

Dynamic Segmentation of Small Image Windows for Visual Servoing

Parthipan Siva

Systems Design Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
psiva@uwaterloo.ca

Carol C. W. Hulls

Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
chulls@uwaterloo.ca

Abstract—Processing of image windows rather than complete images is useful for robots incorporating visual servoing as high image processing rates are required. Each image window is segmented, often by thresholding, to identify features of interest. To adapt to changing conditions and to achieve the thresholding of low contrast and shadowed windows, a sophisticated method for performing dynamic segmentation is required. Segmentation methods from the pattern recognition and optical character recognition fields were studied to determine their effectiveness at thresholding 32 by 32 pixel image windows of circular hole features. The segmentation technique must be capable of preserving the centroid location with sub-pixel accuracy. To this end a new morphological preprocessing method is introduced to improve the performance of most thresholding algorithms. It was found that this new preprocessing method was able to improve the centroid location error by nearly 40% when Yasuda's thresholding algorithm was used. The preprocessing algorithm in combination with Yasuda's thresholding algorithm was able to segment the holes with an average centroid location error of 0.423 pixels and a standard deviation of 0.328 pixels.

I. INTRODUCTION

Visual servoing extends the capabilities of robot systems by utilizing image feedback within the control loop. A camera is used to provide measurements of the location of objects within a robot's environment, enabling the robot to interact with these objects. Visual servoing can be image-based, where the controller moves the robot so as to minimize the error between the current location and the desired location of image features in the image plane. For pose-based visual servoing, shown in Figure 1, the measurements of features in the image are used to determine the relative position and orientation (or pose) between the robot end-effector and the object of interest. The controller error is defined as the difference between the current pose and the desired pose of the end-effector. A more detailed description of visual servoing can be found in the tutorial by Hutchinson *et al.* [1].

Robust visual servoing for real-world robots requires that the image processing to determine the image features must be performed quickly and accurately in a dynamic environment. Liu *et al.* [2] studied a vision-guided control structure that is different from classical visual servoing. However, their observations on the timing of the control loop apply equally well to pose-based and image-based visual servoing. They

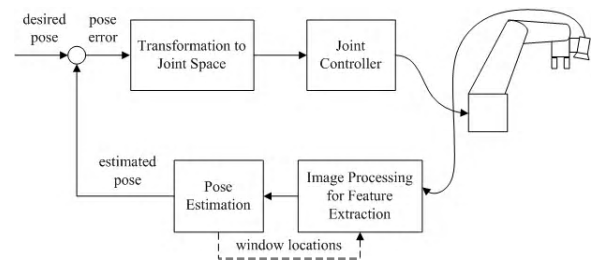


Fig. 1. Visual Servoing Control Loop

state that the rate for complex visual sensing and processing is slower than the minimum rate for mechanical control, and that lags can be introduced due to the latency between sensing and control. A typical image processing algorithm for feature extraction will be $O(m \times n)$, where m is with image width and n the height, with each pixel being visited several times during the algorithm execution.

Thus, an extremely effective way to reduce image processing computation time is to reduce the number of pixels processed. Image windowing can be used when combined with a suitable prediction algorithm that calculates estimated image feature locations for the next sample time. Wilson *et al.* [3] successfully used windowing for pose-based visual servoing. At each time step five 32×32 image windows were processed to determine the first moment or centroid of an object hole feature. The image windows are centered on the predicted feature locations that are calculated using the predicted relative pose from the system's Kalman filter.

The accuracy of the object feature measurements is determined primarily by the ability of the system to classify pixels as object or background. For well-illuminated object hole features with high contrast between the object and background, this process is straightforward, as a fixed threshold can be used to segment the pixels into white or black depending on whether their gray scale value falls above or below the threshold. However, the real world rarely presents such fortuitous conditions, particularly for the applications of interest for visual servoing, such as space or underwater robotics. The lighting may be harsh, and object features can be wholly in shadow resulting in low contrast, or partially shadowed with a mix of high

and low contrast. The lighting and texture of the object can produce reflections. As the object and robot end-effector move during an operation these conditions can change in successive images. Effectively handling challenging lighting conditions is emerging as a key robustness issue for visual servoing systems, and until it is solved it is limiting the adoption of visual servoing in the real world. While more sophisticated algorithms for feature identification could be applied to the problem, a computational price will be paid compared to the use of simpler segmentation algorithms.

Thus, it is worthwhile to consider whether a dynamic thresholding segmentation algorithm can be used to effectively determine object hole feature locations under dynamic and challenging conditions. Thresholding to produce binary images has been widely used in many applications such as optical character recognition (OCR) and medical imaging. Due to its wide use, there have been many different algorithms developed for thresholding images. Several articles compare different thresholding algorithms [4]–[11]. Most of these comparative articles conclude that the choice of thresholding algorithm is highly dependent on the application under consideration.

The image characteristics inherent in OCR and medical imaging are quite different than the characteristics associated with processing small image windows for visual servoing. Thus, none of the test images for the above techniques is a good representation of the images that are used in visual servoing. Experimentation was required to determine which algorithms, if any, are suitable for this novel application area for dynamic thresholding.

To determine the most promising threshold segmentation algorithm for use with visual servoing, existing algorithms were tested. The most promising algorithm was also extended to include a pre-processing step to further enhance its performance for the types of image windows that are characteristic of visual servoing. Section II describes characteristics of the image windows and the criteria by which an algorithm is judged for this application. It also describes the experimental setup used for the work described in this article. Following an initial set of experiments, the most promising algorithms for preprocessing and thresholding were studied. Section III describes these algorithms. It also presents a novel preprocessing algorithm that was developed for this application. In Section IV the algorithms are compared in terms of their effectiveness for visual servoing using image windows. Finally, future directions of the work are presented in Section V.

II. SEGMENTATION FOR VISUAL SERVOING

Thresholding is a widely used segmentation technique used in many applications such as optical character recognition (OCR) and medical imaging. However, unlike most thresholding applications where large images are available, for visual servoing small image windows can be utilized to increase the sample rate due to faster image processing. Based on the experimental setup as described in [3], 32 by 32 pixels was chosen as a suitable image window size. The small number of pixels in the image window greatly reduces the chance

of selecting an effective threshold and also results in high susceptibility to noise.

Another difference in visual servoing systems is the need to preserve the centroid. The centroids are the measurement information that provide feedback to the visual servoing control loop. In order to achieve the desired robot positioning accuracy, the hole centroid measurements need to be determined with sub-pixel accuracy.

In addition to these unique characteristics, there are also the problems of shadows and lighting. The lighting conditions can vary over successive images or frames from the video stream. As a result of changing lighting conditions, the grey scale values for hole and object pixels will vary. At times, the holes can be completely or partially obscured by shadows. For reflective objects, the reflection of a point light source can produce a bright spot.

The experimental visual servoing system uses a Pulnix TM-5LC camera with 508×492 pixels mounted at the robot end-effector. Each pixel is $7.15 \mu\text{m} \times 5.55 \mu\text{m}$. Images were taken of a test object that is used for visual servoing experimental work. The object was selected for the experiments to provide a common basis with earlier experimental work. Some visual servoing systems will incorporate a pattern that is used to provide appropriate features much the same as the way features were designed for the test object. Of key importance is not the construction of the object itself, or whether a pattern has been added to an object to provide easily recognized features, but rather the different lighting conditions that were utilized when capturing images of the object. The images were obtained under varying lighting conditions, resulting in complete and partial shadows, low contrast hole features, high contrast hole features, and bright spots due to reflection of the light.

III. ALGORITHMS

Thresholding algorithms presented in the recent survey by Sezgin and Sankur [4] were tested using the implementation by Sankur [12]. The algorithms were tested on the raw images obtained from the camera and also on images preprocessed by the grayscale morphological algorithm presented below in Section III-A.

A. Preprocessing

To improve the segmentation, a novel preprocessing algorithm was developed. It is an extension of the algorithm presented in [13] for removing variations in illuminations. In [13] the open top hat operation as given in (1) is used to remove illumination variations such as those shown in Figure 2. However, this algorithm will only work so long as the image background is darker than the foreground.

$$f_p = f \hat{\circ} g = f - (f \circ g) \quad (1)$$

Where

- f_p preprocessed image
- f original image
- $\hat{\circ}$ morphological open top hat operation
- g structuring element
- \circ morphological open operation

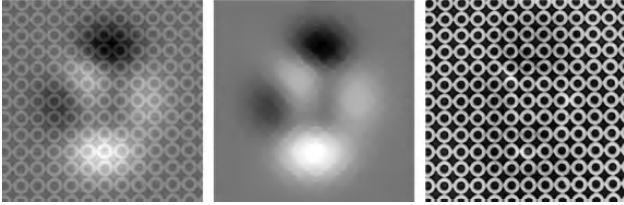


Fig. 2. Uneven illumination compensation. Left - original image, Middle - morphological open operation and Right - normalized morphological open top hat operation. A disk structuring element with radius 10 pixels was used. Images courtesy of [14].

For the images used in our setup, the background is light and the features of interest are dark. As a result (1) will not work. However, it is important to note from Figure 2, that when the morphological open operation is performed the darker regions remain dark and the lighter regions becomes lighter. Based on this observation (1) was modified to give:

$$f_p = f + (f \circ g) \quad (2)$$

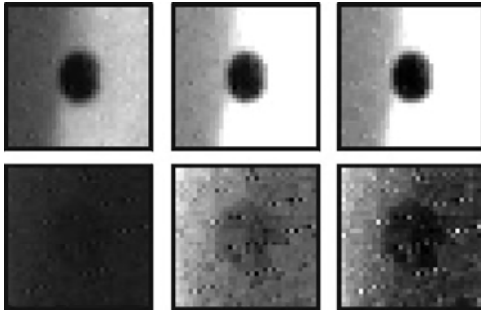


Fig. 3. Preprocessing via morphological open and close operation. A box structuring element of width 3 pixels was used. Left - Original image, Middle - result of (2) and Right - result of (3) with $k = 1$.

The result of (2) is that darker regions will remain dark and the lighter regions will become lighter. Figure 3 illustrates the application of (2) to some sample image windows. The results of (2) can be further improved by replacing the morphological open operator with the close operator. This change is in particular beneficial when there is low contrast between the hole and background. From Figure 3, it can be seen that the hole is much more visible when the morphological close operation is used, rather than the open operation.

In addition to using the morphological close operation, a multiplier k is also introduced in (2) to form:

$$f_p = f + k(f \bullet g) \quad (3)$$

The effects of multiplier k can be seen in Figure 4. As k increases the identification of the hole becomes easier when the original image is very dark. However, when the original image is light, as k increases the area of the hole is reduced. Therefore, if the hole is to be identified without the loss of area, k must be inversely proportional to the image's average gray scale value.

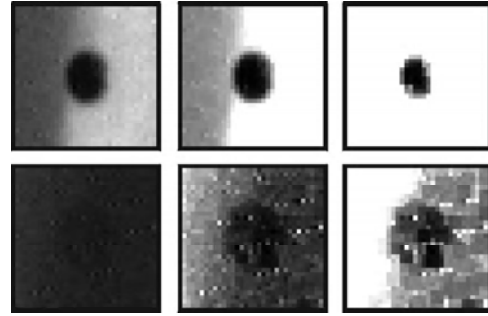


Fig. 4. Preprocessing via weighted morphological close operation. A box structuring element of width 3 pixels was used. Left - Original image, Middle - result of (3) with $k = 1$ and Right - result of (3) with $k = 5$.

This leads to the proposed preprocessing equation:

$$f_p = f + \left(\frac{255}{\bar{f}} - 1 \right) (f \bullet g) \quad (4)$$

Where

- f_p preprocessed image
- f original image
- \bar{f} average grayscale intensity of image
- \bullet morphological close operation
- g structuring element

When implementing (4), be aware of the following:

- i) f is a grayscale image with intensity range from 0 to 255
- ii) if \bar{f} becomes 0 or close to 0 there are numerical issues
- iii) the implementation of the morphological operator followed the implementation described in [13], [14]
- iv) a box structuring element of width 3 was used
- v) the resulting image f_p must be normalized

Some results of this preprocessing system are illustrated in Figure 5.

B. Algorithms of Interest

An initial phase of testing was performed to reduce the number of potential thresholding algorithms. These tests consisted of visually inspecting the result of all algorithms presented in [4]. A sample of 40 images under varying lighting conditions were used for these tests. Based on these tests the four most promising algorithms were chosen to be analyzed for centroid calculations.

Of the chosen algorithms, Niblack [15], Yasuda [16] and Palumbo [17] had the most success in correctly identifying the holes before preprocessing. The algorithm by Ramesh [18] was

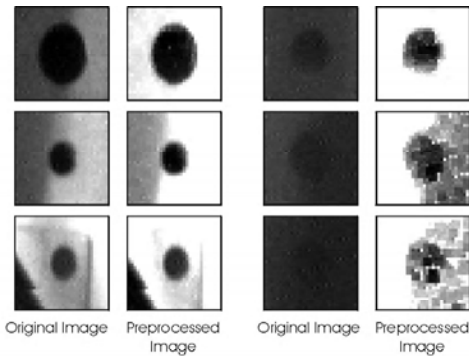


Fig. 5. Original 32 by 32 pixel images and their preprocessed versions

also chosen because unlike the other three algorithms this is a global thresholding algorithm. Though Ramesh's algorithm didn't have much success with the original images, it was quite effective for the preprocessed images.

1) *Niblack*: Niblack's method [15] is a local thresholding method, meaning that the threshold is varied throughout the image based on local statistics. The threshold value, T , at (x, y) is calculated as:

$$T(x, y) = m(x, y) + k \cdot s(x, y) \quad (5)$$

Where

$m(x, y)$	local mean
$s(x, y)$	local standard deviation
k	multiplier

A local window size of 7 by 7 and the multiplier (k) value of -0.1 was determined, by experimentation, to be effective. In addition to this thresholding method, a postprocessing step as introduced in [19] and [5] was used to remove *ghost* objects.

The postprocessing is briefly described here:

- i) Smooth the image using a local 3 by 3 mean filter
- ii) Run edge detection algorithm and thin the edge gradient to 3 pixels wide. The edge thinning process described in [20] was used.
- iii) For all 4 connected black objects in the thresholded image, find the average gradient value along the edge. If the average value is less than 5, invert the values of all pixels within the object.
- iv) For all 4 connected white objects in the thresholded image, find the average gradient value along the edge. If the average value is less than 5, invert the values of all pixels within the object.

2) *Yasuda*: Yasuda's method [16] is yet another local thresholding method. The thresholding process is briefly described here. All constants used below, including window sizes, were obtained from [4].

- i) Increase the dynamic range using

$$f_1 = \frac{f - \min(f)}{\max(f) - \min(f)}$$

Where f is the original image.

- ii) Obtain f_2 by replacing pixels with the average of its 8 neighbouring pixels, if local range, defined as $\max(f_1^{3 \times 3}) - \min(f_1^{3 \times 3})$, is below 50. Where $f_1^{3 \times 3}$ is a local 3 by 3 window of image f_1 .

- iii) Divide f_2 into blocks of 16 by 16 pixels. For each of the blocks perform the following calculations to obtain f_3 .

$$\text{if } ([\max(f_2^{16 \times 16}) - \min(f_2^{16 \times 16})] < 16 \text{ or } f_2(x, y) > \text{avg}(f_2^{16 \times 16}))$$

$$f_3(x, y) = \text{background}$$

else

$$f_3(x, y) = \frac{f_2(x, y) - \min(f_2^{16 \times 16})}{\text{average}(f_2^{16 \times 16}) - \min(f_2^{16 \times 16})}$$

- iv) Obtain the thresholded image f_4 as follows:

$$\text{if } (\min(f_3^{3 \times 3}) < 128 \text{ or } \sigma(f_3^{3 \times 3}) > 35)$$

$$f_4(x, y) = 0$$

else

$$f_4(x, y) = 1$$

3) *Palumbo*: The local thresholding algorithm by Palumbo, Swaminathan and Srihari [17] as presented in [4] is briefly described here. As with Yasuda's algorithm all parameters for this algorithm were obtained from [4]. Five 3 × 3 windows are defined as shown in Figure 6.

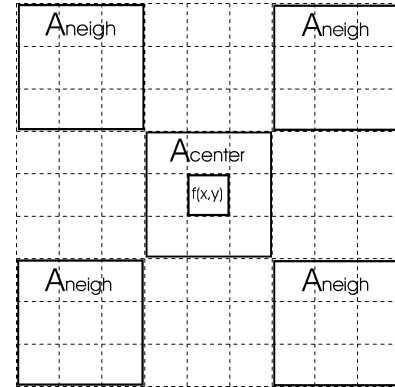


Fig. 6. Relative location of the four, 3 by 3 windows, A_{neigh} and the 3 by 3 window A_{center} with respect to $f(x, y)$

- i) Find the average, m_{center} , of A_{center} .
- ii) Find the average, m_{neigh} , of all pixels in A_{neigh} that is greater than 20.
- iii) Threshold the image as follows:
If $f(x, y) < 20$
set $f(x, y)$ as foreground
else if $0.85m_{neigh} > m_{center}$
set $f(x, y)$ as foreground
else
set $f(x, y)$ as background

4) *Ramesh*: The final algorithm tested, by Ramesh [18], is a global thresholding algorithm based on histogram shape analysis. The algorithm chooses the global threshold value T_{opt} such that,

$$T_{opt} = \min \left[\sum_{g=0}^T (b_1(T) - g)^2 + \sum_{T+1}^G (b_2(T) - g)^2 \right] \quad (6)$$

Where

$$b_1(T) = m_f(T)P(T)$$

$$b_2(T) = m_b(T)(1 - P(T))$$

$$m_f(T) = \sum_{g=0}^T gp(g)$$

$$m_b(T) = \sum_{g=T+1}^G gp(g)$$

$$P(T) = \sum_{i=0}^g p(i)$$

$p(g)$ gray level probability density function

$$g = 0 \dots G$$

G max gray scale value

IV. RESULTS

A set of 462 image windows under varying lighting conditions and with known centroids were used to test the algorithms of interest. The centroid error was calculated as the Euclidian distance between the known actual centroid location and the centroid location defined by the first moment of the thresholded image. The results are summarized in Tables I, II and III.

TABLE I
PERCENTAGE OF THE 462 TEST IMAGE WINDOWS WITH CENTROID LOCATION ERROR LESS THAN 1 PIXEL.

	Original	Preprocessed
Ramesh	74.89%	86.36%
Yasuda	77.92%	94.59%
Niblack	81.17%	86.58%
Palumbo	78.35%	91.13%

TABLE II
AVERAGE CENTROID LOCATION ERROR IN PIXELS. AVERAGE TAKEN OVER THE 462 TEST IMAGE WINDOWS.

	Original	Preprocessed
Ramesh	2.755	0.825
Yasuda	0.695	0.420
Niblack	0.845	0.895
Palumbo	1.005	0.520

From these results it can be seen that the preprocessing step improves the performance of all four algorithms. The best performance was seen when Yasuda's algorithm was used on the preprocessed image windows. The performance of Palumbo's algorithm on the preprocessed image came in close second.

TABLE III

STANDARD DEVIATION OF CENTROID LOCATION ERROR IN PIXELS.
STANDARD DEVIATION TAKEN OVER THE 462 TEST IMAGE WINDOWS.

	Original	Preprocessed
Ramesh	5.728	1.586
Yasuda	0.761	0.328
Niblack	1.223	1.582
Palumbo	1.420	0.562

Without the preprocessing step, it is arguable that Niblack's algorithm performed the best due to the high percentage of image windows with error less than 1 pixel (see Table I). However, it should be noted when Niblack fails the error is high as can be seen from the average error in Table II.

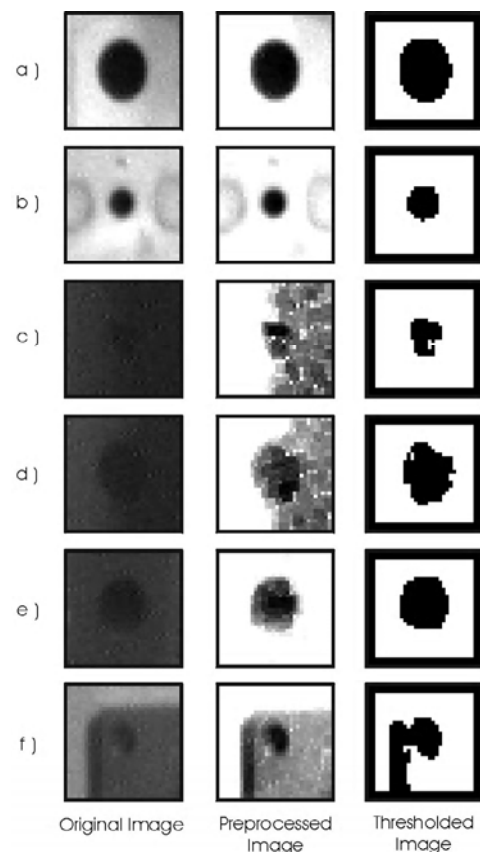


Fig. 7. Image windows preprocessed then thresholded by Yasuda's algorithm. The error in pixels of the centroid locations are a) 0.15, b) 0.19, c) 0.73, d) 0.86, e) 1.06 and f) 2.43 Note the large black border around the thresholded image is due to the thresholding algorithm.

Some results of the final algorithm (preprocessing followed by Yasuda's algorithm) are illustrated in Figure 7. These samples illustrate the algorithm's performance in harsh and optimal lighting conditions. As well the figure shows the algorithm's two modes of failure. The first mode of failure occurs due to a shift in the hole as a result of inconsistent gray scale intensity within the hole. As an example, consider Figure 7 e). Notice that the bottom fraction of the hole is

lighter than the top portion. As a result of this, during the preprocessing phase the bottom portion becomes even more lighter than the top portion of the hole. Then during the thresholding phase the bottom portion of the hole is completely lost, resulting in an upward shift of the centroid.

The second type of failure occurs when a region within the window has a very similar gray scale value as the hole. If this region is separated from the hole by a region of lighter gray scale value, then the thresholded image will identify two objects or holes. In this case, the object closest to the center of the window can be selected and used to calculate the centroid. Most likely this will yield the correct centroid location as the window was placed based on the predicted location of the hole. However, if the region with similar gray scale value as the hole is not separated by a region of lighter gray scale value, then it becomes difficult to separate this region from the hole. The result is an inaccurate calculation of centroid as can be seen in Figure 7 f).

Overall segmentation by the use of the preprocessing algorithm, as introduced in section III-A, followed by Yasuda's thresholding algorithm results in a huge improvement for accurate centroid calculations. This method is far superior than the use of a fixed threshold value for segmentation, currently used in the visual servoing system described in [3]. The use of a fixed threshold value will only work so long as there is constant illumination with no shadows.

The preprocessing algorithm in itself greatly improves the successful segmentation of the hole. The preprocessing algorithm has reduced the best average error seen using dynamic thresholding by nearly 40%. Not only is there an improvement in the average error, but there is also a significant improvement in the standard deviation. The best standard deviation value has been reduced by more than 55% as a result of the preprocessing algorithm.

The success of the preprocessing algorithm is in its ability to increase the contrast between the hole and background. Especially under harsh lighting conditions where, even a person will have trouble determining the location of the hole. This is clearly illustrated in Figure 7 c).

V. CONCLUSIONS AND FUTURE WORK

This study showed that the developed preprocessing algorithm, introduced in Section III-A, followed by Yasuda's dynamic thresholding is a very effective means of segmenting the 32 by 32 pixel image windows used in visual servoing. Yasuda's thresholding algorithm was by itself capable of segmenting the foreground, a black circle, from the background while accurately maintaining the centroid location of the hole down to the sub-pixel range. However, the addition of the preprocessing algorithm further improved this result by nearly 40%. As a result of this method's sub-pixel accuracy in maintaining centroid location, it becomes very useful for use in visual servoing systems.

Though segmentation with the use of the preprocessing algorithm followed by Yasuda's thresholding algorithm proved to be highly accurate, timing analysis must be performed to

determine if this method can satisfy the requirements of a real time visual servoing system. Future work is focusing on the integration of the pre-processing algorithm and Yasuda's algorithm into the visual servoing software so that timing tests can be performed on the experimental system. In addition a timing analysis should also be done on Palumbo's algorithm to determine if it is more feasible to use than Yasuda's. Furthermore the effects of the preprocessing algorithm as well as the thresholding algorithm on the area of the hole should be investigated. The area of the hole feature is under consideration for use in the visual servoing loop, so it is important that the area measurement be accurate as well as the hole centroid measurement.

REFERENCES

- [1] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, October 1996.
- [2] Y. Liu, A. W. Hoover, and I. D. Walker, "A timing model for vision-based control of industrial manipulators," *IEEE Transactions on Robotics*, vol. 20, no. 5, pp. 891–898, October 2004.
- [3] W. J. Wilson, C. C. W. Hulls, and G. S. Bell, "Relative end-effector control using cartesian position based visual servoing," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 684–696, October 1996.
- [4] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146–165, 2004.
- [5] O. D. Trier and T. Taxt, "Evaluation of binarization methods for document images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 3, pp. 312–315, March 1995.
- [6] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern Recognition*, vol. 26, no. 9, pp. 1277–1294, 1993.
- [7] K. S. Fu and J. K. Mui, "A survey on image segmentation," *Pattern Recognition*, vol. 13, no. 1, pp. 3–16, 1981.
- [8] S. D. Zenzo, "Advances in image segmentation," *Image and Vision Computing*, vol. 1, no. 4, pp. 196–210, 1983.
- [9] C. V. Jawahar, P. Biswas, and A. K. Ray, "Analysis of fuzzy thresholding schemes," *Pattern Recognition*, vol. 33, no. 8, pp. 1339–1349, 2000.
- [10] X. Yang, B. Haluk, G. Harkin, and Z. Lewandowski, "Evaluation of biofilm image thresholding methods," *Water Research*, vol. 35, no. 5, pp. 1149–1158, April 2001.
- [11] C. A. Glasbey, "An analysis of histogram-based thresholding algorithms," *CVGIP: Graphical Models and Image Processing*, vol. 55, no. 6, pp. 532–537, November 1993.
- [12] M. Sezgin, "Image and video processing group in BUSIM lab: OTIMEC," (Current 03/31/2005). [Online]. Available: <http://www.busim.ee.boun.edu.tr/image/Image.and.Video.html>
- [13] E. Dougherty and R. Lotufo, *Hands-on Morphological Image Processing*, ser. Tutorial Texts in Optical Engineering. SPIE Press, 2003, vol. TT59.
- [14] "Hands-on morphological image processing: Support to the book Hands-on morphological image processing," (Current 03/31/2005). [Online]. Available: <http://www.mmorph.com/handson/index.html>
- [15] W. Niblack, *An Introduction to Image Processing*. Prentice-Hall, 1986.
- [16] Y. Yasuda, M. Dubois, and T. S. Huang, "Data compression for check processing machines," *Proceedings of the IEEE*, vol. 68, pp. 874–885, 1980.
- [17] P. W. Palumbo, P. Swaminathan, and S. N. Srihari, "Document image binarization: Evaluation of algorithms," *Proc. SPIE Vol. 697*, pp. 278–286, 1986.
- [18] N. Ramesh, J. H. Yoo, and I. K. Sethi, "Thresholding based on histogram approximation," *IEE Proceedings Vision, Image and Signal Processing*, vol. 142, no. 5, pp. 271–279, October 1995.
- [19] S. D. Yanowitz and A. M. Bruckstein, "A new method for image segmentation," *Computer Vision, Graphics and Image Processing*, vol. 46, no. 1, pp. 82–95, 1989.
- [20] K. Treash and K. Amaratunga, "Automatic road detection in grayscale aerial images," *ASCE Journal of Computing in Civil Engineering*, vol. 14, no. 1, pp. 60–69, 2000.