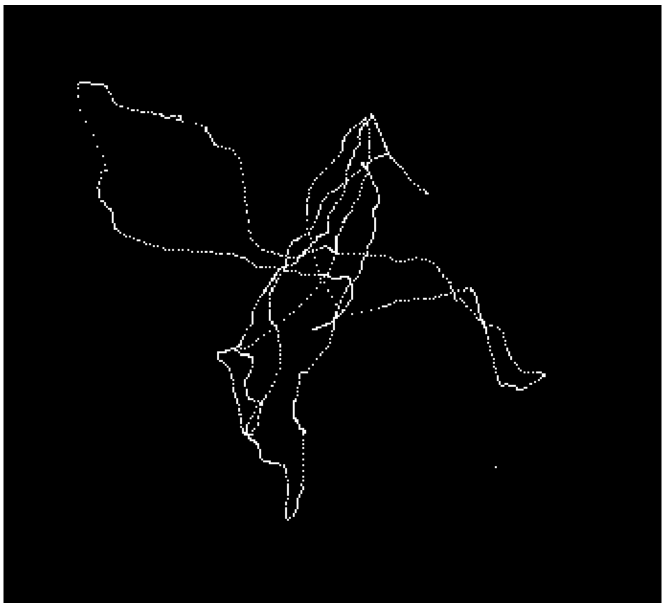


Figure 83. Initial map 3

Map modified:

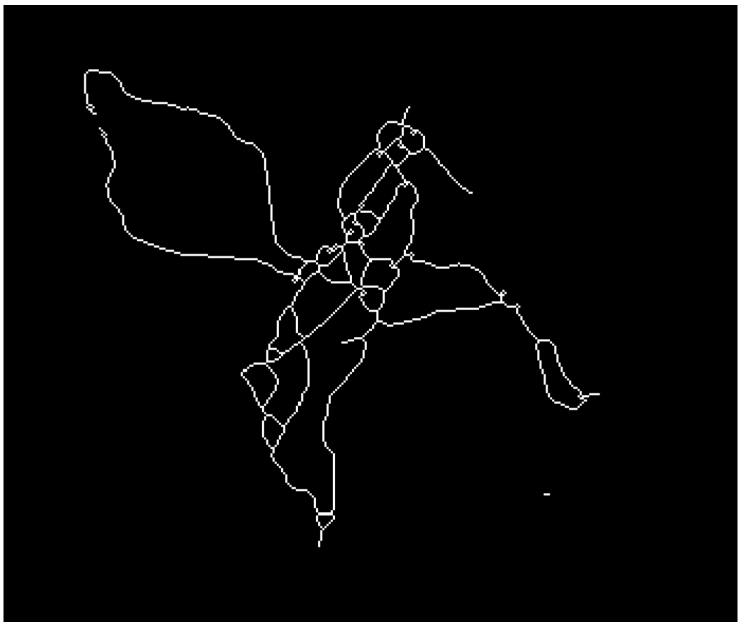


Figure 84. Map 2 after modifications

Initial point: [99,245]. Goal point: [213,151]. Total Cost: 217.16.

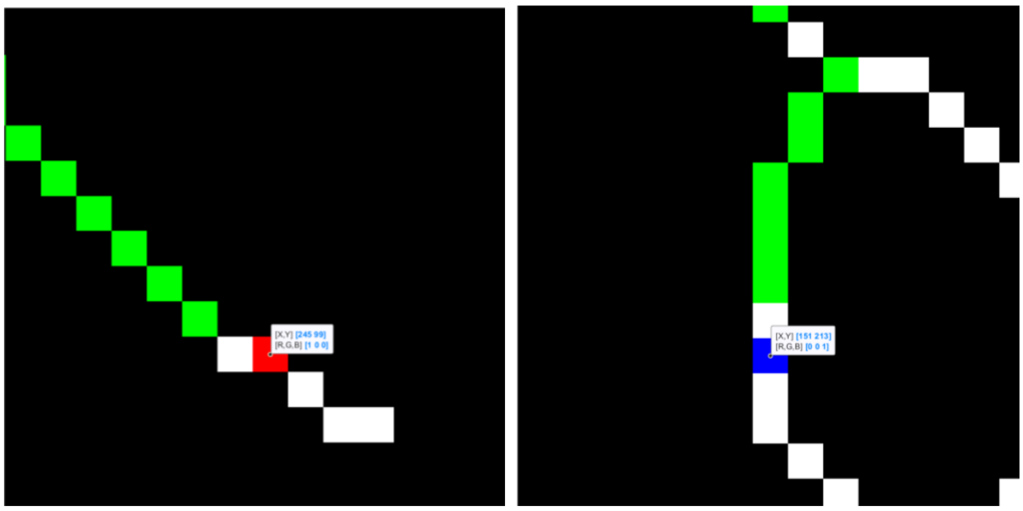


Figure 85. Initial and final path points

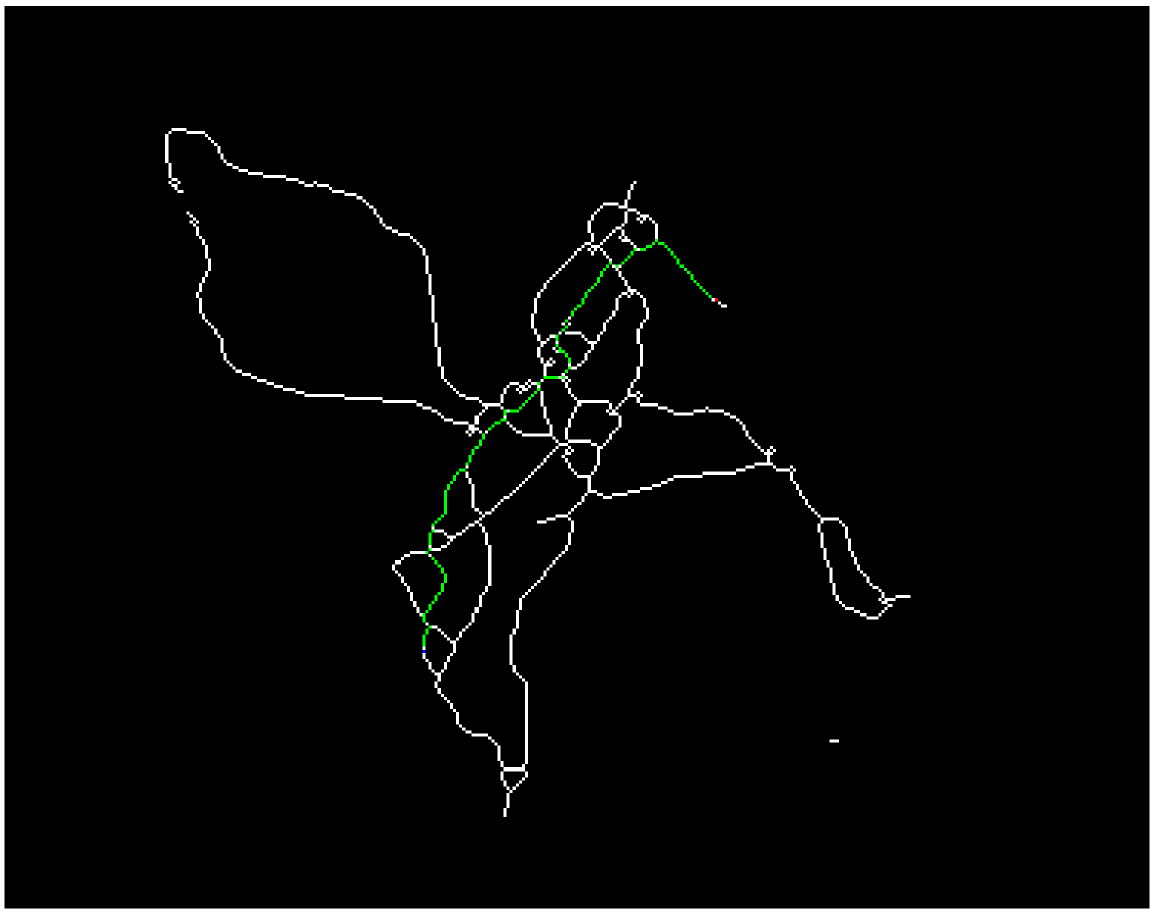


Figure 86. Final result in map 3

8.1.4 Map 4

Initial map:

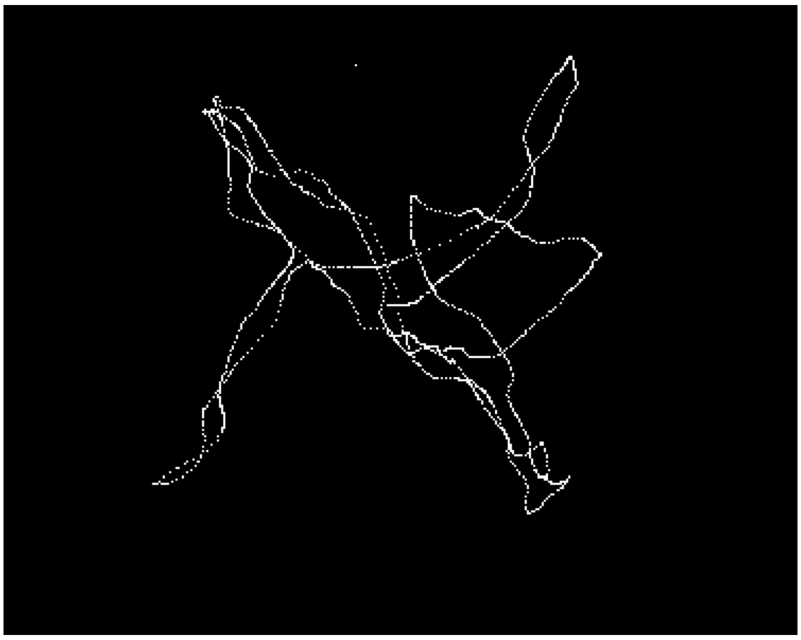
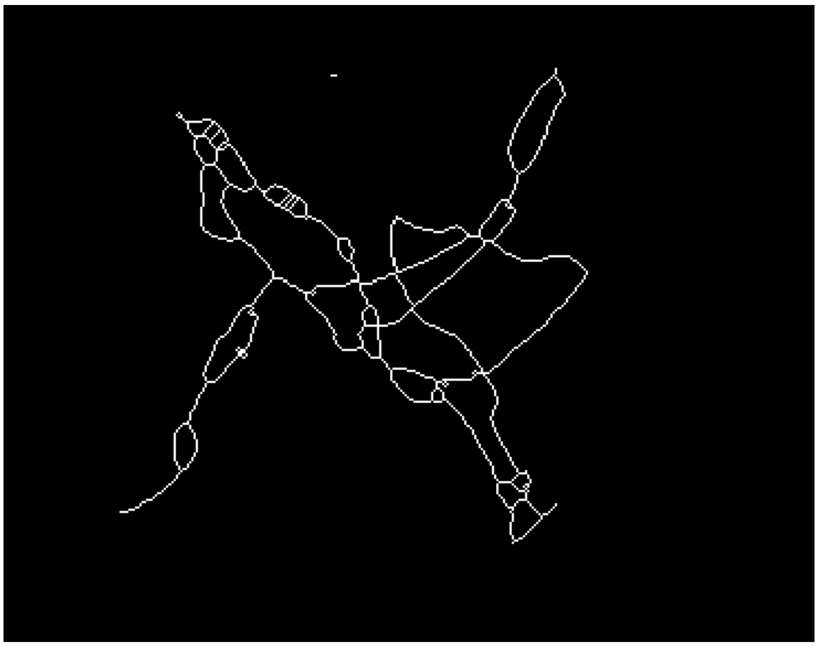


Figure 87. Initial map 4

Map modificated:



Initial point: [79,119]. Goal point: [250,256]. Total Cost: 257.94

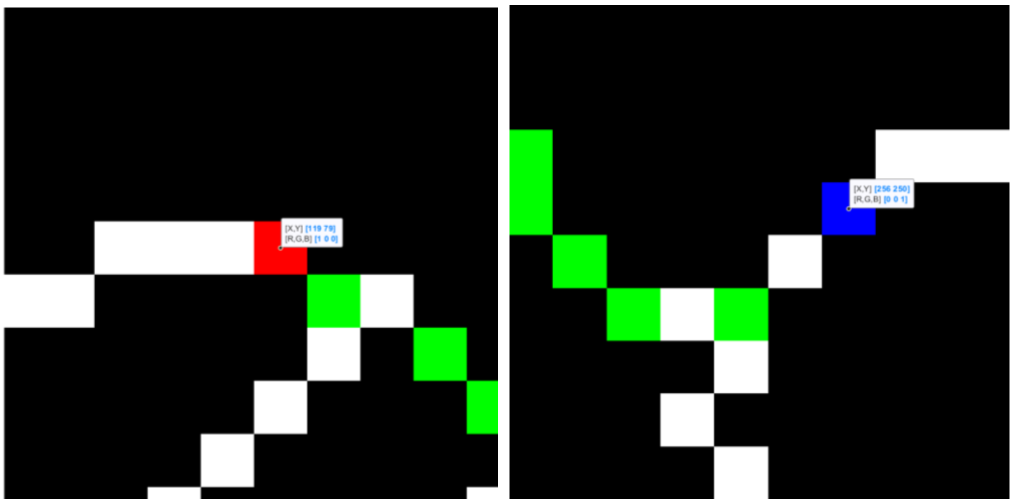


Figure 88. Initial and final path points

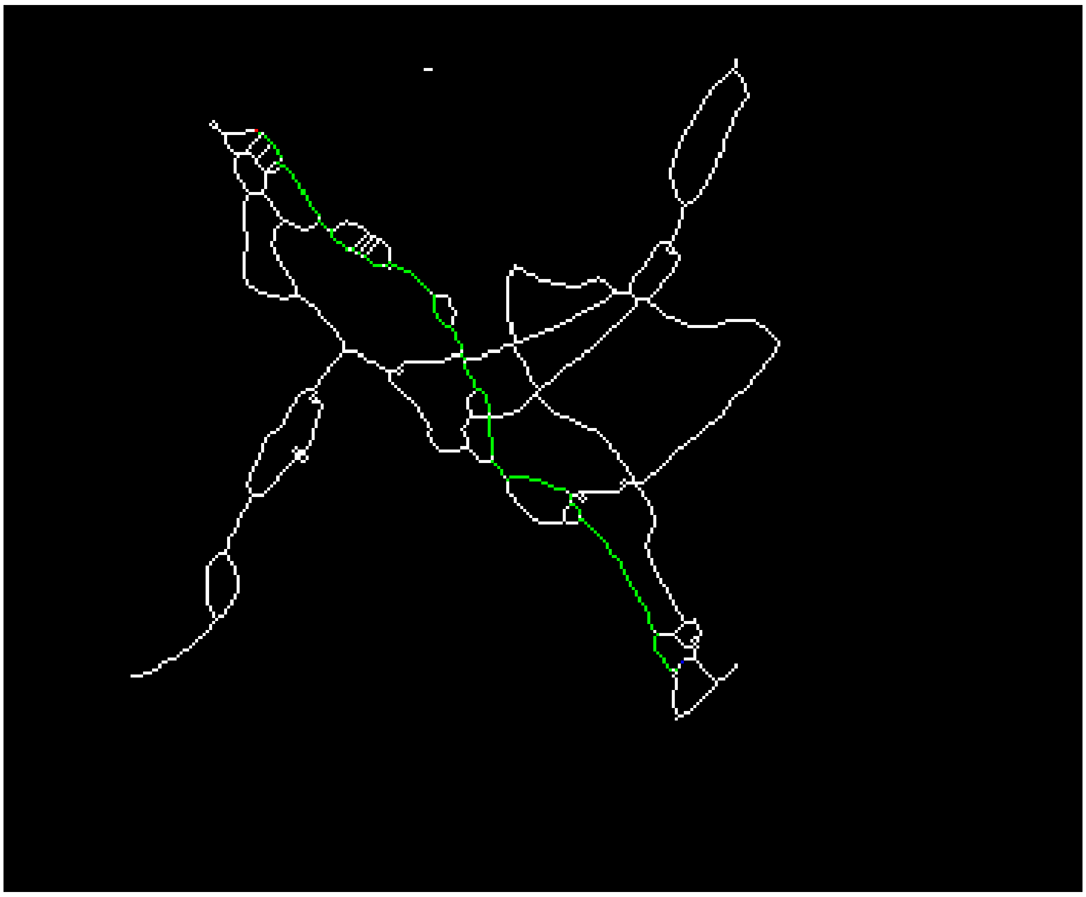


Figure 89. Final result in map 4

9 Conclusions

In conclusion, this project is part of R&D system developed in the ESAII laboratory in the UPC. This system belongs to the Medical Robots field, and its aim is to help and make easier the labor of the surgeons in TTTS surgeries.

Medical Robots is a field where the most of the projects are in R&D. There are some systems currently used, but there is an important difference. The hard tissues are easier to study and to operate by medical robots, as they are mostly non-deformable and remain in their positions, which allows take better references. However, soft tissues are more difficult, and that is why the most of the project are currently in R&D. This is a project based on the automation of a robot over a soft tissue. However, this specific case allows the application of automatic systems due to its controllable work environment.

The system is a teleoperated system and it is composed by a robotic system, whose end-effector is a fetoscope. This is the tool used in this kind of surgeries. There is also a User Interface, which helps the surgeon to perform some operations or given to him important information.

This project is in charge of the automatic movement of the robot and the fetoscope over the placenta´s surface. For this Computer Vision tools are used. Then, all the necessary information is acquired and given to the robot, which must move to the user desired point through the fastest and safest way. In order to do this, following the trajectories manually done by the surgeon is considered safe, while for the speedness, the fast path algorithm Dijkstra is applied to find the fastest path.

Finally, the results obtained, applying the developed algorithm in real scenarios, are successful. The most efficient, safest and fastest path is found in different real maps.

This whole system and this project look for a better performance in these surgeries. The system aims to reduce the time spent during the surgery, and also to reduce the mental and physical work-load suffered by the surgeon.

10 Bibliography

1. Casals, A., Frigola, M. and Amat, J. (2009). *La Robótica, una valiosa herramienta en Cirugía.*. [online] Barcelona. Available at: https://pdf.sciencedirectassets.com/280589//client [Accessed 5 Jun . 2019].

2. Casals, A., Basomba, J., Bergès, E., Frigola, M. and Amat, J. (2008). *Sistema Integrado de Posicionado, Visualización y Acotamiento de Áreas de Trabajo en Cirugía Ortopédica*. [online] Barcelona. Available at: https://upcommons.upc.edu/bitstream/handle/2117/19412/Casals.pdf [Accessed 5 Jun. 2019].

3. M. Shoham, I. H. Lieberman, E. C. Benzel, D. Togawa, E. Zehavi, B. Zilberstein, M. Roffman, A. Bruskin, A. Fridlander, L. Joskowicz, S. Brink-Danan & N. Knoller(2007) Robotic assisted spinal surgery–from concept to clinical practice, Computer Aided Surgery, 12:2, 105-115, DOI: 10.3109/10929080701243981

4. Bauzano, E., García-Morales, I. and Muñoz-Martinez, V. (2013). *Asistencia de Robots Colaborativos para Procedimientos de Sutura Vía Cirugía Mínimamente Invasiva*.

5. Adhami, L. and Coste-Manière, È. (2003). *Optimal Planning for Minimally Invasive Surgical Robots*. IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION. [online] Washington, DC. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1236758 [Accessed 6 Jun. 2019].

6. Vila, A. and Casals, A. (2019). *Interficie per a la interacció cirugià-robot.*. Barcelona.

7. Wolf, A. and Shoham, M. (n.d.). *Medical automation and Robotics*.

8. Mittelstadt, B. and P., H. (1993). *Development of a surgical robot for cementless total hip replacement*.

9. Paul, H. and M., B. (1992). *A surgical robot for total hip replacement surgery*.

10. LaValle and M., S. (2006). *Planning algorithms*. Illinois.

11. Blackwell, M. and N., C. (1998). *An image overlay system for medical data visualization*. Berlin.

12. Galea, P. and M., J. (1982). *Feto-fetal transfusion syndrome*. Royal Maternity Hospital, Glasgow.

13. Cruz-Martínez, R. and G., E. (2014). *Cirugía fetal endoscópica*.

14. Wang, Y. and K., G. (n.d.). *Automated Endoscope System - OptimalPositioning*. Santa Barbara, California.

15. Zhang, T. and Suen, C. (1984). *A Fast Parallel Algorithm for Thinning Digital Patterns*. Image Processing and Computer Vision. Robert M. Haralick.