Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

Real Time Detection and Segmentation of Reflectionally Symmetric Objects in Digital Images

Wai Ho Li, Alan M. Zhang and Lindsay Kleeman
Intelligent Robotics Research Centre
Department of Electrical and Computer Systems Engineering
Monash University, Clayton, Victoria 3800, Australia
{ Wai.Li, Alan.Zhang, Lindsay.Kleeman } @eng.monash.edu.au

***Abstract*— Symmetry is a salient visual feature of many man-made objects. This paper describes research into the detection and segmentation of reflectionally symmetric objects in digital images, without the use of *a priori* object models. The detection method does not assume uniform object colour or texture, and does not rely on prebuilt models such as 3D geometric primitives. A novel detection algorithm has been developed to find lines of reflectional symmetry in images. This detection algorithm can operate at 10 frames per second on 640 by 480 pixel images. Using the detected symmetry, objects are segmented with a dynamic programming approach. Both algorithms have been extended to accommodate skew symmetry.**

I. Introduction

There are many definitions for Object Segmentation in the area of Robotics and Computer Vision. For digital images, it can be seen as an intelligent version of *Image Segmentation*. A review of image segmentation techniques can be found in [1]. By incorporating high level knowledge, sections of an image corresponding to a 3D object in the real world is identified, and separated from the background. By doing this, a robot can obtain useful semantic information about the environment. For domestic robots, the ability to quickly and robustly segment man-made objects in the household is highly desirable. For example, a robot designed to clean and tidy desks will need to locate and segment common objects such as cups, bowls and books.

Object segmentation methodologies differ in their assumptions, as well as in their level of prior knowledge. When models of objects are available prior to segmentation, it can be argued that the system is in fact performing object recognition, by matching sensor data with prebuilt models. The Generalized Hough Transform [2] is an example of a model-based segmentation approach. A *predefined* parameterized model of a 2D shape, essentially an object model, is required before the transform can be applied.

In many situations, the *a priori* object information, such as shape and colour, may not be available. The generation of detailed object models can be costly and in many cases, not fully automated. Hence, a robot that can segment objects without using a complicated model is much desired, especially for use in domestic environments. Returning to the desk cleaning robot example, in the case where it encounters a cup without its model, the ideal solution would be to have the robot generate its own learning data by physically interacting with the cup. In order to do this, the robot must begin by using a model-free approach to object segmentation. The ability to detect and segment objects quickly, ideally in real time, will also greatly benefit the robot's responsiveness and robustness to changing environments.

Colour-based and intensity-based image segmentation is a commonly used model-free approach to finding objects in images. Recent research generally focus on colour images, instead of single channel grey scale images. There are a wide variety of techniques available [3]. Colour has proved to be useful in segmenting a variety of entities. Skin colour filters are widely used in face recognition and human tracking applications. However, many man-made household objects are multi-colour, consisting of contrasting colour segments. For example, the cup shown in Figure 6 is very difficult to segment in its entirety using colour alone.

Reflectional symmetry is a common trait in many man-made objects. A surprisingly large number of objects around domestic environments have strong lines of symmetry. By using an object's symmetry line as its model, the assumption of consistent colour is no longer needed. Also, the exact shape of an object's contour is not required prior to segmentation. The work presented here uses reflectional symmetry to detect and segment objects in 2D digital images.

This paper is divided into two major sections. The first section describes an improved implementation of our Fast Symmetry detection algorithm [4]. Substantial modifications have been made to the original algorithm to further improve computational efficiency. This symmetry detection algorithm has also been successfully employed in a real time, model-free, object tracking system [5]. The second section of the the paper will focus on the segmentation of symmetric objects. A brief review of relevant literature and background information can be found at the beginning of each section.

II. Fast Symmetry Detection

A. Similar Work

The Generalized Symmetry Transform [6] can detect bilateral symmetry at different scales. It operates on every possible pixel pair in the input image, and hence has a computational complexity of $O(n^2)$, where n is the total number of pixels in the input image. The transform does not easily adapt to the detection of skew symmetry. Yip's [7]

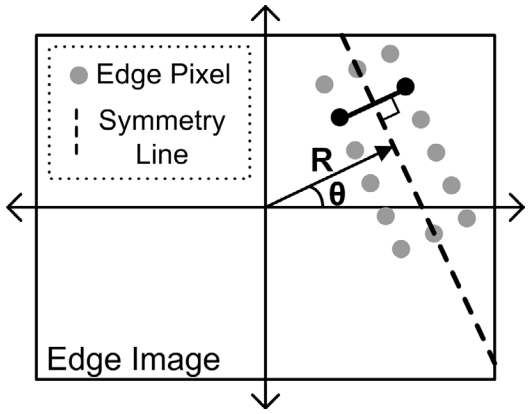


Fig. 1. An edge pixel pair, shown in black, voting for a symmetry line with parameters R and θ. Note that $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$ and R is measured from the image center.

symmetry detector uses mid-point pairs, each generated from two edge pixel pairs. The algorithm has a complexity of $O(n_{edge}^4)$, where n_{edge} is the number of edge pixels. Ogawa suggested the use of Hough transform on edge segments to find lines of reflectional symmetry [8]. Other approaches include the use of ribbons [9] and improvements of the Generalized Symmetry Transform. While radial symmetry has been used in real time applications [10], reflectional symmetry detectors are generally used in offline processing applications, due to their high computational cost.

B. Algorithm Description

Our approach performs symmetry detection on an image's edge pixels. In our experiments, we found that a million-pixel image produces an edge image with roughly 10000 edge pixels. Of course, this number will depend on the visual complexity of the scene, and the characteristics of the edge filter. Apart from reducing data size, symmetry detection can also benefit from the noise rejection, edge linking and weak edge retention properties of edge filters. The Canny edge filter was used to generate the edge images presented in this paper.

A polar parameterization is used for the symmetry lines, as shown in Figure 1. Symmetry lines are represented by their angle and distance relative to the center of the image. Edge pixels are grouped into pairs and each pair votes for a single symmetry line in parameter space. Unlike traditional Hough Transform [11], which requires multiple votes per edge pixel, our approach only requires a single vote per edge pixel *pair*. This *convergent* voting scheme is similar to that utilized in Randomized Hough Transform [12].

For each Hough angle bin θ_{index}, the edge pixels are rotated about the center of the image by a corresponding angle θ. The rotated edge pixels are then quantized into a 2D array Rot. Edge pixels are quantized based on their *scanline* after rotation, as shown in Figure 2. Edge pixels on the same horizontal scanline after rotation are placed in the same row of Rot. Notice that any pair of rotated pixels from the same scanline can only vote for vertical lines of symmetry. This

Algorithm 1: Fast Symmetry Detection

Input: I – Source Image
Output: sym – Symmetry Line Parameters (R, θ)
Parameters:
D_{min} – Minimum distance threshold
D_{max} – Maximum distance threshold
H – Hough Accumulator ($BINS_R$ by $BINS_\theta$ matrix)
$BINS_R$ – Number of radius bins in H
$BINS_\theta$ – Number of angle(theta) bins in H
N_{lines} – Number of symmetry lines returned

$edgePixels \leftarrow$ (x,y) locations of edge pixels in I
$H[\ ][\ ] \leftarrow 0$
for $\theta_{index} \leftarrow 0$ *to* $BINS_\theta - 1$ **do**
  $\theta \leftarrow \theta_{index}$ in radians
  $Rot \leftarrow$ Rotate $edgePixels$ by angle θ. *See Figure 2*
  for *each row in* Rot **do**
    for *each possible pair* (x_1, x_2) *in current row* **do**
      $dx \leftarrow |x2 - x1|$
      if $dx < D_{min}$ **then**
        continue to next pair
      if $dx > D_{max}$ **then**
        continue to next pair
      $x_0 \leftarrow (x2 + x1)/2$
      $R_{index} \leftarrow x_0$ converted to Hough index
      Increment $H[R_{index}][\theta_{index}]$ by 1
for $i \leftarrow 1$ *to* N_{lines} **do**
  $sym[i] \leftarrow max(R_{index}, \theta_{index}) \in H$
  Bins around $sym[i]$ in $H \leftarrow 0$

corresponds to the dashed symmetry line with angle θ prior to the rotation operation. The algorithm only allows edge pixels on the same scanline (same row in Rot) to pair up during Hough voting. This guarantees that all votes will be made for the same angle, shown as θ in Figure 2. Note that points separated by a distance greater than D_{max} or less than D_{min} are never paired, even if they belong to the same row.

There are two benefits in having a constant θ for each voting iteration. Firstly, random memory access across θ in the Hough accumulator is removed. Therefore, only a single row of the accumulator needs to be cached during each iteration, instead of the entire Hough accumulator. Secondly, the arithmetic requirements to calculate the symmetry line parameters from an edge pixel pair, as described in Figure 1, becomes greatly simplified. The line radius R can be found by taking the average of the x coordinates of the edge pixel pair, which are readily available in Rot.

Symmetry line parameters are found by looking for peaks in the Hough accumulator. The final for-loop in Algorithm 1 describes the non-maxima suppression algorithm used for peak finding. Maximum values in the Hough accumulator H are found iteratively. Each iteration is followed by setting the maximum value and the neighbourhood of bins surrounding

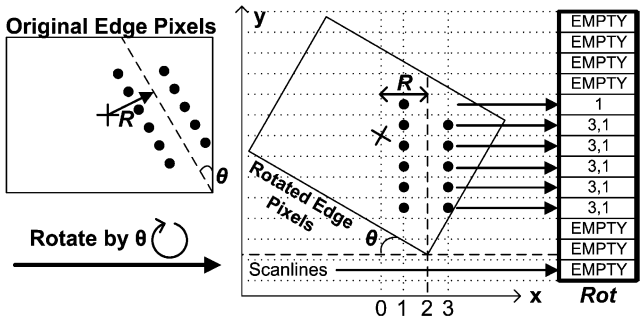
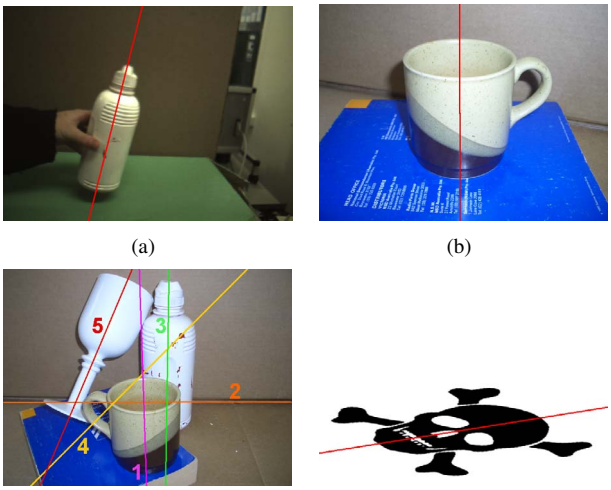


Fig. 2. Edge pixel rotation and discretization procedure. Edge pixels (•) are rotated by θ about the image center, marked as a $+$. Then, the horizontal coordinate of the rotated pixels are inserted into the 2D array Rot. Pixels from the same scanline are placed into the same row in Rot. The pixels in the same row are paired up and votes for symmetry lines in $R-\theta$ parameter space. The [3,1] rows in Rot will vote for the dashed symmetry line



(a) (b) (c) (d)

Fig. 3. Fast Symmetry Detection Results (a) Partially occluded, non-uniformly textured object; (b) Multi-colour object; (c) Multi-object scene. Top five symmetry lines shown, numbered in the order they were found during detection; (d) Fast Symmetry algorithm extended to detect skew symmetry

it to zero. In general, the contribution of peak finding to execution time was negligible when compared with the Hough voting stage of the algorithm.

The edge pixel rotation operation has a computational complexity of $O(n_{edge})$, where n_{edge} is the number of edge pixels in the image. It is repeated $BINS_\theta$ times. Its effect on the overall algorithm execution time is negligible as it runs in *linear* time. The majority of computation occurs during the voting portion of the algorithm, where the accumulator H is incremented. The amount of computation needed depends on the Hough angle quantization and the vertical quantization used in Rot. Assuming uniformly distributed edge pixels across the rows of Rot, the algorithm requires $\frac{BINS_\theta}{D} \times n_{edge}^2$ voting operations, where $BINS_\theta$ is the number of Hough angle divisions and D is the number of rows in Rot. By increasing the number of angle divisions, we improve the accuracy of the method but sacrifice execution time. The reverse is true if we increase the number of rows in Rot. In essence, the $\frac{BINS_\theta}{D}$ term allows for an adjustable trade off between detection accuracy and computational efficiency.

The voting scheme has been extended to allow for the detection of skew symmetry. Figure 3(d) shows detection results for a horizontally-skewed skull-and-crossbones poison logo. This was achieved by voting for all lines passing through the mid-point of an edge pair, in the same way as the standard Hough transform. The algorithm's order of computational complexity remains the same when detecting Skew Symmetry, as the number of angle divisions in the Hough accumulator is fixed. In addition to the constant-factor increase in computational cost, two additional matrices, the same size as the Hough accumulator, are required. Please refer to Lei and Wong's work on skew symmetry detection [13] for additional information concerning these extra Hough matrices.

C. Results

Figure 3 contains detection results of our fast symmetry algorithm. Subfigures 3(a) and 3(b) show the symmetry lines with the most votes in red. Note that in 3(a), both the symmetry of the forearm and the symmetry between the forearm and its shadow were also detected, but they had much fewer votes than the bottle. Subfigure 3(d) shows the detection of skew symmetry using the algorithm, using a modified voting procedure.

Subfigure 3(c) displays the detection results for a more complicated arrangement of objects. The lines are labelled according to the number of votes they received, with 1 being the symmetry line with the most votes. Notice that the symmetry lines of all three objects were found. However, background symmetries were also detected. Line 2 is due to the symmetry of the long horizontal shadow and line 4 is caused by inter-object symmetry, primarily between the two cups as well as symmetry between the multi-colour cup and the horizontal shadow. The detection of background symmetry and inter-object symmetry is unavoidable due to the lack of prior knowledge available to the detection algorithm. Whether this behaviour is desirable will depend on the application. To reject *narrow* symmetry, such as line 2 in subfigure 3(c), the distance threshold D_{min} in algorithm 1 can be increased. The orientation of the symmetry line can also be used to reject unwanted symmetry, especially when some prior knowledge of the scene is available to the robot. For example, a humanoid robot trying to manipulate cups and bottles on a table will generally deal with near-vertical lines of symmetry only. As such, the voting angles available to the symmetry detector can be constrained accordingly.

An implementation of Algorithm 1 in C++ was used for all experiments. The experimental platform was a desktop PC with a Xeon 2.2GHz CPU and 1GB of main memory. No platform-specific optimizations, such as SSE2 macro functions, were used in the code. Referring to the parameters defined in Algorithm 1, the hough accumulator had 180 angle divisions ($BINS_\theta$). The number of radius divisions ($BINS_R$)

TABLE I
EXECUTION TIME OF THE FAST SYMMETRY DETECTION ALGORITHM

Image Number	Image Dimensions	No. of Edge Pixels	Execution Time (ms)
1	640 X 480	9766	136
2	640 X 480	15187	224
3	640 X 480	9622	153
4	640 X 480	9946	141
5	640 X 480	9497	128
6	640 X 480	9698	145
7	640 X 480	11688	167
8	640 X 480	11061	180
9	640 X 480	12347	196
10	640 X 480	8167	81
11	610 X 458	6978	97

was equivalent to the size of the image diagonal in pixel units. D_{max} was half the image width and D_{min} was set to 5 pixels. The peak finding returned the five strongest symmetry lines. Borrowing from randomized hough transform [12], the list of edge pixels were sampled at a ratio of 25% to obtain the timing results shown in Table I. The execution times include time required for edge filtering as well as non-maxima suppression peak finding.

The execution times confirm that the amount of computation increases as the number of edge pixels extracted from the input image increase. More complicated images require more time as they tend to generate more edge pixels. The detection time for 640x480 images ranged from 80 to 224 milliseconds. This will allow for frame rates between 5 to 12 Hz. This is acceptable for many real time applications. The use of a processing window or smaller images, a faster PC and processor-specific optimizations can further improve the speed of the algorithm to meet more stringent real time requirements. The sampling ratio of edge pixels can also be adjusted at run time to alter the detection speed. Frame rates of 20Hz were achieved by reducing the input image size to 320x240 pixels.

III. SEGMENTATION OF SYMMETRIC OBJECTS

Having obtained the location of the symmetry line, the next logical step is object segmentation. The approach chosen performs segmentation by finding an object's outline. Simply identifying all edge pixels that voted for the symmetry line is not acceptable due to the large number of coincidentally matching pairs of edge points. A more robust method is needed. But before continuing, a definition for "object outline" is required. Because no prior model or geometric properties of the object are assumed apart from its symmetry, a definition is difficult. We define an object outline as the most *continuous contour* symmetric about the object's line of symmetry. While the definition is not perfect, it does allow for the problem to be solved robustly.

The task then becomes finding the most *continuous and symmetric contour* in the edge image, about a detected symmetry line. For real time applications the algorithm used to solve this problem needs to have predictable execution times. This makes approaches that require initialization and multiple iterations, such as active contours, unsuitable. The proposed algorithm uses a single pass Dynamic Programming (DP) approach. While much research has been performed on the use of DP to find contours in images [14]–[17], they require a human-selected starting point. For the object outlines being considered, a human user would have to provide an initial pair of symmetric pixels on the outline. As a major goal of object segmentation is image understanding without human intervention, we chose an approach that requires no human intervention. Section III-A describes a preprocessing step that removes non-symmetric edge pixels, tentatively named the *Symmetric Edge Pair Transform*. Section III-B describes the dynamic programming stage. Results and processing times are presented in Section III-C.

A. The Symmetric Edge Pair Transform

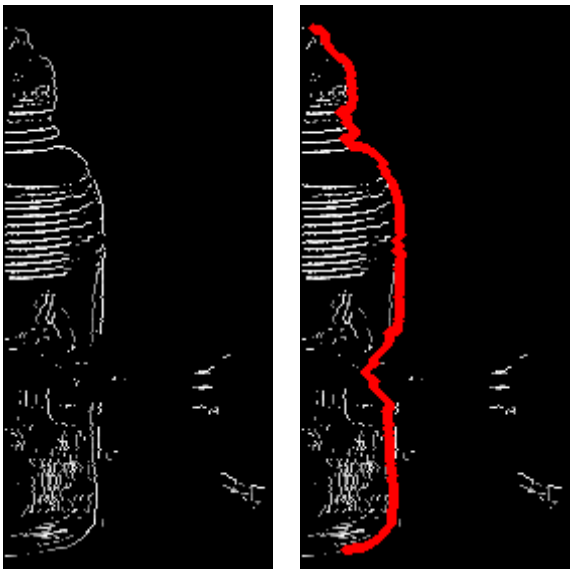
Algorithm 2: Symmetric Edge Pair Transform (SEPT)

Input: Edge image, Edge Gradient Orientation information
Output: $SeptBuf$ – Result of Transform
Parameters:
MAX_{hw} – half of the maximum expected width of symmetric objects
WND – the allowable deviation of the mid point of an edge pair from the symmetry line.

```
Set all SeptBuf to −1
for each row r of the edge image do
    for each pair of edge pixels in the same row do
        if the gradient orientations of the pixels are symmetric then
            w ←distance between the pixel pair
            d ←distance between the mid point of the pair and the symmetry line
            if w/2 < MAX_hw And d < WND then
                if SeptBuf[r + 1][ceil(w/2)] < W(d) then
                    SeptBuf[r + 1][ceil(w/2)] ← W(d)
                    where W(•) is a weighting function
```

We introduce the Symmetric Edge Pair Transform (SEPT) as a preprocessing stage. The edge pixels are first rotated such that symmetric edge point pairs lie in the same row. The idea behind the transform is to parameterize pairs of near-symmetric points by their distance of separation and deviation of their mid point from the symmetry line. The algorithm has also been generalized to accommodate skew symmetry by using a non-vertical symmetry line *after* rotating the edge pixel pairs. The transform is described in Algorithm 2.

The weighting function $W(\bullet)$ in Algorithm 2 is required to be a monotonically decreasing function, such that the higher



(a) (b)

Fig. 4. SEPT performed using the symmetry detection result in Figure 3(a). (a) The SEPT buffer, where brighter pixels represent higher weighting values; (b) Highest scoring continuous contour thickened and overlaid on the SEPT buffer.

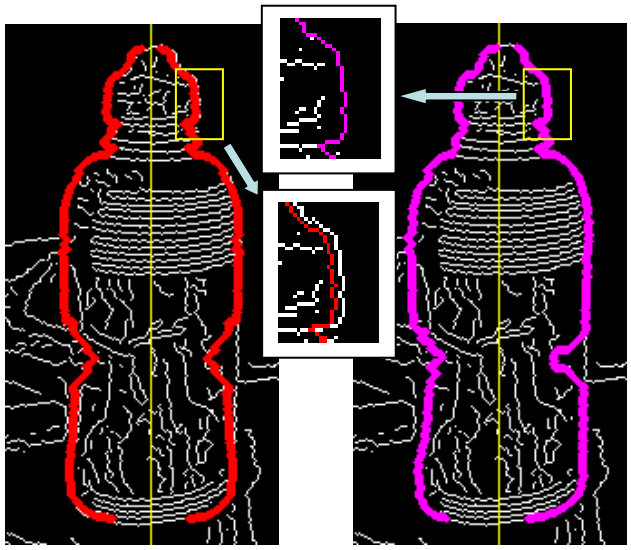


Fig. 5. Object Segmentation and Contour Refinement. The object outline discovered by back tracking from the maximum value in the score table is shown on the left. Object outline after Contour Refinement is on the right

the deviation of the mid point from the symmetry line, the lower the weight given. In our implementation, the weighting function $W(d) = 1 - \frac{d}{2WND}$ was used. The variables d and WND are defined in Algorithm 2. Figure 4(a) is an image visualization of the $SeptBuf$ generated from Figure 3(a), where brighter pixels represent higher weights. The entire $SeptBuf$ is 573 by 355 pixels, but only the portion containing the object is shown in Figure 4(a). Notice the numerous coincidentally symmetric edges on the interior of the bottle, and in the background.

B. Finding Continuous Symmetric Contours with Dynamic Programming

If $SeptBuf$ is treated as a weighted edge image, then the problem of finding the most continuous and symmetric contour in the original edge image becomes one of finding the most *continuous* contour in $SeptBuf$. And this is done using a dynamic programming approach. The algorithm constructs a table of cumulative scores the same size as $SeptBuf$, where a high scoring cell indicates that there is a continuous contour passing through it. The weights in $SeptBuf$ affect the score when moving from one cell to the next. The details of the approach is represented in Algorithm 3. Step 1 calculates the score of the current cell from its 3 neighbours in the previous row. It finds the best score arising from vertical continuity. $backPtr$ keeps track of *which* of the 3 neighbouring cells produced the best score for the current cell. Step 2 scans $SeptBuf$ from left to right, calculating the scores obtained from horizontal continuity. Step 3 looks for horizontal continuity from right to left. This could cause cycles to form in the same row of the $backPtr$ array. To prevent these cycles a copy of $backPtr$ is made in Step 1. If the score from horizontal continuity is higher than from vertical continuity then the higher score is recorded, and $backPtr$ is updated. Horizontal continuity is given less reward than vertical continuity. The reason for this is to reject long horizontal edge segments, which are technically symmetric about its mid point. The symmetry detected for these straight lines very rarely represent actual objects. Humans generally consider symmetric objects as those that have symmetric boundaries *along the direction of* the mirror line. If the horizontal continuity reward is too high, it may also lead to unwanted zig-zag patterns in the generated contours.

After filling the score table, the "best" symmetric contour can be found by starting at the highest scoring cell, and back tracking through cells of lower weight. The back tracking algorithm is described in Algorithm 4. Note that both copies of back pointers are utilized by the algorithm so that there will be no purely horizontal segment in the contour. By keeping a list of position indices $\{r, c\}$ during the back tracking process, the contour of the object can be extracted. The column index c indicates the horizontal distance of the contour from the symmetry line. An example of the resulting contour can be seen superimposed on to $SeptBuf$ in Figure 4(b) and in the left image of Figure 5.

The contour obtained thus far does not directly correspond to edge pixels. This is due to the tolerance introduced in the SEPT preprocessing. In order to produce a contour that corresponds to actual edge pixels, a refinement stage was added. In this stage, the same window size used in the SEPT, WND, is employed to refine the contour. By looking for edge pixels near the symmetric contour, within the window, the algorithm produces a new, near-symmetry outline. The contour refinement stage is similar to Algorithm 3, substituting

Algorithm 3: Finding Continuous Symmetric Contours with Dynamic Programming

Input: $SeptBuf$
Output: $sTab$ – Table of scores, same size as $SeptBuf$
$backPtr$ – back pointers
Parameters:
H_{img} – image height
MAX_{hw} – half of the maximum expected width of symmetric objects
$\{P_{ver}, R_{ver}\}$ – penalty/reward for vertical continuity
$\{P_{hor}, R_{hor}\}$ – penalty/reward for horizontal continuity

$sTab[\,][\,] \leftarrow 0$
for $r \leftarrow 1$ *to* H_{img} **do**

Step 1, vertical continuity

for $c \leftarrow 1$ *to* MAX_{hw} **do**
if $SeptBuf[r][c]$ *is not* -1 **then**
$cost \leftarrow SeptBuf[r][c] * R_{ver}$
else
$cost \leftarrow P_{ver}$

$$vScore[c] \leftarrow max \begin{cases} 0 \\ sTab[r-1][c-1] + cost \\ sTab[r-1][c] + cost \\ sTab[r-1][c+1] + cost \end{cases}$$

if $vScore[c] > 0$ **then**
Set $backPtr[r][c]$ to record which of the 3 neighbouring cells was used to produce $vScore[c]$
$backPtrAux[r][c] \leftarrow backPtr[r][c]$

Step 2, horizontal continuity from left to right

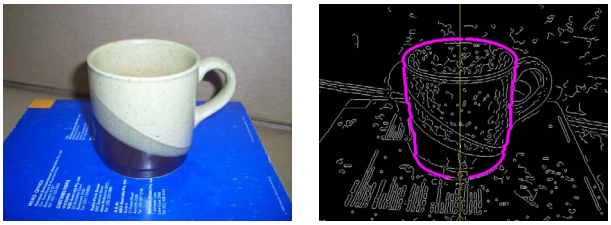
$prevScore \leftarrow neg.\ inf.$
for $c \leftarrow 1$ *to* MAX_{hw} **do**
if $SeptBuf[r][c]$ *is not* -1 **then**
$cost \leftarrow SeptBuf[r][c] * R_{hor}$
else
$cost \leftarrow P_{hor}$
$hScore \leftarrow prevScore + cost$
if $vScore[c] >= hScore$ **then**
$prevScore \leftarrow vScore[c]$
$columnPtr \leftarrow c$
else
$prevScore \leftarrow hScore$
if $sTab[r][c] < prevScore$ **then**
$sTab[r][c] \leftarrow prevScore$
Set $backPtr[r][c]$ to record position $\{r, columnPtr\}$

Step 3, horizontal continuity from right to left
Repeat Step 2, moving right to left in column index

Algorithm 4: Back tracking highest score in the score table

Input: $sTab$, $backPtr$, $backPtrAux$
Output: $\{r, c\}$ – {Row, column} indices

$\{r, c\} \leftarrow$ position of MAX($sTab$)
while $sTab[r][c]$ *is not zero* **do**
$\{r, c\} \leftarrow backPtr[r][c]$
if r *did not change, i.e. no vertical position change* **then**
$\{r, c\} \leftarrow backPtrAux[r][c]$



(a) (b)

Fig. 6. Segmentation of a multi-colour object. The contour has been thickened, and is shown in purple. The symmetry line, in yellow, was obtained directly from the symmetry detection results shown in Figure 3(b)

the $SeptBuf$ with the edge image. Results from contour refinement is shown in the right image of Figure 5.

C. Results

Figure 6 shows the segmentation of a multi-colour object. This result demonstrates the algorithm's ability to segment objects of non-uniform colour. Note that the edge image was also quite noisy due to texture on the cup surface and on the book in the background. This noise did not adversely affect the segmentation results. In Figure 7, all three symmetric objects were able to be segmented using our approach. Note that, in all of our results, no prior information in the form of geometric models, object colour or texture were used. The only information received by the segmentation algorithm are the detected symmetry line parameters and the edge image. Due to shadows and specular reflections, the vertical side edges of the small, multi-colour cup were distorted and had very large gaps. Hence, the more symmetric and continuous elliptical contour of the cup's opening was returned by the segmentation algorithm. There is a slight distortion in the detected ellipse, which resulted because of large gaps in the outer rim of the cup in the edge image. This produced a contour that contained a combination of the inner and outer rim of the cup's opening.

Table II contains execution times of a C++ implementation of the object segmentation algorithm. The same computer described in Section II-C was used for these experiments. The image numbers are the same as those used in Table I. The test cases with smaller score tables are able to be processed at 30 frames per second (FPS). Test cases with larger tables can still be processed at 20FPS. The third column of Table II, labelled "No. of Edge Pairs", was the number of edge pixel

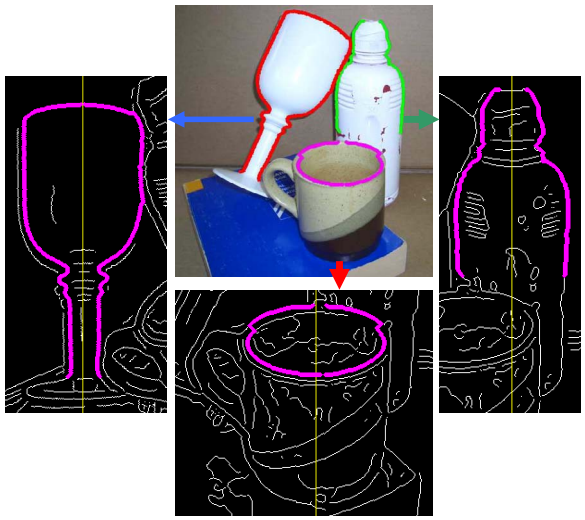


Fig. 7. Object segmentation performed on a scene with multiple objects using the results from Figure 3(c). The object outlines have been thickened and rotated such that their symmetry lines are vertical

TABLE II
EXECUTION TIME OF THE OBJECT SEGMENTATION ALGORITHM

Image No.	Size of Cumulative Score Table	No. of Edge Pairs	Execution Time (ms)	
			SEPT+DP	†CR
1	356 x 576 (= 205056)	77983	26	4
2	322 x 482 (= 155204)	142137	25	4
3	322 x 482 (= 155204)	65479	19	3
4	326 x 493 (= 160718)	68970	21	4
5	402 x 476 (= 191352)	67426	36	7
6	382 x 801 (= 305982)	44901	36	8
7	349 x 556 (= 194044)	90104	26	6
8	345 x 546 (= 188370)	133784	28	4
9	402 x 777 (= 312354)	121725	40	8
10	393 x 705 (= 277065)	177077	32	7
11	383 x 722 (= 276526)	51475	32	6

†CR: contour refinement stage

pairs processed by the SEPT. In our implementation, the SEPT code that filled the $SeptBuf$ and the dynamic programming code were placed within the same loop to improve efficiency. As such, their combined execution time is shown in the "SEPT+DP" column. Looking at Table II, the size of the cumulative score table appear to be the main factor affecting the execution time. This agrees with expectations as a score is calculated for each entry in the table. The maximum expected size of objects is set to be the width of the image. In practice, the size of the objects can be restricted to more reasonable bounds, especially considering the presence of distance thresholds in our symmetry detection algorithm. This will further improve execution time.

IV. CONCLUSION

A pair of algorithms designed to detect and segment reflectionally symmetric objects in digital images have been detailed. Their execution times indicate that both algorithms can be applied in real time applications. The Fast Symmetry detector has been successfully applied to multi-colour objects and objects with irregular textures. It is also able to locate multiple symmetric objects, including a partially occluded object, from the same scene. The dynamic programming approach is able to segment objects in the presence of noisy edge pixels caused by shadows, background clutter and object texture. Both algorithms are quite robust to inter-object occlusions, and were able to find and segment the visible portion of occluded objects.

ACKNOWLEDGEMENT

The authors would like to thank the ARC Centre for Perceptive and Intelligent Machines in Complex Environments (pimce.edu.au) for their financial support.

REFERENCES

[1] N. R. Pal and S. K. Pal, "A review on image segmentation techniques." *Pattern Recognition*, vol. 26, no. 9, pp. 1277–1294, 1993.
[2] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," in *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, M. A. Fischler and O. Firschein, Eds. Los Altos, CA.: Kaufmann, 1987, pp. 714–725.
[3] W. Skarbek and A. Koschan, "Colour image segmentation — a survey," Institute for Technical Informatics, Technical University of Berlin, Tech. Rep., October 1994.
[4] W. H. Li, A. Zhang, and L. Kleeman, "Fast global reflectional symmetry detection for robotic grasping and visual tracking," in *Proceedings of Australasian Conference on Robotics and Automation*, M. M. Matthews, Ed., December 2005.
[5] W. H. Li and L. Kleeman, "Real time object tracking using reflectional symmetry and motion," Accepted for publication at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, China, October 2006.
[6] D. Reisfeld, H. Wolfson, and Y. Yeshurun, "Context-free attentional operators: the generalized symmetry transform," *Int. J. Comput. Vision*, vol. 14, no. 2, pp. 119–130, 1995.
[7] R. K. K. Yip, "A hough transform technique for the detection of reflectional symmetry and skew-symmetry." *Pattern Recognition Letters*, vol. 21, no. 2, pp. 117–130, 2000.
[8] H. Ogawa, "Symmetry analysis of line drawings using the hough transform." *Pattern Recognition Letters*, vol. 12, no. 1, pp. 9–12, 1991.
[9] J. Ponce, "On characterizing ribbons and finding skewed symmetries," *Comput. Vision Graph. Image Process.*, vol. 52, no. 3, pp. 328–340, 1990.
[10] G. Loy and A. Zelinsky, "Fast radial symmetry for detecting points of interest," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 959–973, 2003.
[11] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, 1972.
[12] L. Xu and E. Oja, "Randomized hough transform (rht): basic mechanisms, algorithms, and computational complexities," *CVGIP: Image Underst.*, vol. 57, no. 2, pp. 131–154, 1993.
[13] Y. Lei and K. C. Wong, "Detection and localisation of reflectional and rotational symmetry under weak perspective projection." *Pattern Recognition*, vol. 32, no. 2, pp. 167–180, 1999.
[14] P. Yan and A. A. Kassim, "Medical image segmentation with minimal path deformable models," in *Proceedings of the International Conference on Image Processing ICIP'04*, vol. 4, 2004, pp. 2733–2736.
[15] B. Lee, J.-Y. Yan, and T.-G. Zhuang, "A dynamic programming based algorithm for optimal edge detection in medical images," in *Proceedings of the International Workshop on Medical Imaging and Augmented Reality*, 2001, pp. 193–198.
[16] E. Mortensen, B. Morse, W. Barrett, and J. Udupa, "Adaptive boundary detection using live-wire two-dimensional dynamic programming," in *IEEE Proceedings of Computers in Cardiology*, 1992, pp. 635–638.
[17] T. Yu and Y. Luo, "A novel method of contour extraction based on dynamic programming," in *Proceedings of the 6th International Conference on Signal Processing (ICSP'02)*, Beijing, China, 2002, pp. 817–820.