University of Padua  Faculty of Engineering

Image-based localization for mobile robots in dynamic environments

Supervisor: Prof. *Enrico Pagello*

Co-supervisor: Prof. *Stefan Wermter*

Student: *Nicola Bellotto*

Laurea in ELECTRONIC ENGINEERING

Department of INFORMATION ENGINEERING

Academic Year 2003/2004

Alla mia famiglia

SOMMARIO

La tesi in oggetto tratta l'implementazione di un sistema di localizzazione per robot mobili in ambienti dinamici. Per svolgere tale compito, l'unica informazione utilizzata proviene dall'odometria e da una semplice videocamera unidirezionale. La posizione stimata è relativa ad una mappa topologica fornita dall'utente. L'approccio qui descritto è basato fortemente su un nuovo algoritmo di confronto immagini per riconoscimento di luoghi, il quale è assolutamente indipendente dalla presenza di particolari riferimenti, siano essi naturali od artificiali. Facendo uso dello stesso algoritmo, si creano immagini panoramiche dalle posizioni prescelte. Tali immagini, insieme ad alcune informazioni metriche relative ai loro punti origine, formano la mappa utilizzata dal robot per localizzarsi. L'ambiguità nell'informazione utilizzata per riconoscere luoghi differenti viene gestita da un sistema di ipotesi multiple con una procedura di selezione basato sulla localizzazione di Markov. In questo modo si riesce a far fronte a casi di "perceptual aliasing" o di totale assenza di informazione affidabile dai sensori. Il sistema di localizzazione in questione è stato implementato su un robot reale. Esperimenti condotti in un classico ufficio dimostrano la robustezza dell'approccio anche in casi di ambienti dinamici.

To my family

ABSTRACT

This thesis is concerned with the implementation of a localization system for mobile robots moving in dynamic environments. To accomplish this task, the only computed information comes from odometry and from a simple unidirectional camera. The estimated position is a topological location on a map provided by the user. The approach here described is strongly based on a new image matching algorithm for place recognition, which is absolutely independent from the presence of particular landmarks, either natural or artificial. Panoramic images of the topological places are reconstructed making use of the same algorithm. These images, together with some metrical information relative to their origins, form the map that the robot utilizes to localize itself. The ambiguity of the information used for recognizing different places is resolved with a multiple-hypotheses tracking and a procedure of selection inspired by Markov Localization. In this way, the system can deal with cases of perceptual aliasing or total absence of reliable sensor information. The localization system has been implemented on a real robot. Experiments carried out in a typical office scenario demonstrate the robustness of our approach even in case of dynamic environments.

1 INTRODUCTION

In robot navigation a fundamental role is played by the localization system, since it is necessary for every kind of path planning. This consists in providing the robot with the capability to identify its current position in the environment with respect to a certain point of reference. Before introducing the localization system described in the following chapters, we would like to present a taxonomy normally adopted by the research community to classify the numerous existing approaches and introduce some of the recent works in this field.

1.1 The localization task

In [Fox98] three main parameters are taken into account for evaluating the localization performances:

1) how the robot's position is represented inside the environment with respect to a particular fixed reference;
2) the kind of environment that the robot has to deal with;
3) the way the robot interacts with the environment to improve the capability of localizing itself.

The first point above refers to the distinction between *local* and *global* localization. In the local approach, the robot is normally supposed to localize itself with respect to an initial, known position. This kind of problem, also called "position tracking", is often solved using techniques based on Kalman filtering, which gives remarkable results in case of sufficient information from the sensors [GBFK98]. On the other hand, a global localization is able to recover the robot's location without knowledge of the initial position. This feature is not important only at the beginning but also during the localization process, because it means also recovering the correct position when the estimation of the previous one came out to be completely wrong, therefore dealing with the so-called "kidnapped robot" problem.
The second point takes into account an important property of the explored environment: it may be *static* or *dynamic*. In a static environment most of the features, or at least those directly observable by the robot, do not change over time. This facilitates the task of the localization system because it implies the perceived changes to be only dependent on the robot's position. Naturally this assumption is often false, since typical situations the robot has to deal with are presence of people,

different disposition of the furniture, doors open or closed, etc.; shortly, the localization must be reliable also in dynamic environments.
The third point mentioned above is related to the extension of a *passive* localization system, which does not influence the robot's motion, to an *active* localization, where this means having also a direct control on the navigation. The reason is to improve the localization process moving towards places where the robot can perceive additional data and eventually resolve situations of ambiguity.
The way explained so far to classify the different approaches highlights in particular the performances of the localization system. A further extension, which takes into account also the several methodology adopted to resolve this task, is very well described in [FM03]. Here the localization strategies are divided in three main groups and for each group there is a further distinction in other three categories, for an overall classification of nine localization types. The strategies are identified as follows:

1) *direct position inference* – this is the capability of the robot to estimate its current position relying only on the current observations and without taking into account the past history. Being totally independent from any previous position, this method permits a global localization but fails completely in case of *perceptual aliasing* (i.e. when two or more observed location look the same).

2) *single-hypothesis tracking* – this strategy keeps always track of the previous estimated position using also the relative displacement given by the internal readings (i.e. odometry information). Even if this in general helps to resolve situations of perceptual aliasing, the localization could fails in case the previous position is completely wrong. The methods based on Kalman filter can be inserted in this category.

3) *multiple-hypothesis tracking* – such solution directly derives from the previous strategy. Here, instead of keeping track only of one position, the robot considers the possibility to have moved from several locations and then takes into account all of them to get an estimation of the current one. In practice the localization system always keeps track and updates in parallel a set of hypothetical positions, identifying time by time the most probable.

As we said, for each of the strategies above there is an addition sub-division. This is strictly related to the kind of map used for the localization and how the robot's position is represented within the map. We can then distinguish three different combinations "map type" / "position type":

a) *metrical map / metrical position* – the map represents the environment using Cartesian coordinates to describe the shapes of the involved objects and the spatial relations among them. The map is normally a plan of the environment in form of CAD drawing. The robot's position is also represented with two coordinates (eventually plus its heading direction) in the same Cartesian frame.

b) *topological map / metrical position* – in this case the map contains only a finite number of locations, eventually connected by links that represent the possible transition paths between two different places. The topological map can be

thought as a graph, where each location is a node (vertex) connected by links (edges). All the nodes contain information about the surrounding environment, so as it can be perceived standing on that point, plus their relative coordinates with respect to a common frame of reference. These coordinates are used by the localization system for inferring a more precise metrical position.

c) *topological map / topological position* – this last representation makes use again of a topological map but, differently from the cases above, consider also the robot's position laying on its nodes. This combination is useful when the application does not require an exact position in terms of centimeters. The attention instead is directed just on recognizing the "area" where the robot is currently moving. Although this topological map does not necessarily contain the coordinates of its nodes, in some case this metrical information might come useful for computations involving odometry.

A fourth combination, *metrical map / topological position*, is normally not considered, since the loss of precision in localization resulting from the conversion to a topological representation is not compensated for.

1.2 Overview of localization systems

In the last years there has been a growing number of real robot applications where the localization was an essential part of the navigation system. Some of the most famous examples are the tour-guide robots RHINO [TBB99] and MINERVA [BCF99], or the robot-waiter ALFRED [MMA99]. They used different approaches and different sensors for localizing themselves. In [TBB99] perceptions were based just on proximity sensors (sonar and laser), while in [BCF99] they mad use of laser plus an additional camera directed towards the ceiling (so the observed scene was mostly static). The robot of [MMA99], instead, used artificial landmarks to recognize places of interest.
Other several localization approaches making use of vision have been presented in recent years. In [GM00] the robot was equipped with an unidirectional camera pointing ahead to the floor; the localization was performed thanks to the basic hypothesis that the floor had an uniform texture so, after camera calibration, it was possible reconstructing from the images a local map. The localization was the result of the comparison between the current local map and a pre-recorded global map.
The solution of [XYOH03] was based instead on a natural landmark model and a robust tracking algorithm. The landmark model contained sets of three or more natural lines such as baselines, door edges and linear edges in tables or chairs. The localization depended on an algorithm that allowed the robot to determine its absolute position with a view of a single landmark in one image.
Other recent approaches made use of Monte Carlo Localization. It has been demonstrated that this technique is reliable and, at the same time, keeps the processing time low. Indeed, Monte Carlo localization has been successfully applied in the RoboCup four-legged league, where the Sony dog's hardware has critical limitations. For example in [EFR01] they implemented a Monte Carlo approach for

vision-based localization that made use of sporadic features, extracted from the images of the robot's unidirectional camera. The probability of being in a certain location was calculated against an internal model of the environment within the robot moves. The experiments proved that the method was reliable enough, even with a restricted number of image samples, and improved drastically increasing the number of features. Some tests in a typical office environment seemed also promising.
Another application of Monte Carlo Localization in the RoboCup context is described in [Pre03; MPP04]. In this case the video input came from an omni-directional sensor and the images were processed in a way to simulate a laser scanner, using the distances from points with colour intensity transitions. Even here the localization system made use of an internal representation of the football field.
An omni-directional camera was also the sensor used for the topological localization in [UN00]. Here they present an appearance-based place recognition that used only panoramic vision, without any odometry information. Colour images were classified in real-time based on nearest-neighbour learning, image histogram matching and a simple voting scheme.

Independently from the sensors used to perceive the world, innumerable systems have been also presented to resolve the ambiguity that arises from such perceptions. No sensor reading indeed is immune from noise and errors, both coming from the sensor itself and from the surrounding environment. A wide range of localization systems have been tested and compared in the works of [GBFK98; GF02; KJ03], covering methods based on Extended Kalmn Filter (EKF), Markov Localization (ML) alone or combined with the first one (ML-EKF), Monte Carlo Localization (MCL) and Multi Hypotheses Localization (MHL). The results of such experiments are a good starting point for choosing the most suitable localization approach, depending on our own application. They are also useful for getting precious suggestions and ideas on how developing new methods.

1.3 Current application

Now that we have a general view of the localization problem, we can go further introducing the particular application for which our system has been developed. The main objective was implementing a map-based localization system for indoor environments. It had to be eventually integrated in a more complex navigation architecture where the robot is able to perform tasks typical of a waiter or tour-guide scenario. This idea arose from some similar successful experiences [MMA99; BCF99; TBB99]. The localization module had to deal therefore with the presence of people and general changes of the environment. The task was particularly challenging because our robot, an ActivMedia PeopleBot, is provided just with a normal color camera and sonar sensors. Unfortunately, since the beginning these latter came out to be very unreliable in our environment because of many kinds of surface that could not be correctly detected. Furthermore, an accurate map of the environment was not available and occasionally some furniture, which would have influenced the sonar readings, had to be moved. The choice then was developing an image-based localization system with the only additional information provided by the robot's odometry.

In our application, it was not necessary to know exactly the metrical position. Instead, a topological localization was the most appropriate solution for a robot-waiter or tour-guide scenario. We implemented then a system that, given a topological representation of the environment, can estimate the area where the robot is currently moving. Our method is strongly based on a new place recognition algorithm that does not need any specific landmark. The same algorithm is also used for reconstructing panoramic images from the place of interest, combining a sequence of snapshots taken with the normal camera. Such images, together with approximate coordinates of the topological locations, form the map used by the robot. The place recognition process is followed by a procedure that resolves cases of perceptual aliasing or absence of reliable sensor information. The system keeps track of a hypotheses' set and for each update step chooses the most probable, with an approach inspired by Markov Localization. From the experiments, carried out in a typical office scenario, this method seems to be quite robust even in case of dynamic environments.

1.4 Structure of the thesis

The following chapters are organized as follows.

2 PLACE RECOGNITION

The algorithm for image-based place recognition is explained in detail. Then we describe how the same procedure is applied to reconstruct panoramic images. Successively we derive a method to recover the heading direction of the robot from such images. The chapter terminates with the use of digital zoom for enhancing the place recognition performances.

3 MULTI-HYPOTHESES TRAKING

This part starts with a brief introduction to Markov Localization theory. Then it is followed by the assumptions imposed in our implementation and the description of the update rules for the tracked hypotheses. We also try to give an intuitive explanation to the method adopted. At the end there are some consideration about the odometry correction and finally the overall localization algorithm is illustrated.

4 SOFTWARE IMPLEMENTATION

This chapter describes the practical implementation of the localization system. First of all we describe the software library that contains the algorithms previously illustrated. Then we explain how the system has been integrated in the robot's behavioural framework.

5 EXPERIMENTS AND RESULTS

Here we present several experiments conducted in a typical office environment and relative results. The first group of tests are concerned with the performances of the place recognition algorithm. The second is related to the overall system, with experiments including several types of environment and with analyses of the cases where the localization fails.

6 CONCLUSIONS

In this last part there is a critical evaluation of the implemented localization and recommendations for future works.

2 PLACE RECOGNITION

In this chapter we describe a new method to recognize a place among a finite set of possible places. This set is basically a *topological map* of the environment, provided by the user. Each place is identified by a point in the Cartesian space and by a panoramic image of the scene observed from such point. The procedure is mainly based on the comparison between a new image, taken from the camera, and all the panoramic images of the map. For each comparison, a value is assigned, which is a measure of the match's quality. The comparison is done using a new *image-matching algorithm*, that for simplicity we will call *IMA*. The entire procedure of comparison on the whole set of panoramic images constitutes the *place recognition*.

2.1 Image matching algorithm

In an indoor environment, the most relevant changes are due to objects or people moving with respect to a horizontal plane. A person walking, a chair moving, a door opening or closing: all these examples can be thought as "columns" moving horizontally along an image of the original scene. The algorithm described in this section arises from this simple consideration.
The main idea is to divide the new image into several column regions that we will call "*slots*" and then compare each of them with a stored image of the original scene. The measure of the similarity between a slot of the new image and a region of the stored image is given by the Normalized Correlation Coefficient, simply called *NCC* (see appendix A)
The image-matching algorithm, or *IMA*, can be divided into two steps. In the first one, every slot of the new image is shifted and matched inside the stored image. The matching values calculated are then used, during the second step, to determine the position where the match is maximum. The procedure is better described[1] in the following paragraphs.

[1] For simplicity, error cases, like image over-bound or array overflow, are not considered here. They will be discussed and treated on chapter 4.

2.1.1 First step: slots matching

Let us consider the new image I_{new}, single channel, of width W_{new}. This is divided into N_s slots; each of them has width $W_{slot} = W_{new} / N_s$. We refer to one of them as $slot_n$, with $n = 1, \ldots, N_s$. Then consider a reference image I_{ref}, also single channel, of width $W_{ref} \geq W_{new}$. The images I_{new} and I_{ref} have the same height. We also refer to a region of I_{ref}, delimitated by the pixel columns c_{left} and c_{right}, as $I_{ref}[c_{left}, c_{right}]$. The columns c_{left} and c_{right} belong to this region. In Figure 2.1an example is shown.

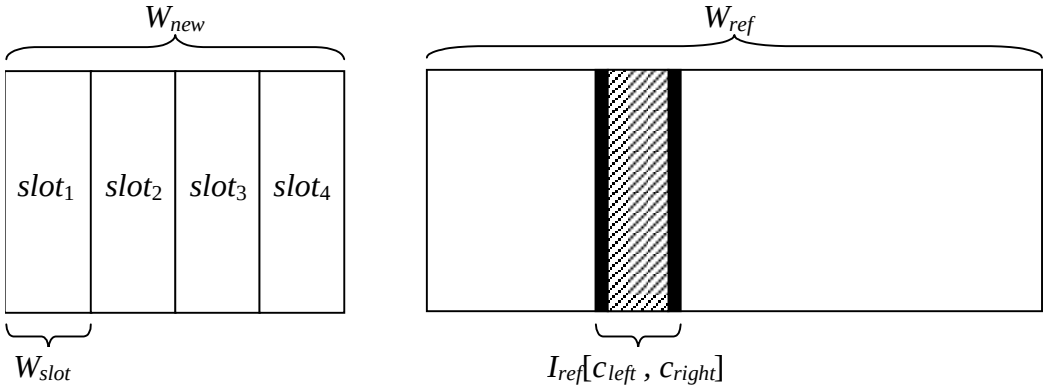


Figure 2.1 Examples of I_{new} (divided into four slots) and I_{ref}

First of all, we need to clarify how the *NCC* matching function works (see appendix A for a mathematical description). Given a source image I_{ref} and a template image $slot_n$, it compares $slot_n$ with a region $I_{ref}[c, c + W_{slot} - 1]$. The comparison is repeated for each column $c = 1, \ldots, W_{ref}$ After each comparison, a value[2] between 0 and 1.0 is stored inside an array *VAL* of length W_{ref} (see Figure 2.2). So, if the width of the slots is $W_{slot} = 10$ and $VAL[5] = 0.7$, it means the similarity between the current template $slot_n$ and $I_{ref}[5, 15]$ measures 0.7.

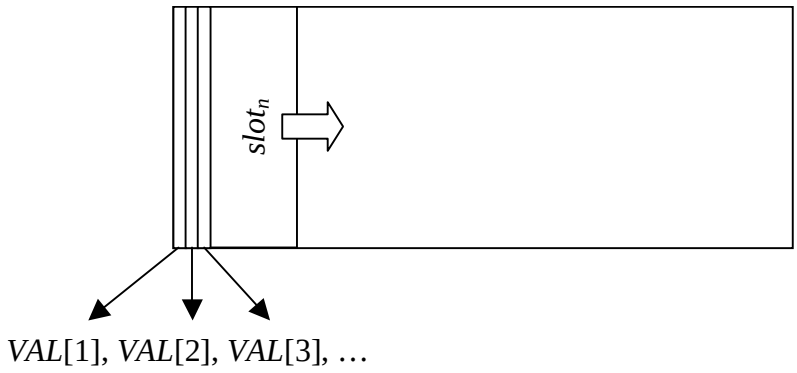


Figure 2.2 Slot of I_{new} shifted along I_{ref} and compared with *NCC*

[2] The original *NCC*, actually, calculates a value between −1 and +1, but in our implementation it is modified so that its output varies between 0 and 1, as described in appendix A.

The first part of *IMA* works as follow. At the beginning, two arrays, *MATCH_SLOT* and *MATCH_VAL* are created and initialized. The length of both the arrays is W_{ref}. The *NCC* function is used to calculate the matching values. At the end, *MATCH_SLOT* will contain the indices *n* of the slots that match best, while *MATCH_VAL* will contain the relative matching values. That is, if $slot_3$ is that one which matches best on the region $I_{ref}[5, 15]$, then *MATCH_SLOT*[5] = 3 and *MATCH_VAL*[5] contains the relative matching value. The procedure is described in Code 2.1.

Code 2.1 First part of *IMA* (slots matching)

```
MATCH_SLOT[W_ref] = {0, …, 0}
MATCH_VAL[W_ref] = {0, …, 0}

for n = 1 to N_s
    VAL = NCC(I_ref , slot_n)
    for c = 1 to W_ref
        if VAL[c] > MATCH_VAL[c] then
            MATCH_SLOT[c] = n
            MATCH_VAL[c] = VAL[c]
        end if
    end for
end for

END
```

2.1.2 Second step: best matching

As previously said, in the second step of *IMA*, the position and the value of the best match are extracted. Basically, this consists in finding the best sequence of slots with the highest matching value. The procedure compares a mask $\{1, 2, \ldots, N_s\}$ along the whole array *MATCH_SLOT*. The mask is a sequence of slot indices, at distance W_{slot} each other, as shown on Figure 2.3. The comparison is done "positioning" the beginning of the mask on each element of *MATCH_SLOT*. Every comparison starts resetting a variable *SUM*. If an element *n* of the mask and the relative i^{th} element of *MATCH_SLOT* are equal, then the i^{th} value of *MATCH_VAL* is added to *SUM*. At the end of each comparison, *SUM* contains the total matching value, given by the slots in the right order and position.

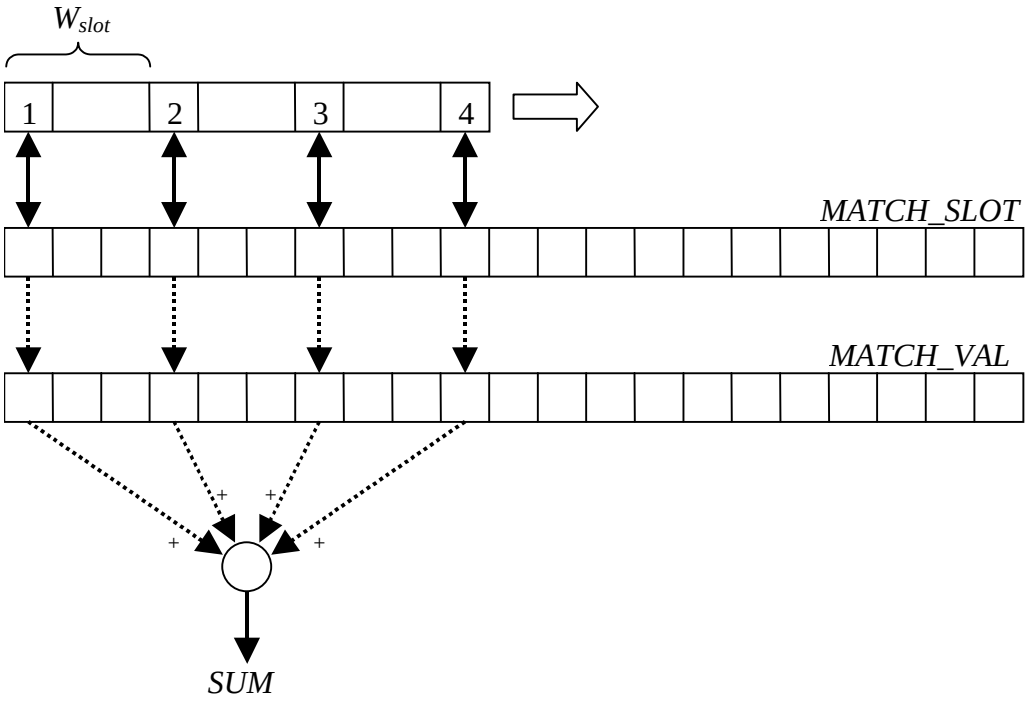


Figure 2.3 Comparison scheme of the best matching extraction

Below, Code 2.2 describes the procedure. The variables *MAX* and *COL* are used, respectively, to keep track of the maximum matching and relative position. At the end of the procedure, *MAX* is also normalized.

Code 2.2 Second part of *IMA* (best matching)

```
MAX = 0
COL = 1

for c = 1 to W_ref
    SUM = 0
    for n = 1 to N_s
        i = c + (n – 1) W_slot
        if MATCH_SLOT[i] = n then
            SUM = SUM + MATCH_VAL[i]
        end if
    end for
    if SUM > MAX
        MAX = SUM
        COL = c
```

end if

end for

$MAX = MAX / N_s$

END

2.2 Panoramic image

For every place in the environment, a panoramic image can be reconstructed using the *IMA* algorithm. If θ is the view-angle of the camera and λ, with $\lambda < \theta$, is the step angle between two consecutive images, then a full panoramic view can be reconstructed combining the $2\pi / \lambda$ images. Between two consecutive images there is an overlap $\theta - \lambda$ wide, as shown in Figure 2.4. In the whole section, for simplicity, the positive orientation for all the angles is clockwise.

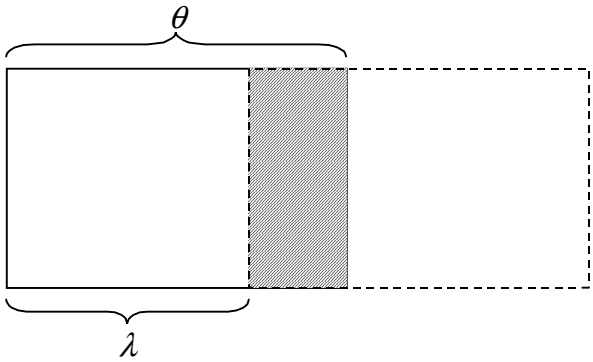


Figure 2.4 Sequence of images for panoramic view reconstruction

Using the same notation, we call I_{ref} the panoramic image, which is the reference for the future new images. The width W_{ref}, of course, depends on the view-angle of the camera, so that:

$$W_{ref} = W_{new} \cdot \frac{2\pi}{\theta} \tag{2.1}$$

The panoramic image I_{ref} could be obtained inserting, at the exact position, the sequence of images I_{new}, taken at angular intervals 0, λ, 2λ, 3λ, …. However, in a real application, it is difficult to take images with such precision. If the chosen interval, for example, is 30° ($\lambda = \pi / 6$), the images should be taken exactly at 0, 30°, 60°, 90°, …, 330°. This would be a tedious and time-expensive work. Instead, the angular interval is just an approximation of λ, with a succession like 0, 27°, 65°, 99°, …, 323°. In order to correctly align this "imprecise" sequence of images, a modified version of *IMA* is used.

NOTE - For simplicity, the procedure here explained assumes that the right part of an image always matches perfectly with the left part of the next. This is obviously not true. In section 5.1.2 there are experiments showing robustness and limitations of the panoramic image reconstruction.

2.2.1 Variant of *IMA* for panoramic image reconstruction

The panoramic image is initially just a black image. The value returned by *NCC* for a black image is exactly 0.5[3]. With a simple modification to the second part of *IMA*, as highlighted in Code 2.3, this situation can be handled and used for the correct insertion of a new image in the panoramic view. Basically, whenever a slot is compared with a black zone, the matching value assigned is the mean of the previous comparisons[4]. This permits the positioning of two sequential images like in Figure 2.4.

Code 2.3 Second part of *IMA* modified for panoramic image reconstruction

```
MAX = 0
COL = 1

for c = 1 to W_ref
    SUM = 0
    for n = 1 to N_s
        i = c + (n – 1) W_slot
        if MATCH_SLOT[i] = n then
            SUM = SUM + MATCH_VAL[i]
        else if MATCH_VAL[i] = 0.5
            SUM = SUM + SUM / (n – 1)
        end if
    end for
    if SUM > MAX
        MAX = SUM
        COL = c
    end if
end for

MAX = MAX / N_s
```

[3] Actually, *NCC* returns 0.5 every time one (or both) of the compared images has just one colour. This happens because the mean intensity is the same for all the pixels. See Appendix A for details.
[4] Of course, this is valid only if the images are inserted in the exact order, from left to right.

END

The reconstruction process is done as follows. All the images are compared with *IMA* and inserted, one by one, with respect to the position *COL* returned by the algorithm. This is the position where the matching is maximum. At the beginning, since the panoramic image is completely black, the first normal image is inserted on the very left. The following images are then inserted moving gradually to the right, until filling the whole panoramic image. Every image inserted overlaps the previous one approximately by an angle $\theta - \lambda$, as in Figure 2.4.

A consideration about this variant of *IMA* described in Code 2.3. Though the new added condition can be easily inserted just for the procedure of panoramic image reconstruction, it has been noticed that it does not influence the result of *IMA* even if kept during the place recognition stage. When used in a normal environment, indeed, the probability to have exactly a matching-value of 0.5 is practically insignificant.

2.2.2 *CLAHE* filtering

During the panoramic image reconstruction, an important factor for the final result is the quality of the taken images. Sometimes, there are not enough distinguishable features in the image. This is often due to the particular observed scene, like a wall or a big cupboard, or to the light conditions (see for example Figure 2.5) In these situations, the procedure of panoramic image reconstruction may fail because not able to correctly align two sequential images. In order to resolve or at least reduce such problem, the images are filtered using the *Contrast Limited Adaptive Histogram Equalization* (*CLAHE*).

Figure 2.5 Original image

Figure 2.6 Image filtered with *CLAHE*

A normal histogram equalization does generally improve the information visibility, but it applies the same equalization function to the whole image. *CLAHE* breaks the image into “tiles” and determines the best function to use for each tile. The result is an image that shows artificial boundaries between tiles and so an interpolation scheme is used to smooth the pixel intensities between tiles. The effect of *CLAHE* filtering can be observed in Figure 2.6.

2.3 Heading angle extraction

An important feature of *IMA* applied to panoramic images is the capacity to extract the position where the new image matches better. This position is given by the value *COL*, which is the left pixel column of the region on I_{ref} where the best match is. If the angle 0 is set on $COL = 1$, then the angle α of the camera's direction is simply given by the following expression (like in section 2.2, the positive orientation is clockwise):

$$\alpha = 2\pi \cdot \frac{COL - 1}{W_{ref}} \tag{2.2}$$

Therefore, if all the panoramic images have been reconstructed with a common angle of reference, α can be used to estimate the robot's heading. Of course, this measurement of the heading angle depends on the correctness of the image matching. It could be also completely senseless in case the new image was associated with the wrong panoramic image. Therefore, particular care must be taken in using it. The angle α can be considered reliable when the matching, on the correct panoramic image, is high. In this case, its precision is normally good enough for correcting the odometry's heading angle, as experimentally demonstrated in section 5.2.7.

2.4 Enhancement with digital zoom

The place recognition method, explained so far, suffers of the problem of being really sensitive to the distance from the original point, where the panoramic image has been constructed. This means that, moving the robot some tens of centimeter far from that point, the output of *IMA* decreases quickly.
To resolve this problem and enhance the place recognition, a digital zoom processing is added in a way to enlarge the area detection.

2.4.1 Geometric description of digital zoom

The principle of digital zooming can be explained using the well known "pin-hole camera" model. Differently from the mechanical zoom provided on some devices (where the focus length *f* is varied to increase or reduce the scene's dimension on the image plane), with digital zoom the focus length is fixed. The scene's dimension, conceptually, is modified directly on the image plane. Therefore, given some definitions, it is easy to find out some geometric relations.
Let us consider an object of height *H*, placed on the position x_A. Its relative height on the image plane is h_0, with a fixed focus length $f = f_0$. Said ρ the zoom factor, the robot can be "virtually" moved from a position x_0 to $x(\rho)$ modifying such factor, as shown in Figure 2.7.

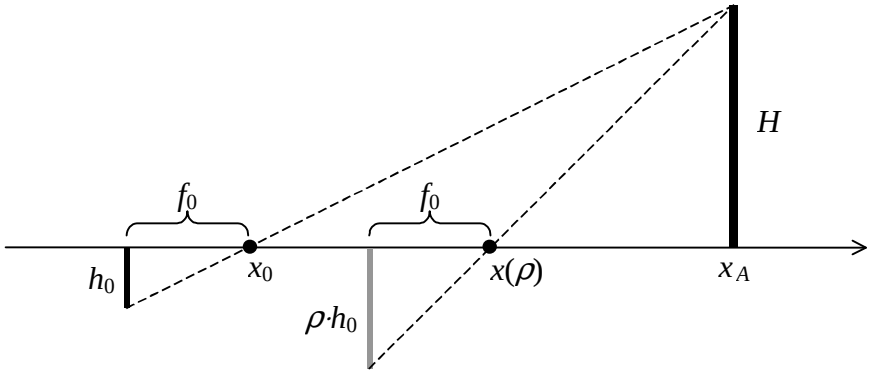


Figure 2.7 Pin-hole camera model of digital zoom

Observing the figure above, the following relations can be extracted:

$$\frac{h_0}{f_0} = \frac{H}{x_A - x_0} \tag{2.3}$$

$$\frac{\rho \cdot h_0}{f_0} = \frac{H}{x_A - x(\rho)} \tag{2.4}$$

From the expression above, we can easily calculate the "virtual" shift $\Delta x(\rho)$ from the original position x_0:

$$\Delta x(\rho) = \left| x(\rho) - x_0 \right| = \left| x_A - x_0 \right| \cdot \left| 1 - \frac{1}{\rho} \right| = D \cdot \left| 1 - \frac{1}{\rho} \right| \tag{2.5}$$

where D is the absolute distance of the object from the camera.

2.4.2 Application to place recognition

The equation (2.5) is based on a very simplistic model. In the real world, of course, an observed scene is a combination of several three-dimensional objects at different distances from the camera. Nevertheless, it can be still used to intuitively understand what explained below.

Given a panoramic image of a place at position $P_0(x_0, y_0)$ and moving the robot along a rectilinear line $y = y_0$, on an interval $[x_0 - \Delta x, x_0 + \Delta x]$, *IMA* returns values that can be approximated by a gaussian function, like in Figure 2.8 (see section 5.1.5 for experimental results).

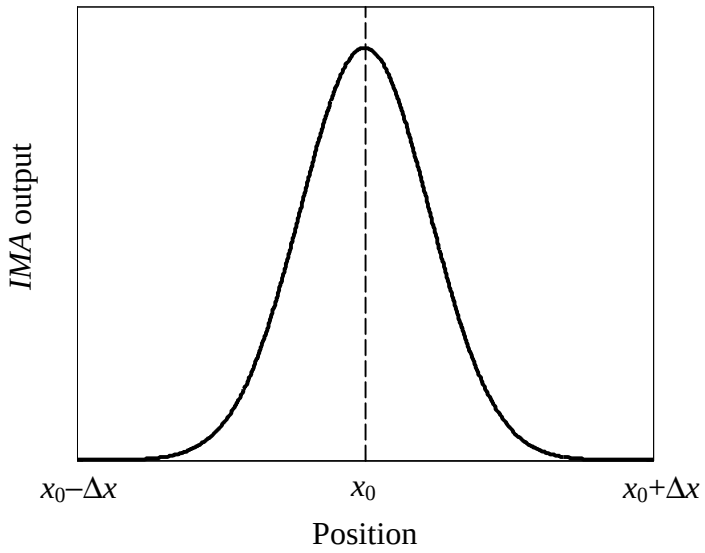


Figure 2.8 *IMA* output

To expand the width where *IMA* output is higher, the input image from the camera can be digitally zoomed and compared again with the stored panoramic image. More precisely, after a normal comparison, the image is zoomed in and compared again, then zoomed out and compared once more. The addition of these two comparisons results, theoretically, in the addition of two new gaussian waves to the graph of Figure 2.8. Let us consider a place x_0, equidistant from all the surrounding objects. If we choose a zoom factor $\rho_{\text{in}} > 1$, the centre of the relative gaussian x_{Zin} can be calculated using equation (2.5):

$$x_{Zin} = x_0 + \Delta x(\rho_{in}) = x_0 + D \cdot \left(1 - \frac{1}{\rho_{in}}\right) \tag{2.6}$$

To have then another gaussian, symmetrical with respect to x_0, a new zoom factor $\rho_{\text{out}} < 1$ must be used[5]. To calculate it, we can start from the following expression:

$$x_{Zout} = x_0 - \Delta x(\rho_{in}) = x_0 + D \cdot \left(1 - \frac{1}{\rho_{out}}\right) \tag{2.7}$$

After simple steps, the formula for the new zoom factor is like follows:

$$\rho_{out} = \frac{\rho_{in}}{2\rho_{in} - 1} \tag{2.8}$$

[5] This is presented here just for completion to the geometric description. In reality, a digital zoom out cannot be used, because the data outside the image boundaries is missing. The problem is then solved comparing the image with a zoomed in panoramic image, as described in chapter 4.

The combination of the three *IMA* output is then like shown in Figure 2.9. More precisely, the output considered for place recognition is the maximum of the three gaussians, as highlighted by the thickest line.
Considering the robot moves on the Cartesian plane, the place recognition with digital zoom could be then represented by a three-dimensional "Mexican hat, still considering only the directions passing by P_0. Again, this is the case for a supposed place where all the surrounding objects are equidistant. Some considerations on the effect of the zoom for more realistic environments are given in the next section.

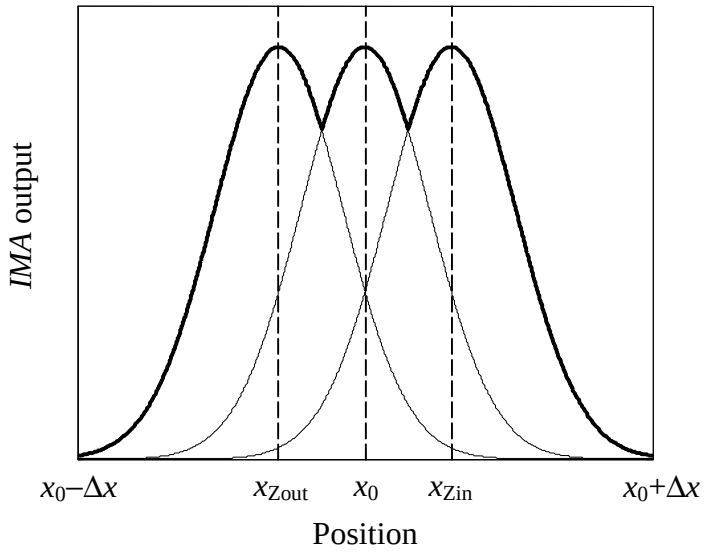


Figure 2.9 *IMA* output with digital zoom

2.4.3 Some considerations

As previously said, the explanation of digital zoom, applied to place recognition, is based on the assumption that all the objects, around the considered place, are equidistant from its centre. This results in a symmetrical "Mexican hat" output of *IMA*. However, in real environment this is not true: scenes (and objects within them) always have different distances from the point they are observed. Therefore, the real shape of place recognition output is considerably different from Figure 2.9. We can take, for example, an indoor environment like an empty room. Since the virtual shift Δx, given by equation (2.5) for a fixed zoom factor ρ, changes linearly with the distance D, we can intuitively presume the place recognition output follows the shape of the room. This is shown for two different places in Figure 2.10. The crosses are the place centres (where the panoramic images are taken) and the dashed lines identify the maximum mach for the zoomed image.

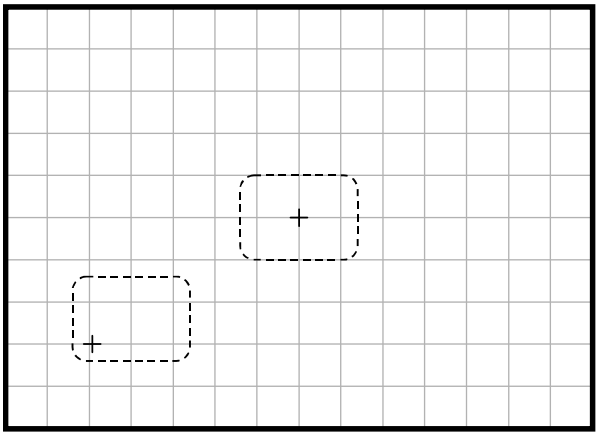


Figure 2.10 Place recognition with digital zoom

The observations above suggest some care must be taken in choosing the places to recognize and the zoom factor, depending on the dimension of the environment. In particular, if they are to close each other or the zoom is too high, the risk of overlaps among them augments and thus the probability of *perceptual aliasing* (two different places look the same). Moreover, since the zoom based recognition works properly only on those directions including the area reference point, it is preferable to use a low zoom factor. In this way, the area is smaller but the probability to be correctly aligned is higher.
A final consideration on another aspect of digital zoom might be also noteworthy to explore in future works. So far, the zoom has been assumed fixed and used uniquely to increase the recognized areas. Instead, once an area has been identified, the zoom factor could be varied in a way to maximize the place recognition output. With such a factor, it would be then possible to calculate the distance from the area reference point. Finally, in combination with the heading angle (section 2.3), it would give a more precise position inside the current area.

3 MULTI-HYPOTHESES TRACKING

The main problem in using image-based place recognition for localization arises when two or more places are very similar and difficult to distinguish. This case, referred as *perceptual aliasing*, is not typical of vision systems only, of course, but of every kind of sensor providing information about the perceived world (sonar, laser, etc.). This happens frequently in indoor environments like offices, where rooms and furniture are often similar and may cause the recognition of different places difficult, even for a human exploring for the first time such environment.
The procedure described on the previous chapter, based on *IMA*, is normally able to distinguish different places because it considers a significant amount of information coming from the vision input. Nevertheless, cases of perceptual aliasing often occur because of occlusions or relevant changes of the scenes with respect to those ones originally stored. For example, if a person in front of the camera is covering part of a scene, important to distinguish it from another one, then uncertainty comes out and it might not be possible to correctly identify the relative place.
To handle this kind of uncertainty, we adopt an algorithm inspired by the Markov Localization [Fox98]. It starts from a series of hypotheses generated by the place-recognition procedure and chooses the most likely according to the previous hypotheses and the robot's movement.

3.1 Overview of Markov Localization

The Markov Localization is a direct application of state estimation within the framework of "Partially Observable Decision Markov Processes" (*POMDP*). Two assumptions must be valid for the considered environment where the robot moves:

1) independence of the actions
2) independence of the observations.

For the first one, the knowledge of the state and of the action at time t–1 is sufficient for the prediction of the state at time t. If s_k is the state of the robot at time k, with relative observation v_k, and a_k is an action performed starting at time k from s_k, the same concept can be written as follows:

$$P(s_t \mid s_1,...,s_{t-1},v_1,...,v_{t-1},a_1,...,a_{t-1}) = P(s_t \mid s_{t-1},a_{t-1}) \qquad (3.1)$$

Basically, s_t and s_{t-1} are respectively the current and previous positions of the robot and a_{t-1} is the movement between them as recorded by the odometry.
For the second one, instead, an observation v_k at time k depends only on the relative state s_k, so that we can write the next expression:

$$P(v_t \mid s_1,...,s_t,v_1,...,v_{t-1},a_1,...,a_{t-1}) = P(v_t \mid s_t) \tag{3.2}$$

Here, the observation v_t is simply the data extracted from the world by a robot's sensor at state s_t.
Below we adopt some notation commonly used for such method[6]. In particular, we call *belief* of s_t, or $Bel(s_t)$, the probability $P(s_t \mid v_1,\ldots,v_t, a_1,\ldots,a_{t-1})$ of the state s_t that we have to estimate. Applying Bayes' rule, it is possible to write the following equation:

$$Bel(s_t) = \frac{P(v_t \mid s_t,v_1,...,v_{t-1},a_1,...,a_{t-1})P(s_t \mid v_1,...,v_{t-1},a_1,...,a_{t-1})}{P(v_t \mid v_1,...,v_{t-1},a_1,...,a_{t-1})} \tag{3.3}$$

We can now take the terms in the right part of (3.3) and examine each of them separately. From the assumption (3.2) of independence of the observations, the first term of the numerator can be rewritten as follows:

$$P(v_t \mid s_t,v_1,...,v_{t-1},a_1,...,a_{t-1}) = P(v_t \mid s_t) \tag{3.4}$$

Supposing S is the entire space of possible states and applying the "Total Probability Theorem" to the second term of the nominator in (3.3), we have the next equation:

$$P(s_t \mid v_1,...,v_{t-1},a_1,...,a_{t-1}) = \sum_{s_{t-1}\in S} P(s_t \mid s_{t-1},v_1,...,v_{t-1},a_1,...,a_{t-1})P(s_{t-1} \mid v_1,...,v_{t-1},a_1,...,a_{t-1}) \tag{3.5}$$

At the first term of the sum in (3.5) we can apply the independence of the actions, expressed by equation (3.1). Moreover, the second term of the same sum can be substituted remembering the definition of *belief* [7]. Equation (3.5) can be then rewritten in a simpler form:

$$P(s_t \mid v_1,...,v_{t-1},a_1,...,a_{t-1}) = \sum_{s_{t-1}\in S} P(s_t \mid s_{t-1},a_{t-1})Bel(s_{t-1}) \tag{3.6}$$

Finally, the denominator of (3.3) can be seen just like a normalization factor of the *belief*. Indeed, applying the "Total Probability Theorem", it can be substituted with the following expression:

$$P(v_t \mid v_1,...,v_{t-1},a_1,...,a_{t-1}) = \sum_{s_t\in S} P(v_t \mid s_t,v_1,...,v_{t-1},a_1,...,a_{t-1})P(s_t \mid v_1,...,v_{t-1},a_1,...,a_{t-1}) \tag{3.7}$$

[6] To make the reading easier, the notation is not always formally correct.
[7] Indeed the state s_{t-1} is independent from the action a_{t-1}, because the latter is executed just after such state. Then $P(s_{t-1} \mid v_1,...,v_{t-1},a_1,...,a_{t-2},a_{t-1}) = P(s_{t-1} \mid v_1,...,v_{t-1},a_1,...,a_{t-2}) = Bel(s_{t-1})$.

$$= \sum_{s_t \in S} \left[P(v_t \mid s_t) \sum_{s_{t-1} \in S} P(s_t \mid s_{t-1}, a_{t-1}) Bel(s_{t-1}) \right] \tag{3.8}$$

A new version for calculating $Bel(s_t)$ can now be written substituting (3.4), (3.6) and (3.8) in (3.3):

$$Bel(s_t) = \frac{P(v_t \mid s_t) \sum_{s_{t-1} \in S} P(s_t \mid s_{t-1}, a_{t-1}) Bel(s_{t-1})}{\sum_{s_t \in S} \left[P(v_t \mid s_t) \sum_{s_{t-1} \in S} P(s_t \mid s_{t-1}, a_{t-1}) Bel(s_{t-1}) \right]} \tag{3.9}$$

In particular, we can note that the denominator in equation (3.9) is just a normalization factor. From the new expression of $Bel(s_t)$ it is now easy to implement an algorithm to update the states' *belief*. This is called "Markov localization algorithm" and can be divided in three steps.

1) *Prediction* when a new action is executed:

$$P'(s_t) = \sum_{s_{t-1} \in S} P(s_t \mid s_{t-1}, a_{t-1}) Bel(s_{t-1}) \tag{3.10}$$

2) *Update* when a new observation is available:

$$P''(s_t) = P(v_t \mid s_t) P'(s_t) \tag{3.11}$$

3) *Normalization* when all the $P''(s_t)$ have been calculated:

$$Bel(s_t) = \frac{P''(s_t)}{\sum_{s_t \in S} P''(s_t)} \tag{3.12}$$

To apply the "Markov localization algorithm", therefore, only two main elements must be known. The first one is the term $P(s_t \mid s_{t-1}, a_{t-1})$ of the *prediction* step, in equation (3.10). It is normally called *action model* and provides the probability of being in the position (state) s_t, given that the robot executed the action a_{t-1} starting from s_{t-1}. The second element is necessary for the *update* step and is the term $P(v_t \mid s_t)$ of (3.11). It is the probability that an observation v_t is done when the robot is in s_t. From here the name *sensor model*. It must be also noted that at the beginning ($t = 0$), the *belief* is equally distributed on all the possible states, as the robot does not know its initial position.

3.2 Assumptions and notation

Let us indicate the state[8] of the robot, at time t, by a triplet $< x_t, y_t, \varphi_t >$, where x_t and y_t are the Cartesian coordinates of the robot's position and φ_t is its heading angle. The couple (x_t, y_t) belongs to a finite set of two-dimensional points, which is the topological map where the robot is supposed to be localized. The heading angle φ_t, instead, has continuous values inside the interval $[0, 2\pi)$. The entire set S of possible states, therefore, contains an infinite number of elements. To make the problem computationally treatable, we impose here some assumptions. We assume that the probability distribution, at time t, of being in a certain position $< x_t, y_t, \varphi_t >$ is completely contained in a sub-set $S_t \subset S$. The elements of S_t are all the positions for which *IMA* (chapter 2), at time t, returns a matching-value higher than a certain threshold, plus an additional "virtual" position given by the odometry (this is explained in 3.2.1). That is, the real position of the robot is always supposed to be one of those recognized by the place recognition[9] or calculated using the odometry information. Of course, the number of possible states so generated is limited by the nodes of the topological map; therefore, S_t is a numerable set.
In the next sections, we will refer to the set S_t with the letter D and we will call *destination* an element $d \in D$. We will also refer to the set S_{t-1} with the letter O and call *origin* an element $o \in O$. It is clear that a set D of destinations at time t will become the set O of origins at time $t + 1$. Moreover, to distinguish our "local" probability distribution from that one used in the Markov Localization, we substitute the word *belief* with *activity*, as in [FM02]. Thus, $Bel(s_t)$ and $Bel(s_{t-1})$ become $Act(d)$ and $Act(o)$ respectively (activity of the destination and activity of the origin).

3.2.1 Virtual destination hypothesis

The assumption of considering only the destinations given by the last observation, that is the *IMA* output, would be too restrictive. To be sure that the right position is in effect one of those recognized, the threshold applied to the *IMA* output should be very high. This would limit excessively the possibility to consider good hypotheses just because some changes in the environment, temporary or permanent, have reduced their distinctiveness. On the other hand, with a lower threshold, the number of possible destinations increases, together with the probability to choose the wrong one. Even worse if none of the current hypotheses are correct.
To handle this kind of ambiguity, sometimes they make use of a "Zero Hypothesis", that is a way to handle the case when all the other hypotheses are wrong. In [JK01], for example, the authors have a finite set of hypothesis generated by new observations and updated simultaneously using Kalman filtering. The zero hypothesis is used to close the probability space and is kept updated considering the uncertainty of the observations. When the probability of such hypothesis is higher than the others, the robot is in a state of indecision.

[8] Sometimes, we will use indifferently the words "state" and "position".
[9] This is justified by the fact that, most of the times, the correct position is in effect one of the best recognized with *IMA*.

In our approach we found useful inserting a “virtual” destination, that is the position on the topological map closest to that one given by the odometry. More precisely, the latter is calculated from the last winning destination adding the relative displacement given by the odometry and corrected as explained in 3.7. The heading angle of this new hypothesis is also given by the odometry. The term “virtual” is because we assign to it a matching-value, like all the other destinations generated by an observation, and then we treat it at the same way. The assigned matching-value, in particular, is equal to the threshold used to generate the other hypotheses. In practice, it would be like *IMA* has recognized an additional place, with the minimum matching-value. On the next update process, such “virtual-destination” becomes the “virtual-origin”.

3.3 Action model

We saw in section 3.1 that a first fundamental component of the “Markov localization algorithm” is the *action model*. Using the notation introduced in 3.2 and simply calling a the action a_{t-1}, we can rewrite such model as follows:

$$P(s_t \mid s_{t-1}, a_{t-1}) \equiv P(d \mid o, a) \tag{3.13}$$

This model expresses the probability that a destination d is reaching performing the action a from the origin o. This probability is estimated taking into account the location and the heading angle of the robot. The action a is simply the displacement given by the odometry. For our scope, no sophisticated models are used for handling the cumulative errors typical of the odometry. Its information, indeed, is always relative to the previous estimated state and corresponds to a short path. Therefore, it is considered reliable enough for being directly used in the *action model*, as explained below.
Let us say Q_o the two-dimensional position of the origin o, with a heading angle φ_o. Then Q_a is the position reached from Q_o after the execution of a and Q_d is the position of the destination d. The first parameter used for the estimation is the distance Δl between Q_d and Q_a:

$$\Delta l = \left\| Q_d - Q_a \right\| \tag{3.14}$$

Now let us consider the heading angle φ_a after the action and that one of the destination, φ_d. The second parameter we use is the difference $\Delta\varphi$ between φ_d and φ_a[10]:

$$\Delta\varphi = \left| \varphi_d - \varphi_a \right| \tag{3.15}$$

[10] Note that for the case “virtual-origin”→”virtual-destination”, introduced in 3.2.1, the angle φ_d and φ_a are the same, so $\Delta\varphi = 0$.

A graphical representation of all the quantities, introduced so far, is shown in Figure 3.1. The dark-grey circle, on the left, represents the robot's origin *o*. The white one is the position given by the odometry after the execution of the action *a*. Finally, the light-grey circle is the current destination *d*. Please note that Q_o and Q_d are two pre-defined positions within the topological map, while Q_a could be any point in the environment.

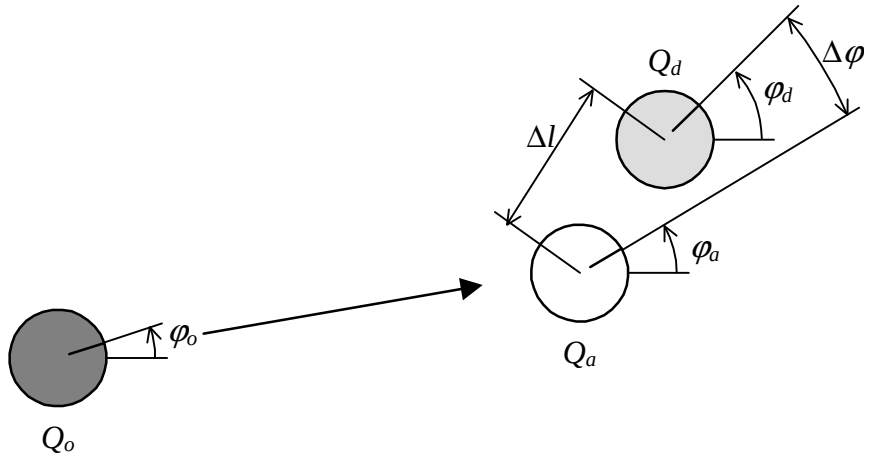


Figure 3.1 Parameters for the *action model*

The quantity Δl is then use to estimate the probability of the destination's position with the following formula:

$$g_l(\Delta l) = \frac{1}{\sqrt{2\pi}\cdot\Delta l_{\max}}\cdot e^{-\frac{\Delta l^2}{2\Delta l_{\max}^2}} \tag{3.16}$$

Δl_{max} is the maximum Δl among all the current combinations origin-destination. The function $g_l(\Delta l)$ is a gaussian, centred in zero and with standard deviation Δl_{max}. Therefore, its width varies at every update, depending on the current Δl_{max}. In the same way, the quantity $\Delta\varphi$ is use to estimate the probability of the destination's heading. The formula applied is again a gaussian centred in zero and with standard deviation $\Delta\varphi_{max}$, which is the maximum $\Delta\varphi$ among all the current ones:

$$g_\varphi(\Delta\varphi) = \frac{1}{\sqrt{2\pi}\cdot\Delta\varphi_{\max}}\cdot e^{-\frac{\Delta\varphi^2}{2\Delta\varphi_{\max}^2}} \tag{3.17}$$

Finally, the *action model* is then calculated combining (3.16) and (3.17) into the following expression:

$$P(d \mid o,a) = g_l(\Delta l)\cdot g_\varphi(\Delta\varphi) \tag{3.18}$$

$$= \frac{1}{2\pi \cdot \Delta l_{\max} \Delta \varphi_{\max}} \cdot e^{-\frac{\Delta l^2}{2\Delta l_{\max}^2}} \cdot e^{-\frac{\Delta \varphi^2}{2\Delta \varphi_{\max}^2}} \qquad (3.19)$$

3.4 Sensor model

In many localization systems, the environment is sensed through low-dimensional devices, like sonar or laser range finders, for which accurate models are already available [BFHS96; MO88]. Other approaches instead use vision to calculate the robot's position with respect to some particular features. In [MPP04], for example, an omni-directional image is processed using a ray-tracing method, simulating a laser range sensor that returns distances of chromatic-transition features. Even in this case, an accurate model is provided, with parameters extracted by a modified EM algorithm [DPLR77] applied to a set of 2000 sample images. There are also approaches where the sensor models are learnt using neural networks, both in case of data from vision or sonar [Th98; Th99; OHD97].

The data given by the image-based procedure for place recognition, i.e. the *IMA*'s matching-value, differs from all the above-mentioned implementations. There are no measures of distances or extraction of particular features. What we have, instead, is the result of a comparison between a pre-recorded panoramic image and a new image from the camera. A similar situation is described in [FM02], where they compare images from an omni-directional camera with images previously stored. In that case, the difference between the two images, new and pre-recorded, was passed to a gaussian function, obtaining a value between 0 and 1 proportional to this difference. In [DN01] the sensor model is directly derived by their image-based place recognition, where the match of image histograms provides also the probability of obtaining a certain sensor reading from a place hypothesis. The *IMA* works in a more sophisticated way, but the concept is almost the same. The *sensor model*, in some way, is implicitly "included" in the pre-recorded panoramic image. An ideal image would return 1 when it matches perfectly on the panoramic image and would decrease to 0 as more as the match is bad. Of course, the perfect match, for which *IMA* would return 1, cannot happen. First of all because of the panoramic image, which is far away to be a "real" 360° scene[11]. Second, because the environment cannot be completely static. So, for example, in an office many particulars change day by day and the recorded images become less representative. However, these kind of problems are generally common to all the panoramic images and the decrease of their quality can be considered the same for all of them. Therefore, this does not influence very much the performances of the localization system. Back to the *IMA*'s match, we should also note that a little noise always exist (with the exception reported in [3]), since in practice two images are never completely different. This problem is tackled by the fact that the observations we are dealing with are higher than an appropriate threshold.

[11] This could be improved using an omni-directional vision sensor, as discussed in 6.2.

From what we said above, the probability of the observation given the current state can be considered directly the matching-value by *IMA*. Using the notation introduced before and referring to the observation v_t simply as v, we can write the *sensor model* as follows:

$$P(v_t \mid s_t) \equiv P(v \mid d) \tag{3.20}$$

Therefore, remembering from Code 2.2 that the matching-value is contained by the variable *MATCH*, we can refer to it with *MATCH*(*d*), relatively to a particular destination *d*. For the update of the activities then, we use the next probability:

$$P(v \mid d) = MATCH(d) \tag{3.21}$$

3.5 Update of the activities

As previously said, we call *activity* the correspondent of the Markov Localization's *belief*. The update of the activities is done with the same formulas, but taking into account the assumptions in 3.2. In particular, since the possible destinations are generated as far as a new observation is available, the three phases, *prediction* → *update* → *normalization*, are always executed at the same time step[12]. Then, given a set of destinations $d \in D$ and origins $o \in O$, the procedure for the calculus of the new activities is reported below:

1) *Prediction*:

$$P'(d) = \sum_{o \in O} P(d \mid o, a) Act(o) \tag{3.22}$$

2) *Update*:

$$P''(d) = P(v \mid d) P'(d) \tag{3.23}$$

3) *Normalization*:

$$Act(d) = \frac{P''(d)}{\sum_{d \in D} P''(d)} \tag{3.24}$$

In the formulas above we apply the expressions discussed in the previous sections. In particular, the probability $P(d \mid o, a)$ of (3.18, 3.19) is used in the *prediction* (3.22) and the *IMA* matching-value of d , also called *MATCH*(*d*), substitutes $P(v \mid d)$ in (3.23).

[12] In the original "Markov localization algorithm", we remember that the *prediction* could be done whenever an action was executed, while the *update* and *normalization* whenever a new observation was available.

3.6 An intuitive explanation of the update process

So far, the way the activities are updated has been presented in a context similar to that one used for the Markov Localization. As already said in the introduction of this chapter, the method explained is just inspired by the Markov model, but it is clear that some of the assumptions adopted does not "fit", in strict mathematical terms. Nevertheless, in most of the real applications, where the Markov Localization has been used, there are violations and/or adaptations of the formal model (for example, the independence of the observations). The same Markov model and its assumptions are just approximations of the real world.

We try here to give an intuitive explanation of the update process, out of the mathematical context and just helped by some graphical examples. We examine separately the four main terms involved in the *prediction* → *update* process of equations (3.22) and (3.23). These are the activity of the origin $Act(o)$, the two parameters Δl and $\Delta\varphi$ for calculating the *action model* $P(d \mid o, a)$ and finally the *sensor model* $P(v \mid d)$. In the next figures, we have the origins on the left and the destinations on the right. The short line inside the circles indicate the heading of the robot. The arrows represents the last action executed by the robot and the dashed circle is the position after such action, as given by the odometry. For simplicity, we suppose the odometry without errors.

3.6.1 Activity of the origin

Suppose two new destinations are generated, d_1 and d_2, both with the same matching-value[13]. The possible origins are also two, o_1 and o_2. The activities of these, $Act(o_1)$ and $Act(o_2)$, are different. In particular, the activity of o_1 is higher than o_2, as represented by the lighter gray in Figure 3.2. There are just four possible transitions: $o_1 \rightarrow d_1$, $o_1 \rightarrow d_2$, $o_2 \rightarrow d_1$ and $o_2 \rightarrow d_2$.

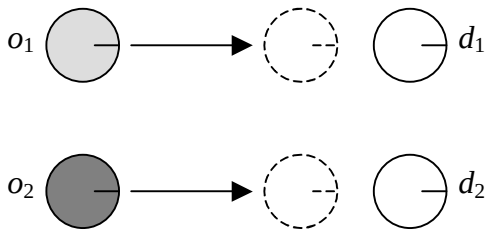


Figure 3.2 Selection based on the origins' activity

Because of the symmetry of the problem, the final activities of the destinations depend only on the activities of the origins. Since $Act(o_1) > Act(o_2)$, it is clear that the most likely destination is d_1.

[13] This would be a case of *perceptual aliasing*.

3.6.2 Distance from the destination

In this case, we have one origin o_1 and two destinations d_1 and d_2, again with identical matching-values. In Figure 3.3 we can see the graphical representation. What make us choose d_1 as the most likely destination is the fact that the distance, with respect to the odometry's position, is shorter. Such distance, in effect, is Δl.

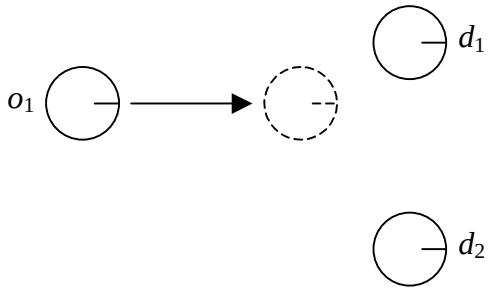


Figure 3.3 Selection based on the distance Δl

In the *action model* (3.18), this distance is "weighted" and normalized from the gaussian $g_l(\Delta l)$ (3.16), so that a short distance has an influence much bigger than a long one. In case of multiple origins, for each destination all the possible distances are taken into account.

3.6.3 Heading angle difference

A consideration similar to the previous case can be done for the heading angle difference, which we called $\Delta\varphi$. It is calculated comparing the heading of the considered destination and the supposed one given by the odometry after the execution of an action.

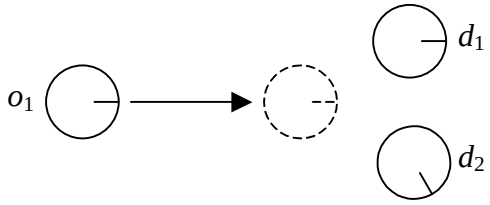


Figure 3.4 Selection based on the heading angle difference $\Delta\varphi$

The origin is still one, distances and matching-values are the same for both the destinations. However, in Figure 3.4 we can see the destination d_2 having an heading angle different from that one in the odometry position. The destination d_1, instead, is oriented in accordance with it. Therefore, d_1 is the most likely choice. As in the previous case, we should note again that $\Delta\varphi$ is passed through a gaussian, the $g_\varphi(\Delta\varphi)$

in (3.17), and that, for multiple origins, all the heading angle differences are considered for each destination.

3.6.4 Current observation

The last situation we examine is when the only thing that makes the difference in the destination choice is the current observation.

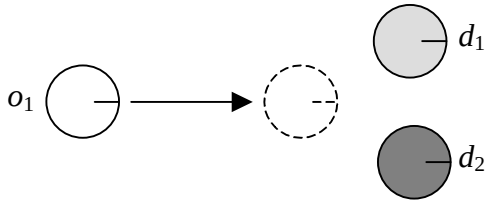


Figure 3.5 Selection based on the current observation

In the figure above, we see again two destinations, but this time with two different matching-value. In particular, $MATCH(d_1) > MATCH(d_2)$. Being the same all the other parameters, the most probable destination is d_1.

3.7 Correction of the odometry information

An important role in the selection of the current destination is played by the odometry. Indeed, the *prediction* step of (3.22) makes use of the *action model* (3.18, 3.19) and this one strongly depends on the odometry information.
We can examine first of all the distance Δl (3.14) used in (3.16), supposing the heading angle is correct with respect to the absolute zero direction of the environment. Such measure is simply the Euclidean distance calculated from the coordinates of the considered destination considered, stored into the map, and the coordinates of the robot, as given by the odometry. It is well known that the odometry is not reliable on long paths, due to wheels slippage, irregularities of the floor, collisions and so on. Nevertheless, when used on short ranges, it is information is quite precise. We then use such information just to measure relative displacements, resetting the odometry[14] every time an update of the topological position has been done. Of course, despite the fact that the destination is not always correct, this reset can introduce an error, since a topological position is not a geometric point in a two-dimensional plane, but a sort of “area” or “region”. However, the fact that the area’s radius is reasonably smaller than the distance between two consecutive destinations reduces the effect of this error. On the other side, the vantage is significant, since it fixes a limit to the cumulative error of the dead reckoning.

[14] Actually it is not really reset, but the software program uses the difference between the current odometry and that one recorded during the last update.

Let us now consider the heading angle. It has a double importance: first of all, it is explicitly used in the calculus of the parameter $\Delta\varphi$ (3.15), which is the main variable of the gaussian (3.17). The latter is part of the *action model*. Furthermore, the heading angle is also the rotation of the internal frame of reference of the robot, with respect to an absolute 0 direction of the environment. Hence, an error on such angle means a rotation of the topological map used from the robot, and from this map depends also the calculus of the parameter Δl above. In many applications, instead of using the heading angle calculated by the wheels encoders, an external magnetic compass is mounted on the robot, like in [DN01; DMS02]. This has the advantage of being independent from the cumulative errors of the odometry, since it gives an absolute direction of the North. Using such direction as reference, the robot is then able to correct its internal heading. On the other hand, this device is not immune from errors, mainly because of metallic objects in proximity of the robot. In [FM02], for example, they observe as a magnetic compass turned out to be inefficient in their office environment. For our approach, we chose instead another solution. Since for every new environment we want to move on a new map and a new set of panoramic images are needed, we reconstruct these latter always starting from the same direction. As already described in 2.3, this permits the robot to recover its absolute heading using the formula (2.2). To limit the cases for which the wrong panoramic image is used, that is when the localization fails, we correct the odometry's heading angle only when the matching-value of the estimated destination is higher than a given threshold. It has been noted, indeed, that good matches usually mean correct estimations. This is due to the fact that *IMA* can distinguish different places with a certain accuracy and without incurring in errors of perceptual aliasing, at least in a typical office environment like ours. In 5.2.7 the reliability of this heading correction is demonstrated even in extreme conditions.

3.8 Overall algorithm

At this point, it is possible to write the whole algorithm for generating the destination hypotheses, calculating the relative activities and selecting the most likely one.

The case when no destination hypotheses are generated is treated in the first part of the localization algorithm. After the comparison of a new image with all the panoramic images in the topological map, it may happen indeed that no new destinations are considered because all the matching-values calculated are below the decided threshold. The only way to update the robot's position is then relying on the odometry and calculating the displacement relative to the last recognized place. Since the required output is just a topological position, we find out the place in the map supposed to be closest. Note that this kind of behaviour is like trying to guess the current location being "blind" and, in such circumstance in fact, the only information is provided by the internal odometry. Once the closest topological location has been calculated, the localization algorithm ends and waits for a new observation, which is hopefully good enough to generate one or more hypotheses.

When some of the comparisons among the panoramic images are greater then the threshold, the set of destinations is not empty and the algorithm can proceed with the

procedure for selecting the most likely one. First of all, for each combination origin-destination, the distances Δl and the angles $\Delta\varphi$ explained in section 3.3 are calculated, keeping track of the maximum values Δl_{max} and $\Delta\varphi_{max}$. All these quantities are then used in the following step, which is the calculus of the activities for all the new destination. The *prediction* and the *update* formulas in 3.5 are applied to each destination, always keeping track of that one with the highest activity[15]. When the *prediction* $\rightarrow$ *update* loop is completed, the activities are normalized among all the considered destinations. After that, the odometry's coordinates are reset to the most active destination. For the correction of the heading angle, there is a further check on the matching-value of such destination: if this is higher than a certain threshold, it means the observation was good and the orientation of the robot can be adjusted according to the angle given by *IMA*. Finally, the current set of destinations becomes the set of origins for the next execution of the localization algorithm.

In order to reduce the computational expense, the whole algorithm is executed only after the robot has moved of a certain distance or has rotated of a minimum angle. As in every program that makes use of video processing, indeed, we have to deal with time constraints and with the fact that other processes require to be executed. However, this also has the advantage of effectively generating new different destinations (i.e. different states) and reduce the cases of failures.

A scheme for the localization algorithm is reported in Code 3.1. Some new symbols are introduced:

- ε_M is the threshold used for extracting the destinations with the best matching-values
- ε_φ is the matching-value threshold for correcting the odometry's heading and is $\varepsilon_\varphi \geq \varepsilon_M$
- d^* is the destination with the higher activity and corresponds to the relative state $< x^*, y^*, \varphi^* >$
- o^* is the most likely origin, i.e. the last d^* extracted

The quantities ε_M and ε_φ are determined experimentally and the values adopted are illustrated in 5.2.

Code 3.1 Localization algorithm

Calculate the location given by o^* plus the odometry displacement and find the topological position $d_0 \in D$ closer to such location

Compare with *IMA* the new image with all the panoramic images in the map and extract the possible destinations $d \in D$ with a matching-value $MATCH(d) > \varepsilon_M$

[15] We say *activity* even if, at this stage, it is not normalized yet. However, the normalization part has not influence on the choice of the best destination.

if no new destinations are generated with *IMA*

Return the topological position d_0

END

end if

for each $d \in D$

for each $o \in O$

Calculate Δl and $\Delta\varphi$ (3.14, 3.15)

Keep track of Δl_{max} and $\Delta\varphi_{max}$

end for

end for

for each $d \in D$

Calculate $P(d \mid o, a)$ with (3.19)

Update the activity (3.22, 3.23)

Keep track of the destination d^* with the higher activity

end for

Normalize the activities (3.24)

Reset the odometry's coordinates

if $MATCH(d^*) > \varepsilon_\varphi$

Set the odometry's heading to the angle φ^*

end if

The destinations become the next origins, $O \leftarrow D$

Return d^*

END

4 SOFTWARE IMPLEMENTATION

The practical implementation of the localization module is a tedious work and, in terms of time expense, certainly comparable to the study of the theoretical approach. There are three main criteria followed for the software development:

- performances in terms of speed and memory space
- reusability and platform independence
- easy update and extension.

These led to the choice of creating a C++ library for Linux operating systems, that includes all the procedures described in chapter 2 and 3. The implementation as library permits its use in different contexts, being practically independent from the software architecture of the robot. In theory, it should be possible using such library on every robot provided with Linux OS, odometry information and a normal camera. In the following sections it will be explained the structure of the library and the main functionalities it provides, together with an introduction to an additional library for video processing. Its integration in the robot's middleware is then illustrated.

4.1 A general purpose library for localization

The library is a complete software framework for all the localization procedures described in the previous chapters. The code is written using Object-Oriented philosophy. In particular, one main object includes the map representation, with all its panoramic images and relative coordinates, and provides the necessary functions to insert the data and retrieve an estimate position. Note that such implementation is, in itself, a passive localization approach, in the sense that it does not interfere absolutely with the robot movement. On the other hand, it can still be integrated in a more sophisticated system of active localization.

4.1.1 The TopoMap object

The main object of the library is implemented with the class *TopoMap*. The name derives from the fact that it keeps an internal representation of the topological map. This means that all the panoramic images and the coordinates of the positions we

chose to map inside the environment are internally stored. Other information held by the *TopoMap* object is the current odometry and the video input, continuously updated by the robot during the localization. Finally, there are several methods to interact with the map, update the estimated position and retrieve all the necessary information.
When the *TopoMap* object is created, some important parameters must be specified. The most significant are the following:

- *scale*: the scaling factor applied to reduce the resolution of all the processed images;
- *slots*: the number of slots for the matching algorithm (*IMA*);
- *zoom*: the factor ρ_{in} for the digital zoom;
- *match_threshold*: the minimum matching-value ε_M for the generation of new destination hypotheses;
- *head_threshold*: the minimum matching-value ε_φ for the correction of the heading angle.

4.1.2 Panoramic image reconstruction

One of the first methods provided by the class *TopoMap* is a static function that reconstructs a panoramic image from a sequence of snapshots, as explained in 2.2. This method, called *getPanorama*, accepts as input parameters the path of the directory where the sequence of images has been recorded, the number of slots to use for the reconstruction, the activation of *CLAHE* filtering and other less significant settings. The returned value is a pointer to the reconstructed panoramic image. It is worth specifying that, during the creation of the panorama, the procedure of insertion of the snapshots can be visually debugged, so that the user can understand the reasons of eventual failures and then modifying the input parameters to resolve the problem. We will see in 4.2.3 how the function *getPanorama* has been used for a semi-automatic procedure that permits to get a panoramic image quickly.

4.1.3 Insertion of the topological map

Once all the panoramic images of the topological locations are available, it is necessary to provide the *TopoMap* object with the map. The task is accomplished by the method *setMap*. This needs, as only parameter, a reference to a text file providing the path of the directory where the panoramic images are stored, the names assigned to each topological position and their relative coordinates. Such file has a very simple syntax and we can see in Figure 4.1 an example of map file, in this case for our laboratory. Observing its text, it is simply to understand the three main information: the first line is the directory of the image files, the left column contains the names assigned to the topological nodes and the remaining columns are the relative coordinates *x* and *y*.

```
/home/belush/images/panoramic
CHARGER       0       0
DESK_1        1000    0
DESK_2        1000    1000
DESK_3        0       1000
SHELVES       0       -1000
CENTRE        1000    -1000
CUPBOARD      2000    -1000
DOOR          2000    0
```

Figure 4.1 Example of map file

4.1.4 Input data functions

Before every update step, we must provide the current information given by the internal odometry sensors and by the camera. For the first one, the dedicated method is *setCurrentOdometry*, which requires as input parameters the coordinates *x* and *y* plus the heading angle given by the odometry. The image retrieved by the camera, as buffer of bytes, is the input for the *setCurrentImage* method.

4.1.5 Update function

The core function of the whole software is *updateActivities*, for which it is worth spending some more words. Such method implements in practice the localization algorithm illustrated in Code 3.1.
First of all, using the new information introduced with the previous *setCurrentOdometry*, the *updateActivities* function estimates approximately the current metrical position. This is done as follows: using the odometry's position previously recorded (when a destination was chosen among all the generated ones) and the current metrical position, it calculates the vector connecting these couple of two-dimensional points. The direction of the vector is eventually corrected according to the last supposed heading angle error. The estimate of the current metrical position is then calculated "adding" such vector to the last winning destination or, as we use to call, the most probable origin o^*. Comparing the coordinates of this new position with those ones provided by the map, it finds out the closest topological node. Such node has a double importance: in case no destinations are generated in the following part, it is the "supposed" place; if some destination is generated, it is the additional "virtual" one, which we call d_0.
The following part of the *updateActivities* function is dedicated to the processing of the video input, previously provided by *setCurrentImage*. If the digital zoom is not used ($\rho_{in} = 1$), this simply means to apply *IMA* between the current image given by the camera and all the panoramas of the topological nodes. All those having a matching-value higher than the threshold ε_M are possible destinations to consider in the next part. If the zoom is active ($\rho_{in} > 1$), things are a little more complicated. In practice, for every topological node *IMA* is applied three times, one for each of the following combinations: "current image"→"panorama", "current image zoom in"→"panorama" and "current image"→" panorama zoom in". The first two cases are quite obvious (see 2.4.2). The third one instead is the practical implementation of

the zoom out for the current image. It is impossible, indeed, performing a (digital) zoom out of the image given by the camera without loosing all the contour pixels. Therefore, the solution adopted is reversing the procedure and comparing the original current image with a zoom in version of the panorama. Finally, for each node only the best of the three matching-values is considered.
Once all the comparisons with *IMA* are completed, the nodes for which the matching-value is higher than the threshold ε_M are considered as new possible destinations, together with the virtual d_0. The choice of the most probable depends on their activities, updated as explained in 3.5: the destination with the highest activity, which we called d^*, is the winning one. Instead, if no match is high enough, the function simply returns d_0, which is only a supposed position in this case.
At the end of *updateActivities*, after generation and choice of the new destination, the current odometry is recorded and will be used for future updates. This has in practice the same effect of resetting the odometry, as we often said, but without truly interfering with the real measurement. A similar method is applied to the heading direction, corrected by the vision with the angle explained in 2.3. Every time there is a winning destination d^*, if the relative matching-value is higher than ε_φ the program calculates the difference between the odometry's heading angle and that one of such destination, extracted with the formula (2.2). This difference is then used as offset in the successive calls of *updateActivities* to correct the odometry's heading angle. The function finally terminates returning the estimated destination d^*.

4.1.6 Video processing with OpenCV

It is clear that a fundamental role in the localization software is played by the video processing. For this application we chose to utilize an open-source library that was a former project of Intel®, freely downloadable from Internet. Basically, this consists of a collection of C functions that implement some popular algorithms for Image Processing and Computer Vision. It does not rely on external numerical libraries and is platform-independent, that is Linux and Windows compatible.
Most of the functions we made use from this library are for image transformation, in particular conversion from RGB to grey scale and resize of the dimensions. The camera indeed provides a colour RGB image with a resolution 384×288 pixel, while we need to work on grey images scaled three or more times, in a way to keep the processing time acceptably low. Furthermore, OpenCV provides a set of useful functions to load and save image files in several formats, plus the opportunity to create simple graphic user interfaces (GUI) to show the images and interact with them. This revealed to be particularly useful for debugging, for example during the panoramic image reconstruction.
The most important function we made use in our software is probably *cvMatchTemplate*, fundamental part of *IMA*. In practice this returns the Normalized Correlation Coefficient *NCC* for an input image and a template. In our algorithm the input image is the panorama and the template is a slot of the camera's snapshot (both converted to grey scale).

4.2 Integration with the robot middleware

The access to the different devices available on a mobile robot is normally provided by an intermediary software placed between the low level part, i.e. the Operating System's drivers, and the high level programs, which are the modules forming the "intelligent" behaviour of the robot. This middle-level software is called, indeed, *middleware*. Though our platform, an ActivMedia PeopleBot (see appendix B), was already provided with a basic middleware, we chose to make use of a more powerful framework called *Miro*. This is richer of useful features and makes easy the integration of the localization with other modules. In this section we introduce the main characteristics of *Miro* and how the localization system is implemented inside it. We also describe the initial realization of an automatic module for panoramic image reconstruction.

4.2.1 The *Miro* framework

Miro is a distributed object oriented framework for mobile robot control, based on CORBA (Common Object Request Broker Architecture) technology and developed at the ULM University, Germany. The *Miro* core components have been developed in C++ for Linux, but due to the programming language independency of CORBA further components can be written in any language and on any platform that provides CORBA implementations. Moreover, thanks to this technology, it is possible developing distributed applications. This means the opportunity to execute programs on different computers for controlling the robot, useful for example when its computing power is not adequate to perform many tasks in real time.
One of the most interesting features of *Miro* is the integration of the behavioural control paradigm by its own behaviour engine. In practice this permits to implement any kind of software that requires access to the robot's peripherals and that is structured as behavioural architecture, where each behaviour can interact with the other ones. This organization facilitates the subdivision of a complex program in small tasks and consents the reuse of each behaviour in different architectures. Moreover, *Miro* provides a script-based method to create any sort of behavioural architecture; the scripts can be easily created and modified with a nice graphic user interface.
The structure of a behaviour module is also very simple. It is derived from a standard class provided by the *Miro*' source code. The only methods that need to be reimplemented are *init*, called by the behaviour engine the first time the module is activated, and *action*, which is executed periodically by the same engine. In practice, the first function contains all the initial settings necessary to the behaviour, like assigning particular values to some variables or resetting the odometry. The second function instead implements the real behaviour and contains the code to be executed.
Several behaviours can be then grouped to form an *action pattern*, which is a special structure accomplishing a more complex task. Inside the action pattern, each behaviour has a given priority (i.e. the priority of an "avoid-obstacle" behaviour would be higher than a hypothetical "wander" behaviour). Sometimes, an *arbiter* module must be also assigned to an action pattern, for example to handle the cases of

concurrent access from different behaviours to particular devices (i.e. wheels' motors). Finally, an action pattern can interact with other action patterns through the use of *transition*, which can be thought as a sort of signals. A set of action patterns with relative transitions forms the so-called *policy*, which is in practice the final architecture performing the intelligent task of the robot. The reader may also consider a policy like a state machine, where the action patterns are states with transitions among them.

4.2.2 Localization as behaviour

The localization system has been implemented as behaviour within the context of the *Miro* framework. The objective was to make it available as module for complex navigation tasks in future applications. Having the localization system already integrated in *Miro*, indeed, permits an easy interaction with other behaviours and the execution of high-level plans that require the current position of the robot. Practical applications for which such implementation would be useful are, for example, scenarios where the robot acts as waiter or tour-guide[16]. The idea would be inserting the localization behaviour as independent thread, so that it keeps to be updated during the motion. At the same time it could provide the estimation of the current topological position on any other behaviour's request.
The localization module, which we called *Localize* behaviour, makes use of the *TopoMap* object previously described in 4.1.1. This object is created within the *init* function of the behaviour class and initiated with the necessary parameters. During this first stage we also load the topological map (see 4.1.3). The input data, coming from odometry and camera, are then continuously provided to the *TopoMap* object from the other behaviour's method, *action*; after this data insertion, the activities of the nodes inside *TopoMap* are updated and the most probable position estimated. The latter can be eventually used by other behaviours to accomplish tasks dependent on the current robot's location.

4.2.3 A behaviour for creating panoramic images

The procedure of panoramic image reconstruction has also been implemented as behaviour for *Miro* and called *VideoScan*. The first advantage of such module is the opportunity to take automatically a sequence of snapshots rotating on a fixed position; this sequence is then used to reconstruct the relative panorama. The whole procedure is quite fast and normally completed in less than two minutes. The second advantage is that the *VideoScan* behaviour can be integrated in more sophisticated systems of "active localization", in a sense of self-update of the recorded video information, or "map-learning", where the robot discovers and maps unexplored locations.
The behaviour is very simple: the robot performs a full rotation taking snapshots of the surrounding environment and then, calling the *getPanorama* function introduced

[16] Some initial tests on this direction have been carried out but they are still in a very early stage.

in 4.1.2, creates the relative panoramic image. This image then can be used by the *Localize* behaviour.

5 EXPERIMENTS AND RESULTS

In this chapter are illustrated the results of several experiments conducted in the Centre for Hybrid Intelligent Systems of the University of Sunderland. In particular, most of the data has been collected in the Neuro-Robot Laboratory. This consists in a room of approximately $6\times6m^2$, with furniture typical of an office (see Figure 5.22). Along two sides of the room there are ample windows, often cause of light conditions particularly challenging. Sometimes we made use also of an adjacent corridor, about $2\times10m^2$ wide, connected to the laboratory through a small entry. A rough map of the environment can be seen in 0.
The first experiments presented here are relative to the performances of our image-based place recognition. We cover most of the topics treated in chapter 2. The second part of experiments are relative to the whole localization system, therefore including the method described in chapter 3.

5.1 Place recognition performances

The performances of the place recognition are particularly important, since the whole localization strongly depends on it. Even in the method we chose to generate new hypotheses and tracking them, as explained in 3.2, we made the assumption that the place system works fine, at least in most of the cases. Since such system depends on *IMA* and on the panoramic images, we start with some application of the algorithm to a static image (that maybe also helps to better understand how it works). Once some examples of panoramic images have been presented, we test *IMA* on these and we compare the output among different places, thus to see how distinctive it is. We conclude this first part of experiments with the heading-angle extraction, very important for the success of the localization, and with some interesting data about the digital-zoom implementation.

For most the following experiments, we use grey-scale image with a resolution of 72×58 pixel. The number of chosen slots for *IMA* is 8.

5.1.1 Moving obstacles

It has already been explained in 2.1 how the matching procedure works, dividing the new image in columns, that we called *slots*, and then comparing each of them with a

pre-recorded image. In this experiment we want to show the output of *IMA* in typical situations of dynamic environments. In order to avoid the noise introduced by the irregularities of a panoramic image, these tests make use of a single pre-recorded image. Further experiments taking into account the panoramic image's noise are shown later in this chapter.
As first try, we want to plot the output of *IMA* in the most classical situation, i.e. when a person is moving in front of the camera. In Figure 5.1 there are some snapshots taken with a person walking from one side to the other of the observed scene. The resulting output of *IMA* is plotted in Figure 5.2. As expected, such output is close to 1 when no people are in front of the camera and decrease of about 25% when a person is crossing the scene. This decrease, in particular, is proportional to the size of the region occluded by the person. In terms of "columns", he could be considered as a column that shifts left and right, occupying a quarter of the scene.

Figure 5.1 Sequence of a person walking (time-steps 10, 15, 20, 25, 30)

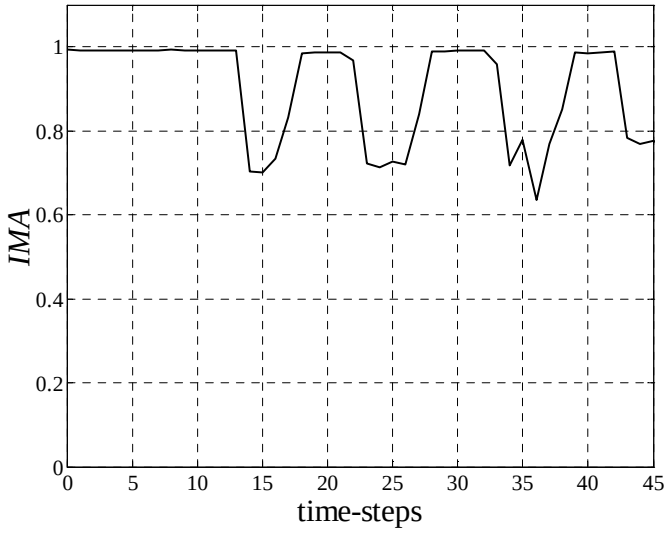


Figure 5.2 *IMA* output for a person walking

In the next example, there is a different situation, but with similar results. The case examined is the opening and crossing of a door, as illustrated on the sequence of Figure 5.3. Again, the decrease of the *IMA* output is close to 25%, as reported in Figure 5.4, and the same considerations of the previous example are applicable.

Figure 5.3 Sequence of door opening (time-steps 5, 15, 25, 35, 45)

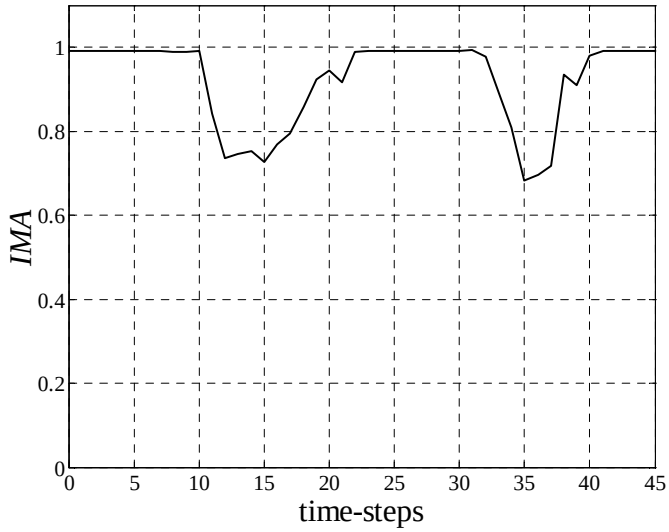


Figure 5.4 *IMA* output for door opening

5.1.2 Examples of panoramic image reconstruction

The reconstruction of panoramic images has a fundamental role in our implementation. Unfortunately, without having an omni-directional vision sensor, is not easy to obtain such images from a normal camera. The first thing we have to deal with is the limit angle of view of this device. In most of the cases, this angle is approximately 40°, which means the minimum number of images to reconstruct a full panoramic view is 9. Of course, even supposed the angle of view is known exactly, many other factors make the reconstruction very difficult. In particular, it would be practically impossible to take snapshots with such a precision that two sequential images fit perfectly. In most of the cases indeed the camera is fixed to the frame of the robot; even if it was provided with a controllable pan system (and this is our case), normally this does not allow a full rotation. The only way to take images of the surrounding environment is thus making the robot rotate with its own wheels. This means imprecision of the angle step between to sequential snapshots. Imprecision that arises as much as we increase the speed at which we want to take the images.
It is clear then that the best solution is having a number of snapshots a little higher than the minimum, in a way that between two of them there is a small over-lap. Taking advantage from the latter one, it is eventually possible to align correctly a sequence of images, two by two, until fill up the panorama. This is the method we used for the examples shown here.

The procedure of image-taking has been made completely automatic implementing an appropriate behaviour in the *Miro* framework (see 4.2.3). The program implemented permits us to take a sequence of images in one or two minutes, depending on the interval chosen between two consecutive snapshots. As said above, the robot simply turns around, slowing down each time the predefined angle step has been reached and then getting the image from the camera. Since the only way to measure the current rotation is reading the odometry, the accuracy of the angle step is very bad. Furthermore, the fact that our platform is a two-wheeled robot and that it has to rotate on a carpet floor increases the error. In most of the cases, indeed, the robot was not able to complete an entire turn, since the final error at the end was about 10-20°. Nevertheless, the panoramic image reconstruction does not suffer of such lack. This was something we already realized before implementing the automatic procedure, when we had to turn the robot manually and take snapshots relying only on the "human precision", without any sort of instruments for measuring angles.
Once a complete sequence of images is available, the software starts the reconstruction of the panorama using an *IMA*'s based function, as described in 2.2. Despite the amplitude of the angle step, two main factors have been noted to influence the success or the failure of the procedure: the number of chosen *slots* for *IMA* and the application or not of the *CLAHE* filter. For the first one, it is obvious that higher is the number of slots, better is the "resolution" in the match of *IMA*, resulting in a more precise alignment of the images. The improvement of the contrast with the *CLAHE* filter permits then to highlight features of the scene, useful for the improvement of the match. Of course, richer is the observed environment of features, higher is the probability that the match succeeds. In the following sequence of images we show some results with different number of slots and the contribute given by the *CLAHE* filter. In Figure 5.5, where only 4 slots has been used for *IMA* and without contrast filtering, the procedure obviously failed, resulting in an incomprehensible overlap of images. The next try of Figure 5.6, where the number of slots has been increased to 8, is definitely better, but there are still errors, in particular due to the failure in the alignment of the big cupboard (right part of the image). A further increase of the slots number is not sufficient to resolve the problem, as shown in Figure 5.7. Instead, introducing the *CLAHE* filtering, the panoramic image in Figure 5.8 is correct. It can be seen also that improving the contrast with such filtering method augments a lot the number of visible features. In particular for the images towards the windows, this means enhancing their distinctiveness.

Figure 5.5 Panorama reconstruction with 4 slots

Figure 5.6 Panorama reconstruction with 8 slots

Figure 5.7 Panorama reconstruction with 15 slots

Figure 5.8 Panorama reconstruction with 8 slots and *CLAHE* filter

The last important consideration is about the angle step. On a first thought, it might seem that with a small angle the final result looks better, since and high number of images "follows" with more accuracy the perspective changes of the scene. In practice, this is not true. Despite the fact that more images means also more time for reconstructing the panorama, increasing the their number (i.e. decreasing the angle step) introduces a significant noise on the final image, consisting of vertical lines in correspondence of insertion. This is particularly evident observing Figure 5.9, where the angle step used was 15°, and Figure 5.10, with a step of 30°. Since this noise decreases the performances of the place recognition, it is always better trying to maximize the angle step, reducing it just when absolutely necessary (for example when the environment is so poor of features that the reconstruction fails). For all the panoramic images of our office environment, including the corridor, a good choice has been an angle step of 30°. With the view angle of our camera, about 40°, this means an overlap of 10° between to consecutive images.

Figure 5.9 Panoramic image with angle step of 15°

Figure 5.10 Panoramic image with angle step of 30°

5.1.3 *IMA* applied to panoramic images

In this section we show some results of the matching algorithm applied to a panoramic image. All the data has been collected during the day, whit people and objects moving, even because a completely static environment was not available. In Figure 5.11 there is the panoramic image reconstructed from the centre of our laboratory. A few minutes later, after the panorama has been recorded and the software reset, we made the robot perform a complete rotation on the same point the images were taken, approximately at 10°/s.

Figure 5.11 Panoramic image of reference

The relative output of *IMA* (with images scaled to 72×58 pixel, 8 slots and *CLAHE* filter) is illustrated by the black line on the graph of Figure 5.12. It can be noticed that the match has a mean value higher than 0.8. The worst cases, for which *IMA* returned about 0.7, are in correspondence of the cupboard (around 100°, right part of the panorama) and the shelves (about 350°, left part of the panorama). This is probably due to a combination of imprecision in the panoramic image and errors derived by the change of the perspective.

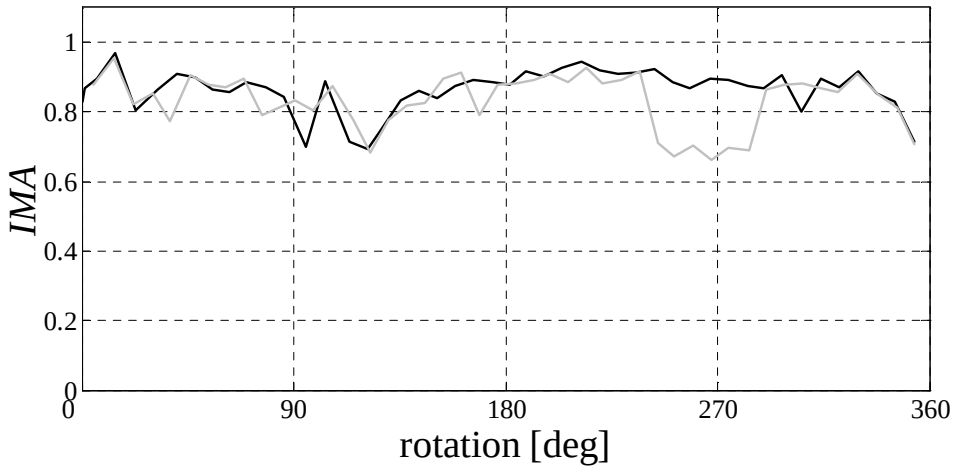


Figure 5.12 *IMA* outputs for panoramic image

On the same graph is illustrated also the output on a further turn, when the person seated in front of the desk moved away. The relative change can be observed on the grey line of Figure 5.12, where the output decreases around 270°, which is in effect the direction where the person was. It is important to notice that, even if the output decreased, the position inside the panoramic image where we had the maximum match (which is also the supposed heading angle) was always correct. Something

different we had instead the day after, when we read again the *IMA* output rotating on the same position and using the panoramic image of the day before. The result is shown in Figure 5.13, where the black line is the old output and the grey one is the most recent. Despite the fact of a small decrease due to light condition and small objects in different positions, the main loss of quality is around 90° and 270°, due to chairs moved (in the second case, also the absence of the person). In particular, the arrow on the graph indicates one point where the supposed position inside the panorama was completely wrong.

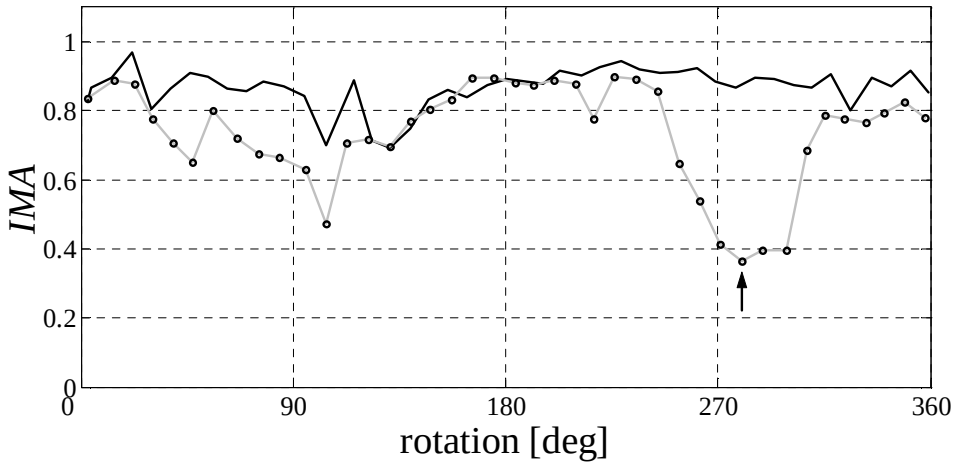


Figure 5.13 *IMA* output the day after

A last reading has been done trying to "simulate" the presence of people moving around the robot. In practice, a person was walking around it during the measure, at a distance of about one meter. The panoramic image of reference was again that one of the day before. The output result is shown in Figure 5.14 with the new grey line. The points A, B, C, D are relative to the instants when the person was in front of the camera and the arrow indicates the only point where the position in the panoramic image was wrong.

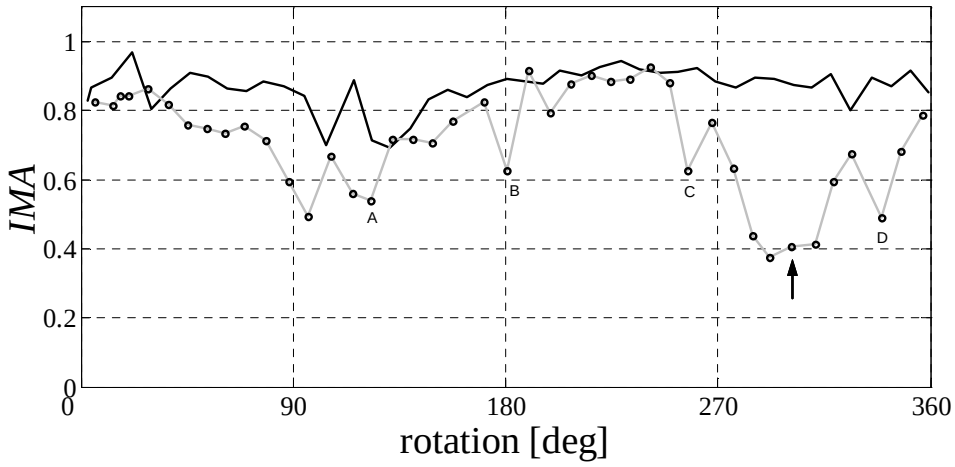


Figure 5.14 *IMA* output the day after with obstacles

The last result about *IMA* applied to panoramic images is perhaps the most important. As its main purpose is distinguishing different locations, we wanted to compare the result obtained in the last case (old panoramic image and moving obstacles) with the output of a comparison between the same snapshots and the panoramic image relative to another position. For the latter, we chose a location in the same room, just one meter far from the original. The resulting output is represented by the black line in Figure 5.15 and compared with the previous one, in grey.

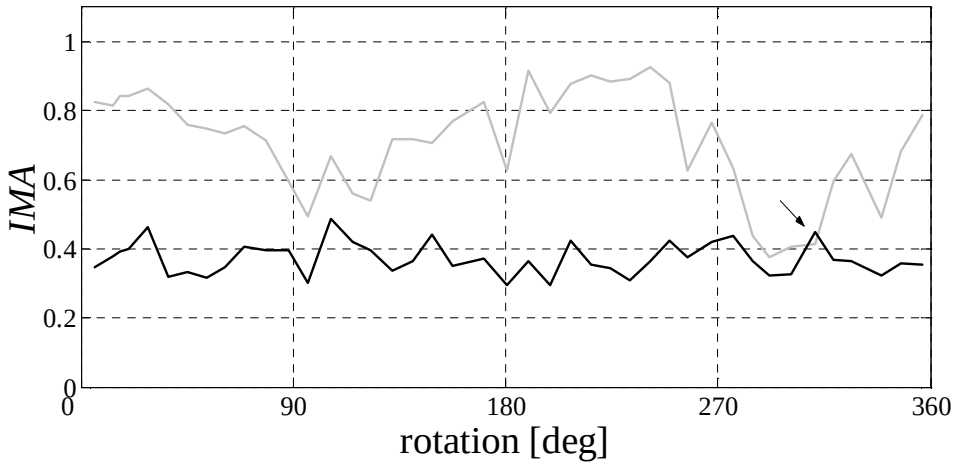


Figure 5.15 *IMA* output from a different position

Even if the output of the second comparison is not very low, in general it is well distinguishable from that one obtained on the right panoramic image. The cases where it clearly fails, like the overlap around 315° indicated by the arrow, would be situations of perceptual aliasing. Here it is evident the necessity of using additional information for resolving the ambiguity, like that one provided by odometry and previous states.

We want to make also some consideration about the number of slots used by *IMA* during the place recognition. When we talked about the panoramic image in 5.1.2, we already saw that in general an increase of the slots number permits a more precise overlap during the procedure of panorama reconstruction. In the case explained there, a number of 8 slots (combined with *CLAHE* filtering) has been sufficient for the success of the procedure, but for most of the our panoramic images we preferred to use a safer number, that is 15. While the increase of such number does not have any "collateral" effect for the panorama reconstruction, the same is not valid during the place recognition process. As we said, the number of slots is a sort of resolution for the matching algorithm. When this is too low, the output of *IMA* is not reliable because is not able to distinguish clearly different scenes. On the other hand, a high number means also an increase of the selectivity that sometimes may be excessive. This can be seen from the values reported in Table 5.1 and extracted from the comparison between Figure 5.16 and Figure 5.17 with the panoramic image in Figure 5.18. The first one is a snapshot taken in a different moment but from the same position where the panorama has been reconstructed. The second one instead is from

another position, but pointing to the same scene. We would like to have a good match for Figure 5.16, even if disturbed by the presence of the chair. At the same tame, we wish the output for Figure 5.17 is as low as possible.

	Figure 5.16		Figure 5.17	
Slots	Match	Angle [deg]	Match	Angle [deg]
8	0.698817	269°	0.428382	325°
15	0.454976	270°	0.270276	326°

Table 5.1 *IMA* output for different slots

From the data above, we can see that the increase of the slots number effectively reduces the output relative to Figure 5.17, but at the same time there is a significant decrement even on Figure 5.16. In particular, the gap between the two different images decreases from about 0.27 (for 8 slots) to 0.18 (for 15 slots). This means a worse distinction between the two cases that may augment the probability of perceptual aliasing.

Figure 5.16 Same position with chair

Figure 5.17 Different position

Figure 5.18 Panoramic image of reference

5.1.4 Precision of the heading angle extraction

In this section we want to show some important results regarding the heading angle extraction using *IMA*. As we already said in 2.3, this is possible applying the simple formula (2.2), supposed we have a direction of reference. In our experiment, we made the robot spinning around a position where a panoramic image was previously reconstructed. We collected data of the heading angle given by the odometry and by the vision during 10 rotations, measuring every 45° real angle. In Figure 5.19 we can observe the final results: on the abscissa there is the real angle, on the ordinate the

heading angle measured by the robot. The grey line refers to the odometry, the black one instead is the angle extracted using *IMA*[17].

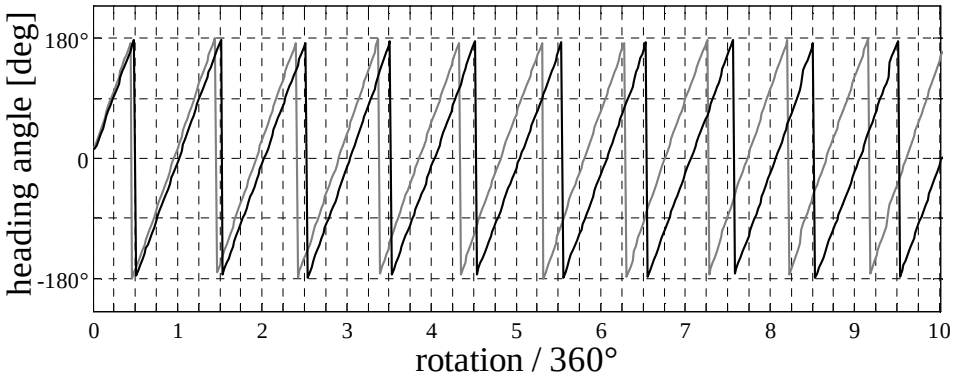


Figure 5.19 Heading angle extraction

It is very clear that the angle given by the odometry, after a few rotations, becomes completely unreliable due to the internal cumulative error. At the seventh rotation its error already reached −45° with respect to the real direction. Instead, the heading angle given from the vision is always between −10° and +10°, without suffering of any cumulative error. The precision of course is not useful for having a perfect measure of the direction; nevertheless it is good enough to correct from time to time the orientation and help the localization.

5.1.5 Effects of the zoom

An important feature of our system is the use of digital zoom for enhancing the place recognition, as explained in 2.4. With the following graphs we want to show in practice what we previously said. In particular we need to demonstrate that the use of the zoom in effect increases the capability to recognize a place, making the robot able to identify not just an exact point in the environment but the whole surrounding area. We show then the variation of this area's dimension depends on the distance of the observed scenes, as mentioned in 2.4.3. The next results are relative to a normal single image of reference, instead of a panoramic one, in order to avoid the noise caused by the latter and cases where the match is relative to a wrong position inside this. Nevertheless, the same principles are valid also when we use panoramic image. Adopting digital zoom, indeed, has been proved to be very useful for our place recognition, as illustrated in 2.4.
In the following illustrations we have the observed scene on the top-left corner and then the relative graphs for five different zoom factors. The first example is reported below, on Figure 5.20, where the distance from the robot and the wall, on the middle of the scene, is about 4 m. The robot has been moved ±1m with respect to the original position. The variation of the *IMA* output is reported on the graphs, where the grey line is the reference, without any zoom, and the black line is relative to the

[17] Please note that the output given by (3.2) has been converted from the interval $[0, 2\pi)$ to $[-\pi, +\pi)$.

current zoom factor[18]. Here, every time we refer to such factor, we mean that one for the zoom-in, called ρ_{in}, and relative to the gaussian on the left. The gaussian on the right is (in theory) symmetrical and depends on the zoom-out factor ρ_{out} given by (3.8).

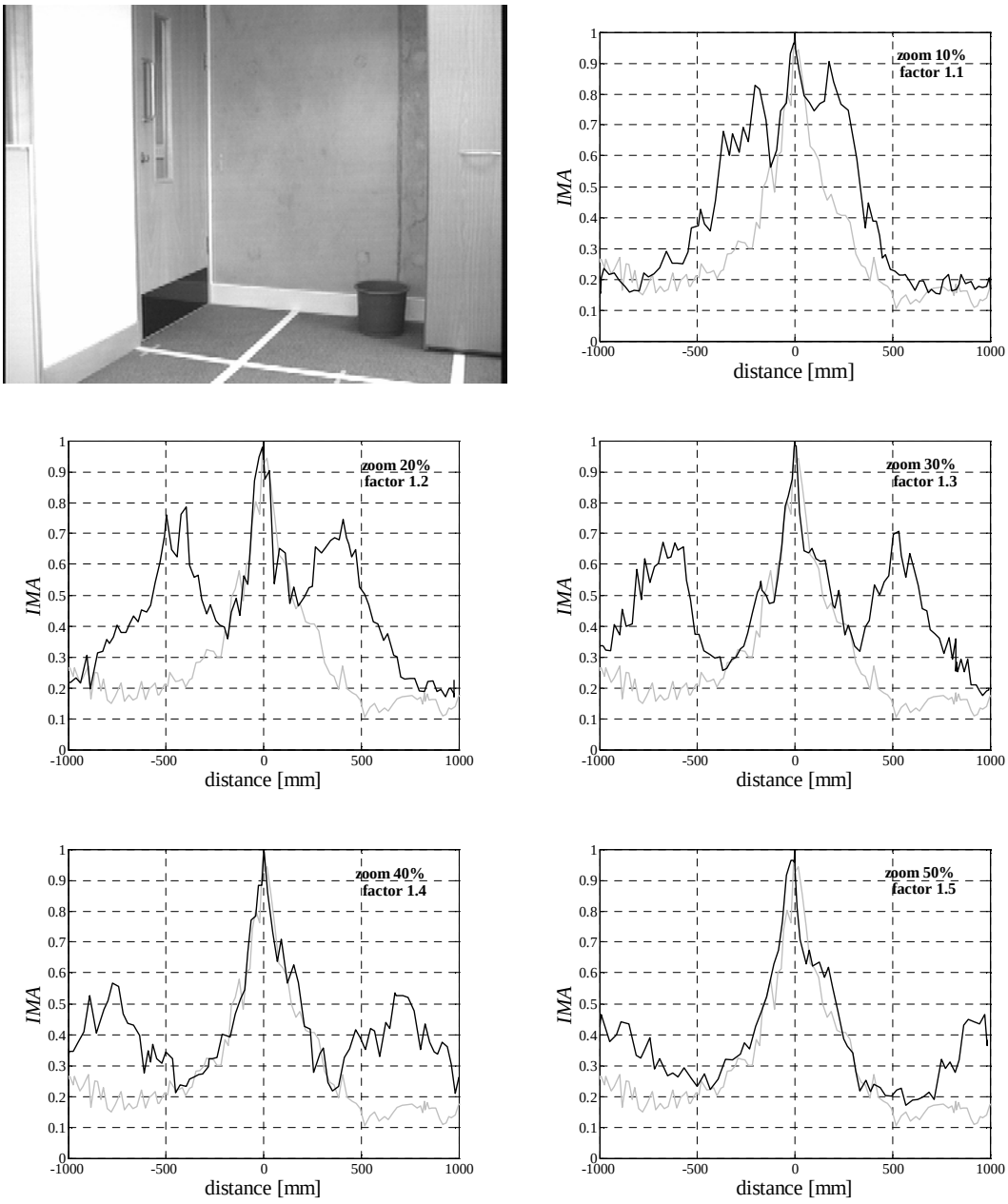
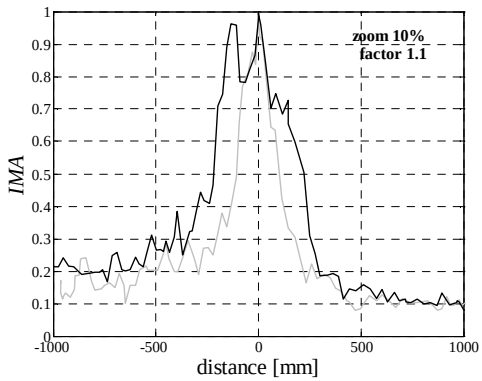


Figure 5.20 Digital zoom performances for a distant scene

[18] We remind from 4.1.5 that, when we use digital zoom, *IMA* is applied three times (no-zoom, zoom-in and zoom-out) and the considered result is the highest one.

Two important considerations arise from the observation of the graphs. First of all, the output is something similar to the combination of three different gaussians, as we already supposed in 2.4.2. Unfortunately, the amplitude of the external peaks decreases considerably augmenting the zoom. Despite the fact it is not easy to keep the robot on the same direction during the translation, the main reason of this decrease is probably the loss of resolution implicitly derived from the zooming process. The second consideration regards the gaps between the external gaussians and that one in the middle. From these we can see that the internal (local) minimum goes quickly below 0.5, already with a zoom of 20%. This is because the width of the gaussians is not very large and the distance from the observed scene is quite long. We remember indeed, from (2.5), that the "virtual" displacement obtained with the digital zoom is directly proportional to such distance. We can finally note that the formula (2.5) was just an approximation for an ideal case, but in practice the "virtual" displacement cannot be calculated in a simple manner. In the case of zoom 20% (ρ_{in} = 1.2), for example, the hypothetical displacement Δx should be 0.67m, but in practice the relative graph shows two external gaussians not farther than 0.5m from the origin. Again, higher is the zoom factor, bigger is the error. Anyway, except for these lacks, we can also consider the first case, with a zoom of %10, a good improvement of the place recognition compared to the output without any zoom. For most of our localization experiments, indeed, this is the value we used (see 5.2).
Now, with the next results we would like to demonstrate, in a simple way, the final considerations we did in 2.4.3 about the use of digital zoom. There, we said that the shape of the recognized area follows that one of the environment. This because it strictly depends on the distance from the observable scene. According to this, we repeated the same test illustrated before, but this time placing the robot just 1 m far from the closest obstacles. It can be seen on the next picture, in Figure 5.21. The robot has been then translated again between ±1m with respect to the original position and the data collected again for five different zoom factors. In the next graphs, the *IMA* output without any zoom is still grey and the black line corresponds to the current zoom.



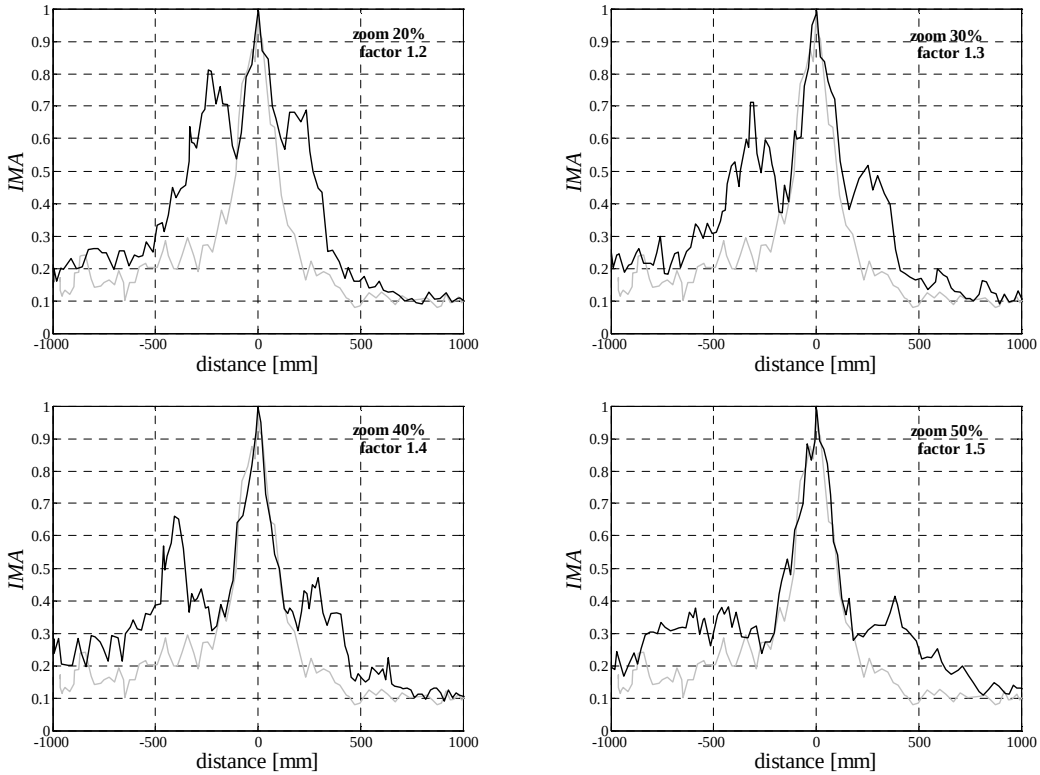


Figure 5.21 Digital zoom performances for a close scene

Comparing these graphs with the previous, it is evident that observing a closer scene the "virtual" displacement becomes shorter. Hence, the recognized area on that direction is thinner. Indeed, on these last graphs the gaussians are in general contained in the interval ±0.5m, while in the previous case this limit was already passed for a zoom of 30%. Finally, even with a such a close scene, a zoom factor ρ_{in} = 1.1 seems to be a good choice.

5.2 Global localization

So far, we examined the performances of the place recognition based only on the video information. The next step is evaluating the behaviour of the whole localization system, in particular how it handles situations of ambiguity, i.e. perceptual aliasing, and cases where the observations provided by the vision is not good enough to estimate a position. In order to do that, we have to take into account the odometry information and integrate it with the video input. This task is accomplished by the algorithm illustrated in Code 3.1.

The reader should keep in mind Figure 5.22, contained in the next section, because in the descriptions of the tests we will often refer to the nodes and sometime to the path there illustrated. Please note also that, when not differently specified, the next experiments have been conducted using the following parameters for the localization systems:

- grey images scaled to 72×58 pixel
- *IMA* slots: 8
- *CLAHE* filtering
- digital zoom: 10% (ρ_{in} = 1.1)
- matching-threshold for destination hypotheses generation: ε_M = 0.5
- matching-threshold for heading angle correction: ε_φ = 0.6
- minimum displacement before updating: 10° of rotation or 0.3m of translation.

5.2.1 Odometry error

First of all, we need to make some considerations about the use of the odometry. It is well known that the information provided by such internal reading is subjected to cumulative errors. These depend on the mechanical imprecision of the sensors for the motion control, normally encoders, but also on external factors like slippage of the wheels or collisions with possible obstacles. In our environment, these last factors have a particular relevance. The whole floor of the environment available for the tests, indeed, is covered with carpet that increases the slippage. Moreover, at intervals of about two meters from each other, there are small griddles for the heating system and every time the robot moves on them, it is shaken because of their small gaps. To have an idea of the unreliability of the odometry for medium-long path, we can simply have a look to Figure 5.22 and Figure 5.23. On the first one it is illustrated a rough map of the laboratory where the tests have been carried out and the path followed by the robot, composed of eight topological nodes on a grid of squares 1m×1m[19]. In the second one, it is shown the position given by the odometry repeating three times the previous path, from 1 to 8. In practice, at the end of the third round, when the robot was back to the initial node 1, the odometry reported a position between 3 and 4. Nevertheless, it can be seen also that for short paths its precision is quite reliable (for example between 1 and 4 in the first round) and can be successfully used for improving the localization system.

[19] Please note that some area of the laboratory, that might be look free on the map, was not available for the presence of other object not represented, like chairs, cables or power suppliers.

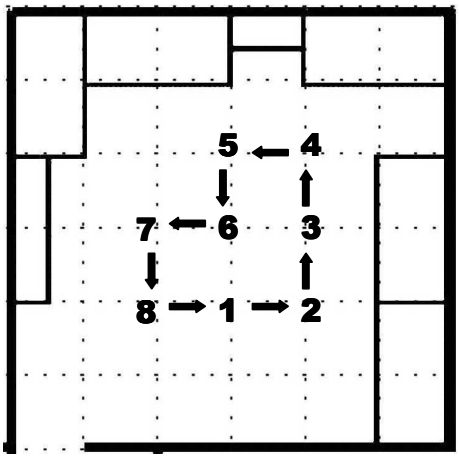


Figure 5.22 Real path of the robot

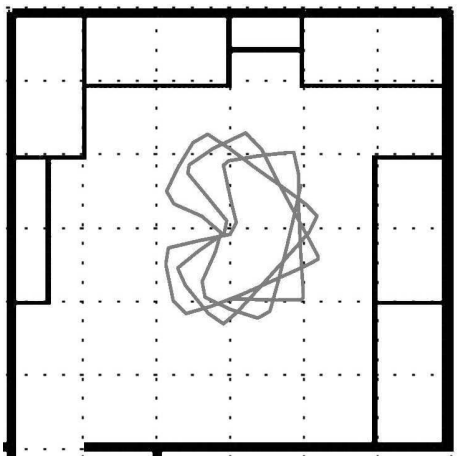


Figure 5.23 Path given by odometry

5.2.2 Initial position and "kidnapped robot"

Before proceeding with the description of the experiments, we want to make clear the independence of the localization procedure from the initial position, important factor that distinguishes global localization from position tracking. When the localization algorithm in Code 3.1 is executed for the first time, the activities are equally distributed among all the possible topological positions, which means also the *origins* set contains all the given nodes and each of them has an initial activity so that the sum is 1. For the path in Figure 5.22, for example, the real initial position was node 1, but at the first iteration of the localization algorithm it was supposed node 5. Nevertheless, since at the beginning all the nodes were considered possible origins, each one with activity 0.125, the correct position was estimated quickly. In particular, as we will see in the following section, the value 0.125 is very low compared to the mean activity of the next estimated destinations, therefore the initial position was normally recognized in a few update steps, as soon as good observation was available.

A similar consideration can be done for the so-called "kidnapped robot" case, that is when the previous estimated location comes out to be completely wrong. Even in this situation, since the localization system is strongly based on its place recognition part, the correct position is recovered in a few update steps, provided that the robot is in proximity of a recognizable place. In practice the kidnapping is resolved only by destination hypotheses generated after a good observation, when one or more image matches returns a value higher than the chosen threshold. For all the tests we did, depriving the robot of the video input and repositioning it without influencing the odometry, the correct destination was recovered in no more than two or three update steps, starting from the moment the video was available and the current position was close to a map's node.

5.2.3 Localization with old panoramic images

As we already said at the beginning of the chapter, the laboratory where most of the tests have been carried out is a square room[20] of about $6\times6m^2$. Looking at the map in Figure 5.22, on the top and right edges there are ample wide windows, so the light conditions changed significantly during the day, even with closed curtains. Most of the furniture consists of similar desks and office chairs, identical computers, shelves and a big cupboard. The desks in particular were the main cause of perceptual aliasing. The bottom part of the room was also occupied by power-supplier, plugs and cables, not illustrated on the map. In practice, the nodes 1-8 covered quite uniformly most of the space available for the robot's move. We will see later some experiments conducted in a bigger environment, where we used a corridor and a small entry adjacent to the laboratory.

In this first localization test, we provided the robot with a map of the possible locations, containing the coordinates and the panoramic images relative to the eight nodes of Figure 5.22. All the panoramic images used for the laboratory were two days old; this means that several small changes in the environment, like different positions of chairs, people and objects, introduced a significant noise on the place recognition process. It is also important to note that, during this test, we kept the robot always at a certain distance from the centre of the nodes, generally moving it on an external perimeter about 30cm far from them. The reason was obviously because we wanted to avoid the exact points where we got the panoramic images and verify also the efficiency of the digital zoom. Therefore, we were not recognizing the precise metrical position but the more general area currently explored. The only case when the robot was exactly on node 1 is at the beginning of the first round, just to speed up the initial localization as explained in 5.2.2. On the following Table 5.2 we can observe a sequence of locations identified by the system at the tenth round. Each line corresponds to a time-step for the execution of the localization algorithm, with the node of the most probable position and relative activity. When no activity is specified, it means there were not new destination hypotheses in that case and the position was calculated only using the previous estimation and the odometry displacement, as explained in 3.8. In this trial the topological positions were always correct. Looking at Table 5.2, it is worthwhile to note that for the node 3 there are not activities because the observations at that moment did not return a matching-value higher than the threshold. In practice, during the path between node 2 and 4 the robot could rely only on the past history and the current odometry. One could also have expected a higher number of updates, since the distance covered in one round was longer than 8m (perimeter of the path 1-8) and the minimum displacement in translation was 0.3m, as we already said. Considering also that several updates were done just turning around the corners, for a minimum angle step of 10°, the theoretical number of localizations should be even higher. In reality there were many other factors that introduced a delay between two consecutive update steps; some of them were the video-processing time, the wireless network communication, the data

[20] Actually, on the bottom edge of Figure 5.22 there is another small area part of the laboratory, about $2\times4m^2$, but it was interdicted to the robot by a step and isolated with a grey panel 1.5m high. Eventual changes inside this area did not influence the vision system.

logging and, last but not least, the fact that the robot was continuously moving, without stopping at any particular location.

node	activity
1	0.380993
1	0.549102
1	-
2	0.386157
2	-
3	-
3	-
4	-
4	0.536337
4	0.572498
4	-
5	0.520742

node	activity
5	0.569589
5	0.277236
6	0.33679
6	-
7	-
7	0.430568
7	0.332822
8	0.55233
8	-
8	0.309363
8	0.326927
1	0.199804

Table 5.2 Localization steps after ten rounds

Unfortunately the localization did not always succeed completely. On a total amount of 255 update steps, we registered 12 cases of errors. Among them, 11 were within 1m, that is the robot thought to be on a topological place adjacent the real one. In one occasion the estimated position was completely wrong. A detailed description of these cases is reported in the following Table 5.3 and Table 5.4.

Number of update steps	255	
Destinations	193	(mean activity: 0.4633)
Normal	153	(mean match: 0.6448)
Virtual	40	
Supposed places	62	

Table 5.3 Description of the update steps

Error cases	12	
Adjacent position	11	
Node 6	6	
Node 7	3	
Node 5	1	
Node 2	1	
Completely wrong	1	
Node 4 (real was 1)	1	(match: 0.644425 - activity: 0.289293)

Table 5.4 Description of the error cases

On Table 5.3 we can observe that during the localization process in 193 cases the returned position was a destination generated by the algorithm, therefore involving the update of the activities. Of these, 153 where normal destinations, for which the observation gave a matching-value (in mean 0.6448) higher than the threshold. The other 40 were instead virtual destinations, as described in 3.2.1. Before analysing the errors for which the real position was an adjacent one, let us consider the last case of Table 5.4, which is the completely wrong position. The main reason of such fail is a situation of perceptual aliasing with a high (wrong) matching-value of 0.644425, comparable to the mean of the correct observations, together with a bad combination of odometry and origins. As can be see from the usual map in Figure 5.22, the robot moved from node 8 to 1, but the scene pointed by the camera (a desk in front of a window) could look very similar in case the robot moved from node 3 to 4.
The other cases, besides being smaller and perhaps more "acceptable" for a dense topological disposition like ours, are also easier to understand. We saw from Table 5.4 that the most frequent error was on node 6, in a sense that it has been omitted several times on the short path 5→6→7, as indicated by the grey arrow between node 5 and 7 in Figure 5.24. The main reason was the fact we were moving following an external perimeter, similar to the grey dashed line in figure, in order to avoid the exact centres where the panoramic images had been reconstructed. It was possible therefore that sometimes, for its position on the concave part of the path, node 5 was omitted. More significantly are instead the other five error cases, which basically suffered of a common lack. These errors are the omission of the nodes 7, 5 and 2, as indicated by the remaining three grey arrows in Figure 5.24, respectively 6→8, 4→6 and 1→3. For them, it has been noted that the problem is mainly due to the way we reset the odometry each time a destination has been estimated. Let us consider the last error, for example. When the robot is moving from node 1 to node 2, in the middle of the tract it might happen that a new observation (and relative update process) returns again node 1 as current position. Consequently the odometry is reset (with the meaning we gave in [14]) to the coordinates of node 1. At that point, if no good observations are available passing through node 2, the robot can estimate its position only using the odometry[21], which could still return node 1 as more proximate. This position would be eventually kept until a new observation recognizes node 3 or the displacement given by the odometry is long enough.

[21] Note that a position given exclusively by the odometry means there are not destinations generated by *IMA*, hence there is not any reset to the coordinates of such position. See the localization algorithm in Code 3.1.

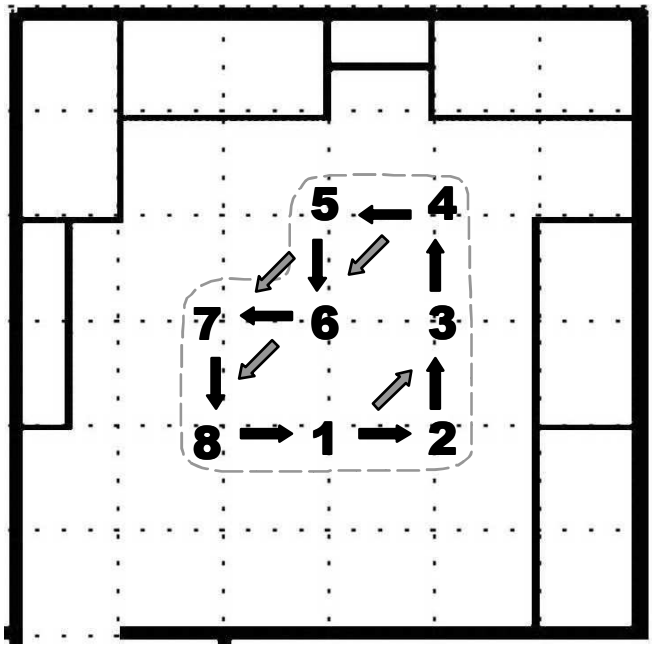


Figure 5.24 Real path of the robot and error cases

As can be seen on the figure, the same consideration could be applied to the other error cases. This suggests a more accurate use of relative displacements should be thought, maybe including a distinction between observations with or without zoom, so to reduce excessive resetting of the odometry.

A last note about the results obtained for this experiment. We would like to remember that the panoramic images used for the place recognition had been reconstructed two days before, as we already said. The same tests performed with fresh images, taken just one or two hours before, gave much better results and no cases of error. This because with updated information the cases of perceptual aliasing were significantly reduced and the few situations with such ambiguity were resolved correctly by an enhanced matching-value. For example, the mean match for a series of correct destinations, generated during a trial with new panoramic images, was 0.73275. Compared with the previous mean, 0.6448, this denotes an increase of about 14% in the matching process. This improvement make us think that a future implementation of active localization, where the robot keeps constantly up to date the internal panoramic images, could successfully handle big changes in the environment.

5.2.4 Localization performances for dynamic environment

The same procedure has been used to test the localization system in a dynamic environment. This time we made use of updated panoramas, reconstructed the same day. The robot performed again 10 rounds following the path of Figure 5.22, still moving on an external perimeter so to avoid the exact centres of the topological position. During these trials, two people were continuously moving around the robot, sometimes walking or standing in front of the camera and sometimes simply sitting

on the chairs. Examples of such situations are illustrated in the robot's snapshots of Figure 5.25.

Figure 5.25 Snapshots of dynamic environment

With the same method adopted for the previous experiment, we can describe this new one with the next Table 5.5. Observing it, we can notice that the results are very similar to the previous case, where the panoramic images were older but the environment was also less influenced by the people.

Number of update steps	253 (90 involving people presence)
Destinations	184 (mean activity: 0.4986)
Normal	150 (mean match: 0.6654)
Virtual	34
Supposed places	69

Table 5.5 Description of the update steps

From the table above, we can see we had almost the same number of total updates, as expected since the length of the path was similar. In particular, checking all the snapshots taken during those updates, we found in 90 of them the scene was partially or completely obstructed by the people. In practice, for about 35% of the path the robot had to handle the loss of information due to human presence. In most of the cases, when the view was just partially obstructed, the localization system was able to correctly distinct the place using the remaining part of image. When instead a person occupied the whole camera's view, the robot relied only on the odometry from the last recognized place, so just "supposing" to be on a certain position. An example of such situation is explained by the sequence in Figure 5.26, where the robot identified the correct place by the first snapshot and then, still moving with the person in front of the camera, estimated the correct positions just with the odometry until two nodes ahead, when finally the person moved out of the view.

Figure 5.26 Complete obstruction of the camera's view

In this experiment the total number of failures was considerably lower than in the previous one: 3 incorrect positions against 12. This is mainly due to the improvement given to the place recognition process with the updated panoramic images. However, the errors occurred during this localization test are in some way more critical, as for all of them the correct positions were not on adjacent nodes (at least with respect to the sequence of the path in Figure 5.22). It seems anyway that the errors, in terms of metrical distance, were limited by a radius of 1.5m, about the length of a square's diagonal (remind the nodes laid on a grid of squares 1m×1m). Since the objective of this experiment was verifying the performances of the localization system when considerably disturbed by people presence, in Table 5.6 we separated the errors occurred while someone was obstructing the camera's view from that one happened without human interference.

Error cases	3	
People	2	
Node 3 (real was 6)	1	(match: 0.678683 - activity: 0.330785)
Node 3 (real was 5)	1	(match: 0.570505 - activity: 0.512187)
No people	1	
Node 1 (real was 7)	1	(match: 0.600372 - activity: 0.517251)

Table 5.6 Description of the error cases

The first two errors are relative to the first two images in Figure 5.27, where the presence of a person obstructed part of the scene. The reduced video information obtained from the real positions node 6 and 5 has not been sufficient to resolve the perceptual aliasing with node 3, even with the help of odometry. The third error case, relative to the last snapshot in Figure 5.27, did not depend on people presence but just on the poor quantity of features of that particular scene.

Figure 5.27 Snapshots of error cases

Analysing the detailed log of the update processes, we noted that the correct positions were always present among the current generated destinations. Unfortunately, in all the three cases the odometry information has not been strong enough to reduce the influence of the wrong hypothesis. This is probably due to the same lack revealed in 5.2.3, that is the rough reset of the odometry, combined also with the particular dense distribution of topological nodes.

5.2.5 Perceptual aliasing

We know that perceptual aliasing is one of the biggest problems in place recognition, independently of the sensor used for the observation of the environment: camera, sonar, laser and so on. With this example we want to show a case where the localization algorithm resolved a situation of perceptual aliasing. The robot was in the position characterized by node 1 on Figure 5.22 and performing a rotation. At a certain time-step, two different destination hypotheses have been generated. These were the real position of the robot, node 1, and another possible destination, node 2. For the latter *IMA* gave a matching-value higher than for the correct place. However, after the update process, the destination with the highest activity was in effect node 1, the right position. In Table 5.7 are shown some values recorded during such update process. On the left there are the possible origins, as result of the previous update, with their relative activities. The node with the symbol ^{v} is the virtual-origin (previous virtual-destination), which is equivalent to the normal origin in this particular case. On the right of the table there are the possible destinations, with a matching-value higher then 0.5. Again, the destination with the symbol ^{v} is the virtual one.

Origin		Destination		
node	activity	node	match (*IMA*)	activity
1	0.470396	1	0.611119	0.418523
1^{v}	0.529604	2	0.65027	0.249694
		1^{v}	0.5	0.331782

Table 5.7 Activities update for solving perceptual aliasing

The destination highlighted is most active and also the correct one in reality. We see that believing only on the video information would have made the localization fail, since node 2 had the highest matching values. Situations like this happened quite often, in particular when the camera was pointing to a scene considerably changed from the original one, or when some temporary obstacle, like a person, was obstructing the view.

5.2.6 Virtual destination

In the following case, we illustrate an example where the virtual-destination, introduced in 3.2.1, helps to localize the robot correctly, despite the absence of a correct observation. Referring again to the path illustrated in Figure 5.22, the robot moved from node 8 to 1. In this case it was not able to recognize the correct position with the vision. Instead, it got an observation saying that it was in node 2 with a matching-value 0.541098. Nevertheless, at the end the virtual-destination 1^{v} had the higher activity and the localization succeeded. The exact values involved in this step are reported below on Table 5.8.

Origin		Destination		
node	activity	node	match (*IMA*)	activity
1	0.235045	2	0.541098	0.441696
8	0.341719	1^v	0.5	0.558304
4	0.129596			
8^v	0.29364			

Table 5.8 Activities update for virtual-destination choice

5.2.7 Heading angle correction

We saw in 3.7 the importance of the heading angle, directly connected to the orientation of the internal frame of reference. We also showed in 5.1.4 that it is possible to retrieve the absolute orientation of the robot using the panoramic images, whenever these have been reconstructed starting from the same absolute direction. This orientation was not influenced by cumulative errors, like the odometry, and its precision was good enough for correcting the internal heading angle. With the following experiment we want to demonstrate that in effect this method has been applied successfully in our localization system. In some way, it can substitute a compass whenever this is not available or the magnetic disturbances in the environment do not permit its use. Referring to the next Figure 5.28, we started from a situation where the robot was corrected localized on node 1 and the odometry reset. Then we covered the camera view, so that the robot was not able to use the video input and could only localize itself with the odometry, as if it was "blind". Still from node 1, we turned the robot of about −90° (quarter of clockwise revolution) without influencing the odometry and then we made it performing the same complete path we used for the previous experiments.

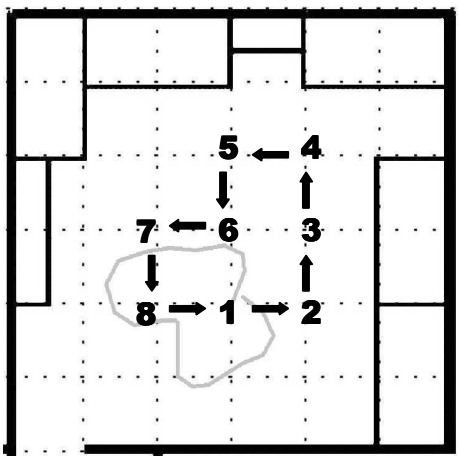


Figure 5.28 Wrong heading angle

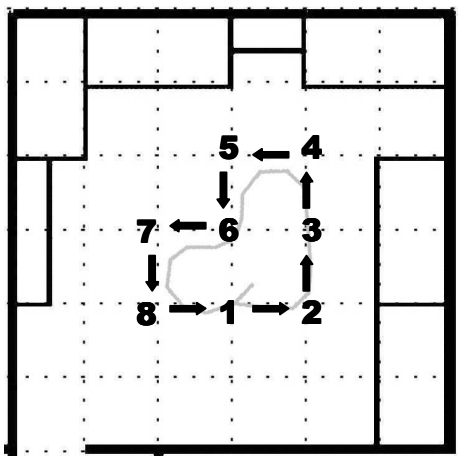


Figure 5.29 Heading angle corrected

In Figure 5.28 we can see the real path indicated by the arrows and the path given by the odometry with the grey line. As expected, the shape is almost similar, but the odometry's is clearly rotated of 90°. Of course, while the robot moved without using vision, the nodes estimated with the only odometry were completely wrong, in order 1→6→7→8→1→2→1 (we omitted here repetitions of the same node). After having completed the first round, we uncovered the camera and let the robot relocalize itself on node 1 with a couple of update steps. Finally we obstructed the camera again and moved the robot following the previous path. This time the odometry's track was well oriented, as shown from the grey line in Figure 5.29, and the reported sequence of nodes was 1→2→3→4→5→6→7→8→1. This means also the internal frame of reference was correctly aligned with that one of the map, thanks to the last extraction of the heading angle from the panoramic image of node 1.

5.2.8 Localization in a bigger environment

So far we presented results for experiments conducted in the laboratory. Though such room was very challenging, in particular for the presence of repetitive furniture and difficult light conditions, we wanted to show the performance of the localization system in a bigger environment. We then mapped an adjacent corridor connected to the laboratory through a small entry, as shown in Figure 5.30. The new added places are quite narrow, just $2\times2m^2$ for the entry and about $2\times10m^2$ for the corridor, and both illuminated only by artificial light. We already saw, in the previous Figure 5.10, an example of panorama taken from the corridor.

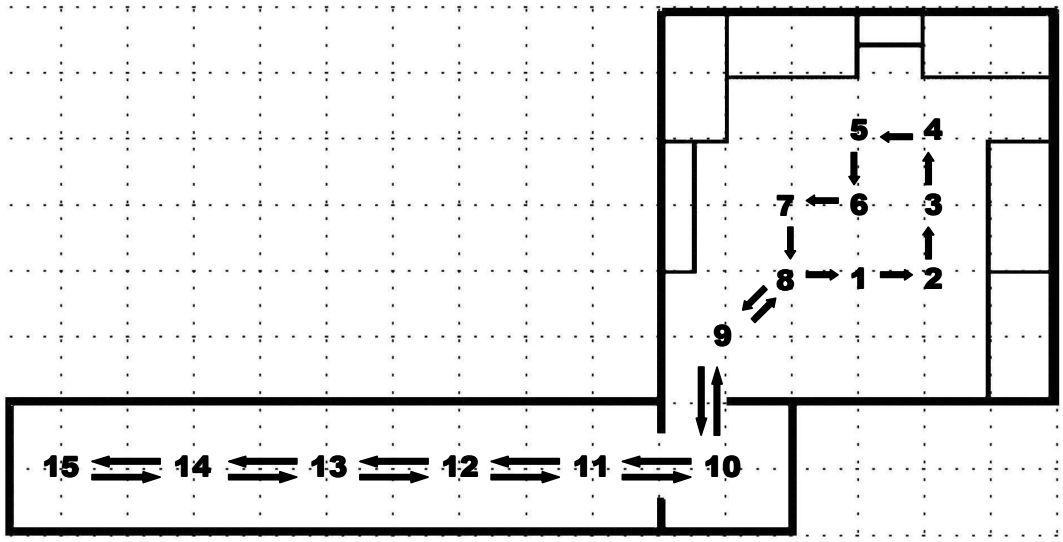


Figure 5.30 Map of laboratory and corridor with reference path

For this enlarged environment we fixed a new path of reference, extended that one used until now in a way to cover all the length of the corridor. This path is indicated by the arrows in Figure 5.30. In practice, starting again from node 1, the robot followed again the sequence of nodes from 1 to 8. There it changed to 9 and moved towards the end of the corridor, which is node 15. Finally it came back to reach again

node 1. It is interesting to have a look to the path measured by the odometry after just one trial. We can see from Figure 5.31 that, instead of terminating on the starting point node 1, the final measure reported a position several meters far and with an error of almost 90° for the direction (compare the orientation of the final track with the direction of the real sub-path 9→8→1).

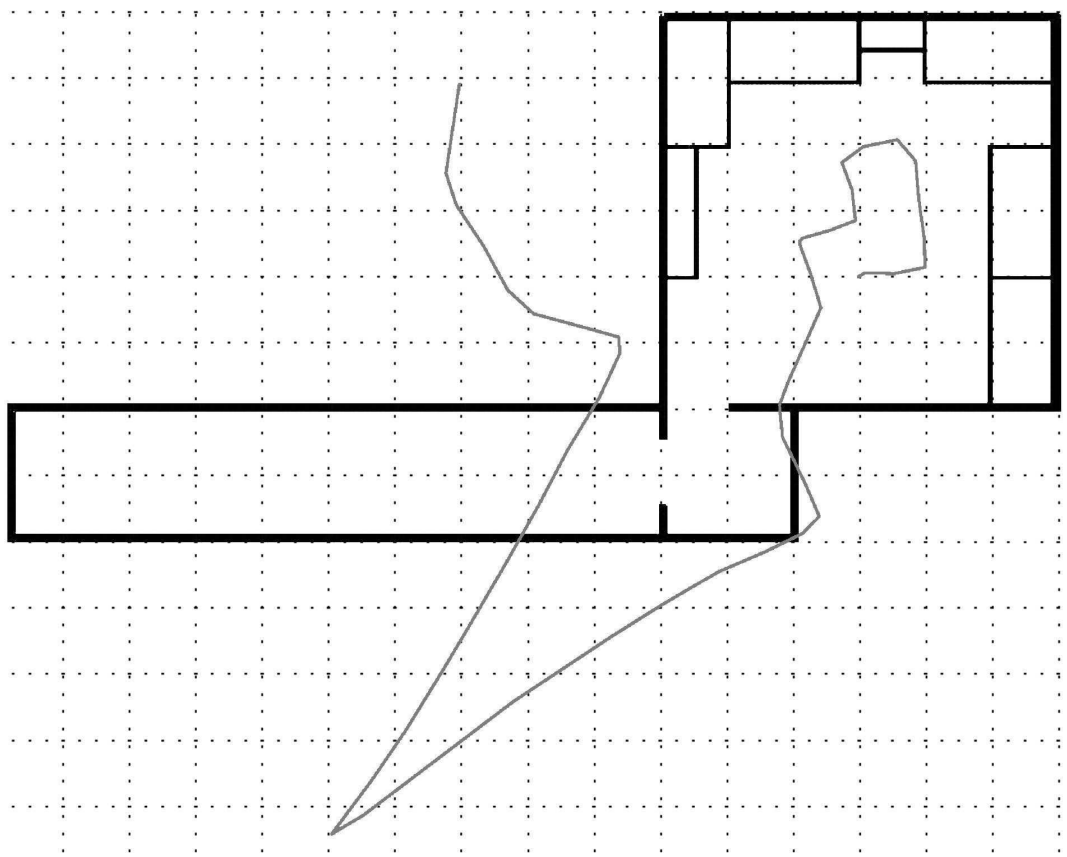


Figure 5.31 Odometry error for path including corridor

Below we report some data related just to one trial, since successive repeats of the same path returned similar results and are therefore not very significant. Like for the former cases, the robot moved always avoiding the exact centre of the topological places. All the parameters of the localization algorithm (slots, zoom, thresholds, etc.) were the same previously adopted for the laboratory. The only difference was the use of additional panoramic images for the added nodes (together with an updated topological map, of course). Here some specification must be done. Between two different rooms there are doors, not represented on the map, so one door between the laboratory and the entry and another one for going into the corridor. The panoramas of the nodes between 1 and 8 had been reconstructed in the same day of the experiment but with closed doors. For node 9, instead, we provided the robot with a new panoramic image taken with the doors open. All the remaining nodes, from 10 to 15, had images taken with closed doors. This condition, besides the fact that made us save some time for the panoramas' reconstruction, was useful to introduce some additional noise on the place recognition. Furthermore, the latter images had been

taken a few weeks before but this did not influence the place recognition since practically there has not been any change in the entry or the corridor. Details of this experiment, after following the new path once, are reported in Table 5.9

Number of update steps	51	(30 in the laboratory, 21 in the entry / corridor)
Destinations	33	(mean activity: 0.5026)
Normal	23	(mean match: 0.6642)
Virtual	10	
Supposed places	18	

Table 5.9 Description of the update steps for path including corridor

From the table above, we can see that the averages of activity and matching-value are in practice the same as the previous experiment conducted only in the laboratory. What is considerably changed is the number of normal destinations compared with the virtual ones and the supposed places. The decrease of normal destinations means that in general the place recognition was less reliable and more credibility has been done to the odometry information. Most of the cases of virtual destination or supposed place happened inside the corridor and the entry, showing the difficulty of recognizing places in narrow places like those. Nevertheless, the reduced number of good observations was enough for the localization to succeed. The only error indeed was missing node 10 (the small entry) when the robot was coming back from the corridor. This was probably due to the doors, closed instead that open like in the recorded panorama, and to the inaccurate odometry reset, as we already mentioned in 5.2.3 and 5.2.4.

5.2.9 Some considerations on the experiments

We would like to conclude the experimental evaluation of the localization system with some clarifications. First of all, the reader should have noted that for the whole set of experiments here presented we always made use of the digital zoom. Indeed, such feature was an essential component for the success of the localization, since without it the system would not have been able to identify most of the topological nodes, at least those for which the robot did not pass through exactly.
We want also to specify that in these experiments we made use of fixed, circular paths just for having simple and clear references. In reality, the number of tested routes was much higher, including more or less nodes, fixed or random paths and so on. A particular case that is worthy to mention is the random autonomous navigation inside the laboratory using an additional omni-directional vision sensor. This device has been designed by the author and utilized by a colleague for obstacle-avoidance [MDA04]. Details on the design are reported in Appendix 0. During the navigation inside the laboratory, with the presence of fixed or moving obstacles, the localization software was also active and kept track of the current robot's position. Even if still in an early stage, the successful combination of the localization and the obstacle-

avoidance modules let us hope in promising results for a more sophisticated navigation system in the future.

6 CONCLUSIONS

6.1 Evaluation

We presented an image-based localization system for indoor environments that makes use of a simple unidirectional camera and odometry information. Our approach is strongly based on place recognition, for which we developed a new algorithm that makes use of classical image processing. The same algorithm permits also the generation of panoramic images used for mapping the environment. Within a probabilistic framework, the odometry is then integrated with the video information to resolve cases of ambiguity. Finally, the software implementation of the localization system has been thought in a way to be as more general as possible, so to be easily portable on any robot platform provided with a camera.
The image-matching algorithm here adopted does not rely on particular features of the environment. Compared to other methods based on landmarks, either natural or artificial, our system takes into account much more information for the place recognition task. This is true also if we consider solutions based on histogram matching. Furthermore, we do not make use of any complicated sensor model for our vision system, which is often one of the critical points in the performances of many localization approaches.
The procedure for generating panoramic images and the relative heading angle extraction gave remarkable results, considering of course the hardware limitations we had to deal with. The numerous experiments presented show also the robustness of our approach, even in case of dynamic environments, making the localization suitable for service-robot applications.
However, from our results arose also the necessity of improving the image-matching algorithm to make it less sensible to occlusions and light conditions. A more appropriate way of resetting the odometry would also help to reduce considerably cases of failure.

6.2 Recommendation for further work

Besides resolving the lacks mentioned above, there are two main topics that would be worth exploring in the future: the automatic update of panoramic images and the use of incremental digital zoom. The first one could certainly boost up the place

recognition, becoming also a natural step towards a complete system of self-localization and map-learning (SLAM). The second one is an innovative technique that we have just introduced but which shows great promise to improve the effectiveness of the localisation. By adding incremental digital zoom to the frame captured by the camera indeed, we can identify “off node” map locations. It would be interesting extending this technique to support interpolation of location between map nodes.

ACKNOWLEDGEMENTS

There are several people I would like to thank. First of all my supervisor Prof. Enrico Pagello, for the education and the continuous support I received from him in these years, and all the components of the Intelligent Autonomous Systems laboratory. The next is Prof. Stefan Wermter and his staff at the Centre for Hybrid Intelligent Systems, where I had the pleasure to work during the last year. A special thank to Dr. Cornelius Weber for his help and precious suggestions.

APPENDICES

A Normalized Correlation Coefficient

Normalized Correlation Coefficient, or simply *NCC*, is a measure of the similarity between two images. The equation of *NCC* is as follows:

$$c(x,y)=\frac{\sum_{a,b}\left[T(a,b)-T_m\right]\cdot\left[I(x+a,y+b)-I_m(x,y)\right]}{\sqrt{\sum_{a,b}\left[T(a,b)-T_m\right]^2\cdot\sum_{a,b}\left[I(x+a,y+b)-I_m(x,y)\right]^2}} \tag{A.1}$$

$$T_m=\frac{1}{w\cdot h}\sum_{a,b}T(a,b) \tag{A.2}$$

$$I_m(x,y)=\frac{1}{w\cdot h}\sum_{a,b}I(x+a,y+b) \tag{A.3}$$

T is the template image, of dimension $w{\cdot}h$ pixel, and T_m is the relative mean brightness. I is the reference image, equal or greater than T, and $I_m(x, y)$ is the relative mean brightness calculated for an area $w{\cdot}h$ starting from the pixel (x, y).
NCC is normalized since the output range is in the interval [−1, +1]. However, for our purpose, such range is modified to lay on [0, +1]. This is done with the following simple conversion:

$$c^*(x,y)=\frac{c(x,y)+1}{2} \tag{A.4}$$

B The PeopleBot robot

The robot used for the experiments is a Performance PeopleBot™ produced by ActivMedia Robotics and available at the Centre for Hybrid Intelligent Systems, University of Sunderland. This robot, which is based on a standard two-wheels Pioneer-2 platform, is particularly indicated for studies and applications involving human-robot interaction. The PeopleBot indeed is quite tall (the height of the top deck is 1115mm) and rich of devices that make it feasible for applications including navigation, vision, speech and simple objects handling. Below we illustrate the main features of the robot; they can also be observed on the draft in Figure B.1.

BASIC FEATURES

- Computing hardware: integrated PC with CPU Pentium III, 700MHz and 256MB of RAM. Support for PC104 expansion cards.
- Operating Systems: RedHat Linux and MS Windows
- Networking: integrated PCMCIA Wireless Card (with additional antenna on the top deck) and RJ-15 connector for Ethernet communication
- Autonomy: three batteries 12V, 7Ah (total 252Wh)

SENSORS / INPUT

- Odometry: quadrature shaft encoders, 500 ticks per revolution
- Vision: colour PTZ camera, video resolution 380.000 NTSC pixels
- Audio: two microphones
- Proximity: sonar arrays (bottom and front-top of the robot), protective IRs and bumpers
- Object-detectors: IR table-sensors and gripper breakbeams

ACTUATORS / OUTPUT

- two DC motors (wheels)
- gripper (2 d.o.f.) with grasping pressure control
- two speakers

A real image of our robot, called *MIRA*, can be observed in Figure B.2, where it is also equipped on the top with an omni-directional vision sensor for obstacle avoidance (see Appendix C).

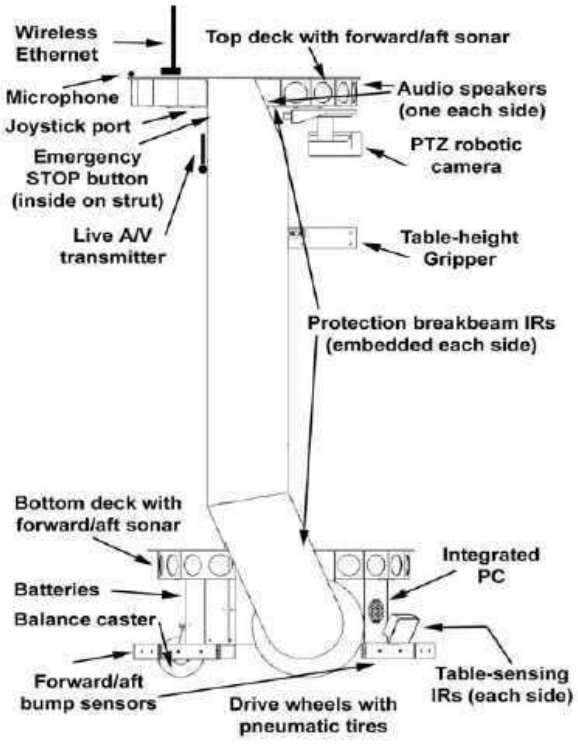


Figure B.1 PeopleBot features

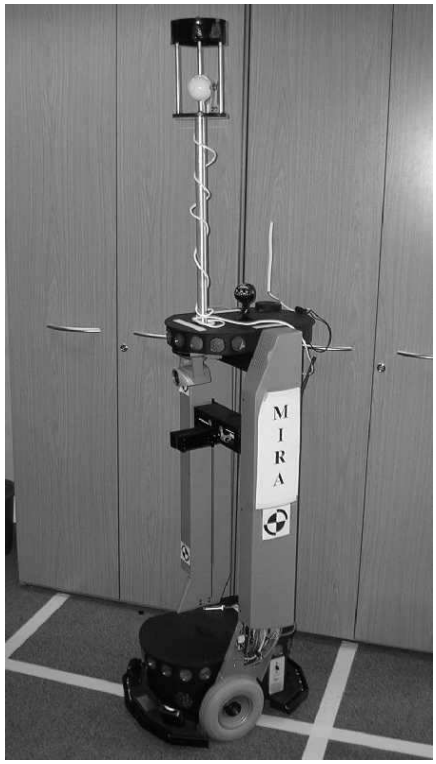


Figure B.2 *MIRA* robot

C Omni-directional vision sensor for obstacle avoidance

In Mobile Robotics, an essential part of the navigation task is the Obstacle Avoidance module. Nowadays, many techniques still adopt simple sonar sensors. Unfortunately these are proved to be very unreliable in the mid-long range and, even in the short range, the quality of the measures strongly depend on the physic characteristics of the obstacles (i.e. material and shape). In this sense, an omni-directional vision sensor is a valid alternative. Therefore we designed and implemented an inexpensive omni-directional camera based on a simple conical reflector.

C.1 An overview of the omni-directional vision sensor

The whole system consists basically of three main components: a camera, a reflector and a frame to sustain and mount them on the robot. The way it works is quite simple and is illustrated in Figure C.1. On the top there is the reflector, the shape of which could be conical, hemi-spherical, parabolic or any user defined axial symmetric profile. The surface of the reflector is polished to make it into a mirror. Below the mirror, a camera is mounted with the optical axes aligned with the axes of the reflector. The camera captures the image mirrored by the reflector, which provides a 360° view of the environment around the robot. The conical view volume and image transfer function depends upon the shape of the reflector. The system can be realized with relative inexpensive components, whilst enabling a high quality of images to be retrieved. In our design, the use of an aluminium reflector and a simple webcam give remarkable results. The reflector is aluminium made, built and polished on Numeric Control machines. The camera, a compact commercial USB webcam with a resolution of 640x480 pixel and a frame-rate of 30 fps, is suitable for visual processing in real-time tasks. Both these components are sustained by a robust frame, built using light materials like aluminium and plexiglas type. Several adjusters are provided to align the optic axes of the reflector and camera.

C.2 Design of the conical mirror

The easiest shape of the reflector, both for the realization and for the image interpretation, is a cone. The dimensions of the conical reflector must be calculated considering its height from the floor and the range of the panoramic view we want to obtain. Of course, we must take care of some constraints given by the available technology. Using a general procedure that consists of three simple steps, the dimensions of a conical reflector can be easily calculated. All the symbols in the following formulas are explained by Figure C.1, Figure C.2 and Figure C.3.

1) Given *d* and *H*, calculate the angle γ:

$$\gamma = \frac{1}{2}\arctan\left(\frac{d}{H}\right) \tag{C.1}$$

2) Given D, calculate the angle β (we considered r << H and h << H):

$$\beta \cong \arctan\left(\frac{D}{H}\right) - 2\gamma \tag{C.2}$$

3) Finally, the next relation can be used to dimension the reflector:

$$\frac{l}{h} = \frac{1}{\tan\beta \tan\gamma} - 1 \tag{C.3}$$

From the last step, fixing one dimension, *l* or *h*, it is possible calculate the other one (note also that $h = r \cdot \tan\gamma$).

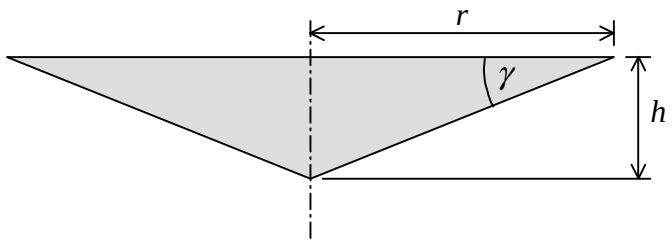


Figure C.1 Dimensions of the conical reflector

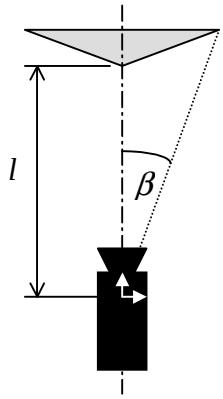


Figure C.2 Dimensions of the system camera-reflector

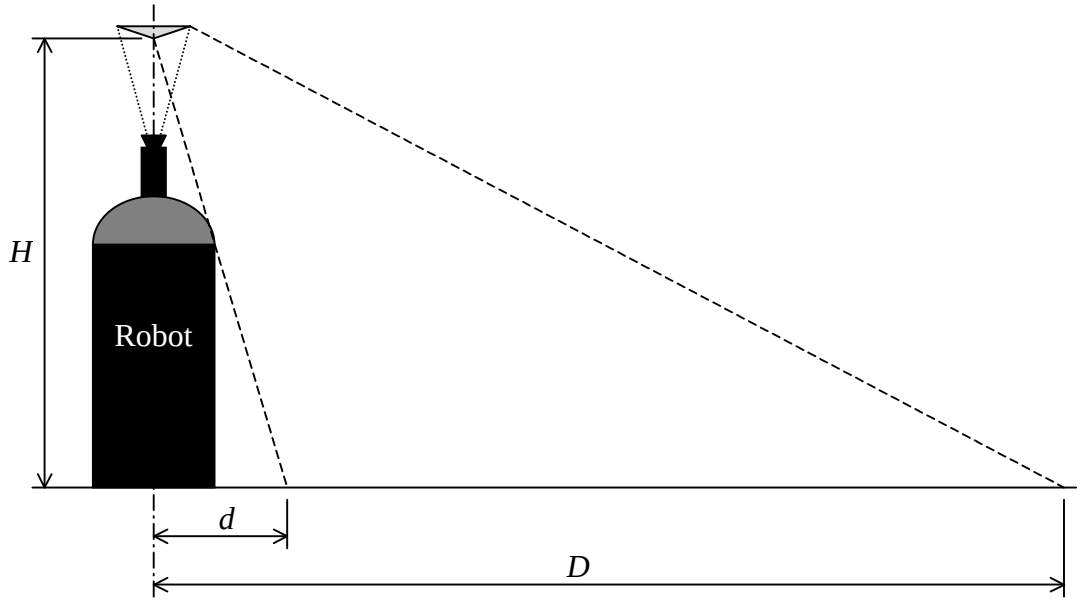


Figure C.3 Height and range of view of the omni-directional vision sensor

C.3 Camera and supporting frame

The system can be realized with relative inexpensive components, whilst enabling a high quality of images to be retrieved. In our design, the use of an aluminium reflector and a simple webcam give remarkable results. The reflector is aluminium made, built and polished on Numeric Control machines. The camera, a compact commercial USB webcam with a resolution of 640x480 pixel and a frame-rate of 30fps, is suitable for visual processing in real-time tasks. Both these components are sustained by a robust frame, built using light materials like aluminium and plexiglas type. Several adjusters are provided to align the optic axes of the reflector and camera. The whole omni-directional vision sensor is shown in Figure C.4 and also in the previous Figure B.2, where is mounted on the robot. Finally, Figure C.5 reports an example of omni-directional image taken with our sensor.

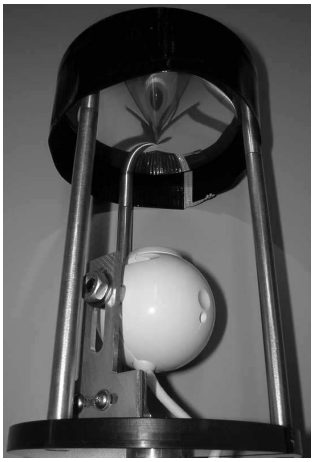


Figure C.4 Omni-directional vision sensor

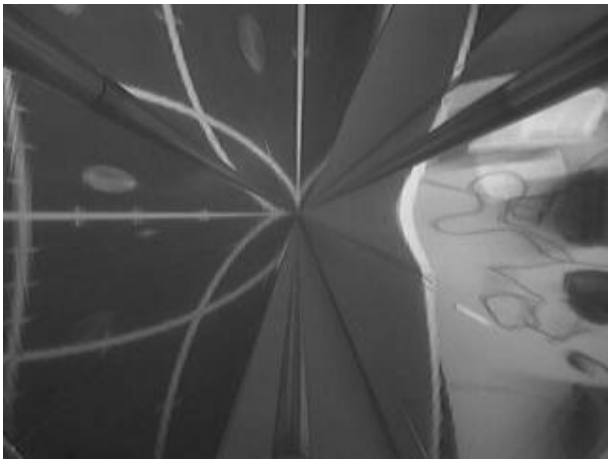


Figure C.5 Example of omni-directional image

BIBLIOGRAPHY

[BCF99] Burgard W., Cremers A.B., Fox D., Hähnel D., Lakemeyer G., Schulz D., Steiner W., Thrun S. (1999). *Experiences with an interactive museum tour-guide robot*. Artificial Intelligence 00 (1999), pp. 1-53.

[BFHS96] Burgard, W.; Fox, D.; Hennig, D.; and Schmidt, T. 1996. *Estimating the absolute position of a mobile robot using position probability grids*. In Proceedings of the Thirteenth National Conference on Artificial Intelligence. Menlo Park: AAAI.

[DMS02] Duckett T., Marsland S., Shapiro J. (2002). *Fast, On-Line Learning of Globally Consistent Maps*. Autonomous Robots 12, pp. 287-300.

[DN01] Duckett T., Nehmzow U. (2001). *Mobile robot self-localisation using occupancy histograms and a mixture of Gaussian location hypotheses*. Robotics and Autonomous Systems 34 (2001), pp. 117–129.

[DN01] Duckett T., Nehmzow U. (2001). *Mobile robot self-localisation using occupancy histograms and a mixture of Gaussian location hypotheses*. Robotics and Autonomous Systems 34 (2001), pp. 117-129.

[DPLR77] Dempster A.P., Laird N.M., Rubin D.B. (1977). *Maximum likelihood from incomplete data via the em algorithm*. In Journal of the Royal Statistical Society, volume 39 of B, pp. 1-38.

[EFR01] Enderle S., Folkerts H., Ritter M., Sablatnög S., Kraetzshmar G., Palm G. (2001). Vision-based Robot Localization using Sporadic Features. Workshops Robot Vision 2001, Auckland, New Zealand.

[FM00] Filliat D., Meyer J.A. (2000). *Active perception and map-learning for robot navigation*. In "From Animals to Animats 6", Proceedings of the 6th Conference in Simulation of Adaptive Behavior. The MIT Press.

[FM02] Filliat, D., & Meyer, J. A. (2002). *Global localization and topological map learning for robot navigation*. In From animals to animats 7. The seventh international conference on simulation of adaptive behavior (SAB02).

[FM03] Filliat D., Meyer J.A. (2003). *Map-based navigation in mobile robots – I. A review of localization strategies*. Cognitive Systems Research 4 (2003) 243-282.

[Fox98] Fox D. (1998). *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*. Doctoral Thesis. Institute of Computer Science III, University of Bonn, Germany.

[GBFK98] Gutmann J.S., Burgard W., Fox D., Konolige K. (1998). *An Experimental Comparison of Localization Methods*. Intl. Conference on Intelligent Robots and Systems (IROS-98).

[GF02] Gutmann J.S., Fox D. (2002). *An experimental Comparison of Localization Methods Continued*. Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS-02).

[GM00] Gini G., Marchi A. (2000). Indoor Robot Navigation with Single Camera Vision. Proceedings of Pattern Recognition in Information Systems (PRIS 2002).

[JK01] P. Jensfelt and S. Kristensen (2001). *Active global localisation for a mobile robot using multiple hypothesis tracking*. IEEE Trans. on Robotics and Automation, 17(5), pp. 748-760.

[KJ03] Kristensen S., Jensfelt P (2003). *An Experimental Comparison of Localisation Methods, the MHL Sessions*. Proceedings of the 2003 IEEE/RSJ Intl.Conference on Intelligent Robot and Systems.

[KMS02] Kidono K., Miura J., Shirai Y. (2002). Autonomous visual navigation of a mobile robot using a human-guided experience. Robotics and Autonomous Systems 40, pp. 121–130.

[MDA04] Muse D.A. (2004). *Navigation and Obstacle Avoidance Using an Omni-Directional Camera*. MSc Project Dissertation, University of Sunderland.

[MIR03] Enderle S., Kraetzschmar G., Mayer G., Pages G., Sablatnög S., Simon S., Utz H. (2003). *Miro Manual - version 0.9.4*. University of ULM, Germany.

[MMA99] Maxwell B.A., Meeden L.A., Addo N., Brown L., Dickson P., Ng J., Olshfski S., Silk E., Wales J. (1999). *Alfred: The Robot Waiter Who Remembers You*. AAAI Conference 1999.

[MO88] Moravec, H. P. 1988. Sensor fusion in certainty grids for mobile robots. *AI Magazine* 61–74.

[MPP04] Menegatti E., Pretto A., Pagello E. (2004). *A New Omnidirectional Vision Sensor for Monte-Carlo Localization*. In Proceedings of RoboCup Symposium 2004 (to appear)

[OHD97] Oore S., Hinton G.E., Dudek G. (1997). *A mobile robot that learns its place*. Neural Computation 9(3), pp 683-699.

[PP02] ActivMedia Robotics (2002). *Performance PeopleBot™ Operations Manual*. Version 1.0, Sept. 2002

[Pre04] Pretto A. (2003). *Localizzazione di Monte Carlo in ambiente RoboCup*. Laurea Thesis, Università di Padova, Italy.

[TBB99] Thrun S., Bennewitz M., Burgard W., Cremers A.B., Dellaert F., Fox D., Hähnel D., Rosenberg C., Roy1 N., Schulte J., Schulz D. (1999). *MINERVA: A Second-Generation Museum Tour-Guide Robot*. Proceedings of the IEEE international conference on robotics and automation (ICRA-1999). IEEE Press.

[Thr98] Thrun S. (1998). *Finding landmarks for mobile robot navigation*. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 98).

[Thr99] Thrun S. (1999). *A bayesian approach to landmark discovery in mobile robot navigation*. Machine Learning.

[UN00] Ulrich I., Nourbakhsh I. (2000). *Appearance-Based Place Recognition for Topological Localization*. IEEE International Conference on Robotics and Automation (ICRA 2000), pp. 1023-1029.

[XYOH03] Xuan Dao N., You B.J., Oh S.R., Hwangbo M. (2003). *Visual Self-Localization for Indoor Mobile Robots Using Natural Lines*. Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems.