Detecting doors for 3D stereomodeling using superpixels and raw image data.

Chris Cornelius
St. Olaf College Computer Science
Senior Capstone 2012

0. Abstract

A door-detection algorithm using superpixels – polygonal image segments determined by similar color – is proposed in the context of St. Olaf College’s Palantir project, which uses stereoimaging to create 3D models of buildings. Images are processed with Canny edge detection, and superpixel edges are filtered by their intersection with edge pixels on the binary edge map. The resulting “strong edge segments” are filtered and grouped by their colinearity to make global lines which cross through the entire image, and quadrilateral door candidates are constructed from all possible groups of four of these lines. Door candidates are filtered by geometric criteria to select the final output data. However, experience during development and comparison with image-based strategies indicates that using superpixels to compute door locations may perform less effectively than methods that do not use superpixels, in both computation time and accuracy of detection.

1 Introduction

For the creation of an accurate 3D computer model of a building, the locations of doors must be determined so that the model can contain information about the interconnectedness of rooms and hallways in the building it reproduces. In the context of a stereoimaging approach to 3D capture and modeling, we aim to use data derived from camera images to identify the locations of doors.

This paper describes a door-detection method designed for use in the Palantir project at St. Olaf College, which aims to create a three-dimensional computer model of a large building using image pairs taken with a stereo-vision camera rig. Over three thousand image pairs provide coverage of every location in the building from multiple angles. These images are then used to feed a pipeline which rectifies the images for focal length and distortion, computes superpixels.

1 The creation of “superpixel” is a method of image segmentation which associates image pixels into...
from the images, determines the distance of each superpixel from the cameras, and associates superpixels with 3D surfaces which are then incorporated into the final model. Door information could be incorporated into the 3D model through tagging certain regions as door regions, which would allow those navigating the 3D model to pass. To facilitate this process, our method aims to indicate the probable corner points for all doors in any given image, as well as which superpixels from the image make up those doors.

A review of three representative published papers (see [1], [2], and [3]) reveals insight into standard methods used for the extraction of basic feature data from an image, an identification of doors based on that feature data, and methods of evaluation of the effectiveness of those techniques. Door detection procedures generally begin by performing some form of edge and corner detection on the raw image data, which yields quadrilateral regions which are potential doors. Some algorithms perform additional filtering to focus the detection on certain “regions of interest”, which are more likely to contain doors. At this stage, all three methods surveyed detect the corners of doors and use those four points as references for the door candidates. Once this basic information has been decoded from the pixel data of the image, various algorithms evaluate the candidate doors and determine which are doors and which are not. Many make use of “scoring” algorithms, which give confidence for each object’s nature of being a door based on one or more metrics of evaluation. Our method draws on aspects of these approaches, while adding and adapting general strategies to fit best into an environment that uses superpixels as its primary form of data.

2 Method

1. Overview

Door detection in the Palantir project begins with an image and its corresponding set of superpixels, searching first for edges in the image which might be the edges of doors. The raw image, supplied in an RGB format, is processed to create a binary edge map, tagging all pixels in the image as edge pixels or not-edge-pixels. Then the set of all superpixels is filtered to determine which of the polygons have edges that correspond to edge pixels. This results in a group of line segments derived from superpixels which describes all edges in the image. Resultant
line segments are merged together if their slopes and positions closely match, finding lines through the image which are then assembled into quadrilaterals. All quadrilaterals are filtered based on their shape and size, and the top candidates are returned as candidate door objects.

2. Discussion of inputs

The standard file formats of the Palantir project are .ppm and .off files. A standard image format, Portable Pixel Map files represent uncompressed RGB data of image files and are read into memory via specially designed Image objects which hold the RGB data and allow easy access and modification. In the superpixel detection process, polygons are created associating groups of similarly colored pixels in a given image. These are stored using the Object File Format, easily loadable into arrays of Polygon objects whose vertices correspond to points in the image’s coordinate space. The door detection process takes as input an Image object and a vector-type array of Polygon objects, loaded from disk.

3. Calculation of gradient map and binary edge map

A Canny edge detection process is used to quickly determine which pixels in the source image are involved in visually strong edges (FOOT: wikipedia provides a thorough and useful explanation of this process, which can be used as additional reference to the information presented here). The Canny method, a well-known algorithm for edge detection, begins by computing the gradient at every pixel in the grayscale-converted image. Delta values between adjacent pixels in both X and Y directions are computed using a Sobel mask, which gives varying weight to pixels in the neighborhood of the pixel under evaluation (see fig. 2.3.1). Post-weighting, the grayscale values of all pixels in the neighborhood are summed into a total gradient in the X direction and total gradient in the Y direction, which can be converted into a direction and magnitude of the greatest instantaneous pixel intensity change using a simple arctangent computation. The gradient map holds important information about the edges in the image, because areas of high gradient magnitude indicate regions where the value of pixels is changing rapidly, which may indicate an edge.

To simplify the relatively noisy gradient map, the Canny procedure next creates a binary edge map, flagging pixels using boolean values as either edge pixels or not-edge-pixels. A naive approach might
simply apply a threshold to gradient magnitude, accepting all pixels above a certain value as edge pixels. But this approach results in indistinct regions, instead of crisp lines of pixels, being marked as edges. Following application of a very low threshold, non-maximum suppression compares each pixel to the pixels in its immediate neighborhood, and only the pixel with a locally maximum value is flagged as an edge pixel. Important in this process is the comparison with only those pixels in the direction of the gradient, preventing the suppression of pixels in the “core” of the edge region and quickly eliminating those of the periphery.

At the end of Canny edge detection, an Image object is stored in memory with each pixel flagged as either an edge pixel or not. This process produces clear information about pixels which correspond to lines in the source image.

4. Extraction of strong edges

Our algorithm next associates superpixel edges (line segments between discrete points) with the binary edge map by checking which superpixel edges overlay pixels that are marked as edge pixels. This algorithm is inspired by the work of Tian, Yang, and Arditi, in which the path between two possible corners is tested against a binary edge map by checking the overlap between the line and the pixels representing the real edge [1]. Our method samples the pixels overlaid by a line segment and flags as strong segments those which overlay a significant number of edge pixels. The resultant collection of line segments is the source for the lines that will make up the edges of door candidates later in the process.

Each superpixel is decomposed into a set of LineSegment objects which correspond to the edges of the polygon. Each segments is decomposed into a set of pixel locations that lie along the line through a naive method that steps along the line via deltaX and deltaY equal to one-half of a pixel width. The pixels along the line are counted and the number of edge pixels in the line is divided by the line length to produce a “fill ratio” which indicates the strength of that particular line segment. Simple thresholding for fill ratios greater than 65% satisfactorily extracts line segments from the set of all superpixels and produces a set of segments which overlap the strong edges in the binary edge image.

5. Merge strong edges into global lines

Though it is useful to know which line segments overlap strong edges, this
information is not enough to create candidate doors because the edges of superpixels are often very short and only span a small section of a strong edge in the binary image. To circumvent this problem, segments which are approximately collinear are merged into global lines which pass through the entire image.

To merge approximately colinear segments, a vector-type data structure containing all the strong-edge LineSegment objects is ordered by their slope using a standard merge-sort algorithm. This sorted list is then subdivided into “bins” which will associate segments of similar slope. Division into bins is accomplished by iterating through the list of LineSegments, keeping track of the slope of the first segment in the current bin. When the iterator indicates a segment whose slope differs from that of the first segment in its bin by a given threshold value (chosen to be two degrees), the iterator’s segment becomes the first segment of a new bin. The result of this first step is bins of LineSegment objects grouped according to their slopes.

Within each bin, an identical process is applied to the Y-intercepts of all the LineSegment objects, assuming that the segment was extrapolated to meet the Y-axis. This strategy further refines the grouping of segments, linking together segments that are approximately colinear. All the segments in each sub-bin are bounded and merged, creating global lines which cross through the entire image. Parallel lines, which have the same slope but radically different Y-intercepts, will not be merged together in this step, resulting in a set of lines, each of which overlaps a strong edge in the image. Some information is lost because line segments have been converted into lines, which lack information about endpoints, but the advantages of using global lines will be discussed in the next section.

6. Convert line segments into quadrilaterals

Door candidates are generated by computing all combinations of four global lines via a quadruple-nested-loop. Lines are used instead of segments with the goal of tolerance for occlusion of door corners – intersections points that lie outside the image or behind a closer object can still be found due to the extrapolation inherent in this method (see fig. 2.6.1). During generation, a simple filter is applied excluding door candidates which have one or more corner points a significant distance outside the image plane (outside of a 20%
bounding box). All generated door candidates are stored using DoorObject objects, a data structure which not only stores the lines which form the door but also allows easy calculation and caching of the door’s corner points, area, and any other quantities desired.

A word must be said about the detection of the corners of a quadrilateral when only given four lines. A set of four lines can in fact intersect in up to six locations, so determining which intersection points represent corners of a quadrilateral shape requires some computation. In our algorithm, all intersections are calculated and placed as nodes in a nondirected graph according to their adjacency along the source lines. Only nodes which have a cycle to themselves of length four are corners of a quadrilateral (see fig 2.6.2). Once computed, these corner points can be cached and used again and again without needing to construct the graph again.

7. Geometric filtering of door candidates

The large set of DoorObjects representing all possible doors must be filtered to return only reasonable candidates. Our strategies assign a floating-point “weight” to each DoorObject based on various geometric evaluations, with a weight of 0.0 indicating complete failure of the evaluation and a weight of 1.0 denoting complete certainty that this object is a door according to the evaluation. For the filtering process, an average “weight” from all tests is limited by a thresholding process, and all remaining door candidates can be sorted by their “weight” and only top-qualifying candidates selected.

Our main strategy of geometric evaluation checks the width-to-height ratio of doors. Standards in the United States place the ratio of door height to door width around 2.2:1, so the average length of the long and short edges of DoorObjects is divided against this ideal ratio to return the “weight” of the door. If the computed ratio is instead greater than the ideal ratio, it is divided into the ideal ratio to again produce a number between 0.0 and 1.0. Similarly, preference is given to DoorObjects whose interior corners exhibit symmetry, with opposite corners in the quadrilateral being close to equal.
- 3 -

Results

Though the methodology described above shows promise, this method cannot be completely because the implementation does not work correctly. The method successfully highlights door regions in the dataset by producing a concentration of door objects which bound doors, but none of the DoorObjects returned bounds the door region completely. In addition, many quadrilaterals are detected in each given region, with only slight differences between the sometimes hundreds of DoorObjects overlaying a given door. Without properly working code, it is difficult to make meaningful comparisons between this algorithm and others.

Other door detection algorithms evaluate their success by processing a test dataset selected by the developers. A general survey of [1], [2], and [3] indicates that a suitable test dataset would contain one hundred or more images around 1000 pixels wide, each image containing one or more doors. In addition, some approaches rank the complexity of images, scoring correct identifications in complicated images as more valuable than identification in less cluttered or object-filled images. Benchmarking in these three papers considers one or more of the following: the percentage of total doors detected correctly in the dataset by the algorithm, the number of false positive identifications (with luck, a very low number), and/or the time of execution per image.

A test data set of eleven door-containing images has been assembled, but no tests have been made because the code is not ready for a benchmarking run. Applying the algorithm as it stands would yield a high percentage of false positives, and may fail to identify any doors correctly at all. One clear and easy comparison which can be drawn, however, involves running time. On a 1188x792 image, our algorithm executes in more than ten seconds, not counting the sixty or so seconds required to run the firstOFF program to generate superpixel information. This indicates relatively poor time efficiency when compared to the sub-seven-second times of [2] or the almost-real-time performance of the algorithm described in [1]. It appears that without significant optimization, this algorithm will not be able to outperform other approaches with respect to execution time.
Conclusions

This paper presents a strategy which uses superpixel information in the detection of doors. Although the use of superpixels makes this method unlike others in the field, it is reasonable to make comparisons with other methods - and in general, methods based not on superpixel edges but on image-based calculations seem to perform with equal or better accuracy in certainly less time. It seems that relying heavily superpixels, while useful in the context of the St. Olaf Palantir project, may not be a sound strategy when searching for acceptable performance, both for time and for accuracy. However, the largest obstacle to evaluating the performance of this method is problems with its implementation. If the codebase were perfectly functional, a superpixel-based method might indeed prove useful for the detection of doors in the Palantir dataset.
Figures

fig. 2.3.1 - Sobel masks used for gradient detection

\[
\begin{align*}
G_x &= \sum \text{neighborhood} \times \text{mask}(X) \\
G_y &= \sum \text{neighborhood} \times \text{mask}(Y)
\end{align*}
\]

fig. 2.4.1 - interpolating the endpoints of a line segment and checking against pixels
in the binary edge map.

fig. 2.6.1 - occluded or missing door corners are still detected by the door candidate generation process due to line extrapolation.

fig. 2.6.2 - a nondirected graph is used to determine the corner points of the quadrilateral formed by the intersection of four global lines.
References


All papers can be found linked on the St. Olaf CS Wiki