Detecting Junctions in Photographs of Objects

A thesis presented
by

Hyunho Richard Lee

to
Computer Science
in partial fulfillment of the requirements
for the degree of
Bachelor of Arts

Harvard College
Cambridge, Massachusetts
April 2011

Thesis advisor

Todd Zickler

Author

Hyunho Richard Lee

Detecting Junctions in Photographs of Objects

Abstract

Line drawings are inherently simple, yet retain most of the information of a full image. This suggest that line drawings, also called contour images, could be useful as an intermediate representation for computer understanding of images. However, extracting physically meaningful contours from photographs has proved to be difficult. Recently, there has been significant progress in detecting contours. As one example, the *gPb* algorithm (Arbeláez et al., 2010) uses local cues such as brightness, color, and texture, along with global cues that interpret the image as a whole in order to achieve high-performance contour detection.

However, contour detectors perform poorly near junctions. Junctions are points where two or more contours meet, creating two or more regions near a single point. Contour detectors, including *gPb*, assume that contours occur at the meeting of only two regions. We find that this assumption breaks down near junctions, resulting in poor performance. This is unfortunate because the location of junctions and the orientation of the incident edges at a junction are useful in reasoning about the geometric relationships between contours, and thus reasoning about the shape of the object.

These two observations, that junctions carry significant information and that contour detectors perform poorly near junctions, suggest that finding a complete and useful contour image requires complementing a contour detector with a junction detector. However, while there has been substantial recent progress in contour detection, there has not been comparable work and progress on the problem of junction detection.

We thus build and implement a junction detector. We are informed by the insight that junction points are a generalization of contour points, and we adapt ideas from the gPb contour detector in developing our algorithm. We also capture images of objects, and run our implementation on our dataset. Although our results are qualitative, they suggest that our junction detector could be useful as a complement to existing contour detection schemes.

Contents

Chapter 1

Introduction

The simple but powerful summary of images that line drawings offer makes them seductive—to the human eye in the form of cartoons and graphic art, and to computer vision as an intermediate visual representation for scene analysis. In this work, we suggest a technique for improving the extraction of line drawings from photographs. As is evident in Figure 1, line drawings, also called contour images, carry almost as much visual information as the original images but are less complex. This apparent distillation of scene information makes contour images potentially useful in computer interpretation of images. Although cartoonists and other visual artists can create line drawings with ease, computer extraction of contours from photographs has proved to be difficult.

We discover in our work that contour detectors are especially prone to error near junctions, which are points where two or more contours meet. Every contour detector assumes that contours are points where two regions meet, but the third region that occurs near junctions of degree greater than two violates this assumption. Thus,

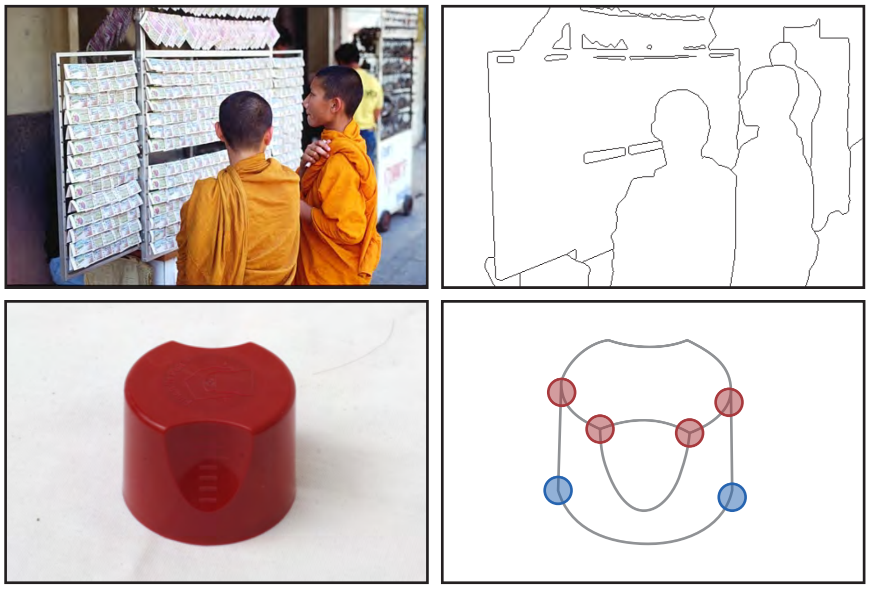

Figure 1: Contour and boundary images carry significant visual information. *Top row:* An image from the Berkeley Segmentation Dataset [24] and its object boundaries. *Bottom row:* The bottom row is an image from our dataset and its contour image. Junctions of degree three have been highlighted in red, and junctions of degree two have been highlighted in blue.

junction points and contour points near junctions will theoretically not be detected well for any contour detector. One state-of-the-art contour detector is Arbeláez et al.'s *gPb* algorithm [1]. *gPb* uses local cues such as brightness, color, and texture, along with global cues that interpret the image as a whole. The *gPb* algorithm is successful in robustly detecting contours from photographic images, but we show experimentally that this contour detector does not perform well near junctions.

However, junction points are critical parts of the contour image. They have proved useful on their own in object recognition [33], and previous work [2] suggests that corners and junctions are the most significant parts of the contour image. Junctions are also critical to reasoning about the three-dimensional interpretation of contours [21]. For example, an object whose contour image has no junctions cannot have more than one visible smooth surface in three dimensions. This is a simple example, but a great deal of information about shape can be determined by reasoning about contours and junctions.

Thus, in order to obtain a comprehensive contour image, the contour image generated by contour detectors must be completed by the output of a junction detector, which is our motivation for this work. Our junction detection algorithm is informed by the insight that junction points are a generalization of contour points, as contour points are points where two regions meet, while junctions are points where two or more regions meet. In addition, the problem of junction detection has not been studied as thoroughly as contour detection. Thus, in developing our algorithm, we adapt ideas from the *gPb* contour detection algorithm, using similar local and global cues in order to detect junctions from images.

In order to evaluate our junction detection algorithm, we capture object-level images. We run our implementation on these images and show that the different cues that we incorporate into our algorithm are useful for detecting junctions. Furthermore, although our algorithm requires many parameters that we do not have sufficient data to optimize, we can still achieve promising results. Our algorithm detects many false positives, but many true junctions are detected, and the angles detected at true junctions are quite accurate.

1.1 Definitions

We define *edges* as points in the image at which a line can be drawn such that image features change rapidly across the line. Image features, for our purposes, will include brightness, color, and texture. Edges are an example of a low-level feature, while high-level features are defined in terms of the object in the image. Contours are points in the projected image of an object where there is a discontinuity of depth or a discontinuity of surface orientation on the object. Contours and edges usually but not necessarily correspond. For example, discontinuities in surface paint color, such as the one highlighted in yellow in Figure 2, will be detected as edges but do not meet the definition of contours. Also, discontinuities in surface orientation, such as the one highlighted in green in Figure 2, will not always create sharp changes in image features. If, as in our example, the surfaces that meet at the contour are of similar brightness, color, and texture, the edge detector will not fire along the contour. While edges are defined such that they can be detected using purely local information, in order to detect contours, one must reason globally about the image. Reasoning about

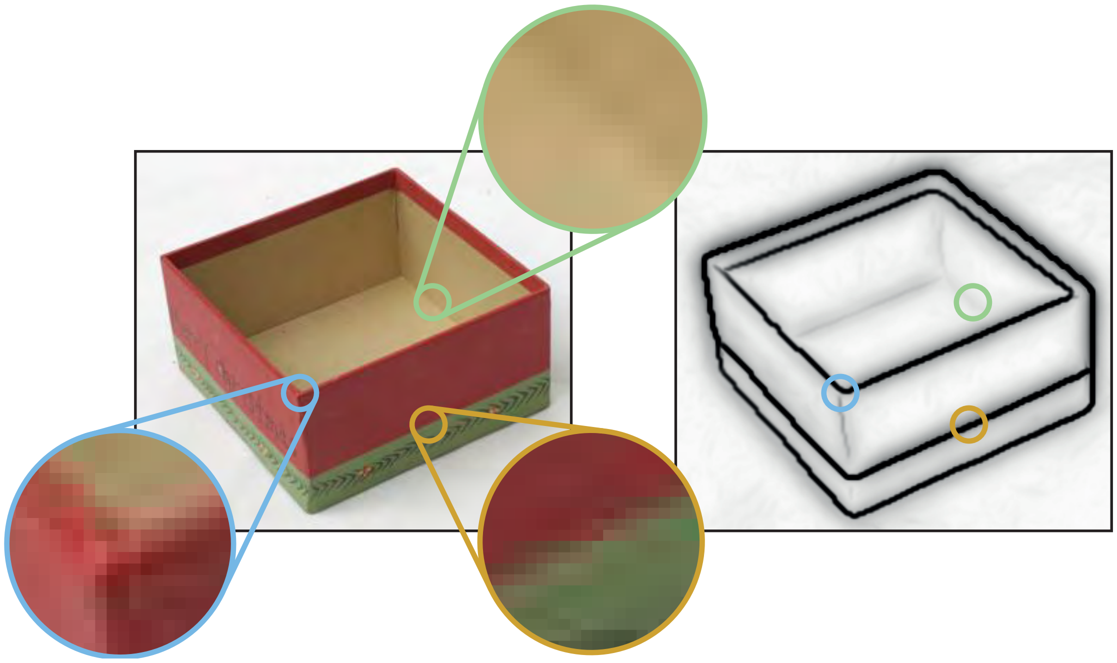

Figure 2: An image of a box and the output of a contour detector on this image (the *gPb* algorithm [1]). Features have been circled in corresponding colors in both images. *Blue:* A junction of degree three. Note the three wedge-like regions and poor contour detector performance. *Green:* A surface orientation discontinuity creates a contour, but does not create a strong edge. Note the similarity in image features across the contour. *Yellow:* An edge that is created by a paint boundary and is thus not a contour.

junctions is one way to reason globally about the image.

Object boundaries are defined to be points at which different objects are on either side of a line drawn through the point. Contours are not always object boundaries, but object boundaries are almost always contours. Contours are more useful for doing object-level analysis, while object boundaries are more useful for doing scene-level analysis. We use the term *scene* to refer to images with multiple objects in them. The image in Figure 1 from the Berkeley Segmentation Dataset [24] (top row) is a *scene-level image*, and the image from our dataset is an *object-level image* (bottom row).

Junctions can be defined at a low level as points at which two or more edges meet, or at a higher level as points at which two or more contours meet. The resulting neighborhood of a junction point will be comprised of two or more distinct *wedges*. Again, the high- and low-level definitions usually, but not necessarily, coincide. The degree of the junction is the number of edges or contours that meet at the junction. "Terminal junctions" of degree one are possible [21] (see, for example, center of Figure 6), but we do not consider them in our work. We focus mostly on junctions of degree three as the canonical case, but our work can be easily extended to junctions of higher degrees, and to junctions of degree two. (We define junctions of degree two to be points where the contour changes line label—see Section 2.3.1. Although junctions of degree two are interesting, they are not as problematic for contour detectors.)

We define a *contour image* is an image where the value of each pixel gives the probability of that point being a contour The contour image should also include junctions. We define a *segmentation* of an image assigns each pixel to a region, where regions correspond to objects or parts of objects.

1.2 Contour Detectors Near Junctions

Contour and boundary images are useful for applications such as object recognition [19] and 3D shape reconstruction [16, 17]. Boundary images are closely related to image segmentation, and segmentations of images are also used in object recognition [18, 31]. There has been substantial progress in contour and boundary detection in recent years [1, 14], to the extent that these contour detectors are robust enough that the detected contour images can successfully be used as input for some practical applications.

When we attempt to reason about an object's shape from its contour image, we find that junctions are critically important [2]. But even the best contour detectors consistently perform poorly near junctions, making the output less usable for applications. This failure is not surprising. Contour detectors rely on the idea that contours are boundaries between two regions, but near junctions, there are three or more regions. Comparing the blue circle (a junction) in Figure 2 to the yellow one (a contour), we can see that the local neighborhoods around a junction and a contour are very different. All contour detectors assume a model of a contour that is similar to the yellow neighborhood. Because the junction and contour neighborhoods are significantly different, every contour detector that assumes this model will not respond strongly at junctions and at contour points near junctions.

1.3 Motivation for Detecting Junctions

These two observations, that junctions carry significant information and that contour detectors perform poorly near junctions, suggest that finding a complete and useful contour image requires complementing a contour detector with a junction detector. The contour detector would find points where two regions meet, and the junction detector would find points where three or more regions meet. With a complete contour image, many interesting applications are available, such as reasoning about the shape of objects using contour images extracted from photographs. Thus, in this paper, we propose a junction detection algorithm.

We note that a contour neighborhood is just a special case of a junction neighborhood. Namely, the neighborhood of a junction of degree two where the regions are separated by a straight line will be identical to the neighborhood of a contour. Thus, one strategy for developing a junction detection algorithm is to take a high-performance contour detector and generalize it. For our junction detection algorithm, we start by modify Arbeláez et al.'s gPb boundary detector [1] by replacing the model of the contour neighborhood with the model of the junction neighborhood. We also incorporate new ideas that improve our junction detection.

As we develop, implement, and evaluate our junction detection algorithm, we keep in mind that one of the primary guiding motivations for our work is to be able to eventually reason about the shape of objects from natural images using contour images.

Chapter 2

Prior Work

In this chapter, we begin by introducing the gPb boundary detector, as we will adapt ideas from this algorithm. Next, we investigate the performance of gPb near junctions and discuss possible applications of junction detection. These two discussions motivate the need for a junction detector. Finally, we cover previous work on junction detection.

2.1 The gPb Boundary Detector

We have defined edges as sudden changes in intensity, color, or texture, and noted that boundaries often coincide with edges. One framework that has been successful in incorporating all of these cues is the gPb boundary detector, developed by Arbeláez et al [1]. We will keep this framework in mind when we develop our junction detector because junctions, as previously mentioned, are a generalization of edges. Edges can be thought of as degree-two junctions where the wedges cover π radians each.

The *gPb* algorithm aims to output a value $gPb(x, y, \theta)$, which is the probability that there is a boundary at the point (x, y) along the orientation θ for each point and orientation. The basic building block of the *gPb* algorithm is a difference operator, which is run at multiple scales on multiple channels (brightness, color, and texture) of the image. The response of the difference operator is used to create an affinity matrix whose elements indicate the likelihood that each pair of pixels are in the same region. A spectral clustering technique is used to find soft, global segmentations of the image. A gradient operator is applied to these segmentations, and these responses are combined with the original gradient responses to find the final probability of boundary at each point and orientation.

The *gPb* algorithm was originally developed for scene-level images, and was thus originally conceived of as a boundary detector. We refer to it as a boundary detector in its original context, but we use it as a contour detector, and refer to it as such later on. We discuss the implications of this discrepancy later.

2.1.1 Local Cues: *mPb*

The first part of Arbeláez et al.'s boundary detector is introduced by Martin et al. [23]. This portion of the algorithm is named Multiscale Probability of Boundary, or *mPb*. In this first step, points that match the model of an edge are found.

Here, an edge (x, y, θ, r) is modeled as a point (x, y) for which we can draw a line in the direction θ at a scale r such that the image features will be significantly different on each side of the line. An approximation to the gradient is computed by drawing a circle of radius r around each point, computing a histogram for the image features on each side of the line within the circle, and taking the difference between the histograms. The radius r determines the scale of the edge that we want to detect. For brightness, Martin et al. use the L^* channel from the CIELAB color space. The χ^2 histogram difference function is used to compute the difference:

$$\chi^2(g,h) = \frac{1}{2}\sum \frac{(g_i - h_i)^2}{g_i + h_i}$$

For color, the same process is done on the a^* and b^* channels. This computation is visualized in Figure 3.

Finally, a similar idea is executed on a texture channel. To create the texture channel, a bank of filters is convolved with the image at each point. At each point, the filter response vector consisting of the responses to each filter is created. These vectors are then clustered using k-means, and the centroids of the clusters are taken as the *dictionary of textons*. Textons function as the centers of the histogram bins in the texture channel. Thus, each point is associated with the texton it is closest to, and histograms over the dictionary are created for the two semicircles. These histograms are compared using the χ^2 function again.

Thus we have the probability that there is an edge at each point, in each direction, at each scale, in each of our brightness, color, and texture channels. Let us call this probability $C_i(x, y, \theta, r)$, where C_i is the ith channel (in this case we have four channels). Our Multiscale Probability of Boundary is then defined as a linear combination of these probabilities:

$$mPb(x, y, \theta) = \sum_i \sum_r a_{i,r} C_i(x, y, \theta, r)$$

where $a_{i,r}$ are weights learned from human-labeled training data (the Berkeley Segmentation Dataset [24]). The best scales, i.e. values of r, are also learned from the dataset.

2.1.2 Global Cues: sPb

The mPb detector, although relatively successful, only takes into account local cues. Thus, in later work, Arbeláez et al. [1] build on mPb and arrive at the Spectral Probability of Boundary, i.e. sPb. The general idea is to create an affinity matrix and use a spectral clustering technique to create a soft segmentation of the image, using global relationships between pixels. A gradient operator is used on the soft segmented image in order to create a signal that can be used to incorporate these global cues.

First an affinity matrix is generated such that for two pixels i and j,

$$W_{ij} = \exp\left(-\max_{p \in \overline{ij}} mPb(p)/\rho\right)$$

where $\overline{ij}$ is the line segment connecting i and j and ρ is a constant. Then, we define $D_{ii} = \sum_j W_{ij}$ and solve for the generalized eigenvectors $\{\vec{v_0}, \vec{v_1}, \ldots, \vec{v_n}\}$ of the system $(D-W)\vec{v} = \lambda D\vec{v}$ corresponding to the $n+1$ smallest eigenvalues $\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_n$. These eigenvectors hold contour information and can be interpreted as different soft segmentations of the original image (see Figure 3).

These segmented images are used similarly to the way the L^* channel, for example, is used. Arbeláez et al. convolve the image with smoothed directional derivative filters at multiple orientations at each point, but this is similar to using the previously defined difference operator, which takes histogram differences between semicircles around the point. Thus Arbeláez et al. define the second component of their edge

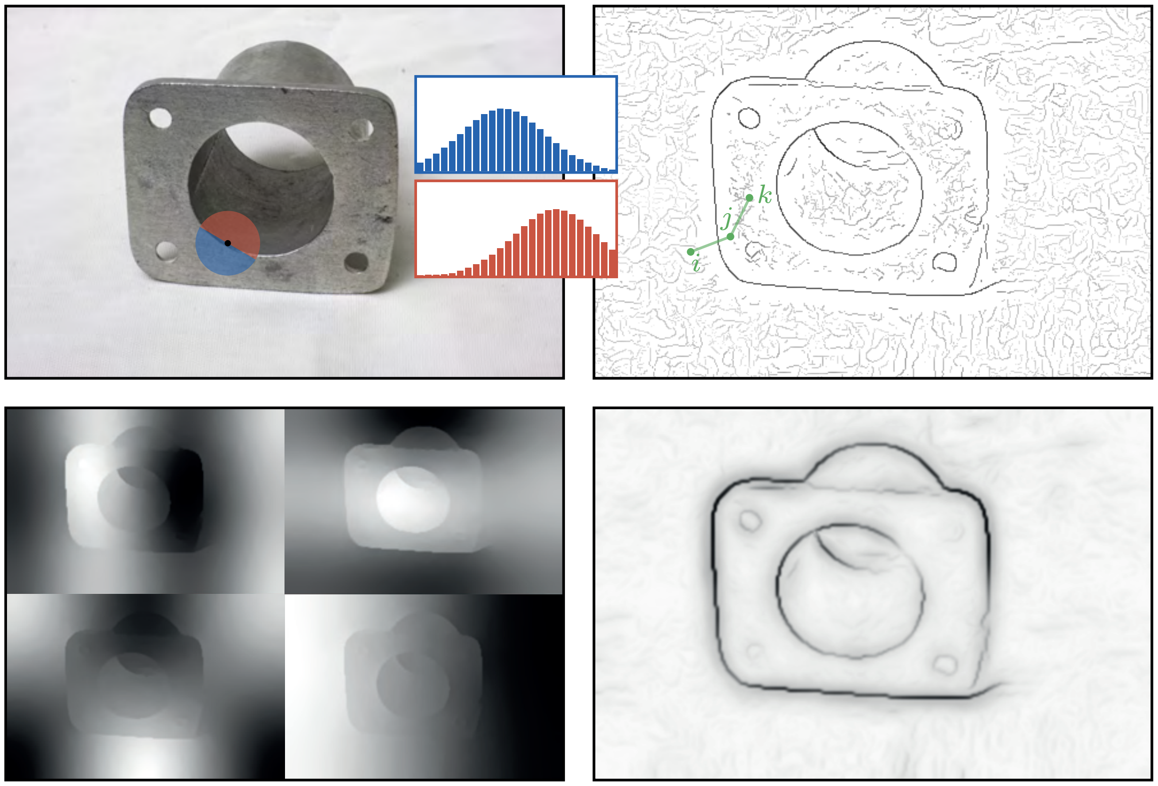

Figure 3: *Top left:* An image of a mechanical part. The red and blue histograms summarize intensities in the red and blue semicircles. The difference operator is defined to be the difference between the histograms *Top right:* The combined response of the difference operator on the channels at different scales, called mPb. Note that points i and j have low affinity, while points j and k have high affinity. *Bottom left:* Four eigenvectors from the spectral clustering of the affinity matrix, interpreted as images. The result of the gradient operator on these images is called sPb. *Bottom right:* The output of gPb on this image. Note that only magnitudes are visualized in this figure, but gPb also outputs orientation information.

detector:

$$sPb(x, y, \theta) = \sum_{k=1}^{n} \frac{1}{\sqrt{\lambda_k}} \cdot \nabla_\theta \vec{v_k}(x, y)$$

where ∇_θ is the smoothed directional derivative at the orientation θ.

The final probability of boundary given by the algorithm is then a linear combination of mPb and sPb, where the weights b_m and b_s are learned from a human-labeled dataset:

$$gPb(x, y, \theta) = b_m mPb(x, y, \theta) + b_s sPb(x, y, \theta).$$

Also, $gPb(x, y)$ is defined at each point to be the maximum over all orientations:

$$gPb(x, y) = \max_\theta gPb(x, y, \theta)$$

Finally, in order to visualize the output and evaluate performance, the information in $gPb(x, y, \theta)$ is thinned into one-pixel-wide lines.

2.2 gPb Near Junctions

Although the gPb boundary detector performs very well on the Berkeley Segmentation Dataset [1], it has trouble near junctions. This observation and our exploration of why this happens partially motivates our development of a separate junction detection algorithm.

A model of a low-level junction is shown in Figure 4. The model consists of three regions which meet at a point and are of different brightnesses. (We will only discuss junctions where brightness varies, but this discussion generalizes to other image features such as color and texture.) Although the wedges in this model are maximally separated in the brightness channel, the gPb output fails to be useful near the junction, as we can see in Figure 4. The response's magnitude along one of the edges falls off sharply as the points approach the junction, and the detected orientations change erratically.

We can see from our model junction why gPb has trouble near junctions. In Figure 4, we have drawn the semicircles for the gradient operator in the orientation for which there should clearly be an edge at the point. The gPb algorithm assumes that when we draw these two semicircles, the brightness should be homogeneous within the semicircles, and different across the semicircles. However, at this point, the assumption breaks down. Instead, we have two semicircles where one is of homogeneous brightness, but the other consists of two parts of very different brightnesses. In every case, we can number the regions 1, 2, and 3, in increasing order of their brightness. Then, in every case, along the edge between region 2 and region 3, when we draw the semicircles (as we have in Figure 4), the first semicircle will contain mostly points from region 2, and the second semicircle will consist of mostly points from regions 1 and 3. Thus, because the brightness of region 2 will be some positive linear combination of the brightnesses of regions 1 and 3, the average brightness of the semicircles will be very similar for some points along that edge. Although gPb does not rely on average brightness over the semicircle and histograms the values instead, with the smoothing of the histogram bins and noise, the algorithm is still subject to this kind of error and does not perform well at these points.

In fact, almost every edge detection algorithm will have trouble near junctions, because every edge detection algorithm has as an underlying assumption a model of an edge as being a boundary between two different regions. When a third re-

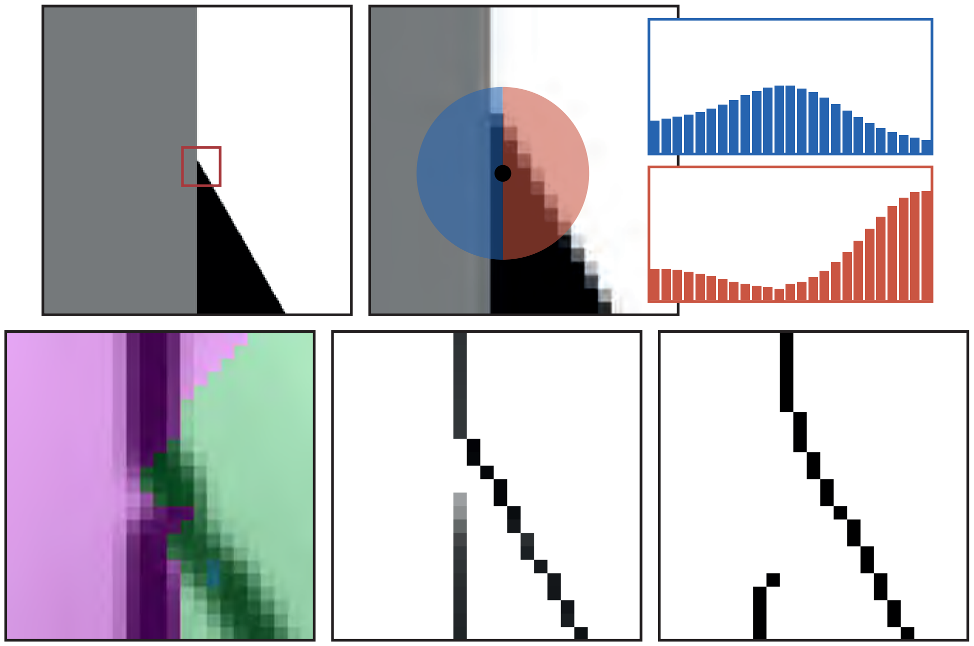

Figure 4: *Top left:* Our model of a junction. The red box indicates the small neighborhood that we will focus on. *Top right:* The gradient operator near a junction. Instead of only two regions as the *gPb* algorithm assumes, we have three regions that get summarized by two histograms, resulting in a smaller histogram difference. *Bottom left:* The response of the *gPb* operator near this junction. Darker values represent stronger edge responses. Color indicates the best edge orientation found for each point. Purple indicates a vertical angle, green indicates an angle $\pi/8$ counterclockwise from vertical, and blue indicates an angle of $\pi/4$ counterclockwise from vertical. This response is less than ideal. *Bottom center:* A non-maxima suppressed output of the *gPb* operator. The result of these errors in magnitude and orientation are made more clear by the visible gaps in the thinned output of the edge detector. *Bottom right:* The response of the Canny edge detector [5] to this model. Most edge detectors will not perform well near junctions.

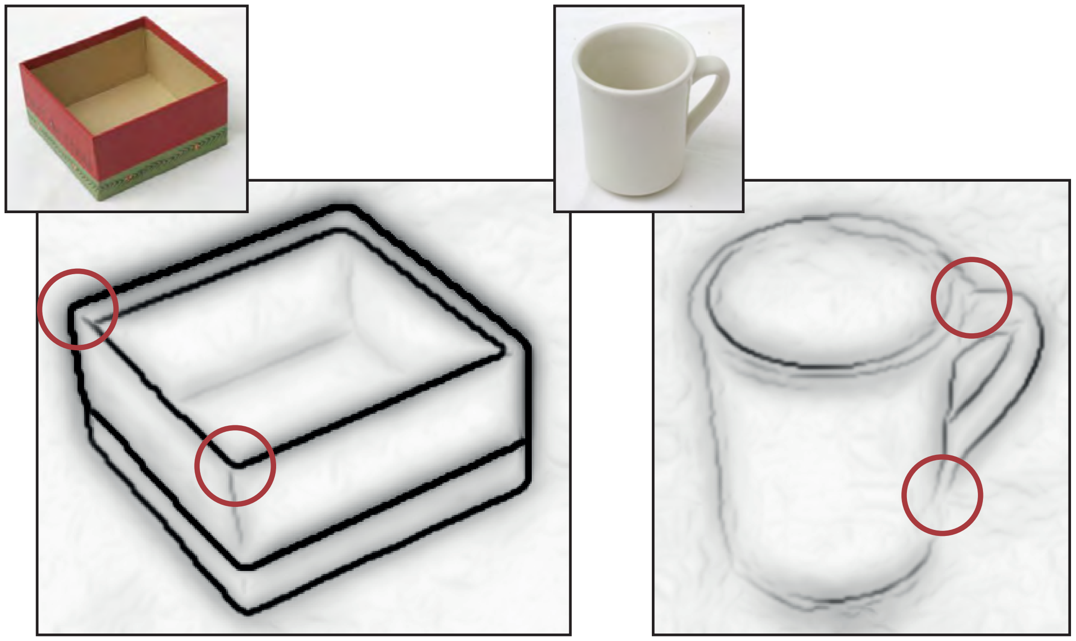
Figure 5: *Left:* A box and the output *gPb* on this image. *Right:* A mug and the output *gPb*. Poorly detected junctions are circled in red.

gion is introduced near an edge point, this assumption fails. This has been noted before. Beymer [4], for example, explores the properties of edge detectors based on directional derivatives near junctions, using models that are similar to ours. Beymer discovers that in almost every case, the edge detector will give incorrect results near the junctions. His analysis is particular to the details of the derivative operators, but the essential failure is the incorrectness of the assumed model of an edge. In Figure 4, we see the output of the classical derivative-based Canny edge detector [5] on our junction model and its inaccuracies. Beymer proposes a solution to this problem in derivative-based edge detectors, but his solution is not applicable to the *gPb* edge detector.

Furthermore, *gPb* performs even more poorly near junctions in natural images, leaving gaps where edges should intersect, which is demonstrated in Figure 5. This evidence should dissuade us from attempting to use the output of gPb or any other edge detector as our sole evidence for junctions. This is called the edge-grouping approach, and it interprets detected edges a set of line segments, and looks for junctions where they intersect. If we wish to localize junctions precisely and obtain a robust answer for how the edges meeting at that junction behave, we cannot rely on this approach too heavily. These observations also partially motivate the development of our junction detection algorithm.

2.3 Applications of Junction Detection

The poor performance of boundary detectors near junctions cannot be overlooked because of the importance of junctions in contour images. When we consider the potential applications of a contour image in understanding the image, it becomes clear that the contour image generated by a boundary detector is not sufficient, and must be supplemented by a junction detector. These potential applications of a junction detector not only motivate our algorithm, but also inform its development, as we attempt to obtain output that would be useful for these applications.

2.3.1 Line Labeling

One of the primary applications of contour images that motivates our work is line labeling. Finding a line labeling of a contour image can yield a great deal of information about the shape of an object, but requires a robust detection of junctions and contours. Malik [21] gives a description of line labelings for piecewise-continuous objects, and we will briefly describe his work here, which defines line labels and explains why they are useful. This problem is interesting because finding a line labeling requires robust detection of junctions. Futhermore, this is an attractive possible application of junction detection because a line labeling generates additional constraints on what configurations of junctions are possible, possibly providing cues for revising the original detection of junctions.

Definition and Application of Line Labels

First, recall that contours are points in the projected image of an object where there is a discontinuity of depth or a discontinuity of surface orientation on the object. Each contour, then, can be described by one of the following line labels:

1. '$+$' A convex discontinuity in the surface orientation.

2. '$-$' A concave discontinuity in the surface orientation.

3. '$\rightarrow$' A depth discontinuity occuring at a convex edge.

4. '$\twoheadrightarrow$' A depth discontinuity occuring at a smooth part of the surface where the tangent plane contains the view vector.

A line labeling of a contour image affixes one of these labels to each of the contours. Equivalently, we can think of the problem as one of junction labeling. The image structure graph is a graph where the nodes are junctions, and nodes are connected if their corresponding junctions are connected by a contour. Malik [21] gives an algorithm that outputs a line labeling given an image structure graph.

Such a line labeling, then, is useful because it provides many more constraints on the shape of the object than an unlabeled contour image. As just one example,

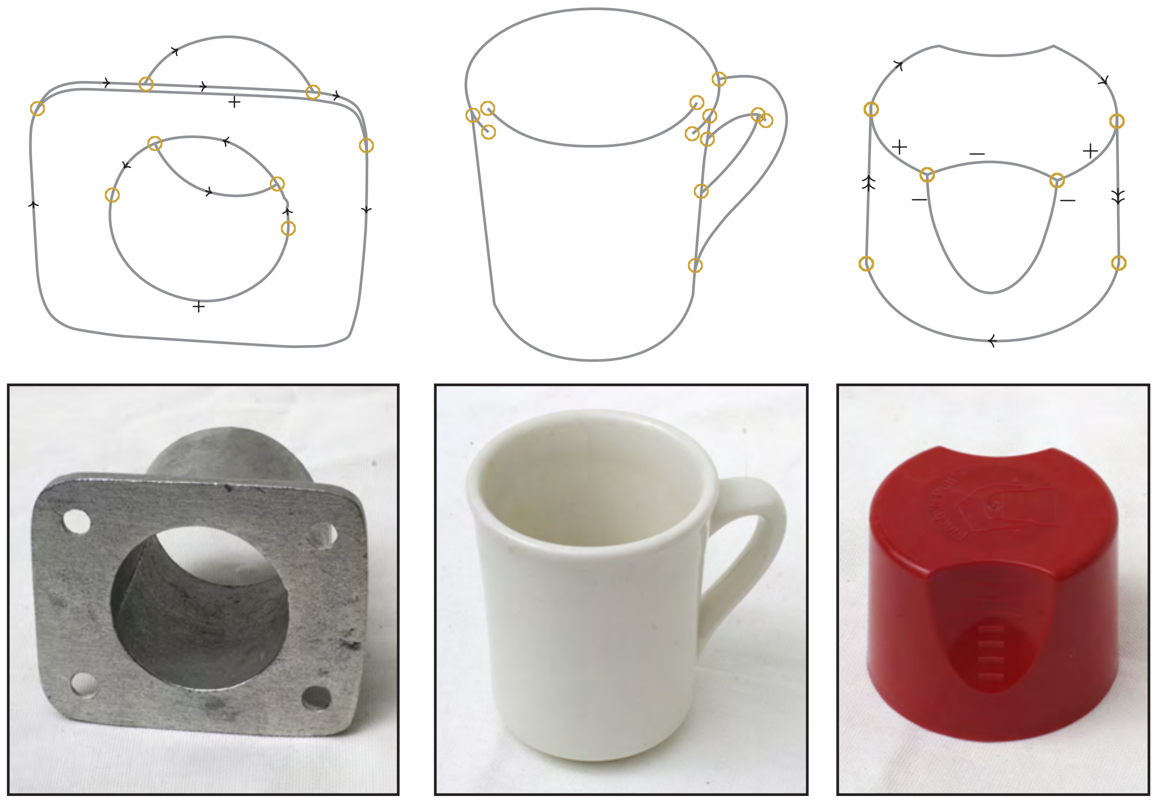

Figure 6: Ideal contour images are shown in gray above the full images. Junctions are marked by gold circles, and contours are labeled as described by Malik [21]. *Left:* A mechanical part. Note the two "phantom junctions". These points are junctions, as the contour changes from being a discontinuity of surface orientation to a discontinuity of depth in the image. These junctions are not detectable by low-level methods, and require reasoning about the other junctions. *Center:* A mug. Note the junctions of degree one. These are possible to detect using low-level processing, but require a different approach than the one we use, which assumes junctions of degree at least two. *Right:* A spray-paint can cap. Note the junctions of degree two. These are not edges, as the nature of the contour can change.

we know that the normal vectors to the surfaces that meet at a surface orientation discontinuity must yield a cross product that projects to a vector that is in the direction of the edge. In fact, Malik also presents a shape reconstruction algorithm that works with a line-labeled contour image as input [22].

Relationship of Line Labeling to Junctions

The image structure graph is a prerequisite for any line labeling algorithm, and in order to construct it, junctions and their connectivity must be accurately identified. Malik's algorithm for generating a line labeling from an image structure graph depends entirely on reasoning about how contours can come together. As just one example, two surface orientation discontinuities and a single depth discontinuity cannot meet at a degree-three junction. These kind of constraints allow a line labeling to be generated. Thus, the algorithm critically depends on an accurate idea of the location of junctions and their relationship to the contours.

Although Malik's line labeling algorithm works with ideal image structure graphs as input, it could potentially be modified to take image structure graphs where each node represents a possible junction and the probability that the node is indeed a junction. This could then take the output of our junction detection algorithm as input directly. Work by Rosenfeld et al. [30] suggests that it is possible to take image structure graphs with probabilities and compute the definite image structure graph that is most likely to be true.

This kind of post-processing after the inital detection of junctions is attractive because junction detection is difficult. Indeed, we see this kind of post-processing in many other junction detection algorithms [7, 32].

As a sidenote, we briefly justify our choice to focus on junctions of degree three. Malik's work [21] assumes that all vertices on an object will not have more than three faces adjacent to it. Given this assumption and the generic viewpoint assumption [13], we can then assume that all junctions will be of degree two or three. We will also take Malik's assumption in our work, as it will hold for many objects. However, work by Cooper [8, 9] expands the junction catalog for more complex objects. We will keep this possibility in mind, and when possible, leave our algorithm to be extensible to junctions of degree four and higher.

2.3.2 Object Recognition

Another possible application of junction detection is object recognition. One powerful framework for object recognition has been shape context. The shape context descriptor introduced by Belongie et al. [3], refers to a descriptor that describes the relative positions of other features at each feature. These features can simply be 2D points as in Belongie et al.'s work, connected contour segments as in Lu et al.'s work [19], regions as split up by contours as in Lim et al.'s work [18], or junctions as in Wang et al.'s work [33].

Furthermore, a psychophysical study by Attneave [2] on the perception of contour images suggests that junctions carry most of the information in the contour image. Attneave's cat (Figure 7) creates an image of a cat by connecting junctions of degree two and three with straight lines, and thus illustrates this principle. This importance of the junctions suggests that improving the contour image around junctions is

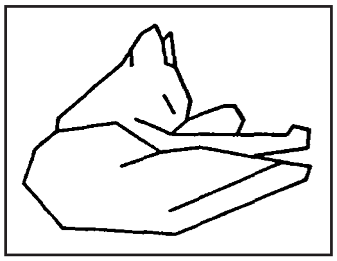

Figure 7: Attneave's cat [2] demonstrates the importance of junctions by reducing an image to its junctions of degree two and three connected by straight lines.

important for applications like object recognition with a shape context operator.

Another possible feature that could be used in a shape context-like operator is line-labeled contours. Such a shape context operator could be potentially much more powerful because line labels introduce significant constraints on the shape of the object near the contour. As far as we know, this has not been pursued, most likely because of the lack of a robust algorithm for generating line-labeled contours from natural images.

2.4 Other Junction Detectors

Finally, before we proceed, we will briefly cover existing junction detection algorithms. Although we motivate the development of our algorithm as a generalization of the *gPb* edge detector, we are also influenced by other junction detectors. The two main strategies in junction detection are template-matching and edge-grouping. Template-matching algorithms attempt to match a local area around a point with a model of a junction, and edge-grouping algorithms process the output of edge detectors. Both strategies can also be used together.

2.4.1 Template-Based Junction Detectors

Parida et al. [27] use a purely template-matching approach, using a model similar to ours, but there are significant differences. Their junction template only works with intensity images, and does not account for color and texture. Information that can be obtained by considering the image globally is not used. Instead of accounting for differences between wedges, Parida et al. focus on finding the wedges that will yield the minimum description length of the neighborhood, which is a less direct way of accounting for the properties that we look for in a junction. Also, their method simplifies the image more than ours does, reducing wedges to single intensity values rather than histograms. The algorithm's focus is on detecting the correct degree of the junction rather than the probability that a point is a junction. Finally, their method does not take into account any sort of detected edges. Nevertheless, the model of the junction and the method used to optimize the model were strong influences for our work. Parida et al. use a dynamic programming approach to their optimization, and our optimization problems also yield dynamic programming solutions with similar structures.

2.4.2 Corner Detectors

Corner detectors are similar to template-matching junction detectors and have received much more attention. Corner detectors, though, like the one developed by Harris and Stephens [15] simply look for points that are different from their surroundings. Junction points as we have defined them will almost always be corner points, but some corner points will not be junctions. For example, the center of a small black dot will be a corner point, but is certainly not a junction.

Both Regier [28, 29] and Trytten and Tuceryan [32] attempt to create comprehensive pipelines that takes grayscale images and outputs line labels. A critical component of this is junction detection. Both pipelines use derivative-based edge and corner detectors (similar to the Harris corner detector) to find contours and junctions as the first step in processing natural iamges. As these edge and corner detectors are not robust enough, both systems use post-processing and reasoning about what configurations of contours are physically possible to improve their results.

Because edges and corners are low-level concepts while contours and junctions are higher-level concepts driven by interpretation of an image as consisting of objects, this kind of post-processing is most likely inevitable, and will continue to be helpful in improving the outputs of contour and junction detectors. As we noted earlier line labeling is attractive as post-processing for precisely this reason.

2.4.3 Edge-Grouping Junction Detectors

Examples of purely edge-grouping junction detection algorithms are Maire et al.'s [20] and Matas and Kittler's [25]. Both take the output of edge detectors, interpret the edges as lines, find intersections of these lines, and determine whether these intersections are junctions. Maire et al.'s is interesting because it uses the output of the *gPb* boundary detector as its input. By finding junctions, it manages to improve the performance of the edge detector meaningfully. This evidence buttresses our claim that junction detection can aid in edge detection, even for a robust edge detector. Matas and Kittler's is also interesting, as further reasoning about the connectivity of junctions, using the output of their edge detector, is used to select which intersections qualify as junctions.

One of the problems with using a purely edge-grouping based method is that most edge detectors will have small gaps along contours. As a result, Maire et al.'s detector, for example, fires along straight contours, even though there are no junctions present. This is actually beneficial if we are interested in linking contours together, but not beneficial if we are interested in finding intersections of contours rather than simply piecing the contours together.

In this respect, Maire et al.'s junction detector bears significant resemblance to contour grouping algorithms. Contour grouping algorithms attempt to connect edges together into longer contours. There has been much work on this problem, [11, 10, 19, 34], and although we do not investigate these algorithms in detail, we find it interesting for two reasons. First, finding gaps in contours is similar to the edge-grouping approach to junction detection in general, because gaps in contours usually occur when a junction breaks them apart. Second, the principles used in grouping contours, such as parallelism and proximity [11], could also be used to group and reason about junctions after they have been initially detected.

Cazorla et al. [6, 7] combine template-matching and edge-grouping. They build off of Parida et al.'s work [27] for the template-based part of their approach, but they also incorporate the output of an edge detector on the image in a way that is similar to ours. Edges that lie along the angles between wedges are counted if they are oriented along the radial line. We will incorporate a similar term in our junction algorithm, which we will call "coincidence with edges."

Interestingly, though, Cazorla et al. [6, 7] also introduce a junction grouping algorithm which reasons about the connectivity of the junctions in order to correct errors in the original junction detection. This kind of post-processing is a persistent theme in junction detection schemes.

Chapter 3

Junction Detection

We will introduce an algorithm, which we call gPj, for detecting junctions. Our algorithm is informed by the gPb boundary detector, our discussion in Section 2.2 of why the gPb boundary detector (and edge detectors in general) perform poorly near junctions, and prior work on junction detection [7, 27]. The algorithm is guided and motivated by our discussion in Section 2.3 of the applications of detected junctions, such as line labeling.

3.1 Overview

Our model of a junction of degree k is a point at which we can find k radial angles to divide the neighborhood around the point into k wedges with certain properties. (This is depicted in Figure 8.) Our image features are brightness, color, and texture. In addition, we find soft segmentations of the image following Arbeláez et al. [1], and also use these as image features.

We compute the extent to which the wedges fit our model and call this gPj, our probability of junction. We expect the wedges to have three properties. Our first property, *difference between wedges*, captures the idea that each wedge should be different from its neighbors in its image features. Second, *homogeneity within wedges* captures the idea that each wedge should be homogeneous in its image features. Finally, the *coincidence with edges* term captures the idea that we should find strong contours detected along the radial lines that we have chosen. These terms are computed at different scales, then linearly combined to find the junction score for a given neighborhood and choice of wedges. Our algorithm, then, finds the radial lines that optimize the junction score and outputs the positions of these lines and the resulting junction score at each point. Finally, we offer a dynamic program for computing the optimal selection of radial lines.

Our algorithm incorporates template-matching and edge-grouping techniques. The difference between wedges term, which is a generalization of the difference operator from Arbeláez et al.'s work [1], and the homogeneity within wedges term are derived from a template-based approach. On the other hand, the coincidence with edges term, influenced by Cazorla et al.'s work [7] is derived from an edge-grouping approach.

3.2 Design Decisions

Recall that low-level junctions are defined to be points where two or more regions of distinct image features meet at a single point, and high-level junctions are defined as intersections of contours. We are primarily interested in detecting high-level junctions for use in applications, and we are only interested in low-level junctions because they

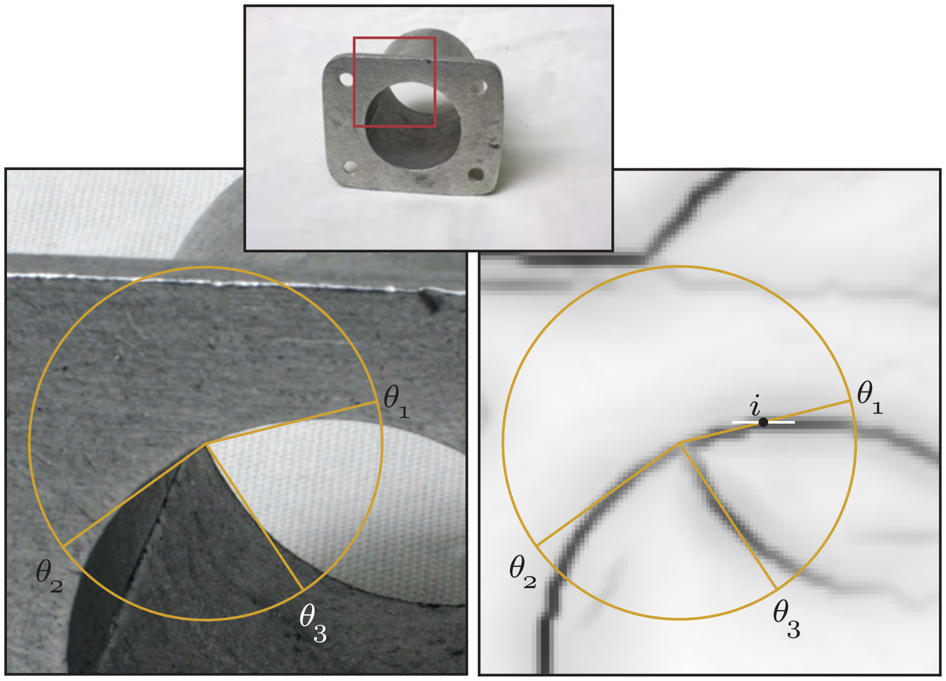

Figure 8: *Top:* A mechanical part. The image region we focus on is outlined in red. *Bottom left:* Our junction detector looks for angles θ_1, θ_2, and θ_3, assuming a degree-three junction. The angles shown are optimal, as the wedges are maximally different from each other and internally homogeneous. *Bottom right:* The output of the edge detector *gPb*. Contours that coincide with the angles we choose also contribute to the junction score. Contours are weighted by the degree to which they are aligned with the angle of the radial line that they lie on. The orientation of the edge at point i has been visualized with a white line. The edge at point i will have a large weight, as the orientation at i and θ_1 are similar.

usually but not always coincide with high-level junctions. The difference of wedges and homogeneity within wedges terms on the brightness, color, and texture channels correspond directly to our concept of low-level junctions.

However, some high-level junctions are not low-level junctions. Furthermore, a psychophysical study by McDermott [26] suggests that humans have difficulty recognizing junctions without global context. Our usage of the soft segmentations as image features and the coincidence with edges term contributes towards finding high-level junctions. The soft segmentations and the contour detector output both incorporate global information about the image.

Finally, junctions, like edges, can occur at different scales. Thus, we compute all of our terms at different scales and combine the results linearly.

3.3 Image Features and Definitions

We describe our algorithm in more detail. The first step is to obtain brightness, color, and texture channels as we do in the *gPb* algorithm. Let us call these channels $\{C^1, C^2, C^3, C^4\}$ where C^1, C^2, and C^3 are the L^*, a^*, and b^* channels, respectively, in the CIELAB color space. Let C^4 be the texture channel as described in Section 2.1. Also recall from Section 2.1 that using the output of the *mPb* part of the *gPb* algorithm, we create an affinity matrix and perform a spectral clustering on the image. From this, we obtain soft segmentations of the image. We treat these soft segmentations as channels, and call them $\{C^5, C^6, \ldots, C^m\}$. We also run the *gPb* algorithm on the image and have boundary magnitude and orientation available at each point.

Given a point $\vec{x_0}$ in the image, we wish to determine the probability that the point is a junction. We draw a circle of radius r around the point. Then, given two angles α and β such that $0 \leq \alpha \leq \beta < 2\pi$, let us define the wedge $W[\vec{x_0}, \alpha, \beta, r]$ to be the set of points $\vec{x}$ such that if θ is the angle between the vector $\vec{x} - \vec{x_0}$ and the x-axis, the distance from $\vec{x}$ to $\vec{x_0}$ is less than r, and $\alpha \leq \theta < \beta$. In other words, the wedge is defined by the circle of radius r and the two radial lines at angles α and β. Now, let us define $H^i_{\alpha,\beta}$ to be the histogram of the kernel density estimate of the ith channel for points in the wedge $W[\vec{x_0}, \alpha, \beta, r]$. (Figure 9 shows a graphical representation of these definitions.)

$$H^i_{\alpha,\beta,r}[h] = \frac{1}{n_W} \sum_{\vec{x} \in W[\vec{x_0},\alpha,\beta,r]} K(h - C^i[\vec{x}])$$

where h goes from 1 to n_H, where n_H is the total number of bins in our histogram. Also, n_W is the number of points in the wedge, and K is a kernel density estimate function. In our case, we will use a normalized Gaussian as our kernel density estimate function in the brightness and color channels. In the texture channel, we will not use any smoothing, so K will be the Dirac delta function. We refer to the work on the *gPb* detector [23] for an explanation of why these are reasonable choices. Finally, we must specify what we mean when we say that an angle is between α and β. We note that at first we require $0 \leq \alpha \leq \beta < 2\pi$, and say that an angle θ is between α and β if $\alpha \leq \theta < \beta$. But we can expand our definition to allow "wrapping around" the x-axis such that in the case where $0 \leq \beta < \alpha < 2\pi$, $H^i_{\alpha,\beta,r}$ is defined to be equivalent to $H^i_{\alpha,2\pi+\beta,r}$.

For now, let us assume we are looking for junctions of degree k. We then look for k distinct angles, $\vec{\theta} = \{\theta_1, \theta_2, \ldots, \theta_k\}$. Assume that the angles are sorted (so

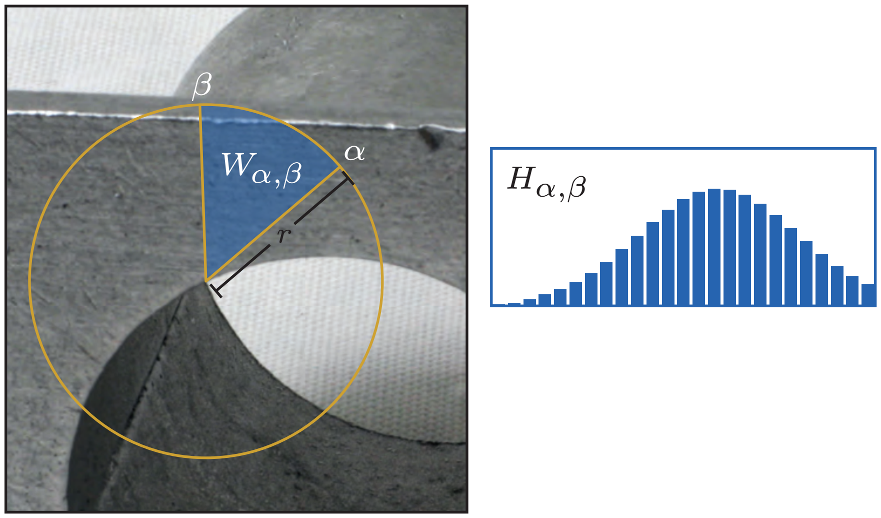

Figure 9: *Left:* Given a neighborhood of radius r and angles α and β, the points belonging to the wedge $W_{\alpha,\beta,r}$ have been visualized in blue. *Right:* The histogram summarizing $W_{\alpha,\beta,r}$, called $H_{\alpha,\beta,r}$ is visualized. Smoothing has been applied to the histogram.

$\theta_1 < \theta_2 < \ldots < \theta_k$). For notational convenience, we will consider the expression θ_{k+1} as referring to θ_1.

3.4 Terms

3.4.1 Difference Between Wedges

Now, we define our terms. The first, difference between wedges, captures the degree to which the wedges are different from each other in a certain channel. This term is the sum of the histogram differences between neighboring wedges. Note that we are not concerned with wedges being different from each other if they are not adjacent. We wish to maximize this term, similarly to how in the *gPb* algorithm, we pick the orientation at each point that maximizes the histogram differences between the two semicircles. In our case, though, instead of two semicircles, we have k wedges. (We drop the index into the channels i for notational convenience in the following equation.) where $d(H_1, H_2)$ is a difference function between two histograms. We use the χ^2 histogram difference operator, and again refer the reader to Martin et al.'s work [23] for why this is a reasonable choice. Thus,

$$d(H^1, H^2) = \chi^2(H^1, H^2) = \frac{1}{2}\sum_{h=1}^{n_H} \frac{(H^1[h] - H^2[h])^2}{H^1[h] + H^2[h]}$$

Note that our histograms are normalized to sum to 1, so the maximum value of our difference function is 1, and the maximum value of *dbW* is 1. We will design all of our terms to sum to 1 to make them easier to combine.

3.4.2 Homogeneity Within Wedges

The second term we define, homogeneity within wedges, captures the degree to which the wedges are homogeneous. We will want to maximize this term. Recall that for a histogram H, we have $H[h]$ elements in the hth bin, and there are n_H bins in total. (Again, we omit the i.) where $v(H)$ is a function that calculates some measure of statistical dispersion in the histogram. In our case, we will simply choose the statistical variance of the histogram. For the purposes of calculating this variance, we interpret the histogram to range from 0 to 1. Thus, the center of the hth bin will be $(h - 0.5)/n_H$.

$$v(H) = 4 \cdot \sum_{h=1}^{n_H} \left(H[h] \cdot \left(\frac{h - 0.5}{n_H} - \mu \right)^2 \right)$$

where μ is the mean of the histogram:

$$\mu = \sum_{h=1}^{n_H} H[h] \cdot \frac{h - 0.5}{n_H}$$

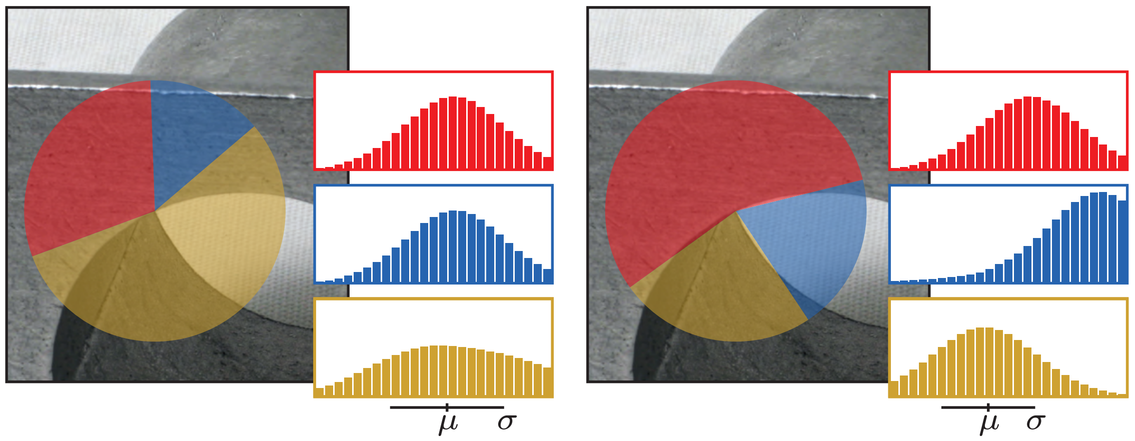

Figure 10: Two choices of angles and the resulting histograms. Wedges and histograms are visualized in corresponding colors. *Left:* A choice of angles that will not yield a high junction score. The difference between wedges will not be very high, as the red and blue wedges are nearly identical. The homogeneity within the yellow wedge will be low (mean and standard deviation visualized below the histogram for the yellow wedges). *Right:* The optimal choice of angles for maximizing our junction score measure. The wedges are maximally different from each other as we can see by comparing the histograms, and relatively homogeneous.

The maximum value of $v(H)$ is 1, due to the normalizing factor of 4 in the function. Thus, hwW will be between 0 and 1.

There many possible choices for the function $v(H)$, and more complex methods of evaluating the homogeneity of a region are available [12]. We choose statistical variance because it is computationally simple and gives acceptable results.

3.4.3 Coincidence With Edges

The final term we define, coincidence with edges (cwE), counts the number of edges detected by an edge detector that are located along the radial lines defined by the chosen angles. The angles that we select correspond to boundaries between regions in our model of the junction. Therefore, the edge detector should have detected edges

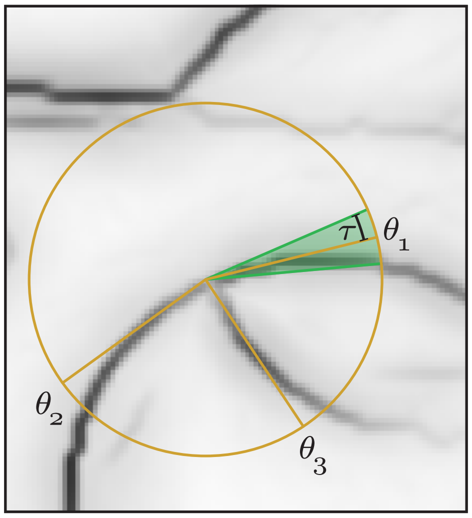

Figure 11: The wedge $W_{\theta_1-\tau,\theta_1+\tau,r}$ is highlighted in green. The detected edges (shown in black) inside this wedge will contribute to the coincidence with edges term for θ_1.

along this boundary that align with our selected angles. Although we note that edge detectors do not behave as desired near junctions in Section 2.2, if our radius r is large enough, we will still be able to glean usable data from the edge detector's output. Also, the edge detectors generally do not fire near junctions, producing fales negatives rather than false positives. Thus, our count of edges that contribute to the junction will be an underestimate, which is acceptable, as this term is used in conjunction with other methods of determining whether a point is a junction.

Edges contribute to the cwE term according to a weight determined by the degree to which they align with the radial line that they are near. Edge detectors will output, at each location, the probability that there is an edge at each orientation. For gPb, this probability is $gPb(\vec{x}, \theta)$. At each point, let us call the most likely orientation $\theta_{\vec{x}}$.

$$\theta_{\vec{x}} = \arg\max_{\theta} gPb(\vec{x}, \theta)$$

Then, we will call the probability that there is an edge in the direction $\theta_{\vec{x}}$ simply

$gPb(\vec{x}) = gPb(\vec{x}, \theta_{\vec{x}})$. We are now ready to define our term (again, we drop the channel index i):

$$cwE(\vec{\theta}, r) = \frac{1}{k}\sum_{j=1}^{k}\frac{1}{n_W}\sum_{\vec{x}\in W_{\theta_j-\tau,\theta_j+\tau,r}} |\cos(\theta_j - \theta_{\vec{x}})| \cdot gPb(\vec{x})$$

where τ is a threshold parameter that we will let be determined by our quantization of the angles and n_W is the number of points in the wedge and is included as a normalization factor. (Figure 11 shows a visualization of this term.) Note that any edge detection algorithm could be used for this term, as all we require is the probability that the edge exists and the orientation of the edge at each location in the image. Also, we assume that the probabilities outputted by the edge detector are between 0 and 1. Thus, as the magnitude of the cosine function is between 0 and 1, the entire term will vary from 0 to 1.

3.5 Combination

Now, we combine our three terms at n_r different scales. Let us call the chosen scales $\vec{r}\{r_1, r_2, \ldots, r_{n_r}\}$ We wish to choose angles that will maximize the differences between the wedges, maximize the homogeneity within the wedges, and maximize coincidence with edges. We can now define the expression that we wish to maximize at each location in the image:

$$gPj(x, y, \vec{\theta}) = \max_{\vec{\theta}}$$
$$\frac{1}{n_r}\sum_{r\in\vec{r}}\left(\left(\sum_{i=1}^{m} a_{i,r}\cdot dbW^i(\vec{\theta}, r)\right) + \left(\sum_{i=1}^{m} b_{i,r}\cdot hwW^i(\vec{\theta}, r)\right) + \Big(c_r \cdot cwE(\vec{\theta}, r)\Big)\right)$$

where the $a_{i,r}$s, $b_{i,r}$s, and c_r are positive parameters to be tuned. If these coefficients are chosen such that they sum to 1, the entire expression gPj will be between 0 and 1, because each of the three terms have been shown to be between 0 and 1. Also, we will call the optimal vector of angles $\vec{\theta^*}$

The final output of the algorithm, then, will be a probability that each point is a junction, gPb, and a vector of angles, $\vec{\theta^*}$, that tells us the angles that the edges incident to the junction are at.

3.6 Parameters

There are many parameters in this algorithm, and we briefly summarize them here. The difference function for histograms is $d(H_1, H_2) = \chi^2(H_1, H_2)$. The kernel density estimates for the histograms use a Gaussian kernel for all of the channels other than texture. The number of eigenvectors we pick, m, will be 16, and the number of histogram bins, n_H, will be 25. All of these choices follow Arbeláez et al. [1]. The variance function for histograms is simply statistical variance.

The radii that determine the scales we run the algorithm at, $\{r_1, r_2, \ldots, r_{n_r}\}$, and the coefficients in the final linear combination are more significant, and will be tuned by hand using our qualitative results. An interesting direction for future work would be to learn these from labeled data.

Another possible parameter the shape of the wedges. Namely, Parida et al. [27] add a hole in the middle of the neighborhood around each point. Thus, instead of considering a disc of points around each point, Parida et al. consider a donut. Parida et al. claims that this improves junction detection, and we consider the effects of this parameter in our experiments.

Finally, the degree of the junction, k, must be decided. First, let us revise our notation. Let gPj_k and $\vec{\theta}_k^*$ be the probability of junction and optimal angles when we assume that the junction must be of degree k. Second, we note that in our work, we limit our junctions to be of degree 3 or less. The objects in our dataset did not present junctions of degree greater than 3, and such objects are not very common (also see Section 2.3.1). Nevertheless, this is not a necessary limitation of the algorithm. There are many possibilities for computationally finding the optimal junction degree, but we leave this as an open issue for future work.

3.7 Implementation and Running Time

Now that we have defined our optimization problem, we will discuss how this can be implemented efficiently. Note that we are return to discussing the problem on the scale of a single point with a predetermined degree of the junction, k. Also, to simplify matters, we will assume that we are only running the computation at one scale, r. The first step in implementing the algorithm is to determine the resolution of the quantization of the orientations. In other words, we must pick a value p such that our possible orientations will be $\{0, 2\pi/p, 2 \cdot 2\pi/p, \ldots, (p-1) \cdot 2\pi/p\}$.

We reproduce the main terms to be computed below for reference:

$$dbW(\vec{\theta}, r) = \frac{1}{k} \sum_{j=1}^{k-1} \left(d \left(H_{\theta_j, \theta_{j+1}, r}, H_{\theta_{j+1}, \theta_{j+2}, r} \right) \right) + \frac{1}{k} \left(d \left(H_{\theta_1, \theta_2, r}, H_{\theta_k, \theta_1, r} \right) \right)$$

$$hwW(\vec{\theta}, r) = \frac{1}{k} \sum_{j=1}^{k} 1 - v(H_{\theta_j, \theta_j + 1, r})$$

$$cwE(\vec{\theta}, r) = \frac{1}{k} \sum_{j=1}^{k} \frac{1}{n_W} \sum_{\vec{x} \in W_{\theta_j - \tau, \theta_j + \tau, r}} |\cos(\theta_j - \theta_{\vec{x}})| \cdot gPb(\vec{x})$$

$$gPj(x, y, \vec{\theta}) = \max_{\vec{\theta}} \frac{1}{n_r} \sum_{r \in \vec{r}} \left(\left(\sum_{i=1}^{m} a_{i,r} \cdot dbW^i(\vec{\theta}, r) \right) + \left(\sum_{i=1}^{m} b_{i,r} \cdot hwW^i(\vec{\theta}, r) \right) + \left(c_r \cdot cwE(\vec{\theta}, r) \right) \right)$$

First, we note that the histograms, $H_{\alpha,\beta}$ can be precomputed for all α, β in $O(p^2 \cdot r^2)$ time, as each wedge has $O(r^2)$ points, and there are p possible values that α and β can each take on. (Recall that r is the radius of the neighborhood that we consider.) We can also define and precompute the homogeneity within wedges term for individual wedges in time $O(p^2 \cdot r^2 \cdot n_H)$, as the variance function takes time $O(n_H)$ time to compute (recall that n_H is the number of bins in the histogram).

$$hww(\alpha, \beta) = 1 - v(H_{\alpha,\beta})$$

Similarly, we can precompute the coincidence with angles term for individual angles, defining $cwE(\alpha)$ to be:

$$cwE(\alpha) = \sum_{\vec{x} \in W_{\alpha - \tau, \alpha + \tau, r}} |\cos(\theta_j - \theta_{\vec{x}})| \cdot gPb(\vec{x})$$

for all angles α. This will take time $O(p \cdot r^2)$ (a conservative overestimate). As a sidenote, the threshold parameter τ is determined by our quantization:

$$\tau = \frac{2\pi}{2p}$$

This choice of τ ensures that each detected edge gets counted towards just one angle.

Thus all of our precomputations together will take time $O(p^2 \cdot r^2 \cdot n_H)$.

3.7.1 Brute Force Algorithm

Assuming these precomputations, we can analyze the time complexity for calculating dbW, hwW, and cwE given a choice of angles, $\vec{\theta}$. dbW requires calculating our histogram difference function k times, and hwW and cwE each require summing k precomputed values. The histogram difference function, χ^2, and takes $O(n_H)$ time. Thus, having chosen an angle, the time complexity for computing all three terms will be $O(k \cdot n_H)$.

There are $O(p^k)$ possible values of $\vec{\theta}$. Thus, the brute force algorithm that simply computes $dBw + hwW + cwE$ for each possible value of $\vec{\theta}$ and finds the maximum will take time $O(p^k \cdot k \cdot n_H)$, in addition to the $O(p^2 \cdot r^2 \cdot n_H)$ time required for precomputation.

3.7.2 Dynamic Programming Algorithm

As an alternative to this exponential-time brute force algorithm, we can use a dynamic programming solution. In order to introduce the dynamic program, we will start by developing a dynamic program for minimizing a modified version of one term, hwW. We call this modified dynamic program hwW', and it seeks to minimize the expression

$$\sum_{j=1}^{k-1} 1 - v(H_{\theta_j,\theta_j+1})$$

Note that in contrast to hwW, this expression omits the homogeneity of the final wedge, $1 - v(H_{\theta_k,\theta_1})$. We will define the term $hwW'(\alpha, \beta, \ell)$ to be the sum of the variances of ℓ wedges that cover the angles in the interval $[\alpha, \beta)$. Thus our recursion is

$$hwW'(\alpha,\beta,\ell) = \min_{\varphi\in(\alpha,\beta)} (hwW'(\alpha,\varphi,\ell-1) + 1 - v(H_{\varphi,\beta}))$$

with a base case of

$$hwW'(\alpha,\beta,1) = 1 - v(H_{\alpha,\beta})$$

In order to solve for the expression that we actually desire, we must do another minimization to recover the final term that we left out of hwW':

$$hwW = \min_{\alpha,\beta} (hwW'(\alpha,\beta,k) + 1 - v(H_{\beta,\alpha}))$$

The time complexity of computing the dynamic program is $O(p^3 \cdot k)$, as α and β each can take on p values, ℓ takes on k values, and the minimization varies over $O(p)$ values. (We assume that we have precomputed values of H and v as mentioned earlier.) The final minimization is an additional $O(p^2)$ for a total time complexity of $O(p^3 \cdot k)$.

We can write a similar dynamic program, but because the dbW requires comparing each wedge with its neighbors, it requires a more complex dynamic program that yields a running time of $O(p^5 \cdot k)$. Considering that the brute-force exponential time algorithm takes time $O(p^k \cdot k \cdot n_H)$, and that we are unlikely to be concerned with junctions of degree 5 and higher, the brute-force algorithm will be faster than computing the results of this dynamic program.

In light of this, we propose an alternative value to maximize. Instead of computing the differences between the full wedges, we propose to approximate dbW using only local computations. For each wedge, we will not assume that we know the size of the neighboring wedges. Instead, we will simply take a small neighboring wedge from each side and use the difference as an estimate for the differnce from the full wedge. We will call our approximation difference next to wedge, or dnW.

$$dnW(\vec{\theta}) = \sum_{j=1}^{k} \frac{1}{2} \Big(d(H_{\theta_j-\delta,\theta_j}, H_{\theta_j,\theta_{j+1}}) + d(H_{\theta_j,\theta_{j+1}}, H_{\theta_{j+1},\theta_{j+1}+\delta}) \Big)$$

where δ is a parameter that decides how large the neighboring wedge should be.

This calculation, then, behaves just like the calculation of the variance of the wedges. We can define $dnW(\alpha, \beta)$ for a single wedge to be:

$$dnW(\alpha,\beta) = \frac{1}{2} \left(d(H_{\alpha-\delta,\alpha}, H_{\alpha,\beta}) + d(H_{\alpha,\beta}, H_{\beta,\beta+\delta}) \right)$$

This term can be precomputed for all values of α and β in time $O(p^2)$.

We can now fold the computation of dnW, hwW, and cwE into a single dynamic program, all of which involve only simple lookups into a precomputed table. The dynamic program gPj' gives us the best division into ℓ wedges of the angles between α and β.

$$gPj'(\alpha,\beta,\ell) = \max_{\varphi\in(\alpha,\beta)} \left(gPj'(\alpha,\varphi,\ell-1) + a\cdot dnW(\varphi,\beta) + b\cdot(1 - v(H_{\varphi,\beta})) + c\cdot cwE(\beta) \right)$$

where $cwB(\beta)$ on a single angle is defined above. Recall that a, b, and c are tuned coefficients for each term. The base cases are

$$gPj'(\alpha,\beta,1) = a\cdot dnW(\alpha,\beta) + b\cdot(1 - v(H_{\alpha,\beta})) + c\cdot cwE(\beta)$$

Finally, we need to account for the final wedge:

$$gPj = \max_{\alpha,\beta} \left(gPj'(\alpha,\beta,k) + a\cdot dnW(\beta,\alpha) + b\cdot(1 - v(H_{\beta,\alpha})) + c\cdot cwE(\alpha) \right)$$

As before, the dynamic program takes time $O(p^3\cdot k)$ time to compute, as we are still only doing lookups into precomputed tables inside the maximization. The final maximization takes time $O(p^2)$, so the overall time complexity of this algorithm is $O(p^3 \cdot k)$, plus the $O(p^2 \cdot r^2 \cdot n_H)$ required for precomputation. We should note that this is only asymptotically faster than the brute-force exponential algorithm, which has time complexity $O(p^k \cdot k \cdot n_H)$, when the degree of the junction is higher than three. Because our work only focuses on junctions of degree three and lower, we use the brute-force algorithm, but keep in mind that the running time need not be exponential if we wish to compute results for higher-degree junctions.

3.8 Further Optimization

We are left with a $O(p^3 \cdot k)$ algorithm, then, which we must run on every point in our image. This is not entirely unreasonable, but our implementation takes on the order of 100 minutes to find degree-3 junctions in a 200×200 pixel image. For reference, *gPb* on the same image on the same machine takes approximately one minute. One option to significantly reduce the running time by a constant factor is to not run the junction detection algorithm on image locations that are unlikely to be junctions. In order to do this, we need a fast way to obtain a reasonable estimate of the possible presence of a junction. The estimate does not need to be very accurate, as long as it reduces the number of points we need to consider but retains all true junctions.

One simple estimate that we offer is to simply sum up the number of edges that coincide with all radial lines from a point within a radius. This is similar to our coincidence with edges term, but we do not spend time finding angles for which there are many coinciding edges. Instead, we simply consider all possible edges. To determine if a point $\vec{x_0}$ might be a junction, we compute:

$$est(\vec{x_0}) = \frac{1}{|N(\vec{x_0})|} \cdot \sum_{\vec{x} \in N(\vec{x_0})} (gPb(\vec{x}) \cdot |\cos(\theta_{\vec{x}} - \theta_{\vec{x}-\vec{x_0}})|)$$

where $N(\vec{x_0})$ is the set of points within a radius r of $\vec{x_0}$, and $\theta_{\vec{x}-\vec{x_0}}$ is the angle from the x-axis that the vector $\vec{x} - \vec{x_0}$ makes. Computing this estimate is much faster, as it is linear in the number of points in $N(\vec{x_0})$ at each point in the image. On our reference machine, it takes on the order of 3 seconds to compute estimates for the 200×200 pixel image. We thus compute each point's estimated junction score, and only process points whose estimated junction scores are above a threshold.

Chapter 4

Data

We captured 103 images of 21 different objects in natural lighting conditions in a room. The objects were set against a white background. Each object was photographed several times from different viewpoints. Objects were chosen that have several identifiable junctions.

It is important to note the differences between our dataset and the Berkeley Segmentation Dataset (BSDS) [24], which is the dataset used to tune the gPb boundary detector. In the BSDS, as examples in Figure 14 demonstrate, the images are at the level of scenes. There are many objects in each image. In contrast, our dataset

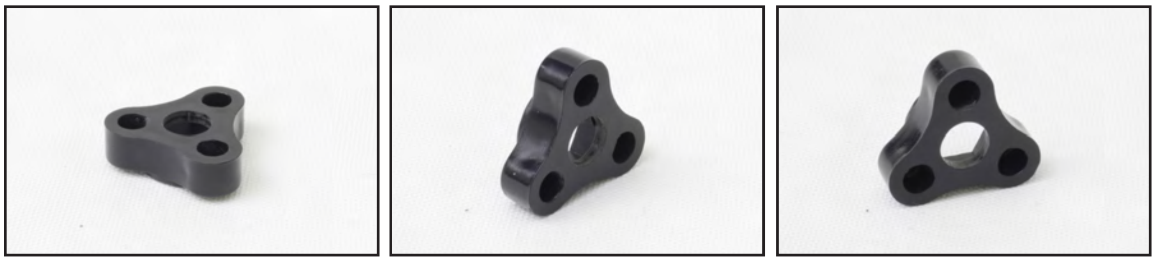
Figure 12: A mechanical part. Note that different viewpoints create images with different kinds of junctions.

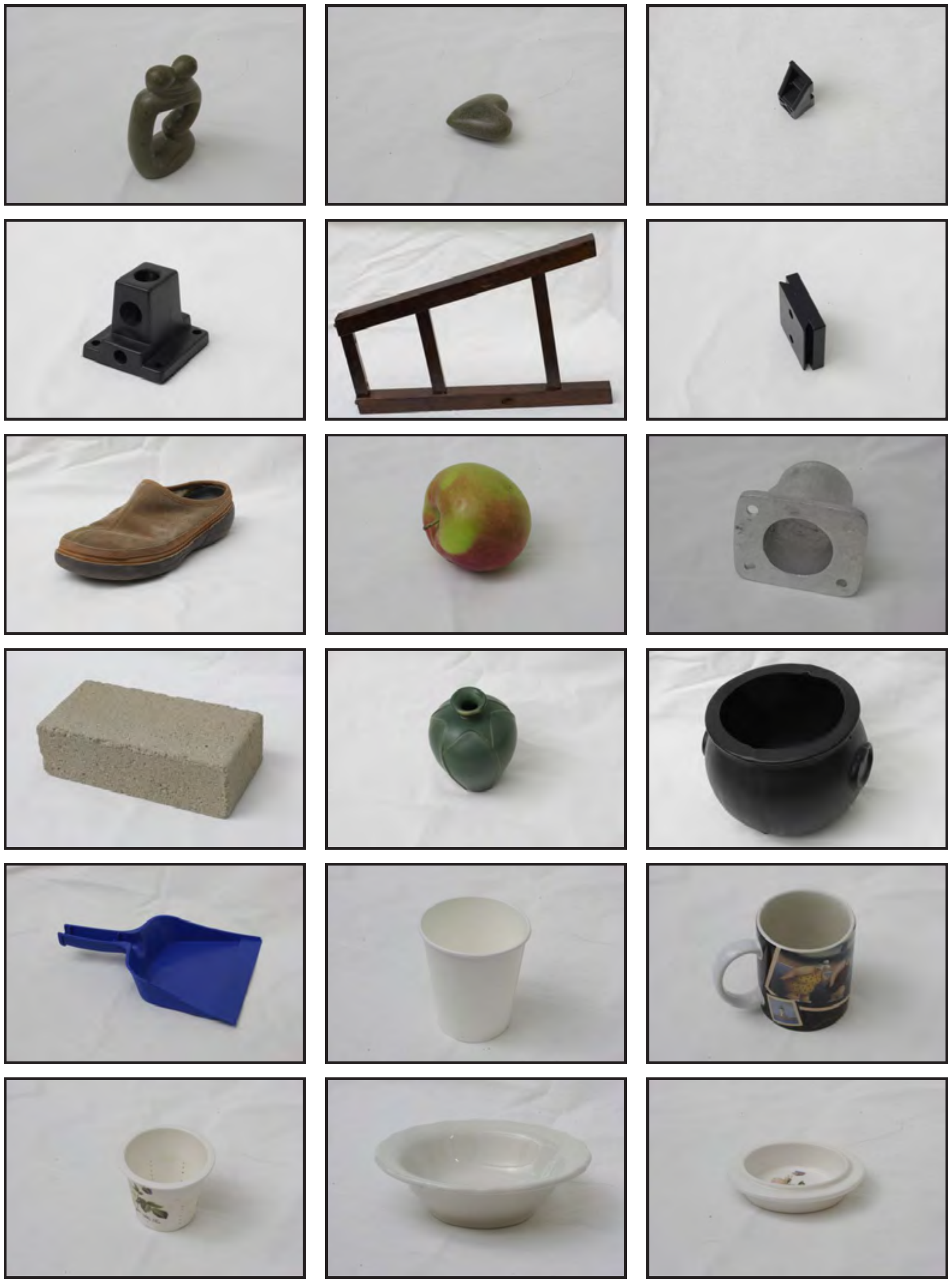

Figure 13: Example images from our dataset.

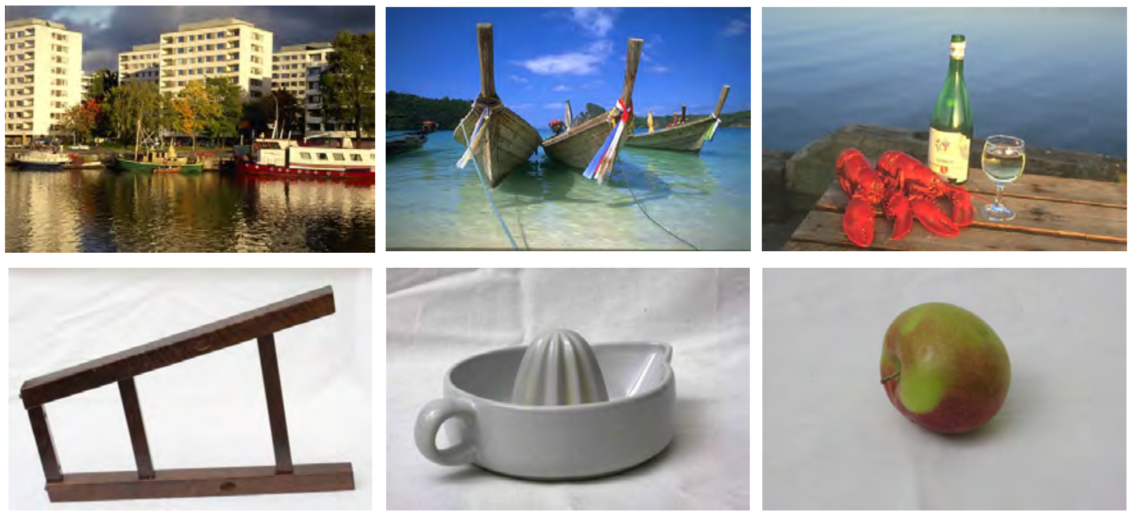

Figure 14: *Top row:* Data from the Berkley Segmentation Dataset [24], which was used to optimize and evaluate the *gPb* algorithm. *Bottom row:* Our images, which is of objects rather than multiple-object scenes.

focuses on single objects against a plain background. This is because our work deals with junctions at the object level. It is worth noting, then, that *gPb*, which is a critical input into our algorithm, could perform better if it were tuned to our dataset, or even another object-level database.

Although Maire et al. [20] use the BSDS to evaluate junction performance, we choose not to. Maire et a take human-labeled contour images, find ground-truth junctions from these, and evaluate junction performance on this datset. However, because we are motivated by applications of junction detection on object-level images rather than scene-level images, we chose not to use this evaluation metric. Also, Maire et al. do not explicitly test for the angles of the incident edges at each junction.

Chapter 5

Results

We run the implementation of our junction detection algorithm and explore how different parameters affect the output for images in our dataset. We also construct simple models of junctions and also use these to evaluate the different components of our algorithm. Although the parameter space is too large for us to optimize the algorithm without labeled data, we experiment and report qualitative results. Although not all of the terms are effective on all of the channels, the different terms and channels that we use in our algorithm seem to work in concert with each other, and are not overly redundant. Furthermore, we show that we can already achieve promising junction detection output without extensive manipulation of the parameters. On our dataset, although we detect many false positives, most of the true junction are detected. Furthermore, at true positives, the detected angles are almost always accurate.

5.1 Evaluation

Our algorithm was implemented in C++ and run on images from our dataset along with computer-generated models of junctions.

We only focus on junctions of degree three. The objects whose images we work with can only generate images that have junctions that have junctions of degree three or less if we take images from a generic viewpoint. Junctions of degree two are less interesting, because they are not as problematic for edge detectors (although a sharp bend in the contour may reduce the edge score, there are still only two regions for the edge detector to deal with). Junctions of degree one are very different from other junctions, and our template would have to be modified to fit junctions of degree one.

5.1.1 Organization of Terms and Channels

In order to identify the contributions of the terms in our expression for gPj, which we defined in Section 3.5, we run our algorithm using different terms on different channels in isolation, effectively setting the coefficients for other terms to 0. For example, if we wish to use only the difference between wedges term on the brightness channel, we would set $a_{1,r} = 1$, and set $a_{i,r} = 0 \forall i \neq 1$, set all $b_{i,r} = 0$ and $c_r = 0$ (recall that i indexes the channel and that brightness is the first channel).

We also group terms and channels in order to make sense of them. We will call the brightness, color, and texture channels *local channels*, and the soft segmented images *global channels*. The *difference term* will refer to the difference between wedges, the *homogeneity term* will refer to the homogeneity within wedges, and the *edge-grouping term* will refer to the coincidence with edges. We will group the terms and channels together into the following units: the difference term on local channels, the homogeneity term on local channels, difference and homogeneity on the global channels, and the edge-grouping term.

5.1.2 Visualization

At each point, the gPj algorithm will output a junction score, along with (for junctions of degree three) three angles that describe the directions in which the incident edges radiate from the point. One option for visualizing our output is to simply show the junction score at each point. However, the detected angles are not visible in this visualization. In order to visualize the angles, we will take only the points that meet a certain threshold and are maximal in a small neighborhood. At these points, we will draw lines in the detected directions of length proportional to the magnitude of the junction score. These representations of the detected angles will be overlaid on the original image. Figure 15 shows an example of both kinds of visualizations.

5.1.3 Initial Estimate

Recall that we perform an initial estimate of where we are likely to find junctions in order to improve the efficiency of our algorithm. This initial estimate simply counts the number of edges that coincide with radial lines from a point. Figure 16 visualizes the estimate at each point, and also shows the points that are selected when a threshold is applied.

Because we are only looking for an initial estimate, we are not concerned with obtaining precise results—we only need to ensure that all true junction points meet

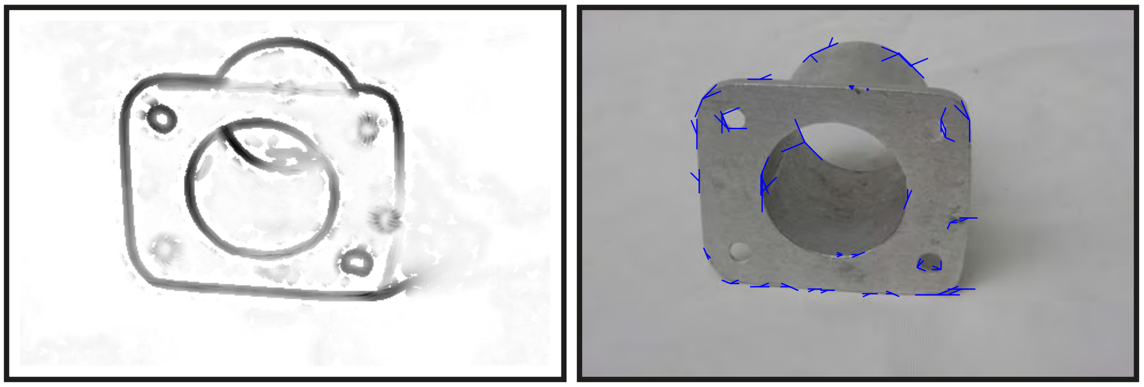

Figure 15: *Left:* The junction scores generated by gPj for all points of the image. Found angles are not shown. *Right:* gPj output with a non-maxima suppression filter and a threshold applied. For remaining points, the length of the lines is proportional to the strength of the junction, and lines are in the directions that optimize the junction score.

the threshold, while trimming as many points as we can for efficiency. Our crude estimate measure succeeds by this standard. For example, in Figure 16, we can see that no true junction points are eliminated by the threshold. All of the results we give in this chapter were run only on points that passed the threshold of the initial estimate.

5.2 Results on Models

We modeled junctions of degree three with different incident angles where the regions are distinct in the brightness, color, and texture channels. Examples of these models are in the top row of Figure 17. Before evaluating our algorithm on photographs, we confirm that our algorithm performs well on these models. The simplicity of our models makes it easier to identify the strengths and weaknesses of individual terms and channels, and our observations of this nature also apply when the junction

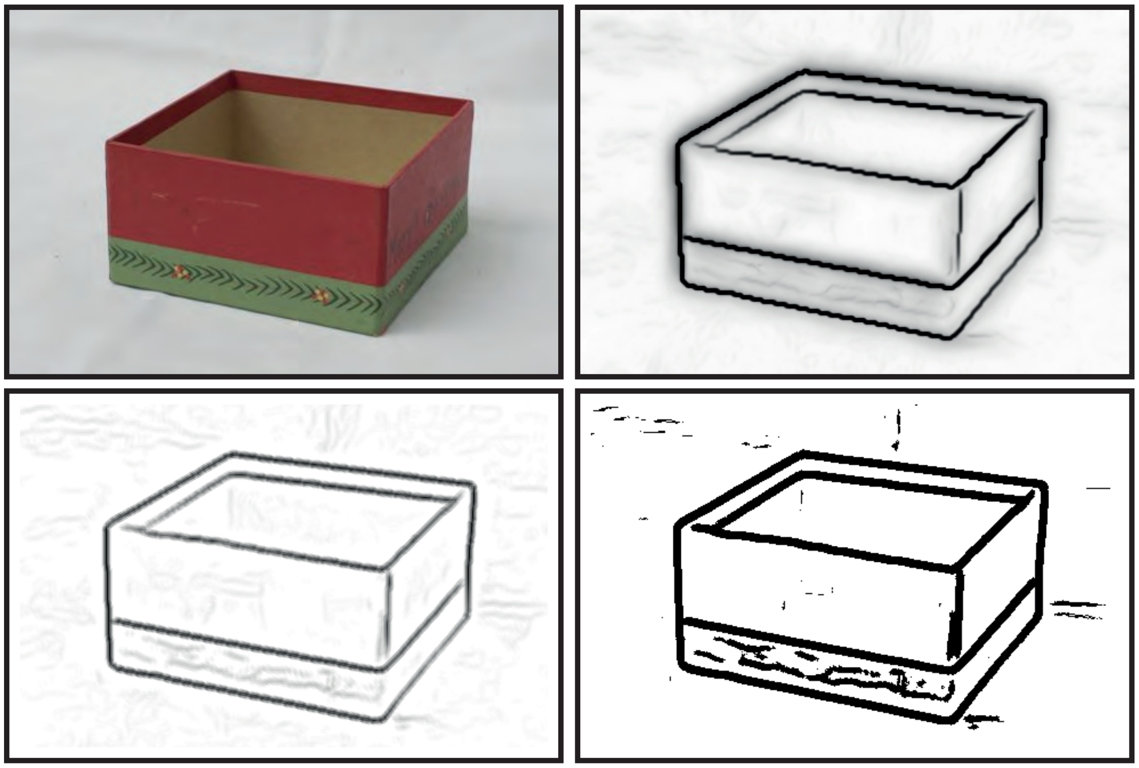

Figure 16: *Top left:* A box. *Top right:* The detected edges from the *gPb* algorithm. *Bottom left:* The initial estimate of possible junctions at every point. *Bottom right:* The points which will be considered, obtained by thesholding the estimate.

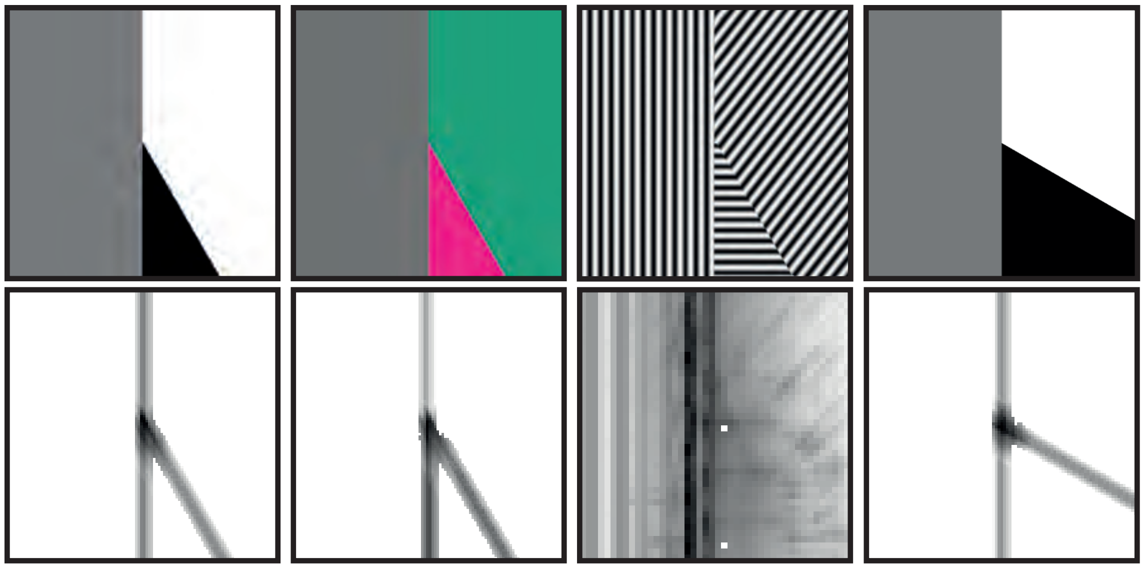

Figure 17: *Top row:* Modeled junctions of degree three in brightness, color, and texture channels, and with different incident angles. *Bottom row:* The junction scores at each point, cropped to the region of interest. The difference term was used on the relevant channel for each image. The detected angles are not shown but are correct.

detector is used on photographic images.

5.2.1 Local Difference Between Wedges

We demonstrate the results of running our junction detection algorithm on these models using only the difference term on only the local channels in Figure 17. We find that on these models, this term performs well, responding strongly at the junction and weakly elsewhere. Furthermore, the detected angle is always correct. The exception is the texture channel, which requires tuning the parameters involved in creating the texture channel to improve performance. We did not focus on the texture channel, as our objects were not strongly textured.

Note that in these ideal cases, the detected junctions surrounding the maximal

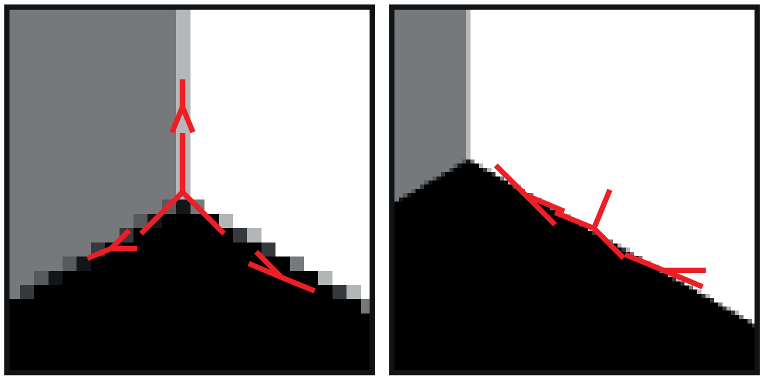

Figure 18: *Left:* Several points were selected (not necessarily maxima) and visualized to demonstrate that the detected angles are well-behaved as we move away from a junction, at least in the ideal case. *Right:* A visualization of the detected angles at edge points.

junction are well-behaved. By this we mean that the magnitudes fall off proportionally to the distance from the junction, and the detected angles remain relatively similar, but fold inward towards the junction as we move away (this is illustrated in Figure 18). Although performance on natural images will be much noisier, this behavior could be used to improve non-maxima suppression. Currently, we only use magnitudes non-maxima suppression, but we could also conceivably use the detected orientations. The *gPb* algorithm uses a similar technique for better non-maxima suppression.

5.2.2 The p-norm

Also evident from these results is that edge points, or junctions of degree two, are detected by our junction detector, even though we are looking for junctions of degree three. As we can see in Figure 18, the algorithm will select two angles that correspond to the actual edge, and a third angle somewhat randomly. Two of the differences between the wedges will be relatively high, but two of the wedges will be essentially identical. Our junction score, because we are only dealing with differences between wedges at the moment, is the sum of three terms: $gPj = dbW_1 + dbW_2 + dbW_3$. At these edge points, two of these terms will give high values, but the third is not. One solution to this is to combine the terms differently. Instead of taking the sum linearly, we could take:

$$gPj = dbW_1^p + dbW_2^p + dbW_3^p$$

where $0 < p \leq 1$. We call this using the p-norm, because this sum is similar to the norm of a vector. Changing the value of p did not significantly affect our algorithm's performance on our models, but did have some effect on photographs.

5.2.3 Homogeneity Within Wedges

The homogeneity within wedges term is not useful in isolation, as the results in Figure 19 demonstrate. On its own, the homogeneity term will be high for many points, including obviously non-junction points where there is little variation in image features. However, even in these results, we can see how the homogeneity term might be helpful. Points near the junction score low for homogeneity, but a handful of points that are very close to the junction score high. Thus, the homogeneity term, when used in conjunction with other terms, could help localize junctions and eliminate points near junctions from consideration. Points near junctions will score high on other terms, such as the difference term, so the homogeneity term will complement the difference term in this way.

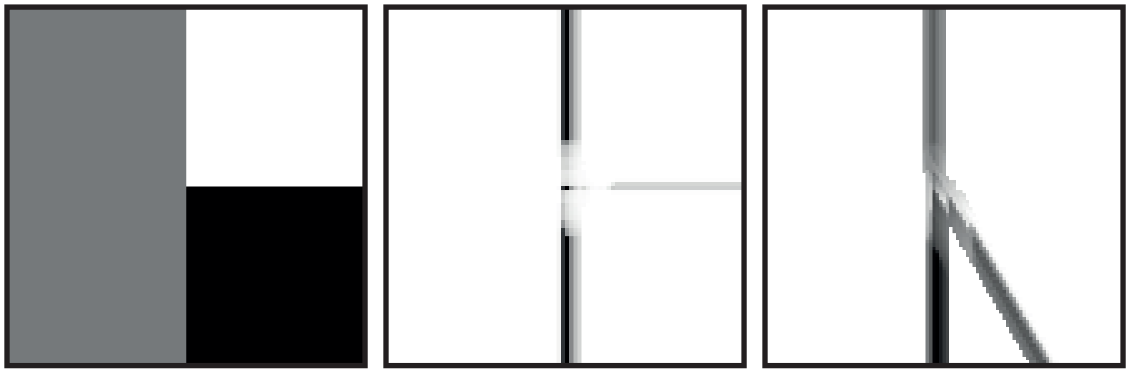

Figure 19: *Left:* A model of a degree-three junction. *Center:* The junction scores when using the homogeneity term on this model. *Right:* The junction scores when using the homogeneity term on a similar model. This term is not useful on its own, but is useful in conjunction with other terms. The junction detector outputs have been cropped to the area of interest.

5.2.4 Global Soft Segmentation Channels

Using the difference and homogeneity terms on the soft segmentation channels does not yield good results on our model images, as we can see in Figure 20. The central junction is detected, but many extraneous junctions are also detected. The reason for this is evident when we look at the soft segmentations themselves (in Figure 20). The soft segmentations break up large regions into smaller ones, which is why the normalized cuts segmentation algorithm, which uses a spectral clustering technique similar to what we do, suffers from the "broken sky" problem. Regions that were homogeneous in the original image are split up in the soft segmentation, and the junction detector finds the junctions of this incorrect segmentation.

This problem is exacerbated by the large, flat regions in our model, but the same problem occurs for the same reason in more complex photographic images. Nevertheless, processing the soft segmentation channel does result in finding true junctions as

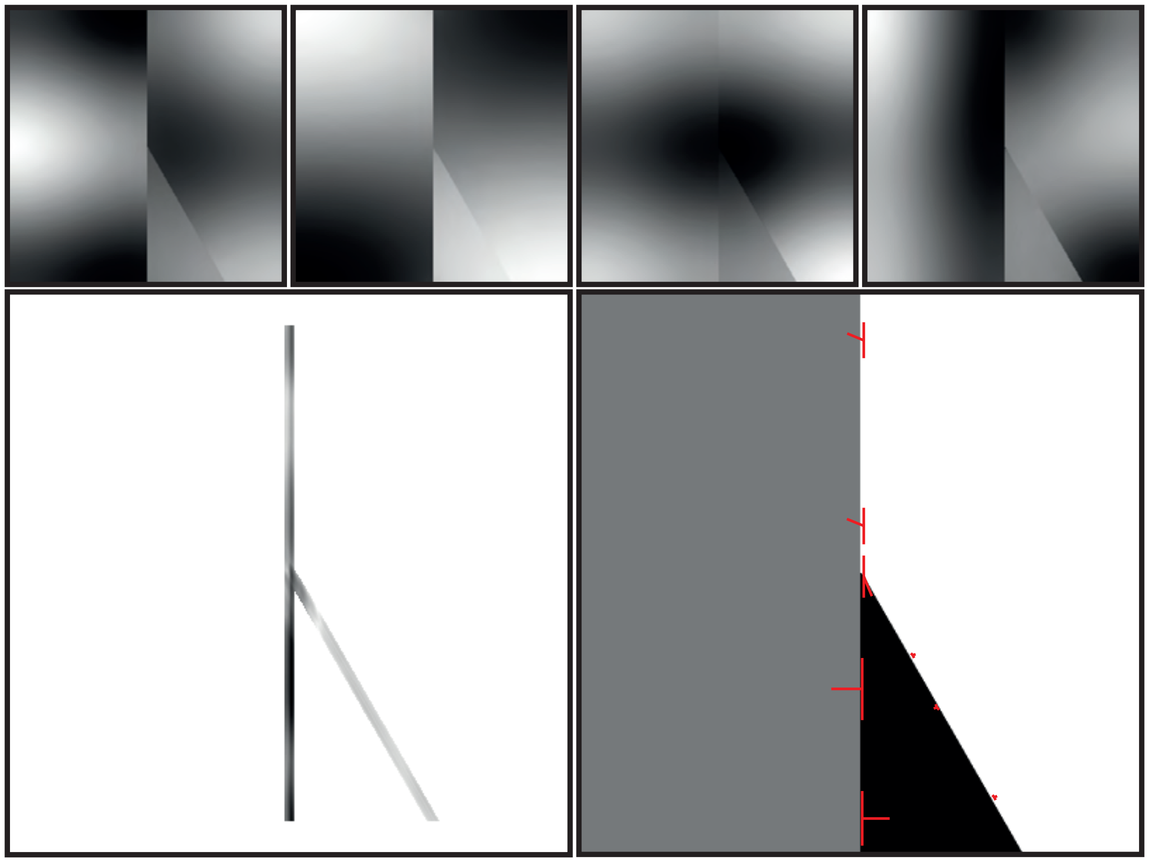

Figure 20: *Top:* Four soft segmentations of a model image. *Left:* The junction scores at each point when the difference term is used on global channels. *Right:* The non-maxima suppressed visualization of the detected angles when the difference term is used on global channels.

well as false ones, and it is more useful for photographic images than for our model.

5.2.5 Coincidence With Edges

Finally, we turn to the edge-grouping term, coincidence with edges. As shown in Figure 21, this term is effective for our models. But, as we have mentioned earlier, contour detectors do not detect edges well near junctions in photographic images, which implies that this term will not suffice on its own.

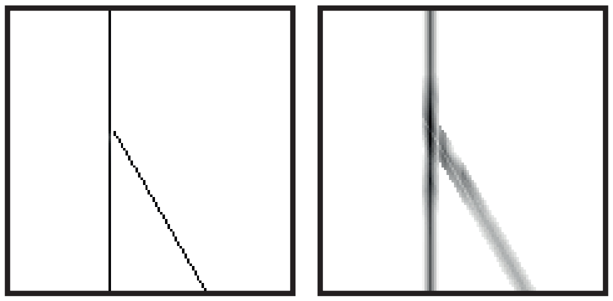

Figure 21: *Left:* The thinned output of the *gPb* contour detector on a model. *Right:* The junction scores using the edge-grouping term on the same model.

5.3 Results on Photographs of Objects

We also apply our junction detector to photographic images. We obtain an idea of how using different terms and channels affects performance and qualitatively evaluate the performance of our junction detector. We show that although our junction detector in general generates many false positives, it does detect most true junctions. As we mentioned before, we can reason about junctions and eliminate junctions that are physically inconsistent in a post-processing step based on a physical understanding of the object. Creating junctions in a post-processing step is much more difficult. This means that false negatives are more problematic than false positives. In addition, when junctions are detected, the detected angles are almost always correct.

Throughout this section, we have hand-labeled detected junctions that appear to be "close" to actual junctions in location and angles of incident edges. These "close" junctions will be colored red, while spurious detected junctions will be colored blue.

5.3.1 Scale

One difference from the model images is that photographic images contain junctions at different scales. By scale, we refer to the radius r of the neighborhood which we analyze in each point (see Section 3.5). As we show in Figure 22, some junctions are only detected at a small scale and some are only detected at a large scale. In the interest of developing a general junction detector, we compute our terms over multiple scales and combine them linearly. Note that this is not the same as adding the resulting junction scores together, because we need to account for the angles at each point. In other words, $dbW(\vec{\theta}, r_1)$ may be maximal for one $\vec{\theta}$, and $dbW(\vec{\theta}, r_2)$ may be maximal for a different $\vec{\theta}$. We thus need to find

$$\max_{\vec{\theta}} \left(dbW(\vec{\theta}, r_1) + dbW(\vec{\theta}, r_2) \right)$$

rather than

$$\max_{\vec{\theta}} \left(dbW(\vec{\theta}, r_1) \right) + \max_{\vec{\theta}} \left(dbW(\vec{\theta}, r_2) \right)$$

When we combine scales in this way, we see (in Figure 22) that using multiple scales results in detecting junctions at both scales.

Another parameter related to scale is the shape of the neighborhood at each point. Parida et al. [27] notice improvements in their junction detector by omitting a pixels from the center of the neighborhood around each point, and they cite psychophysical studies that suggest humans use a similar technique. We saw improvements when using the edge-grouping term, as we show in Figure 23. The difference and homogeneity terms were largely unaffected by changing this parameter.

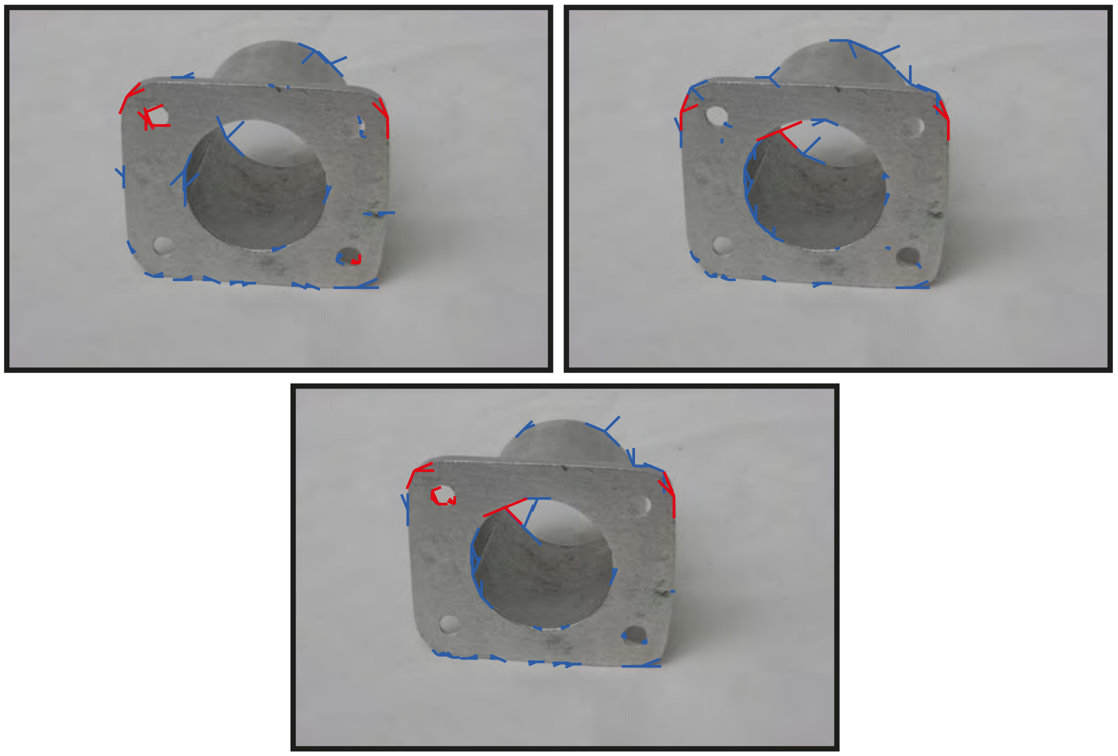

Figure 22: *Top left:* Junctions detected at a scale of 12 pixels with the difference term on local channels. *Top right:* Junctions detected at a scale of 24 pixels with the difference term on local channels. *Bottom:* Junctions detected when results from both scales are combined. Note that some of the junctions from each scale are detected in this version. Throughout this section, junctions that are qualitatively "close" to true junctions will be highlighted in red.

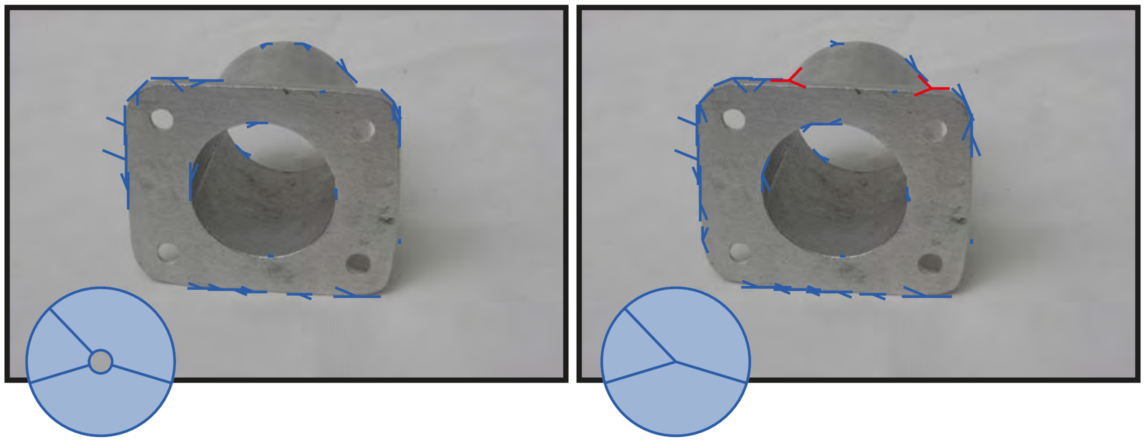

Figure 23: *Left:* Junctions detected using the full neighborhood with the edge-grouping term. (In both images, The blue circle visualizes the neighborhood used at each point.) *Right:* Junctions detected omitting the center of the neighborhood with the edge-grouping term.

This suggests that some parameters might affect some terms and channels but not others. Two examples from our qualitative testing are that omitting the center of the neighborhood only seems to affect the edge-grouping term, while the global channels seem impervious to changes in the scale parameter. These kinds of observations have implications for any attempt to learn parameters from labeled data.

5.3.2 Local Channels

As we noted earlier, the homogeneity term is not useful on its own, but is useful in conjunction with other terms when applied to the local channels. We demonstrate the result of using the difference term and the homogeneity term in Figure 24. For the box (on the left), adding the homogeneity term adds two additional junctions that were not detected when using only the difference term. For the mechanical part (on the right), adding the homogeneity term did not improve results significantly. We find this improvement in the detection of junctions in the image of the box encouraging, as it suggests that even though the homogeneity term alone produces what looks like noise, it can be combined with other terms to create meaningful output.

Note that the junction detector on the local channels detects “junctions” where two of the incident edges are contours, but the third is a paint discontinuity (Figure 24, top left). This is a persistent theme in contour and junction detection. The low-level concept of a junction and the high-level concept of a junction do not always coincide, and fixing these errors requires reasoning about the physical structure of the image in the way that, for example, line labeling does.

5.3.3 Global Channels

The homogeneity term is useful when applied to the local channels, but is not nearly as effective when applied to the global channels. For example, in Figure 25, we can see that adding the homogeneity term to the difference term in the global channels merely results in a noisier output.

Also, we observe the same problems with the global channels on photographic images as we did on models. The global channels are soft segmentations that split up large regions. For example, in Figure 25, we see that the background of the image is split up by the soft segmentation. Junctions are thus detected at the intersection of these splits and true contours. Two of the detected angles follow true contours, but the third angle follows the false split generated by the soft segmentation. Nevertheless, the global information provided by these soft segmentations is valuable, and does detect some junctions that are not detected in the other channels. Despite the problems

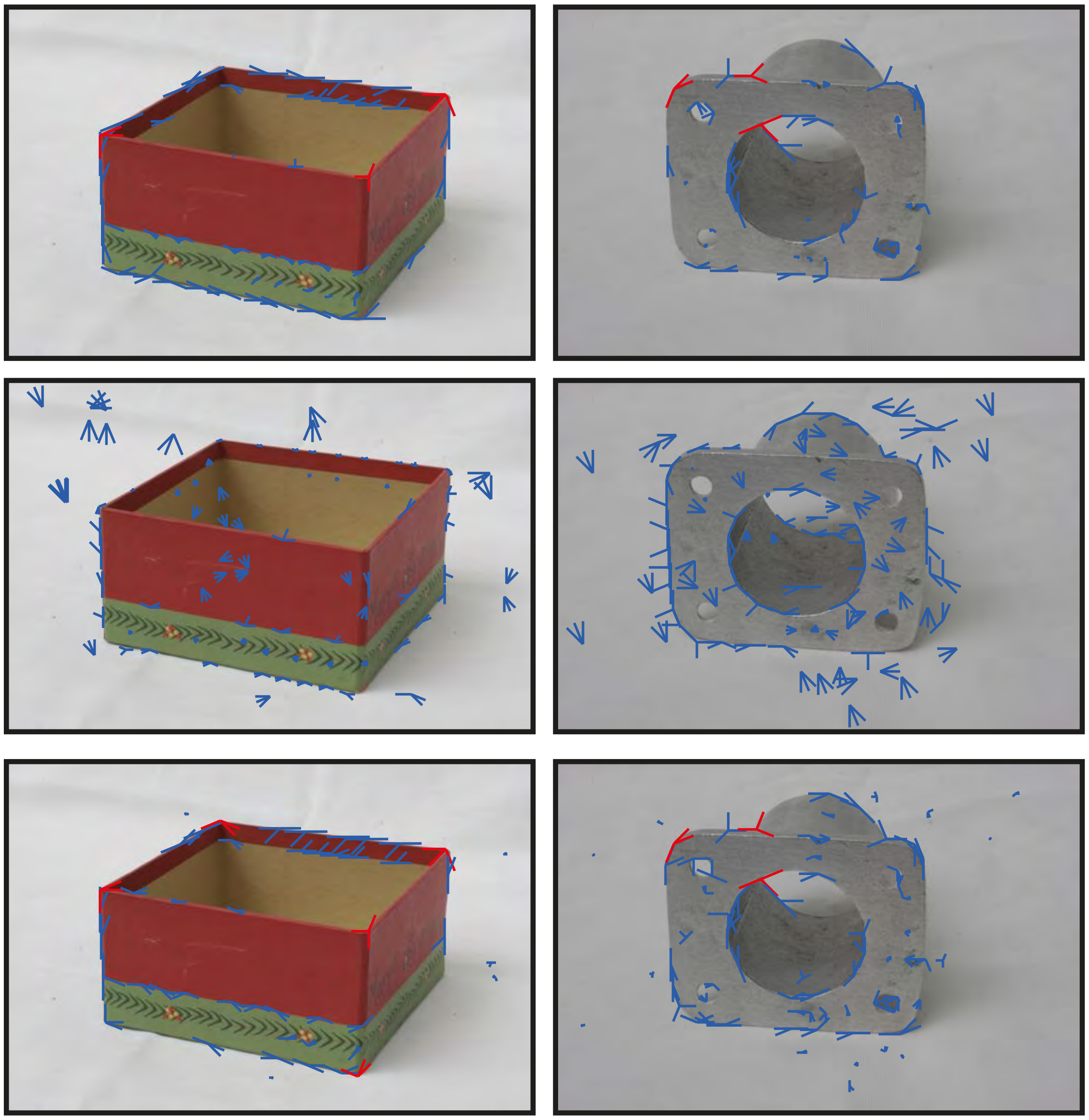

Figure 24: *Top row:* The output when using the difference term on local channels using a p-norm where $p = 0.5$. *Middle row:* The output when using the homogeneity term on local channels. *Bottom row:* The output when combining the difference term and homogeneity term on local channels using a p-norm where $p = 0.5$.

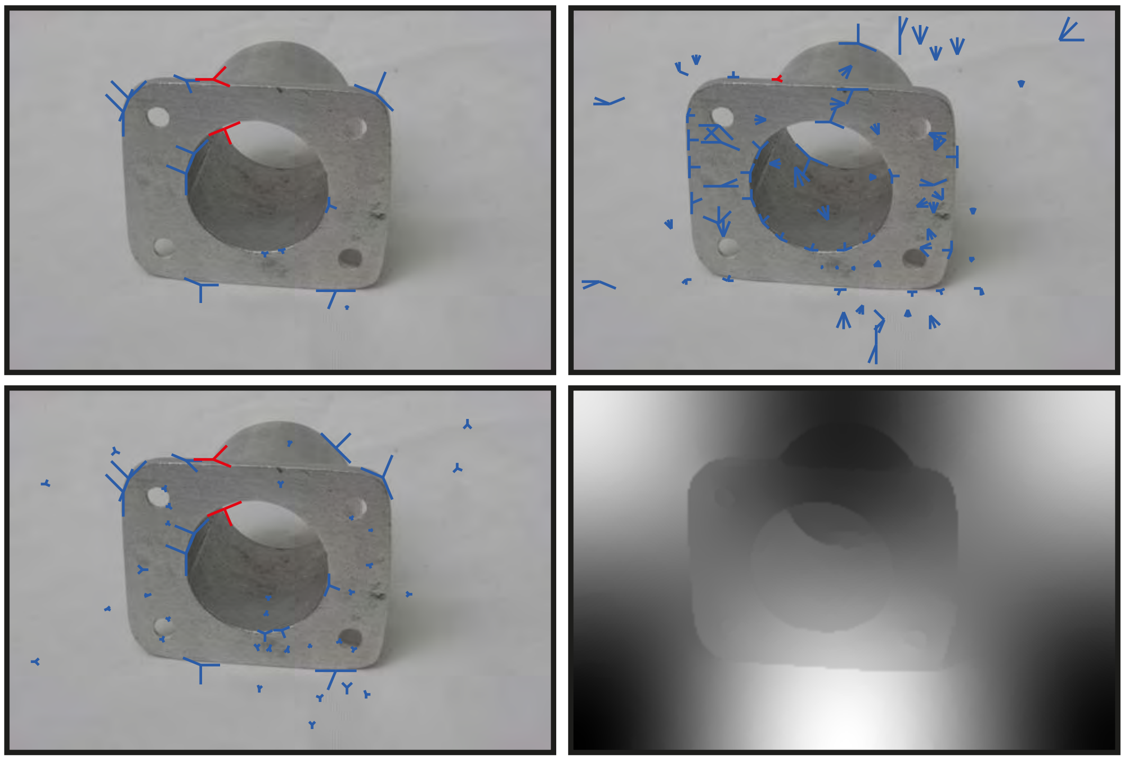
Figure 25: *Top left:* The output when using the difference term on global channels. *Top right:* The output when using the homogeneity term on global channels. *Bottom left:* The output when combining the difference and homogeneity terms on global channels. *Bottom right:* One soft segmentation, i.e. global channel. Note that the soft segmentation splits up large regions such as the background.

with the global channels, they cannot be ignored.

5.3.4 Combining Channels

The terms and channels all appear to be useful and not entirely redundant. The exception is the homogeneity term on the global channels, so we will ignore that term/channel combination. Omitting pixels at the center of the neighborhood is also useful, as it seems to improve the performance of the edge-grouping term while not affecting the performance of any other terms. Using a p-norm with $p \leq 1$ also seems to aid in junction detection. We perform a handful of experiments with parameters tuned by hand and demonstrate them in Figure 26. We note that different parameter settings result in detecting different sets of junctions, which leads us to suspect that with better parameters, our algorithm could achieve better results.

However, even without obtaining the optimal parameters, we have shown that the terms and channels we use complement each other in detecting junctions, and that even without optimal parameters, we can achieve promising results. A broader selection from our results is in Figure 27. We show that although we detect too many false positives, junction localization is promising and the detection of angles is quite accurate at true junctions.

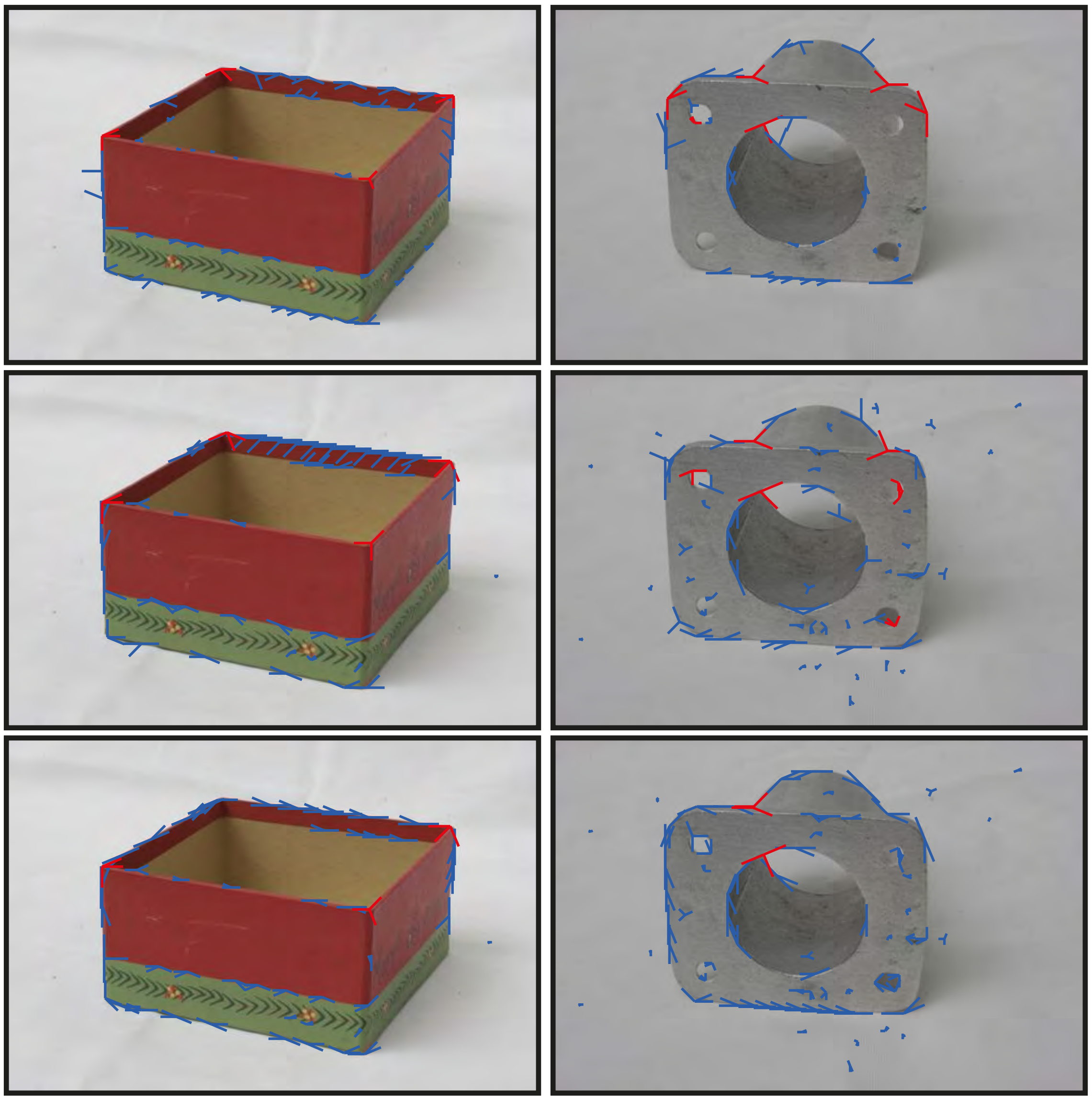

Figure 26: Each row is the output of the algorithm with different set of parameters. Note that different parameter settings result in capturing different junctions.

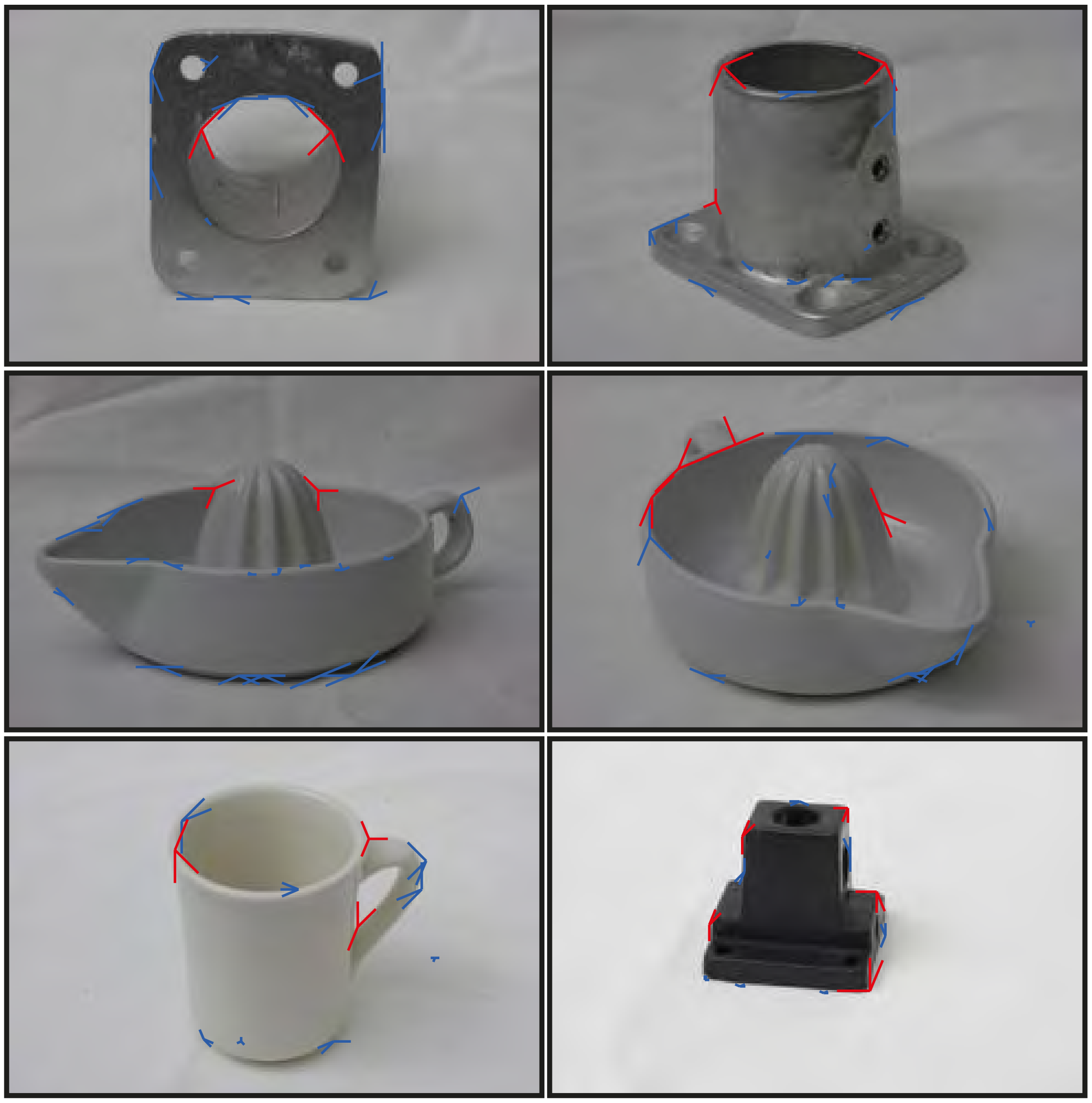

Figure 27: The output of our junction detection algorithm on selected examples from our dataset. All of these tests were done with the same parameters.

Chapter 6

Conclusion

Our main contribution is to identify the underlying model of a contour that the *gPb* boundary detector uses, replace that model of a contour with the model of a junction, and create a new algorithm for detecting junctions. We have outlined our algorithm and have shown promising preliminary results. Our algorithm localizes junctions mostly correctly, and detects angles at true junctions accurately. However, we note that it generates many false positives. Also, junctions that are detected with one combination of parameters will sometimes not be detected with different parameters.

We meet the two central problems with suggestions for future work. First, false positives could be reduced by post-processing. As we noted in Section 2.4.3, junction detection algorithms are often accompanied by post-processing that reasons about the connectivity of the junctions and discards those that are not consistent with the reasoning about the three-dimensional geometry of the object in the image. Our algorithm would benefit from such post-processing as a way to threshold the responses to our detector more intelligently.

Second, missed junctions may be a result of a non-optimal combination of parameters. There are too many parameters in our algorithm for hand-optimization to find a combination of parameters that will detect the most junctions. Furthermore, we have also shown that the terms that we use to define a junction, difference between wedges, homogeneity within wedges, and coincidence with edges, were shown to each be useful and provide non-redundant information about the image. This suggests that changing parameters will have a significant effect. Thus, obtaining a labeled dataset and using machine-learning to tune the parameters of the algorithm would most likely result in improved performance.

Another possibility for improving the number of missed junctions is to adjust gPb. As we have alluded to earlier, the gPb boundary detector was designed and optimized for scene-level images. We are interested in object-level images, and retuning the gPb algorithm to our dataset or any object-level dataset might result in better results, as detected contours are an important building block of our algorithm. There is reason to believe that scene-level boundary detection and object-level contour detection are slightly different tasks. Boundary detectors only need to distinguish between different kinds of objects, but contour detectors need to also distinguish between different parts of the same object. These developments would likely significantly improve the output of our junction detector.

We were motivated to develop a junction detector because, as we showed, although junctions are critically important to the contour image, the gPb boundary detector does not perform well near junctions. As we show in Figure 28, our junction detector

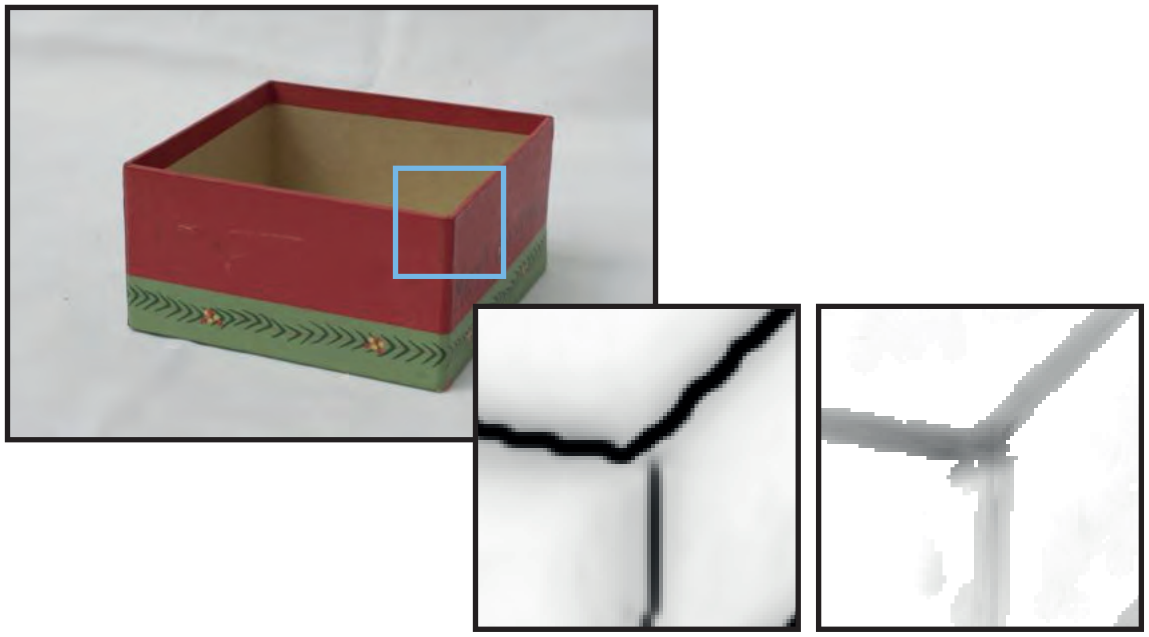

Figure 28: *Left:* An image of a box. The area we will focus on is outlined in blue. *Center:* The output of the *gPb* boundary detector on this part of the image. *Right:* The output of the *gPj* junction detector on this part of the image. Note that the junction detector fills in the gaps left by the boundary detector.

does indeed fill in some of the gaps left by *gPb*.

Extracting contour images from photographic images is an exciting problem because of the many possible applications of a complete contour image. Our suggestion of combining the results of a contour detector and a junction detector appears to be one important step towards extracting complete contour images from photographic images.

Bibliography

[1] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour Detection and Hierarchical Image Segmentation. *TPAMI*, 2010.

[2] Fred Attneave. Some informational aspects of visual perception. *Psychological Review*, 61(3):183–93, May 1954.

[3] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *TPAMI*, 24(4):509–522, April 2002.

[4] David J. Beymer. Finding Junctions Using the Image Gradient. In *CVPR*, 1991.

[5] John Canny. A computational approach to edge detection. *TPAMI*, 8(6):679–698, 1986.

[6] M. Cazorla, F. Escolano, D. Gallardo, and R. Rizo. Bayesian Models for Finding and Grouping Junctions. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, number Mdl, 1999.

[7] M. Cazorla, F. Escolano, D. Gallardo, and R. Rizo. Junction detection and grouping with probabilistic edge models and Bayesian A*. *Pattern Recognition*, 35(9):1869–1881, September 2002.

[8] Martin C. Cooper. Interpretation of line drawings of complex objects. *Image and Vision Computing*, 11(2):82–90, March 1993.

[9] Martin C. Cooper. Interpreting Line Drawings of Curved Objects with. *Image and Vision Computing*, 15:263–276, 1997.

[10] Vittorio Ferrari, Loic Fevrier, Frederic Jurie, and Cordelia Schmid. Groups of adjacent contour segments for object detection. *TPAMI*, 30(1):36–51, January 2008.

[11] Vittorio Ferrari, Tinne Tuytelaars, and Luc Van Gool. Object Detection by Contour Segment Networks. In *ECCV*, volume 3953, pages 14–28, 2006.

[12] Bernd Fischer, Thomas Zoller, and Joachim M. Buhmann. Path Based Pairwise Data Clustering with Application to Texture Segmentation. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2001.

[13] William T. Freeman. The generic viewpoint assumption in a framework for visual perception. *Nature*, 368(6471):542–5, April 1994.

[14] Meirav Galun, Ronen Basri, and Achi Brandt. Multiscale Edge Detection and Fiber Enhancement Using Differences of Oriented Means. In *ICCV*, 2007.

[15] Chris Harris and Mike Stephens. A Combined Corner and Edge Detector. In *Alvey vision conference*, volume 15, pages 147–151. Manchester, UK, 1988.

[16] Olga A. Karpenko and John F. Hughes. SmoothSketch : 3D free-form shapes from complex sketches. *SIGGRAPH*, 25(3), 2006.

[17] Yunfeng Li, Zygmunt Pizlo, and Robert M. Steinman. A computational model that recovers the 3D shape of an object from a single 2D retinal representation. *Vision Research*, 49(9):979–991, 2009.

[18] J.J. Lim, Pablo Arbeláez, and Jitendra Malik. Recognition using regions. In *CVPR*, pages 1030–1037. Ieee, June 2009.

[19] Chengen Lu, Longin Jan Latecki, Nagesh Adluru, Xingwei Yang, and Haibin Ling. Shape Guided Contour Grouping with Particle Filters. In *ICCV*, number Sept. 29 2009-Oct. 2 2009, pages 2288–2295, 2009.

[20] Michael Maire, Pablo Arbeláez, Charless Fowlkes, and Jitendra Malik. Using Contours to Detect and Localize Junctions in Natural Images. In *CVPR*, 2008.

[21] Jitendra Malik. Interpreting Line Drawings of Curved Objects. *IJCV*, 1(1):73–103, 1987.

[22] Jitendra Malik and Dror Maydan. Recovering three-dimensional shape from a single image of curved objects. *TPAMI*, pages 555–566, 1989.

[23] David Martin, Charless Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *TPAMI*, 26(5):530–49, May 2004.

[24] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, volume 2, pages 416–423. IEEE Comput. Soc, 2001.

[25] J. Matas and J. Kittler. Contextual Junction Finder. *Vision as Process*, pages 133–141, 1995.

[26] Josh McDermott. Psychophysics with junctions in real images. *Perception*, 33(9):1101–1127, 2004.

[27] Laxmi Parida, Davi Geiger, and Robert Hummel. Junctions: detection, classification, and reconstruction. *TPAMI*, 20(7):687–698, July 1998.

[28] Terry Regier. Line Labeling and Junction Labeling : A Coupled System for Image Interpretation. In *International Joint Conference on Artificial Intelligence*, 1991.

[29] Terry Regier. Line Labeling Using Markov Random Fields. *Image*, (November), 1991.

[30] Azriel Rosenfeld, Robert A. Hummel, and Steven W. Zucker. Scene Labeling by Relaxation Operations. *Transactions on Systems, Man and Cybernetics*, 6(6):420–433, 1976.

[31] Alexander Toshev, Ben Taskar, and Kostas Daniilidis. Object Detection via Boundary Structure Segmentation. In *CVPR*, pages 950–957, 2010.

[32] Deborah Anne Trytten and Mihran Tuceryan. The construction of labeled line drawings from intensity images. *Pattern Recognition*, 28(2):171–198, February 1995.

[33] Bo Wang, Xiang Bai, Xinggang Wang, Wenyu Liu, and Zhuowen Tu. Object Recognition Using Junctions. In *ECCV*, 2010.

[34] Qihui Zhu, Gang Song, and Jianbo Shi. Untangling Cycles for Contour Grouping. In *ICCV*. Ieee, October 2007.