Solving Square Piece Jigsaw Puzzle using Computer Vision

University of Cambridge
Computer Science Tripos
Part II Project Proposal

Name: Matej Hamas
CRSid: mh704
College: Robinson
Originator: Matej Hamas
Project Supervisor: Dr C. Town
Director of Studies: Dr A. Beresford
Overseers: Dr S. Clark & Dr P. Lio
Date: 17th Oct 2014
1 Introduction

Jigsaw puzzle is undoubtedly one of the most favourite and well-known puzzles, attracting attention of people of all ages. Jigsaws have been entertaining human solvers for centuries. First puzzle dates back to late 18\textsuperscript{th} century when London mapmaker John Spilsbury carved the puzzle by cutting out the countries from a wooden map of the world. Nowadays there exist numerous different puzzle variants, ranging from simple children puzzles with several tens of large pieces to challenging puzzles containing thousands of pieces displaying relatively uniform image such as sky or landscape panorama. One can even buy 3D puzzles in form of jigsaw spheres, although 2D variants are more popular.

Assembling jigsaw is certainly challenging problem and there is even world jigsaw championship held annually in Belgium. Thanks to its intrinsic complexity, the puzzle started to draw attention of researchers from various fields such as computer vision, image processing or combinatorial optimization, pondering whether it is possible to automate the solution procedure.

Besides jigsaw being fascinating problem for its intrinsic complexity, there exist many practical applications of a computational method for finding the solution. They include challenges such as reassembling archaeological artefacts and fossils, recovering shredded historical documents, DNA modelling or even speech descrambling.

First attempt to invent a computational jigsaw solver dates back to 1964 when Freeman and Gardner [3] explored ways for solving an apictorial puzzle (i.e. pieces don’t display an image and only shape information is available), coming up with computational solution to jigsaw of nine pieces. This work, together with raising popularity and availability of computers, inspired many researchers to look at the numerous variants of the problem in past fifty years. Many of recent ones [1, 4, 6, 7, 8] concentrated on solving pictorial puzzle (i.e. one displaying an image), discarding shape information by considering only square pieces (a.k.a. patches) and using colour as the primary source of information for computational assembly of the pieces.

This project aims to take similar approach and tackle computational assembly of coloured pictorial jigsaw puzzles of known dimensions consisting of pieces of exclusively square shape with correct initial orientation, but not location (Type I puzzle according to Gallagher [4]). The primary aim is to construct a program (later referred to as Solver) that accepts two photographs as input, one of disassembled puzzle, one of the final solution (later referred to as final image clue) and finds the mapping of individual square pieces to correct positions in the final solution. In other words, given the initial state of the puzzle and the final image clue, the program should be able to find the correct place of every piece.
This task will certainly require splitting into subtasks that will deal with input image processing, piece identification and later combinatorial assembly of the pieces using various approaches, possibly consulting the final image clue. Such project organisation also allows for prototyping, since the two input images can be prepared artificially to decrease the complexity in earlier phases of the project. Artificial preparation of input will probably also help to test the program on larger inputs that couldn’t be manually manufactured in the scope of this project.

2 Description of Project

As outlined at the end of Introduction, the project aims to implement a computational Solver for pictorial 2D jigsaw puzzles with the possibility to use final image as a clue to Solver. To limit the scope of the project, some restrictions need to be introduced.

2.1 Restrictions

Pieces are of square shapes. Image dimensions, in term of number of pieces, are known. Moreover, the shape of assembled puzzle is rectangular and can be split into square pieces exactly, implying no leftovers or partial pieces. Two images are required on input, disassembled puzzle and final image. The goal of the project isn’t to deal with inputs that are deliberately of low quality. That means the images should be taken on background of uniform colour that is different to colours prevailing in the picture and particularly on its edges. Moreover, since all shapes of pieces are uniform squares, the image displayed on assembled puzzle must be of high enough resolution and contrast that doesn’t prevent from doing the assembly. Core project won’t deal with images with excessive image self-similarity or colour uniformity, although this can be considered as an extension. Both input images should be also taken at the same time, by the same camera located at approximately same location to achieve as similar conditions for photographing as possible.

In the initial state of the puzzle, square pieces must have correct orientation, they mustn’t occlude one another and they should be reasonable spaced to allow their extraction from the input image. The position of camera and lightning should be as good as possible in given circumstances in order to avoid low resolution, noisy and badly focused images. I won’t assume usage of state-of-art cameras for taking input images, but minimum requirement for camera standard will have to be specified and is likely equivalent to middle class Android phone camera\(^1\).

\(^1\)such as Samsung Galaxy SII available to the author
In principle, neither limit on the dimensions of puzzle, nor on the absolute size of the individual piece should be imposed, but the input should comply with the earlier description. Due to the practical issues of manually manufacturing the puzzle for testing purposes and the hardware limits of mediocre cameras, this probably won't allow solving either puzzles of huge dimensions or puzzles consisting of tiny pieces.

I assume to be able to manually manufacture several puzzles consisting of up to hundred pieces, but I will also artificially generate puzzles with up to thousands pieces to test the parts of the software responsible for puzzle assembly. I will also consider adding artificial noise to generated puzzles to mimic the limitation of photography. Regarding manually manufactured pieces, I intend to create at least 5 sets of puzzles consisting of tens of pieces, with largest set having up to hundred. The plan is to print out pictures of various scenes (both indoor and outdoor), glue them to the cardboard and cut the cardboard to square pieces of size approximately 3x3 cm. I will also consider using Computer Laboratory facilities, such as large size paper printer and paper cutter. However, this is not crucial for the project success at all as I find it possible to manufacture pieces without these facilities as well.

2.2 Proposed Structure of Solver

Given the description of the goal and input restriction, the structure of Solver can be summarized as follows:

1. Image preprocessing module
2. Piece extraction module
3. Final image clue extraction module
4. Puzzle assembly module
5. Artificial input creation module

2.2.1 Image Preprocessing Module

Although the input images will be of as high quality as possible in given circumstances, it is very likely that some preprocessing will have to be carried out on the image. Such preprocessing aims to enhance the image by removing noise and improving contrast to make it easier to distinguish and extract both square puzzle pieces and the final image clue.
2.2.2 Piece Extraction Module

The module will be responsible for detecting and extracting square pieces out of the input image in order to create bag of pieces, i.e. the set of all pieces of the puzzle in correct orientation. Even though pieces are rotate correctly in the initial state of the puzzle, the human error during preparation of the initial state of puzzle may make slight rotations necessary. The position of camera will certainly create need for some other, hopefully only minor, affine transformations.

2.2.3 Final Image Clue Extraction Module

Similar to piece extraction module, this module will be responsible for detection of final image clue from the second input image and its extraction. Furthermore, it will be responsible for splitting the final image clue into the square pieces according to known dimensions of the puzzle to create a bag of control pieces.

2.2.4 Puzzle Assembly Module

This module will make use of extracted bag of pieces and bag of control pieces from previous step in order to assemble the puzzle. There are multiple strategies how to do it and several will be probably attempted to reach satisfiable solution. These will likely include histogram comparison approach and template-matching techniques, such as using sum of square differences (SSD) or correlation measures. Usage of more advances feature-matching techniques such as SIFT (Scale Invariant Feature Transform) or SURF (Speeded Up Robust Features), used e.g. in [5], will be considered as a possible extension for boosting performance.

Depending on results achieved with aforementioned techniques, other approaches using piece compatibility measures to find neighbouring pieces out of bag of pieces may be tried (e.g. SSD between edges or Mahalonobis distance [6]), accompanied with greedy strategy of puzzle assembly [7].

2.2.5 Artificial Input Creation Module

As discussed in section 2.1, it is probably not feasible to manually manufacture large enough input sample to exhaustively test puzzle assembly module. Artificial input preparation module will split the given image into square pieces to create and sample input image complying with restrictions discussed earlier.
2.3 Structure of the Project

The project itself may be split into following section, in chronological order:

1. Background reading on the subject and familiarizing with tools such as OpenCV.
2. Implementation of Solver
3. Gathering test samples and evaluation of Solver
4. Dissertation write up

2.4 Possible Extensions

- Implementing some additional technique outlined in previous sections
- Improving Solver to easily deal with thousands of pieces
- Weakening input restrictions to allow pieces in arbitrary \((0, 360^\circ]\) initial rotation
- Implementing genetic algorithm-based solver as mentioned in [8]
- Using Solver for creating an Android application that solves jigsaw puzzle by sending input to the server that runs Solver and sends back the correct solution
- Improving Solver to deal with inputs of substantially lower quality or images with high level of self – similarity and colour uniformity

3 Evaluation Metrics and Success Criteria

3.1 Evaluation Metrics

I intend to use 3 metrics for evaluating Solver’s performance, all of which are inspired by the previous work of several authors and I consider them to be reasonable quantitative measures.

1. Direct comparison [2] - percentage of pieces that are at correct absolute positions in the final solution produced by Solver
2. Neighbour comparison [2] - percentage of correct pairwise relationships between adjacent pieces
3. Largest component [4] - percentage of pieces that are part of the largest component, i.e. the largest area of the puzzle formed by pieces that are correctly placed with respect to their neighbours from that component
3.2 Success Criteria

The project will be considered successful if the following objectives are met for the puzzles of up to hundred pieces:

1. Given input images of reasonable quality, image preprocessing module can prepare images suitable for further processing by other modules.

2. Piece extraction module can correctly extract the square pieces from an input image, preparing the bag of pieces for further use and final image clue extraction module can correctly extract the final image from the second input.

3. Given bag of pieces, final image clue and bag of control pieces from previous modules, puzzle assembly module can correctly assemble the puzzle with reasonable accuracy. We will use all three metrics as described above and Solver will be tested on manually manufactured puzzles as described at the end of section 2.1. There will be three test cases for every puzzle, starting in random initial configuration, and each individual test case will be evaluated under every metrics. Average of results will serve as evaluation figure of Solver’s performance under that particular metrics. The puzzle assembly module will be deemed successful if it can correctly assemble each puzzle with reasonable accuracy under at least one metric.

4 Starting Point

I am completely new to Computer Vision and the course on it is lectured only in Lent term this year. My knowledge to Image Processing is limited to Part 1b Graphics course and my problem specific knowledge in computational jigsaw puzzle solving is limited to few papers I’ve seen in past couple of days.

The project will be most likely implemented in C++ using some features of Open Computer Vision library (OpenCV). I have some experience with C++ from Part 1b course and summer internship, but I have never used OpenCV before. With exception of this library, I expect my project to be build from scratch and not be based on any existing codebase.

Having participated in Android Summer Camp, I have very basic knowledge of Android SDK and quite good understanding of Java thanks to Part 1a and 1b Computer Science Tripos courses. I would use these skills if I chose to develop an Android application as an extension to the project in the event of time abundance.

I will be typesetting my dissertation in \LaTeX which I haven’t used before. Regarding version control systems, I will use GitHub that I have experiences with.
5 Work Plan

Week 1 – 2, (20th Oct – 2nd Nov)

**Deadline:** Project proposal submission on 24th Oct

- Complete and submit project proposal
- Background reading on Computer Vision and more research on work on computational jigsaw solvers
- Reading about and experimenting with OpenCV
- Setup development, backup and version control systems

Week 3 – 4, (3rd Nov – 16th Nov)

- Continue background reading and familiarizing with concepts OpenCV
- Decide what techniques to use for image preprocessing module
- Implement image preprocessing module and smoke test it

Week 5 – 6, (17th Nov – 30th Nov)

- Decide which techniques to use for piece and final image clue extraction modules
- Implement these two modules and do some preliminary testing on a few test samples

Week 7 – 8, (1st Dec – 14th Dec)

- Create more test samples to test implemented modules
- Test the modules
- Decide which approach to use first for implementation of puzzle assembly module and start implementing it

Week 9 – 11, (15th Dec – 4th Jan)

- Finish implementation (probably inefficient) of puzzle assembly module
- Make smoke test of the implementation and Solver as whole
Week 12 – 13, (5th Jan – 18th Jan)

- Catch up time for any outstanding issues
- Write the Progress report

Week 14 – 15, (19th Jan – 1st Feb)

Deadline: Progress report submission on 30th Jan

- Prepare and rehearse the presentation
- Implement the artificial input creation module

Week 16 – 17, (2nd Feb – 15th Feb)

Deadline: Presentation on 5th – 10th Feb

- Prepare reasonable sets of test puzzles for evaluation (as described at the end of section 2.1)
- Evaluate Solver both using manually manufactured puzzles and inputs from artificial input creation module

Week 18 – 19, (16th Feb – 1st Mar)

- Having evaluation data, make improvements to Solver, possibly using different techniques.
- Re-evaluate Solver, obtaining decent statistics
- Process evaluation data to have them ready for the dissertation write up

Week 20 – 21, (2nd Mar – 15th Mar)

- Catch up time for any outstanding issues
- Plan and draft the introduction and preparation parts of the dissertation
- If no issues emerge and time allows, choose, research and start implementing one of extensions
Week 22 – 23, (16\textsuperscript{th} Mar – 29\textsuperscript{th} Mar)

- Plan and draft the implementation parts of the dissertation
- Complete the extension if time allows

Week 24 – 25, (30\textsuperscript{th} Mar – 12\textsuperscript{th} Apr)

- Finish the complete draft of the dissertation by finishing evaluation and conclusion sections
- Send the draft to the Supervisor
- Catch up time for unforeseeable complications

Week 26 – 27, (13\textsuperscript{th} Apr – 26\textsuperscript{th} Apr)

- Format and edit the dissertation over Easter break, including all appendices
- Incorporate Supervisor’s feedback into the dissertation and send ameliorated version to Supervisor and Director of Studies

Week 28 – 29, (27\textsuperscript{th} Apr – 10\textsuperscript{th} May)

**Deadline:** Final deadline for submission on 15\textsuperscript{th} May

- Make final edits to the dissertation based on the received feedback
- Complete the final version, including all the compulsory formal sections, such as proforma and declaration of originality
- Print, bind and submit the dissertation on Monday 11\textsuperscript{th} May (Monday before hard deadline)

6 Resources Required

I will be using my own laptop for the whole duration of the project (Mac OS X, 2.2 GHz Intel Core i7, 16 GB RAM). If this computer failed, I would work on MCS workstations. I will be using Git and GitHub for version controlling and will also make weekly backup on 600 GB external hard drive using Time Machine. After every significant updates, I will also make back ups of important documents (such as dissertation drafts) to Google Drive.
Regarding camera and possible Android device, should time allow for implementing an Android extension, I will use my Samsung Galaxy SII (GT-I9100) with Android 4.1.2 and 8MP camera resolution as stated by tech specs.

I will use Xcode with OpenCV library as main IDE for development.

References


