Histogram Matching for Camera Pose Neighbor Selection

Parris K. Egbert
eyberg@cs.byu.edu

Bryan S. Morse
morse@byu.edu

Kevin L. Steele

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

Part of the Computer Sciences Commons

Original Publication Citation

BYU ScholarsArchive Citation
Egbert, Parris K.; Morse, Bryan S.; and Steele, Kevin L., "Histogram Matching for Camera Pose Neighbor Selection" (2006). Faculty Publications. 308. https://scholarsarchive.byu.edu/facpub/308

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 ellen_amatangelo@byu.edu.
Histogram Matching for Camera Pose Neighbor Selection

Kevin L. Steele, Parris K. Egbert, Bryan S. Morse
{steele, egbert, morse}@cs.byu.edu
Department of Computer Science, Brigham Young University
3361 TMCB, Brigham Young University, Provo, Utah 84602

Abstract

A prerequisite to calibrated camera pose estimation is the construction of a camera neighborhood adjacency graph, a connected graph defining the pose neighbors of the camera set. Pose neighbors to a camera C are images containing sufficient overlap in image content with the image from C that they can be used to correctly estimate the pose of C using structure-from-motion techniques. In a video stream, the camera neighborhood adjacency graph is often a simple connected path; frame poses are only estimated relative to their immediate neighbors.

We propose a novel method to build more complex camera adjacency graphs that are suitable for estimating the pose of large numbers of wide- and narrow-baseline images. We employ Content-Based Image Retrieval techniques to identify similar images likely to be graph neighbors. We also develop an optimization to improve graph accuracy that is based on an observation of common camera motions taken when photographing with the intent of structure-from-motion. Our results substantiate the use of our method for determining neighbors for pose estimation.

1. Introduction

Camera pose estimation, the recovery of a camera’s extrinsic parameters, is a long-studied problem in computer vision, and researchers have generated significant results and some robust algorithms [8]. In many pose-estimation algorithms, image features such as points and lines are matched between image pairs, triplets, or sequences and the matches are used to compute the camera pose. A prerequisite to this feature matching is the identification of image pairs or an image neighborhood defining similar images with which to match features. These image neighbors can be expressed as an undirected adjacency graph [18], where nodes of the graph are cameras and their respective images, and edges infer proximity between cameras whose view frusta overlap to include common scene structure. Edges essentially connect cameras that have a high likelihood of successful pose estimation due to the overlapping image content (see Figure 1 for an example).

Many pose-estimation algorithms depend on the transitive correctness of successive camera poses:

\[(A \rel P B) \land (B \rel P C) \Rightarrow (A \rel P C),\]

(1)

where \(A \rel P B\) is a binary relation between camera pair \(\{A, B\} \in S_{cams}\) (the set of all input cameras) indicating that the pose of camera B is correctly estimated relative to that of camera A. Since there is often an explicit ordinaity...
to the pose estimation of a camera sequence, a poor estimate early in the chain can propagate large pose errors. It is therefore important that the camera neighborhood adjacency graph be as connected as possible (Figure 1). Our objective in this paper is to propose a novel process of camera neighbor selection for construction of the adjacency graph.

The emergence of inexpensive and high-quality digital cameras and camcorders, together with the improved ability to quickly move image and video content into computer memory has enabled many applications of 3D reconstruction and visualization. It is increasingly desirable to construct dense camera networks of hundreds of cameras for visualization and reconstruction purposes. However, given the limitations of current pose estimation algorithms, especially with wide-baseline cameras, creating the prerequisite adjacency graph is difficult. Current algorithmic deficiencies include the inability to involve all close camera neighbors in an initial pose estimate while excluding images that do not overlap at all. Another deficiency is the lack of a coherent method to include all types of footage, e.g., video streams and still images, simultaneously in the pose estimates.

In this paper we propose a novel and efficient way to create the camera neighborhood adjacency graph in the presence of hundreds of input images by using methods from content-based image retrieval systems. We use color histograms to identify similar images as candidate camera neighbors, and partial histogram functions (defined in Section 4) to more accurately determine neighbors given constraints unique to the purpose of pose estimation. Our method has the ability to find accurate camera neighbors for large numbers of input images without imposing a specific pose order (at the expense of an $O(n^2)$ algorithm), and makes no distinction between image input formats (still images or video streams).

2. Background

Often the adjacency graph creation for a pose solution is done by hand in order to exploit known matching characteristics and optimize the quality of the pose estimation. When few images are to be matched, neighbor selection is trivial. In this section we describe past methods in determining pose-neighbor selection. Often the selection process is implicit to a pose-estimation algorithm, and an adjacency graph is not explicitly constructed.

Teller et al. [18] create omni-directional images of outdoor urban environments for pose estimation. They determine camera neighbors by taking the $k$-nearest neighbors of an initial adjacency graph constructed from GPS sensor data acquired at the physical camera location. Most other methods (including ours) attempt to build the adjacency graph exclusively from image content rather than utilizing external sensors.

Much recent work has focused on using video streams in the matching process [5, 9, 10, 11, 12, 16], in which features (points, lines, etc.) are identified in an initial frame and then tracked through subsequent frames until the features are no longer identifiable. New features are typically added throughout the tracking process so that at any given frame of the stream many features exist between the current frame and its immediate predecessor and successor. Thus an image’s match (and pose) neighbors are the frames immediately preceding or following it in the video stream, and the corresponding adjacency graph degenerates to a path graph, i.e. a path containing all the nodes of the graph (see (a) in Figure 2). All algorithms that operate on sequences

![Figure 2. Adjacency graphs created from an implicit ordering of the input cameras. The pyramids in (a) and (b) show the camera locations of two hypothetical input video streams pointing toward a central point (X). The input for (a) is a one-dimensional tracked sequence about X. Most current algorithms construct the (degenerate) adjacency graph using immediate frame neighbors, as shown with the arrows. The input for (b) is a viewing sphere [9] from a zigzag about X. Looking beyond the immediate frame sequence [9, 10], one can create better graph configurations with more pose neighbors, and consequently more accurate pose estimates.](image)
of images produce similar degeneracies, regardless of input format. For instance, Lhuillier and Quan [11] use still images rather than a video stream, but they nonetheless enforce an ordering on the input images to define match partners. Sainz et al. [16] designed their calibration solution to include video sequences, manually tracked sequences, and still images. However, they still require an imposed ordering on all input images. In contrast, our method makes no assumption on the input order and produces adjacency graphs in which most or all correct world-space neighbors are detected and included, not simply those nearby in a linear sequence.

Algorithmic variants include matching to images several frames separated (2, 3, or more frames ahead or behind the current frame) to improve matching characteristics such as sharpness or depth variance [13]. The first attempts to depart from the conventional sequential ordering requirement are those in [9, 10]. Koch et al. [9] proposed the method of sweeping a camcorder over a viewpoint surface in a zigzag pattern to construct a viewpoint mesh, a 3D polygonal mesh whose vertices are the viewpoints of the reconstructed cameras. Rather than restricting their pose neighbors to frames before and after a current frame, they exploit the zigzag nature of the sweeping pattern to find additional neighbors. Their method backtracks at each frame to examine the 3D pose locations of the previous cameras—any prior cameras within a distance threshold are included as neighbors to the current frame. Figure 2 (b) shows an example of their method. In [10] the authors predict a very coarse estimate of an unknown camera pose by first computing the fundamental matrix $F$ of an image pair from feature matches. They use the epipole extracted from $F$ to predict the new pose direction, and the residual correspondence error of the rectified image pair$^1$ to predict the distance from the pose-partner. Given this coarse pose estimate, they can use a world-space proximity measure to determine camera neighbors from previously estimated cameras. Thus the authors are able to implicitly build a more complete, non-degenerate camera neighborhood adjacency graph. This method is similar to ours in that it attempts to build a non-degenerate adjacency graph. However, their video stream must be centered on one central object of the scene, since the coarse pose estimate cannot account for camera rotation. Our method is able to deal with arbitrary camera motion around any part of the scene, provided there is sufficient overlap of scene content in the input set.

Several recent contributions associate overlapping images using local image features. Brown and Lowe [1, 2] extract SIFT features from each image and build a k-d tree containing all features from all the images of the input set. Images of the same objects or scenes are identified by searching the k-d tree for the nearest neighbors of a given query feature found on the object of interest. This allows the implicit creation of an adjacency graph, which the authors use to determine the metric pose of each image through bundle adjustment.

Schaffalitzky and Zisserman [17] find geometrically and photometrically invariant feature descriptors in all images and store them in a BSP tree. Feature vectors within a threshold distance are considered matches, and an adjacency graph in the form of an explicit spanning tree is constructed. In both [1, 2] and [17] local image features are used to identify adjacent images. Their algorithmic complexity is linear in the number of images, plus a non–linear (though small) tree searching component.

In this paper we propose the use of content-based image retrieval (CBIR) techniques for the task of image neighbor determination for camera pose estimation. By using CBIR to build the camera neighborhood adjacency graph, rather than relying on implicit input order, we remove the requirement of a sequentially-ordered input set. The input set can thus be seen as an image database from which to draw pose neighbors for a given image. Queries made on the database have the constraint that all returned images are neighbors in the adjacency graph.

### 3. Content-Based Image Retrieval

Content-based image retrieval is a set of techniques for retrieving images from a database using features automatically derived from the images themselves. The features used to query CBIR databases often include color histogram content, texture, color location, shape and image composition. CBIR has received widespread research attention, and a number of general-purpose CBIR search engines exist—IBM QBIC [6], MIT Photobook [14], and Berkeley Blobworld [3] to name a few.

Image retrieval in the larger context is concerned with finding the images in a database that are semantically relevant or similar to a query image. Often the relevant images are of the same class or category, such as “all brown dogs” or “all persons on a beach.” In the context of pose estimation, however, we are concerned with finding locationally relevant images, or images of the same part of a scene as the query image. This puts a large constraint on typical CBIR usage, and restricts the useful feature types.

We use color histograms as the feature type to match images. The color histogram is a widely-used feature representation for image retrieval [15]. Its advantages include invariance to rotation and translation of the image content. One major drawback of using color histograms for image retrieval is that they discard any spatial information in the image content. For example, a histogram of an outdoor scene loses all information that the blue sky is at the “top” of
Figure 3. Pyramids representing camera view frusta. Each of the darker blue cameras in the left column has been rotated to the right about the origin in 12 increments of 3°; each row of the pictured matrix of viewpoints comprises a rotational set \( S_{rc} \). Each set has also been rotated 5° up from the previous set to represent varying grazing angles of the geometry with the cameras. The view frusta for all cameras intersect the ground plane \( y = 0 \) forming individual quadrilaterals. The quads for two such cameras (circled) are outlined in bold red and black lines. Note that the overlap between the two quads is exactly the geometric scene content shared between the two images of the circled cameras. The image of the overlap in this figure is shown as the shaded area of Figure 4.

Figure 4. Grid of quadrilateral re-projections. Each square of the grid shows the re-projection of an initial camera’s quad to a rotated camera of the same set. The white area in each square is the overlap of the re-projected quad with the quad of the corresponding rotated camera. It is easily seen that the coverage pattern shifts to the right (the square labeled B) as the cameras increase in their \( y \)-axis rotation, and the pattern rotates clockwise as the camera sets decrease in their \( x \)-axis rotation (the square labeled C). This suggests the search patterns illustrated in Figure 5. The green shaded area in the figure corresponds to the quadrilateral overlap from Figure 3.

We transform RGB image color to the HSV color space and make comparisons using the hue component exclusively. A simple \( L_1 \)-distance comparison function between two hue histograms \( H \) and \( I \) works well in practice:

\[
d(H, I) = \sum_{k=1}^{n} |H_k - I_k| \tag{2}
\]

where \( n \) is the number of color bins. Given a query image histogram \( H \), and a database image histogram \( I \), the histogram distance \( d(H, I) \) represents the dissimilarity between the two images. For the \( L_1 \)-distance metric, \( d(H, I) \) is precisely the number of pixels that differ in hue in both images. For each query image, we can sort the remaining images in our input set (the database) on increasing values of \( d \), then threshold either the number of neighbors or the value of \( d \) to determine the actual neighbors of the query image.

By performing histogram matching using all images in the input set as independent query images, we can build a directed adjacency graph. If the degree of each node is constant, as would be the case in a typical implementation, then the graph is directed due to the lack of a symmetric binary relation in the set of all neighbors, i.e. \( I_1 \) having neighbor \( I_2 \) does not imply \( I_2 \) has neighbor \( I_1 \). For many pose estimation algorithms, a directed adjacency graph is sufficient. If an undirected graph is necessary, it can be assumed from the directed graph with the allowance of a non-uniform threshold (for example, each node may have differing numbers of neighbors).

4. Optimization

We now propose a novel optimization that improves the accuracy of the adjacency graph as constructed in Section 3.
The optimization is based on an observation of typical camera motions made when photographing scenes for eventual structure-from-motion applications:

**Observation.** When photographing for the purpose of later 3D reconstruction or visualization, we have a tendency to follow specific camera-motion patterns—we tend either to rotate about a subject’s up-axis, or to track (translate) horizontally or vertically past a subject.

We also tend to avoid extreme or arbitrary camera motions such as cyclo-rotation, panning (except in the case of creating panoramas), diagonal tracks, large rotations and large tracks. While we haven’t performed user studies on the validity of this observation, we believe it to be a fair characterization based on our own camera motion patterns and those observed in the computational stereo literature.

This observation will guide the development of our algorithm. Let the ground plane \( y = 0 \) be a coarse approximation of the geometry of interest in a structure-from-motion application. If we map the typical camera motions taken from the observation, we get a set of translated and rotated camera positions centered on a point, e.g., the Euclidean origin. Figure 3 illustrates candidate sets of rotated cameras. The set of rotations can be expressed by

\[
S_{rc} = \bigcup_{\theta=\alpha}^\beta M(\theta, t)P
\]

where \( S_{rc} \) is the set of homogeneous camera projection matrices representing the rotated cameras, \( P \) is the homogeneous camera matrix of an initial camera, \( M(\theta, t) \) is the parameterized matrix that rotates the initial camera by \( \theta \) about a point on the principal axis \( t \) units from the camera center, and \( \alpha \) and \( \beta \) are the limits of rotation. \( M(\theta, t) \) is expressed in Equation (4) as the composition of the individual affine transformations in (5). These matrices translate the camera’s rotational center to the origin, rotate the camera about the positive \( y \)-axis, and then translate it back again.

\[
M(\theta, t) = \begin{bmatrix}
cos \theta & 0 & -t \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[(4)\]

\[
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -t \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[(5)\]

The intersections of the view frustas with the ground plane form quadrilaterals, as shown in Figure 3. Let \( q_i \) be the image of the quadrilateral formed by an initial camera \( C_i \), and \( q_r \) be the image of the quadrilateral formed by a rotated camera \( C_r \), where \( C_i \) and \( C_r \) are cameras whose defining matrices \( M_i, M_r \in S_{rc} \). The image \( q_i \) can be re-projected to \( C_r \) by a homography \( H \). This re-projection \( \hat{q}_r \) is the original quadrilateral as seen from the vantage point of \( C_r \). The overlap of the re-projected quad \( \hat{q}_r \) with \( q_r \) is precisely that geometry that is seen from both cameras. Figure 4 illustrates the overlap of \( \hat{q}_r \) with \( q_r \) for the array of cameras in Figure 3.

In the context of pose estimation, the overlap represents the image content from which to derive useful features for feature matching. The quality of the matches directly determines the success of the pose estimation. Since larger image overlaps imply larger quantities of matches, we would like to quantify the amount of overlap between a query image and all other images, and assign neighbors to the query image from among those with the most overlap. Herein lies the optimization: we can improve the accuracy of the adjacency graph by detecting this overlap, then computing color histogram comparisons only on the overlapping portions of images rather than on entire images.

Examining the family of overlap patterns created from typical camera motions suggests an efficient method to determine the correct overlap. Consider the example of the two cameras labeled A and B in the top corners of Figure 4. Camera B is horizontally rotated 36° about a point visible to both cameras. The overlap pattern seen at label B in Figure 4 is approximately a right-shift of the image contents. If the scene contains any depth variation, there will be parallax in the shift, but for moderate depth variation the shift still maintains most of the color content. If we compute a color histogram on only the shifted portion of image B, and its equal but opposite shift (we denote as the conjugate shift) in image A, we exclude from the histograms those pixels that are not likely to be shared between the images.

Since we do not know in advance the amount of rotation (if any) between two cameras, we can parameterize the shift and perform an iterated search for the smallest histogram distance. The histogram distance function then becomes

\[
d(H_\phi, I_{\phi*}) = \frac{1}{|\phi|} \sum_{k=1}^n |H_{\phi,k} - I_{\phi*,k}|
\]

where \( H_\phi \) is the partial histogram of the query image given a shift pattern, \( I_{\phi*} \) is the partial histogram of the test image given the conjugate shift pattern, and \( |\phi| \) is the size in number of pixels of the partial histogram. The function \( \phi \) returns the set of pixels of a specific shift parameter, and \( \phi * \) the set of pixels of the conjugate shift. The distance is weighted by the size of the partial histogram to give distance per unit pixel. The histogram distance of the best overlap is given by

\[
d_{\text{min}} = \min_l d(H_{\phi(l)}, I_{\phi*(l)})
\]

where \( l \) is the shift parameter. In the above example, \( l \) iter-
Figure 5. Histogram search coverage patterns—each pair consists of a search pattern and its conjugate. We propose six pairs of search patterns (a-f) that follow the coverage patterns in Figure 4. Pattern (a) proceeds left-to-right in one image and right-to-left in the other image. Patterns (b-d) proceed similarly according to the arrows. Patterns (e) and (f) proceed from the corners as illustrated to detect low-grazing angle rotation, for example the pattern labeled C from Figure 4. Note in this case that we do not need to proceed from the bottom corners for two reasons: the effect of perspective decreases the overlap much more at the top corners, and camera angles from the underside of surfaces are not included in the list of common camera motions.

\[ H(\phi) \text{ computes the histogram of the pixels to the right of column } l \text{ in image B, and } I_{\phi}(l) \text{ computes the histogram of the pixels to the left of column } (\text{ImageWidth} - l) \text{ in image A.} \]

This process can be repeated using other shift patterns that represent the common camera motions. We use six different shift patterns, given in Figure 5, and take the minimum \( d_{\text{min}} \) of the six patterns to be the final partial histogram distance between two images. Figure 6 shows the outlines of an optimal partial histogram region from an image pair.

The algorithmic complexity of partial histogram comparison is similar to that of using global histograms. Computing the global histogram is \( O(n) \) in the number of pixels, and computing partial histograms is also \( O(n) \) for each shift pattern since, for each value of the shift parameter \( l \), we simply add a line of pixel values to the histogram data computed for the previous parameter value. However, while comparing histograms is \( O(n) \) in the number of hues chosen for both global and partial histograms, we make \( m \) more comparisons for partial histograms, \( m = \text{ImageWidth} \), \( m = \text{ImageHeight} \), or \( m = \text{ImageWidth} + \text{ImageHeight} \) depending on the

Figure 6. Image pair showing optimal partial histogram regions. The partial histogram distance function found the minimum at column 219 (640x480) in the left image using coverage pattern (b) from Figure 5.

shift pattern. We have observed this commensurately increased running time in practice.

5. Results

We tested our optimized and non-optimized adjacency graph building algorithms using four sets of wide-baseline images. Sample images from each of the four sets are shown in Figure 7. We use precision vs. recall comparisons to measure the accuracy of the set \( S_i \) of computed neighbors for each image. To use this measure, we also constructed the set \( T_i \) of “true” neighbors for each image. Precision measures the percentage of \( S_i \) in \( T_i \), or \( |S_i \cap T_i|/|S_i| \). Recall measures the percentage of \( T_i \) in \( S_i \), or \( |S_i \cap T_i|/|T_i| \).

To automatically construct the set \( T_i \) we utilized feature correspondences that are often-times used to estimate internal and external camera parameters. We computed an accurate set of point matches and their connecting vectors between all pairs of input images. The proportion of image overlap for each pair was determined from its average matching vector. Since low variances of point match vectors correlate well with close image neighbors, we rejected outlying images with no overlap by thresholding on \( \sigma^2 \). Finally, we constructed \( T_i \) for each image from the images exhibiting the most overlap.

We use Equation (7) to build the candidate neighbors for each image in the test sets. For a given query image, the minimum histogram distances are computed to all other images in the set. The distances are sorted, and the test images corresponding to the smallest \( n \) distances are kept as neighbors to the query image. We measure the precision/recall accuracy for values of \( n = \{1..k - 1\} \), \( k \) being the size of the test set.
Figure 7. Representative images from four test sets. The images from both the Museum and Office sets have camera motions containing significant horizontal and vertical rotation, while Outdoor1 has horizontal and some vertical rotation, and Outdoor2 has only horizontal rotation.

Results of the four test cases are given in Figure 8. Each data point on the four graphs represents the average precision and recall of the computed neighbor sets for a specific \( n \) number of neighbors. The graphs show a typical inverse relationship between precision and recall \([4]\), and in all four cases using partial histogram distances improves the precision accuracy from simply using the global histogram function in Equation (2). Partial histogram distances improve precision on average by 26.9% for 5 neighbors, and by 24.3% for ten neighbors. Table 1 lists percentage improvements for each of the four test cases.

<table>
<thead>
<tr>
<th></th>
<th>5 Neighbors</th>
<th>10 Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Museum</td>
<td>35.8%</td>
<td>36.2%</td>
</tr>
<tr>
<td>office</td>
<td>24.0%</td>
<td>24.1%</td>
</tr>
<tr>
<td>outdoor1</td>
<td>24.7%</td>
<td>23.7%</td>
</tr>
<tr>
<td>outdoor2</td>
<td>23.0%</td>
<td>13.3%</td>
</tr>
</tbody>
</table>

Table 1. Improvement of precision accuracy for using partial histogram distances rather than global histogram distances.

6. Summary and Conclusions

We have shown a novel and efficient method to create the camera neighborhood adjacency graph for use in pose
estimation without having to first find feature matches between two images—a potentially time-consuming process for wide-baseline images [7]. In addition, our method can process sequential and non-sequential image collections simultaneously, such as images from still cameras and video streams from camcorders. An ordering such as a least-cost Hamiltonian path could later be imposed on the adjacency graph for use in existing sequential pose-estimation algorithms, although we foresee hierarchical or other non-sequential posing schemes to ultimately be more efficient.

We have also proposed an optimization for improving the accuracy of the adjacency graph by computing partial histograms, i.e., histograms on just the overlapping portions of image pairs. The optimal overlapping portion is found as the minimum of a distance function that computes partial histograms on a pre-determined pattern of image overlaps. The pre-determined pattern comes from an observation of common camera motions used when photographing content for visualization or 3D reconstruction.

Our method works well for the image sets we have shown in this paper and for many other image sets we have tested. Since our algorithm requires each image in a set to act as the query image, and each query image is compared to all other images in the set, it is an \( O(n^2) \) algorithm. In practice one can perform the algorithm on sets containing hundreds of images without undue computation time on a modern processor. However, for sets of extremely large size (thousands of images) it may be beneficial to hierarchically cluster the input images as they are matched. Traditional clustering methods could be used for this using coarse histogram content as a cluster feature.

References