Hough Transform for Directional Orientation

Ivan Papusha, Matthew Ho
Stanford Department of Electrical Engineering
EE368 Spring 2010

***Abstract*—In this work, we propose several direct Hough-space and image-space methods for detecting vanishing points in an image, with an eye towards embedded implementation in an Android mobile device. In addition, we propose and implement an exhaustive scheme for calculating a best-case estimate of the vanishing points, given that the number of vanishing points is known ahead of time.**

I. Introduction

There has been increased interest in image processing on a mobile platform in recent years. Due to consumer demand and advances in processor technology, computationally demanding applications previously untenable are becoming more and more ubiquitous. For example, image analysis and processing is one of the primary drivers of GPUs on mobile devices.

A mobile platform optimized for image processing lends itself well to a large number of consumer applications. Our objective for this project is to investigate one of these applications: detecting vanishing points in an image, which might be used in a vision-based system for, e.g., robot localization and 3D environment extraction. A vanishing point is the apparent convergence of parallel lines under perspective distortion. As human environments are characterized by straight lines and orthogonal edges, we believe that our project can serve as a basis for a wide range of robust indoor localization schemes.

In Section II we discuss the motivation and related work. Section III details our implementation for detecting one vanishing point, and discusses the issues involved with multiple vanishing points. Finally, Section IV gives the experimental results, and hints at further research.

II. Related work

Zhang et.al. [1] make the observation that human-made indoor and outdoor environments, e.g., buildings and hallways, are endowed with a large number of structured and orthogonal lines, which define the image perspective and can act as features for building recognition. Jung et.al. [2] propose direct Hough-space methods for detecting rectangles in an image by considering relations among the Hough peaks imposed by the underlying geometry of a rectangle. Cantoni et.al. [3] describe a parameter-space Hough procedure, effectively a higher-order Hough transform, which they use with limited success to count the number of vanishing points in an image. Finally, Bosse, et.al. [4] successfully use vanishing points to navigate a Pioneer robot in an indoor environment.

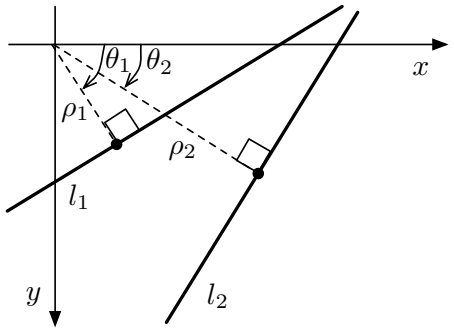

Fig. 1. Illustration of Hough parameters as they relate to lines in image space. Here line l_1 is uniquely identified by its perpendicular to the origin, which has length ρ_1, and angle from x-axis given by θ_1. Similarly, for l_2.

III. Directional orientation

A. Hough transform and thresholding

The Hough transform of a binary image is given by a binning operation. Associated with each pixel (x, y) is a continuous set of all possible lines of infinite extent, which could go through that point. Such lines are parameterized by

$$\rho = x\cos(\theta) + y\sin(\theta).$$

Figure 1 illustrates this setup. Without any loss of information we can take θ to range over $[-\pi/2, \pi/2]$. If the original image has width W and height H, then ρ can range over $[-\rho_{\text{max}}, \rho_{\text{max}}]$, where $\rho_{\text{max}} = \sqrt{W^2 + H^2}$. The Hough space is defined by a finite rectangle $[-\rho_{\text{max}}, \rho_{\text{max}}] \times [-\pi/2, \pi/2]$, which is split into rectangular bins. For each nonzero point (x, y) in the original binary image, the Hough transform draws a "sinusoid" in Hough space by incrementing the bin counts along the discretized curve $\rho = x\cos(\theta) + y\sin(\theta)$ by one. Because the bins are each labeled by a set of parameters (ρ, θ), these in turn correspond to possible lines in image space. Conversely, if several points in image space lie along a line, the bin count for the (ρ, θ) corresponding to the line through these points will be high. Peaks in the the Hough domain correspond to a high likelihood of a line in the image domain. The Hough procedure is readily extended to gray-level images, where the bins are incremented in proportion to the gray-level value of the pixel in original image space.

For the purposes of detecting strong lines using the Hough transform, it is profitable to first preprocess the image by finding the edges. There is extensive literature on the problem of detecting edges in an image. See, e.g., Gonzalez et. al. [5].

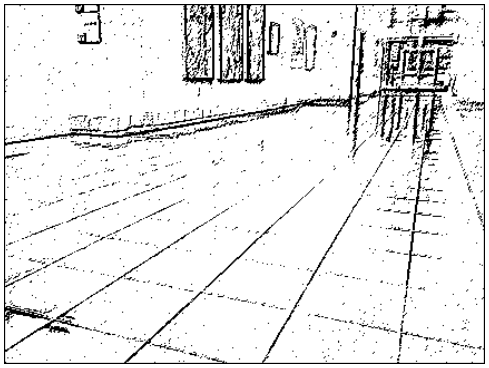

Fig. 2. Typical thresholded, horizontal and vertical differences edge map as a preprocessing step before Hough transform. Here, edges between the floor tiles guide the eye to a visible vanishing point at the top right, and an invisible vanishing point off-image and to the left. The vertical wall edges and paintings also hint at a vanishing point at infinity. Original image displayed in Figure 4

While Canny and Sobel methods are typically more robust, we found that simple horizontal and vertical differencing sufficed for our application. A typical thresholded edge map is shown in Figure 2. For speed and implementation on the mobile Android phone, all processing was done on the single blue channel, as using the true average gray-level values gives comparable results with an unnecessary extra averaging step.

The binary Hough procedure takes time proportional to the number of active edge pixels in the thresholded edge map, thus it is helpful to de-noise the edge detected image from stray pixels that ostensibly do not belong to any edges by pre-blurring and morphologically closing the edge map.

B. One vanishing point

The Hough transform procedure generates a set of n Hough peaks $(\rho_1, \theta_1), (\rho_2, \theta_2), \ldots, (\rho_n, \theta_n)$ and their bin counts, which we interpret as weights $w_1, w_2, \ldots, w_n$. A large weight w_k, for instance, corresponds to a large Hough bin count at point (ρ_k, θ_k), giving credence to the line parameterized by

$$\rho_k = x\cos(\theta_k) + y\sin(\theta_k),$$

in proportion to its length in the original image. Supposing that all n lines are to intersect at the same point, or nearly the same point (x_0, y_0), we can extract that intersection point by solving the approximation problem:

$$\begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ \cos(\theta_2) & \sin(\theta_2) \\ \vdots & \vdots \\ \cos(\theta_n) & \sin(\theta_n) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \approx \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_n \end{bmatrix} \tag{1}$$

In other words, we find the best-fit sinusoid parameterized by (x_0, y_0) directly in the Hough domain by forming a matrix $A \in \mathbb{R}^{n\times 2}$ and vector $\rho \in \mathbb{R}^n$, and seeking to make the difference $\|Ax - \rho\|$, $x \in \mathbb{R}^2$ as small as possible. Formally, x is a solution to the optimization problem

$$\min_{x\in\mathbb{R}^2} \|Ax - \rho\|, \tag{2}$$

which is convex for any L_p-norm ($p \geq 1$) and is readily solved for several norms and choices of p:

- L_2 *norm*: When A is full-rank, the optimal x satisfies the normal equations

  $$A^T Ax = A^T \rho,$$

  that is, $x = (A^T A)^{-1} A^T \rho$ minimizes the objective in (2) using the Euclidean distance metric. If the two columns of A are given by $a_1, a_2 \in \mathbb{R}^n$, then $A^T A$ is 2×2 symmetric with elements:

  $$A^T A = \begin{bmatrix} a_1^T a_1 & a_1^T a_2 \\ a_2^T a_1 & a_2^T a_2 \end{bmatrix}$$

  for which there is an explicit formula for the inverse, thus the optimal x is

  $$\begin{aligned} x &= (A^T A)^{-1} A^T \rho \\ &= \frac{1}{\det(A^T A)} \begin{bmatrix} a_2^T a_2 & -a_1^T a_2 \\ -a_2^T a_1 & a_1^T a_1 \end{bmatrix} \begin{bmatrix} a_1^T \rho \\ a_2^T \rho \end{bmatrix} \end{aligned}$$

- L_1 *norm*: In this case, no closed-form solution for x can be given, however the problem is readily reformulated as an inequality constrained linear program, for which fast interior point iterative methods (e.g., log-barrier, primal-dual) exist. See, for instance, Boyd et.al. [6]
- *Quadratic W-norm:* The weighted least-squares setting, where the quadratic norm $\|\cdot\|_W$ induced by the $n \times n$ matrix $W = \mathbf{diag}(w_1, w_2, \ldots, w_n)$ is defined by $\|x\|_W := (x^T W x)^{1/2} = \|W^{1/2} x\|_2$ is equivalent to a least-squares L_2-norm problem. Here, one interprets the weights w_j as the degree to which one should incorporate the line parameterized by (ρ_j, θ_j) as contributing to the intersection point. The Hough transform bin counts serve as the weights, with high bin counts corresponding to large weights.

In practice, the L_2 and W-norm solutions are easiest to implement on the Android phone, although in our experiments the L_1-norm solution usually produced better results. Evidently, the L_1 solution places much less emphasis on outlying line intersections than the quadratic norms, which allows for more robust intersection detection. (See Figure 4.)

In the quadratic norms, the matrix inversion $(A^T A)^{-1}$ is characterized by the size of the determinant, $|\det(A^T A)|$, which if small, suggests poor conditioning. Looking back at (1), this corresponds to a low-rank condition, where all the θ_j values are close to each other, meaning that all the extracted lines are approximately parallel, i.e., the vanishing point is at infinity.

Just as the (ρ, θ) parameterization of the Hough transform avoids numerical instabilities for vertical lines in the image domain, this numerical problem of parallel lines can be avoided by considering a reparameterization of the image domain, where intersection points at infinity are projectively mapped to a sphere. For instance, Bosse, et.al. [4] work in an omnidirectional projective geometry, which means that vanishing points at infinity are gracefully handled from the start.

Our application does not require such precise localization, however, we choose to remain in the more familiar linear regime, and simply output a direction-to-vanishing point when it is sufficiently far off-screen, rather than a precise estimate of its location.

C. Multiple vanishing points

In general an image contains multiple vanishing points. Depending on the major lines in the image, that number, K, can be quite large, but in the highly structured indoor environments that we consider, K is usually no more than 2 or 3. If one knows K ahead of time, an easy first step is to consider all pair-wise intersections between the n lines, and cluster them into K clusters. We used a standard K-means procedure described by Algorithm 1.

Algorithm 1 K-means (intersections)

Input: number of vanishing points: K,
input lines: $(\rho_1, \theta_1), \ldots, (\rho_n, \theta_n)$,
number of pairwise intersections: $N = n(n-1)/2$,
pairwise intersections: $\{(x_j, y_j)\}_{j=1}^{N}$

```
for j = 1 to N do
    label_j ← random label from {1, ..., K}.
end for
repeat
    for k = 1 to K do
        V_k ← {j | label_j = k}
        centr_k ← arg min_{(x,y)∈R^2} Σ_{j∈V_k} ||(x_j, y_j) − (x, y)||
    end for
    for j = 1 to N do
        label_j ← arg min_{k=1,...,K} ||(x_j, y_j) − centr_k||
    end for
until label_j's are stationary
return {centr_k}_{k=1}^{K} as the K vanishing point estimates.
```

For the case $K = 1$, as illustrated in Figure 4, the centroid of the all-pairs intersection points of the vanishing lines (Algorithm 1, $K = 1$) do not in general coincide with the best-estimate intersection achieved by solving the full optimization problem (2), which is initially surprising, but has a ready interpretation: whereas (2) "wiggles" the lines a little so as to find a common point of intersection, Algorithm 1 "wiggles" the intersection points so as to get them close to some purported vanishing point. Evidently, by computing the all-pairs intersections, one loses all the line information from which they were generated, thereby reducing vanishing point estimation accuracy. This makes sense, because the vanishing points in an image are characterized by vanishing *lines*, which happen to intersect at vanishing points by the rules of projective geometry, not by all possible line *intersections*, some of which might happen to be vanishing points. Clearly, the first-class object to consider is a *line* (point in Hough space), which belongs to a vanishing point, not a *point* (sinusoid in Hough space), which belongs to a vanishing line.

When the number of vanishing points K increases from 1, the K-means procedure described in Algorithm 1 is able to cluster the line intersections, with clusters giving a good direction-to-vanishing point, especially if the vanishing point is off-screen. However, even when a vanishing point is visible in the original image, the cluster centroids are adversely "pulled" away from the purported vanishing point by stray lines intersecting with existing lines, and thus by themselves provide poor estimates of the vanishing point location. The stray lines create extra pairwise intersections, changing the cluster centroids' locations.

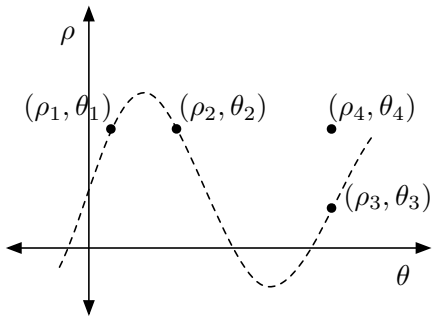

Fig. 3. Visualization in the Hough domain. Points $(\rho_1, \theta_1), \ldots, (\rho_4, \theta_4)$ in Hough space each correspond to lines in image space. Here, $(\rho_1, \theta_1), \ldots, (\rho_3, \theta_3)$ lie on the same sinusoid, whose best fit parameters (x, y) in accordance with Equation (3) we interpret as the mutual intersection of the three lines in image space. (ρ_4, θ_4) is not on the same sinusoid, and hence does not give credence to a mutual intersection.

As a result, we really wish to classify the n lines (corresponding to peaks in the Hough transformed-image), rather than the $n(n-1)/2$ intersection points, as belonging to one of K vanishing points. The main observation in Algorithm 2 is that a set of lines characterized by $(\rho_1, \theta_1), \ldots, (\rho_n, \theta_n)$ coordinates in the Hough domain all intersect at the same point in the image domain if and only if the best-fit sinusoid

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{3}$$

in the Hough domain, with parameters x and y, actually goes through the points $(\rho_1, \theta_1), \ldots, (\rho_n, \theta_n)$ See Figure 3.

In principle, if the number of lines n is small (~ 10), and we expect that there are exactly K (~ 2 or 3) vanishing points in the image, one can run an exhaustive calculation, which is tractable even on a mobile device. Algorithm 2 searches all $\binom{n}{K}$ ways to assign the n lines to K vanishing points. In each case, the best assignment is given by the smallest least-squares error achieved by all the pairings.

IV. Experimental results

Despite the fact that Best-L_2 is exhaustive, its execution time is small in comparison with the Hough transform line-extraction. At the same time, its shortcoming is in relying on a fixed value K of vanishing points for which to search, which is in general not known ahead of time. As shown in Figure 4, in which we use $K = 2$, Best-L_2 finds both the vanishing points beautifully. The two best-fit sinusoids for that image are illustrated in Figure 5. However, when the assumption

Algorithm 2 Best-L_2 (exhaustive)

Input: number of vanishing points: K,
input lines: $(\rho_1, \theta_1), \ldots, (\rho_n, \theta_n)$.

$\mathcal{L} \leftarrow \{1, 2, \ldots, n\}$
$best\_err \leftarrow \infty$
$\mathcal{V} \leftarrow \emptyset$
for all pairwise disjoint partitions $S_1, S_2, \ldots, S_K \subseteq \mathcal{L}$, with $S_i \cap S_j = \emptyset, i \neq j$, and $\bigcup_{j=1}^{K} S_j = \mathcal{L}$ **do**
  for $j = 1$ to K **do**
    $A_j \leftarrow$ rows S_j of A in equation (1).
    $\rho_j \leftarrow$ rows S_j of ρ in equation (1).
    $x_j^* \leftarrow \arg\min_{x \in \mathbb{R}^2} \|A_j x - \rho_j\|_2$
  end for
  $cum\_err \leftarrow \sum_{j=1}^{n} \|A_j x_j^* - \rho_j\|_2$
  if $cum\_err < best\_err$ **then**
    $best\_err \leftarrow cum\_err$
    $\mathcal{V} \leftarrow \{x_1^*, x_2^*, \ldots, x_K^*\}$
  end if
end for
return $\mathcal{V}$ as the K vanishing point estimates.

$K = 2$ is violated, as in Figure 6, Best-L_2 suffers from "overfitting." Incidentally, in both cases, where the vanishing point is visible and on-screen, the L_1 method finds the "main" vanishing point with no prior assumptions on K. Note that in all cases, the Hough line fidelity, which is heavily dependent on thresholding parameters in the edge detection and Hough peak finding steps, has a big impact on the extracted vanishing point.

Table I shows typical timing for a 320×240 downsampled image. Here we see that the majority of the time is taken by our simple implementation of the Hough transform. Fernandes et.al. [7] suggests a heuristic for an improved Hough transform voting scheme, and get close to real-time performance, which we expect can be achieved for the small images we consider, especially if we use Android's Native Development Kit (NDK) to speed up the operations on the mobile device. In addition, when the number of vanishing points is low, Table I suggests that even the exhaustive Best-L_2 is sufficiently fast for real-time embedded implementation. We find that smaller image sizes and more coarse Hough domain discretizations give even better performance, at the expense of decreased line detection accuracy. A potential area for further research is to consider Hough-space methods like Best-L_2 which deal directly with a larger number of poorly localized lines.

V. Conclusion

The methods presented in this project are able to extract vanishing points in structured indoor environments, which can be used to reconstruct a partial 3-dimensional environment map. After tuning thresholding parameters, and relevant Hough transform resolutions, we were able to find the vanishing points within milliseconds on a current desktop computer, and within a few seconds on a mobile Android device. The biggest time sink, as expected, is the Hough transform procedure, which bins over all pixels in the edge transformed image. The noisy line methods developed here, including Best-L_2 and K-means, work with the low-resolution Hough images to find a best estimate of the vanishing points. We also make reference to a surprisingly robust family of line-intersection algorithms based on the L_1 norm.

Mobile devices have reached a sufficient level of com-

320×240 image	PC (2.2 GHz)	Motorola DROID
Edge Detect	34ms	0.6s
Hough Transform	100ms	7.2s
Peak Detect	$<$ 20ms	200ms
Least Squares L_2, $K = 1$	$<$ 20ms	$<$ 1ms
Best-L_2, $K = 2$	80ms	$\sim$
Best-L_2, $K = 3$	0.7s	$\sim$

TABLE I
Typical timing for 320×240 down-sampled image

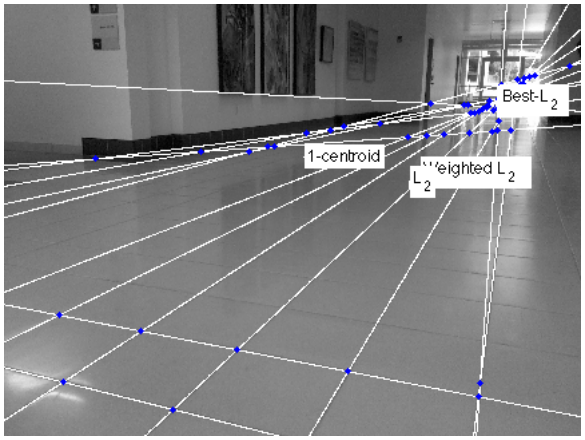

Fig. 4. Original grayscale image with one visible vanishing point, and 15 strongest lines (corresponding to floor tiles) superimposed. Line intersections are marked by blue solid circles (many are off-screen), and estimates of vanishing point location using the methods described in this paper. In order of worst to best: $K = 1$ centroid, L_2, Weighted L_2, L_1, and Best-L_2. The L_1 and Best-L_2 methods estimate the vanishing point almost spot-on, however Best-L_2 also detects the other vanishing point off screen and to the left.

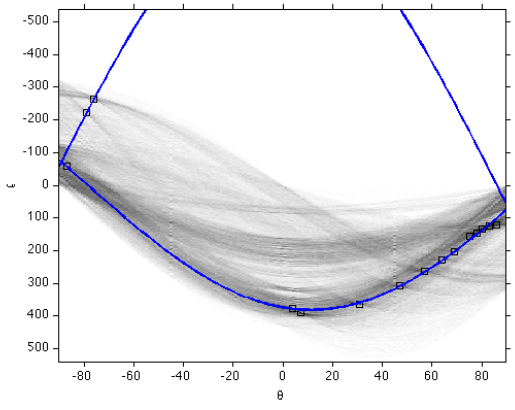

Fig. 5. Hough-domain view of the edge map, with 15 largest peaks denoted by squares, and best-fit $K = 2$ sinusoids that explain the peaks.

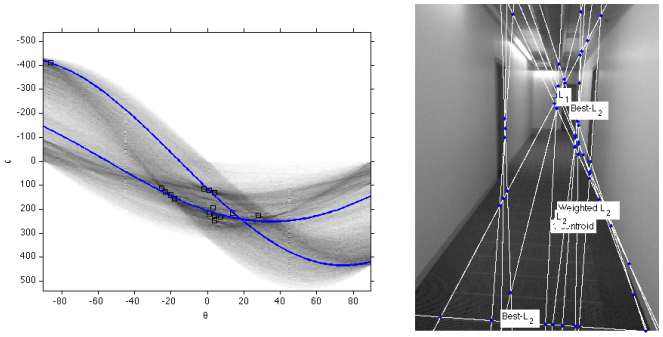

Fig. 6. Hough transform with best-fit two $K = 2$ vanishing point sinusoids (left) and associated hallway image (right) for which Best-L_2 fails. Here the assumption of $K = 2$ vanishing points is a poor one, since there is only one detected line (bottom) which reenforces that assumption.

putational capability, where direct implementation of fairly complex image processing algorithms allows for useful applications. Further work would continue to optimize the Hough vanishing point procedure for robustness and speed. We also direct the reader to higher-order Hough [3], [7], as well as spherical projection [4], and RANSAC-based methods.

ACKNOWLEDGMENT

Ivan Papusha developed and tested the main algorithms in Matlab, while Matthew Ho developed and tested the Android-based algorithms in parallel. Finally, the authors would like to thank Prof. Bernd Girod and the EE368 staff for the excellent course materials and guidance.

REFERENCES

[1] W. Zhang and J. Kosecka, "Experiments in building recognition," George Mason University, Tech. Rep. GMU-CS-TR-2004-3, 2005.

[2] C. Jung and R. Schramm, "Rectangle detection based on a windowed hough transform," in *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on*, 17-20 2004, pp. 113–120.

[3] V. Cantoni, L. Lombardi, M. Porta, and N. Sicard, "Vanishing point detection: Representation analysis and new approaches," in *Proceedings of the 11th International Conference on Image Analysis and Processing*, 2001, pp. 26–28.

[4] M. Bosse, R. Rikoski, J. Leonard, and S. Teller, "Vanishing points and 3d lines from omnidirectional video," in *Proceedings of the International Conference on Image Processing*, vol. 3, September 2002, pp. 513–516.

[5] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Prentice Hall, 2008.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[7] L. A. Fernandes and M. M. Oliveira, "Real-time line detection through an improved hough transform voting scheme," *Pattern Recognition*, vol. 41, no. 1, pp. 299 – 314, 2008.

[8] M. Zuliani, "Ransac for dummies," http://vision.ece.ucsb.edu/ zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf, 2009.

[9] M. Langer, "Finding vanishing points," Lecture notes, http://www.cim.mcgill.ca/ langer/558.html, October 2009.

[10] J. Wang, F. Dong, T. Takegami, E. Go, and K. Hirota, "A 3d pseudo-reconstruction from single image based on vanishing point," in *Journal of Advanced Computational Intelligence and Intelligent Informatics Vol.13*, 2009.