SAND REPORT

SAND2004-1871
Unlimited Release
Printed May 2004

Matching a Statistical Pressure Snake to a Four-Sided Polygon and Estimating the Polygon Corners

Hanspeter Schaub, ORION International Technologies
Chris Wilson, Sandia National Laboratories

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2004-1871
Unlimited Release
Printed May 2004

Matching a Statistical Pressure Snake to a Four-Sided Polygon and Estimating the Polygon Corners

Hanspeter Schaub
ORION International Technologies
Albuquerque, NM 87106
schaub@vt.edu

Chris Wilson
P.O. Box 5800, MS-1003
Sandia National Labs
Albuquerque, NM 87183-1003
ccwilso@sandia.gov

Abstract

Given a video image source, a statistical pressure snakes is able to track a color target in real time. This report presents an algorithm that exploits the one-dimensional nature of the visual snake target outline. If the target resembles a four-sided polygon, then the four polygon sides are identified by mapping all image snake point coordinates into Hough space where lines become points. After establishing that four dominant lines are present in snake contour, the polygon corner points are estimated. The computation burden of this algorithm is of the $N \log N$ type. The advantage of this method is that it can provide real-time target corner estimates, even if the corners themselves might be occluded.

Acknowledgment

Thanks to Chris Smith from the University of New Mexico for valuable discussions on the use of Hough transformations.

Contents

Figures

Intentionally Left Blank

Matching a Statistical Pressure Snake to a Four-Sided Polygon and Estimating the Polygon Corners

Introduction

Statistical pressure snakes[1] are a means to have a closed curve (snake) track an object. Recent modifications made to Chris Smith's snake routines have made it possible to track targets in a variety of lighting conditions.[3] The potential applications for these snakes are many. The snake algorithm is envisioned to be used in closed-loop servoing tasks, as well as an off-line aid to the operator to select certain targets in the image. Reference 4 investigates methods to extract primary target shape features in a fast and reliable manner with the goal to use this information in a closed-loop visual servoing system. Fast computational speeds are possible because the algorithm only needs to work on a relatively small number of discrete points. If the target is a rectangular box with right corners, then routines are presented in Reference 4 that estimate the box center, orientation, and length and height dimensions.

This technical report describes the algorithm that processes the discrete snake points and attempts to find the four dominant corner points. The statistical pressure snakes attempt to track a given set of image colors while keeping the snake points evenly and smoothly distributed. Thus, if the target is a square rectangle, then the corners of the rectangle will not be tracked well and are rounded off. To track the center of mass or orientation of the target box, this small rounding off has a minimal effect. However, if the user desires to locate the precise corner points, the standard snake by itself does not provide a very good solution.

Figure 1 illustrates the end result of the new algorithm. Points corresponding to the four dominant straight line segments along the parametric snake curve are determined first. After finding a least squares fit to each of the four lines, the four intersection points are computed (highlighted as yellow circles in Figure 1. With this routine, note that the actual corners need not be visible. Also, the straight line segments of the target polygon can be interrupted to some degree, as long as the box edges still are the four largest straight line segments of the snake.

The 4 corners of a box-shaped target could be visually tracked in many ways. For example, a corner finding routine could be applied to the entire image (by searching the

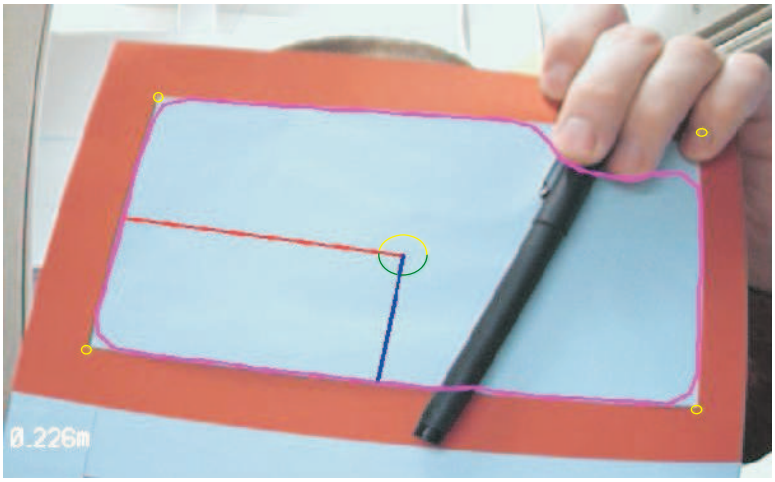

Figure 1. Illustration of the 4-Sided Polygon Corner Tracking Routine

image eigenvalues). However, such algorithms need to process all $n \times m$ pixels of the image and are much slower than this algorithm that only needs to operate on a relatively small number of snake points. Further, corner finding routines can easily be confused by other corners in the image. For example, note the clear corners of the red frame surrounding the blue target box in Figure 1. Since the snake robustly tracks the target shape, this snake-based corner detection method is a lot more robust than a standard corner detection scheme. Lastly, note that in the image shown in Figure 1 the upper right corner is obscured and not visible. A corner detection routine would not be able to estimate the invisible corner location, where the snake-based corner detection routine is still able to estimate the corner location rather accurately. An alternate method to track straight lines is to process the image using a Hough Transform and detect all linear line segments in the image. Again, this is a rather expensive computation process when applied to the entire $n \times m$ image, and which also requires a very sophisticated post-processing logic to determine which straight lines found belong to the target. For example, in the image shown in Figure 1 a black pen is held across the image. A standard straight-line detection routine could easily become confused by this additional straight line in the image. As is shown in the Figure 1, the snake-based corner detection scheme is not sensitive to such minor errors in the snake tracking capability.

However, if sub-pixel accuracy of the box corners is required, then the eigenvalue corner finding routines could be used along with this routine, if the box corners are visible. The snake-based corner finding routine would provide a very good estimate of where the

box corner is. This estimated would then be refined by the image eigenvalue based corner finding routine which would use the previous corner estimate to seed the search region.

Typically this routine requires more snake points to be available than the standard snake routine used for performing real-time tracking of the target center and orientation. Being a fast $N\log_2(N)$ routine, the corner identification will only be roughly as fast as the N^2 snake routine, where N is the number of segments in the snake. A fast computer must be used to do this in real-time for control application. However, the main application of this routine is for the off-line use by the remote manipulator operator. A common task for the operator is to select corner points on an object to define a coordinate system for the robotic path planning algorithm. With the presented corner detection algorithm, it should be possible for the operator to click on a square object (for example a suitcase) and the routine would provide the user with the various corner points. Combined with a stereo-vision system, the same box could be detected in the second view to compute polygon plane position and orientation.

This report explains each step of the snake-based corner detection algorithm in detail. The first step is to map the (x,y) snake points into a Hough space where lines are represented by points. After identifying tight clusters of line points in Hough space, the four largest clusters are identified. The corresponding (x,y) coordinates of the snake points which form a common line are curve fit optimally using a least squares routine. The final step is to determine the proper intersection points.

Transformation to Hough Space

Lines in traditional Cartesian coordinate space are typically represented through

$$y = c + mx \tag{1}$$

where c is the zero crossing offset and m is the slope. Any point (x,y) that lies on this line will share the same (c,m) values. Thus, the parameters (c,m) can be considered to be the invariant properties of the line define by Eq. (1). If a set of points of the snake truly form a straight line, then the line segments between each set of two adjacent snake point would yield identical (c,m) values. This is the general idea behind the Hough transform that is used in the robotic vision literature to detect straight lines in images. Applying this principle to the entire $n \times m$ image is computationally very expensive. However, with the snake this process is greatly simplified. Here we look at the points preceding and super-ceding the current snake point to estimate a local line, and then compute the corresponding line invariant parameters. If a series of snake points form a common line, then their equivalent (c,m) parameters would cluster closely together.

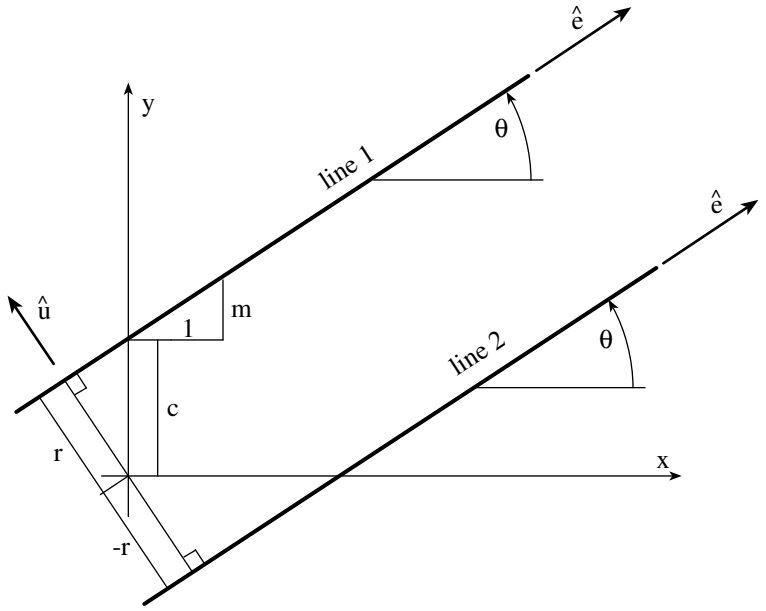

Figure 2. Illustration of the (r, θ) line parameters

In this report we are referring to the line invariant parameter space as the Hough space. However, we do not use the (c, m) parameters as shown in Eq. (1) due to the singularities in m as lines become vertical. Instead we use the parameters (r, θ), with r being the miss distance of the line to the origin and $\pi/2 < \theta \leq \pi/2$ being the line slope. These parameters are illustrated in Figure 2. The unit vector $\hat{\boldsymbol{e}}$ is the direction of the line. Note that $-\hat{\boldsymbol{e}}$ would model the same line. To avoid singularities through non-uniqueness of the line description, the line direction vector is defined such that it never points in the negative x axis direction. This corresponds to the slope being being limited to $\pi/2 < \theta \leq \pi/2$. The line direction vector is defined in terms of θ as

$$\hat{\boldsymbol{e}} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \tag{2}$$

The unit direction vector $\hat{\boldsymbol{u}}$ is defined to be orthogonal to $\hat{\boldsymbol{e}}$ and rotated from $\hat{\boldsymbol{e}}$ 90 degrees counter clockwise.

$$\hat{\boldsymbol{u}} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix} \tag{3}$$

The parameter r is the smallest distance between points on the line and the coordinate origin. Note that r is treated here as a signed parameter. The vector $r\hat{\boldsymbol{u}}$ provides the miss

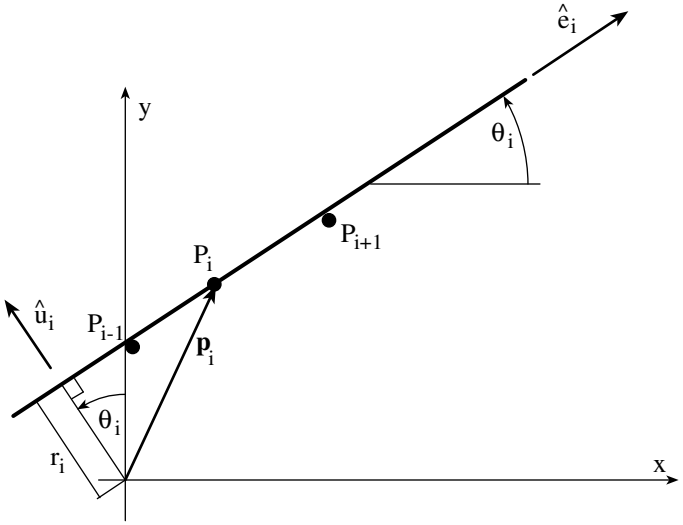

Figure 3. Computation of the (r,θ) line parameters

vector of the line to the origin. If the line lies on the positive side of $\hat{\boldsymbol{e}}$, (where positive is defined through the direction of $\hat{\boldsymbol{u}}$), then r is positive. This idea is illustrated in Figure 2 through lines 1 and 2 which only differ in the sign of their r value.

Given the image coordinates $\boldsymbol{p}_i = (x_i, y_i)$ of a snake point P_i, the corresponding Hough space parameters (r_i, θ_i) are computed as illustrated in Figure 3. Let d be the distance between the two neighboring points

$$d = \sqrt{(x_{i+1} - x_{i-1})^2 + (y_{i+1} - y_{i-1})^2} \tag{4}$$

The $\hat{\boldsymbol{e}}_i$ unit vector coefficients are then determined through

$$e_{i_x} = \frac{x_{i+1} - x_{i-1}}{d} \tag{5}$$

$$e_{i_y} = \frac{y_{i+1} - y_{i-1}}{d} \tag{6}$$

Since $\hat{\boldsymbol{e}}_i$ must not point in the negative x axis direction, the sign of e_{i_x} must be non-negative. If it is, then reverse the sign of both e_{i_x} and e_{i_y}. Note that the $\hat{\boldsymbol{e}}$ could also be computed using only $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$. However, using both neighboring points provides for a smoother local slope computation. The miss distance r_i is now computed as

$$r_i = \boldsymbol{p}_i \cdot \hat{\boldsymbol{u}}_i = -e_{i_y} x_i + e_{i_x} y_i \tag{7}$$

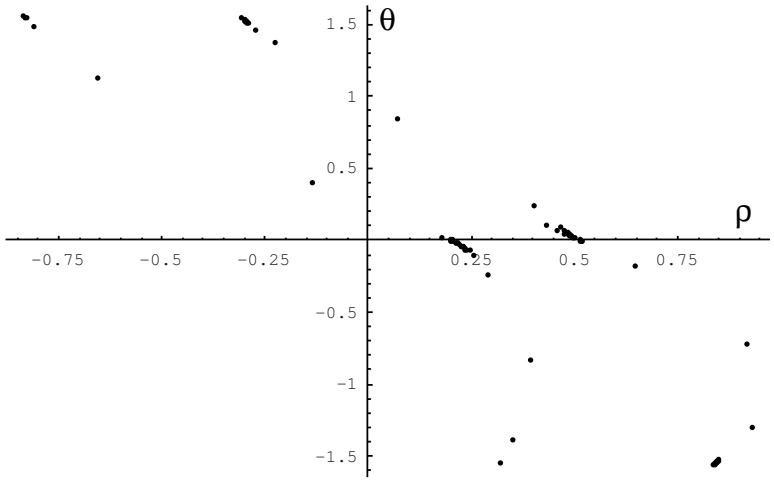

Figure 4. Hough Space Plot of Sample Box-Shaped Snake

while the line heading angle θ_i is given by

$$\theta_i = \tan^{-1}\left(\frac{e_{i_y}}{e_{i_x}}\right) \tag{8}$$

When numerically computing θ_i, be sure to use the `atan2()` function to avoid singularities near $\theta_i \rightarrow \pm\pi/2$.

Determining Line Point Clusters

Using Eqs. (7) and (8), we are able to quickly map all (x_i, y_i) snake points into corresponding Hough space (r_i, θ_i) points. The next task is to identify which groups of Hough space points belong to a common line. Figure 4 shows an example where a box shaped snake image has been transformed into the Hough space. Note that the miss distance r has been normalized here using the image width w to yield

$$\rho = \frac{r}{w} \tag{9}$$

For the results shown in Figure 4, two sides of the box are near vertical, while the other two sides are near horizontal. This results in two clusters of points that are near the $\theta = 0$ line, while the other two line point clusters are at either $\theta = +\pi/2$ or $\theta = -\pi/2$. Note that

as the angle θ rolls over from $\pi/2 \rightarrow -\pi/2$, or vice versa, then the sign of the miss distance r must also be reversed. When computing the nearness between two points, this roll-over behavior must be taken into account.

Note the interesting behavior of the line point clusters. They do not form tight groups of points, as might have been expected, but rather form drawn out chains of points. The reason for this has to do with how the snake line will chatter about the target edge. Figure 5 shows a close-up view of how the snake points behave near the target edge. As implemented, they will never perfectly settle down on the target edge line. They will always either try to push the snake point out if it is currently over the target, or pull it back it is no longer over the target. When computing the snake line segments about a current snake point $\boldsymbol{p}_i$, this minor chatter will cause some slight perturbation about the local straight line approximations about point $\boldsymbol{p}_i$. These perturbations are what is causing the Hough space clusters to be elongated into small chains of points.

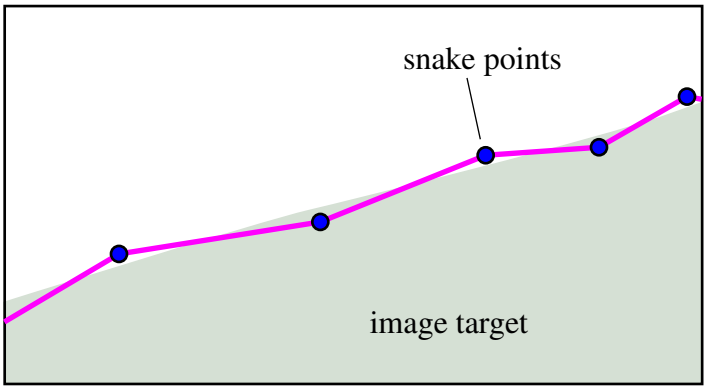

Figure 5. Close-Up View of Snake Point Edge Behavior

Because of this behavior, we cannot simply look at the RMS value of the distance between two Hough space points to determine if they might belong to a common line. For example, it is possible for two points at the opposite ends of the chain cluster to be relatively far apart, but they clearly belong to the same line point grouping. Figure 6 illustrates the logic that is applied to search through all the Hough space points to determine which points might belong to a common line.

The snake points are first sorted by their non-dimensional ρ values using a fast $N\log_2(N)$ HeapSort() algorithm.[5] The sorted ρ values are stored in the array `r_abs[]`, while keeping track of which snake point these values correspond to. If a slope angle magnitude is larger than 1.3 radians and $\rho > 0$, then the snake point ρ value is duplicated in the array `r_abs[]` as $-\rho$. The reason for this is to account for the roll-over behavior of the Hough space

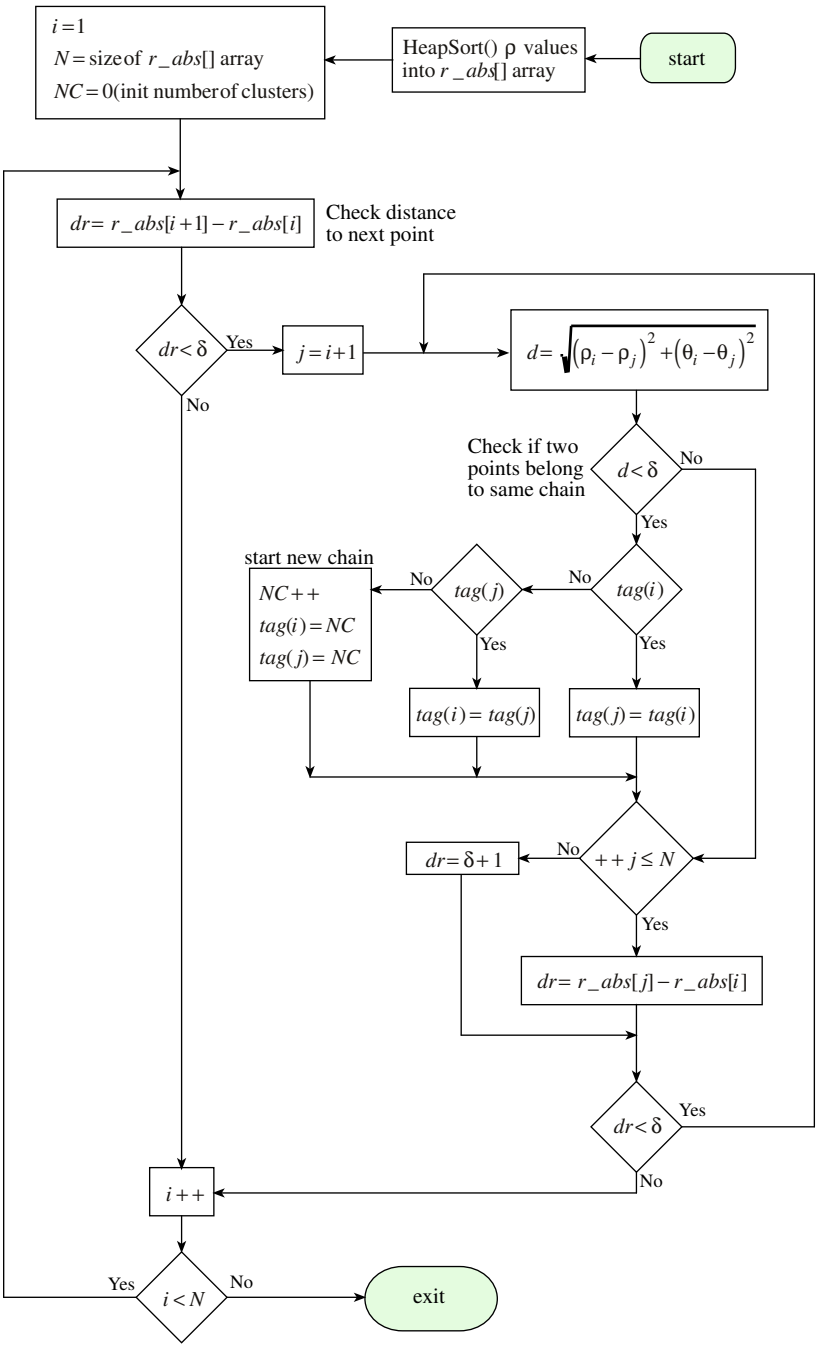

Figure 6. Flowchart Diagram of Common Line Point Cluster Identification

points near $\theta = \pm 90$ degrees. If $\theta > 1.3$ radians, then it is possible that the current snake point could be close to points near $-\rho$. This process will slightly increase the number of loops N that are performed later on to determine near snake points. However, the major advantage is that by keeping a signed ρ value (as opposed to simply taking $|\rho|$ to account for roll-over behavior), the ρ value will be more unique for each line cluster and make the cluster-finding algorithm quickly reject points that are far away.

After sorting the snake points by their ρ value, we loop over the N snake points to determine chain-clusters of near points. The counter variable *NC* keeps track of how many chain clusters have been found and is initialized to zero. First we compute the distance $dr > 0$ between the two points $p(i)$ and $p(j)$ in terms of the sorted ρ values. If dr is smaller than some threshold value δ, then it is possible that $p(j)$ could be a common chain cluster point. If not, we skip forward and increment the i counter. After initializing the inner loop counter j, we check if the point $p(j)$ has been assigned a chain tag ID number. If yes, then this point is already part of a chain and we skip forward to increment the j counter. If $p(j)$ has not been assigned an ID tag, then we compute the distance d between $p(i)$ and $p(j)$ in the Hough space. Note that the roll-over behavior of the Hough space points about $\theta = \pm 90$ degrees must be taken into account here. If $d < \delta$, then these two points belong to a common chain. If the current point has not been assigned an ID tag yet, then a new chain is formed. If yes, then the chain ID tag of point $p(j)$ is set equal to that of point $p(i)$. Next we increment the j counter and compute the value dr between the new `r_abs[j]` and `r_abs[i]`. If dr remains less than δ, then the j-loop is continued until this condition fails. At this point we know that all remaining snake points cannot be part of the current set of chain cluster points and we skip forward to the next $p(i)$ value.

Note that since the snake points have been sorted by their signed ρ values, we are performing the above cluster search not by looping through successive snake points, but rather by considering snake points in ascending order of their ρ value. If a difference in ρ values is large, we can immediately conclude that the current, and all later points, will not belong to this chain cluster. Also, note that it is very unlikely for this chain naming scheme to generate sub-chains that belong to a common larger chain. This avoids the need to introduce a more complicated scheme which would join chain ID tags which belong to a common chain.

As shown, this ends up being roughly a $N \log_2(N)$ operation. No assumptions have been made here as to the ordering of the snake points. If a line is partially obscured, this method will still identify points left and right of this intrusion that belong to a common line. Compared to an earlier generation N^2 method of identifying snake point clusters in Hough space, the presented algorithm is 3-4 times faster and more robust.[2]

Figure 7 illustrates the Hough space line grouping identification process as the algo-

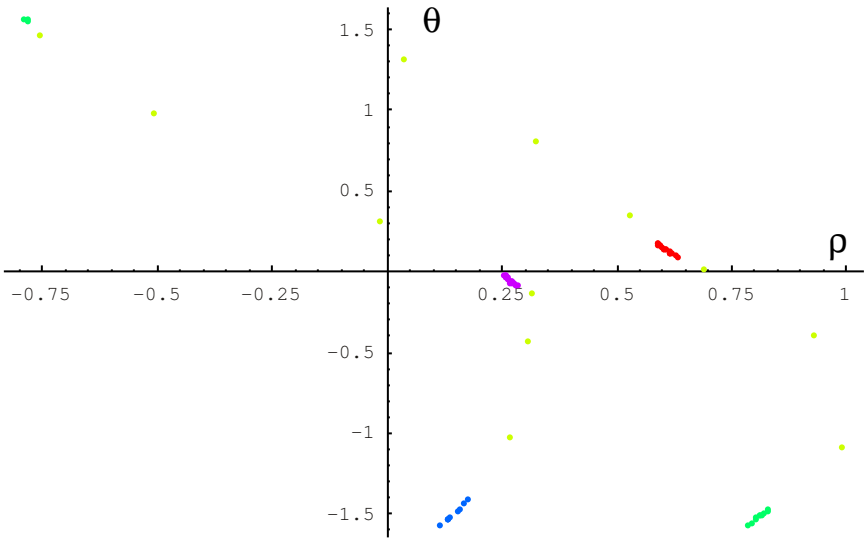

Figure 7. Groupings of Common Chain Clusters after the ρ-Sorting and Chain Identification Algorithm

rithm passes through the sorting and identification loops. A threshold of $\delta = 0.05$ is used here. The target box was partially obscured in this test, causing various erroneous line segments to appear in the data. Points belonging to a common chain are identified through a common color. Note that the roll-over effect is handled properly when comparing Hough space points. A near-vertical line is identified with slopes near both $\theta \rightarrow 90$ degrees and near $\theta \rightarrow -90$ degrees.

The smaller the δ value is, the less likely the algorithm will identify wrong points belong to a common line. However, at the same time we are also more likely to reject points that should belong to a common line.

After identifying chains of points in the Hough space, the number of points in each chain cluster is evaluated. The largest 4 chains are then identified for further processing. If the object is partially obscured, it is possible to have other small line segments in the image. By using only the 4 largest sets of snake points that form common lines, we are able to keep the algorithm robust to such small erroneous line segments.

Determining the Optimal Line Fits

Now that the snake points have been identified that belong to the 4 box sides, we need to evaluate the best line fits to these sets of points. A simple method would be to average the (r_i, θ_i) parameters belonging to a common line to obtain the mean parameter $(\overline{r_i}, \overline{\theta_i})$. Note that care must be taken when computing the average values due to the roll-over behavior of the Hough space points about $\theta = \pm 90$ degrees. These mean parameters would define the line that is later intersected with the other lines to determine the four polygon corner points. Implementing this averaging scheme, the resulting corner point predictions worked reasonably well. However, optimizing the (r, θ) values did not result in corner predictions which were as accurate as those obtained by computing the least-squares best line fits to the (x, y) snake points. The least-squared line fit provides a best *local* line fit to the (x, y) data points. The averaging of the (r, θ) parameters provides the best fit of the miss-distance r and slope angle θ. These states are defined in the proximity of the coordinate system origin. Thus, having a small error in θ could result in noticeable corner prediction errors if these corners are far removed from the origin. The local least-squares solution does not suffer from such sensitivities. Further, performing a full least-squares solution to the line segments is not computationally expensive.

To use the standard least-squares algorithm to determine a best line fit to the snake points (x_i, y_i), the line parameters must appear linearly. This means that we cannot use the non-singular (r, θ) parameters, since θ appears in trigonometric functions describing the (x, y)-space line. Instead, the standard (c, m) parameters are used shown in Figure 2. Here a line is defined as

$$y(x) = c + mx \tag{10}$$

However, note that this line description goes singular if the line is vertical. For near vertical lines, we can use the alternate line description

$$x(y) = c' + m'y \tag{11}$$

The optimal parameters (c, m) (in a least squares residual error sense) are found using the classical formula:

$$\begin{pmatrix} c \\ m \end{pmatrix} = \frac{1}{\Delta} \begin{pmatrix} \sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i \\ N \sum x_i y_i - \sum x_i \sum y_i \end{pmatrix} \tag{12}$$

where

$$\Delta = N \sum x_i^2 - \left(\sum x_i \right)^2 \tag{13}$$

The non-singular line parameters (r, θ) are then found through

$$\theta = \tan^{-1} m \tag{14}$$
$$r = c * \cos\theta \tag{15}$$

To avoid singular solutions in our algorithm, we use either of the above line descriptions in Eq. (10) and (11) when computing the least squares solutions. Let Δ' be given by

$$\Delta' = N \sum y_i^2 - \left(\sum y_i\right)^2 \tag{16}$$

If $\Delta' > \Delta$, then we compute (c', m') instead of (c, m) using

$$\begin{pmatrix} c' \\ m' \end{pmatrix} = \frac{1}{\Delta'} \begin{pmatrix} \sum y_i^2 \sum x_i - \sum y_i \sum y_i x_i \\ N \sum y_i x_i - \sum y_i \sum x_i \end{pmatrix} \tag{17}$$

If $m' > 0$, then the non-singular line parameters (r, θ) are computed using

$$\theta = \frac{\pi}{2} - \tan^{-1} m' \tag{18}$$
$$r = -c' * \cos\theta \tag{19}$$

If $m' < 0$, then the non-singular line parameter (r, θ) are computed using

$$\theta = -\frac{\pi}{2} - \tan^{-1} m' \tag{20}$$
$$r = c' * \cos\theta \tag{21}$$

By switching between the two line descriptions in Eqs. (10) and (11), we are able to obtain the optimal least-squares line fits to *any* line without singularities. The additional algebra required to achieve a non-singular solution is minimal.

Computing the Four Corner Intersection Points

At this point we have identified four lines l_i using the (r, θ) parameters. Before computing the 4 corners, these lines are ordered such that two near-colinear lines are never directly in sequence. This is done by checking the difference in slopes between the first line and the second and third lines. If the first and third line have a larger slope difference, then lines 2 and 3 are switched. This process is then repeated by comparing the slope differences

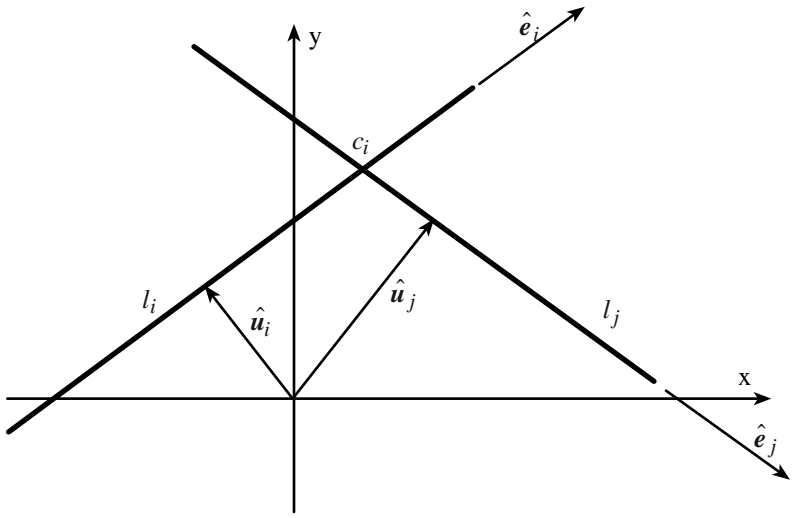

Figure 8. Intersection Point Between Two Lines

between the second line and and lines three and four. If the second and fourth line have a larger slope difference, then lines 3 and 4 are switched.

The final step is to intersect two lines l_i and l_{i+1} to compute the desired polygon corners. Two lines l_i and l_j are illustrated in Figure 8. Given the corresponding line parameters (r_i, θ_i) and (r_j, θ_j), the line vectors $\hat{\boldsymbol{e}}_i$, $\hat{\boldsymbol{e}}_j$, $\hat{\boldsymbol{u}}_i$ and $\hat{\boldsymbol{u}}_j$ are given by

$$e_{i_x} = \cos\theta_i \qquad e_{i_y} = \sin\theta_i \tag{22}$$

$$u_{i_x} = -e_{i_y} \qquad u_{i_y} = e_{i_x} \tag{23}$$

$$e_{j_x} = \cos\theta_j \qquad e_{j_y} = \sin\theta_j \tag{24}$$

$$u_{j_x} = -e_{j_y} \qquad u_{j_y} = e_{j_x} \tag{25}$$

Note that the $\hat{\boldsymbol{u}}$ vectors are always orthogonal to the $\hat{\boldsymbol{e}}$ vectors and rotated 90 degrees counter clock-wise. Let t_i and t_j be arbitrary scaling factors. The lines $l_i(t_i)$ and $l_j(t_j)$ are then given by

$$l_i(t_i) = \hat{\boldsymbol{u}}_i r_i + \hat{\boldsymbol{e}}_i t_i \tag{26}$$

$$l_j(t_j) = \hat{\boldsymbol{u}}_j r_j + \hat{\boldsymbol{e}}_j t_j \tag{27}$$

At the corner point $\boldsymbol{c}_i$ we must have $l_1(t_1) = l_2(t_2)$. This leads to the system of equations

$$\begin{bmatrix} e_{i_x} & -e_{j_x} \\ e_{i_y} & -e_{j_y} \end{bmatrix} \begin{pmatrix} t_i \\ t_j \end{pmatrix} = \begin{pmatrix} u_{j_x} r_j - u_{i_x} r_i \\ u_{j_y} r_j - u_{i_y} r_i \end{pmatrix} \tag{28}$$

This matrix equation can be solved for both t_i and t_j. However, to find the corner point $\boldsymbol{c}_i$, only finding t_i is sufficient. Solving this system of equations leads to the required scaling factor t_i^*.

$$t_i^* = \frac{1}{e_{j_x}e_{i_y} - e_{i_x}e_{j_y}} \left((e_{j_y}r_j - e_{i_y}r_i)e_{j_y} + (e_{j_x}r_j - e_{i_x}r_i)e_{j_x} \right) \tag{29}$$

The intersection point $\boldsymbol{c}_i$ between the lines l_i and l_j is then given by

$$\boldsymbol{c}_i = \begin{pmatrix} -e_{i_y} \\ e_{i_x} \end{pmatrix} r_i + \begin{pmatrix} e_{i_x} \\ e_{i_y} \end{pmatrix} t_i^* \tag{30}$$

This process is repeated to find the remaining three corner points.

Conclusion

A fast $N\log_2(N)$ algorithm is presented that takes the snake curve and finds the four dominant line segments. After intersecting the four lines, the polygon corners points are computed. Care was taken to obtain an algorithm which is free of singularities, yet is also fast to compute. The end result is a convenient method to detect 4-sided polygon corner points in a robust manner. Having the sides partially obscured does not affect the algorithm, as long as the target polygon sides are the four largest straight line segments of the snake. Further, the algorithm does not require the polygon corner point to actually be visible. Since its location is computed by intersecting two lines, the corner point itself can be hidden by another object. The algorithm could be extended to detect more than 4 line segments.

References

[1] Perrin, D. P. and Smith, C. E., “Rethinking Classical Internal Forces for Active Contour Models,” *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, December 2001, pp. 615–620.

[2] Schaub, H., Wilson, C., “Matching a Statistical Pressure Snake to a Four-Sided Polygon and Estimating the Polygon Corners,” Technical report version 1, Sandia National Labs, Feb. 24 2003.

[3] Schaub, H., “Statistical Pressure Snakes Based on Color Images,” Technical report, Sandia National Labs, 2003.

[4] Schaub, H., “Extracting Primary Features of a Statistical Pressure Snake,” Technical report, Sandia National Labs, Albuquerque, NM, 2003.

[5] Press, W. H., Teukolsky, S. A., Vetteling, W. T., and Flannery, B. P., *Numerical Receipes in C*, Cambridge University Press, 1992.

DISTRIBUTION:

1 Virginia Polytechnic Institute
Attn: Hanspeter Schaub
Aerospace and Ocean Engineering Department
228 Randolph Hall
Blacksburg, VA 24061-0203

1 MS 1125
Phil Bennett, 15252

1 MS 1125
Dan Marrow, 15252

1 MS 1125
Robert J. Anderson, 15252

1 MS 1003
Chris Wilson, 15211

1 MS 9018
Central Technical Files, 8940-2

2 MS 0899
Technical Library, 4916

2 MS 0619
Review & Approval Desk, 4916