Chapter 3

Mapping and Navigation

As already mentioned, the task the robot has to perform is to navigate through an unknown unstructured environment and reach a target landmark specified by a human operator. This task is not easy to solve, since it has to be carried out in a complex environment, and the target can be occluded by other objects. Purely reactive robotic systems would have problems trying to accomplish this task, since they do not build any model of the environment. If the target were lost, it would be difficult to recover its visibility and continue the navigation towards it. For this reason, we thought that the robot should build a map of the environment in order to navigate through it. The information stored in the map must permit the robot to compute its location, the location of the target, and how to get to this target. Although the objective of this PhD thesis is to develop a navigation system for indoor environments, we have used a map representation that also works outdoors, since this is the next milestone of the project in which we are involved. Thus, instead of using a grid-based approach, the most widely used approach for indoor environments, we have used a topological one, most appropriate also for outdoors.

Our approach is based on the model proposed by Prescott in [55]. The principles underlying this model are inspired by studies of animal and human navigation and wayfinding behavior. This model, called *beta-coefficient system*, does not only deal with how to represent the environment as a map, but also adds a mechanism for computing the location of landmarks when they are not visible, based on the relative positions of the landmarks. This mechanism is what we have used to provide the robot with orientation sense, since it captures the relationship among different places of the environment. The robot makes use of this orientation sense to compute the location of the target when it is occluded by other objects or obstacles.

In this chapter we firstly describe how Prescott's model works when the robot is able to have exact information about its environment, and then we explain how we have extended it to work with imprecise information. We also describe the method used for dividing the environment into appropriate topological regions, and finally how the topological map is used to navigate through the environment.

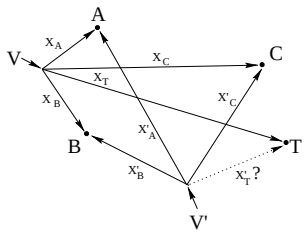


Figure 3.1: Possible landmark configuration and points of view. Landmarks A, B, C and T are visible from viewpoint V. Only landmarks A , B and C are visible from viewpoint V'

3.1 Beta-coefficient System

The idea behind Prescott's model is to encode the location of a landmark (which we refer to as target – not to confuse with the target or goal of the Navigation system) with respect to the location of three other landmarks. Having seen three landmarks and a target from a viewpoint (e.g., landmarks A, B and C and target T from viewpoint V, in Figure 3.1), the system is able to compute the target's position when seeing again the three landmarks, but not the target, from another viewpoint (e.g., V'). A vector, called the β-vector of landmarks A, B, C and T, is computed as

$$\beta = X^{-1}X_T \tag{3.1}$$

where $X = [X_A X_B X_C]$ and $X_i = (x_i, y_i, 1)^T$, are the homogeneous Cartesian coordinates of object i, $i \in \{A, B, C, T\}$, from viewpoint V. This relation is unique and invariant for any viewpoint if landmarks are distinct and non collinear. The target's location from viewpoint V' is computed as

$$X'_T = X'\beta, \tag{3.2}$$

where $X' = [X'_A X'_B X'_C]$.

It should be noted that, although Prescott's system works with Cartesian coordinates, once all the computations have been done, the resulting target's location is converted to polar coordinates, since, as will be seen in next chapters, this is the coordinate system that uses the Navigation system.

This method can be implemented with a two-layered network. Each layer contains a collection of units, which can be connected to units of the other layer. The lowest layer units are *object-units*, and they represent the landmarks the robot has seen. Each time the robot recognizes a new landmark, a new object-unit is created. The units of the highest layer are *beta-units* and there is one for each β-vector computed.

When the robot has four landmarks in its viewframe, it selects one of them to be the target, a new beta-unit is created, and the β-vector for the landmarks is calculated. This beta-unit will be connected to the three object-units associated with the landmarks (as incoming connections) and to the object-unit associated with the target landmark (as an outgoing connection). Thus, a beta-unit will always have four connections, while

an object-unit will have as many connections as the number of beta-units it participates in. An example of the network can be seen in Figure 3.2b. In this figure there are six object-units and three beta-units. The notation ABC/D is understood as the beta-unit that computes the location of landmark D when the locations of landmarks A, B and C are known.

This network has a propagation system that permits the robot to compute the location of non-visible landmarks. It works as follows: when the robot sees a group of landmarks, it activates (sets the value) of the associated object-units with the egocentric locations of these landmarks. When an object-unit is activated, it propagates its location to the beta-units connected to it. On the other hand, when a beta-unit receives the location of its three incoming object-units, it gets active and computes the location of the target it encodes using its β-vector, and propagates the result to the object-unit representing the target. Thus, an activation of a beta-unit will activate an object-unit that can activate another beta-unit, and so on. For example, in the network of Figure 3.2b, if landmarks A, B and C are visible, their object-units will be activated and this will activate the beta-unit ABC/D, computing the location of D, which will activate BCD/E, activating E, and causing BDE/F also to be activated. In this case, knowing the location of only three landmarks (A, B and C), the network has computed the location of three more landmarks that were not visible (D, E and F). This propagation system makes the network compute all possible landmarks' locations. Obviously, if a beta-unit needs the location of a landmark that is neither in the current view nor activated through other beta-units, it will not get active.

This propagation system adds robustness to the computation of non-visible landmarks, since a landmark can be the target of several beta-units at the same time. Because of imprecision in the perception on landmark locations, the estimates of the location of a target using different beta-units are not always equal. When this happens, the different location estimates must be combined. Prescott uses the size of the β-vector as the criterion to select one among them. A beta-unit with a smaller β-vector is more precise than those with larger β-vectors (see [55] for a detailed discussion on how to compute the estimate error from the size of the β-vector). The propagation system does not only propagate location estimates, but also the size of the largest β-vector that has been used to compute each estimate. When a new location estimate arrives to an object-unit, its location is substituted with the new one if the size of the largest β-vector used is smaller than that used for the last location estimate received.

The network created by object and beta units can be converted into a graph where the nodes represent triangular shaped regions delimited by a group of three landmarks, and the arcs represent paths. These arcs can have an associated cost, representing how difficult it is to move from one region to another. Although the arcs are created immediately when adding a new node to the graph, the costs can only be assigned after the robot has moved (or tried to move) along the path connecting the two regions. In the case the path is blocked by an obstacle, the arc is assigned an infinite cost, representing that it is not possible to go from one region to the other. This graph is a topological map, and we call its nodes *topological units*. An example of how the topology is encoded in a graph is shown in Figure 3.2c.

This topological map is used when planning routes to the target. Sometimes, when

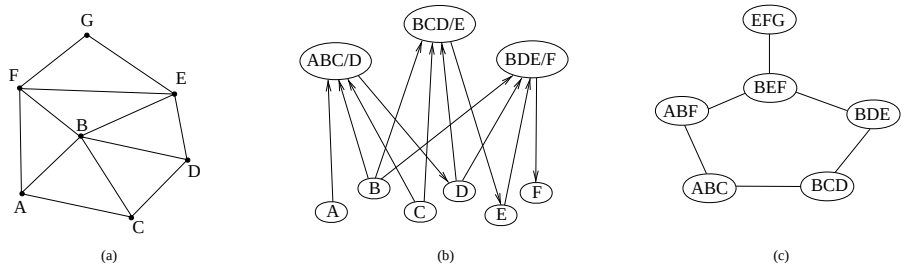


Figure 3.2: (a) Set of landmarks (b) associated network (partial view) and (c) associated topological map

the position of the target is known, the easiest thing to do is to move in a straight line towards it, but sometimes it is not (the route can be blocked, the cost too high...). With the topological map, a route to the target can be computed. In Section 3.4 a detailed explanation on how to compute routes to the target is given.

3.2 Extending Prescott's System: Moving to Fuzzy

The beta-coefficient system, as described by Prescott, assumes that the robot can compute the position of the landmarks with small errors, in order to create the beta-units and use the network. But this is never the case: the Vision system provides the robot with inexact information about the location of landmarks. To work with this imprecise information we use fuzzy numbers.

3.2.1 Fuzzy Numbers and Fuzzy Operations

A fuzzy number can be thought of as a weighted interval of real numbers, where each point of the interval has a degree of membership, ranging from 0 to 1 [7]. The higher this degree, the higher the confidence that the point belongs to the fuzzy number. The function $F_A(x)$, called *membership function*, gives us the degree of membership for x in the fuzzy number A.

Before defining the arithmetic with fuzzy numbers, we have to introduce the concept of α-cut. The α-cut ($\alpha \in [0,1]$) of a fuzzy number A, is the interval $\{A\}_\alpha = [a_1, a_2]$ such that $F_A(x) >= \alpha, \forall x \in [a_1, a_2]$.

Let A and B be fuzzy numbers, and $\{A\}_\alpha$ and $\{B\}_\alpha$ α-cuts. The fuzzy arithmetic operations are defined as follows,

$A + B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \oplus \{B\}_\alpha\ \forall\alpha$
$A - B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \ominus \{B\}_\alpha\ \forall\alpha$
$A \times B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \otimes \{B\}_\alpha\ \forall\alpha$
$A \div B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \oslash \{B\}_\alpha\ \forall\alpha$

where the operations $\oplus, \ominus, \otimes$ and $\oslash$ are performed on intervals and are defined as

$[a_1, a_2] \oplus [b_1, b_2] = [a_1 + b_1, a_2 + b_2]$
$[a_1, a_2] \ominus [b_1, b_2] = [a_1 - b_2, a_2 - b_1]$

$[a_1, a_2] \otimes [b_1, b_2] = [min(a_1b_1, a_1b_2, a_2b_1, a_2b_2), max(a_1b_1, a_1b_2, a_2b_1, a_2b_2)]$
$[a_1, a_2] \oslash [b_1, b_2] = [a_1, a_2] \otimes [\frac{1}{b_2}, \frac{1}{b_1}],\ 0 \notin [b_1, b_2]$

3.2.2 Fuzzy Beta-coefficient System

To use the beta-coefficient system with fuzzy numbers, we simply perform the calculations described in the previous section using the fuzzy operators defined above. However, because of the nature of fuzzy operators, some landmark configurations may not be feasible (the matrix inversion used for computing the β-vector – Equation 3.1– may produce a division by 0), so not all configurations can be stored in the network.

When using the network to compute the position of a landmark, we obtain a fuzzy polar coordinate (r, ϕ), where r and ϕ are fuzzy numbers, giving us qualitative information about its location. An advantage of working with fuzzy coordinates is that it gives us information about how precise the location estimate is, since it represents the location not as a crisp coordinate, but as a spatial region where the landmark is supposed to be.

Another difference with Prescott's model is the criterion used to select among different estimated locations for the same landmark. In our extended system, instead of looking at the size of the β-vectors, we use the imprecision of the estimated location itself. The imprecision of a landmark location, $I(l)$, is computed by combining the imprecision in the heading and in the distance as follows. $I_h(l)$ is the imprecision in heading, and it is defined by taking the interval corresponding to the 70% α-cut of the fuzzy number representing the heading to the landmark (see Figure 3.3). This imprecision is normalized dividing it by its maximum value of 2π. Similarly, $I_d(l)$ is the imprecision in distance, and it is defined as the 70% α-cut of the fuzzy number representing the distance. It is normalized by applying the hyperbolic tangent function, which maps it into the $[0, 1]$ interval. Finally, the two imprecisions are combined according to:

$$I(l) = \lambda \cdot \tanh(\beta \cdot I_d(l)) + (1 - \lambda) \cdot \frac{I_h(l)}{2\pi} \tag{3.3}$$

where λ weighs the relative importance of the two imprecisions, and β controls how quickly the transformed I_d approaches 1. In our experiments, we set $\beta = 1$ and $\lambda = 0.2$. When an object-unit receives a new location estimate, it computes the imprecision of this estimate, compares it with the imprecision of the current location estimate, and keeps the least imprecise location.

3.3 Building the Map

In Section 3.1 we mentioned that when the robot has four landmarks in its viewframe, it creates a new beta-unit for them. However, with four landmarks, there are four candidates to be the target of the beta-unit. Moreover, if the robot has more than four landmarks in the viewframe, there are many possible beta-units to be created. More precisely, if there are n visible landmarks, there are $\binom{n}{4} \cdot 4$ candidates for being new beta-units. However, it is not feasible to store them all, firstly because of the huge num-

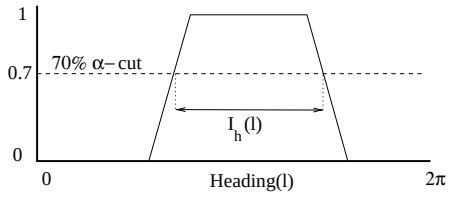


Figure 3.3: Computation of the imprecision of the heading toward landmark l as a fuzzy number

ber of combinations, and secondly, and more important, because some configurations are better than others. Thus, some selection criterion must be used.

Before describing the criterion we have used, we explain how the obstacles are represented in the map. We differentiate two types of obstacles: *point* obstacles and *linear* obstacles. *Point* obstacles are those the robot can easily avoid by slightly modifying its trajectory, since they do not completely block the path. In our indoor environment such obstacles are boxes and bricks. In outdoor environments they could be small rocks, trees, etc. These obstacles do not affect the global navigation, as the Pilot can tackle them alone, so the Navigation system does not take them into account and they are not stored in the map. On the other hand, *linear* obstacles are long obstacles that completely block the path of the robot. They can also be avoided by the Pilot, but the trajectory has to be drastically modified. In our indoor environment we use several bricks to form these obstacles. In an outdoor environment these obstacles could be fences, walls, groups of rocks, etc. Since these obstacles do highly affect the navigation task, they have to be represented in the map, so that they are taken into account when computing routes to the target. The information about these obstacles is stored on the arcs of the topological map. An arc is labelled with an infinite cost to indicate that there is an obstacle between the two regions connected by the arc. Notice that with this representation we can only represent those obstacles placed along the line connecting two landmarks. Although in our experiments we have designed the environments so that they satisfy this condition, the system would also work if it were not satisfied. However, in this latter case, the Navigation system could not take all the obstacles into account, and thus, its performance would be worse. The arcs' labels are updated whenever the Pilot system informs about the presence of an obstacle between two landmarks.

Going back to the selection criterion, given a set of landmarks, for which their location is known, we seek to obtain a set of triangular regions with the following constraints:

- *Low collinearity*: the collinearity of a region is computed as

$$Col(R) = 1 - \frac{\alpha\beta\gamma}{(\frac{\pi}{3})^3} \tag{3.4}$$

  where α, β and γ are the three angles of the triangular region. The best quality is associated to equilater triangles, where $\alpha = \beta = \gamma = \frac{\pi}{3}$, and hence their collinearity is 0. When one of the angles is 0, landmarks would be maximally

collinear and $Col(R) = 1$. The higher the collinearity, the higher the error on the computation of the β-vector and landmark locations (see [55] for a detailed explanation). Therefore, the regions with lower collinearity are preferred. For example, in Figure 3.4 the two regions on the right are preferred over the two on the left, since the region ABD is too collinear.

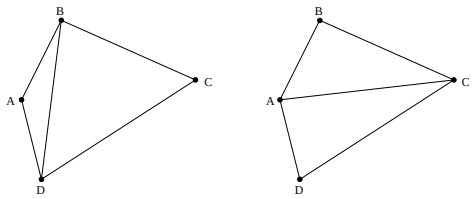


Figure 3.4: *Left*: bad set of regions; region ABD is too collinear. *Right*: good set of regions

- *Connectivity*: the set of regions must be converted into a graph with a single component, so that there is a path between any two nodes of the graph. In Figure 3.5, the set of regions on the left is not acceptable, since there are two disconnected components, whereas in the set on the right all the regions are connected.

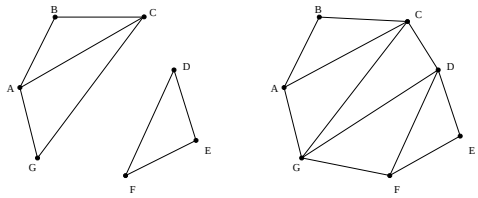


Figure 3.5: *Left*: bad set of regions; there are two disconnected components. *Right*: good set of regions

- *Convex hull covering*: the regions must cover the convex hull of the set of landmarks, so that the environment is represented completely, with no unrepresented regions. In Figure 3.6, the set on the left is not acceptable, since the region DFG is not represented.

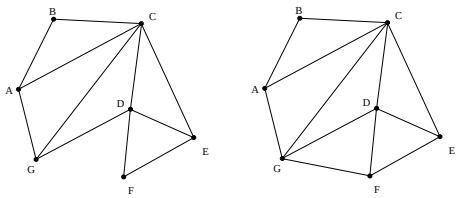


Figure 3.6: *Left*: bad set of regions; region DFG is missing. *Right*: good set of regions

- *Non overlapping*: the regions should not overlap with each other. If this were the case, the robot could be in more than one region at the same time, which could cause some problems when computing routes to the target. For instance, if the robot were in the overlapping area of the two regions, it would make no sense to order the robot to move from one region to the other, since it would already be inside both regions, and the order would not have any effect. Moreover, if one of the overlapping edges is an obstacle, the path from one side of the adjacent region to the other side would be blocked, which is obviously a bad representation of the environment, since the robot must be able to move around the whole space of a region. In Figure 3.7, the set of regions on the left is a bad set, since part of the obstacle between landmarks B and D lies inside the region ADC. In this case, the associated graph would have two nodes, ABD and ACD, which would be connected, so the robot would think that it can move from region ABD to region ADC, but it would find the path blocked because of the obstacle.

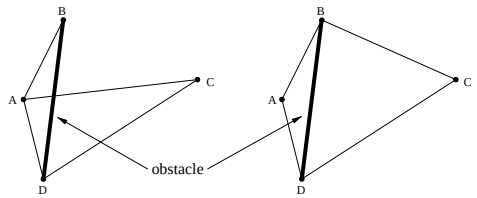


Figure 3.7: *Left*: bad set of regions; the obstacle between landmarks B and D is inside the region ACD. *Right*: good set of regions

- *Keep obstacles*: if an edge of a region is marked as an obstacle, this edge must be kept in the map, even if it causes the robot to keep high collinear regions. The obstacle edges are the only ones that cannot be removed from the map. If we did so, the information about the location of obstacles would be lost and would not be taken into account when computing routes to the target.

To compute the optimal set of regions for a given set of landmarks, we have developed an incremental algorithm that treats landmarks one by one to update the map. However, the algorithm only starts working when the locations of at least four landmarks are known, since this is the number of landmarks needed to create a beta-unit. With these four landmarks, the mapping algorithm computes the best set of regions according to the constraints given above. Then, the rest of visible landmarks, if any, are added one by one to the already built map. When adding a new landmark to the map, two situations can happen: (1) the landmark is inside an already existing region, or (2) the landmark is outside any region. In the first case, the region containing the new landmark is replaced by three new regions (see Figure 3.8). In the second case, all the possible new regions are created (see Figure 3.9). No matter the situation of the landmark, once the new regions have been created, the algorithm checks if the resulting map is still optimal. This optimization consists of analyzing each pair of adjacent regions and checking if their configuration is optimal according to the constraints. If it

finds that some regions could be changed so that a better configuration is obtained, it does so. An example of this step by step updating is shown in Figure 3.10.

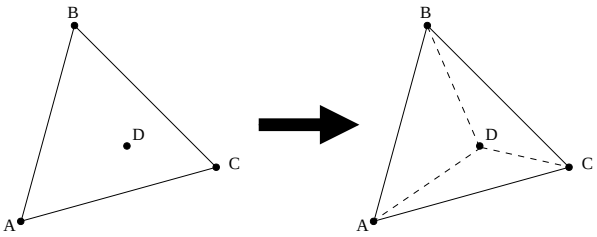


Figure 3.8: Adding a new landmark (D) located inside an existing region (ABC) resulting in the substitution of the original region for three new regions (ABD,ACD,BCD)

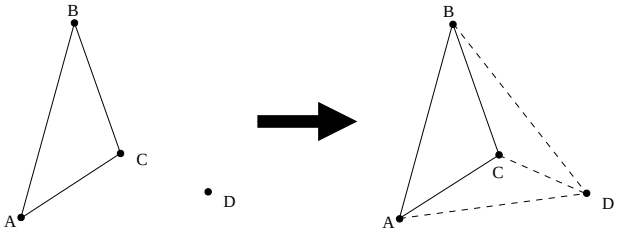


Figure 3.9: Adding a new landmark (D) located outside any existing region resulting in the addition of two new regions (ACD,BCD)

Once the set of regions is computed, new beta and topological units can be created. For each new region a beta-unit is created for each region adjacent to it, taking the three landmarks of the first region as the encoding landmarks, and the landmark of the second region that is not in the first one as the target. In other words, for each pair of adjacent regions, two "twin" beta-units are created. An example can clarify this explanation: with the regions ABC and ACD shown on the right in Figure 3.4, the beta-units ABC/D and ACD/B would be created. One topological unit is also created for each new region, and the graph is updated according to the adjacency of regions. Initially, the arcs are labelled with a default cost of 1, and they are changed to ∞ whenever an obstacle is detected. The topological units corresponding to regions that are not used any more are removed from the graph. However, beta-units are never removed, since they add robustness to the system, as in Section 3.1.

This triangulation algorithm needs the location of the landmarks to be known (either recognized by the Vision system or computed by the beta-coefficient system). However, not all landmark locations can always be known. The algorithm only takes into account those landmarks whose locations are known. This ensures that the five constraints explained above are satisfied only for the located landmarks. When one of the unlocated landmarks is seen or computed, some constraints might become unsatisfied. Whenever any constraint is broken, the map is rebuilt in order to satisfy again all the constraints. This constraint break can also be caused by the fuzziness of the locations. Because of

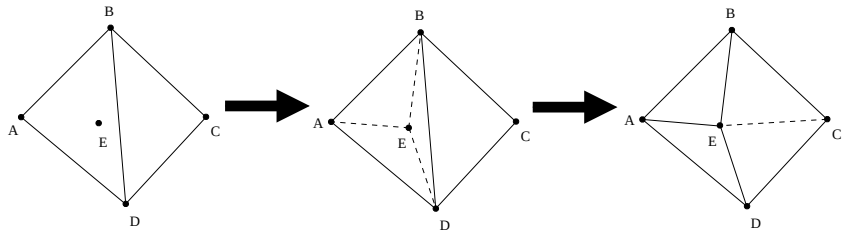


Figure 3.10: Adding a new landmark (E) into a map with two regions (ABD and BCD): first, region ABD is substituted for three new regions (ABE,ADE,BDE); after that, optimization for regions BCD and BDE is performed and they are substituted for the new regions BCE and CDE

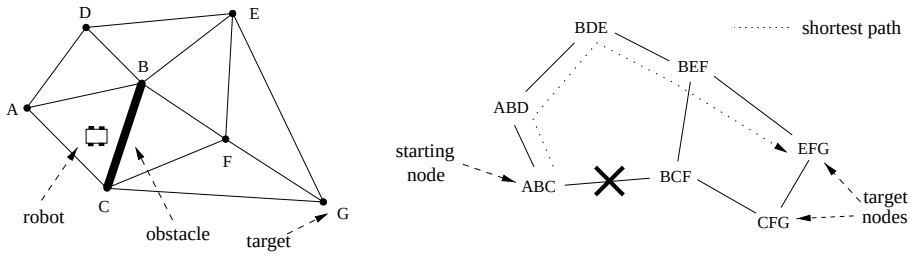
the imprecision of the locations, the map can suddenly be breaking some of the constraints. To avoid having an inconsistent map, every once in a while the satisfaction of the constraints is checked, and, if needed, the map is rebuilt.

3.4 Navigating Through the Environment

The beta-coefficient system described above provides the means for computing the location of a target even if it is not visible. This is very useful if the robot is navigating in an environment with a high density of landmarks and obstacles that occlude the target. In this case, the robot is able to go towards the target by seeing other landmarks. However, in some cases the obstacles might be blocking the direct path to the target. In this case, knowing the location of the target is not enough and an alternative route to reach it must be computed using the topological map.

Although a route consists of a sequence of regions the robot should navigate through in order to reach the target, only the first region is taken into account. The reason for doing so is that since the environment is never fully known, the robot cannot commit to a given route because it might encounter new landmarks and obstacles that would change the shape of the map, and possibly, the route to the target. Therefore, hereafter, instead of talking about routes, we will talk about diverting targets. A diverting target can be: (1) an *edge* between two landmarks, which the robot has to cross in order to go from one region to another, or (2) a *single landmark* to which the robot has to approach.

When the system is asked for a diverting target in order to reach another target, it first finds out in which region the robot is currently located, using the information about the landmarks whose location is known. This region will be the starting node on the topological map. The shortest path from this node to any of the nodes containing the target landmark (a landmark can be component of several topological regions) is computed. The edge connecting the current region with the next one on the shortest path will be the diverting target. The edge is given as a pair of landmarks, one that has to be kept on the left hand side of the robot and another to be kept on the right hand side, so the robot knows which way the edge has to be crossed. An example is shown in Figure 3.11. In this case, the robot is in region ABC, the target is G, and the shortest



Real environment Topological map

Figure 3.11: Diverting target computation

path to the target would be {ABC,ABD,BDE,BEF,EFG}. Thus, the diverting target would be the edge AB.

However, it could happen that there is no such shortest path. The cases in which such path does not exist are the following:

- The robot is not in any topological region.
- The cost of the shortest path is infinite. This means that the path is blocked by an obstacle, so it is not a valid path.
- The target is not found in any topological region.

To solve the first two cases, the map has to be enlarged with virtual regions through which the robot can navigate. The idea is to let the robot move in an unknown area outside the map. The virtual regions are built by placing some virtual landmarks around the existing map, and creating the appropriate regions using the same algorithm as described in the previous section. An example of these virtual regions is depicted in Figure 3.12. To force the robot to use regions of the original map, the arcs connecting virtual regions are labelled with a high cost (though not infinite), so that they are used only if it is absolutely necessary. With this enlarged map, the shortest path is computed again. However, it can be that the edge to be crossed contains one virtual landmark. In this case, the edge cannot be given as the diverting target, since the virtual landmarks do not exist on the real environment and cannot be tracked. In this situation, the direction to the middle point of the edge is computed and given as the diverting target. We assume that there is always some free space around the explored area, so that the regions created with the virtual landmarks can be traversed.

In the case the target is not in any topological region, there is no way to compute which should be the next region to visit, since there is no destination node. When this happens, the diverting target is set to any of the visible landmarks, hoping that on the way to this diverting target, the map is updated and the target for which a diverting target has been computed is incorporated into it.

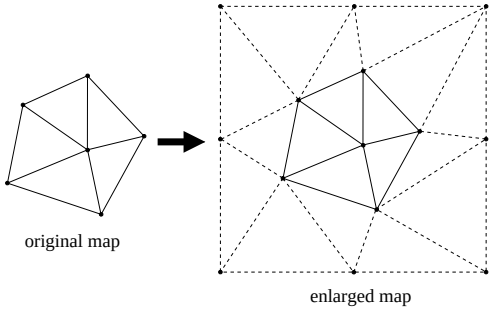


Figure 3.12: Enlarging the map with virtual regions (dotted lines)

3.5 Future Work

Although the extension of Prescott's method, together with the algorithms to compute diverting targets, is enough for permitting a robot build a map and navigate through an unknown environment, we would like to explore other mapping methods, so that the combination of the different methods adds robustness to the Navigation system. With the current mapping method, the robot needs to see at least three landmarks in order to be able to use the information stored in the map. We would like to develop some other mapping methods to cope with the situations in which the robot has very little information (i.e. less than three landmarks). These methods would be even more qualitative than our fuzzy extension of Prescott's method. We could, for example, look at the field of Spatial Cognition, which works with spatial relationships such as "landmark X is at the left hand side of the line connecting landmark Y and landmark Z".