Real-time Extraction of Colored Segments for Robot Visual Navigation *

P.E. López-de-Teruel[1], A. Ruiz[2], G. García-Mateos[2], and J.M. García[1]

[1] Dpto. de Ingeniería y Tecnología de Computadores
[2] Dpto. de Informática y Sistemas
Universidad de Murcia (Spain)
{pedroe,jmgarcia}@ditec.um.es,{aruiz,ginesgm}@um.es

Abstract. We propose an image representation method appropriate for real-time visual geometry applications. If the expressive power of raw segments is augmented with robust color information from their two sides, the most relevant geometric and photometric structure in the image can be concisely captured. In this paper we describe an efficient algorithm to compute this kind of representation, which can be successfully exploited in several projective geometry problems, such as 3D reconstruction, motion estimation or calibration, and in interpretation related tasks. We also show how these enhanced primitives are powerful enough to recover a very acceptable approximation of the original image, especially for partially structured scenes, like interior of buildings, man-made objects, and so on. The algorithm works at frame rate for medium size images (PAL/2), using low-cost hardware, such as an off-the-self image acquisition card and a standard PC. This makes it very useful as an *on-line* feature extraction method for robot visual navigation, where more elaborated (and slower) methods can not be used. We describe some applications of the algorithm in this kind of tasks.

1 Introduction

3D reconstruction and scene interpretation have been active research topics in the last years, mostly due to significant advances in the application of projective geometry to computer vision [8]. This approach typically uses points, lines and segments as working primitives when trying to perform scene reconstruction, autocalibration, and object localization and recognition tasks. In the robotics community, however, some authors have highlighted the fact that successful achievements in reconstruction of scenes from image sequences have not been accompanied by advances in real-time methods [5]. This is due to the complexity of some of the involved methods, as well as their inherent *batch* methodology, where sequences of images are first acquired and then analysed *off-line*.

Color information, on the other hand, has also been successfully used in several real-time vision problems, such as segmentation [2], tracking [12], or

* This work has been partially supported by the Spanish CICYT under grants DPI-2001-0469-C03-01 and TIC-2000-1151-C07-03.

learning and classification [3]. But these approaches usually discard geometric information, often working only with unstructured sets of spatially related pixels.

In this paper we propose a computationally efficient algorithm to reduce images to sets of colored segments, trying to take advantage of both the color information and the geometric structure of the scene. We justify that the obtained representation effectively summarizes the contents of the input data, by showing how the original image can be recovered with high accuracy using only the extracted information. Finally, we show a number of applications in a machine vision system with strong real-time requirements: the visual calibration and guidance of a robot through an indoor environment, that performs *on-line* reconstruction and interpretation of the scene while navigating.

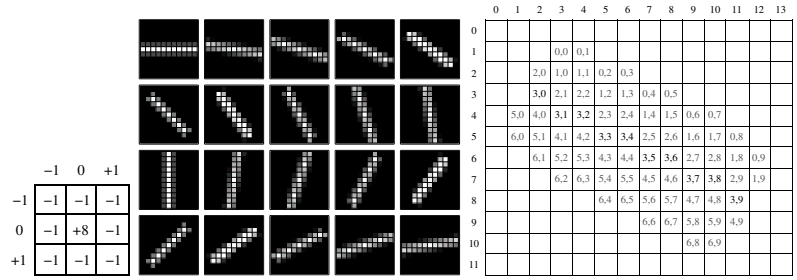
2 Color Augmented Segment Extraction

Finding lines or segments in an image is a basic problem frequently studied in the computer vision literature. Most of the available methods are variations of the well known Hough Transform [10]. Other authors prefer to find segments by locally grouping edge pixels into contours (for example, using the Canny operator [4]), which are then split using piecewise linear polygonal approximations [13]. A drawback of many of these methods is that they only account for geometric properties of the segment (i.e. position in the image), thus discarding other visual information useful for tracking or interpretation, such as color or texture of the local vicinity of each segment. Of course, there are some exceptions (see, for example, [14]), but using the neighborhood of the features often slows the process down, making it unusable under real-time constraints.

An alternative is to describe the segment vicinity in a simple, robust and accurate way, for example, sampling color from both sides of the segment. Color information is obtained in the segment extraction phase itself, with negligible additional computational effort. This is especially appropriate for real-time applications. In the following we explain the fundamentals of our approach, including a detailed algorithm description.

Preprocessing: Most of the existing edge segmentation methods rely on a preprocessing stage to remove noise from the input image, for example by means of a Gaussian or median filter. Then, the image gradient is approximated using a pair of perpendicularly oriented masks, such as Roberts' or Sobel's, and this output is postprocessed to estimate both the magnitude and the orientation of the gradient. This information is finally used to group pixels into lines, or any other edge primitive (circles, contours, etc). This is the basis of the extensively used Canny edge detector [4], and similar line extractor procedures [1].

Instead, we only perform a simple high-pass filtering of the input image, to minimise computational effort. This filtering will be followed by an efficient sequential grouping stage that takes into account the local orientation of the edges, and that, at the same time, has a noise suppression effect (see next paragraph). Therefore, the input (gray converted) image must only be preprocessed using a simple 3×3 mask (see Fig. 1a), alleviating the computational burden of image



(a) (b) (c)

Fig. 1. (a) High-pass mask. (b) Density plot (0-1 valued) of the orientation masks for $R = 5$, $A = 20$ and $W_{edg} = 2.80$. (c) Table of pixel positions for the vicinity of width $W_{seg} = 5.0$ of the segment $((y_1, x_1), (y_2, x_2)) = ((3, 2), (8, 11))$, (a pair (i, j) in position (y, x) means that table position (i, j) stores image position (y, x)).

preprocessing. Fig. 3b shows the result of applying the mask to the input image of Fig. 3a. If the input image is very finely textured, a previous noise removal filtering of the monochrome image (3×3 median or Gaussian, for example) could also be useful. This would still be faster than the preprocessing stages of classical methods.

Local orientation and grouping: To find segments in the high-pass image obtained in the previous step, we look for local alignments of pixels around the edge input points (that is, those with high response to the filter). For this, our method computes the response to a set of adequate masks (see Fig. 1b) centered in those pixels. These masks have been designed to estimate a local orientation with the desired accuracy from noisy input edge images. This is an alternative to traditional gradient based methods. To control its sensitivity, the masks are generated depending on three parameters: R, the radius in pixels of each mask, W_{edg}, the expected *edgel* width, and A, the number of discrete orientations to be considered. For example, the set of masks of Fig. 1b has been generated using $R = 5$, $W_{edg} = 2.80$ and $A = 20$. Reasonable values for these parameters are in the ranges $R \in \{3, 4, 5\}$, $W_{edg} \in [1.50, 3.0]$ and $A \approx 4R$. In general, the smaller the values of R, W_{edg} and A, the faster the algorithm, but also the less accurate.

The grouping method proceeds as follows: if a pixel value in the edge image is greater than a given threshold τ_h, the masks in every orientation are applied centered on it, to get A output values, of which only the maximum is taken into account. Therefore, the filter is a nonlinear robust estimator, despite of the linear nature of the individual orientation detectors. A new segment is then created, and a grouping process is initiated in the selected orientation, following a precomputed pixel ordering, beginning in the mask center and towards the extremes. This process is recursively repeated in each segment extreme, clustering points with high-pass response greater than another lower threshold τ_l, in both directions. Thus, parameters τ_h and τ_l are analogous to those of the Canny hysteresis procedure [4]. An additional parameter is n_{min}, the minimum number of collected points in a window to keep on grouping edge pixels for that segment before stopping. The process also finishes if orientation in the current segment extreme differs from the original one in more than one step. This is

a tolerance threshold to cope with slight variations caused by noise. Captured points are marked as visited to be discarded in subsequent processing. Note that the masks are not applied to all the pixels, but only to a small number of the active ones in the edge image. Finally, the centroid and main eigenvector of the covariance matrix of the set of points are used to accurately adjust the segment extremes. The grouping and segment refinement processes are summarized in the corresponding sections of the algorithm, shown in Fig. 2.

Color sampling and postprocessing: When a segment is found, the next step is to label it with robust color information. We use the median red, green and blue values of image pixels in both sides of the segment. Our experiments provide extensive evidence that, contrary to intuition, independent estimation of median color channels does not generate color artifacts. The obtained color models are, for all practical purposes, perceptually indistinguishable from the original ones. Therefore, the computational overhead of a more complex robust multivariate estimator is not justified.

Color sampling is performed by scanning the pixels in the vicinity of the segment, using an access table computed to this effect. The table contains image positions located along both sides of the segment (see Fig. 1c for an example). Its generation is based on Bresenham's algorithm, a standard computer graphics method for drawing lines on pixel grids [7]. We extend this algorithm to "thicken" the segment, by replicating the pixels in both sides. The algorithm uses only $O(W_{seg}L)$ integer operations, being L the segment length and W_{seg} the desired width. So, even though the table must be recalculated for each new segment, this will not be a computational bottleneck.

Tables computed with this algorithm have two advantages. First, they cover the segment neighborhood *densely*, that is, without leaving uncovered pixels. Second, they also arrange the pixels in an *orderly* fashion, by means of a rectangular array of positions with r rows and c columns, ($(7, 10)$, respectively, in the example table of Fig. 1c). These tables will be used for two complementary tasks, as can be seen in the last section of the algorithm in Fig. 2: First, to find the median value of the color channels for each side of the segment, using rows 0 to $\lfloor \frac{r}{2} \rfloor - 1$ for the left side, and $\lfloor \frac{r}{2} \rfloor + 1$ to $r - 1$, for the right one. Second, to clean up the surroundings of the segment of possible edge points that were not caught during the grouping process, marking them as visited. This avoids the extraction of spurious small segments close to the good ones. For this, we can take the central rows of the table (for example, covering a width of $W_{seg}/3$).

Performance evaluation: Fig. 2 outlines the colored segment extraction algorithm. We have implemented it using the OpenCV and Intel Image Processing Libraries [9] for the basic operations (RGB to gray conversion, high-pass filtering, median or Gaussian previous smoothing for noisy images, and so on). These libraries are optimized to run on the different Pentium processors. The procedure works at 15-25 fps (depending on the complexity of the scene) for 288×384 (PAL/2) images, running on a 533 MHz Pentium III. For more modern ($\sim$1 GHz) processors, it works always at the camera frame rate (25 fps). We show an example of the segments obtained in an indoor scene in Fig. 3c.

INPUT:
- RGB image (I^{RGB}).
- Algorithm parameters (τ_h, τ_l, W_{seg}, and n_{min}, see text).
- Weighting masks and pixel paths for each orientation (see Fig. 1b).

OUTPUT:
- Set of S segments with left and right color information.

ALGORITHM:

Initialisation and preprocessing:
- Initialize the segment counter, $S := 0$.
- Convert the RGB image, I^{RGB}, to get the gray image I^{gray}.
- Filter I^{gray} using the high-pass mask (Fig. 1a), to get I^{hp} (use absolute value).
- Mark all the pixels in the edge image I^{hp} as "not visited".

High-pass image traversal:

for each image position (y, x) ***do***

if $I^{hp}_{(y,x)} \geq \tau_h$ ***and*** (y, x) is marked as "not visited" ***then***

- Increment the segment counter, $S := S + 1$.

Local orientation and grouping:

- Apply the set of orientation masks (Fig. 1b) to I^{hp}, centered on (y, x). Let a^{max} be the mask index with maximum response.

for $i := 1$ ***to*** 2 ***do*** (both directions)

- Initialize current orientation, $a^{cur} := a^{max}$.
- Initialize current segment extreme, $(y_i, x_i) := (y, x)$.

repeat

- Using the precomputed pixel path for orientation a^{cur} and current direction i, centered on (y_i, x_i), keep on capturing image pixels with $I^{hp} > \tau_l$, marking them as "visited".
- Update current segment extreme with the last captured point in the previous step, $(y_i, x_i) := (y^{last}, x^{last})$.
- Apply the set of masks (Fig. 1b) to I^{hp} again, but now centered on (y_i, x_i). Let a^{cur} be the new mask index (orientation) with maximum response.

until (Number of captured points in this step $< n_{min}$) ***or*** (Distance from a^{cur} to $a^{max} > 1$)

endfor

Segment refinement:

- Compute the centroid, covariance matrix and major eigenvector of the set of points captured for the current segment S.
- Update segment extremes (y_1, x_1) and (y_2, x_2) by projecting them onto the principal direction computed in the previous step.

Color sampling and postprocessing:

- Compute access table (Fig. 1c) corresponding to segment S.
- Compute the median of the RGB channels of image I separately, for the set of pixels whose position is indicated by rows $0 \ldots \lfloor \frac{r}{2} \rfloor - 1$ of the access table. The obtained vector $(r^S_{left}, g^S_{left}, b^S_{left})$ is the left color information for segment S. Repeat the process for rows $\lfloor \frac{r}{2} \rfloor + 1 \ldots r - 1$ to get $(r^S_{right}, g^S_{right}, b^S_{right})$.
- Mark pixels pointed by rows $\lfloor \frac{r}{2} \rfloor - \lfloor \frac{r}{6} \rfloor \ldots \lfloor \frac{r}{2} \rfloor + \lfloor \frac{r}{6} \rfloor$ of the table as "visited".

endif

endfor

Fig. 2. Algorithm that extracts colored segments from a RGB image.

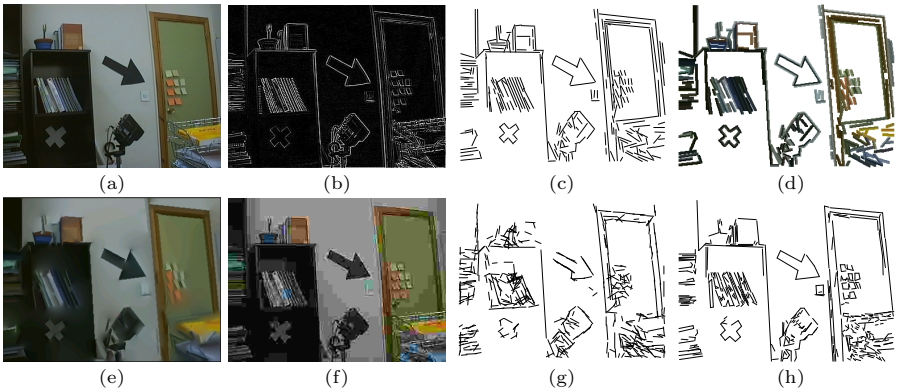


Fig. 3. Results in a typical indoor image: (a) Original image. (b) High-pass filtered image. (c) Segments extracted by our method. (d) Segments with color information. (e) Recovered image. (f) JPEG image with the same compression ratio. (g) Segments extracted by PPHT. (h) Segments extracted by polygonal approximation.

Due to space limitations, we cannot include here an exhaustive study of the execution time and precision of the algorithm. Nevertheless, to give a flavor of the advantages of our approach, we will compare it with two popular segment extractors, representing the alternatives mentioned at the beginning of the section. Anyway, a direct comparison is difficult, since, to the best of our knowledge, methods dealing with color have not been previously described in the literature. First, we tested the *Progressive Probabilistic Hough Transform (PPHT)*, a modern variant of the classical HT adapted to run in real-time applications [11]. For the same image size, scene and CPU, the PPHT method, as implemented in the OpenCV, obtains qualitatively worse segments (see Fig. 3g) while running at only 3-5 fps on average, including the necessary edge detection stage using the Canny operator. The second method is based on polygonal approximations [13] of the contours obtained by the Canny procedure. Though not directly implemented in the OpenCV, this alternative obtains better solutions than those of the PPHT (see Fig. 3h), but still worse than our segment extractor (Fig. 3c), and at a lower rate (7-9 fps). Observe also that, besides being faster and more robust, our method adds color information to the segments, making them more useful for many applications, as we will show in the next sections.

3 Expressive Power of Colored Segment Representation

The color augmented segment is a very powerful visual representation primitive, whose main advantages can be exploited in a number of general image processing and computer vision tasks:

Image compression: Image data can be concisely represented in terms of such primitives without significant loss of 'large scale' photometric information. A simple iterative diffusion procedure can be used to recover a good approximation to the original pixel array. The pixels located immediately to the left and

right of each segment are initialized and fixed (using tables like the one shown in Fig. 1c) with their respective RGB median values (see Fig. 3d). The rest of pixels are marked as uninitialized. Then, in each iteration a RGB pixel value is recomputed as the mean of its initialized neighbors in a 3×3 window, and marked as initialized too. The procedure acts as an iterative linear interpolation technique. Typically, 200-400 iterations are enough to get a good result, depending on the number of segments and the desired precision. In image sequences, initialization can be based on the previous frame, speeding up convergence.

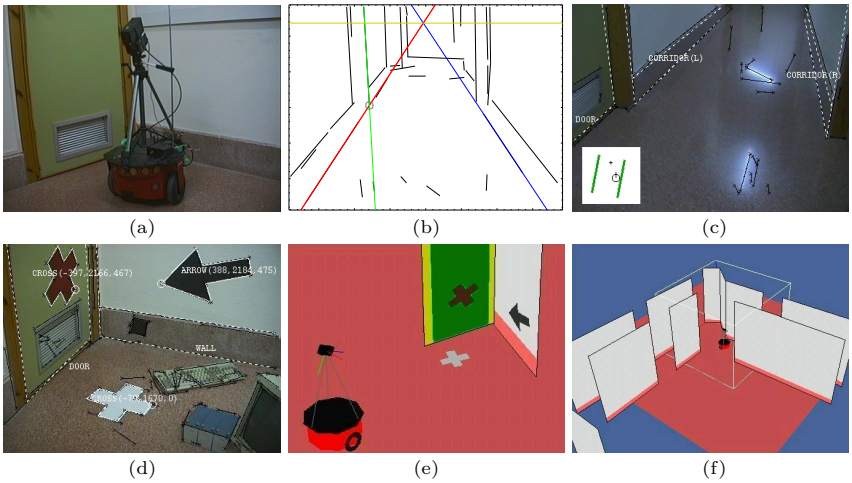
Fig. 3e shows the results of the recovery procedure on the image shown in Fig. 3a. The original, uncompressed RGB image size is $288 \times 384 \times 3 = 324$ KB. The compressed information is formed by 261 segments, each labelled with two RGB values. A segment extreme can be coded using $\lceil \log_2(288 \times 384) \rceil = 17$ bits, and a RGB value using 24 bits. Therefore, the whole set of segments will use $261 \times ((2 \times 17) + 2 \times 24) = 21402$ bits = 2.61 KB, i.e. a 99.2% compression factor. The reconstructed image is essentially equivalent to the original one in terms of medium and large scale color and geometric structure. For comparison, Fig. 3f shows a JPEG image with the same compression ratio.

Of course, explicit recovery of the image is rarely needed. It is shown here only to illustrate the expressive power of the representation. For many applications, once the colored segments have been obtained, the original image is not needed for any subsequent processing. This can be exploited if we want to send the sequence of compressed data through a low bandwidth network, for example.

Segment tracking: Reliable matching and tracking of features (points or segments) is essential to carry out 3D reconstructions from multiple views of the scene. Some tracking methods only take into account the position of the feature in successive images [16]. But tracking is more robust if the local neighborhood of the feature is also taken into account. For isolated points, one of the best methods is to compare local neighborhoods of candidates through intensity correlation, perhaps taking into account a previous warping [15]. For segments, this kind of technique can be applied to individual points along them. For example, in [14] the epipolar restriction is used to obtain corresponding points in candidate matching segments, and a correlation measure is computed for the pixels paired this way. But this method needs robust estimates of the involved fundamental matrices [8], which is computationally hard for real-time operation.

We propose to simplify the characterisation of the feature, making it adequate for faster processing, as needed by *on-line* applications such as robot navigation. The median RGB values on the sides of each segment can be used to compute a similarity measure between candidate pairs of segments, taking into account both geometric distance and photometric information. Hence, tracking becomes more robust without reducing the speed of the system.

Segment classification: The obtained median color values could also be useful in interpretation. For example, if we search for objects with a characteristic, *a priori* known color, we can use the color information of each segment in order to directly associate it with a real world object.



(a) (b) (c)

(d) (e) (f)

Fig. 4. Color segments for robot navigation: (a) *GeoBot*. (b) Calibration. (c) Typical situation (corridor). (d) Walls, doors, and signs detection in the presence of clutter. (e) 3D reconstruction corresponding to (d). (f) 3D reconstruction of the environment during a visually guided walk.

Contour following: Finally, even if we do not know *a priori* the color of an object, we can still find closed contours by looking for segments with nearby extremes and similar colors, grouping and sorting them without being disturbed by other segments in the vicinity that have different colors.

4 Colored Segments for Visual Robot Navigation

Taking advantage of the above properties, the algorithm described in this paper is being successfully used in a mobile robot which builds and interprets structured 3D world representations in real-time from pure visual information. Fig. 4a shows *GeoBot*, the mobile platform in which we are currently developing our research. *GeoBot* is a Pioneer AT equipped with a monocular visual sensor (a Mitsubishi 400-E camera) and an on-board Pentium 200 MMX PC.

The first application of the colored segment extractor is the autocalibration of the robot-camera system. To accomplish this task, the height of the horizon is first determined by detecting two parallel floor lines. The camera intrinsic and extrinsic (robot-relative) parameters are then easily obtained by tracking just one line and one point in the scene during a controlled movement. These minimal requirements for a full calibration of the system (position and orientation of the camera with respect to the robot, and focal length) are possible due to availability of odometric (egomotion) information, and a few realistic simplifications on the camera model [6]. Fig. 4b shows how, using the extracted segments, the system finds the horizon and the line and point to track in order to perform the calibration. Color is essential to interpret the segments as belonging to the floor or a door, and to robustly track these segments during the movement.

Using the estimated camera parameters, and following reasonable clues obtained from the geometric and color information of the segments, *GeoBot* can detect relevant planes (floor, walls, doors) in indoor environments. This information is used to categorize different high level typical situations (rooms, corridors, corners, and so on), that allow the robot to exhibit a non-reactive behaviour, depending on abstract properties of the environment. Fig. 4c shows one of these situations, when the robot is entering a corridor. Observe how a door and the left and right walls are located, while the spurious segments caused by reflections are adequately rejected. A calibrated sensor allows for the metric rectification of the relevant planes of the scene (the floor and the walls, in this case). The system is then able to construct an abstract interpretation of the scene (see the bottom left part of the image) to give the adequate control orders to navigate through the corridor.

The interpretation is extended to handle obstacles and external signs that can also guide the navigation. Fig. 4d shows an image in which several contours of different colors are located and recognized as signs (arrows and crosses), without being disturbed by other objects, that are simply categorized as unknown obstacles. The contour following procedure outlined in the previous section is used for this purpose. The exact position (in mm) of the interpreted signs has been superimposed in the original image for clarity. The signs can be metrically rectified, again using the calibration information, by determining the plane of the image (wall, floor or door) in which it is contained. Thus, the sign interpretation procedure does not have to cope with projective deformations. Rather, simpler similarity-invariant sign classification is performed once the metric rectification of the contour has been accomplished. Fig. 4e shows the 3D reconstruction of the interpreted elements in the scene, without any scale ambiguity.

Finally, the robot is able to accumulate the reconstruction of the visited environment as it navigates, using odometry and tracking. Fig. 4f shows a 3D reconstruction of a large portion of a building, as interpreted by *GeoBot* during a visually guided walk.

Though *GeoBot*'s processor has become somewhat obsolete, our efficient implementation of the segment extractor still allows for an overall 5 Hz perception-action cycle, including calibration, reconstruction, interpretation, and motion control. This is enough to robustly navigate at $\sim$20 cm/s.

We conclude the discussion with an example of how the compression capabilities of the colored segments can also be exploited in autonomous agents. *GeoBot* is equipped with an on-board PC connected to the outside world through a Radio Ethernet interface. This kind of interfaces are usually very slow (1 Mb/s bandwidth in our case). This is not enough to send images of moderate sizes at video rates of 25 fps. But, if segments are extracted in the robot PC, and then sent through the network, the only bottleneck is the local CPU speed, not the radio interface. The rest of the processing can be performed in an external, more powerful and unloaded computer, from which the control orders are sent back to the robot. This way, we save CPU time for possible additional tasks.

5 Conclusions

We have described an efficient image representation algorithm for visual geometry applications. It works by detecting relevant segments that are then labelled with color information. Images can be compressed down to a $\sim$1% of the original size, while capturing the essential structure of the scene: the original image can be easily recovered by means of a simple diffusion procedure. The compression factor is interesting by itself, but the main advantages of the algorithm are computational efficiency and the convenience of colored segments for further geometric processing. Finally, we have shown several applications in real-time vision tasks, such as robot navigation and environment reconstruction and interpretation in indoor scenarios. Nevertheless, the procedure is not limited to this field, and can be successfully used in many other computer vision domains.

References

1. M. Aste and M. Boninsegna. A fast straight line extractor for vision-guided robot navigation. Technical Report I-38050, Ist. Ric. Sci. e Tecn., Trento (Italy), 1993.
2. J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color vision for interactive robots. In *Proceedings of the Intelligent Robots and Systems*, 2000.
3. S.D. Buluswar and B.A. Draper. Color machine vision for autonomous vehicles. *Engineering Applications of Artificial Intelligence*, 11(2):245–256, 1998.
4. J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
5. A.J. Davison and N. Kita. Sequential localisation and map-building for real-time computer vision and robotics. *Robotics and Auton. Systems*, 36(4):171–183, 2001.
6. P.E. López de Teruel and A. Ruiz. Closed form self-calibration from minimal visual information and odometry, 2003. (*Submitted*).
7. J.D. Foley, A. Van Dam, S. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.
8. R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.
9. Intel Corporation. The open source computer vision library (OpenCV) homepage, 2002. `http://www.intel.com/research/mrl/research/opencv/`.
10. V.F. Leavers. Survey: Which hough transform? *Computer Vision, Graphics, and Image Processing: Image Understanding*, 58:250–264, 1993.
11. J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic Hough transform. *Comp. Vis. & Im. Und.*, 78:119–137, 2000.
12. S.J. McKenna, Y. Raja, and S. Gong. Tracking colour objects using adaptive mixture models. *Image and Vision Computing*, 17:223–229, 1999.
13. P.L. Rosin. Techniques for assesing polygonal approximation of curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:659–666, 1997.
14. C. Schmid and A. Zisserman. Automatic line matching across views. In *Proceedings of the Computer Vision and Pattern Recognition*, pages 666–671, 1997.
15. T. Tommasini, A. Fusiello, E. Trucco, and V. Roberto. Making good features to track better. In *Proceedings of the Comp. Vis. and Pattern Recognition*, 1998.
16. Z. Zhang. Token tracking in a cluttered scene. *International Journal of Image and Vision Computing*, 12(2):110–120, 1994.