Project Title - Puzzle Solver

EE 368 – Digital Image Processing

Sravan Bhagavatula
Stanford ECE
sravan.bh@gmail.com

Abstract— **Most jigsaw puzzles that are available today consist of several 100 pieces of a single image, and can take several hours to solve. This project proposes to solve the puzzle simply by using a picture of all of the pieces, and an image of the final solution.**

Keywords-component; jigsaw puzzle; Scale Invariant Feature Transform;SIFT

I. INTRODUCTION

This Most jigsaw puzzles commonly available on the market today consist of several pieces which can take many hours to solve. The aim of this project is to use a picture consisting of the pieces of the puzzle, arranged in any order, in addition to a picture of the solved puzzle.

The following section details a high-level implementation of the algorithm.

The main tool that will be used in the project is known as Scale Invariant Feature Transform (SIFT) [1]. The reasons for this will be included in section III, which details the workings of this technique, and also include reasons as to why it was chosen for this particular application. The algorithm used will also be described here in more detail.

The fourth section, Results, shows the results obtained from this algorithm, with a few examples.

The final section of the report details the future steps that can be taken to expand and improve the current implementation.

II. HIGH LEVEL IMPLEMENTATION

The project aims to find the placement of each individual piece with respect to the completed image. In order to achieve this, the pieces must have their own label so their placement in the solution image is known. For this reason, we first need two major inputs – the image consisting of all the pieces, and an image of the final solution of the puzzle. Figure 1 shows a typical example of the images needed.

The pieces image is crucial since it will be used to compare with similar features found in the original image and then label them accordingly. Thus, one of the outputs is to include a modified version of the pieces image, such that each piece is labeled with numbers. Similarly, the second output is a modified version of the solution image, which consists of the same numbers shown in the first output, but with each labeled piece shown in the right order of the solution, so that the user may use this diagram to know the right placement of each respective piece.

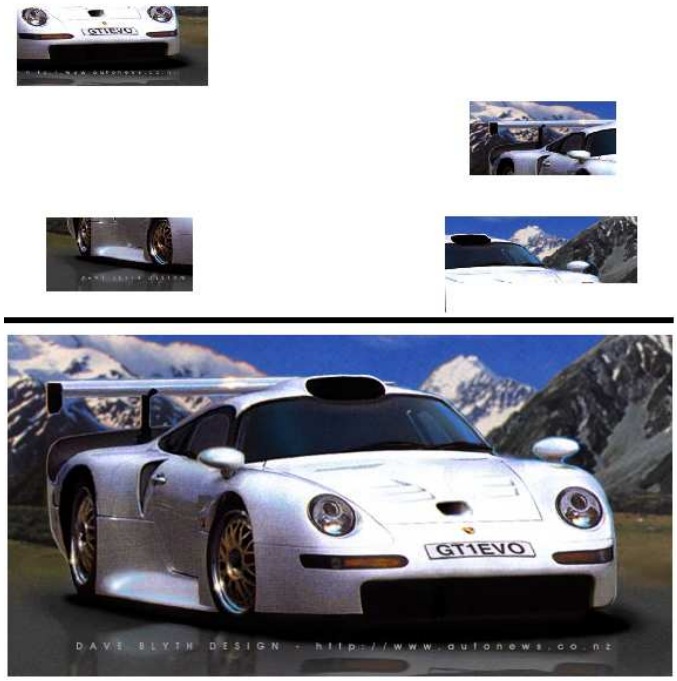

Figure 1. An example of the two input images needed.

III. SIFT AND ALGORITHM OVERVIEW

The first subsection details the main technique used in this project, namely, the Scale Invariant Feature Transform (SIFT). The reasons why SIFT was chosen will also be discussed below. The second subsection will explain the overall algorithm used in the project in greater detail.

A. Scale Invariant Feature Transform

Scale-Invariant Feature Transform was an object recognition technique developed by David Lowe in 1999. The main problem with most of the object recognition techniques before this was that they did not perform well when the object projective transforms. SIFT transforms an image into a collection of local feature vectors, and these are invariant to the changes described above. The features obtained from this algorithm are very similar to those obtained from the responses of neurons in the inferior temporal cortex in primate vision.

These scale-invariant features are detected efficiently through a stage filtering approach. The first of these stages identifies key locations (keypoints) in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. Each point is then used to make a feature vector that describes the local image region sampled to its scale-space coordinate frame. As mentioned earlier, this approach is based on a model of the behavior of complex cells in the cerebral cortex of mammalian vision.

The SIFT algorithm is computationally efficient, as there can be an order of 1000 features for some images, but computing these features takes no longer than a second. Figure 2 shows an example of the types of features detected in an example image. One can also notice that there are model outlines used in the diagram in order to recognize the different objects in the figure, as that was the primary objective of the SIFT technique. We can also see that the model works for cases where there are partial occlusions as well for the objects. It has been tested in [1] quite extensively to see if these features are retained when the image is rotated, or there have been slight changes to the lighting. They have observed that the method works very well in order to effectively retain most of the features after these changes occur.

Figure 2. Matching results using the model outlines and image keys for individual objects. Images from [1].

The reason for choosing SIFT, as the name implies, is that it is scale-invariant when detecting these features. This means that images where the scale of the object (zoom) changes, or rotation, or there are slight modifications to the lighting of the object do not affect the detection of these features. This basically implies that features detected in one image are unique to the entire object and do not change when encountering changes in the rotation, scale, or lighting of the object in question. This quality of SIFT is highly suitable for the application of finding matching puzzle pieces with respect to their keypoints, which can then be simply compared with ones that we find in other images. This comparison is done by using the descriptors for the keypoints, which are unique to each keypoint image.

B. Overview of Algorithm

Let's assume that of the two input images, the one consisting of the pieces is called P, while the solution image is S. This section consists of some function names and terms that are found and used in MATLAB. The algorithm consists of the following steps –

1. Find the keypoints in both P and S with vl_sift [2]
2. Output a modified P, with piece labels
    - Use kmeans() [3] to cluster the keypoints in each piece
3. Take a small number of points per cluster
    - Around 20 – 30.
4. Compare these keypoints with ones in S
    - 2-norm comparison of the SIFT keypoint descriptors.
5. Find locations in S of matches
    - These basically count as the location of each piece
6. Classify each region of matches into clusters
    - I.E., choose a "central point" to designate as the label of the region
7. Output a modified version of S using these cluster labels
    - One that has the same labels as the one in P, such that similar pieces are in the right locations

First, we find the keypoints in P and S using vl_sift() [2]. In step 2, we take our input P and cluster the keypoints in each piece such that we know where each keypoint lies with respect to its piece. This is done using kmeans(), which is a clustering function. However, an input needed here is the number of clusters, or pieces, that we will use to segment the image.

In step 3, we use a small number (points per cluster) that will be used for the comparison between the keypoints in P with respect to S. This number is usually around 25-30. More points will ensure a better comparison, but this amount is sufficient for our needs.

Step 4 conisists of comparing the keypoints' descriptors we put aside in step 3 with the ones we found from S. This comparison is done with respect to a threshold, so that we can find features that are most similar, and not necessarily the exact same. The descriptors are 128-dimension and are of type UINT8.

In step 5, we locate the matching keypoints record their occurrences in image S. Following this, in step 6, we classify

each region into their respective pieces by repeating steps 3-5 for each piece.

Finally, we output a modified version of image S which has each region (from step 6) labeled based on the pieces found in the modified version of image P. Please refer to the following section to see how the outputs appear.

IV. RESULTS

After applying the algorithm mentioned above to a few test images, here's what was obtained. Figures 3 and 4 show a couple of examples of the final results obtained.

As we can see, in figure 3, the two images obtained are modified versions of images P and S. The top image shows the modified P image, with each region labeled based on the piece, whereas the one of the bottom shows the modified S, with each pieces' label in the right location.

Figure 4 also has a similar result, as is clear in the images seen below.

In each case, the computation time was roughly 5~10 seconds, when running on a quad-core 2.66 GHz system. Of course, this can vary depending on certain variables, such as the image size, the number of pieces in the image P, the number of points of comparison per cluster, etc. Overall, the algorithm seems to work fairly quickly.

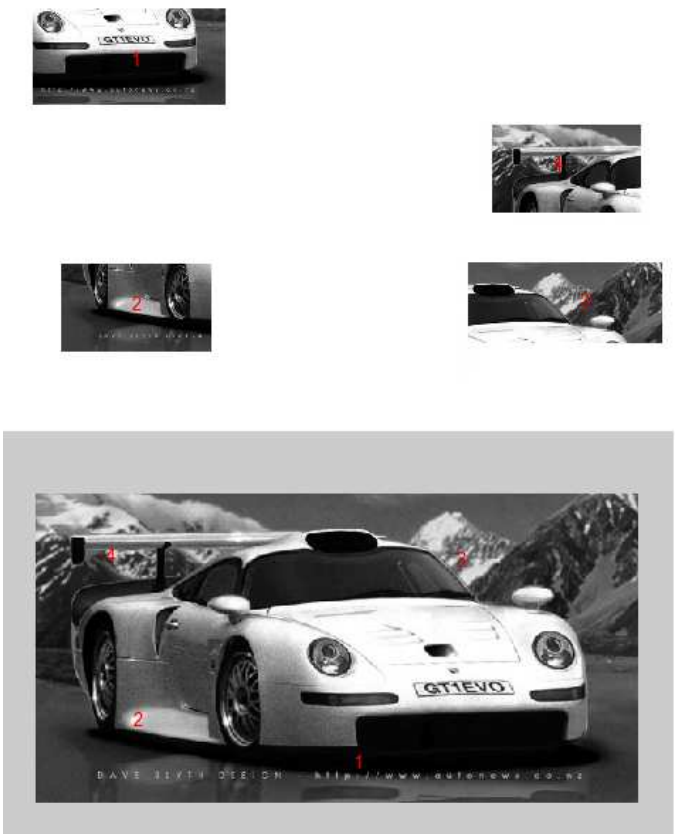

Figure 3. Test 1 of the algorithm.

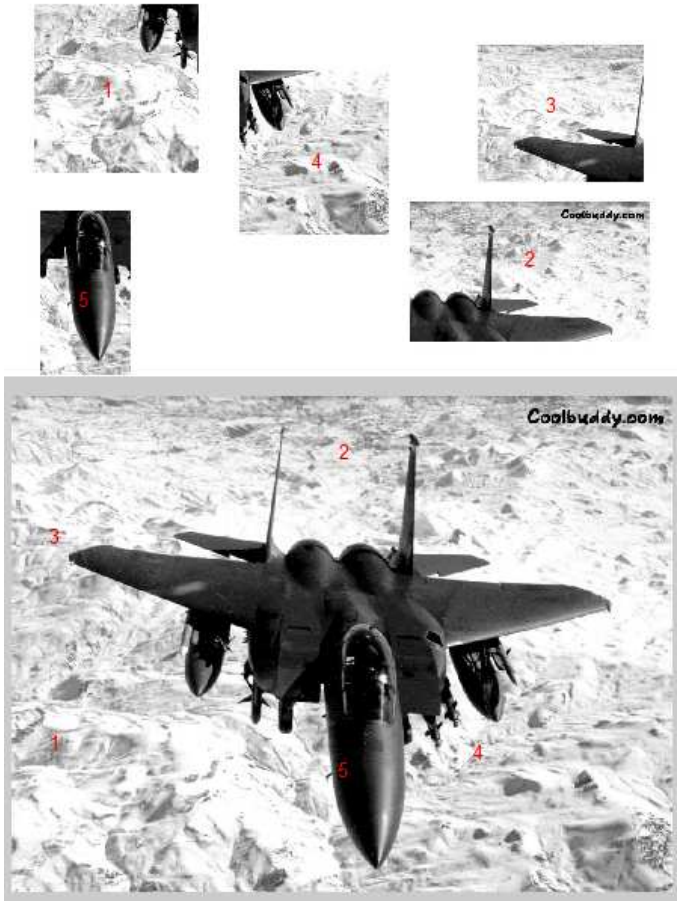

Figure 4. Test 2 of the algorithm.

V. FUTURE WORK

As we can see from the test images, the backgrounds of each image are fairly consistent and uniform. This is a current requirement, since the clustering that takes place in step 2 of the algorithm requires identifying the pieces and labeling them based on how scattered the keypoints are. Thus, we would ideally like to create a background that has no significant number of keypoints in it so that the keypoints in each cluster will only be those of the pieces. A possible solution would be observe all of the keypoints in the entire image and finding the ones that don't match with the solution image, and replacing these points with blank segments, such as the ones seen in the test images.

The project must also be tested on "P" images on which the pieces have different lighting, orientation, or scale changes. However, this will most likely not be an issue for the current algorithm, since SIFT is designed to deal with scale and rotation changes, as well as differences in lighting. However, lighting changes are not guaranteed to work effectively since it's not the primary change that SIFT is designed to work with.

This will need further testing and/or modifications to ensure the current algorithm works well.

Another set of tests that will need to be performed include testing how well the algorithm will work with very large scale images. Images that are not only are larger in size, but also have a much larger number of pieces. This will be computationally hard, as the code will definitely take a longer time to run. However, using the current algorithm would most likely handle the increased load without issue.

A minor change that can be made in the future would be to be able cluster the keypoints in P without the need of a "number of pieces" variable, which is currently a necessary input for the clustering process. However, kmeans() requires an input of the number of clusters in order to determine the number of clusters needed.

A final and fairly ambitious goal would be to solve the puzzle without the need for a solution image, i.e., only use P and not need S. This is far more difficult to implement and would require major modifications to the current algorithm, as not only will it be a matter of feature matching, but it will also include matching between pieces based on the features' continuity, as well as mixing and matching piece combinations to ensure that there aren't any "discontinuities".

ACKNOWLEDGMENT

S.B. would like to thank Dr. Bernd Girod, his teaching assistants David Chen and Derek Pang, who have all provided a great deal of resources to present the EE 368 – Digital Image Processing course. He would also like to thank Maryam Daneshi, who was assigned as his project mentor.

REFERENCES

[1] D. G. Lowe, "Object recognition from local scale invariant features," in *IEEE ICCV*, 1999, vol. 2, p. 1150.

[2] VL_SIFT "VL Feat Documentation" http://www.vlfeat.org/mdoc/VL_SIFT.html.

[3] K-Means Clustering http://www.mathworks.com/help/toolbox/stats/kmeans.html