JIGSAW PUZZLE SOLVER

FINAL REPORT

Pooja Vijayan
poojavijayan@mail.usf.edu
CS Graduate Student
University Of South Florida

Ravali Yerrapothu
ravaliy@mail.usf.edu
CS Graduate Student
University Of South Florida

1. Introduction

Jigsaw puzzle is an interesting problem of assembling all the given pieces of puzzle in order to obtain a whole image or big picture.The image that has to be formedmay or may not be available. Our system tries to solve the problem without using the image. Hence, we need to consider several cues that are used by humans to solve the problem. For example, few people tend to pick edges and corners first, few try to match shape, few try to pick based on color of the pieces. We intend to use all of these in our implementation. Our system first tries to use the shape information along the four edges of each puzzle piece to detect shape features-tabs, holes or simply lines. Secondly, we compare the corresponding endpoints of the edges to obtain the deviation in the arclengths and eliminate the incorrect matches. Finally, incase of having more than one match at this stage, we further check for similar color histograms at regular intervals along the matching edges of the two pieces.

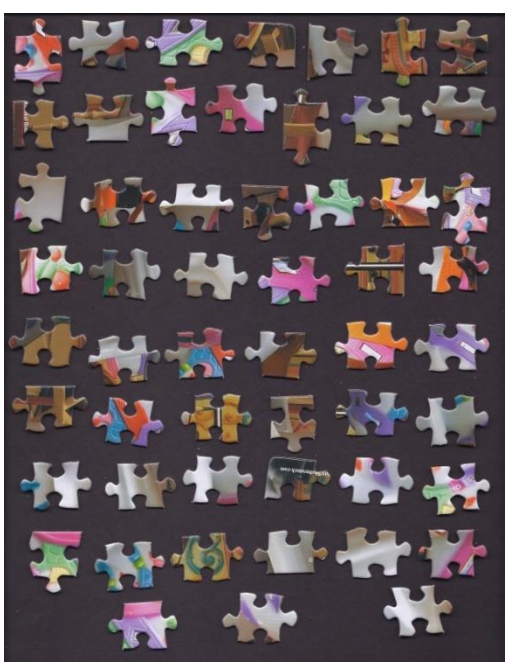
2. Implementation

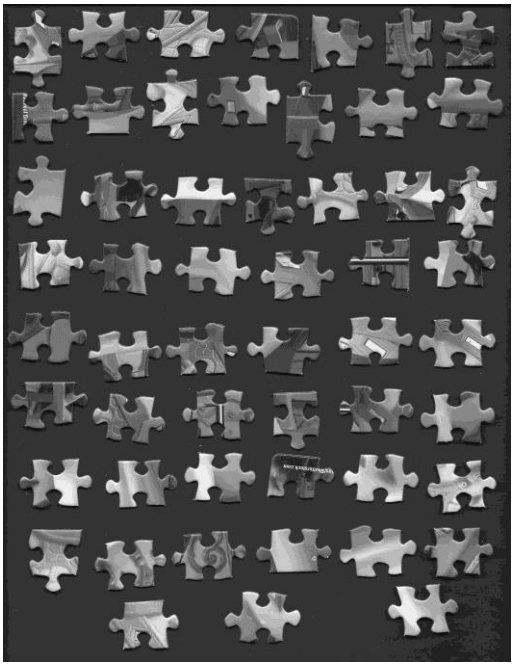
Following is the overview of the approach being adopted.

Our Algorithm involves following steps:

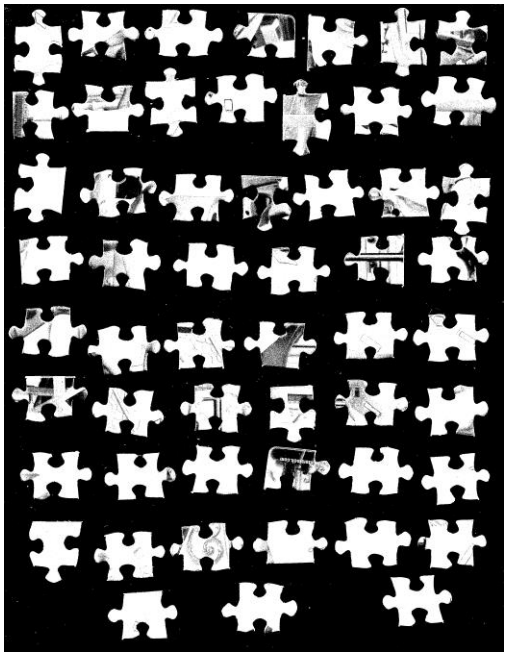
i) **Image Preprocessing for Noise Reduction:**The inputted image is converted to grayscale and normalized to have the intensities within the range 0 through 255. We calculate the derivatives along X and Y axes for the input image using OpenCV inbuilt function Sobel() as a pre-processing method to reduce the impact of shadows which might otherwise adversely affect the results of contour detection performed on the image. Using Sobel helped us reduce the shadow like regions from the input which was then processed for noise reduction. Median blur was applied to remove any noise that is still present and to smooth out the edge.

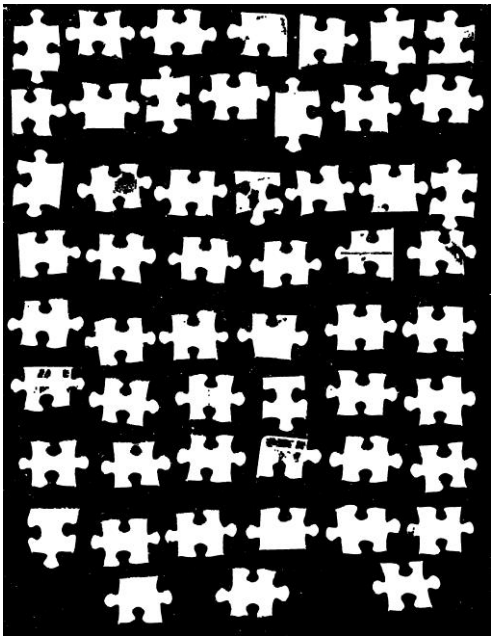
ii) **Thresholding and finding contours:** We then try to binarize the image by traversing through each pixel using a threshold value slightly higher than the backgroundIntensity - which is the intensity at pixel [10,10] of the input image.Inorder to obtain contours, we planned to do morphological erosion operation on the image and initially tried using MORPH_RECTANGLE. But since most of the borders of pieces were having curves, this did not give a good result for detecting smooth contours. Hence after experimentingwith different structuring elements we were able to zero in on MORPH_CROSS with size(2,2). After obtaining those contours, we filter the contours based on the approximated area of the pieces and candidate contours were then closed using CV_FILLED operation in OpenCV. This eliminated any other disturbances in the image and was used to segment and obtain solid object pieces.





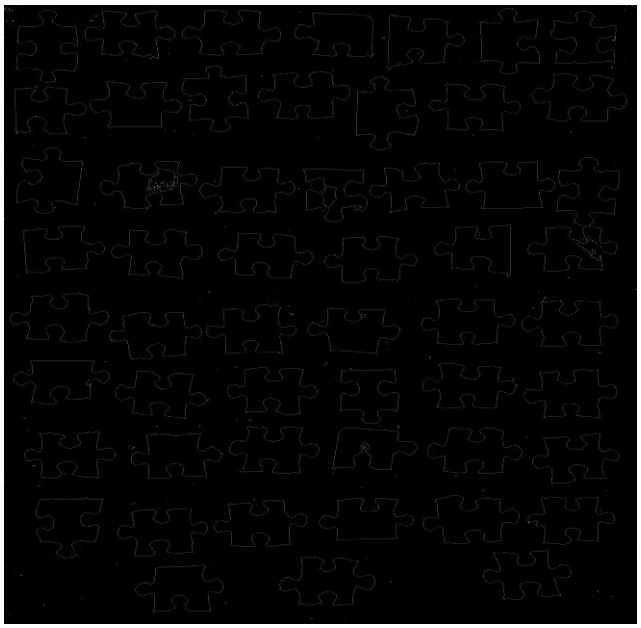
Input image Normalized GrayScale Image



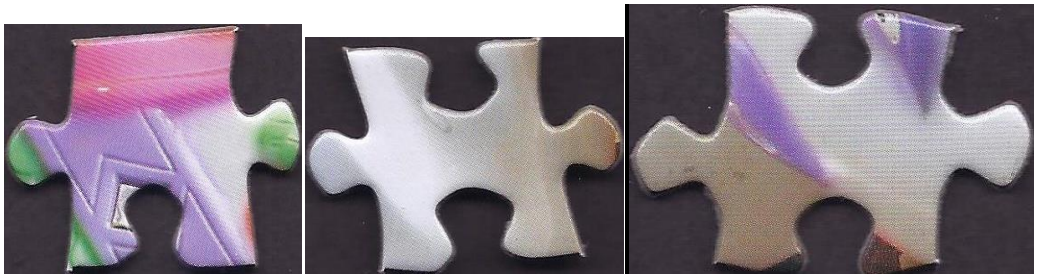


Threshold output (MeanIntenisty-10) Morphology Closing

iii) **Splittingobject pieces**: Once we have individual objects, we draw bounding boxes and crop it. Thresholds have been used on Contour Area to pick a contour only if its area is within a Range to further filter any noise. We were able to pick 52 images from the above input with this approach. We were unable to extract 2 pieces which are in contact in this input.
Assumptions have been made that every object has its own enclosing area and that they do not overlap or come in contact with other pieces.

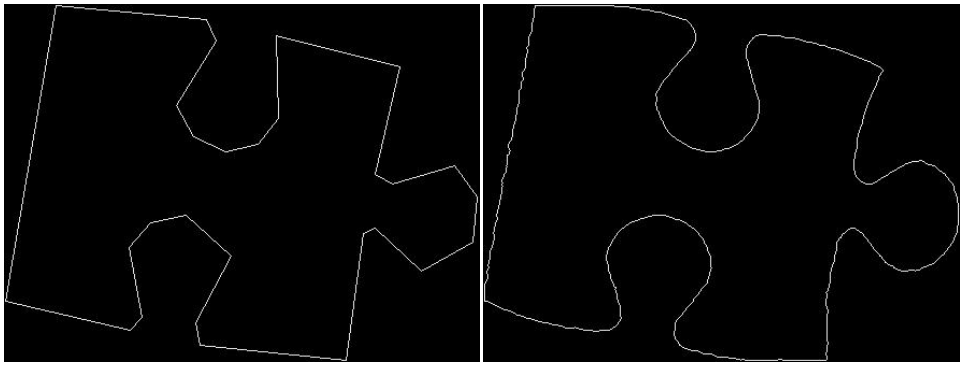


Contours of image



Pieces obtained from cropping using bounding boxes

iv) **Orientation and Corner Detection**: After obtaining individual pieces, before matching, we need to see that they are properly alignedi.e., all the pieces can only differ by multiples of right angles. In order to achieve this, we perform polygonal approximation on the contours to approximate the curves. With this, we obtain a contour representation consisting of many small line segments.



Polygonal Approximation Image Contour

We then run Hough Transform to determine all possible lines. By adjusting Hough transform to fill gaps of upto 20 pixels which is the average size of holes in this puzzle, we are able to detect all 4 edges of all the pieces including the corner pieces. Next, we choose the longest line representing the edge of the individual piece and rotate the image so as to align this longest line along X- axis.

For corner detection for each of the individual piece, we made use of goodFeaturesToTrackwhich is iteratively called until exactly four corners are detected. Based on the piece size , we estimate the closest two corners and the distance between them is passed as the mindistance to the function. We opted for the Harris corner detector specifically.

v) **Feature Extraction and Shape Representation:**In order to find a matching pair of images, we need to understand the shape of each object and its edges on each side. To find this, we need to detect corners and separate the contours to their constituent sides.Once we have aligned pieces, we draw bounding box around each piece and then shrink each box by certain number of pixels such that the bounding box cuts through each hole and tab on all four sides. We assume that the hole or tab is present at the centre of the edge at which it is present, for each piece.Hence we calculate midpoints on each edge using the detected corners or endpoints of that corresponding edge.We obtain the intensities above and below or either side based on the edge orientation and identify it as a hole, tab or a line.

We save these shapes in the form of a string –Line(L), Hole(H),Tab(T). Hence, for each image, a string is stored. Example: LLHT.

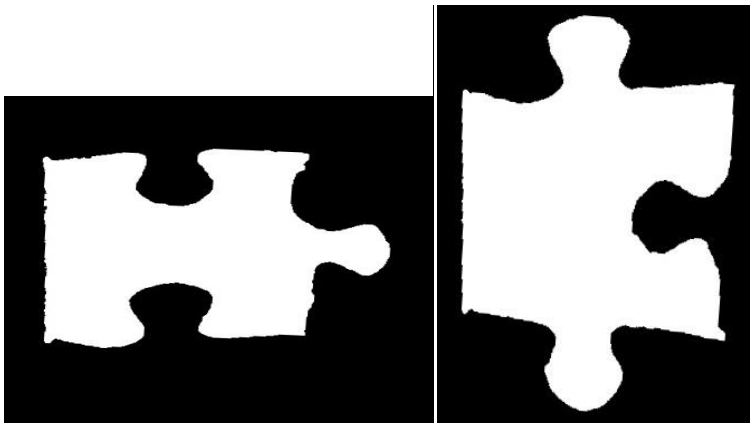
vi) **Piece classification:** Each piece is classified into one of the three groups. Corner pieces, Edge pieces and Inner pieces

Corner piece :	If a piece has 2 lines it is put into Corner group.
Edge piece :	If a piece has exactly 1 line, it is put into Edge group.
Inner piece :	All other remaining cases which have no lines.

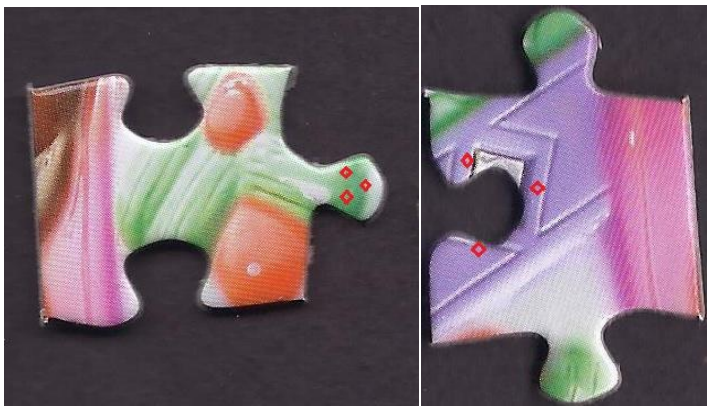
vii) **Matching**: We start by picking one piece from the corner group randomly and then we choose to solve the puzzle moving from the corner piece towards one of its sides which will have a hole or a tab.The best approach would be to pick top left corner, but since we don't have the finalimage, it is not possible to identify left corner. Thus our approach aims to produce a single final image. We use multiple levels of filters in choosing the candidate pieces to reduce the time consumption.

1) Based on the piece for which you are trying to find a match, you would know what piece you are looking for - an edge or corner or inner piece.

2) Look for a complement image – if you have a hole look for a tab and vice versa.
3) We compare the corresponding endpoints of the edgesto obtain the deviation in the arclengths, and also by using the tab length on one piece and the hole length on the corresponding other puzzle piece which we are trying to match. This helps us to elimate the incorrect matches upto a certain extent and we can then move to the next step to do color matching to find the candidate pieces for solving the puzzle.

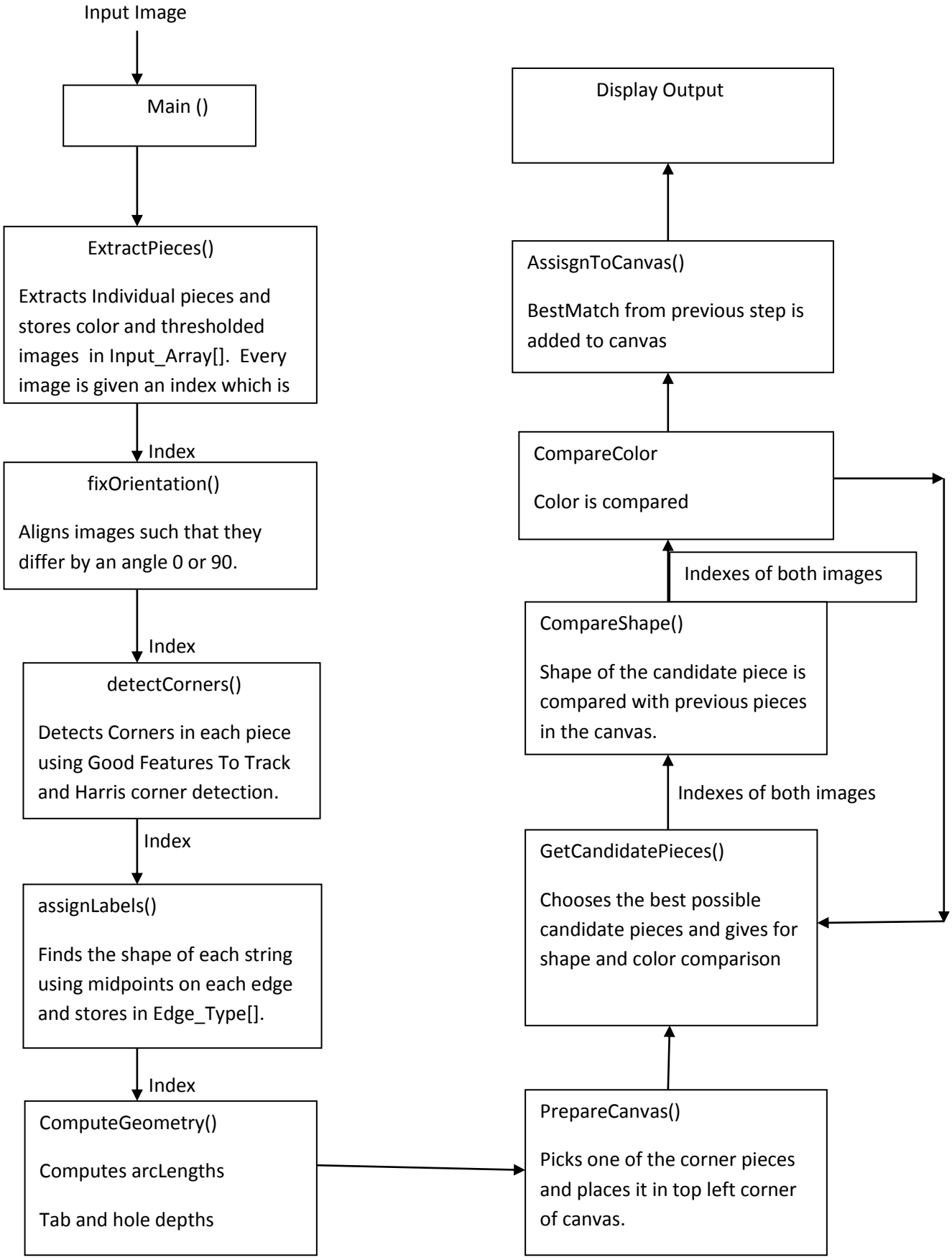


4) Try to match color samples at regular intervals near the joining edge of the two pieces to match at regular intervals. In order to match a tab and a hole, we considered the color histogram for the pixel window inside the tab region , and then tried to match it with the color samples from bins taken at three regions around the hole – one above the hole , one below the hole and one adjacent to the hole.If a match is found for any of these , then that piece is chosen to be the best match.



viii) **Assembling the pieces**: Once a match is obtained, translate the best match such that the left most corner of the piece is on the right top corner of the matching image.

FLOWCHART OF PROCESS:



CHALLENGES:

One of the main challenges we faced during the initial phase of the puzzle solver implementation was to choose an optimum value for thresholding to obtain a good noise-free binary image. The input images with white background for the puzzle pieces posed challenges for shadow removal and the edges were not sharp which made the selection for threshold value quite difficult. We spent a lot of time in optimizing our binary threshold output for the different sets of input images. We also tried our code with the input image with the black background, but found that the background had intensities which are lying in a range in between that of the image intensities of the puzzle pieces. Thus, we observed that a single threshold value would not suffice. And hence, we tried using adaptive thresholding and also used methods to choose different threshold values for different regions which are at a particular distance from the border of the input image set of puzzle pieces.

We also tried with input image samples from scanned puzzle pieces with the scanner lid left open, thus obtaining the pieces over a black background. We were able to obtain a nice input image with very less noise and no shadows. And we were able to obtain good contour results for these input images.

As we spent a sufficient amount of time obtaining the material from reading papers and gathering information and finalizing on our steps, and as we had to keep trying with different sets of input images and their thresholding challenge, we were not able to progress much in the final stage of the code implementation. If we would have fixed on one input image choosing either one of white or black background and finding an almost good value for thresholding we would have been able to implement more as we understood the complexity of the project as we were progressing with it and couldn't achieve all of what we had planned initially for the code implementation.

REFERENCES:

An Automatic Jigsaw Puzzle Solver, David **A.** Kosiba, Pierre M. Devaux, Sanjay Balasubramanian, Tar& L. Gandhi, and Rangachar Kasturi

A Global Approach to Automatic Solution of Jigsaw Puzzles, David Goldberg, Christopher Malon, and Marshall Bern

G.C. Burdea, H.J. Wolfson**, Solving jigsaw puzzles by a robot, IEEE** Trans. Robot. Automat. 5 (1989) 752–764

A.C.Gallagher, "**Jigsaw puzzles with pieces of unknown orientation**," cvpr, pp.382--389, 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012

OpenCV Online Documentation - http://docs.opencv.org/java/3.1.0/

General and coding queries - http://stackoverflow.com/questions/