An Ordered List Approach to Real-Time Line Detection Based on the Hough Transform

Chris Madden and Robert Mahony
Faculty of Engineering and Information Technology,
Australian National University
ACT 0200.
Robert.Mahony@anu.edu.au

Abstract

This paper presents preliminary results for real-time image processing of video frame rate data. The goal is to identify multiple linear features from video data derived from a camera mounted on a small UAV. The algorithm proposed in this paper adapts the well-known Hough Transform to use sequential image sampling and an ordered list to reduce the computational time and memory allocation required. The results indicate that for images with few lines this algorithm should be able to achieve speeds of better than 25Hz on current desktop computing equipment.

1 Introduction

Visual control of dynamic systems in a real world environment requires very fast processing of visual data. Continuing advances in computational power of affordable processor chips provides a basis for a realistic investigation into real-time visual control of such systems. Early applications in real-time vision processing were based on identification of image 'blob' centroids [Andersson, 1988]. Dealing with more complex image features requires more sophisticated algorithms. In this paper, we consider the question of identifying linear features in real-time, however, the conceptual base of the algorithm we propose is aimed at more complex features. There are three distinct stages in real-time line detection. The first stage is the image capture operation, where the video data is digitised and stored in memory. The second stage concerns identifying pixels that are associated with lines in the image – either as pixels in the line or as edge pixels. The Third stage is processing the pixel data to find the parameters of all the lines in the image. Elapsed time for the image capture is hardware dependent. Identifying image pixels is a complex process in itself for a real-world image. Contemporary work in this direction involves tracking colour and brightness changes in the image sequence [Austin and Barnes, 2003]. By using lines that have a distinct contrast with the background, the pixel identification problem is reduced to a constant thresholding process [Gonzalez and Woods, 1992, Bruce et al., 2000].

The final step in feature identification involves obtaining the parameters of the line. The most widely used technique is the Hough transform [Hough, 1962, Illingworth and Kittler, 1988] because of its robustness, and its accuracy. The Hough transform is particularly of interest as it does not apply purely to line features, but can be used equally well for arbitrary features [Ballard, 1981]. A disadvantage of the Hough transform is that both the voting process and the subsequent search of Hough accumulator space can be computationally expensive (especially for non-standard features). Much research has investigated increasing the performance of the Hough transform. The Integer Hough Transform [Olmo and Magli, 2001] aims to reduce the number of computations necessary by approximating the Hough parameters using integer based calculations. The Adaptive Hough Transform [Illingworth and Kittler, 1987] aims to reduce the computations required when searching for the lines by first using a broad parameterisation, then narrowing the parameter range to increase the accuracy of the parameters. The Real-time CAM-based Hough Transform [Nakanishi and Ogura, 2000] was implemented using a parallel hardware approach to increase the amount of computational power available to increase the speed of the parameterisation. A Local Line Detection approach [Lefèvre et al., 2002] has also been proposed where the image is broken up into smaller segments to be analysed individually, then integrated together to get an overall picture of the lines in the image. A clear advantage of the Hough transform is that line pixels need not be identified with a given line feature before the algorithm is used to identify the image features. In contrast, alternative methods, such as the linear least squares method [Adams, 1995], or method that compute the image moments of a group of pixels [Shapiro and Stockman, 2001], require generation and testing of a hypothesis. This becomes particularly challenging when there are multiple features present in the image.

In this paper, we consider the problem of extracting multiple line features from real-time video data. The long term goal of the research is to use visual feedback for the control of inexpensive small autonomous aerial robots. The bandwidth of the response of the type of aerial robot considered is expected to be in the order of 3-5 Hz. To apply successful discrete-time feedback it is preferable to sample at least 5 times faster than the system band width [Franklin et al, 1994]. This can just be achieved using the

PAL video standard (25 Hz) as long as the processing can be accomplished at video rate. The algorithms developed must be highly robust to poor quality, inexpensive camera technology. Finally, it is desirably to develop algorithms with low memory as well as computational requirements, since the algorithms may need to be implemented on embedded hardware.

The approach taken in this paper is based on a modification of the Hough transform [Hough, 1962, Illingworth and Kittler, 1988]. The key contributions of the paper address the memory and computational requirements of the Hough transform in two ways. Firstly, a batch sequential search is used for feature pixel identification and correlation. Thus, a given feature pixel will lie in one sample batch of data from the original image and need only be correlated with other feature pixels from that sample. Properly implemented, this approach significantly reduces the computational load of cross correlation of feature pixel information. Secondly, the Hough accumulator space is replaced by a dynamic ordered-list memory array. This significantly reduces the memory requirements of the algorithm, especially for complex features with many parameters, and completely avoids the costly search of Hough accumulator space that the classical algorithm requires. The ordered-list is updated after each sample batch of data is processed, with additional votes for existing features added to their vote count and new features added to the list. The list is reordered during this process so the highest vote features are always the top entries on the list. Unsuccessful features are culled from the bottom of the list to minimize memory usage. The proposed algorithm was developed with linear features in mind, however, the authors stress that the modifications proposed above (with a suitable batch sequential search routine chosen for a given feature class) are equally valid for more general features. The algorithm was implemented in closed-loop with a position-based visual-servo experiment [Hutchinson, et al., 1995]. The experimental platform used was the a Scorbot ER-VII Robotic arm with data provided by a wireless 2.4 GHz Colour CCD Video Camera surveillance camera transmitting via a wireless link. Results indicate that despite the lens distortion of the camera accurate estimates of line features are obtained and the algorithm should be able to achieve speeds of better than 25 Hertz on current desktop computing equipment.

2 Structure of Algorithm

In this section, a modified Hough transform algorithm is proposed. The proposed algorithm takes advantage of the Hough transform voting system, along with the Hough space parameterisation. Two modifications of the Hough transform are proposed, sequential batch sampling of the image prior to cross-correlation of feature pixels, and the use of an ordered-accumulator-list. The sequential batch sampling significantly reduces the number of feature pixel correlations that must be done. The ordered-accumulator-list (OAL) replaces the Hough accumulator array, substantially reducing memory requirements of the algorithm, and eliminates the costly accumulator array search.

Sequential batch sample sets: The classical Hough transform for identification of linear features takes each feature pixel and counts one vote in Hough accumulator space for every line that potentially passes through that point. The robustness of the Hough transform is based on the distribution of valid feature correlations compared to spurious feature correlations in the full correlation process. Thus, a true feature has many valid correlations and accumulates a high vote, while a spurious feature accumulates only a low vote and is discarded in the search of accumulator space. A common modification Hough transform takes all combinations of two feature pixels, computes the line parameters of the unique line that passes between those pixels, and counts one vote for these parameters in accumulator space. This reduces the number of votes made in accumulator space without reducing the number of valid votes made and improves the robustness of the transform. For the standard Hough transform it is necessary to have an efficient algorithm to compute all possible lines that pass through a given pixel [Hough, 1962]. For the modified transform the number of correlations that must be made is n(n-1)/2 where n is the number of feature pixels in the whole image. A key idea in this paper is to batch-sample the full image in such a way that the distribution of valid correlations in the batch sample does not distort the distribution of correlations in the full image. The sequential process in the ordered-accumulator-list update will reinforce correlation of the true features in an analogous manner to the correlation of all image features in the full Hough transform. The generic concept is independent of its application to line features, as long as a suitable batch sampling process can be found. For linear features we propose a batch-sampling of concentric border regions of the image. This choice is well suited to linear features that traverse the entire image. One may equally well imagine a random sampling process, or some other deterministic search routine that is suited to the particular image feature considered.

Ordered-accumulator-list (OAL): The ordered-accumulator-list is proposed to minimize memory requirements and to propagate information between sequential batch samples. The OAL consists of a list of feature parameters along with an accumulated vote for each feature. The features are ordered from the most to least likely to represent a true feature. When image features are correlated in a given batch sample they contribute a vote for a feature parameter. This vote is compared to entries in the OAL (with most likely features considered first) and the vote is either added to an existing feature or a new feature is added to the list. The list is reordered once at the end of each sample batch. The reordering process is normally computationally cheap since the order of the features does not tend to change significantly between iterations. Since the length of the OAL and the number of cross correlations considered in each batch sample is small, this leads to an overall saving in the number of calculations undertaken. A process must be put in place to cull feature parameters from the OAL that do not get regular votes during the sequential iterations to avoid growth of the list. A key aspect of the OAL is that the computationally expensive search of Hough accumulator space is achieved iteratively via the ordering process of the accumulator list. After the final iteration of the sequential batch samples the first few entries of the OAL should have high vote counts and be

clearly differentiated from the other entries on the list.

3 Algorithm Development

In this section, the principals outlined in Section 2 are used in the development of a line search algorithm.

The following assumptions are made regarding the images in the video sequence considered
- The lines remain relatively straight, even with lens distortion.
- The lines traverse the entire image.

Despite the assumptions, it is expected that the algorithm developed will be robust minor non-compliance to both these assumptions.

The six-step structure of the developed algorithm is:

1) Scan the current concentric border of the image to find the middle of any black regions on that border. Such points are assumed to be junctions of line features and are the 'pixel features' used in the feature identification.

2) Correlate the border point pairs to identify all possible lines in the image that could have generated the pixel features observed.

3) Compare the line parameters found to the parameters on the ordered-accumulator list and modify the votes accordingly. Add new list entries for new line features.

4) Reorder the accumulator list based on the vote count of each feature. Renormalise the feature vote counts.

5) Cull line features with low vote counts that have been present for several iterations.

Steps 1-5 are repeated for each batch sample of data, concentric borders of the image.

6) Read off the top line features from the ordered-accumulator-list.

The sequential batch sampling process proposed is concentric border zones of the image. Due to the assumptions on the data, the lines visible in the image should register in the border zone of the image until that line feature is no longer contained in the interior of the scan region. It is clear that the distribution of votes for true verses spurious image features from a single batch sample is essentially uniformly distributed. That is all correlations get a single vote. This is sufficient to avoid distortion of the Hough accumulator count as only the true features will be reinforced in subsequent iterations of the algorithm. Entries in the ordered-accumulator-list consist of r and θ values as well as its vote count and a count of how many iterations have passed since it was first found.

Step 1; is a one dimensional scan to find the points on the line at the current border. The border referred to here is the current scan line. This scan line consists of all the pixels a set distance from the edge. Thus 10 pixels in from the edge is one scan line, whilst 15 pixels in is another.

This gives the concentric borders as shown in *Figure 1*.

Each border is scanned for line pixels using a threshold criteria. This can be done using windows to reduce the number of line pixels found. The preferred option uses the middle of each set of black pixels found as this creates fewer points to parameterise. It is also important to take into account regions of dead pixels, or other effects that might distort the line pixels found.
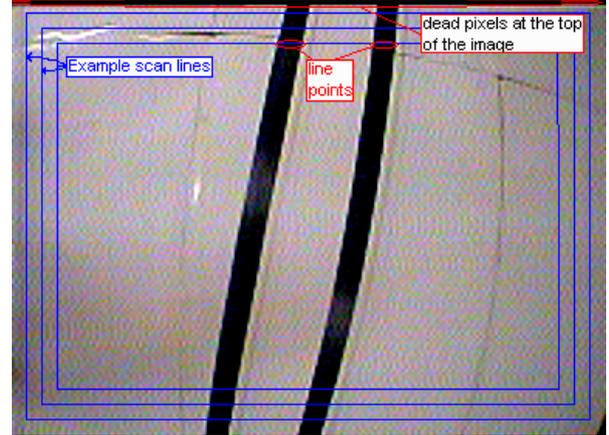

Figure 1: Line detection algorithm scan lines

Step 2; takes the feature pixels found in step 1and connects them to find all the possible lines from that scan. Every combination of two points found in a single scan is correlated and the orientation θ and offset r.

Step 3; compares the lines found in Step 2 to entries in the ordered-accumulator-list (OAL). Parameter thresholds are used to determine if an observed line feature corresponds to an existing entry of the OAL or should be classified as a new entry.

$$|\theta_{new} - \theta_{old}| \leq anglechange \qquad (1)$$
$$|r_{new} - r_{old}| \leq rchange \qquad (2)$$

The parameters a*nglechange* and *rchange* are selected to give the minimum allowable change, whilst still allowing enough variation for the lines to be detected. The principle cause of variation is due to lens distortion. If an observed feature satisfies the threshold conditions then the vote of that entry of the OAL is increased by one. Furthermore, the r and θ values of the entry, weighted by the number of votes prior to this iteration, is averaged with the new feature parameters to obtain averaged r and θ values for the OAL entry. The averaging process is a highly advantageous aspect of the algorithm since it helps compensate for the effects of lens distortion. The outcome of the averaging process is analogous to taking the average value of a local maximum in Hough transform space rather than choosing the median as the optimal value. This aspect of the algorithm improves robustness of the overall algorithm when dealing with approximately linear features whose representation in Hough accumulator space will be spread out. If the new feature does not previously exist in the OAL, then a new entry in the list is created with weighting of 1, and a count of 0.

Step 4; involves incrementing the count on all the lines, and resorting the OAL. The sort is based upon the vote and the count of each entry. This allows newly identified lines a chance to accumulate some weight before they might be discarded. The sort of the OAL is based upon the measure

$$weight - \left(\frac{count}{2} \right) \qquad \qquad (3)$$

Step 5; eliminates those entries in the OAL that are deemed to be spurious. Robustness of the process is important, so as many lines as possible are kept, however every new line that is kept needs to be compared to the old lines, which means more computations are needed. This has lead to the compromise involving the weighting and the count. Each true line should theoretically be found with each successive iteration. Due to noise in the image, as well as other factors including lens distortion, lines are not always found on each iteration. Unlikely lines will not be found very often, and so their weighting will often be much lower than their count, whilst true lines will have a count similar to their weighting. Based upon this assumption lines will only be removed according to**:**

$$weight - \left(\frac{count}{2} \right) < 0 \qquad \qquad (4)$$

Based upon the idea that we don't want to delete lines if we do not need to, a limit of 20 lines (for the particular application considered in Section 4) is also imposed in the algorithm. This is based upon the comparisons needed to determine if a newly found line is similar to an existing one. If there are less than 20 lines identified, then no lines are discarded at all. Thus lines will only be removed at this stage if there are more than 20 lines that have been identified, and they satisfy equation **(4)**.

Steps 1 to 5 are performed iteratively until the current border reaches the centre of the image. At this point the iterations finish and step 6 is performed to identify the best possible lines in the image.

Step 6; finds the best lines from the array based upon weighting. The highest weightings indicate the best possibility that it is a true line. The only modification to this is that occasionally a line can be incorrectly found that is close to a true line. Due to the weighted averaging this line can finish with a similar parameterisation to another line found. Thus the top lines are selected as the best lines, but if any two lines have close parameters, then the one with the largest weighting is selected, and the other is ignored.

4 Experimental Design

There were two sets of experiments conducted as a part of this research. The first experiments were based around verifying the computational performance of the algorithm. The second set of experiments determined the accuracy of the line detection and pose estimation algorithms.

The timing experiments were conducted by running the algorithms in a loop fashion 10 times (100 for the pose estimation). This allows for greater accuracy of determining the algorithm speeds due to the limitations of measuring millisecond timing. The results provide an estimate of the maximum and minimum computational times as well as giving an average expected time. The algorithm is used as described in section 2 for detecting the lines in the image, but an extension of this process is used with a four-line target to conduct pose estimation. Thus a two-line image is used for pure line detection, and a second pose estimation is performed using a four-line target.
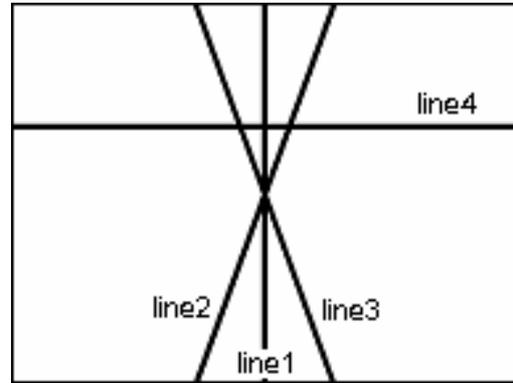

Figure 2: Pose Estimation Target

The pose estimation component of the algorithm is found by solving the Four Point Perspective Problem [Horaud et al., 1989]. The target used consists of four lines as shown in *Figure 2*. The intersection points of the lines are used as the data for the four point algorithm.

The experiment was conducted on a Scorbot ER-VII robotic arm, with an eye in hand camera configuration. Precise mounting provided the accuracy to locate the camera position for the pose estimation, and Position Based Visual Servoing (PBVS) [Hutchinson, et al, 1996] of the camera.
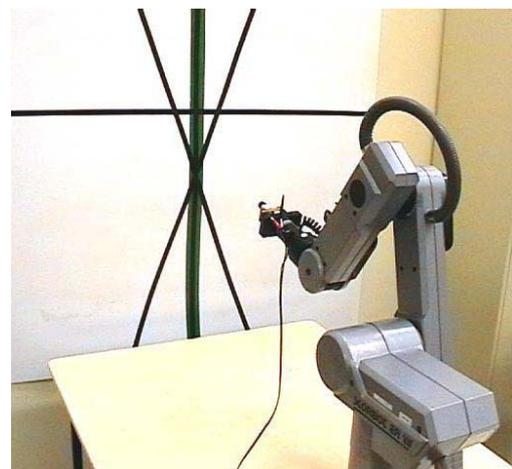

Figure 3: Experimental Configuration

The pose estimation process has the following four step structure:

1) Identify the input lines with respect to the target as lines 1 through 4.

2) Identifies the roll directly from the orientation of line 1.

3) Identify two plane angles using the four-point perspective problem for lines 1 and 2. The intersection points of the line estimations are used for this problem. These equations are solved to give the pitch and yaw angles give two equations, but are of the following form:

$$Planeangle = Pitchangle \sin(\sigma) + Yawangle \cos(\sigma) \quad \textbf{(5)}$$

where σ represents the roll of the line.

4) The final step estimates the position of the camera with respect to the target.

Step 1; receives the lines as they are output from the line finding algorithm. This output is in no particular order, so the line number needs to be found so that the intersection points can be used correctly in the pose estimation.

Step 2; obtains the roll of the camera directly from the orientation of line 1. This step ensures that it is the correct 0-360° rotation rather than the 0-180° representation obtained using the Hough transform.

Step 3; uses the intersection points of the lines as points for use in solving the four point perspective problem. This solution provides a plane angle for lines 1 and 2, which can be used to find the pitch and roll of the camera.

Step 4; uses the final piece of information from the four point perspective problem solution to determine the distance of the camera from the intersection of the three lines in the target. This provides the final piece of pose estimation data.

The pose estimation data is used as the measurement input for position based visual servo (PBVS). The difference in location between the goal and the current locations is estimated from the respective poses, and used as a point and go movement to try and reach the goal [Hutchinson, et al, 1996]. A second iterative approach was also used. Rather than estimating the full displacement, the camera was moved short distance in the correct direction according to:

$$_{C}^{O}T_{k+1} = _{C}^{O}T_{k} \exp\left(\Delta \begin{bmatrix} \Omega & v \\ 0 & 0 \end{bmatrix}\right) \quad \textbf{(6)}$$

for $\Delta \leq 1$ a small positive constant. Here the estimated displacement to goal at point k is

$$_{G}^{C}T_{k} = \exp\begin{pmatrix} \Omega & v \\ 0 & 0 \end{pmatrix}$$

where Ω is a skew matrix and v is a vector velocity. As the current position approaches the goal position, the error in the relative pose estimations converges to zero. This allows the camera pose to approach the goal pose even with estimation errors.

5 Results

The results provided are based upon the MATLAB implementation of the algorithm. Though this environment was suitable for the algorithm development and refinement, it is optimized for vector calculations (not loops or scalar calculations). A direct programming language, such as C, would be a much faster environment to run the program, with an expected 5-15 times decrease in execution times [Mathworks, 2003].

The first test conducted was based upon the accuracy of the algorithm to detect the lines. This test used two different targets; one consisting of two straight parallel lines, and the four-line target shown in *Figure 2, above.*

Target	Repetitions	Times failed	% Failed
Two Line	50	2	4
Four Line	50	16	32

Table 1: Line Detection Results

It is important to note that during these experiments lines that lay near the edge of the image suffer from the largest degree of lens distortion, and was also the most poorly detected lines. These results also showed that the more lines there were in the image, the more easily confused the line detection becomes. This is due to the correcting affect of the lens distortion, allowing lines within a range of the existing line to be counted. If lens distortion were to be corrected using camera calibration, then there would only be a minimal variation allowed, and even with many more lines in the image, the detector would be more accurate.

The second set of results obtained were the timing data for the algorithm. The algorithm was split up into sections to give a better breakdown of times. Because of the short time intervals involved, this timing data was found over a series of 10 (100 for the pose estimation) repetitions of the different sections, and the overall algorithm. This was then used to get the average time for that section.

Section		Minimum Time	Maximum Time	Average time
Read image in RGB format		387.2	429.4	399.8
Convert to a binary format		140.9	144.2	142.7
Find lines	2 line target	107.4	110.1	108.4
	4 line target	156.2	161.2	158.9
Pose detection (4 lines)		9.0	10.0	9.7
Full pose estimation (4 lines)		696.3	727.1	707.3

Table 2: Average algorithm section times in milliseconds

Table 2 gives an indication of the times taken for each algorithm section in the Matlab environment. The important thing to note is that over 50 percent of the time is spent reading the image into Matlab. This step is not necessary in C, as the DIB format is almost exactly the array of data that is useful in C, and that is the format of the acquired image. With most video capture cards there is also the option to obtain greyscale images. Greyscale images would also reduce the conversion to a binary image if large contrasting lines are used, such as in our application.

The section of the algorithm where the lines are actually found is too slow in the Matlab environment to be useful.

As early discussed, this time should drop to the order of 11-22 milliseconds for the two-line target, and approximately 16-30 milliseconds for the four line target when the algorithm is coded in C. With a video sample-rate of 40 milli-seconds, these computational times are within a range to be used considering the time required to capture an image (less than 10 milliseconds). The pose estimation is not required in the overall project of the automated line following UAV. It could be useful for real-time pose estimation for a robotic manipulator because of the 1-3 millisecond processing time required.

The visual servo control results were conducted for relatively small motion due to the hardware limitations. The results show that even with lens distortion the motion performed accurately approaches the goal pos. For the 'point-and-go' algorithm it was found that the lens distortion did significantly impact the number of iterations taken to reach the goal position. The results given in Table 3 used a Δ value of 0.8 so the position rapidly approached the goal.

Position	X (mm)	Y (mm)	Z (mm)	Pitch (°)	Roll (°)
Start	490	15	610	-5	0
1	510	-31.7	586.3	-0.2	0
2	530.5	-34.1	564.7	0	0
3	540.2	-34.1	552.8	0	0
Goal	550	-35	550	0	0

Table 3: PBVS Positioning Results

6 Conclusion

This research has devised an algorithm to help improve the speed of the Hough Transform using sequential image sampling and an ordered accumulator list. The algorithm presented is tuned for identification of linear features, however, the authors believe the general principles used should be applicable to general features. The results obtained indicate that a C implementation of the algorithm should approach the 40 millisecond desired speed. The accuracy obtained in the results is indicative of errors due to the lens distortion effect in the image. Full camera calibration would dramatically improve performance of the algorithm, although this would be against the goals of the work. The PBVS experimental results show that the algorithm provides good measurement data for visual servo control of a robotic manipulator.

References

[Adams, 1995] R. A. Adams, *A Complete Course Calculus, Third Edition,* Addison-Wesley Publishers Limited, 1995. pp 729-735.

[Andersson, 1988] R. A. Andersson, *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control*, MIT Press, Cambridge, MA, USA, 1988.

[Ballard, 1981] D. H. Ballard, *Generalising the Hough Transform to detect Arbitrary Shapes,* Pattern Recog., v. 13(2):111-122.

[Bruce et al., 2000] J Bruce, T Balch, and M Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proc IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS'00)*, vol. 3, pp. 2061-2066, 2000.

[Austin and Barnes, 2003] D. Austin and N. Barnes. Red is the new black: - Or is it., In *Proc. Australian Conf. on Robotics and Automation (ACRA03)*, 2003.

[Franklin et al, 1994] G.F. Franklin and J.D. Powell and A. Emami-Naeini, *Feedback control of dynamic systems*, third edition, Adddison Wesley Longman Inc, Menlo park, CA, USA, 1994.

[Horaud et al., 1989] R. Horaud, B. Conio, O.Leboulleux, and B.Lacolle. An analytical Solution for the Perspective 4-Point Problem. *Computer Vision, Graphics and Image Processing*, Academic Press, 1989.

[Hough, 1962] P. V. C. Hough, *Method and Means for Recognising Complex Patterns*, U.S. Patent 3069654 1962.

[Hutchinson, et al, 1996] S. Hutchinson, G. Hager, P. Corke, *A tutorial on Visual Servo Control*, IEEE Transactions on Robotics And Automation, Vol. 12, No. 5, October 1996.

[Illingworth and Kittler, 1987] J. Illingworth and J. Kittler, *The Adaptive Hough Transform*, IEEE Trans. Pattern Anal. Mach. Intell., Vol. 9, pp. 690-697, 1987.

[Illingworth and Kittler, 1988] J. Illingworth and J. Kittler, *A survey of the hough transform.* Computer Vision, Graphics, and Image Processing, 44, 1988.

[Lefèvre et al., 2002] S. Lefèvre, C. Dixon, C. Jeusse, N. Vincent, *A Local Approach for Fast Line Detection*, RFAI publication: IEEE International Copnference on Digital Signal Processing, Santorini (Greece), July 2002, pp 1109-1112.

[Mathworks, 2003] No author, Mathworks publicity for MATLAB C/C++ compiler, http://web.ccr.jussieu.fr/ccr/Documentation/Calcul/matlab5v11/docs/00002/00278.htm, last visited 17/09/2003.

[Nakanishi and Ogura, 2000] M. Nakanishi, T. Ogura, *Real-time CAM-based Hough Transform Algorithm and its performance evaluation,* Published in Machine Vision and Aplications (2000) 12: 59-68.

[Niku, 2001] S. B. Niku, *Introduction to Robotics Analysis, Systems, Applications,* Prentice Hall Inc, 2001.

[Olmo and Magli, 2001] G. Olmo, E. Magli, *All-Integer Hough Transform Performance Evaluation*, ICIP 2001 - IEEE International Conference on Image Processing, Thessaloniki, Greece, October 2001.

[Russ, 1995] J. C. Russ, *Image Processing Handbook, 2nd Edition*, Boca Raton : CRC Press, 1995.

[Gonzalez and Woods, 1992] R. Gonzalez and R. Woods *Digital Image Processing*, Addison-Wesley Publishing Company, 1992, Chapter 7.

[Shapiro and Stockman, 2001]L.G. Shapiro, G. C. Stockman, *Computer Vision,* Prentice Hall, 2001. Chapter 3.