Deriving and matching image fingerprint sequences for mobile robot localization

Pierre Lamon[1], Illah Nourbakhsh[2], Björn Jensen[1] and Roland Siegwart[1]

[1]Swiss Federal Institute of Technology, Lausanne (EPFL)
[2]The Robotics Institute, Carnegie Mellon University (CMU)

Pierre.Lamon@epfl.ch, illah@ri.cmu.edu, Bjoern.Jensen@epfl.ch, Roland.Siegwart@epfl.ch

ABSTRACT

This paper proposes a method for creating unique identifiers, called fingerprint sequences, for visually distinct locations by recovering statistically significant features in panoramic color images. Fingerprint sequences are expressive enough for mobile robot localization, as demonstrated using a minimum energy sequence-matching algorithm that is described. Empirical results in two different places demonstrate the reliability of the system for global localization on a Nomad Scout mobile robot.

1. INTRODUCTION

Vision-based localization has recently witnessed a newfound popularity. The CCD Camera is a popular choice for mobile robot sensing because it is not inherently dependent on environmental geometry like ranging devices [13]. Therefore, it is hoped that a transition to indoor and outdoor navigation will be more straightforward with vision despite that each of them has their proper challenges.

Simple ranging devices require integration over time and high-level reasoning to accomplish localization. In contrast, vision has the potential to provide enough information to uniquely identify the robot's position.

Recent vision-based navigation methods have overcome the challenges of vision to produce mobile robots that can track their position using only a CCD camera. Some of the successful work is currently limited to indoor navigation because of its dependence on ceiling features [4, 15], room geometry, or artificial landmark placement [16]. Other means for visual localization are applicable both indoors and outdoors, however they are designed to collect image statistics while foregoing recognition of specific scene features, or landmarks [3, 6].

This research aims to create a visual localization system based on recognition of sets of visual features. Our goal is to implement a system with a minimal number of implicit assumptions regarding the environment, such that the system may be directly applicable both outdoors and indoors.

2. THE FINGERPRINT SEQUENCE

As the fingerprints of a person are unique, so each location has its own unique visual characteristics (save in pathological circumstances). The thesis of this localization system is that a unique virtual *fingerprint* of the current location can be created and that the sequence generation methods can be made insensitive to small changes in robot position. If locations are denoted by unique fingerprints in this manner, then the actual location of a mobile robot may be recovered by constructing a fingerprint using its current view and comparing this test fingerprint to its database of known fingerprints.

2.1 Fingerprint sequence encoding

We propose to create a fingerprint by assuming that a set of feature extractors can identify significant features in the image. Furthermore, we use a 360 degrees panoramic image because the orientation as well as the position of the robot may not be known *a priori*.

We define a fingerprint as a circular list of features, where the ordering of the set matches the relative ordering of the features in the panoramic image. In order to encode efficiently this circular list, we denote the fingerprint sequence using a list of characters, where each character represents the instance of a specific feature type.

Although any number of feature detectors may be used in an implementation of our system, we have used only two in our implementation thus far: a vertical edge detector and a color patch detector. We use the letter 'v' to characterize a vertical edge and the letters A,B,C,...,P to represent hue bins as detected by the color patch detector (See Fig. 6,7 and 13).

2.2 Extraction of edges and color features

Edge detection

Edge features are of particular value in artificial environments such as indoor office buildings. For these reasons, they have been popular throughout prior work in vision-based localization [1]. Like other researchers, we have chosen to concentrate on vertical edges because of the instability and rarity of horizontal edges due to projection effects.

Because we use a color CCD camera, the channel used to compute the gradient must be chosen carefully. Knowing that the blue channel of such a camera has a remarkably higher noise level than the other channels, we

use only the sum of the red and green in order to increase the signal/noise ratio.

Histogram based edge detection

From the gradient image several methods are used to extract edges. One of them consists of the application of a threshold function on the gradient values followed by the application of a non-maxima suppression algorithm [10]. The most difficult step then remains, which is to group the resulting edges fragments together in order to obtain true vertical edges. This problem is further exacerbating when luminosity changes along the segment.

To group the resulting edge fragments together, first, we construct a histogram by adding the red-green gradient intensity of every pixel in the same column. To avoid the apparition of parasite peaks due to the noise, we apply a window filter {1,2,3,2,1} on the raw histogram. Its triangle shape permits to keep the peakiness of the spikes.

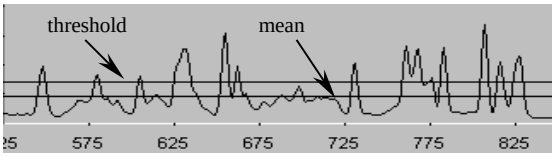


Figure 1: Filtered gradient histogram (See Fig 3.)
(mean and threshold)

One can see on the Fig. 1 that the mean value is actually the level of the noise and provides a bad threshold value. One can compute a more noise insensitive threshold by computing the value t = mean + (max – mean) / c, where c is chosen depending on the number of edges desired. This method is unfortunately very sensitive to occlusion and distance. Indeed, a large peak will provide a big value and the threshold will be high. In such a case, the majority of edges will not be considered.

To solve this problem we use a more statistical approach to choosing the edge threshold. The standard deviation of the values of the histogram is computed and added to the mean in order to fix the base threshold. All edges below the threshold are ignored.

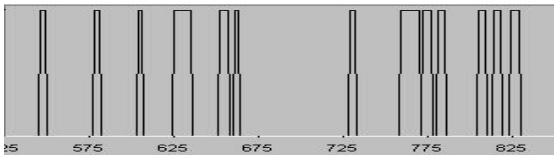


Figure 2: Histogram after group and filter algorithms

Figure 3: Extracted edges (13)

Color patches detection

Color patches can be used for localization as well especially in human environments where one finds often saturated colors. The combination of both edges and patches greatly increases the information for the location. A part of the information is coded in the nature of the features (edge or different colors) and another part in the sequence (order of features).

In order to get more intuitive and natural color representation, we convert RGB images extracted from the camera into the HSI color space (Hue, Saturation and Intensity).

Figure 4: Original image

Because color information is weak for low level of saturation, only high-saturated pixels are considered for the extraction of patches.

Fuzzy voting scheme

The colors in the scene are not known in advance and can cover the entire color space. In order to reduce the quantity of different color patches and memory space similar colors are grouped together considering their hue.

To limit discontinuities and instabilities for pixels near the borders of the intervals, fuzzy sets have been introduced as depicted in Fig. 5.

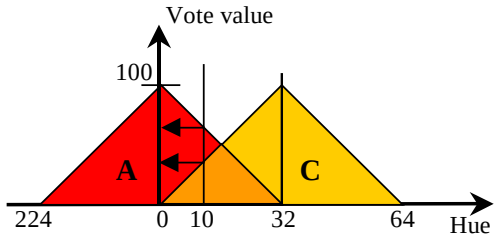


Figure 5: Fuzzy voting scheme

The column histogram for each base color is generated as follows. Each pixel in the image (those that remain after saturation thresholding) will add a value in one or two histograms depending on the hue. For example, a pixel with hue 0 will add 100 in the corresponding column of the red histogram. A pixel with hue 10 will add a bigger value in the red histogram than in the yellow one (see Fig.5).

The same method as described for edge detection is applied but some parameters change. A different window filter, {1,2,2,2,1} is used for color histograms because we want to smooth thin peaks in this case.

The base threshold is also built by adding sigma and the mean of the histogram.

As we can see in Fig. 6 more patches than expected have been extracted from image in Fig. 4. In order to avoid inversion between patches, which can change considerably the resulting string, a color fusion step has been introduced. Intermediate colors are used for the new patch and its horizontal coordinate is the mean of the coordinates of the parent's patches[1].

Figure 6: String before and after color fusion

3. FINGERPRINT SEQUENCE MATCHING FOR LOCALIZATION

To introduce the problem of string matching, let us consider the example below. The first string has been extracted from the current location of the robot and the next two strings are strings from the database.

Place x: vvBEvvCvvvMvOBvvvvv

Place 1: vvBEvMvCvvvMvMOBvvvv

Place 2: LvLvvvBvvOLvBEvOvvv

Figure 7: Strings example

As one can see the new string does not match exactly either of the others because the robot is not exactly located on a map point and/or some change in the environment occurred. Now what sequence match scoring method should we use to determine that the match is Place1 in this case and not Place2 with high confidence?

Great many string-matching algorithms can be found in the literature. Exact string matching algorithms [8] are not applicable in this case. They are designed to indicate if text occurrences are found within a text and are optimized to be very fast.

More elaborate string matching [7,9] algorithms allow a level of mismatch, such as k-mismatch matching algorithms, and string matching with k differences. The first allows matches where up to k characters in the pattern do not match the text, and the second requires that the pattern have an edit-distance from the text of k or less.

Another approach consists in considering strings as digital signals and computing the correlation. A measure of similarity will be in this case the height of the maximum peak of the correlation function. But this method works well only if initial strings have a similar length and fail in case of occlusion and addition. The same problem appears when one computes the SSD (Sum of Square Difference) between two strings.

One of the main problems of the above methods is that they do not consider the nature of features and specific mismatches. We wish to consider the likelihood of specific types of mismatch errors. For instance confusing a red patch with a blue path is more egregious than confusing the red patch with a yellow patch. Furthermore the standard algorithms are quite sensitive to insertion and deletion errors which cause the string lengths to vary significantly.

3.1 Minimum energy algorithm

The approach we have adopted for sequence matching is inspired by the minimum energy algorithm used in stereo-vision for finding pixels in two images that correspond to the same point of a scene [11]. As in the minimum energy case, the problem can be seen as an optimization problem, where the goal is to find the path that spends the minimum energy to go from the beginning to the end of the first sequence considering the values of the second one. The similarity between two sequences is given by the resulting minimum energy of traversal. Value 0 is used to describe a perfect match (e.g. self-similarity).

We describe our sequence matching algorithm using an example consisting of two particular sequences: "EvHBvKvGA" (length n = 9) and "EBCAvKKv" (length m = 8).

Initialization

First the initial n x m matrix must be built. The characters of the first string represent the rows and those of the second string the columns. Because the algorithm is not symmetric, the longest string will always represent the rows. To initialize this matrix only two parameters are needed. The first parameter is a number that represents the maximum mismatch value and the second is used to fix the minimum mismatch value between two different colors. In this particular example Max_init = 20 and Min_col = 5.

Init	E	B	C	A	v	K	K	v
E	**0**	11	8	14	20	20	20	20
v	20	**20**	20	20	0	20	20	0
H	11	20	**17**	17	20	11	11	20
B	11	0	5	**5**	20	11	11	20
v	20	20	20	20	**0**	20	20	0
K	20	11	14	8	20	**0**	0	20
v	20	20	20	20	0	20	**20**	0
G	8	17	14	20	20	14	14	**20**
A	14	5	8	0	20	8	8	20

Figure 8: Init matrix

If the corresponding features are of wholly different types (e.g. a color and an edge) then the corresponding matrix element is initialized to Max_init. If both features are vertical edges or represent exactly the same color the value 0 is used to describe a perfect match. If the comparison is between two colors, then the error is calculated according to the hue distance between the two

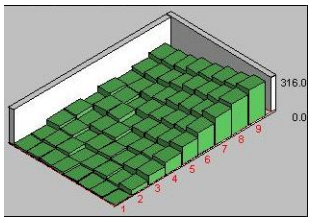
[1] The colors are fused if the difference between the pixel coordinates is less than 10 pixels

colors, adjusted to inhabit the range from Min_col to Max_init.

Although a type-mismatch can be generally assigned a score of Max_Init, any newly introduced feature type must not only include the appropriate feature detector but also a mismatch table, identifying the score for various feature value comparisons within that feature type. This is an important aspect of the present work. We have noted that differences in illumination cause color, for instance, to change one bin at times, but rarely will a color change two or more bins. Therefore, some proportionality of the scoring function based on a distance measure between colors is critical to the success of our method.

Cost	E	B	C	A	v	K	K	v
E	**0**	**11**	**8**	14	20	20	20	20
v	44	**20**	**31**	28	14	40	40	20
H	79	64	37	48	48	25	45	60
B	114	79	66	42	68	59	36	65
v	158	123	99	86	42	82	79	36
K	202	158	137	107	86	42	62	80
v	246	202	178	151	107	86	62	62
G	278	243	216	195	151	121	100	82
A	316	272	248	216	195	153	129	120

Figure 9: Cost matrix (3D)



Neig	E	B	C	A	v	K	K	v
E	-	-	-	-	-	-	-	-
v	1	**1**	2	3	4	5	6	7
H	1	2	**2**	3	4	5	5	7
B	1	1	3	**3**	4	5	6	7
v	1	2	2	4	**4**	4	6	7
K	1	2	3	3	5	**5**	5	8
v	1	2	2	4	4	6	**6**	7
G	1	2	3	4	5	5	**7**	7
A	1	2	3	3	5	6	6	**7**

Figure 10: Cost matrix (3D) and Neig matrix

Only two parameters are needed to compute the Cost matrix: the slope penalty (Slope_pen = 10) and the occlusion penalty (Occ_pen = 24). The first line of the Cost matrix is just a copy of the first line of the Init matrix. Let us consider the cell Cost(2,3)[2] to explain the approach adopted to initialize the other elements.

- Cost(1,1): The slope between cell(1,1) and cell(2,3) is computed by subtracting the respective column indexes and the following sum is evaluated.
  S_1 = Cost(1,1) + Init(2,3) + slope(2) * Slope_pen= 40

- Cost(1,2): In this case the slope between cell(1,2) and cell(2,3) is optimal. Indeed, if the two strings were identical the best path will be the diagonal of the matrix and the result of the match must be 0. That means that no penalty is added.
  S_2 = Cost(1,2) + Init(2,3) = 31

- Cost(1,3): The slope is 0 that means that v is occluded by E. This is a vertical occlusion and the following equation is used.
  S_3 = Cost(1,3) + Init(2,3) + Occ_pen = 52

- Cost (2,2): This time the relative position of cell(2,2) and cell(2,3) represent an horizontal occlusion. The following equation is used.
  S_4 = Cost(2,2) + Init(2,3) + Occ_pen = 64

Finally the minimum value S2 is assigned to Cost(2,3) and the coordinates of the cell (1,2) are stored in Neig(2,3) = 2 (See Fig 10). In case of horizontal occlusion we put a negative sign for the neighbor coordinates.

The best path

The minimum value of the last line of the Cost matrix. This value corresponds inversely to the similarity between the two input sequences. In this particular example the score that results is 381. In order to normalize the result this value is then divided by the worst value that can be obtained with two strings of similar length (in this case, result of the match between a string composed of m edges and one with n colors).

4. IMPLEMENTATION

The camera used to acquire the images is an inexpensive CCD color camera with a 640 x 480 resolution[3]. The interface to the computer is via the USB. Image manipulation is performed with a Microsoft Visual C++ 6.0 application running under Windows'98. The camera is fixed via a 110-CM mast to a Nomad Scout mobile robot research platform[4]. To build the panoramic view of the scene the differential-drive Scout is rotated about its center while a series of 12 images are grabbed from the CCD camera every 30 degrees.

Figure 11: System

Building the panoramic image

Various methods exist to align corresponding pixels in two adjacent pictures. One method consists of computing the SSD between adjacent images and the best alignment is given by the minimum of the function [14].

This method produces panoramas that are of high quality for human consumption; however, such exact alignment is unnecessary for our purposes of color patch and edge extraction. Instead, we simply attach images end to end, taking into account the resulting "seam" by suppressing detection of edges at these seams. To avoid the additional

[2] Cost(i,j) is the value at the ith line and jth column of the Cost matrix. Same for Init(i,j) and Neig(i,j)

[3] Logitech QuickCam Pro. Look at www.logitech.com

[4] More information available at www.robots.com/nscout.htm

computational burden of unwarping images, only the central 70% percent of the images is used during construction of the image.

The point of view of the panoramic is very important and the height of the camera must be chosen carefully. If the camera is too low every item of furniture such as chairs and tables can occupy the view in front of the robot. Since these low objects are apt to move, the resulting image will be highly dynamic. In our implementation we have placed the camera at almost the same height as the eyes of an human so that large-scale features of interest (e.g. door posts, windows, corners) are easily visible while low-level clutter is avoided.

5 EXPERIMENTAL RESULTS

In order to test the system, two maps corresponding to two different environments have been constructed (See Fig. 12). The left map, called White Hall, corresponds to the entrance hall on the first floor of the Smith Hall building. The map on the right, called Ground Floor, covers a path that ends in the conference room on the ground floor in the same building. For the White Hall 15 locations evenly spaced by 90 cm have been chosen arbitrarily in the map in order to represent the *map points*. 21 *map points* have been stored with the same method for the Ground Floor. We use crosses to represent those points in the next figures.

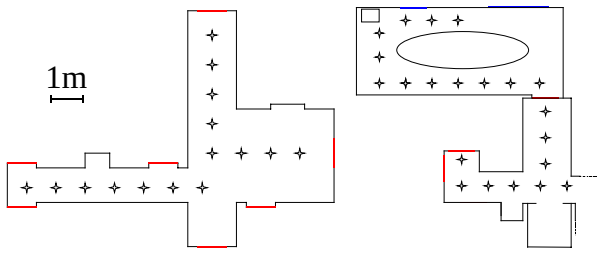


Figure 12: Maps of White Hall and Ground Floor

Fig. 13 shows panoramas and strings associated to map points Pc and Pw3 (See Fig. 14). It is interesting to note that same objects in the scene generate same string fragments even if locations are quite far one from each other. For example, the same sequence "LvBE" has been extracted for the trashcan (blue) and the door (red, green) for both panoramas.

We intend to test global localization by choosing random positions around the map points and compare the corresponding strings with all the stored map points. For the White Hall 18 locations have been chosen to test the system: they are called *test* points and are represented by circles in the maps. For the Ground Floor 22 points have been tested.

In order to determine a percentage of good results the two following criteria have been chosen.

1. A test point is considered as *topologically correct* if the best match among the map points is a point adjacent to the test point. Example: Pe1~ must give Pe1 or Pe2.

2. A test point is considered as *geometrically correct* if the best match among the map points is the closest map point. Example: Pe1~ must provide Pe1 and Pe1~~ must provide Pe2.

For the White Hall 17 test points have been classified as topologically correct that represents 94% of good results. In another hand 82% of locations have been classified as geometrically correct (14 points). 20 test points are topologically correct for the Ground Floor (91%) and 14 have been classified geometrically correct (70%).

In order to get more significant statistics the two experiment sets have been fused. The new database consists as 40 test points and 36 map points. 90% of the test points have been classified as topologically corrects and 75% as geometrically corrects. The test points, which were wrong for the first test sets, remain wrong when databases are fused. Unfortunately, the fusion of the two sets has generated a new false point (Pw3~).

These results make us think about some considerations. The wrong test points are mainly due to two different effects.

First, major occlusions and/or additions can occur in the string. These defaults are generally due to a pathological combination of dynamic changes in the environment e.g. illumination change, reflections, new objects or persons in the scene.

Figure 13: Panoramas and string examples

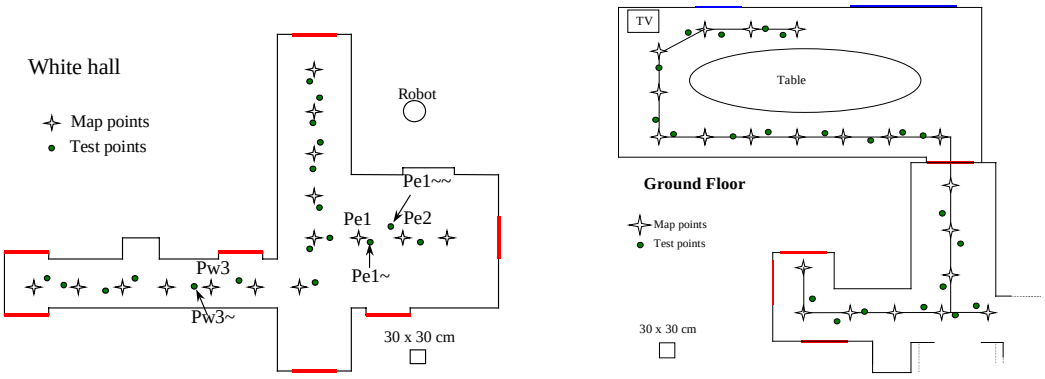


Figure 14: Maps of White Hall and Ground Floor

Second, some locations are locally so unique that they are different of all test points even if they are geographically close. Those pathological cases happen mostly in close areas and for points close to object that can hide a big portion of the environment. Indeed, the displacement/(changes in the string) ratio can be very small in these cases. This explains the relative bad results for the second criteria compared to the first.

This problem makes us think about the necessity to choose carefully the map points. The natural rule is to put more map points when objects are close and fewer points are necessary for open areas. This can be done automatically while the robot is exploring the scene.

6. CONCLUSIONS

The structure of circular chains and the string matching algorithm allows us to insert other kinds of features. Using different features extracted from several kinds of sensors provides several advantages. One can improve the edge detection by fusing information from the camera and a laser range finder for instance. Or, infrared images and laser range finder can be used in dark scenes. Furthermore probabilities related to features can be easily introduced in the string matching algorithm.

For the moment the largest computational burden is construction of the panoramic image. Optical solutions can alleviate this problem, and so one should consider using a panoramic vision system, such as an Omnicam, to capture a panorama instantly.

ACKNOWLEDGEMENTS

Thanks to Iwan Ulrich who provided much help during the project and Jianbo Shi for his panoramic algorithm and good advice.

REFERENCES

[1] Jennings, C., Murray, D., and Little, J.J., “Cooperative Robot Localization with Vision-based Mapping”, IEEE International Conf. on Robotics and Automation, May 1999, pp. 2659-2665

[2] Asensio, J.R., Montiel, J.M.M., and Montano, L., “Goal Directed Reactive Robot Navigation with Relocation Using Laser and Vision”, IEEE International Conf. on Robotics and Automation, May 1999, pp. 2905-2910.

[3] Thrun, S., “Finding Landmarks for Mobile Robot Navigation”, IEEE International Conf. on Robotics and Automation, May 1998, pp. 958-963.

[4] Abe, Y., Shikano, M.,Fukada, T., and Tanaka, Y., “Vision Based Navigation System for Autonomous Mobile Robot with Global Matching”, IEEE International Conf. on Robotics and Automation, May 1999, pp. 1299-1304.

[5] Kunz, C., Willeke, T., and Nourbakhsh, I.R., “Automatic Mapping of Dynamic Office Environments”, Robotics and Autonomous Systems, Autonomous Robots Journal. 7(2) 1999.

[6] Ulrich, I., and Nourbakhsh, I.R., “Appearance-Based Place Recognition for Topological Localization”, Robotics and autonomous systems, Carnegie Mellon University (CMU), Pittsburgh, December 1999.

[7] Baeza-Yates, R., Navarro, G., “Faster Approximate String Matching”, Department of Computer Science, University of Chile, Santiago.

[8] Knuth, D.E., Morris J.H., Pratt V.R., “Fast Pattern Matching in Strings”, SIAM Journal on Computing Volume 6 Number 2 pages 323-350, June 1977.

[9] Alfred V. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, chapter 5, pages 254-300. Elsevier Science Publishers B. V., 1990.

[10] Nicola Tomatis, 'Vision Feedback for Mobile Robots', Diploma Thesis, Institute of Robotics (IfR), Swiss Federal Institute of Technology (ETH) Zurich, February 1998.

[11] Kanade, T., Ohta, Y., “Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming”, IEEE Transactions on pattern analysis and machine intelligence, Vol PALMZ No 3, March 1985

[12] Li-S, Tsuji-S, “Qualitative representation of scenes along route”, vol 17, no 9, July 1999 pp. 685-700

[13] Pérez, J.A., Castellanos, J.A., Montiel, J.M.M., Neira, J., and Tardós, J.D., “Continuous Mobile Robot Localization: Vision vs. Laser”, IEEE International Conf. on Robotics and Automation, May 1999, pp. 2917-2923.

[14] Shi, J., Tomasi, C., "Good Features to Track", IEEE Conf. on Computer Vision and Pattern Recognition, 1994, pp. 593-600.

[15] Thrun, S., Bennewitz, M., Burgard, W., Cremers, A.B., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C., Roy, N., Schulte, J., Schulz, D., “MINERVA: a second-generation museum tour-guide robot”, IEEE International Conf. on Robotics and Automation (Cat. No.99CH36288C) 1999.

[16] Nourbakhsh, I.R., Bobenage, J., Grange, S., Lutz, R., Meyer, R., and Soto, A., “An Affective Mobile Educator with a Full-time Job", Artificial Intelligence, 114 (1-2), pp. 95-124. 1999.

[17] Dellaert, F., Burgard, W., Fox, D., Thrun, S., “Using the CONDENSATION algorithm for robust, vision-based mobile robot localization”, IEEE Computer Society Conf. on Computer Vision and Pattern Recognition 1999

[18] Nourbakhsh, I., Powers, R., Birchfield, S., “DERVISH: an office-navigating robot”, AI Magazine, vol.16, no.2, p. 53-60, sum 1995

[19] Maeda-S, Kuno-Y, Shirai-Y, “Active navigation vision based on eigenspace analysis”, IROS '97 Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems, vol 2, p.1018-23