Indoor Robot Navigation with Single Camera Vision

Giuseppina Gini, Alberto Marchi

DEI, Politecnico di Milano, piazza L. da Vinci 32, Milano, Italy
gini@elet.polimi.it

Abstract. In the large area of autonomous robot navigation, encompassing map creation, path planning, and self-localization, we develop the idea of a simple autonomous agent relying only on vision information. Our integrated navigation system replicates some functions of natural systems, as using little a-priori knowledge, on board computation, no omni-directional vision. Since our goal is essentially to move the robot on a floor, avoiding obstacles and people, the camera is on top of the robot and in fixed orientation to look ahead at the floor. After a simplified calibration, floor images are taken and positions of obstacles detected New images dynamically grow a grid map, constructed with simple mathematics and heuristics. For path planning obstacles are enlarged in minimum way, and path computed.

1 Introduction

An essential task in autonomous robots is to move safely in an unknown environment, possibly using artificial vision to detect and recognize obstacles. Visually guided systems are so being developed since some years. Some of them use artificial landmarks, while more advanced ones rely on natural landmarks [1-3]. The latter is the method of choice when the robot has to move in real, unstructured environments.

Moreover, visually guided navigation is important when limited autonomy is needed as in supervisory control. When remotely controlled by an operator, robots are given the task point to reach from the user, and can run towards the goal position in autonomous way. The integration of virtual reality and autonomous systems is becoming crucial, being a virtual reality model of the environment useful in the GUI user interface. In autonomous systems it is recognized that robots must build their models using their sensors, while in virtual reality it is well known that world models should automatically follow changes in the real world. All those requirements should meet to obtain a viable system.

We develop here a navigation system using as the principal source of data a vision system. A single camera can solve problems in an indoor environment, in a moderately dynamic system, when the robot moves on a plane surface.

The subtasks to approach are so the classical ones:

- From images to models (map learning);
- Path planning and Obstacle avoidance;
- Self-localisation.

Map learning is a well studied area [4-6], but map creation using a single camera can offer advantages over more reliable but expensive solutions as laser beams. After calibration we can transform the visual information of the floor into a grooving grip map, setting a confidence value that the position is free or occupied by an obstacle. Self-localization with natural landmarks is again an important field, and we solve it as a matching problem between maps.

Natural landmarks require comparing the sensory data with the known maps of the environment. The results give the possible position of the robot, usually as a probability density function. After more self-localizations, the uncertainty is reduced. The usual algorithm applies Kalman filtering to integrate past and present data. In our case instead we build a local map using the sensory data, and we integrate the local into the total map.

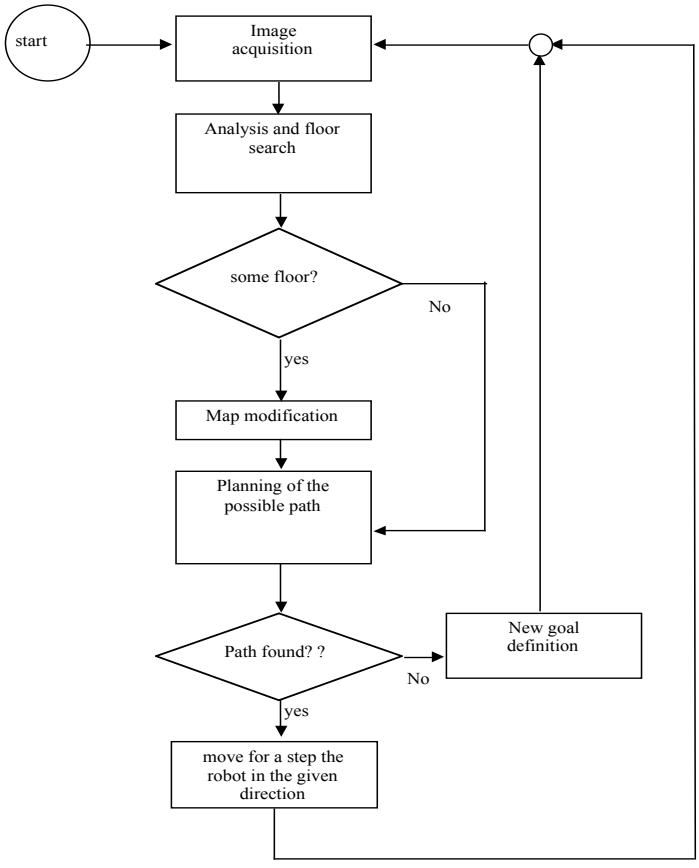

Fig. 1. Flow-chart of the whole system

In order to create the map the robot has to recognise where obstacles are located and where the floor is clear by discriminating pixels belonging to the floor from those which do not. Our algorithm makes statistical analyses on the intensity and saturation

of every single pixel in the image. The main computing steps of the system are shown in Fig. 1. Our experimental setup includes a Robuter, a PC, and a color camera. The Robuter® is a mobile robot with differential drive, equipped with a sonar belt. The on-board computer executes the motion commands and communicates at 9600 baud through a serial link with a PC. The camera is from Sony, PAL standard, with 768 x 512 pixels, and is fixed and pointing to the floor. Matrox Meteor frame grabber is used in single acquisition. The external personal computer, Pentium II 350Mhz, 64 Mb RAM, is for user interface and vision analysis. Programs are developed in Borland C Builder.

The strong points we will illustrate in the following Sections are:

- New reduced calibration algorithm
- Floor analysis and obstacles detection in single camera images
- Map creation using only visual data
- The navigation algorithm and self-localisation as maps matching.

2 Camera calibration

To find the correspondence between the real coordinates and the image coordinates we developed a calibration procedure [7]. As usually we chose the pin-hole model, which gives a way to compute the world coordinates from the image coordinates and the focal distance f. The **H** matrix gives this transform.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = H * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1}$$

Since we need to account for other relevant characteristics, in particular: a) the principal point is not exactly in the projection center; b) the aspect ratio is relevant; c) the axes in the image are not necessarily orthogonal and a parameter α is needed, we build a K matrix from image to geometric-image considering that:

- the origin of the geometric system is $(u_0\ v_0)$,
- the transform of u is $[1, 0, 0]^T$
- the transform of v is the product a . $[\alpha, 1, 0]^T$.

$$K = \begin{bmatrix} 1 & a*\alpha & u_0 \\ 0 & a & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The complete transform from scene to image is so:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = K * \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * H * \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \tag{3}$$

Since we are interested only in the transform between the two reference frames, we can multiply the three matrices and obtain **M**:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} m1 & m2 & m3 & m4 \\ m5 & m6 & m7 & m8 \\ m9 & m10 & m11 & m12 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = M * \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \quad (4)$$

To calibrate the camera we work in two phases: first from image to floor, then from floor to robot. The estimate of the twelve elements of the matrix M is reduced to eleven considering the scale factor.

In the *first calibration phase* the camera takes a picture of a calibration object whose dimension is known. We do not want to use the classical least-squares method, which requires precise measures in world coordinates, so we choose all the points of the calibration object to be on the floor (z is null, and 3 elements of M are null). The calibration object is a white square, 21-cm width. The vertex coordinates are computed. The estimate of M is done on the first picture using least squares and trying to match the reference square. The initial estimate is then improved with Newton method. This is a minimization problem, where the function to minimize is the difference between the estimated segment length and the real length.

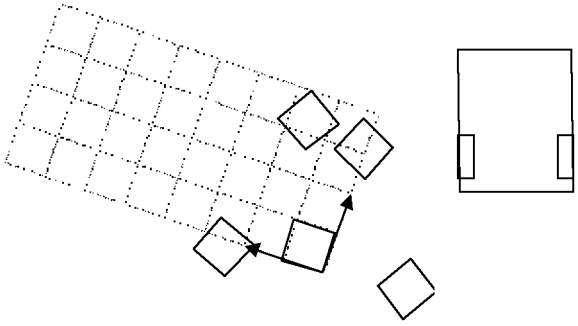

Fig. 2. The reference system from the first calibration phase is robot independent

After the reference system on the floor (see Fig. 2), we construct the matrix to transform it in the reference of the robot. During this *second calibration phase*, pictures of the object are taken from different positions and orientations of the robot, and again this minimization problem is solved as before.

We consider as robot reference system the one used by dead-reakoning of the ROBUTER®. Let x be the coordinate vector on the ROBUTER,

$$x = T * Bi\_to\_Tri(\tilde{x})$$

where $\tilde{x}$ is the vector of the point in the image, *Bi_to_Tri* is the function computing the floor point in the reference computed in the first phase. The **T** matrix to estimate is:

$$T = \begin{bmatrix} a & b & 0 & h \\ c & d & 0 & k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 & \Delta x \\ -\sin(\alpha) & \cos(\alpha) & 0 & \Delta y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

where Δx, Δy, and 0 are translations, α the rotation between the two systems. To compute **T** we put the square on the floor, collect the coordinates of the vertices and the odometry many times moving the robot to obtain coordinates for different viewpoints. Remember that **T** is a roto-translation matrix, where non-linear conditions should hold:

$$\begin{aligned} a^2 + c^2 &= 1 \\ b^2 + d^2 &= 1 \\ a*b + c*d &= 0 \end{aligned} \tag{6}$$

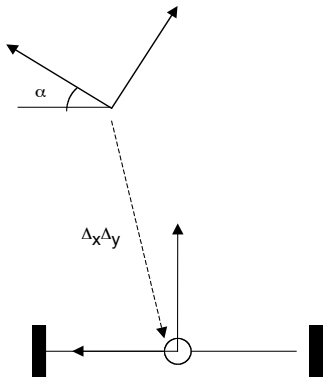

Fig. 3. From the reference system on the floor to the robot reference system

To solve on the unknown α, Δx, and Δy (see Fig. 3) we define a function which accepts the four vertices coordinates of a square in two different positions and the robot pose. The function uses **M** and the available estimation of **T** to project the image points in the world points and computes the distance. The distance is then minimized (since the two points are the same physical point). The algorithm is iterative, starts from a brute estimation of **T**, and divides the parameters space in cells. For each cell **T** and the error are computed. After building **M** and **T**, a point in Cartesian space can be transformed into the pixel coordinates using:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = M * T^{-1} * \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

$$u = \frac{u'}{w'} \tag{7}$$

$$v = \frac{v'}{w'}$$

For the reverse transformation, we invert the equation and get the formulas:

$$S = \frac{m8 - v*m12 - \dfrac{(v*m9 - m5)*(m4 - m12*u)}{u*m9 - m1}}{\dfrac{(v*m9 - m5)*(m2 - u*m10)}{u*m9 - m1} + v*m10 - m6}$$

$$D = \frac{(m2 - u*m10)*S + m4 - m12*u}{u*m9 - m1}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = T* \begin{bmatrix} S \\ D \\ 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x'/z' \\ y'/z' \\ 0 \end{bmatrix} \tag{8}$$

The calibration problem is so completely solved.

3 Image analysis

The basic hypothesis is that the floor has a uniform texture. Using statistical analysis we can extract from the picture of the floor the areas occupied by obstacles because they change the regular pattern [8]. Since the image is in RGB format; for each pixel the algorithm computes the mean and the 4th order moment for each of the RGB component of the pixel. The formulas for the red component are:

$$\mu\_red(x,y) = \frac{\sum_{(i,j)\in\Box(x,y)} red(i,j)}{N}$$

and

$$m\_red(x,y) = \frac{\sum_{(i,j)\in\Box(x,y)} (red(i,j) - \mu\_red(i,j))^4}{N} \tag{9}$$

(where N is the number of pixels in the region)

The 4th order moment is significant because it is a measure of the disparity of the pixels in the considered region. It has low values in uniform areas, high when there is a sharp change of the colors, as when an obstacle is seen on a floor. To reduce computations the average and the moment are not computed for every pixel but for a subset of uniformly distributed pixels and by linear interpolation for the other pixels. After computation a new image is created with new components for pixels constructed from mean and moment for each color:

$$new\_red = \mu\_red(x,y) + (red(x,y) - \mu\_red(x,y)) * m\_red\_norm(x,y) \tag{10}$$

The new image is then transformed in HSL[1] format, filtered and transformed in binary, removing the obstacles. This requires choosing a dynamic threshold; to compute it for each frame we use a little square region in the bottom part of the image as example of the floor. In this region the mean of L and H are computed and used as the reference values for the floor. The formulas to compute the dynamic threshold for

[1] RGB is an additive model, with 3 primary colors. HSL, instead, describes the image as saturation (S), Luminance (L, 0 for black, 1 for white) and hue (H, an angular value).

L and H contain a static component and a dynamic component, modified by a function f acting as a filter. The static and dynamic parameters are manually set and depend on the floor texture and on illumination conditions. The function f is:

$$fx = \frac{(3 - \sum_{red,green,blue} \frac{moment(x,y)}{moment\_\max})}{3} \quad (11)$$

The final picture represents the floor in white, the obstacles in black, as in Fig. 4.

Fig. 4. An example of image analysis

4 Map creation and navigation

The map is a grid map, with square cells initialized to a mean numeric value. The value represents how much the robot "trusts" in the cell classification defined as free, obstacle, or unknown. Whenever the cell is seen again as free its vote is increased, if occupied is decreased; The value is a vote, which can filter also obstacles in movement because temporary obstacles do not affect too much the value of a free cell. The size of the grid cell used can be chosen considering the needed performances: using the Robuter, cells with a side of 5 cm give very good results.

Using the parameters obtained from the camera calibration, the binary image is mapped on the floor plane and added to the map, in a position obtained using the estimated pose. So, the map is created and enlarged after every image analyzed. Considering the images in Fig. 4, the map generated is in Figure 5. The gray pixels represent free space, the darker on the edges represent obstacles, the black ones are unknown and the gray all around are unknown and not yet allocated in memory (to save memory). At start the robot has no information about the environment. To create the map the start position of the robot is defined, and a destination assigned. To reach the destination the robot will explore the world, looking at the floor and generating the first obstacles. Every time a new obstacle is detected, the robot computes the path to reach the destination, and the map grows. To map the entire environment it will be enough to give a goal location unreachable, as outside a wall.

Using the map the robot is able to navigate autonomously in the environment towards a specified goal given as robot coordinates [9, 10]. The path is found with A*

on the visibility-graph of expanded obstacles. Obstacles are enlarged by the half of the robot-width, not considering its length; in fact the robot moves forward and keeps obstacles on the sides. To allow the robot to move in narrow corridors with curves, the obstacles are enlarged and also smoothed. In this way, the robot can pass through very small passages, only a few centimeters wider than the robot. However, not considering the robot length implies that we have to take care of possible frontal collisions with obstacles; this is avoided using the sonars during the movement. In the same way highly dynamic obstacles are avoided just stopping and switching to another path.

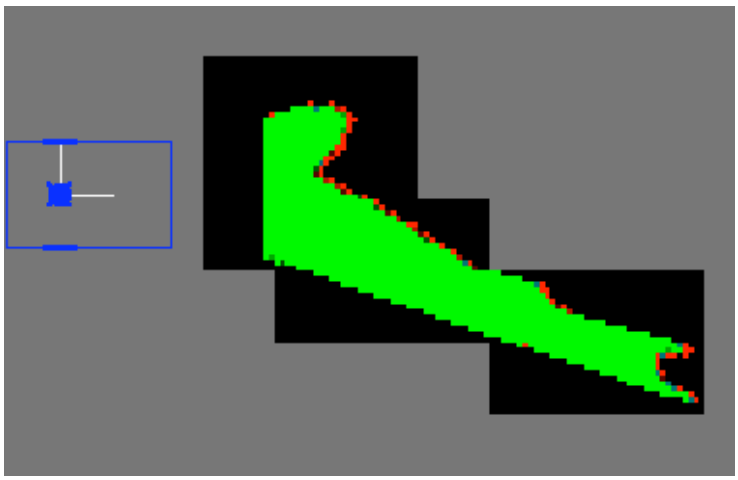

Fig. 5. The map part constructed from Fig. 4.

5 Self-localization

When the robot has a map of the environment, it can auto-localize itself [11]. To do this, it creates a new map, starting from scratch, and compares it with the complete map. The comparison is based on the angles between the walls. The robot can upgrade its location matching the global and the local maps.

The self-location problem is important when the robot has to move in autonomous way. Dead reakoning reduces the location error, but is unable to keep the error under a given threshold. When used in tele-control the self location is again important: the robot starts from any position, builds a map with any origin, but we need to match the new map with the standard one that is in the user interface.

So the problem is to match a partial map onto a complete map, as in Fig.6.

The features to match are necessarily obtained from image analysis. Since the vector quantization is imprecise, we switched to a more robust feature, the angles. Angles too are affected by errors in vector quantization, the vertex position can be slightly different, but the angle in indoor environments is usually of 90°, or 180° if two vectors are found for a single segment. Starting from vector quantization we have

to individuate segments having a common vertex. The user can set a parameter to fix the threshold for the distance of segments to be considered as forming an angle.

The map matching will find the correspondence (position and orientation) of the partial map in the global map using a vote system. Two lists of angles in the two maps are maintained: they report position, orientation, and amplitude. Each local angle is matched against all possible global angles and the rototraslation matrix is computed. Every match generates a positioning hypothesis of the local map with an associated vote (weight)

$$weight = 10 * e^{\frac{-|amplitude1 - amplitude2|}{180}} \qquad (12)$$

with the maximum for same amplitude angles.

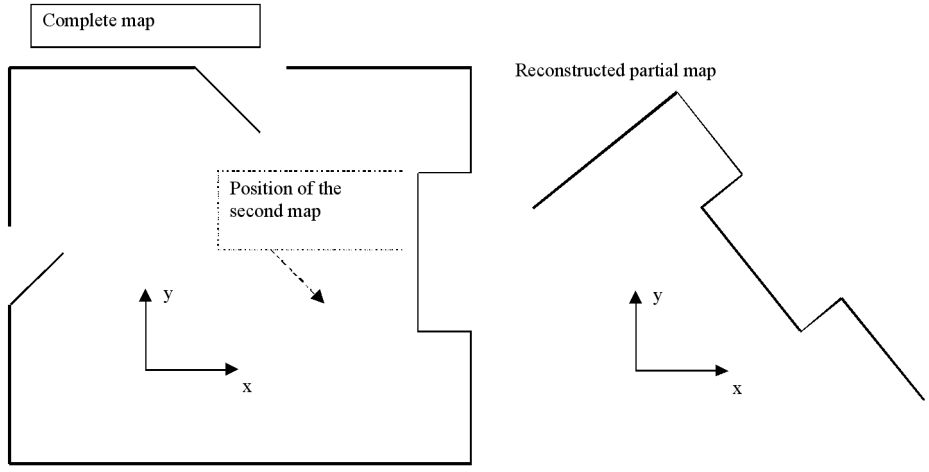

Fig. 6. Matching a global and a partial map.

Each hypothesis has low weight if the match of the angles is poor. The real position has a high weight, since all the angles are correctly matched. To compute the best position we have to find the hypothesis with the highest vote. To account for the imprecision about the data we associate a gaussian, with maximum equal to the vote, to every position. Summing up all the gaussians we obtain a surface with a maximum in the position where the probability to have the robot is maximum.

This algorithm is very good in simulation, but can encounter problems is real world. To improve it we introduce the segments. For each local maximum the vote of the associated position increases if this position matches two segments in the maps. Segment matching is crucial because different segments can describe the same line.

The matching is confirmed only when:

- The angular difference is less than a threshold
- The centroid distance of the segments is less than half of the greater segment
- The maximum distance of the segments is less than a threshold.

The new weight is:

$$weight = K * e^{\frac{-|amplitude1 - amplitude2|}{K1}} * e^{\frac{-\max\_distance}{K2}} \quad (13)$$

which gives a low weight to segments far away and with different amplitudes. In this way only the correct position gets a high vote.

6 Conclusions

The unusual aspects of our navigation system are its simple design, efficient use of special properties of the environment, and large autonomy. Its efficiency and reliability are due to various factors. The specialization of the environment allows the robot to reduce the vision computation. The use of simple reactive strategies reduces the risk of failures, because obstacle avoidance is always active. Finally, the environment does not need any modification to insert the robot. As illustrated in the previous sections, the uncertainty of the world is simply managed through heuristics and through a strict use of information only obtained through sensors.

References

1. Ayache Nicholas "Artificial vision for mobile robots" The MIT Press, Cambridge, Massachusetts, 1991.
2. Betke Margrit, Gurvits Leonid “Mobile robot localization using landmarks”, IEEE Transaction on robotics and automation, Vol 13, No. 2, April 1997, p.251-263.
3. Horswill Ian, “Polly: A Vision-Based Artificial Agent” Proceedings of AAAI-93, AAAI Press/The MIT Press.
4. Anousaki G.C., Kyriakopoulos K.J. “Simultaneous Localization and Map Building for Mobile Robot Navigation”, IEEE Robotics & Automation Magazine, September 1999, p 42-53.
5. Miller David P., Slack Marc G. “Global symbolic maps from local navigation” Proceedings of AAAI-91, AAAI Press/The MIT Press, p 750-755.
6. Thrun Sebastian “Learning metric-topological maps for indoor mobile robot navigation”, Artificial Intelligence, 1998, Vol 99, p 21-71.
7. Freeman Herbert "Machine vision for inspection and measurement" Academic Press, San Diego 1989.
8. Mirmehdi Majod, Petrou Maria “Segmentation of color textures”, IEEE Trans PAMI, Vol 22, n° 2, February 2000, p 142-159.
9. Latombe Jean-Claude "Robot motion planning" Kluwer, Boston, 1991.
10. Canny John F. "The complexity of robot motion planning" The MIT Press, Cambridge, Massachusetts, 1987.
11. Burgard Wolfram, Fox Dieter, Thrun Sebastian “Active mobile robot localization”, Robotics, 1997, p 1346-1352.