DS/NC/ESD 863 Machine Perception

Mini Project Final Report

Rishabh Manoj (IMT2013035) Simran Dokania (IMT2013044)
Udbhav Vats (IMT2013055) Sriveda Reddy (IMT2013047)

April 19, 2017

Contents

List of Figures

List of Tables

1. Problem Statement

The goal of this Mini project was to develop a robust Image Processing algorithm to count saris/T-shirts placed in shelves of a retail shop. The user can specify a bounding box.

2. Motivation

Automating the process of counting clothes on shelves can tremendously help us in reducing the manual effort put in. It saves time and is therefore efficient. It can also minimize the human error. The need to count saris may arise in inventory management, information on available stocks, etc.

3. Literature Review

Contour Detection is a method used in image processing to determine shape of objects that are of the same colour or intensity.[1] The process basically finds a set of boundary points that together identify the objects. An image may have many objects and these objects maybe recognised by their colour. Contour Detection basically finds all the objects as a curve that surrounds the object.

In order to improve the performance of contour detection, binarising the image is a good pre-processing step. M. R. Maire in his dissertation paper [2], came up with an algorithm to find contours. In this approach, he uses local brightness, texture cues and colour to cluster the image into different objects.

In OpenCV, there exists a $findContours$ function which essentially returns a numpy array of all the contours, in the form of (x,y) coordinates of the boundary points of each contour. The parameters you need to feed into the $findContours$ function, is first and foremost the image itself. The second parameter is the contour retrieval mode like the one described in the paper above, the most common of which is the RETR_TREE. The third parameter is the contour approximation method. This third parameter is added in order to save memory wherever possible. In order to represent a straight line, you do not require every point on the line. Infact with just two point, the start and end points of the line, we can describe the line completely. Similarly, for a rectangle, knowing the four corner points will suffice. The CHAIN_APPROX_SIMPLE argument returns the minimum set of boundary points, removes all redundancy and compresses the contour.[1]

The Sobel Edge Detector basically does a 2D spatial gradient measurement on an image and identifies high spatial frequency regions as edges. 2D spatial gradient essentially means to find locations where there are directional changes in the intensity or colour of pixels. If there are a lot of these colour gradients at a particular location, known as high spatial frequency regions, it is very likely an edge. The kernel used by the Sobel operator is less sensitive to noise. Because of the smoothing effect of the Sobel kernel, natural edges in images often lead to lines in the output image that are several pixels wide.

For the purpose of our current project, we found the above two methods to be relevant. In most of the images, we find that if we can blur out the designs on the saris which may constitute noise, each sari is predominantly of one particular colour. Contour detection is hence an intuitive approach to be able to detect the boundaries of all the saris in the image. Similarly, if the entire sari can be detected as an edge by the Sobel Edge Detector due to the colour and intensity change from the shelves, we can basically go through the image and count the number of edges on the shelf to determine the number of saris on the shelf. This was the basic intuitive behind using these two methods in the existing literature to solve the problem of counting saris.

4. Methods

There are four collaborators on this projects, so we decided to use GitHub - which is a platform that allows multiple people to work on and build software. We found this process to be quite useful as we could access the codes that our team members had worked on and avoid starting from scratch. The GitHub link for our project is here

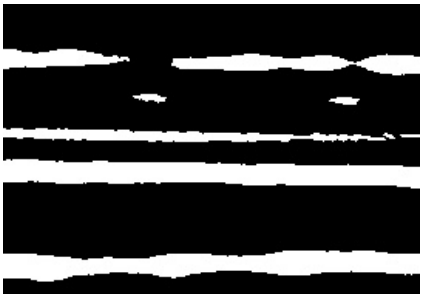
4.1. Using Horizontal Sobel Edge Detector and Contours

We first tried using Horizontal Sobel Edge Detector on Saree images which were not taken at an angle. Before applying this on the image we first blurred out the images by first using Gaussian Filter of window size 21 and then Bilateral Filter of window size 19. Gaussian blurring is done with the function, cv2.GaussianBlur(). It is highly effective in removing Gaussian noise from the image. Bilateral filtering is done with the function, cv2.bilateralFilter(). It is highly effective in noise removal while keeping edges sharp. Since we already blur out the images using Gaussian blurring, Bilateral filtering does not detect the small textures on the surface like sari designs and patterns, but main edges like sari borders are still preserved. Therefore, the reason behind doing this was to blur out the patterns on saris so the edge detection algorithm

is unable to detect them and the error in the result is less.

The code gave incredible results for saris kept in a single column (or shelf). However, if we change two parameters used in the code to a different value (window size of Gaussian blurring to 29 and window size of Bilateral filtering to 21), our code gives nearly an exact count of saris (with $\pm$ 2 error) for multiple shelves in straightened images.

(a) Single Column Image 1



(b) Output of Detector on Image 1

Figure 1: Using Horizontal Sobel Edge Detection a)

(a) Single Column Image 2

(b) Output of Detector on Image 2

Figure 2: Using Horizontal Sobel Edge Detection b)

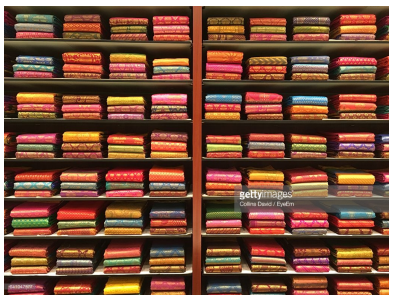
(a) Single Column Image 3

(b) Output of Detector on Image 3

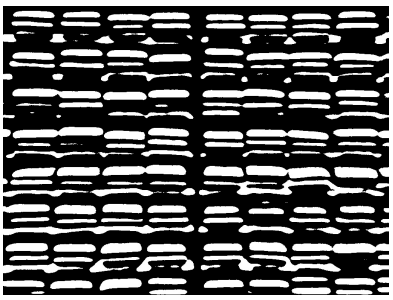
Figure 3: Using Horizontal Sobel Edge Detection c)

Here the number of white lines represent the number of saris in the image. In order to count the number of saris, we start at the highest point at approximately the center of the image and keep moving vertically downwards. As we move down, we increment the count of the saris when we encounter a white pixel. This point represents the start of the sari. Then we hold that count till we encounter a black pixel. We do the same till be reach the bottom of the image. The count we are left with represents number of white lines in the image which translates to the number of saris in the image.

In the case of multiple shelves, we assume that each column of shelf have same saree width. We start counting at the highest point of the midpoint of first column and successively increment the width to count the next shelf column. This is done till all the columns have been counted.



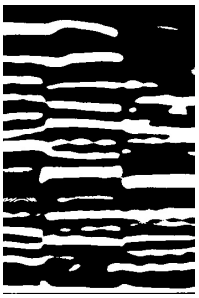
(a) Image 1



(b) Output of Detector on Image 1

Figure 4: Using Horizontal Sobel Edge Detection d)

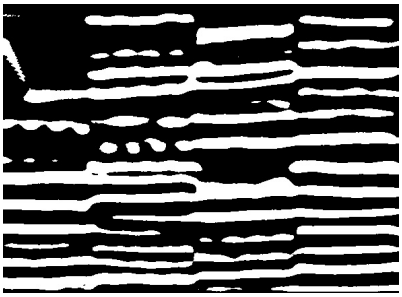
(a) Image 2



(b) Output of Detector on Image 2

Figure 5: Using Horizontal Sobel Edge Detection e)

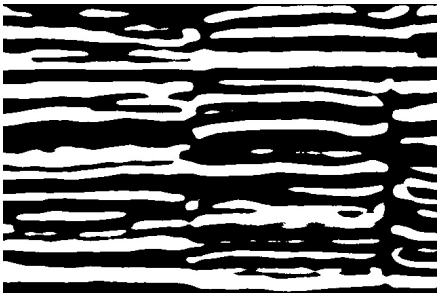
(a) Image 3



(b) Output of Detector on Image 3

Figure 6: Using Horizontal Sobel Edge Detection f)

(a) Image 4



(b) Output of Detector on Image 4

Figure 7: Using Horizontal Sobel Edge Detection g)

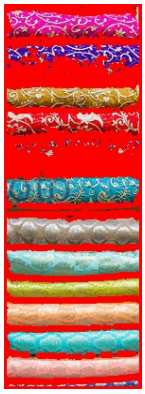
Following is a table which summarizes the results we obtained:

We used contours in order to assure whether the Sobel edge detector is detecting the

Image Name	Our Count	Actual Count	Error
Single Column Image 1	8	8	0
Single Column Image 2	7	7	0
Single Column Image 3	5	5	0
Image 1	183	181	2
Image 2	39	40	1
Image 3	57	56	1
Image 4	64	66	2

Table 1: Table of Saree count using our algorithm and manual counting

saris individually. We detected contours on the original image and drew them on the Sobel edge detector output. We get the following result:



(a) Contour applied on Image

(b) Contours drawn on Sobel Edge Output

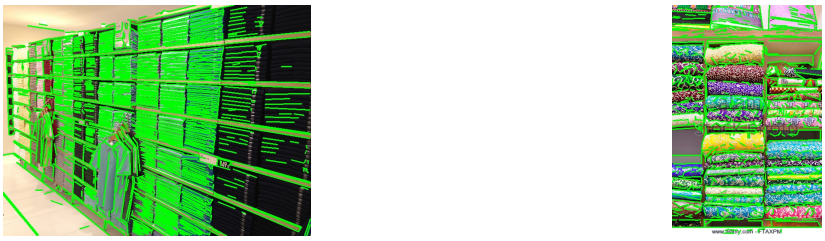
Figure 8: Contours Overlapped with Sobel Edge Detection

From the figures it is evident that the each contour that is indicative of one sari has one black and one white strip. So counting the number of white strips would give us the number of saris.

4.2. Shelf Detection

In every image, Saris are stored in shelves. One of our idea was to detect the shelf edges and automatically create bounding boxes for saris in each shelves.

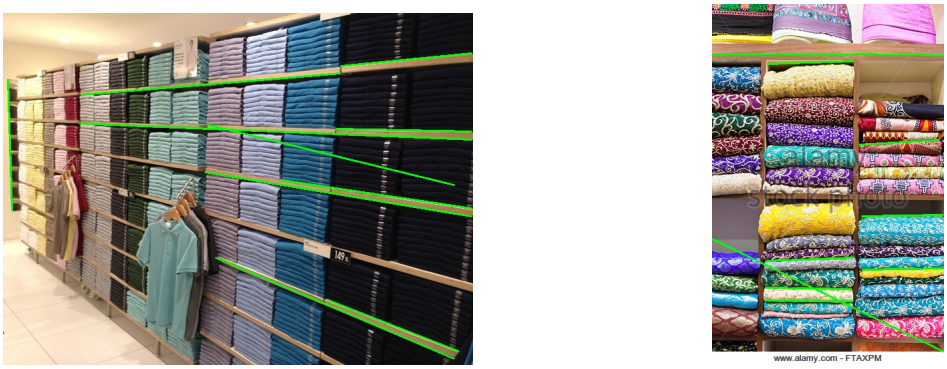
To detect shelf edges we tried out with different line detection methods. We used OpenCV's LSD to detect line segments.



(a) Image 1 (b) Image 2

Figure 9: LSD on Saree Images

As you can see from the images, a lot of line segments were detected which are unnecessary. We only require parallel and perpendicular lines of longer lengths. To filter, we try merging line segments based on orientation and length constraints (i.e) two lines will be merged if the linear and angular distance between them is under some constraint. After merging, we filter the lines based on the length and orientation (i.e) only lines of longer length which are parallel or perpendicular to the horizontal plane are used.

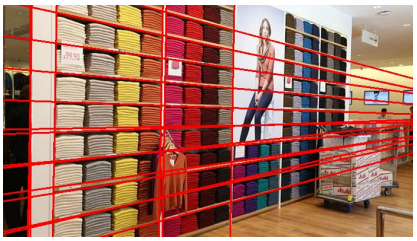


(a) Image 1 (b) Image 2

Figure 10: Filtered Line Segments

From the images we see that while the line segments capture some shelf edges it does not help us segment out each shelf properly. Due to the failure of LSD, we next try out Hough Transform. There are two variations of Hough Transform, Generalised

and Probabilistic. Generalised Hough Transform differs from Probabilistic in the way of outputs. Generalised Hough Transform gives out lines whereas Probabilistic gives out line segments.



(a) Generalised Hough Transform

(b) Probabilistic Hough Transform

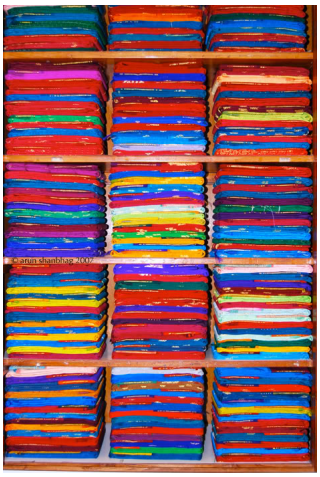
Figure 11: Variations of Hough Transform

After applying both the Hough algorithms on a set of images, we concluded that Generalised is better for the image set than Probabilistic. As you can see from the image above, the output of generalised work better for the image than probabilistic but none of them are ideal for the image. The generalised Hough transform overextends the detected shelf while probabilistic Hough transform finds only certain edges.

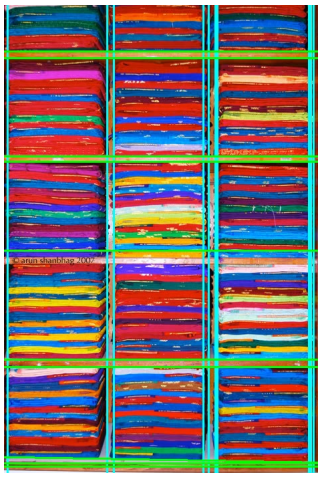
From the horizontal and vertical lines (shelf edges) we find the intersection points. Using the adjacent intersection points we draw a rectangle. These rectangles will hold a single stack of saris that lie between the adjacent shelf edges. This bounding box represents a stack of saris which can be segmented out. An area constraint is added to ensure only stack of saris are segmented out not other minuscule boxes.

Each Bounding box is segmented out and the previous method *Sobel Edge Detector and Contours* is run on them to calculate the count of saris.

In *Sobel Edge Detector and Contours* the method to count sarees for multiple shelves is not automated. We have fixed a value for the width of saree and used it as a parameter for counting. Using the *Shelf Detection* method, we automatically build bounding boxes for the multiple shelves, then give each shelf as an input image to the *single shelf Sobel Edge Detector and Contours* method for counting. Each bounding box acts as the input to the *single shelf Sobel Edge Detector and Contours* method where we automatically count the number of sarees in the shelf.



(a) Original Image



(b) Shelf Edge Detection

(c) Bounding Box

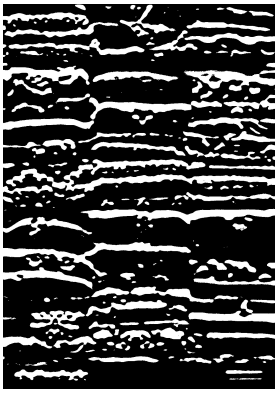
Figure 12: Segmenting Stacks of Sarees

Our main objective is to segment out stacks of saris that can then be counted using the *Sobel Edge Detector and Contours* method. So the purpose of shelf detection is to segment out *each stack* of saris one by one.

Our main challenge here is that of identifying vertical lines between each stack of saris. In most of the images there is no vertical edge between each stack, this makes it very hard for the algorithm to detect this.

5. Future Work

(a) Original Image



(b) Sobel Filter

Figure 13: Failed Case

Our code requires the images to be in parallel with the camera alignment. This makes it easier for our code to perform sobel filter and Hough transform. This is one challenge that has to be overcome in the future.

In the case mentioned above, the design patterns in the saree is too complex. We apply some Gaussian and Bilateral filtering to remove design patterns in sarees but these filters can remove only simple patterns. Here our Sobel filter detects the design pattern of the sarees and puts a wrench in our counting algorithm.

Our counting algorithm works well with thick sarees. The thinner sarees usually clumps together and the Sobel filter mistakes it for a thicker one and the count is incorrect.

Detecting shelf end still remains a challenging task due to the absence of shelf boundary in some of the images.

These are some areas where one could focus on in the future.

References

[1] Opencv. http://docs.opencv.org/3.2.0/d4/d73/tutorial_py_contours_begin.html. Accessed: 2017-04-19.

[2] Michael Randolph Maire. *Contour Detection and Image Segmentation.* PhD thesis, Berkeley, CA, USA, 2009. AAI3410970.