Houghing the Hough: Peak Collection for Detection of Corners, Junctions and Line Intersections

William A. Barrett
william_barrett@byu.edu

Kevin D. Petersen

Follow this and additional works at: https://scholarsarchive.byu.edu/facpub

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/facpub/557

This Peer-Reviewed Article is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Faculty Publications by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Houghing the Hough: Peak Collection for Detection of Corners, Junctions and Line Intersections

William A. Barrett and Kevin D. Petersen
Department of Computer Science
Brigham Young University
barrett@cs.byu.edu, kdp@neptune.cs.byu.edu

Abstract

We exploit the Accumulator Array of the Hough Transform by finding collections of (2 or more) peaks through which a given sinusoid will pass. Such sinusoids identify points in the original image where lines intersect.

Peak collection (or line aggregation) is performed by making a second pass through the edge map, but instead of laying points down in the accumulator array (as with the original Hough Transform), we compute the line integral over each sinusoid that corresponds to the current edge point. If a sinusoid passes through $\geq 2$ peaks, we deposit that sum/integral into a new accumulator array - an array that has a direct one-to-one correspondence with the original image. Thus, "Houghing the Hough" identifies points that correspond to corners, junctions or line intersections in image space.

During initial peak collection, we include in the line integral only the most (locally) significant peaks while sifting out other (comparatively) weaker peaks from the current, as well as subsequent peak collections. This "contextual peak sifting" greatly reduces computation, the effect of noise and the occurrence of false positives.

Virtual line intersections (vanishing points, occluded corners, etc.) are detected as peaks without proximate edge support.

Results in real-world images show the technique performs well in identifying corners, junctions and intersecting lines in a variety of scenes containing man-made objects such as buildings, documents, etc.

1. Introduction

Detection of junctions, or groupings of line segments that meet at a single point, is important to image understanding in general and object recognition in particular. Junctions and line intersections contain more information than line segments alone - not only because they occur less frequently, but because they are formed from line segments and can often be used to group or link segments together to determine the shape of an object.

There has been much research into the detection of corner and junction points in images [1-4]. There are generally two approaches taken: either (1) junctions are discovered using flexible template matching and energy minimization to fit a junction model to the input signal directly [5-7] or (2) edges are first detected and then junctions are found at edge terminations and edge intersections [8-9], taking into account inflections in gradients and arcs. Recent improvements have been demonstrated in both approaches by using more robust statistical approaches to model the distribution of pixel values and edges.

These existing techniques (necessarily) employ local operations, and, as a result, do not exploit the coherence of global structures and (man-made) objects such as those contained in images of buildings, documents, urban scenes, etc., in such images, structures and objects of interest often form regular patterns that are manifested as intersecting line segments. Thus in such images, the detection of corners, junctions and line intersections could be used to automatically recover the two- or three-dimensional geometry of the scene, a subject of growing interest [10-15]. These types of images form the primary target for the methods presented in this paper.

Existing techniques [1-9] also have no way of detecting virtual junctions or line intersections (such as occluded corners and vanishing points) since there is no signal to match or extend. Davies [16] introduces an algorithm for finding the intersection of two lines where the intersection may be blunt or rounded ("occluded") using a generalized Hough Transform [17]. Edge pixels voting with this technique are required to vote using a lateral displacement toward the inside of the generalized shape, assuring that blunt or rounded corners are able to receive votes that would have otherwise been omitted.

One technique for discovering vanishing points uses a (Gaussian) spherical accumulator positioned at the optical center of the camera [18]. Votes are cast by reprojecting edges through the sphere. Vanishing points are then determined by (local) maxima. Subsequent work [19] divides the parameter space hierarchically to allow detection of both finite and infinite vanishing points.

Other Hough-like methods for detection of vanishing points include the use of a histogram of normalized areas of (bounded) quadrilaterals [20] and a Cascaded Hough transform [21], that successively divides the accumulator space into three subspaces. Multiple iterations on each subspace, result in the detection of various image features with each pass, including vanishing points.

0-7695-1272-0/01 $10.00 © 2001 IEEE
2. Finding Junctions in Hough Space

The techniques described in this paper make use of the Hough Transform [22,23] to identify candidate line segments. We then make a second pass through the Accumulator Array, A, to collect peaks (lines) into corners, junctions and line intersections.

Thus, while the Hough Transform has been used previously for detection of corners and other features in both hierarchical and cascaded implementations, the methods used in this paper are fundamentally different from previous approaches referenced above. Rather than extending or searching for local patterns, we exploit the global coherence of lines as they group naturally into (man-made) objects (windows, doors, etc.) defined by reasonably regular patterns of corners and junctions. This reduces the susceptibility to local noise and false positives while improving the accuracy and consistency of detected junctions including virtual junctions (i.e. vanishing points and occlusions, for example).

However, we wish to emphasize that the junction points detected by our system could very well be used as input candidates to any of the previous techniques for junction detection, thereby greatly reducing the search space and the computation required for localization. In addition, local search techniques could also be used to help perform contextual classification of junctions.

2.1 Houghing the Hough: The Basic Idea

We begin by computing the Hough Transform and forming peaks in the Accumulator Array, A. Then we make a second pass through the edge map, E, to form a new accumulator array, P, that corresponds, point-for-point with the original image. In this second pass, we compute the line integral over some neighborhood \( r \) of each sinusoid in A that corresponds to the current edge point in E and deposit that sum into P (Equation 1), where \( \Gamma = x \cos \theta + y \sin \theta \), the usual parameterization of the Hough space.

\[
P(x,y) = \int_{r} A(r, \theta) d\theta
\]  

(1)

Thus we retrace our (sinusoidal) “tracks” in A, summing rather than accumulating, and deposit that sum into a new array, P. Peaks in P correspond to corners, junctions or line intersections. We call this “Houghing the Hough” because after collecting points along a line into a peak, we collect peaks along a sinusoid into a point.

Note that in Figure 1, lines \( l_2, l_5 \) and \( l_8 \) in the edge map, E, correspond to peaks \( p_2, p_5 \) and \( p_8 \) in the Accumulator Array, A. The line integral of the sinusoid passing through \( p_2, p_5 \) and \( p_8 \) forms a (local) maximum sum/peak in P that identifies the junction point, \( J_{258} \).

2.2 Forming the Accumulator Array, A

We compute the gradient magnitude, G, by applying standard (3x3) Sobel or Prewitt kernels to the original image. We also apply a standard 3x3 Laplacian operator to the original image to create a binary zero-crossings image, Z. We then mask G with Z \( G \cdot AND \cdot Z \) to thin edges and minimize noise in the resulting edge map, E (Figure 2, right).

Figure 1. Maximum line integral over the sinusoid passing through peaks, \( p_2, p_5 \) and \( p_8 \), in Accumulator, A, identifies the junction point, \( J_{258} \), of lines \( l_2, l_5 \) and \( l_8 \) in the original image space.

Figure 2. Gradient magnitude, G (left). Edge map, E, after masking G with Laplacian zero-crossings.

Figure 2. Gradient magnitude, G (left). Edge map, E, after masking G with Laplacian zero-crossings.
We apply two different weighting schemes when casting votes in \( A \):

1. We weight each vote by the magnitude of edge points in \( E \).
2. We also weight each vote by the orientation of the edge, \( \theta_e \), since diagonal or sloped lines should cast more votes than they do on a discrete grid.

Thus accumulation in \( A \) proceeds as follows:

\[
A(r, \theta) = \frac{E(x, y)}{\max \left( |\sin \theta_k|, |\cos \theta_k| \right)} \tag{2}
\]

for \( \theta = \theta_k \pm \delta \theta \) to narrow the window of voting in \( A \).

When casting votes, sinusoids almost always pass between two discrete pixel coordinates. To account for this we also employ a fractional voting scheme where the amount of the vote received by a pixel is proportional to its distance from the sinusoid.

Finally, to reduce residual noise and facilitate peak detection we apply a Gaussian matched filter to \( A \). Peaks are identified as local maxima using a simple hill-climbing algorithm.

### 2.3 Forming the second Accumulator Array, \( P \)

\( P \) is formed by making a second pass through \( A \) to find sinusoids with multiple peaks, as described in the Houghing the Hough algorithm below.

**Houghing the Hough Algorithm**

**Input:**
- \( E \) (Edge map)
- \( A \) (Hough Space accumulator of \( E \))

**Output:**
- \( P \) (Integral image, equal in dimension to \( E \);
  Collects peaks = junction points)

**Algorithm:**
1. for \( E(x, y) \neq 0 \) (retrace sinusoidal tracks)
2. \( \Sigma \leftarrow 0 \); (init. current line integral)
3. \( N \leftarrow 0 \); (init. number of peaks to 0)
4. for \( \theta = \theta_k + \delta \theta \) to \( \theta_k - \delta \theta \) (sweep whole space)
5. \( r \leftarrow x \cos \theta + y \sin \theta \); (to find all peaks)
6. if \( A(r, \theta) \) is a peak
   7. \( \Sigma \leftarrow A(r, \theta) \); (increment line integral)
   8. \( N++ \); (increment # of peaks)
   end if
end for
9. if \( N > 1 \) (junction only if \( N \geq 2 \))
10. \( P(x, y) \leftarrow \Sigma \); (deposit line integral in \( P \))
end if
end for

Step 1. indicates that we will form line integrals corresponding only to existing edge points, since junctions can certainly be expected to fall on (or near) edge points. And it turns out that this is an effective procedure for detecting such junction points. However, an extremely useful variant of step 1. is to form line integrals corresponding to all \( E(x, y) \) in order to detect virtual junctions such as occluded corners and vanishing points, as will be shown later.

Note also, in step 4., that we cannot limit \( \theta \) to \( \theta_k \pm \delta \theta \) as we did when \( A \) was formed, or else we might miss a peak in the sinusoidal path, and thereby "miss the whole point" of the algorithm.

Steps 6. And 7. simply add to the tally the amplitude of all peaks encountered in the path, as we would expect. However, again, a very useful variant is to be selective about which peaks we add to the sum in the current, as well as subsequent collecting operations. The importance of this variation is described in detail in section 2.4, below.

And, of course, we only have a junction if we intersect more than one peak (steps 9. And 10.). In addition to the constraint that a single sinusoid must pass through at least 2 peaks for it to be considered a corner or junction, we require that these peaks be sufficiently far apart, (\( \geq 5^\circ \)). Peaks in closer proximity than that are considered noise or redundant.

Since the array \( P \) has a direct one-to-one correspondence with the original image space, finding the peaks (local maxima) in \( P \) is exactly equivalent to finding the corresponding corners, junctions or line intersections in the original image. Clearly the peaks with the highest amplitude (largest line integral) in \( P \) correspond to the sinusoid with the greatest number and/or highest amplitude peaks on it (from \( A \)) which, in turn, correspond to the most dominant junctions and line intersections in the original image.

### 2.4 Contextual Peak Sifting

During initial peak collection, we include in the line integral only the most (locally) significant peaks while sifting out other (comparatively) weaker peaks from the current, as well as subsequent peak collections. This "contextual peak sifting" greatly reduces computation, the effect of noise from \( A \) and the occurrence of false positives in the detected junctions.

An algorithm for extending the Houghing the Hough technique to accomplish contextual peak sifting is presented below. As we re-traverse each of the paths of the sinusoids that created the Hough Space, we create a list of peaks encountered in step 7. This list of peaks is sorted based on peak magnitude in step 8. and used to eliminate weaker peaks based on locality in steps 16 and 17. If a weaker peak is found lying on the sinusoid in
Contextual Peak Sifting Algorithm

Input:
- E [Edge map]
- A [Hough Space accumulator of E]
- S [Sifting Window Size]

Data Structures:
- L [Local peak list]
- P [Integral image, contains junctions]
- A [Updated accumulator (after sifting)]

Algorithm:
1. for (x,y) ≠ 0 {retrace sinusoidal tracks}
2. L ← null; {init. local peak list}
3. for θ = -π/2 to π/2 {sweep whole space}
4. r ← x cos θ + y sin θ; - to find all peaks
5. if A(r, θ) is a peak {add peak coords. to L}
6. end if
7. end for
8. sort L by peak height; {sort L descending order}
9. N ← 0; {init. peak counter}
10. Σ ← 0; {init. curr. line integral}
11. C ← first peak in L {init C to first peak in L}
12. while C ≠ null {process all peaks in L}
13. Σ += A(C.r, C.θ); {increment line integral}
14. N++; {increment # of peaks}
15. for peaks K < C in L {consider all peaks < C}
16. if K in window S of C {consider next peak in L}
17. remove K; {remove all peaks within S of C from L, A, and P}
18. end if
19. C ← next peak in L {consider next peak in L}
20. end while
21. if (N > 1) {deposit integral in P}
22. end if
23. end for

Virtual Juncions

We can also apply these algorithms to the detection of virtual junctions such as missing or occluded corners or vanishing points, which are a special case of a virtual corner. To locate virtual junctions within an image we simply consider all pixels in the original image as edge pixels. Since the image is digitized, that is, executed as if all pixels were of non-zero value.

3.1 Occluded Corners and Vanishing Points

Corners become occluded in many day-to-day settings: a book covering a corner of paper, a bush obscuring the corner of a building, etc. If sufficient edge support exists from two or more line segments feeding into the corner or point of occlusion, a strong peak, and hence a junction, will be registered in P.

Figure 3 shows the lower left and the upper right corners of object b occluded by both objects a and c. However, due to the extent of the visible portion of the edges leading to those corners, the positions of the corners are still detected and displayed over the occluding objects a and c. Note that virtual corners also appear in the darker background areas for the same reason. The upper corner of a which overlaps b is also detected as a virtual corner because of the weakness of the edge strength between a and b. Virtual corners can be distinguished from real corners using local operators such as described in [5-9].

We use such an operator, but with the added efficiency of knowing which directions to probe since we know the direction of the lines feeding into the junction -- they are given by the peaks in A that define the junction.

Junctions are classified as "real" if they have proximate edge support for two or more line segments feeding into
the junction. Pseudo junctions can usually be detected as having proximate edge support for only one line segment.

On the other hand, virtual junctions have no proximate edge support, and can often be distinguished from each other based on the surrounding edge and pixel data and the distance from the edge support. For example, vanishing point junctions would usually be much further away from the edge data that give rise to it than would occluded corners.

Figure 3. Detection of virtual (occluded) corners at the lower left and upper right of b.

Figure 4. Detection of the Vanishing point as a virtual junction.

4. Results and Discussion

We investigate the performance of the algorithm on a variety of images, both synthetic and real, with both real and virtual junctions.

Figure 5 shows a synthetic image with Gaussian noise added. Real junctions, those with line support from the edge image in two or more directions, are shown as white boxes. The technique successfully locates major junctions of most shapes. False positives appear in the rotated square and triangle due to influence of continuous lines from other shapes present in the image, namely the rectangle and circle. Two corners on the small L-type shape were missed due to the length of the shorter supporting line segment. Without this line segment, the two omitted corner points do not receive the second vote necessary to constitute a junction. These types of problems can be easily remedied when each shape is considered independent of neighboring shapes.

Figure 5. Junctions (white squares) detected on a synthetic image.

The algorithm successfully identifies all major line intersections (shown with black dots) within the piece of the document shown in Figure 6. One line at the bottom of the bold text is considered erroneously, causing intersections to occur in the header where that line meets other existing lines. Other falsely identified junctions include those points where continuous line data meets real existing line data. These points can be eliminated using traditional junction finding techniques on the candidate intersections.

Figure 6. Junction points (shown in black) in a gridded document.
Virtual corners in real world images (Figure 7) can successfully be detected as well. Corners of the mouse pad are rounded and have been occluded. Taking the 4 maximum integrals of the junctions classified as "virtual," reveals the true corners of the mouse pad, shown here as white box outlines. While other virtual intersections were present in this image as in the previous synthetic image of this nature, these points can be and were eliminated using a threshold based on the value of their corresponding integral.

Figures 8b-e show corners and junction points detected from the building in Figure 8a by computing sinusoidal line integrals for all points in the original image space (i.e. both real and virtual).

Figure 8b shows the corners as centers of gravity of regions without using the contextual sifting window. Corner point patterns appear non-uniform due to excess data from similar orientation lines found by a simple hill-climbing algorithm on the Hough accumulator. Lines of this nature should be sifted out, allowing the more dominant lines to influence corner location.

In Figure 8c peaks are entirely eliminated from past, present, and future sinusoid integral calculations, using the contextual sifting window, leaving only the dominant direction lines. Corner point locations are much more uniform and noticeable patterns of dominant line directions develop.

In Figure 8d junctions were analyzed for line support from the edge image in order to determine 0 (virtual), 1 (pseudo), or 2 (actual) line junctions. Pseudo and actual intersections are shown with brighter junctions indicating stronger corner information, or higher sinusoidal integral. Darker junctions indicate lower integral values. Figure 8e displays only actual junctions having 2 or more line support from the edge image.
Vanishing points have also been successfully identified using this technique. Figure 9 shows a building scene where a vanishing point is included within the image space. The virtual point of highest integral magnitude is shown with the lines that contributed to its integral value. Again, while the algorithm did produce other virtual points, including points lying near the vanishing point shown, we appeal to the integral values of the virtual points in order to select the best candidate for the vanishing point of the scene.

Figure 9. Building with vanishing point detected shown with contributing lines.

We have also attempted to use the presented technique on more "organic" objects and scenes. Figure 10 shows a mountain range with real junctions as detected by the algorithm. While our approach is not meant for this type of image content, the results are surprisingly decent. Junctions are located at the tops of the mountain range at likely candidate points, as well as throughout the body of the mountain. Future work could be done to improve performance on data of this type. In fact, our approach could be used to detect candidate points for more established junction detection and localization algorithms.

5. Conclusion

We have shown the Houghing the Hough algorithm for peak collection and junction detection to be robust across a variety of images and uses. This technique aids in the discovery of junctions, both existing and virtual, where supporting edge information may or may not be present, including the special cases of vanishing points and occluded corners. While the results are promising, the algorithm certainly does not replace existing techniques for corner and junction finding, or for vanishing point detection, but rather can be used to complement them by supplying candidate junction points that are based on global structures and regular patterns, while minimizing the processing associated with computationally exhaustive searching and matching.

The algorithm demonstrates weakness in naturally occurring, rather than man-made objects, since it is based upon the Hough Transform for line detection. However, at the same time, this is the major strength of the approach: the ability to capture regular, high-level line-based patterns through a simple process of peak collection and contextual sifting in a way that is robust to noise while exploiting the global coherence found in these kinds of images. And to top it off, junctions such as occluded corners and vanishing points are virtually free.

Future improvements to the implementation would include more efficient and accurate ways of classifying junctions discovered using the amount of supporting
evidence shown by the original or edge images. In addition, we envision an interactive framework where regions of interest (buildings, roads, etc.) could be "rubber-banded" from an image, thereby narrowing the context and minimizing the cross-talk otherwise caused by pseudo-intersections with unrelated lines. In fact, such a framework could make this an attractive tool for extracting the three-dimensional geometry of these structures.

References