A Jigsaw Puzzle Solving Guide on Mobile Devices

Liang Liang
Department of Applied Physics,
Stanford University
Stanford, CA 94305, USA

Zhongkai Liu
Department of Physics
Stanford University
Stanford, CA 94305, USA

***Abstract*—In this report we present our work on designing and implementing a mobile phone application that helps people solve jigsaw puzzles by locating the image of a single patch on the complete picture. Details of the algorithm and implementation are discussed and test results are presented.**

Keywords-component; Jigsaw Puzzle; Mobile Application; Template Matching; Image Segmentation; SURF; RANSAC

I. INTRODUCTION

Jigsaw puzzle is a game of assembling numerous small pieces (patches) to construct a complete smooth picture (template). In a typical jigsaw puzzle game, the players have a template to spot the possible locations of the patches. However, with human vision, the matching between patches and the template often turns out to be difficult and time-consuming. The complicated features of the template, irregular shape, unknown orientation and scaling of the pieces all contribute to the challenge of this game.

Given the well developed algorithms of detecting and analyzing images, the enormous capacity of processing speed and memory, computer vision may provide a valuable aid for human jigsaw puzzle players to detect the patches in the template. Thanks to the recent flurry on technology advance in mobile phones, some 'smart' models have been equipped with high resolution built-in cameras, powerful CPU and the wireless internet data transfer. These models can become a handy and smart platform to perform computer vision. In this project, we implement a jigsaw puzzle solver on a Motorola Droid phone to guide players to quickly solve the puzzle.

To overcome the challenges of solving jigsaw puzzles, pattern matching algorithms are required to be invariant to scales, rotations, and have a good tolerance with background clutter, different illumination conditions, as well as the shape distortion caused by taking pictures at variant angles. The other task the image processing algorithms need to carry on is to register the patch image properly with the template. The transformation from the patch to the corresponding part in the template needs to be properly constructed from the matching feature points. In this report, we will present in details how we carry on these tasks and combine them with mobile phone user terminal, the test result and the evaluation on the performance of the application.

II. RELATED WORK ON JIGSAW PUZZLE SOLVER

Automated jigsaw puzzle reconstruction has long been an intriguing problem in image processing community. If the complete puzzle picture is not known, people have to implement complicated algorithms using shape, color [1] and even texture [2] of each piece.

The difficulty of solving jigsaw puzzles lies not only in feature detection, but also in machine learning. The world record, 400 patches assembly, is set by a MIT-Israel team earlier this year [3].

With the guide of the complete picture (which is always given in real life jigsaw puzzles), the puzzle solving reduces to a template matching problem and the solver works in a straightforward way. However, we have not found such an application for mobile phones yet. All the jigsaws on mobile phones are electronic!

III. DESCRIPTION OF SYSTEM AND ALGORITHM

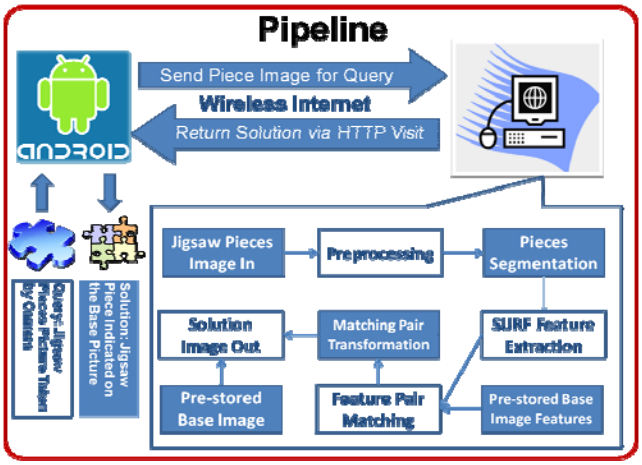

Figure 1. System Pipeline

A. *Mobile Phone Side:*

Hardware

Mobile Phone: Motorola Droid
CPU: 600MHz
RAM: 256MB
Operating System: Android v2.1
Captured Image Resolution: 1280x960
Camera Parameters:
Focus Mode: Macro

White balance: auto

Scene mode: night

Software

We have implemented an easy user interface on the mobile phone. The mobile phone serves as the data acquisition and display terminal for the application. When the user takes a snapshot of the patches, the mobile phone sends the picture to a HTTP server and leaves the heavy duty mathematical computation on the cloud. When the computation is done, the data is retrieved by the mobile phone through HTTP visit to the server.

Discussion

We do not implement the whole application on Android for the following reasons: 1. The performance of a smart phone is still much inferior to a personal computer, in terms of CPU speed, memory and storage size. The exhaustive computation required for this application might slow down the system response and be quite frustrating. 2. In the development and proof-of-concept stage, the accessibility and ease of modifying the algorithm is fairly important. The Matlab software provides us these advantages comparing to compiling codes again and again on Android. Additionally, new features (such as real time display and robotic control) can be further implemented to this application without sacrificing much performance. For the above reasons, only the user interface is coded on Android and the processing algorithm is executed in a Matlab script on a personal computer.

B. Server Side:

Hardware

Computer CPU: Intel i7 920 @2.67GHz

Computer RAM: 8GB

Software

Apache 2.2.15 and PHP 5.2.13 with customized script for http service and file upload

Matlab R2009a with open source toolboxs:

SURFmex V.2 for Windows developed by Petter Standmark;

Matlab RANSAC Toolbox by Marco Zuliani;

Customized functions and scripts to implement these algorithms.

Algorithm

After the patch image is transmitted to the server, Matlab reads the file and starts to implement the processing algorithm on the image.

The algorithm contains the following steps

- Image downsampling to reduce noise and speed up following calculations
- Patch segment extraction by edge detection and morphological operations
- Feature detection by SURF
- Feature descriptor matching
- Geometric consistency check by RANSAC
- Patch image transformation and image output on Droid

Remarks on SURF.

After extracting the segments from the photo, the segments are compared with the template to recognize matching locations. To make the comparison efficient and robust, distinctive image features need to be extracted. Features are usually composed of edges, corners and blobs. While Harris detector [4] provides a shift and rotation invariant method to detect corners, it is not invariant to scaling. However, in the jigsaw puzzle problem, the scales of the patches and the template are often not known beforehand. Depending on the distance to the camera, the patch sizes also vary. A scale invariant algorithm has to be explored. Scale-Invariant Feature Transform (SIFT) [5] is one of the well established shift-invariant feature detection and matching method. In this algorithm, the distinctive locations are points with maximal and minimal Difference of Gaussian in the scale-space to a series of smoothed and resampled images. The sub-pixel / sub-scale accuracy max was determined through 3-d quadratic function fitting. The local curvature is calculated to threshold the feature points. Dominant orientations are assigned and aligned to localized key points. For each local feature point, 4x4 orientation histograms with 8 directions are computed to generate a 128-dimentional feature vector. The feature vectors are then matched between the template and the patch. Inspired by SIFT, Speeded Up Robust Features (SURF) was developed in 2006 by the Bay group [6]. In SURF, the local feature points are identified by comparing the determinant of the Hessian matrix, with the Gaussian derivatives simply approximated by first order 2D Haar wavelet responses. The approximation allows the use of integral images to greatly reduce the processing time. For each feature point, horizontal and vertical pixel differences, dx, dy and |dx|, |dy| are accumulated over 4x4 subregions to give rise to a 64-dimention descriptor. SURF algorithm is several times faster than SIFT, and is claimed by its authors to be more robust to image transformation and noise, etc. In the preliminary test, SURF works about 5 times faster, and detects a fairly similar number / location of feature points. Therefore, for the jigsaw puzzle solver, we decided to implement the SURF method.

IV. EXPERIMENTAL RESULT

A. Image prepocessing

Image acquisition The template "Alice" used for testing the algorithm are shown in Fig2A. The template was carefully chosen to give a relatively large number of feature points and these feature points are relatively uniformly distributed. The photos of patches were taken through the Droid built-in camera. An example is shown in Fig2B.

Figure 2. A) The original template of "Alice", with a resolution 1024x768. B) The original photo of the patches in "Alice", with a resolution 2592x1936.

Image downsampling Both the template and the patch photos in the application were first downsampled after image acquisition.

The templates in the illustrated in this paper were either scanned from real images or took from the website for the convenience, although template directly took from Droid camera gave equally good performance for the jigsaw solving. A good template usually contains a large amount of image features. However, some of them are redundant, since in principle, three non co-linear paired feature points can already define the location of the patch. On the other hand, it takes a considerable amount of computer time to extract and match the feature points, which is one of the speed limiting step of the application. We empirically downsample the template image to its 1 / 2. This shortened the feature extracting time to 1 / 5, while reduce the feature numbers to about half of original, still prereserving enough features for later analysis.

The photos of patches were first downsampled to its 1/ 2 in Droid and then further downsampled to its 1/ 8 in Matlab run in the PC server. The first 1 / 2 saves the file transfer time from Droid to server PC through the wireless network. Another 4 times reduction in Matlab was performed to reduce the noise and speed up the algorithm processing.

Patch extraction Since our algorithm is designed to support multi patch alignment in a single photo shot, it is necessary to identify and separate the patches in the same photo so that the alignment of individual patch can be carried out in later steps.

The edge information was first used to detect the patches. This is calculated through Sobel method in Matlab (*edge*), with automatically chosen threshold. Then the edges are morphologically closed (*imclose*) with a diamond-shaped structuring element. Remaining holes are floodfilled (Matlab *imfill*). The above procedures were first implemented onto the illuminance (Y) component in the YCbCr space, since Y contains the good portion of the feature info. However, as shown in Fig3 B1, only the Y component is not enough to close the edges. To get a smooth mask for the patches, information from all Y (Fig3B1), Cb (Fig3B2), Cr (Fig3B3) component needs to be combined through 'Or' operation (Fig3C). We also noticed that combining the edge-imclose-imfill results from R, G, B components gave similar result image. A second structuring element (square) was applied to close the remaining contours (Fig3D) and resulted in intact, smooth masks for the patches after floodfilling (Fig3E). The masks are then segmented (Matlab *bwlabel*), with the small segment (smaller than 0.3 of the largest segment) considered as clutter and discarded (Fig3 F1, F2).

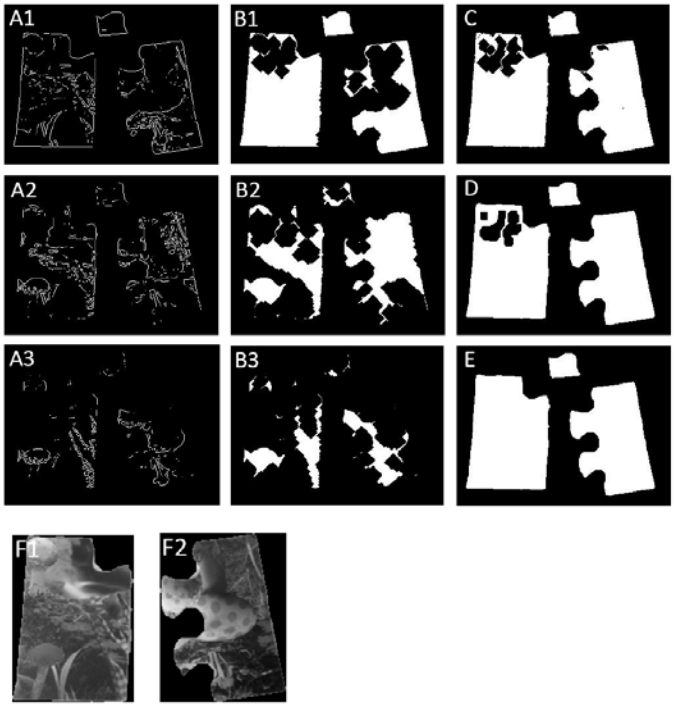

Figure 3. The patch extraction. A1, A2, A3) The Sobel edge of the Y, Cb, Cr components of Fig 1B. B1, B2, B3) The morphologically closed and floodfilled images of A1, A2, A3, respectively. C) The combined image of B1, B2 and B3. D) Morphogocially closed C with a second structuring element. E) Floodfilled image of C). F1, F2) The segmented patches in gray scale.

B. Feature detection

SURF algorithm was implemented to detect the feature points in both the template and the patches. Vectors of size 64 are calculated for the descriptors, since 64-descriptor was shown to be efficient and robust. The SURF computation module was adapted from the SURFmex V.2 for Windows develeoped by Petter Standmark. An example of extracted feature descriptors of the "Alice" template and patch are shown in Fig4.

Figure 4. Detected feature points on the template. 1070 feature points are detected in the template, labeled in blue '+'. 83

feature points are detected in the patch (segment in Fig3F1), labeled in red '+'.

C. Feature comparison

Feature descriptor matching The similarity between the feature descriptor vectors in the template (T) and patch (P) are estimated through the angle θ in between. Since descriptor vectors are all normalized to unit length, the cosθ equals to the dot product between the vectors. For each descriptor p in P, the θ between p and t were calculated against all t in T. Then a ratio test was carried out between the first and second smallest angle θ1 and θ2. Only when θ1 is smaller than 0.5x θ2, p is considered to find a matching descriptor (t1) in the template. 0.5 here is empirically determined. Since Matlab is very efficient at calculating the dot product, this matching evaluation method can run rather efficiently.

Figure 5. The matched feature points. 25 matched feature points are detected and connected by green lines.

D. Geometric Consistency Check

We implement the RANSAC method to check geometry consistency and determining the geometric transformation parameters between jigsaw patches and the template. Our code utilizes the Matlab RANSAC Toolbox by Marco Zuliani with self-defined functions and optimized parameters to search for parameters of an affine transformation. The reasons we choose affine transformation over perspective are: 1. when people take pictures for a flat piece, they hold the camera in parallel with the piece because the focusing of a tilted camera on a flat piece would not be uniform over the surface. Therefore the perspective distortion is small. 2. It takes less data points to define an affine transformation than a prospective one. In this way the algorithm has better chance to work and return correct result with few matching features on each piece and thus allow us to work with smaller jigsaw pieces. Other parameters we choose for RANSAC method are: ε=1e-3, q=0.3, k=5, tolerant noise=10 pixels. For this set of number, the expected trial number is 2839. For practical instances, q is around .8, the converging usually takes place in tens of steps.

An illustration of geometry consistency check is shown below in Fig6. In the left panel, RANSAC filters the matching features generated from previous steps by trying to calculate affine transformation parameters for them. And 13 out of 15 feature pairs are correctly identified as inliers. By applying the affine transformation generated from the optimal parameters on the jigsaw piece, the position of that piece is correctly restored, as is shown in the overlay picture in the right panel.

Figure 6. RANSAC filtered feature matching pairs and patch registration. Left Panel: RANSAC inliers of matching pairs are labeled in cyan, and outliers are labeled in red. Right Panel: With the transformation coefficients calculated from inliers, the patch image is properly registered with the template.

E. Overlay of the patch outline on the template

In the final step, we apply the transformation matrix obtained from RANSAC method on the edge image of the patch (which is generated from patch image - patch image after erosion). And the transformed patch edge is drawn on top of the template figure, and the whole picture is exported as the solution to the query of the patch, which is shown in Fig7.

Figure 7. The jigsaw solver locates the two patches shown in Fig 1B on the template. The outlines of the patches are overlaid on "Alice" in purple and pink.

V. Application Evaluation

A. Overall Performance

We tested our application on jigsaw puzzles: Alice and SnowWhite. Alice is a proof-of-concept test set which has a high definition picture as its template. We print out the picture and cut it into 8 jigsaw pieces. SnowWhite is a real 24 piece jigsaw puzzle we bought from the store. The template image is scanned into the computer and therefore has inferior quality.

The result of testing jigsaw Alice is shown below in table 1. This data set is taken by taking snapshots each patch three times at optimal condition (the phone camera is fixed at the distance where most features are extracted). All key parameters are recorded and get averaged.

TABLE I. Alice Test Result

Patch Number	Key Parameters				
	Area (pixels)	*# of Descriptors*	*# of Matching Descriptors*	*# of RANSAC Inliers*	*Success/ Overall Attempts*
1	21888.3	141.0	6.7	6.7	3/3
2	25687.7	82.3	5.0	2.3	1/3
3	24348.0	144.0	15.0	15.0	3/3
4	25606.3	71.3	12.0	11.0	3/3
5	25744.3	79.0	6.3	5.3	3/3
6	24351.3	160.3	20.0	20.0	3/3
7	22934.3	157.7	10.3	10.0	3/3
8	25321.7	112.3	13.0	13.0	3/3

From the test result we can make following observation: 1. The algorithm works pretty well on Alice jigsaw puzzle. The overall detection success rate is 91.7%. 2. For each patch, SURF algorithm finds out around ~100 feature points. And the matching pair number of descriptors is on the order of 10. This shows that our algorithm applies a strict check in filtering the matching features. The strict check also enhances the robustness of RANSAC method, as is shown here, that most of filtered pairs are RANSAC inliers.

The performance of the application on jigsaw SnowWhite is not as good. Under the same condition as we test Alice, the success rate for 24 patches is 13/24. This result reflects some limitations in our algorithm. SnowWhite is a Disney style comic jigsaw with big chunks of color and smooth edges to depict cartoon characters. The overall number of useful descriptors is much less than Alice. In addition to that, the template of SnowWhite we get is a tiny 3 inch by 2 inch printout and is scanned into computer with 600 dpi resolution. The scanning noise would come into the SURF detector and recognized as descriptors. As a result, the attempt to locate one patch would often fail, when no matching features are detected.

B. Algorithm Robustness

Theoretically, the nature of SURF descriptors guarantees the algorithm is fairly robust against all kinds of distortion, including rotation, scaling, illumination and perspective. In our test, we've measured some of the detected patches under these distortions and the results are quite consistent.

Here we would like to discuss the algorithm robustness against scaling in details. In the first experiment, we change the distance from one patch to the camera and record the number of identified descriptors, matching features after filtering and RANSAC inliers. The result is shown in Fig8A1, A2.

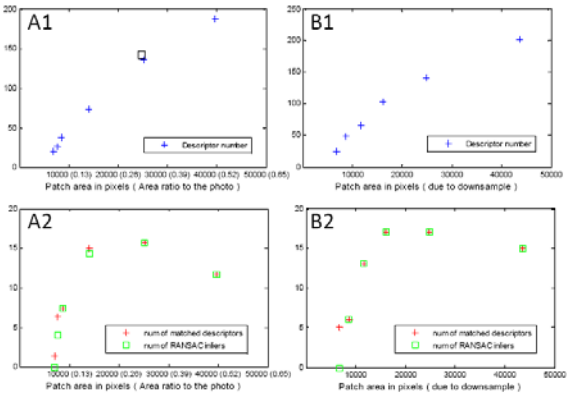

Figure 8. Algorithm robustness test against scaling (Alice patch #7). A1) SURF descriptor number detected versus patch area in the picture, and A2) Number of matching descriptors and RANSAC inliers versus patch area. Patch area is adjusted by changing the patch to camera distance, downsampling rate is fixed B1) SURF descriptor number detected versus patch area, and B2) Number of matching descriptors and RANSAC inliers versus patch are. Patch area is adjusted by downsampling the image in the preprocessing step, patch to camera distance is fixed at the value indicated by a black open square in A1)

This data is very interesting and requires further explanation. Basically, Fig8A1 shows that the number of SURF descriptors scales with the patch area, while the number of matching pairs and thus RANSAC inliers has a plateau around a sweet spot, where the algorithm is most robust against distortion.

This result illustrates how scaling affects SURF algorithm. With closer distance, more and more noise in the image are identified by SURF as descriptors. Even some of the "genuine" descriptors are overwhelmed by local fluctuation when camera and patch are close, leading the number of matching pairs to decrease. However, at the other end of the graph, the patch is so far way that all the descriptors begin to disappear due to worse and worse resolution. Between these two limiting cases, the SURF algorithm does a quite good job in rejecting noise and picking up feature points. And this is the region we claim our application to be robust against scaling.

In the second experiment, we fix the camera-patch distance and change image size by adjusting downsampling ratio. Not surprisingly, the number of descriptors and matching pairs show similar dependence on the patch area, due to the same reason we have explained above. This experiment establishes a criteria for optimizing the downsampling ratio for different patch image size.

In conclusion, our application robustness is tested under various distortions. Specifically, for scaling effect, we find the application is robust over a camera distance range, and downsampling ratio could be tuned for optimizing the robustness.

ACKNOWLEDGMENT

We thank Professor Bernd Girod, teaching assistants David Chen and Derek Pang for the instruction and guidance through the EE368: Digital Image Processing class. We also thank all of our classmates for sharing their interesting ideas and project work with us.

Liang Liang and Zhongkai Liu designed, implemented the algorithms and wrote the report together.

REFERENCES

[1] M.G. Chung, M.M. Fleck, and D.D. Forsyth, “Jigsaw Puzzle Solver Using Shape and Color”, in Proc. of ICSP, 1998.

[2] M. S. Sagiroglu and A. Ercil, “A texture based matching approach for automated assembly of puzzles,” in Proc. 18th ICPR, 2006, vol. 3, pp. 1036–1041.

[3] Taeg Sang Cho, Shai Avidan, and William T. Freeman, “A probabilistic image jigsaw puzzle solver,” to be published.

[4] C. Harris and M. Stephens. "A combined corner and edge detector," Proceedings of the 4th Alvey Vision Conference. pp. 147—151, 1988.

[5] Lowe, D. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision , 91-110. 2004.

[6] Bay, H., Tuytelaars, T., & Van Gool, L. SURF: Speeded Up Robust Features. 9th European Conference on Computer Vision, 2006 .