Attentive visual memory for robot localization

Julio Vega, Eduardo Perdices, Jose María Cañas

Abstract

Cameras are one of the most relevant sensors in autonomous robots. Their main challenges are to extract useful information from captured images and to manage the small field of vision of regular cameras, visual attention systems and active vision may help to overcome them. This work proposes a dynamic visual memory to store the information gathered from a moving camera on board a robot, an attention system to choose where to look at with this mobile camera, and a visual localization approach which uses this visual memory. The visual memory is a collection of relevant task-oriented objects and 3D segments, and its scope is wider than than the current camera field of view. The attention system takes into account the need to reobserve objects in the visual memory, explore new areas and test hypothesis about object existence in the robot surroundings. The visual memory is useful also in localization tasks since it provides more information about robot surroundings than the current instantaneous image. Several experiments have been carried out both with simulated and real robots. The system has been programmed and validated in real robots that use the information in the visual memory for localization tasks.

Index Terms

Visual attention, object recognition and tracking, active vision, visual localization.

I. Introduction

Cameras have been incorporated in the last years to robots as common sensory equipment. They are very cheap sensors and may provide much information to robots about their environment. However, extracting relevant information from the image flow is not easy. Vision has been used in robots for navigation, object recognition, 3D mapping, visual attention, robot localization, etc.

A. 3D visual memory

Robots usually navigate autonomously in dynamic environments, and so they need to detect and avoid obstacles. There are several sensors which can detect obstacles in robot's path, such as infrared sensors, laser range finders, ultrasound sensors, etc. When using cameras, obstacles can be detected through 3D reconstruction.

Computer vision is one of the most successful sensing modalities used in mobile robotics. It would seem to be the most promising one for the long term. Computer vision research is currently growing rapidly, both in robotics and in many other applications, from surveillance systems for security purposes to the automatic acquisition of 3D models for Virtual Reality displays. The number of commercial applications is also increasing, like traffic monitoring, parking access or face recognition.

And we feel that it is well worth continuing with work on the long-term problems of making robotic vision systems.

Vision is the sensor whose main skill lies in giving information about which and where are the objects that the robot is finding over its path. And, although we must be wary when comparing a robot with a biological organism [Nehmzow, 1993], what is clear is that sight is the main sense used by animals when they want to move around the environment.

Humans have an active vision system ([Bajcsy, 2009]). This means that we are able to concentrate on particular regions of interest in a scene, by movements of the eyes and head or just by shifting attention to different parts of the images we see. What advantages does this offer over the passive situation where visual sensors are fixed and all parts of images are equally inspected? First, some parts of a scene perhaps are not accessible to a single sensor are viewable by a moving device. In humans, movable eyes and head give us almost a full panoramic range of view. Second, by directing attention specifically to small regions which are important at various times we can avoid wasting effort trying always to understand the whole surroundings, and devote as much as possible to the significant part. For example, when attempting to perform a difficult task such as catching something, a human would concentrate solely on the moving object and it would be common experience to become slightly disoriented during the process.

Active vision can be thought as a more task driven approach than passive vision. With a particular goal in mind for a robot system, an active sensor is able to select from the available information only that which is directly relevant to a solution, whereas a passive system processes all of the data to construct a global picture before making decisions; in this sense it can be described as data driven.

The emerging view of human vision as a *bag of tricks* [Ramachandran, 1990]; a collection of highly specialised pieces of *software* running on specialised *hardware* to achieve vision goals, rather than a single general process, seems to fit in with active vision ideas when a similar approach is adopted in artificial vision systems. High-level decisions about which parts of a

All of them are with University of Rey Juan Carlos.

E-mails: julio.vega@urjc.es, eperdices@gsyc.es, jmplaza@gsyc.es

scene to direct sensors towards and focus attention on them can be combined with decisions about which algorithms or even which of several available processing resources to apply to a certain task. The flexibility of active systems allows them to have multiple capabilities of varying types which can be applied in different circumstances.

In this work we describe an overt attention system for a mobile robot endowed with a pan-tilt camera, whose will let it to find paper arrows on its surroundings and navigate through the 3D-space avoiding obstacles. This system performs an early segmentation on color space to select a set of candidate objects. Each object enters a coupled dynamics of life and salience that drives the behavior of the attention system over time. That way, our system will continuously keep relevant objects around the robot -such as arrows or parallelograms- in its visual short-term memory and it will know where they are located.

B. Visual localization

Robots need to know their location inside the environment in order to unfold the proper behavior, according to its sensor measures. Classical localization techniques use input data from laser range scanners or ultrasound sensors. In this chapter, a new technique to perform robot self-localization using only visual input data is presented.

Self-localization is at the moment one of the most important challenges in robotics. Using robot sensors, such as cameras, laser sensors or ultrasonic sensors, our robot must be able to calculate its own localization inside a known environment. Once the robot knows its position, it can adapt its behavior depending on where it is located.

However, robot self-localization has proven to be one the most complex task on mobile robots, since they must face unknown situations, such as being surrounded by other robots or people, or being located inside a dynamic environment.

This task is especially tough when our environment is symmetric, that is, robot sensors values are similar at different environment positions. For instance, if our robot is inside an office building and its main sensor is a camera pointing towards a closed door, the robot only will be able to know that it is in front of a closed door, but there will be several positions (one for each door) where this data may be obtained. Thus, it is unlikely to calculate the proper robot localization only with this information.

Nevertheless, robot localization might be more reliable if we are able to accumulate information over time. To achieve this aim, there are different approaches widely used on research, such as particle filters (also known as Monte Carlo methods (MCL) [Fox *et al.*, 1999]), Kalman filters [Kalman, 1960], or Markov models [Simmons y Koening, 1995].

We have designed a new approach specifically designed to bear symmetries, as we will explain in Section VI.

In this paper we propose a visual perceptive system for an autonomous robot composed of two modules. First, a short term visual memory of robot surroundings fed with images from a mobile camera. The memory stores 3D segments of the environment and objects. A gaze control algorithm has been developed to select where the camera should look at at every time. Second, a visual localization algorithm that uses current image or the current contents of this memory to estimate robot position.

The remainder of this chapter is organized as follows. Second section reviews some of the related works. Third section presents the design of the proposed visual system. The next three sections describe its three building blocks: visual memory component, visual attention component and localization component. The seventh section include some experiments, both with simulated and real robots, performed to validate our system. Finally some conclusions are summarized.

II. Related works

Vision has been used in robotics almost from its beginning. In the last years its use is increasing, mainly because of the reduction in the camera price, the available computing power and the potential of cameras as source of information about robot surroundings. Many issues have been taclked in the intersection of computer vision and robotic fields. For instance, vision-based control or navigation, vision-based map building and 3D representation, vision-based localization, object recognition, attention and gaze control among others. We will review some examples here.

Regarding vision based control and navigation, Remazeilles et al.[Remazeilles *et al.*, 2006] presented the design of a control law for vision-based robot navigation. The particularity of this control law is that it does not require any reconstruction of the environment, and it does not force the robot to converge towards each intermediary position in the path.

Recently, Srinivasan [Srinivasan *et al.*, 2006] presented a new system to increase accuracy in the optical flow estimation for insect-based flying control systems. A special mirror surface is mounted in front of the camera, which is pointing ahead instead of pointing to the ground. The mirror surface decreases the speed of motion and eliminates the distortion caused by the perspective. Theoretically, the image should present a constant and low velocity everywhere, simplifying the optical flow calculation and increasing its accuracy. Consequently, the system increases the speed range and the number of situations under which the aircraft can fly safely.

Badal [Badal *et al.*, 1994] reported a system for extracting range information and performing obstacle detection and avoidance in outdoor environments based on the computation of disparity from the two images of a stereo pair of calibrated cameras. The system assumes that objects protrude high from a flat floor that stands out from the background. Every point above the ground is configured as a potential object and projected onto the ground plane, in a local occupancy grid called Instantaneous Obstacle Map (IOM). The commands to steer the robot are generated according to the position of obstacles in the IOM.

Goldberg [Goldberg *et al.*, 2002] introduced a stereo vision-based navigation algorithm for the rover planetary explorer MER, to explore and map locally hazardous terrains. The algorithm computes epipolar lines between the two stereo frames to check the presence of an object, computes the Laplacian of both images and correlates the filtered images to match pixels from the left image with their corresponding pixels in the right image. The work also includes a description of the navigation module GESTALT, which packages a set of routines able to compute actuation, direction, or steering commands from the sensor information.

In autonomous robots it is important to perform a visual attention control. The cameras of the robots provide a large flow of data you need to select what is interesting and ignore what does not; this is the main goal of visual attention. There are two aspects of visual attention: *overt attention* and *covert attention*. The aim of covert attention [Tsotsos *et al.*, 1995; Itti y Koch, 2001], [Marocco y Floreano, 2002] is to select interesting information within an image. Overt attention selects from the environment surrounding the robot, beyond the field of view, those objects of interest, and it looks at them [Cañas *et al.*, 2008].

The visual representation of the interesting objects around the robot can improve the quality of the robot's behavior and the ability to handle more information when making their decisions. This poses a problem when those objects are not in the immediate field of vision. To solve this problem, some studies used omnidirectional vision, in others using a regular camera and a mechanism for overt attention [Itti y Koch, 2001; Zaharescu *et al.*, 2005], which enables fast-to-take samples of a very broad area of interest. The use of a camera in motion to facilitate object recognition was proposed by [Ballard, 1991], and has been used, for example, to distinguish between different forms in the images [Marocco y Floreano, 2002].

One of the concepts widely accepted in the work area is the *salience map*. It is found in [Itti y Koch, 2001], as a covert visual attention mechanism, independent of the particular task to be performed and composed by all visual stimuli that attract attention from the scene. In such work is considered purely a form of "bottom up", where, as we see in Figure 1 in each iteration the different scene-descriptive maps (as colors, intensities or directions) compete between each other. Then, they merged into conspicuity maps (one for each feature) and eventually will form a unique and representative salience map.

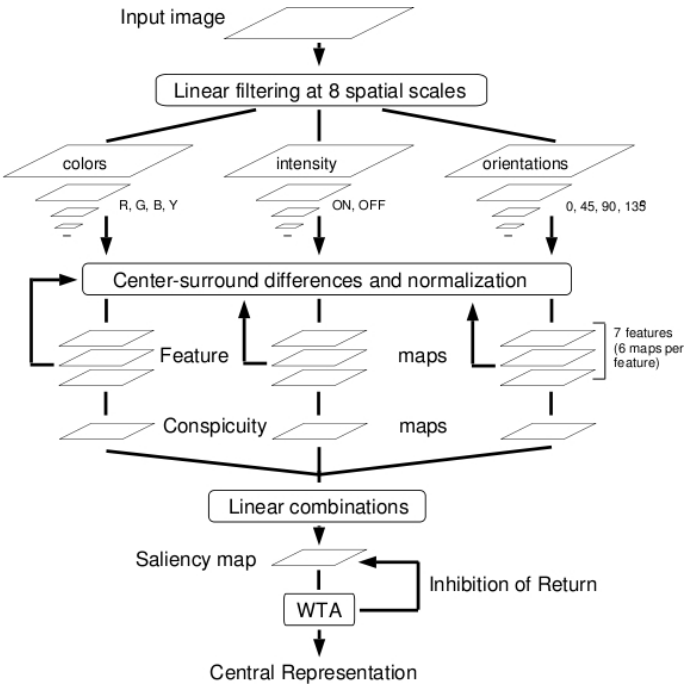

Fig. 1. Salience map as proposed by Itti

Hulse [Hulse *et al.*, 2009] presented an active robotic vision system based on the biological phenomenon of inhibition of return, used to modulate the action selection process for saccadic camera movements. They argued that visual information has to be subsequently processed by a number of cortical and sub-cortical structures that place it: 1) in context of attentional bias within egocentric salience maps; 2) the aforementioned IOR inputs from other modalities; 3) overriding voluntary saccades and 4) basal ganglia action selection. Thus, biologically there is a highly developed, context specific method for facilitating the most appropriate saccade as a form of attention selection.

Arbel and Ferrie presented in [Arbel y Ferrie, 2001] a gaze-planning strategy that moves the camera to another viewpoint around an object in order to recognize it. Recognition itself is based on the optical flow signatures that result from the camera motion. The new measurements, accumulated over time, are used in a one-step-ahead Bayesian approach that resolves the object recognition ambiguity, while it navigates with an entropy map.

Grid based probabilistic localization algorithms were successfully applied with laser or sonar data in small known environments [Burgard y Fox, 1997]. They use discretized probability distributions and update them from sensor data and movement orders, accumulating information over time and providing a robust position estimation. Particle filters use sampled probability functions [**?**] and extend the techniques to larger environments. At the beginning the maps were provided in advance but in the last years the SLAM techniques tackle localization simultaneously with the map building. There are many particle filter based SLAM techniques. In addition, one of the most successful approaches is Mono-SLAM from Andrew Davison [**?**; **?**] based on a fast Extended Kalman Filter for continuous estimation of 3D points and camera position from relevant points in the image.

In ([Mariottini y Roumeliotis, 2011]) Mariottini and Roumeliotis presented a strategy for active vision-based localization and navigation of a mobile robot with a visual memory where previously visited areas are represented as a large collection of images. It disambiguates the location taking into account the sequence of distinctive images, while concurrently navigating towards the target image.

Gartshore [Gartshore *et al.*, 2002] developed a map building framework and a feature position detector algorithm that processes images on-line from a single camera. The system does not use matching approaches. Instead, it computes probabilities of finding objects at every location. The algorithm starts detecting the objects boundaries for the current frame using the Harris edge and corner detectors. Detected features are back projected from the 2D image plane considering all the potential locations at any depth. The positioning module of the system computes the position of the robot using odometry data combined with image feature extraction. Color or gradient from edges and features from past images help to increase the confidence of the object presence in a certain location. Experimental results tested in indoor environments set the size of the grid cells to 25 mm 25 mm. The robot moved 100 mm between consecutive images.

In [Jensfelt y Kristensen, 2001], Jensfelt et al. presented an active global localization strategy that uses Kalman filtering (KF) to track multiple robot pose hypotheses. Their approach can be used even with incomplete maps and the computational complexity is independent on the size of the environment.

III. Design

The proposed perceptive system is designed for autonomous robots that use a mobile camera, like that on the head of humanoids or in robots with pan-tilt units. The block diagram of the robot control architecture is showed in Figure 2. It has been divided into two main components: `active_visual_memory` and `localization`. They will be described in detail in the following sections.

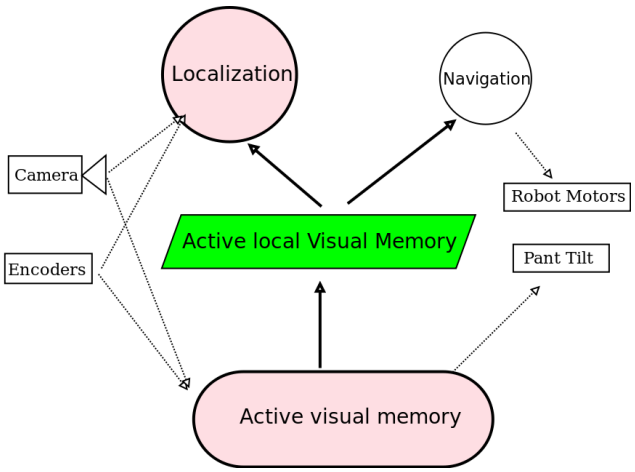

Fig. 2. Block diagram of the proposed visual system

First, the `active_visual_memory` component builds a local active visual memory of objects in the robot's surroundings. The memory is built analyzing each camera image looking for relevant objects (like segments, parallelograms, arrows, etc) and updating the object features already stored in the memory like their 3D position. The memory is dynamic and is continuously coupled with camera images. The new frames confirm or correct the object features stored in memory, like their 3D relative position to the robot, length, etc.. New objects are introduced in memory when they appear in images and do not match any known object.

This memory has a broader scope than the camera field of view and objects in memory have more persistence than the current image. Regular cameras typically have 60 degrees of scope. This would be good enough for visual control but a broader scope may improve robot responses in tasks like navigation, where the presence of obstacles in the robot's surroundings should be taken into account even if they lie out of current field of view.

This memory is intended as local and short-term. Relative object positions are estimated using robot's odometry. Being only short term and continuously correcting with new image data there is no much time to accumulate error in the object relative position estimation. Currently the system deals only with objects on the floor plane (ground hypothesis) and uses a single camera. It can be extended to any 3D object position and two cameras.

In order to keep this short term visual memory consistent with reality, the system has mechanisms to properly refresh it. The camera is assumed to be mobile, typically mounted over a pan-tilt unit. Its orientation may be controlled and changed during robot behavior at will, and so, the camera may look towards different locations even if the robot remains static. In order to feed the visual memory, an overt attention algorithm has been designed to continuously guide camera movements, choosing where to look at at every time. It has been inserted inside the `active_visual_memory` component and associates two dynamic values to each object in memory: *salience* and *life* (quality). Objects with low life are discarded and objects with high salience are good candidates to look at.

The position of objects already in memory are themselves foci of attention in order to refresh their perceived features. Random locations are also considered to let the robot explore new areas of its surroundings. In addition, new foci of attention may also be introduced to check the presence of some hypothesized objects. For instance, once the robot has seen three vertices of a parallelogram, the position of the fourth one is computed from the visual memory and ordered as a tentative focus of attention for the camera.

Second, a vision based localization algorithm has been developed in the `localization` component. It uses a population of particles and an evolutionary algorithm to manage them and find the robot position. The health of each particle is computed based on the current image or based on the current contents of the visual memory. The local visual memory provides information about robot's surroundings, typically more than the current instantaneous sensor readings. In this way, the visual memory may be used as a virtual sensor and its information may be used as observations for the localization algorithm. Because of its broader scope it may help to improve localization, especially in environments with symmetries and places that look like similar according to sensor readings.

The robot control architecture brings together the different capabilities described above in a consistent framework. This architecture is showed in in Fig. 3. As vision is the only sensor utilized, it all starts with a camera stream which serves as input data for a memory structure.

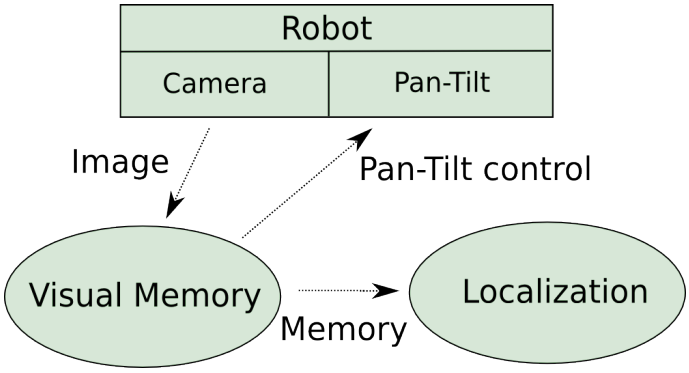

Fig. 3. Architecture designed

IV. Visual memory

The goal of our visual memory is to do a visual tracking of the various basic objects in the scene surrounding the robot. It must detect new objects, track them updating its relative position to the robot and remove them from the memory once they have disappeared.

The first stage of the system is the 2D analysis, which detects 2D segments present in the current image. Then the 3D reconstruction algorithm places these objects in 3D space according to the *ground-hypothesis* (that is, we suppose that all objects are flat on the floor). And finally, the 3D memory system stores their position in 3D space, calculates perceptual hypotheses and generates predictions of these objects in the current image perceived by the robot.

The visual memory component has several building blocks (see Fig. 4). First, an object detector is responsible of identifying basic shapes in the current image. Second, the prediction mechanism allow the system to predict how the stored items will appear in next images, reducing the computational cost of image processing. Third, a 3D reconstruction block is responsible to obtain 3D instantaneous information from objects in the current image and merging them with the objects already stored in the visual memory.

The algorithm also creates perceptual hypothesis with the items stored, allowing the system to abstract complex objects. And all borne on visual memory, which forms the core of the whole attention system. This allows to extend the field of vision to the whole scene surrounding the robot, instead of just the current visual field.

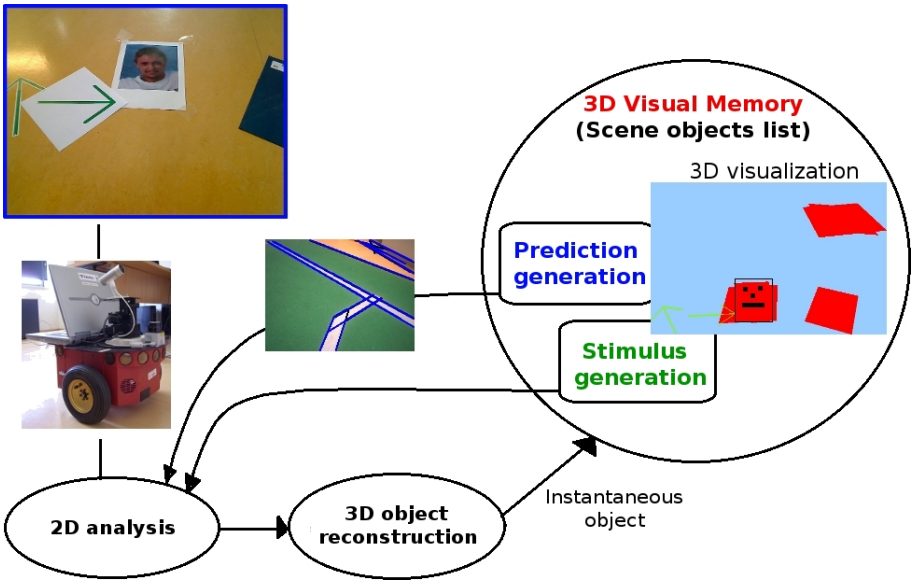

Fig. 4. Visual Memory scheme

To move and track features simultaneously, our system executes a loop at a frequency of 5*Hz*. In each iteration a motion prediction is made, the head is moved, images are acquired and displayed, image processing occurs and all the estimated quantities are updated.

However, we quickly found out that the main factor limiting speed of operation was the processing time needed to perform image processing on the received images. Robot odometry is used to provide the trigger for the loop execution: a new image is acquired when the robot has reached a certain position according to readings from its encoders. In this sense, the update frequency is linked to distances travelled by the robot, and so 5Hz is only an approximate value calculated from the robot calibration.

A. 2D Image Processing

The main goal of this part of the system is to extract 2D straight segments as a basic primitive, based on [Solis *et al.*, 2009]. In recent years we realized that using Canny algorithm and Hough transform to extract straight segments was not an accurate method. Usually, the outcome was not fully effective: line segments were often disconnected.

Now, following this new design, edges extraction from color images is done using much simpler Laplace operator instead of sophisticated Canny edge detector from OpenCV. This algorithm uses a compilation of different image processing steps such as normalization, Gaussian smooth, thresholding, and Laplace edge detection to extract edge contours from input images.

Just as Solis et al. told us, this solution gives us a surprisingly more accurate, faster and simpler answer with fewer parameters than the widely used Hough Transform algorithm for detecting lines segments among any orientation and location inside images. To implement these techniques, we use the OpenCV library.

Figure (5) shows the differences between Canny+Hough and Solis algorithms. While the first one is able to recognize just a really small set of unappreciated segments (blue lines), the second one gets all of them. Furthermore, the floor used here is a textured surface, so the border detection is quite difficult to do; because of that, we see some false positives.

B. Predictions

The 2D analysis system is connected directly to the 3D visual memory to alleviate the computational cost due to image analysis. So before extracting features of the current image, the system makes the prediction of those objects stored in the 3D memory which should be visible from the current position.

We have used our library called Progeo, which provides *projective geometry* capabilities given a calibrated camera. So each 3D visible object is stored and made its projection on the image plane. The system refutes/corroborates such segments predicted, comparing one of these segments with those obtained by the Hough Transform. This comparison leads to three sets of segments, as seen in Figure 6.

C. Reconstruction with 3D segments and parallelograms

The above mechanism extracts a set of 2D segments which must be located in 3D space. To do this, and as we have already mentioned, we rely on the idea of *ground-hypothesis*. Since we have one camera, we need a restriction which will enable to estimate the third dimension. We assume that all objects are flat on the floor.

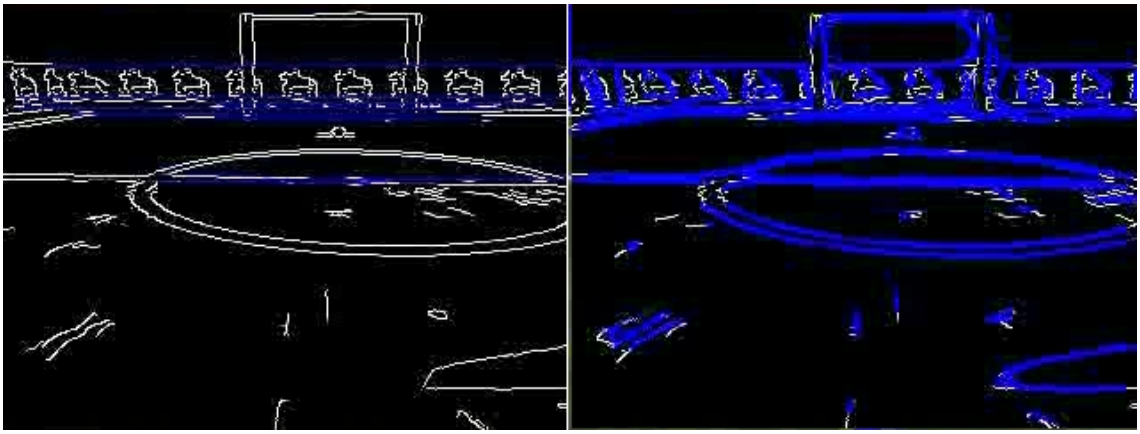

Fig. 5. Differences between Canny+Hough and Solis algorithms

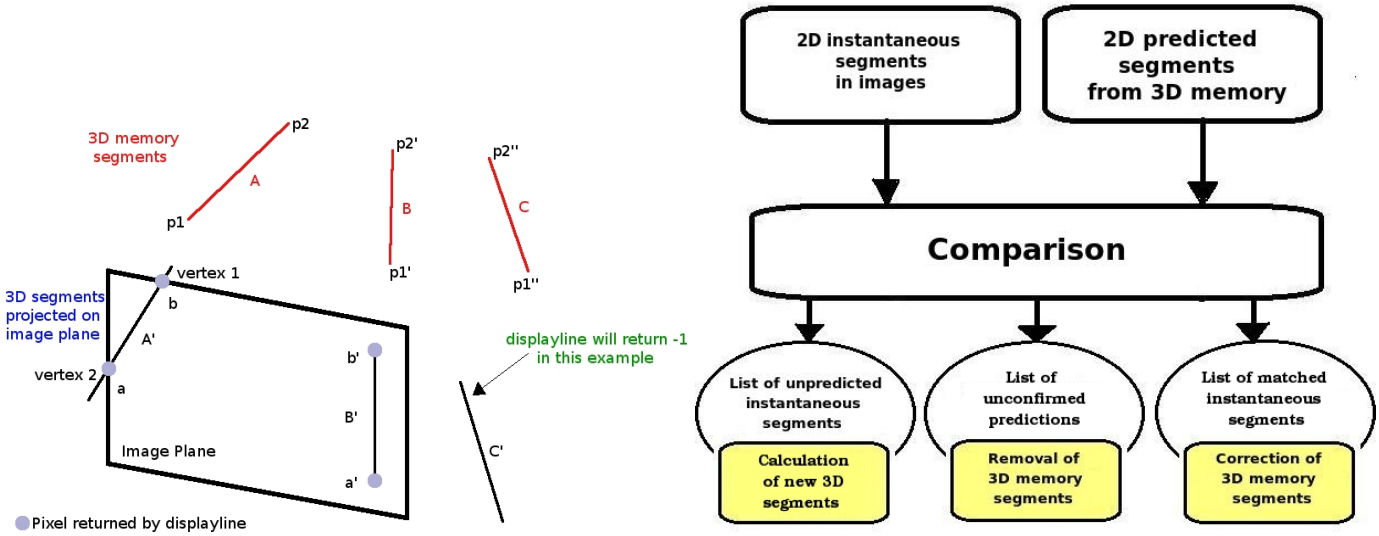

Fig. 6. 3D projection on the image plane and matching between predicted and instant segments

Once we have the 3D objects, and before including them on the 3D memory, postprocessing is needed to avoid duplicates in memory due to noise in the images. This postprocessing compares the relative position between segments, as well as its orientation and proximity. The output is a set of 3D segments situated on the robot coordinate system. Figure 7 shows the 3D scene with the parallelograms reconstructed by the system, the segments detected in the current image and the segments predicted from the current position.

We use four coordinate systems to define the geometric model, as we can see in Figure 8:

- The absolute coordinate system whose origin lies somewhere in the world where the robot is moving.
- The system located at the base of the robot. The robot odometry gives its position and orientation with respect to the previous system.
- The system relative to the base of the pan-tilt unit to which the camera is attached to. It has its own encoders for its position at any given time, with pan and tilt movements with respect to the base of the robot.
- And finally we have the coordinate system of the camera itself, displaced and oriented in a particular mechanical axis from the pan-tilt unit.

D. Inserting segments into the 3D visual memory

3D memory comprises a dynamic set of lists which stores information about the different types of elements present in the scene (position, type or color). The most basic form of structure is the segment. Thanks to the memory we can establish relationships between them to make up more complex elements such as arrows, parallelograms, triangles, circles or other objects.

To store a segment we have a structure called *Segment3D*, consisting in a start and end point and a pointer to other possible structures of which it can be part of: Arrow3D or Parallelogram3D.

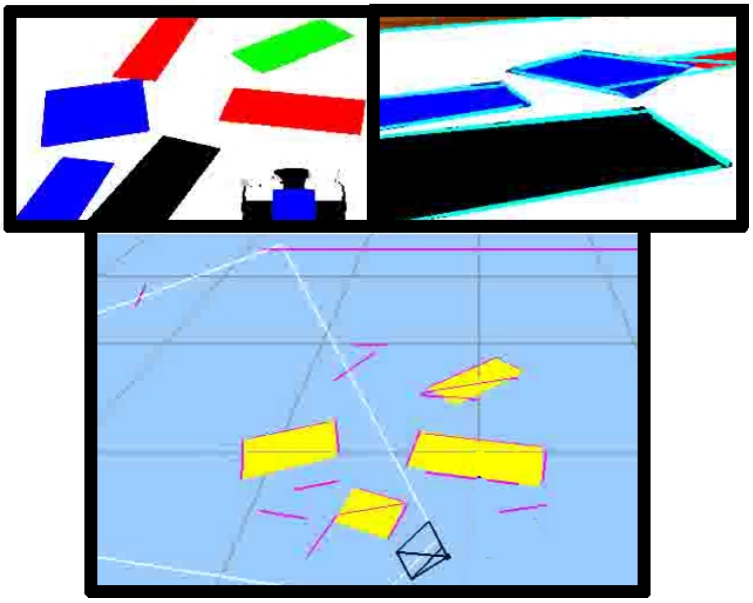

Fig. 7. Scene situation, instantaneous image and 3D scene reconstruction

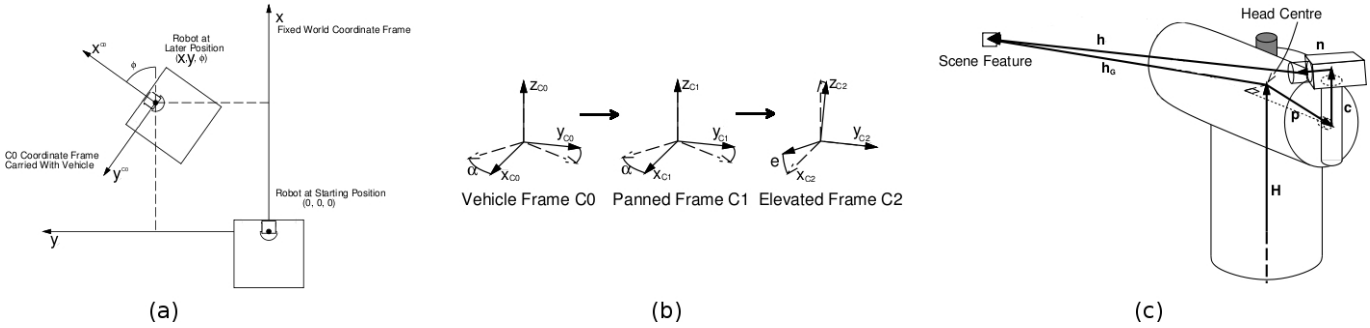

Fig. 8. Coordinate systems used to define the geometric model

Incorporating 3D memory segment basically consists of comparing each segment individually calculated in the snapshot with those already stored. In case of nearby segments with similar orientation, the system combines these segments into a new one taking the longest length of its predecessors, and the orientation of the more recent, as probably it is more consistent with reality (the older ones tend to have more noise due to errors robot odometry).

To make this fusion process computationally lighter, the system has a segment cache with only the segments close to the robot (in a radius of around 4 m.). Its implementation is basically a dynamic list of pointers to these segments. The system always works with subsets of features, which are pointers to the overall 3D memory elements.

E. Predictions: deletion and correction of segments

As mentioned before, the 2D analysis system returns different subsets of segments, as the result of comparison between instant and predicted segments from 3D memory.

If a segment is identified in the current image and it does not match the predictions, the system creates a new one in 3D which might replace an existing one (replacement or correction) under certain restrictions. To reflect this process, system has a parameter called *uncertainty* which will increase as the time segment remains in memory.

The element deletion process is based on the same principle, but here there are more rigorous restrictions. So, the replacement process is a priority compared to deletion.

F. Perceptual hypothesis generation

Our object model consists of a set of segments whose vertices can belong to more abstract structures like parallelograms. The vertices are labeled with the number of segments that are tied to them. This requires an object model for cases in which

certain vertices are not visible at any given time. For instance, for parallelograms the minimum number of visible vertices is small, with three points we are able to estimate the fourth one.

The segments and their corresponding vertices are used to detect parallelograms checking the connection between them and the parallelism conditions. The analysis of the angles formed by each segment provides information about how the segments are connected to each other. In addition, this feature can be used to merge incomplete or intermittent segments. Similarly, we can extract the position of a possible fourth vertex using the information about other edges and/or possible parallelogram vectors. This capability makes our algorithm robust against occlusions, which occur frequently in the real world.

Figure 9-b, c illustrates an example of occlusion that is satisfactorily solved by our algorithm. The results of reconstruction of parallelograms can be seen in Figure 9-a. In this situation, we have a collection of parallelograms spread on the floor. The robot, after several snapshots, captures what is around itself: those parallelograms, and noise extracted by the segmentation algorithm. However, our parallelogram hyphotesis generation is able to extract only the real parallelograms, avoiding such noise.

Similarly, we can abstract other abstract objects such as arrows.

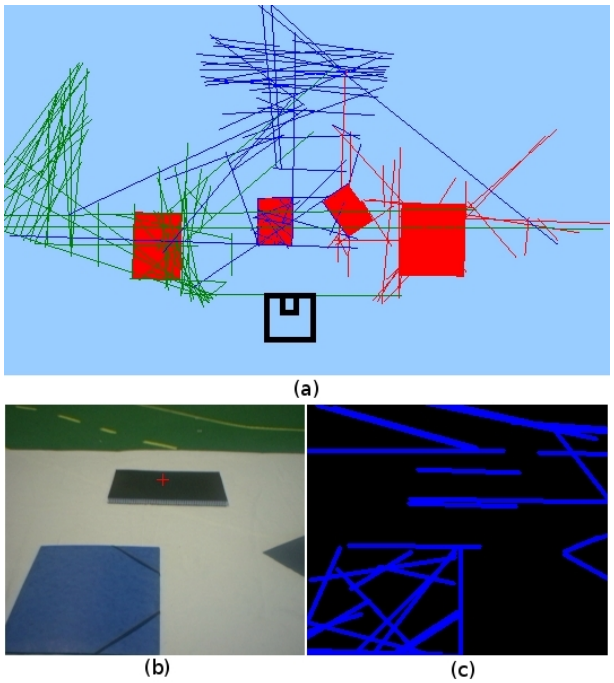

Fig. 9. Generation of hypothesis: parallelogram with occlusion

V. Visual attention system

In the previous section we have described in detail the operation of placing objects on the robot 3D visual memory. Now we will describe the visual attention mechanism implemented based on two of object attributes: *salience* and *life*. The salience is used for deciding where to look at in every moment, while life is the mechanism for forgetting an object, deleting it from memory, when it has disappeared from the scene.

In addition, we have designed a mechanism to control the camera movements, to track objects, and another mechanism to explore new unknown areas from the scene.

A. Gaze control: salience dynamics

Some sort of decision-making mechanism to indicate to the system where to look at in the next instant. Each object in the visual memory has its coordinates in the scene representation. It is desirable to control the movement of the pan-tilt unit to direct the focus to that position periodically in order to reobserve that object. To control the movement of the pan-tilt unit, we introduced the dynamics of salience and attention points. They mainly represent the detected objects in the scene. Each one contains a position in the 3D scene (X, Y, Z), which is translated into mechanical commands to the pan-tilt unit in order to direct the focus at that point.

Anything that attracts attention or stands in a given situation is salient. The focus may be changing over time to look at all the salient points. In this system salience indicates how attention selects the next object to be visited. Each memory element

has an associated salience, which grows over time and vanishes every time that element is visited. The focal point with highest salience will be the next to be visited. If the salience is low, it will not be visited now.

$$Salience(t) = \begin{cases} Salience(t-1) = 0 & \text{if object attended} \\ Salience(t-1) + 1 & \text{otherwise} \end{cases}$$

When a point is visited, its salience is set to 0. A point that the system has not visited recently calls more attention than one which has just been attended. The system is thus similar to the behavior of a human eye, as pointed by biology studies [Itti y Koch., 2005]: when the eye responds to a stimulus that appears in a position that has been previously treated, the reaction time is usually higher than when the stimulus appears in a new position.

The designed algorithm allows the system to alternate the focus of the camera between the different objects in the scene according to their salience. In our system, we consider that all objects have the same preference of attention, so all of them are observed during the same time and with the same frequency. If we assigned different priorities to the objects, we could establish different rates of growth of salience. This would cause the pan-tilt unit to pose more times in the object whose salience grows faster.

The problem of how to revisit a point is solved considering the spatial interpretation, because we consider it is a problem of evolution of the hypothesis in the time among detections in *t* and *t* + *n* (where n is the time a point is not visited).

We assume that a detected object will be found near the location where it was previously.

B. Tracking of a focused object

When the look-sharing system chooses the attention point of a given object, it is going to be looking for a certain time (3 seconds), tracking it if it moves spatially. For this tracking, and to avoid excessive oscillations, we use a *P-controller* (Fig. 10) to control the speed of the pan and tilt and thus continually focus that object on the image center. This driver orders high speeds to the pan-tilt unit if the focus of attention is far from the predicted position; or lower speeds if it requires small corrections. The controller follows next equations.

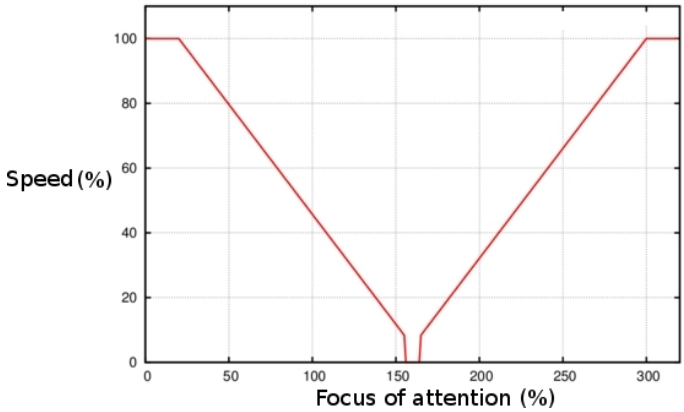

Fig. 10. P-controller mechanism

$$v(Pan) = \begin{cases} 0 & \text{if} \quad \epsilon < 0.3 \\ K_p \cdot (P_t - P_a) & \text{if} \quad 0.3 \leq \epsilon < M_p \\ M_p & \text{if} \quad M_p < \epsilon \end{cases}$$

$$v(Tilt) = \begin{cases} 0 & \text{if} \quad \epsilon < 0.1 \\ K_p \cdot (T_t - T_a) & \text{if} \quad 0.1 \leq \epsilon < M\_t \\ M_t & \text{if} \quad M_t < \epsilon \end{cases}$$

Where: K_p is the P control gain, T_t is the Tilt of the target, T_a is the actual Tilt, P_t is the Pan of the target, P_a is the actual Pan, M_t is the maximum Tilt and M_p is the maximum Pan.

C. Exploring new areas of interest

At any time, it may be interesting to look for new objects in the scene. For that search our system will insert periodically (every *forcedSearchTime*) scanning points with high salience in memory. This search is especially interesting at the beginning, when there are many unknowns areas of the scene where there can be objects of interest.

The scanning points can be of two types: random and systematic ones. The first type are assigned uniformly random coordinates *(pan, tilt)* within the pan-tilt range (pan = [-159, + 159], tilt = [-31, +31]). That way, the system ensures that all areas of the scene will be visited. Thus, these points will range from the lowest position of pan-tilt to the highest position, and also with the tilt coordinate. Systematic scanning points follow a regular pattern to finally cover the whole scene around the robot.

The attention points, whatever their type, have a high initial salience in order to be quickly visited with the camera and thereby check whether any object of interest is found around it. In such a case that object will enter into the memory and into the gaze sharing module.

There will be a proliferation of points of exploration in the beginning, because the most interesting process in that moment is to find areas of interest in the scene as we start from the absolute ignorance of the environment.

As we discover objects, the desire to explore new areas will decrease in proportion to the number of already detected objects.

D. Representation of the environment: life dynamics

As already discussed in previous sections, our visual attention system guides the search and tracking of objects within the scene. The objects may eventually disappear from the scene, and then they should be removed from the memory in order to maintain coherence between the representation of the scene and the reality.

To forget such old elements, we have implemented the *life* dynamics. With this mechanism the system can know whether an object has left the scene or it is still there. The life operation is the reverse of the salience, that is, a frequently visited object will have more life than one less visited. When the life of an object is below a certain threshold, it will be discarded.

Every time the attention system visits an object, its life increases one point, with a maximum limit to provide saturation. The life of unobserved objects will decrease over time. Thus, when the life of an object exceeds a certain threshold, which is still on the scene, whereas when is below it is gone.

$$Life(t) = \begin{cases} min(MAX_{LIFE}, Life(t-1) + \Delta) \\ \qquad \text{if object observed} \\ Life(t-1) - 1 \\ \qquad \text{otherwise} \end{cases}$$

E. Attentive system operation

The objects of the environment surrounding the robot guide the movements of the camera. So the mechanism of attention is so far *bottom-up*. Besides, the underlying mechanism of *top-down* attention is that existing relevant objects are only those that have a certain appearance given by the task at hand: in our examples, parallelograms or arrows. This tendency to look at objects with a certain aspect is similar to the bias detected by ethologists in animals with respect to certain stimuli [**?**].

The visual attention system presented here has been implemented following a *state-machine* design, which determines when to execute the different steps of the algorithm. Thus, we can distinguish four states:

- Discuss next goal (state 0).
- Saccade is completed (state 1).
- Analyze image (state 2).
- Tracking object (state 3).

Periodically the system updates the salience and life attributes of the objects that have already stored in memory following previous equations. It checks whether any of them is already outdated, because its life is below a certain threshold. If not, it increases its salience and reduces its life.

Based on the initial state (or state 0), the system asks whether there is any attention point to look at (in case we have an object previously stored in memory) or not. If so, it goes to state 1. If not, it inserts a new scanning attention point into memory and goes back to state 0.

In state 1 the task is to complete the movement towards the absolute position specified in state 0. Once there, we go to stage 2 where we will analyze whether there are relevant objects or not. In any case, it passed the state 0 and back again.

VI. VISUAL LOCALIZATION

As we explained in Section I-B, we have designed a new approach to solve robot self-localization specifically designed to bear symmetries. That is, an evolutionary algorithm, a type of metaheuristic optimization algorithm that is inspired by the biological evolution to solve a problem.

In this kind of algorithms, candidate solutions are so-called ”individuals”, which belong to a population that evolve over time using genetic operators, such as mutation or crossover, as we will explain later.

Besides, each individual is evaluated with a quality function which calculates its ”health”, that is, a measure to know how good is the current localization with regard to the optimal solution. The algorithm is independent of this health function, so that it works either with instantaneous measures of robot sensors or with the visual memory described in previous sections.

In our approach, the evolutionary algorithm has 4 main members:

- Individuals: An individual represents a concrete candidate solution. It stores a robot pose (X, Y, θ) (Fig. 11) and a probability obtained with the quality function.
- Races: Each race is a set of individuals that represents a possible solution to robot's localization. The algorithm has N races which compete among each other to be the race selected in each iteration, i.e. the solution to the localization problem. The parameters stored in each race are the robot pose (X, Y, θ), its "health", number of times selected and its "life" (iterations left without being able to be deleted).
- Explorers: Independent individuals who try to find new good positions where creating new races.
- Exploiters: Each individual who belongs to a race. They analyze deeply the neighbor positions of the current race pose to get the best solution close to it.

The main idea of the algorithm consists of keeping several races competing among each other in several positions. In case of symmetries from observations, we will create new races on various positions where the robot can be located. After some iterations, predictably, new observations will provide information to reject the most of race poses and we will obtain the real robot pose.

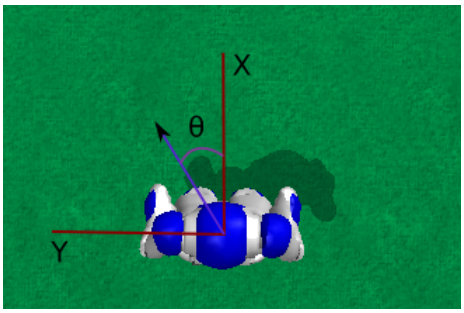

Fig. 11. 2D robot pose

In each iteration of the algorithm there are several steps to be performed to know the current robot pose (Fig. 12).

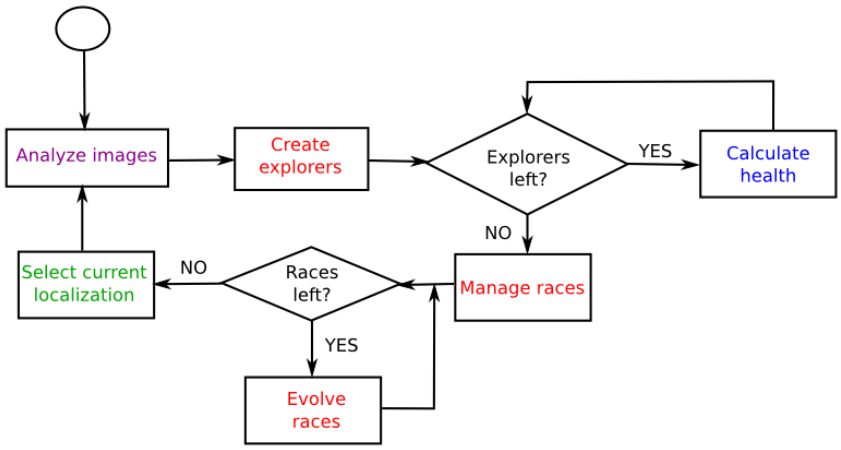

Fig. 12. Basic diagram of evolutionary algorithm

Next we are going to describe the most important steps in each iteration:

- Analyze image: We need to take information from robot's camera, either analyzing each image or using the visual memory. First we are going to explain how our algorithm would work analyzing images and we will explain the differences if we use the visual memory in Section VI-A.
- Health calculation: We use the information obtained after analyzing images and calculate the current race health (including its exploiters), as we will explain in Section VI-B.
- Explorers creation: We spread randomly new explorers with the aim to find new positions where new races could be created. We will detail this process in Section VI-D.
- Races management: We create, merge or delete races depending on their current state and the new explorers health, as we will see in Section VI-E.

- Races evolution: If we have just created a new race, we create its exploiters, or if they already exist, we evolve them by using genetic operators. Besides, if the robot is moving, we update all races and exploiters taking into account this movement. We will explain all these operator in Section VI-F.
- Selecting current robot pose: After calculating each race health, we will select one of them to set the current robot pose with requirements we will detail in Section VI-G.

A. Analyzing images

To validate our approach independently of the visual memory, we have tested the algorithm with instantaneous images of robot camera. Thus, we analyze the the current image to obtain characteristic lines from environment.

The fist step consists of using the line detection algorithm we explained in Section IV-A, this algorithm gives us a set of segments which must be processed. With this segments we created the visual memory explained in Section IV, where we rejected segments over the horizon and we merged them in our 3D visual memory.

However, we proceed now in other way, we don't need to reject segments that don't lie on the floor, since we don't need to backproject segments in 3D, and we only need to merge segments in 2D, to join consecutive segments together (Figure 13).

Before merging, we need to label each segment with a type depending on its colors, so, segments with unknown type are rejected. After labeling each segment, we try to merge segments with the same type if their extremes are very close to each other.

Fig. 13. Image analysis before merging (left) and after merging (right)

Finally, we rule out segments too small and save the rest of them. We could use these lines directly as input data, but we have decided to divide this lines into points to make the comparison between lines easier, as we will explain in Section VI-B.

So, we have created a grid with dynamic size and we only save a new point where the lines detected intersect with this grid (Figure 14). The size of this grid changes because we want to analyze the top of the image more deeply than the bottom, since further objects will be at the top of the image and its resolution will be smaller.

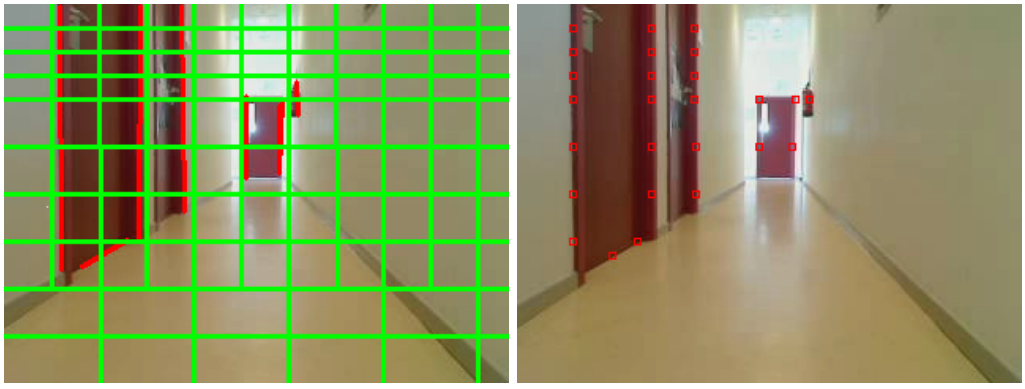

Fig. 14. Image grid (left) and points selected (right)

All these points selected will be the input data we will use to calculate the health of each Individual.

B. Health calculation with instantaneous images

The health of an individual placed at certain location is computed comparing the theoretical set of visible objects from that location (theoretical observation) with objects currently observed (real observation). The more similar the predicted objects and the observed object are, the more likely such location is the correct one.

Thus, we cover all points detected as we explained in Section VI-A and calculate for each point the Euclidean distance d_i in pixels to the closest theoretical line with the same type than the point. These theoretical lines are generated ad-hoc for each concrete position, projecting lines from a predesigned environment (an accurate replica of the real environment) into the current image.

After calculating d_i for all points, we will be able to calculate the Individual's health by the average distance, as we show in the next equation:

$$H = 1 - \frac{\sum_{i=0}^{N} \frac{d_i}{N}}{M} \tag{1}$$

where N is the number of points and M is set to 50 pixels for a 320x240 image size.

We will show in Section VII-B1 several experiments to analyze the health function behavior in different situations.

C. Health calculation with Visual Memory

In case we use the visual memory instead of instantaneous images, as we explained in previous Section, we don't need to analyze each image, but we obtain the visual memory in real time from the visual memory component.

The kind of health function we use to evaluate each Individual is independent to the rest of the algorithm, so we don't need to change the algorithm behavior.

The memory obtained is a set of 3D points relatives to our robot, so we can't compare lines in image as we did before. Besides, we have to take into account that lines may not be detected completely or they may be divided into several small lines.

Thus, to calculate the current health of an Individual we cover all lines belonging to the visual memory, for each line we get its extremes and calculate the Euclidean distances d_{j_s} and d_{j_e} in meters to the closest theoretical line with the same type, that is, similar to health function with instantaneous images but in 3D.

After calculating d_{j_s} and d_{j_e} for each line, we use the next equation to calculate the Individual's health:

$$H = 1 - \frac{\sum_{i=0}^{N} \frac{(\frac{d_{j_s} + d_{j_e}}{2})}{N}}{M} \tag{2}$$

where N is the number of lines and M is set to 0.5 meters.

D. Explorers creation

Explorers are individuals who don't belong to any race and who try to find positions where health obtained is better than current races health.

There are two ways to spread explorers around the environment, randomly or predesigned. If our environment is odd enough, we can deduce certain positions where explorers might obtain high health if current visual memory give us enough environment information.

However, to proceed this way, we would need to adapt our algorithm to each concrete environment. So we have decided to exclude this kind of features, and we just spread explorers around the environment randomly.

When an iteration of explorers creation is performed, we create N explorers, calculate their health and arrange them according to its health. After that, we pick up the M best explorers evaluated and promote them to became a candidate to create a new race.

Number of explorers created N changes dynamically depending on current algorithm status. Thus, if our robot is lost, we increase N, and decrease it if its location is reliable.

Besides, since explorers creation takes a great amount of time, when current location is reliable, we don't execute this procedure during all the iterations, but once every N iterations.

E. Races management

Since we have several races, we have to develop a mechanism to manage them, in other words, we need to know when to create a new race, delete it, or merge two races. Before explaining these mechanisms, we need to introduce two existing parameters in each race, "Victories" and "Life":

- Victories: A race has a "Victory" in an iteration when its health is higher than in the rest of races. At first, races have just been created, so "Victories" is set to 0. This number can increase or decrease depending on how a victory is got in

each iteration, as we will explain in Section VI-G. This parameter is useful to select the race that set the current robot pose in each iteration.

- Life: This parameter has been created with two objectives. The first one is preserving new races against new candidates, to avoid creating races that will be deleted in the next iteration. For that, when a race is created, we set a minimum life of 3 iteration, period during which this race can't be deleted or replaced (although it may be merged). The second objective is avoiding deleting a race because of wrong information when predictable it was a good race. To achieve this, we set a minimum life of 9 iterations when a race has the highest health in an iteration.

  This way, we try to provide stability to races and avoid creating and deleting races continually.

1) Creating races: Whenever the explorers creation has been executed and we have new candidates to became a race, we have to decide when to create new races and when to replace an existing one. Our approach has a maximum number or races N that avoids the exponential increasing of computation time related to the number of races.

The first step consists of checking if there are enough races already created. To that end, we need to check if we have already reached the maximum number of races N. In case N is reached, we need to check if any race may be replaced, that is to say, we can't replace a race if either its life or its number of wins is greater than 0. If any race can't be replaced, candidates are ignored.

If we can create or replace new races, the next step is checking the innovation of the candidates, that is, for each candidate we find out if an existing race is located in the same position (X,Y, theta). If this is the case, we don't try to create a new race, but we assume that the existing race already represents this candidate, and we increase the race's life.

Finally, if candidates are innovative, we create new races until we reach N races or replace a race if the health of the candidate is greater than the health of the race replaced.

2) Merging races: In case two races evolve towards the same localization, that is, position and orientation, we consider that they lead us to the same solution, so we merge them. This merging consist of deleting the race with lower victories, or if both have the same victories, we keep the best race according to its health.

3) Deleting races: In case a race may be deleted, that is, both its victories and its life are 0, we need to check if its current health is lower than a threshold, what would mean that the race is not in the real robot pose anymore. We have set this threshold to 0.6, so if its health is lower than 0.6, we consider this race location wrong and delete it.

F. Races evolution

When a race is created from an explorer, all its exploiters are created applying a random thermal noise to the explorer who created the race. From then on, in the next iterations, its exploiters are evolved through three genetic operators:

- Elitism: We select the N best exploiters of each race, arranging them according to their health. Elitist exploiters are saved in the next iteration without any change.
- Crossover: We select two exploiters of the race randomly and calculate their average with their values (X, Y, θ).
- Mutation: We select an exploiter randomly and apply a thermal noise to its position and orientation.

Once we have evolved all the exploiters of each race, we calculate the final pose of the race as the average of its elitist, to avoid abrupt changes in its localization.

Besides, in case the robot has moved since the last iteration, we apply a motion operator to all races and exploiters at the beginning of each iteration using robot odometry.

G. Selecting current robot pose

After evaluating all the existing races, we need to choose one of them in each iteration to be the current pose of the robot. The race selected will be the one with more victories, so we need to increase or decrease this parameter for each race in each iteration.

The first step is selecting the race with greatest health in the current iteration (R_i). If the race selected was the same in the previous iteration (R_p), we will increase the victories of R_i and will decrease the victories of the rest. However, if R_i and R_p are different, we only change the races victories if the difference between R_i health and R_p health is greater enough.

This distinction is made because we want race changing to be difficult if R_p has been the selected race during a lot of iterations. With this behavior, we try to help the races who have been selected in the previous iterations and we only change the winner race when the health difference is big enough (what would mean that the current localization is wrong).

At the end of the iteration, the selected race will be the one with more victories, and its pose will determine the calculated robot pose by the algorithm.

VII. EXPERIMENTS

To verify our different approaches of visual memory, visual attention and visual localization, we conducted several experiments. Our experimental real platforms were an ActivMedia Pioneer 2DX robot equipped with a Logitech Autofocus camera (2 megapixels) and a Nao Robot from Aldebaran Robotics (v. 3). Besides, we have used Gazebo 0.9 as robot simulator. All our experiments are implemented on C++ under JdeRobot robotics software platform, which uses ICE as a middleware.

A. Attentive visual memory

1) Robot in the middle of a room: For the first experiment, the robot is in the middle of a room (see 15-a). Then the robot turns around on itself. Figure 15-b shows a instantaneous view from the room, where robot is able to detect a simple line on the floor.

After a few seconds, the robot has turned full circle, having stored all the information about their surrounding environment. Thus, we can see in figure 15-c, how the short-term memory provides more information than an instantaneous image.

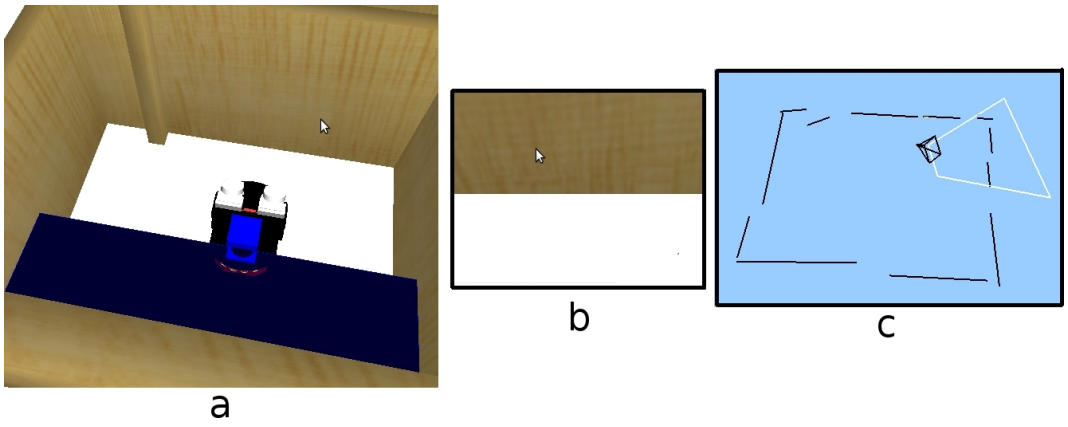

Fig. 15. (a) Situation; (b) Instantaneous image; (c) Short-term memory

2) Robot navigating a curve: In this experiment we wanted to show how the robot is unable to navigate using only the instantaneous information received from the camera.

The situation is as follows: the robot, while navigating through a corridor, is approaching a curve area (see 16-a).

If robot uses instantaneous image (16-b), it is able to see just some lines in front of itself (17-a), but with short-term memory it can observe that the path in front of itself is a curve (17-b).

In addition, robot can quickly explore the surrounding environment thanks to the visual attention mechanism, which forces the system to explore unknown areas of the environment.

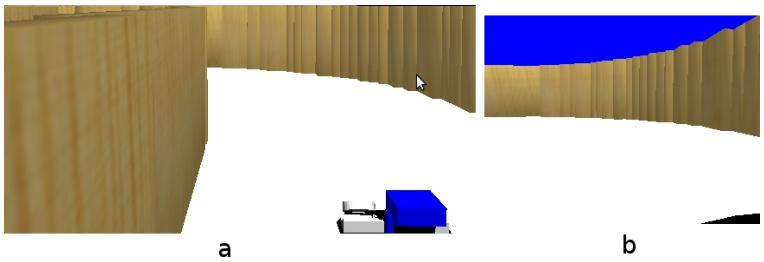

Fig. 16. (a) Situation; (b) Instantaneous image

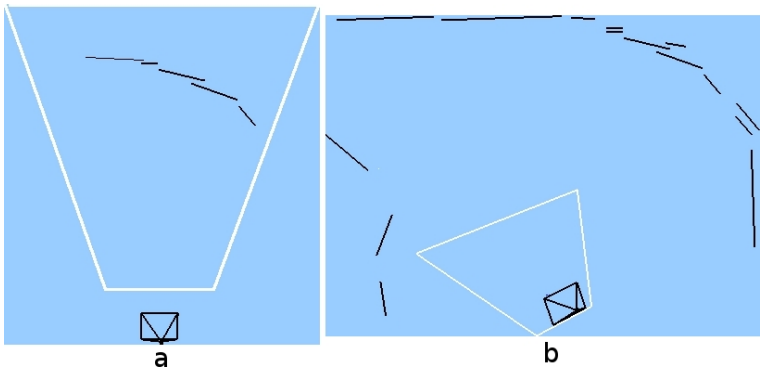

Fig. 17. (a) Instantaneous memory; (b) Short-term memory

3) Robot occlusions: Here, the situation is presented to solve a temporary occlusion. This happens very often in real environments where there are dynamic objects which can obstruct the robot field of view.

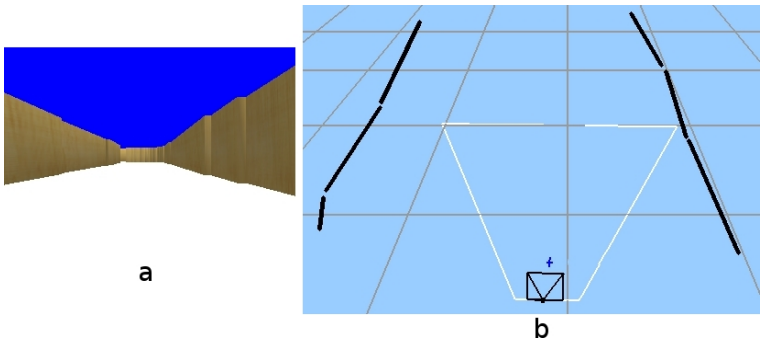

Fig. 18. (a) Situation; (b) Short-term memory got after a while

The initial situation is showed in figure 18-a. After a few seconds, robot has recovered environment information thanks to the short-term memory and the visual attention system. This information is showed in 18-b.

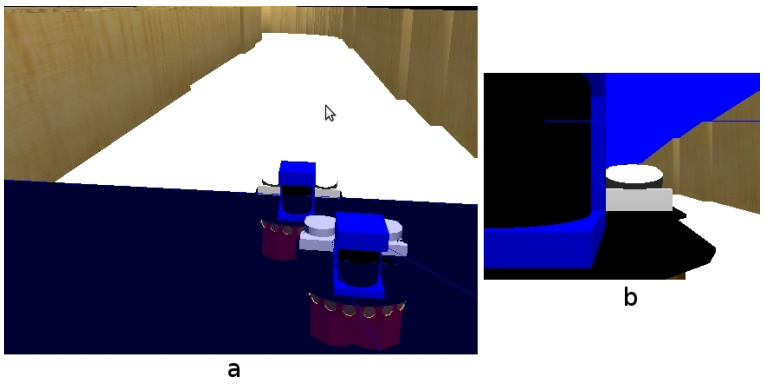

Fig. 19. (a) Situation; (b) Field of view

After a while another robot appears situation showed in figure 19-a occluding the field of view of our robot (19-b), so it is unable to see anything. This situation continue for some time while the second robot moves away from our robot (20-a,b).

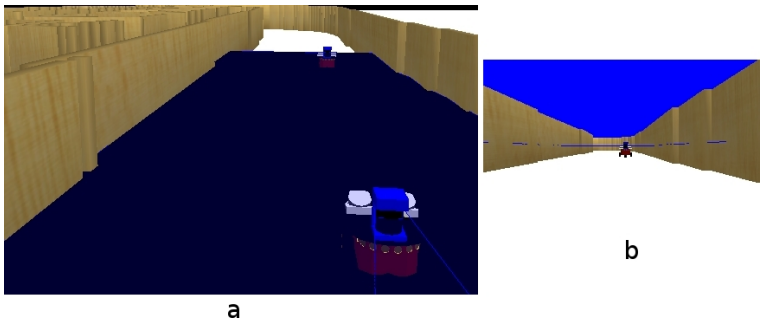

Fig. 20. (a) Situation; (b) Field of view

This situation is solved by our system because of the persistence of the short-term memory. As we discussed in section **??**, the memory is refreshed over time. If it is inconsistent, that is, what the robot sees does not match with the information stored in memory, we give a confidence interval until this situation is solved.

B. Visual localization

We have performed several experiments to validate the evolutionary algorithm as well, especially with real robots. The algorithm has several parameters to be configured, such as the maximum number of races (10), the number of exploiters (30),

and the percentages of the genetic operators we explained in Section VI-F (20% elitism, 20% crossover and 60% mutation). The maximum number of races has been a crucial factor, since a high number of races improves the precision of the calculated localization, but it also increases the algorithm execution time. Because of that, we have selected a value where efficiency and precision balance each other out, this is, 10 races.

The first experiment has been performed with a real Nao robot travelling through a corridor (Figure 21).

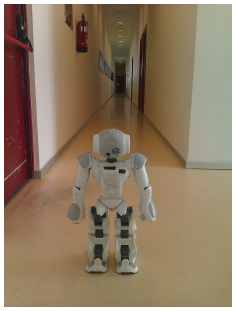

Fig. 21. Nao robot travelling a corridor for experiments

It shows how the algorithm is able to follow the real movement of the robot starting on a known position. At first, the robot is located in a known position, aftwerwards we move the robot around the environment and measure its localization error. The red line in Figure 22 shows the calculated positions, the green line the real robot path, and the brown area is the error measured.

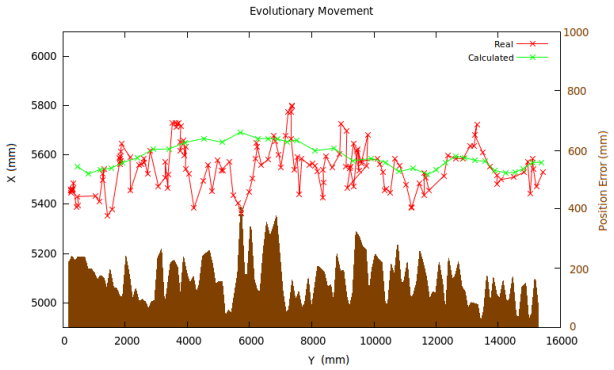

Fig. 22. Estimated localization and position error over time

The average error has been 11,8 cm and 2,1 grades, the algorithm is able to follow the robot movement even when its instantaneous observations don't provide enough information thanks to robot odometry. Besides, we can emphasize that the trajectory followed by the robot is very stable and is always close to the real location of the robot.

The second experiment shows how the algorithm works with symmetries and kidnappings . At first (1.a), we locate the robot in front of a door, so the algorithm creates several races where the robot may be located, the algorithm select one of them (a wrong one) but keeps another one on the right location, then the robot moves and obtains more information from the world and finally rules the wrong location out and selects the correct one (1.b). Afterwards, we kidnap the robot to another location (2.a) and it takes a while until the robot changes to the new right location. It happens because the location's reliability changes gradually to avoid changing with false positives, but after a while, it changes to the new position (2.b). A second kidnap is performed (3.a), this case is similar to the first one, at first it selects a wrong localization (very close to the correct one, 3.b), but after some iterations it changes to the correct one (3.c).

The average error after selecting the correct race has been 14,9 cm and 3,2 degrees. We have also measured the time spent until the algorithm calculates a new plausible pose after a kidnapping (recovery time). In this experiment was 29,6 secs.

1) Health function with instantaneous images: To validate the health function, we have implemented a debug mechanism to show graphically the value returned by the health function in different positions. In Fig. 24 we show the value returned in all positions (X, Y), where red areas are the ones with highest probability and white areas with the lowest.

As we can see, the position with highest probabilities is plausible with the input image.

In case of symmetries we will obtain high probabilities in several positions. In Fig. 25 we show what would happen if we obtain an image in front of a door, we would obtain high probabilities in front of each door of our environment:

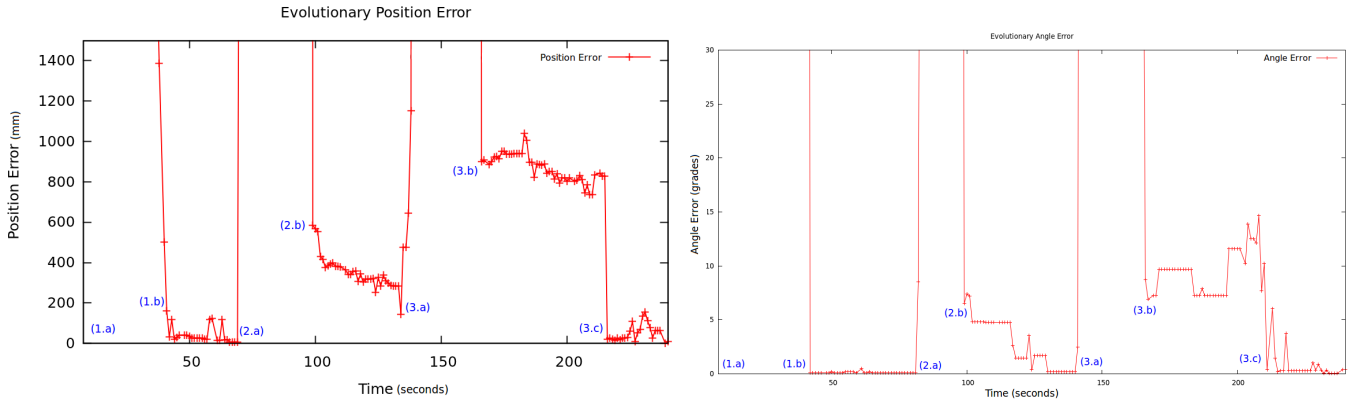

Fig. 23. Position error over time (a) and angle error over time (b)

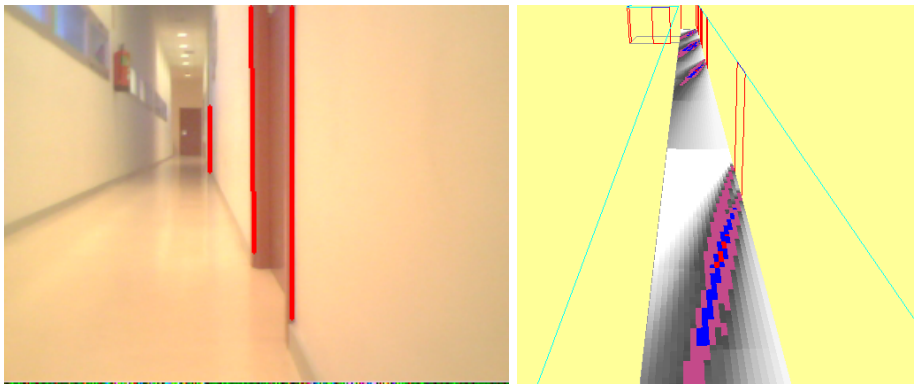

Fig. 24. Image obtained (left) and probabilities calculated with $theta$ equals to 0 radians(right)

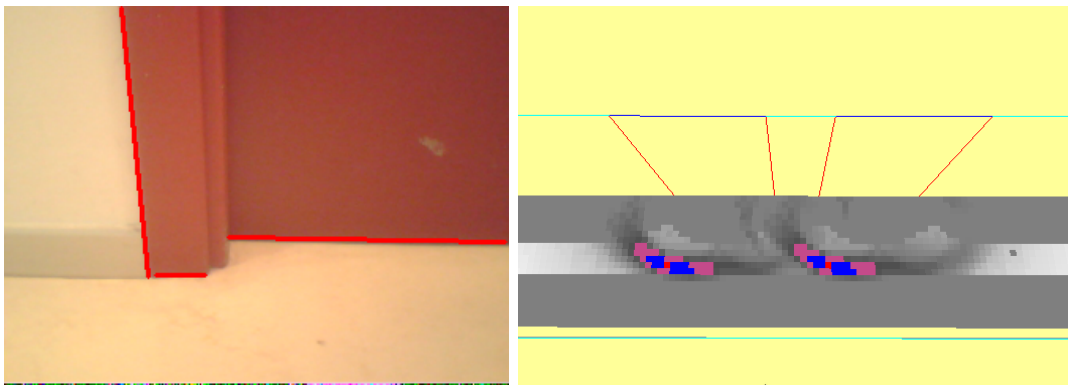

Fig. 25. Image obtained in front of a door (left) and probabilities calculated for any $theta$ (right)

The algorithm has been tested in more complex contexts, such as occlusions or false positives. In cases of occlusions the algorithm keeps a good behavior, since our approach doesn't penalize if an object is not detected in the image when it should be, and the only objection is a higher probability in some field areas (compare Fig. 26 and Fig. 27). However, false positives affects negatively to health function and the calculated localization can be totally wrong (compare Fig. 26 and Fig. 28).

C. Visual localization with attentive visual memory

In this experiment, we have tested our system including visual memory, visual attention algorithm and visual localization on a real robotics platform such as Nao humanoid robot.

The robot is in the middle of our department corridor recovering information about the environment around itself. It is able to move autonomously around the corridor; furthermore, it is moving its neck in order to detect all segments in a few seconds. We can see in figure **??** some snapshots and the short-term memory built with them.

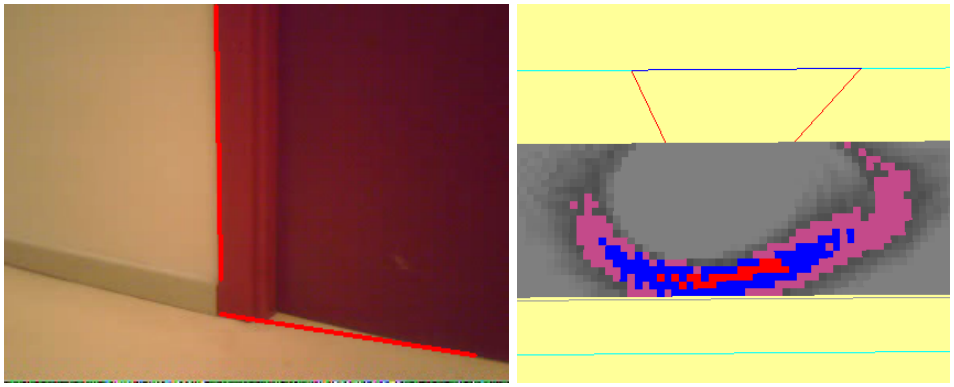

Fig. 26. Image obtained (left) and probabilities calculated without occlusions or false positives for any $theta$ (right)

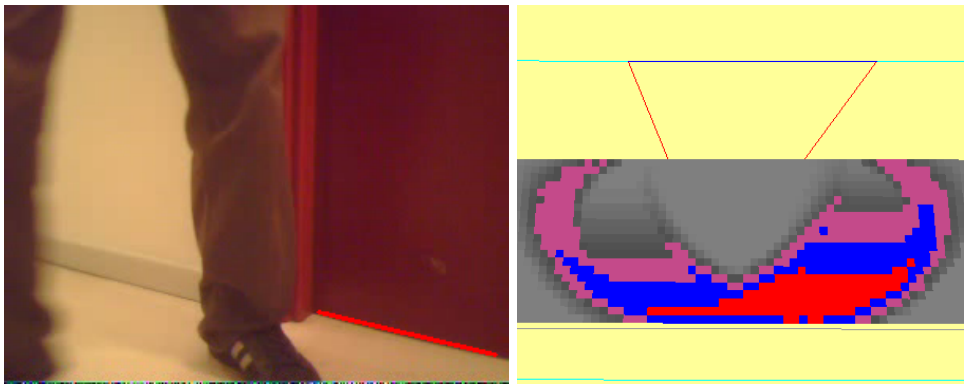

Fig. 27. Image obtained (left) and probabilities calculated with occlusions for any $theta$ (right)

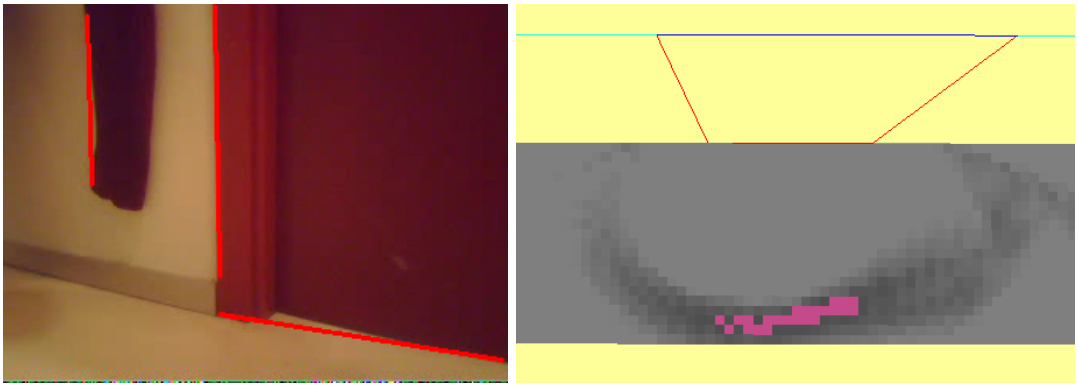

Fig. 28. Image obtained (left) and probabilities calculated with false positives for any $theta$ (right)

1) Health function: In case we use visual memory instead of instantaneous images, as we explained in Section VI-C, the calculated probabilities will be similar to previous health function, but we will get two benefits, there will be less symmetries, since we will get more information about the environment, and temporal occlusions won't affect our health function.

For instance, in Fig. 30 we show the probability map and localization calculated with the visual memory of Figure **??**.

VIII. Conclusions

Location recognition in static environments is really tough. In this work, we presented a new method for robot localization, using a 3D visual memory (see section IV) built through a sequence of images-snapshots. In particular, we adopted a 3D-information recovering method based on the *ground-hypothesis*. Although most effort in computer vision area has been on stereo-vision based methods, we decided to use the first simplified option in order to avoid the difficulty in point matching, which is very challenging in untextured surfaces.

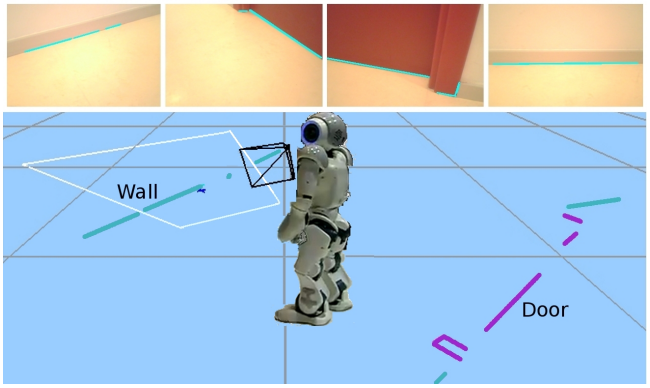

Fig. 29. Visual memory with 3D segments coming from four images of robot surroundings

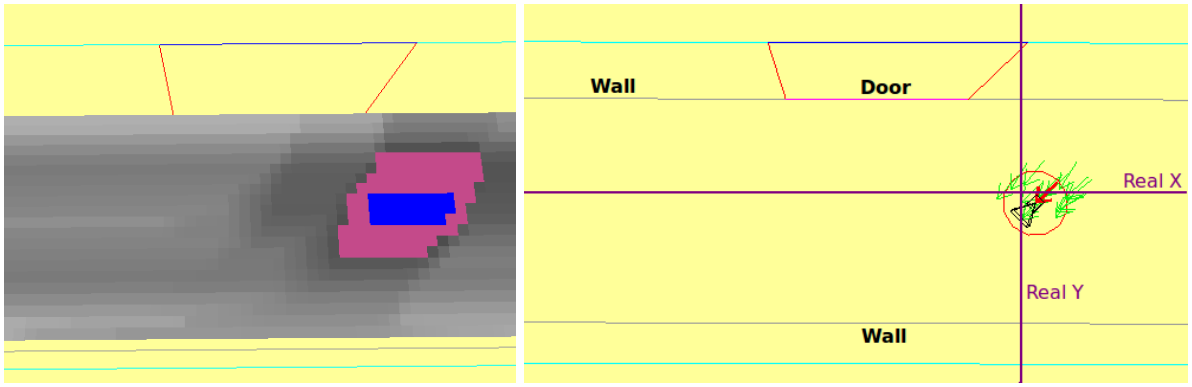

Fig. 30. Probabilities for any $theta$ (left) and localization (right) calculated with visual memory

Another difficulty we solved was to extract useful information from the richness of camera data. To do that we used different methods of image filtering, edge detection and segmentation. The mechanism based on Solis's article was a great improvement over previous methods which we used for segmentation, such as Canny algorithm or Hough transform.

Also, we have presented the implementation of a visual attention system (see section V) employing both top-down and bottom-up information. It runs in real time and it is used to control the overt attention mechanism of our robot. This eventually gives solution to a different sort of problems compared to the more typical implementations that only generate scan paths on static images.

The robot acquires information about objects through active exploration section **??**) and uses it in the attention system as a top-down primer to control the visual search of that object. The model directs the attention towards the abstract-object (parallelogram) or segmented object center of mass, similarly to the behavior observed in humans. In fact it has been proven that the first glance to a simple shape that appears in the periphery tends to look at its center of gravity.

Afterwards, we have presented a self-localization algorithm specially designed to bear symmetries (section VI). The algorithm receives as input data either 2D segments detected from instantaneous images or 3D segments from the visual memory, without influencing the rest of the algorithm.

Our system has been proven in practice to be useful to guide a robot, to select objects to be tracked (helping the visual search and recognition tasks), and to locate a robot inside a known environment in an accurate way.

Our future work will be addressed to extend our experimental results to large scenarios as well as to visit previously unexplored fields. We are also planning to design new methods for recognizing new and more complex abstract objects which are typically inside the buildings; such as tables, chairs, etc. About localization, our future work will be focused to improve the calculated localization precision and to reduce false positives by using more complex objects than segments.

REFERENCES

[Arbel y Ferrie, 2001] T. Arbel y F. Ferrie. Entropy-based gaze planning. *Image and Vision Computing, vol. 19, no. 11, pp. 779-786*, 2001.

[Badal *et al.*, 1994] S. Badal, S. Ravela, B. Draper, y A. Hanson. A practical obstacle detection and avoidance system. *Proc. of 2nd IEEE Workshop on Applications of Computer Vision, pages 97104*, 1994.

[Bajcsy, 2009] R. Bajcsy. Active perception. *Proc. of the IEEE 76, pp. 996-1005*, 2009.

[Ballard, 1991] D. H. Ballard. Animate vision. *Artificial Intelligence 48, pp. 57-86*, 1991.

[Burgard y Fox, 1997] W. Burgard y D. Fox. Active mobile robot localization by entropy minimization. *Proc. of the Euromicro Workshop on Advanced Mobile Robots, Los Alamitos, CA, pp. 155-162*, 1997.

[Cañas *et al.*, 2008] J.M. Cañas, M. Martínez de la Casa, y T. González. Overt visual attention inside jde control architecture. *International Journal of Intelligent Computing in Medical Sciences and Image Procesing. Volume 2, Number 2, pp 93-100, ISSN: 1931-308X. TSI Press, USA*, 2008.

[Courbon *et al.*, 2009] J. Courbon, Y. Mezouar, y P. Martinet. Autonomous navigation of vehicles from a visual memory using a generic camera model. *IEEE Transactions on Intelligent Transportation Systems*, 2009.

[Fox *et al.*, 1999] Dieter Fox, Wolfram Burgardy, Frank Dellaert, y Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *In Proc. of the National Conference on Artificial Intelligence*, 1999.

[Gartshore *et al.*, 2002] R. Gartshore, A. Aguado, y C. Galambos. Incremental map buildig using an occupancy grid for an autonomous monocular robot. *Proc. of Seventh International Conference on Control, Automation, Robotics and Vision ICARCV, pages 613618*, 2002.

[Goldberg *et al.*, 2002] S.B. Goldberg, M.W. Maimone, y L. Matthies. Stereo vision and rover navigation software for planetary exploration. *Proc. of IEEE Aerospace conference Proceedings, pages 52025,52036*, 2002.

[Hulse *et al.*, 2009] M. Hulse, S. McBride, y M. Lee. Implementing inhibition of return; embodied visual memory for robotic systems. *Proc. of the 9th Int. Conf. on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. Lund University Cognitive Studies, 146, pp. 189 - 190*, 2009.

[Itti y Koch, 2001] L. Itti y C. Koch. Computational modelling of visual attention. *Nature Reviews Neuroscience 2, pp. 194-203*, 2001.

[Itti y Koch., 2005] L. Itti y C. Koch. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005.

[Jensfelt y Kristensen, 2001] P. Jensfelt y S. Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Trans. on Robotics and Automation, vol. 17, no. 5, pp. 748-760*, 2001.

[Kalman, 1960] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 1960.

[Lienhart y Maydt., 2002] R. Lienhart y J. Maydt. An extended set of haar like features for rapid object detection. *IEEE ICIP 2002, volume 1, pp. 900-903*, 2002.

[Mariottini y Roumeliotis, 2011] G.L. Mariottini y S.I. Roumeliotis. Active vision-based robot localization and navigation in a visual memory. *Proc. of ICRA*, 2011.

[Marocco y Floreano, 2002] D. Marocco y D. Floreano. Active vision and feature selection in evolutionary behavioral systems. *Proc. of Int. Conf. on Simulation of Adaptive Behavior (SAB-7), pp. 247-255*, 2002.

[Nehmzow, 1993] U. Nehmzow. Animal and robot navigation. *The Biology and Technology of Intelligent Autonomous Agents*, 1993.

[Ramachandran, 1990] V. S. Ramachandran. Visual perception in people and machines. *A. Blake, editor, A.I. and the Eye, chapter 3. Wiley and Sons*, 1990.

[Remazeilles *et al.*, 2006] A. Remazeilles, F. Chaumette, y P. Gros. 3d navigation based on a visual memory. *Proc. of ICRA*, 2006.

[Simmons y Koening, 1995] Reid Simmons y Sven Koening. Probabilistic navigation in partially observable environments. *In Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, 1995.

[Sligte *et al.*, 2009] I.G. Sligte, H. Steven, y V. Lamme. V4 activity predicts the strength of visual short-term memory representations. *Neuroscience, 10 June 2009, 29(23): 7432-7438*, 2009.

[Solis *et al.*, 2009] A. Solis, A. Nayak, M. Stojmenovic, y N. Zaguia. Robust line extraction based on repeated segment directions on image contours. *Proc. of the IEEE Symposium on CISDA*, 2009.

[Srinivasan *et al.*, 2006] V. Srinivasan, S. Thurrowgood, y D. Soccol. An optical system for guidance of terrain following in uavs. *Proc. of the IEEE International Conference on Video and Signal Based Surveillance (AVSS), pages 5156*, 2006.

[Tinbergen, 1951] N. Tinbergen. The study of instinct. *Clarendon University Press, Oxford UK*, 1951.

[Tsotsos *et al.*, 1995] J. K. Tsotsos, S. Culhane, W. Wai, Y. Lai, y N. Davis. Modeling visual attention via selective tuning. *Artifical Intelligence 78, pp. 507-545*, 1995.

[Vega y Cañas, 2009] J. Vega y J.M. Cañas. Sistema de atencin visual para la interaccin persona-robot. *Workshop on Interaccin persona-robot, Robocity 2030, pp. 91-110. ISBN: 978-84-692-5987-0*, 2009.

[Vega y Cañas, 2010] J. Vega y J.M. Cañas. Memoria visual atentiva basada en conceptos para un robot mvil. *Robocity 2030, pp 107-128, Madrid, September 15th 2010. ISBN: 84-693-6777-3*, 2010.

[Vega y Cañas, 2011] J. Vega y J.M. Cañas. Attentive visual memory for robot navigation. *WAF, pp 87-94, Madrid, September 6th 2011. ISBN: 978-84-694-6730-5*, 2011.

[Viola y Jones., 2001] P. Viola y M. Jones. Rapid object detection using a boosted cascade of simple features. 2001.

[Zaharescu *et al.*, 2005] A. Zaharescu, A. L. Rothenstein, y J. K. Tsotsos. Towards a biologically plausible active visual search model. *Proc. of Int. Workshop on Attention and Performance in Computational Vision WAPCV-2004, pp. 133-147*, 2005.