1 Appendices

1.1 Functions overview

1.1.1 Developed functions
Here it is presented certain functions that are used during the development of the project. It must be taken into account that some of them are used at the beginning and later disappear, while others appear later due to the different reformulations of the code. Each one is described and presented in its own context.

- **calc_saltos**: This function is used to locate nodes. Calc_saltos is a function that determines the number of paths that can be followed from a point, counting the jumps or colour changes between the neighbour pixels. The number of jumps indicates the number of existing roads. Knowing that a point, following the definition above, is a node when the possible paths are 3 or more, the points where this function finds 3 or more jumps are defined as nodes (Figure 1 and Figure 2).

  This function is defined from the beginning of the project, and is used during all the time, including it in the final approach and solution.

- **Dijkstra**: This function forms the Dijkstra algorithm that is applied for the search of the fastest way.

  It has as inputs the cost matrix between nodes, the goal point and the initial point; and as outputs it is obtained the total cost of the fastest way (or with less cost) and the succession of nodes that follows that way.

  This function also applies throughout the whole project as it is one of the main and most basic functions.
Figure 1. Visual interpretation of how calc_staltos function works.

There are 2 jumps or colour changes (→).  
2 changes = 2 paths  
Node → ≥ 3 paths  
This point is not a node

Figure 2. Visual interpretation of how calc_saltos function works.

There are 4 jumps or colour changes (→).  
4 changes = 4 paths  
Node → ≥ 3 paths  
This point is a node
• **Map_creation:** This function is completely virtual. It is used to create maps and to choose which one you want to work on. In this function are stored numbered all the maps created, so that when one wants to work with any of them, it is only necessary to call to the function and as entry to introduce the number corresponding to that map. The initial and final points are also stored here, so that when this function is called to create the map, the initial and final points (corresponding to the robot's virtual initial and final position) are already defined. These points can be changed at any time from the same function.

This function is only useful for development, since later the mapping is done through the robot and the system, so it is not necessary to create it.

As an entry it is necessary to have the number corresponding to the map in which the user wants to work. As outputs, the total dimensions of this map are obtained (working with square maps, only one value is obtained), the initial point and the final point.

*Map_creation* is used during the whole project up to the use of real maps; although it is still maintained in this point due to the possibility of needing to work with virtual maps for any reason.

• **Map_form:** it is a function for the necessary modifications on the initial map. Although not used for too long, the approach of this function is important throughout the project, as it forms an important part of the final solution.

*Map_form* is a function that is based on the modification of the map by means of morphological operations. The map must remain as unaltered as possible in order to always maintain the points through which the robot has passed or at least a very close approximation. These morphological operations are used for the method of intermediate paths in small areas and for the union of unconnected points due to the sampling of the mapping.

It has as input the initial map and it is obtained as output both the new map already modified and the map after the morphological operation *imerode* has been applied.
As already mentioned, the use of this function was not for too long, but it did cause some following changes that reformulate and lead the project to the final solution.

- **Num_nodos:** This function is in charge of applying a numerical label to each of the nodes on the map. This is done for the order in the cost matrix that is computed later. These labels are used afterwards to represent which nodes are connected via the rows and columns of the cost matrix. This function also creates a new map with only 1 value at the nodes location. This map is used for other functions and for visualization meanings.

  It should be remembered that the nodes are not only the points with more than 3 possible paths, but also the terminal points, as well as the initial and final points. This function numbers all of them.

  As inputs it is necessary to enter the dimension(s) of the map, the map, the starting point and the goal point.

  This function is used throughout the project, as the numbering of nodes is essential for the creation of the cost matrix and therefore for the application of the Dijkstra algorithm.

- **Path_connection:** Used for connection between roads that have a distance less than a limit. This connection is made in areas smaller than a previously defined limit, to later consider them safe and that the robot can move through them without danger and maintaining the safety of the patient, as well as reducing the cost of the road faster in certain situations.

  As inputs it is needed the map, its dimension(s) and the map after being applied the *imerode* function.

  It is used during the phase of more complex maps, when it is decided to implement an extension of the project on the paths that are little separated from each other. Later it is discarded, when it is applied on the real maps, it causes a
drastic change on the map, creating many nodes and causing much slowness in the computation of the code.

- **Tracking**: Basic and important function. Used for tracking and calculating edges costs. This function starts from the first node, detects all the possible paths of that node (using `calc_saltos`) and repeats in loop as many times as possible paths there are, tracking the edges point by point until finding another node, where it stops and applies the cost calculated in the cost matrix.

  This function has as inputs the map, the G map (only with the points that are nodes), the node matrix (which is updated during the execution of the function), the starting point and the goal point. As outputs, this function generates the final cost matrix and an information matrix applied to each node to know with which other nodes, in which direction and at what cost it connects.

  This function is used throughout almost the entire project. There is only a small amount of time in which an alternative solution to this function is proposed, since it has the problem that by following all the edges point by point, the computation time is prolonged, but finally it is essential for the solution.

### 1.1.2 Matlab functions

- **Bwdist**: It is a function that computes the Euclidean distance of the image, assigning to each pixel the value of the distance between it and the nearest non-zero pixel. If the own pixel has any value different from zero, that pixel will take the value of zero as itself is the closest pixel with non-zero value (Figure 3).

  This function is used in the transformations necessary for the subsequent work on the map and it is the alternative to the morphological operations functions.

  This function will be used for the final solution being easier to understand and analyse its behaviour, while discarding any dependence on the shape used.
It needs as input just the binary image and it computes the output with the distance value in each pixel. Alternatively, it can be added as input the method to use, which defines the kind of metric used: chessboard, cityblock, euclidean and quasi-euclidean.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3. How bwdist function works.

- **Bwdistgeodesic**: this function searches for geodetic distance over a binary image. In the case of a binary image, given a seed that will be the zero distance point, the points that cannot be reached will have a value NaN (Not a Number).

  This function is used during the development of the final result and in the maps coming from a real exploration. However, it is not implemented in the final result due to the lack of information and wrong results in the cost matrix given while using this function.

  This function needs as inputs the binary image in which the distances are calculated, and a mask that will be the same size as the image, and that will contain the values of 1 or true only in the points from where one wants to measure the distance (zero distance points or seeds).

- **Bwmorph**: This is another morphological operation, also applied to the transformation of the map. Bwmorph has different uses and applications, for instance, given a binary image as input, it can compute as output another binary image that only contains the edges (Figure 4). Another example, which will be used in this case, is to compute as output a binary image that only contains the central "skeleton" (Figure 5) of the true or value 1 elements of the initial binary image.
This is done by making a matrix of the same size as the initial image, and only giving the true value to the points that are equidistant to two or more edges of the element itself.

This function will be used in the final result and will be maintained as definitive.

Figure 4. Implementation of bwmorph using edges method

Figure 5. Implementation of bwmorph using skeleton method