Automated Jigsaw Solver

Manu Lam
16458032

Final Year Project – 2019/2020
B.Sc. in Computer Science and Software Engineering

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfillment of the requirements for the

B.Sc. Computer Science and Software Engineering

Supervisor: Dr. Charles Markham

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of Computer Science and Software Engineering qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Manu Lam Date: 14/04/2020

Abstract

The Jigsaw puzzle problem is an important research topic that has many applications in areas such as the reconstruction of archaeological artifacts, 3D reconstruction of individual anatomy from medical image data and recovery of shredded documents or photographs.

There are various methods that have been used to solve Jigsaw puzzles. These methods range from Artificial Intelligence, Image feature extractions, probabilistic methods and in our case techniques based on the shape of the pieces. In this report, we will be looking at our approach to solving a Jigsaw puzzle by using the shape of the pieces alone.

We will be looking at how we extract Jigsaw pieces from an image, the many stages of enrichment of Jigsaw pieces, characterizing the Jigsaw pieces so that the machine can understand this logic, and matching the pieces with others using a simple pixel matching technique.

The Jigsaw Solver created could fully extract, enrich and characterize Jigsaw pieces but the quality of enrichment was not rich enough, leading to a poor matching phase that could not return the best-fit pieces but only returning all pieces that could fit.

Chapter 1

Introduction

1.1 Topic addressed in this project

The automated jigsaw puzzle problem has been around since 1964 [1]. Freeman implements the first automated jigsaw puzzle by using the shape of pieces alone [1]. There are many other implementations for the jigsaw solver problem such as extraction of image features, the probability with image features and generic algorithms. The method of solving the puzzle with shape alone is what we will be looking at in my experiment. The main complexity lies in matching two pieces together and a combination of all matched pieces. Although other problems do arise with the extraction and enrichment of pieces.

1.2 Motivation

Many papers on this topic help reproduce related problems such as the reconstruction of archeological artifact fragments, 3D reconstruction of individual anatomy from medical image data and even digitizing the shape and color of large fragile objects such as statues [2][3][4]. Another very interesting benefit of this problem is the recovery of shredded documents or photographs [5]. By simply capturing an image of shredded paper, we could possibly recover the entire document back with these types of reconstruction algorithms.

Solving jigsaw puzzles using computer vision is an attractive problem. It benefits the advancement in the computer vision field while being both challenging and interesting.

1.3 Problem statement

This section will discuss briefly the problems that occur when solving a Jigsaw puzzle by machine.

Our end goal is to solve a jigsaw puzzle that has been scrambled, entirely by shape. This Jigsaw puzzle must be a scanned image and multiple steps are executed in order to achieve our end result.

The steps that are taken to achieve the end solution:

1st step - Capture a full image of a scrambled jigsaw puzzle
2nd step - Extract all jigsaw pieces from the puzzle and filter out junk noises
3rd step - Apply enhancement algorithms that characterize our jigsaw pieces
4th step - Matching, find any two pieces that have the best fit
5th step - Combinatorial, seeing if the whole picture is solved, if not backtrack

Image quality

The quality of the captured jigsaw puzzle is hugely important, as this could give us poor pixel quality (blurs) or implement noise into our image. We must also consider the lighting, scale and what type of background when capturing the image.

Extraction and Enrichment of Jigsaw pieces

One of the main questions for extraction is how do we convert from an image into separated jigsaw pieces that are characterized so that a computer can understand it?
We need to apply many algorithms together that can break down the problem step by step and then later build it back up as characterized jigsaw pieces.

Matching and Combinatorial

The approach to solving jigsaw puzzles by shape alone has two main difficulties. One of our problems is with the successful matching of pieces. When a scanned piece is matched with another piece, we are not guaranteed that these pieces fit. When fitting two pieces by hand, we can usually feel a click when they fit. Another problem is combinatorial. There are a vast number of ways that pieces can be assembled. If a brute force or recursive algorithm is implemented, this could take a long time to solve a single piece.

1.4 Project Approach

There are many approaches to solving Jigsaw puzzles, this section briefly reviews my approach to the project.

Image quality

It all starts from an image capture of a full jigsaw puzzle that is scrambled. A visualizer was used to capture our image, which provided good lighting, scale, and a good quality image. Although a few noise disturbances did manage to get into our image, it was later filtered out during extraction.

Extraction and Enrichment of Jigsaw pieces

Although we have captured an image from this point, we still need to split this image into multiple smaller images (**Figure 1.4a**) and then characterize each image as its jigsaw piece. These pieces should contain information about how much the piece was rotated, what edges it contains and if they're inner/outer or flat edges. This leads us towards a better machine that understands the characteristics of each piece. The pieces can now be manipulated as an object.

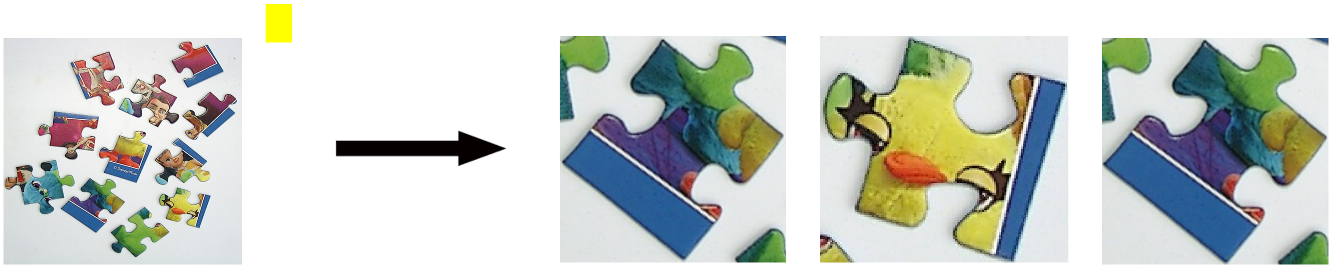


Figure 1.4a - Splitting of full Jigsaw image into separate pieces

Matching and Combinatorial

The goal of matching and combinatorial is to place all the Jigsaw pieces together and solve the puzzle. Matching the pieces together is quite difficult, we must first add some logic to the pieces such as a side piece must have two neighbouring side or corner pieces. It is possible to reduce the load on our matching algorithm and improve performance overall by applying these types of logic. After all of the logic is added, we should be able to get a few pieces that are possible to fit in another piece. We should then compare the pixels between the two sides and see which one has the best fit, we then place that piece and move on. If any mistakes are made (etc. unsolved solution), we recursively return back to the previous best fit and take the other possible fits and so on.

1.5 Project Achievements

- Able to successfully extract all jigsaw pieces from a scrambled jigsaw puzzle while filtering out junk pieces.
- Ability to break down a highly complex problem and use algorithms that are combined with others to enhance and characterize a jigsaw piece.
- High use of multiple algorithms combined to create new algorithms
- Characterizing jigsaw pieces with certain algorithms, providing details on a side by classification, rotation, and possible links.
- Communication with my supervisor to ensure the details of the project were well thought of and not invalid.

Chapter 2

Technical Background

This chapter provides an overview of the various methods that have been used to solve Jigsaw puzzles. It also provides details of the library used in the development of the code for this report.

2.1 Review of existing approaches to solving Jigsaw puzzles

There have been many approaches to solving Jigsaw puzzles, this section reviews some of the approaches relevant to the project.

2.1.1 Techniques based on the shape

The approach to solving jigsaw puzzles based on shape alone is a very complex one. Goldberg discusses this in length, stating that it is very difficult to match the jigsaw pieces together and finding the correct combination to assemble the entire puzzle [6]. It is an NP-Complete problem [7] that could fall into linear time when using a rotation-invariant boundary encoding [8]. This rotation-invariant boundary encoding simply means solving the puzzle according to all pieces that have 2 or more sides attached to others. Altman discusses his way of matching two pieces by using Weiner's string matching technique combined with compact position trees which solved this puzzle within linear time [8].

2.1.2 Artificial Intelligence

State of the art generic algorithms could be used to solve puzzles successfully up to 30,755 pieces and within a reasonable time [9]. These generic algorithms performances have been drastically increased over the years, as the previous attempts were limited to 64 piece puzzles [10]. GA approaches this puzzle by generating many iterations of the final solution (doesn't have to be correct) and using a measure ranking of the likelihood connecting two pieces known as compatibility. The highest compatibility is then found throughout all iterations and returns a correct solution.

2.1.3 Image Features

The image feature approach is used to obtain an accurate measure of edge similarity between two jigsaw pieces and also the image on the pieces. It is done by extracting imagery information from each jigsaw piece and characterizing them. Then using a greedy algorithm where it can successfully solve puzzles up to 320 pieces [11].

2.1.4 Probabilistic methods

A probabilistic image solver approach is similar to the image feature approach but focuses more on probability. This approach does not depend on jigsaw shapes but focuses on the image within the pieces itself. Cho discusses this in length, stating that the problem is reconstructing an image from a bag of square, non-overlapping image patches [12]. Square pieces were used during the research and each piece had its image analyzed as a patch (image colour distribution). These patches were then measured incompatibility with other patches using a dissimilarity-based metric. The solutions were not accurate enough without the piece's edges but the results were fascinating. With the increase of anchor patches (serves sparse-and-accurate local evidence) provided, it has improved the accuracy of the final solutions.

2.2 Overview of SDK’s and libraries used

2.2.1 OpenCV-Python (CV2)

OpenCV is an open-source Python library designed to solve complex computer vision problems [13]. The library has over 2500 optimized algorithms which include state of the art computer vision and machine learning algorithms [14]. In our case, these algorithms can be used to extract information from images by enhancing them with Canny edge detection, Harris corner detector, HoughLinesP and many more.

2.2.2 NumPy-Python

NumPy is an open-source Python library designed to solve complex scientific computing problems [15]. The library contains powerful features such as an N-dimensional array object, linear algebra, and Fourier transform. NumPy works extremely well with OpenCV. As the image pixels extracted are plentiful, we need to contain them in an efficiency array such as NumPy. After the image pixels are stored, we can apply powerful algorithms such as matrix transposing and extracting specific pixels with masking. Matrix transposing is used to rotate an image by a certain degree and extracting specific pixels will allow us to enrich the image further with characteristics.

2.2.3 Python-Imaging-Library (PIL)

PIL is an open-source Python library designed for opening, manipulating and saving different image file formats [16]. The library provides features such as masking, pixel manipulations, image enhancement with filtering and also adding text to images. In our case, PIL was used to merge images to create certain templates and allow us to show the user our end product with a collage of jigsaw piece images (Figure 2a).

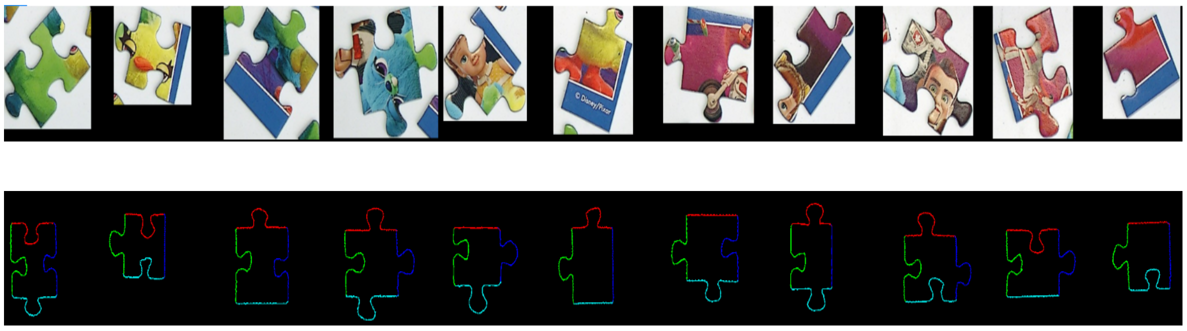


Figure 2a - Jigsaw pieces after edge side classification

Chapter 3

The Problem

This chapter provides a detailed overview of the problems that need to be overcome when solving Jigsaw puzzles by machine. It also provides a high-level thought process of how to approach these problems from my given perspective.

3.1 Problem analysis

There are plenty of difficulties when solving a Jigsaw puzzle by machine. These difficulties range from image capture, image extraction, enhancement and characterizing, matching and combinatorial of Jigsaw pieces. For anyone that has never done anything related to computer vision before, it would be a very difficult task to approach.

3.1.1 Image Capture quality

The image quality is very important to consider within this problem. This issue is related to the quality, scale, lighting, and background of the image. These issues were carefully thought of and a visualizer was used to solve most of these issues. The Jigsaw pieces were simple and large, making it very easy to detect and solve.

3.1.2 Image Extraction of Jigsaw pieces

Firstly, we must translate a captured image into a component the machine can understand. The problem must be broken down completely and every step is laid out as steps of algorithms.

My approach to this difficult new problem was to understand how we could objectize pixels from an image and later manipulate them as Jigsaw pieces (Objects). This would later become the **enrichment and characterizing** stage of the Jigsaw puzzle. The problem was broken down to the very first step, **extraction** of Jigsaw pieces from an image.

This problem can be solved by using Python's CV2 library where it has major uses of contour extraction algorithms which helped me progress. Although this helped, it did include some more difficulties such as noise disturbances that were extracted together with the Jigsaw pieces. A filter was then thought of, to filter out any image with a pixel count smaller than a certain threshold.

3.1.3 Enrichment and Characterization of Jigsaw Pieces

Now that the piece is extracted, it still has too much unnecessary information such as the coloured image itself, it also has other pieces in the background and is not rotated. The next step here would be to enrich the piece further, first by breaking down the problem even further.

Shape Enrichment

One of the problems here is trying to solve the jigsaw by using shape alone. How is this achievable?

One way this is achievable would be extracting the shape outline of each piece alone. This could be done with many edge detection algorithms. The other background pieces could be masked out, this reduces the noise inside each image.

Rotation of Jigsaw piece

Now that the piece is outlined only by its edge, we need to consider rotating the piece so that it is aligned with all other pieces. We do not take into consideration if the pieces are upside down for example. How do we rotate a Jigsaw piece based on everything we have?

My analysis of this problem was to use a corner detection algorithm. The corner detection algorithm is used for returning all points that look like a corner point. Once the algorithm returns the four corners, it should be able to rotate the image parallel to our wanted plane.

Side Classifier (Bottom/Left/Right/Top sides)

Another problem is to differentiate between the four sides of any piece such as the top side, left side, right, and bottom side. This type of classifier is very important, as it allows us to work with each side of the piece separated from one another. How is this achievable?
It is important to use the information from the previous step. My approach to detailing each side of the jigsaw piece was to create a square from the four corners of the piece as received in our previous step **Rotation of the Jigsaw piece**. This square is formed by four different lines, where each line defines a single side. Then for every pixel in the image, we find the closest line to this pixel. This would then return a map of pixels to their closest side. We should then be able to work with these four sides completely separated from one another.

Edge Type Classifier (Flat/Inner/Outer)

Now the problem is understanding what type of edge each side is. These sides could either be a flat edge, inner edge or an outer edge. This enrichment step is crucial, as we later need to add logic in solving the Jigsaw puzzle. For example, we should match all inner edge pieces with outer edge pieces and all flat edges must link with other flat edges. How is this achievable?
Firstly, we would go through each Jigsaw piece and compare their sides. There are two categories for these edges, **Flat edge,** and **Inner/Outer edges**.

Flat Edge

To determine a Flat edge on a single side, we must apply a straight line detection algorithm with a high pixel threshold, so that no gaps can be let through (this will not classify inner/outer as a flat edge).

Inner/Outer Edge

With the four lines computed from the previous step **Side Classifier**, we can use these lines and their corresponding sides (Bottom/Left/Right/Top) to determine if a side is an inner or an outer by calculating the mean of the pixels. The side of the piece should be an inner edge if the mean of the pixels falls beyond the line. Similarly, if the mean of the pixels falls before the line, it is an outer edge.

3.1.4 Matching and Combinatorial

These two parts are by far the most challenging and problematic. Many things are considered here, are the Jigsaw pieces enrichments good enough? Do we have the best-fit piece returned? How many combinations are there if we mess up?

Matching

The difficulty here is to correctly match two Jigsaw piece's sides together and return the best-fit piece. We must consider logic that eliminates unnecessary search such as a side piece must have 2 neighbouring side or corner pieces. This type of logic may reduce the load and increase performance for the machine. Then we should match the pixels between the Jigsaw piece's side we are trying to solve and all the possible pieces we can match, the best pixel match should become the best fit. It is possible to calculate the area of an inner/outer side and the other possible pieces' sides and try to eliminate some matches that do not have the requirements to fit.

Combinatorial

There are many difficulties in the combinatorial matching of pieces. One of the difficulties is when any matched pieces do not actually go together, we will have to go back all the way and use the second best-matched piece and so on. This would take a lot of computing power and time even for a small Jigsaw puzzle. The approach to this would be backtracking with recursion until all possible edges of the best fit are exhausted.

Chapter 4

The Solution

This chapter provides a detailed solution of the steps taken to achieve the final solution. It also includes the algorithms that were used during this project.

4.1 Extraction of Jigsaw pieces

A full jigsaw puzzle is captured as an image at first. We then extract every piece from this image by computing all the bounding contours within this image. Then extracting the best rectangle fit from each bounding contour so that we do not miss any pixels during extraction (Figure 4.1a).

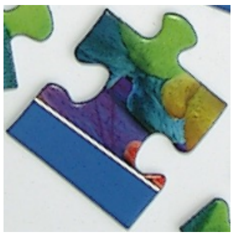


Figure 4.1a: Extracted Piece

4.1.1 Filtered pieces

Figure 4.1b: Junk Piece

As jigsaw pieces are extracted from our image, there is a possibility that we have noise disturbances such as small pixels that get extracted too (Figure 4.1b). A filter is added to exclude any image that is below a certain pixel threshold, this prevents any small noise disturbances from being included in our jigsaw piece set.

4.2 Enrichment and Characterizing of Jigsaw Piece (Multistep process)

After all the pieces are now extracted as their own, we can now apply the enrichment process to each piece. This section reviews most of the algorithms being implemented.

4.2.1 Canny Edge with Median Filter

Before applying Canny Edge to an image, we must turn the image into a grayscale image (Figure 4.2a) and apply a median filter to preserve our edges. This helps us extract the edge pixels better by calculating the mean of the neighbour pixels. We then apply the Canny Edge algorithm which finds all the edge pixels within an image and returns the outline edges of that image (Figure 4.2b).

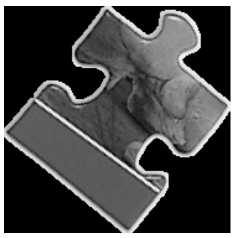


Figure 4.2a: Greyscale

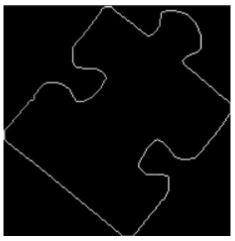


Figure 4.2b: Canny Edge

4.2.2 HarrisCorner and Rotation

This was by far one of the hardest algorithms created during this project. Harris Corner is applied to the Canny Edge applied image. This returns all points that could classify as a corner point on our image (Figure 4.2c). By using these points, the maximum rectangle created by any four of these points were found. This gives us the absolute corners of our piece and then we rotate the piece by transposing the matrix to become parallel to our wanted plane (Figure 4.2d).

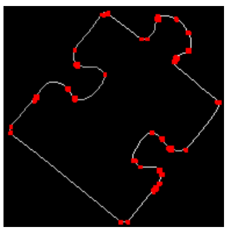


Figure 4.2c: Harris Corner

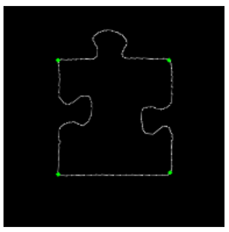


Figure 4.2d: 4 Corner points

4.2.3 HoughLinesP

One of the characteristic steps for our jigsaw pieces is to determine how many straight edges are within the piece. HoughLinesP is applied to each side of our piece (Bottom/Left/Right/Top) and if there is a single straight line that exceeds 70 pixels in length, we count that side as a flat edge (Figure 4.2e). The total count of flat edges determines if a piece is a corner piece or a side piece. This type of edge characteristic helps us to place pieces together.

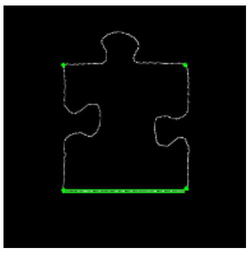


Figure 4.2e: Flat Edge

4.2.4 Classifier for determining sides (Bottom/Left/Right/Top)

A classifier is used to determine within a jigsaw piece, which pixels belong to which side (bottom/left/right/top side). We apply this classifier that was made from the Harris Corner step previously. The four corner points of the Jigsaw piece are joined together and form four lines (Figure 4.2f) that help us calculate the minimum distance between the pixels found and each side of the classifier (Figure 4.2g). Since some sides fall shorter on the distance to another side and return a bad output image, we must apply the nearest neighbour approach to find its true classified side.

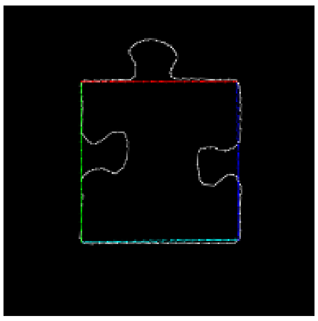


Figure 4.2f: Side type Classifier

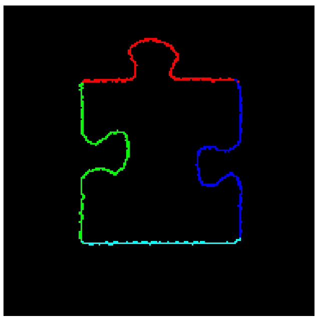


Figure 4.2g: Classified Jigsaw piece by Side

4.2.5 Classifier for determining edge types (Inner/Outer/Flat)

This classifier is used to determine within a jigsaw piece, what the sides are in terms of edges (Inner/Outer/Flat). We first apply HoughLineP to determine if the edge is straight or else we apply this classifier by calculating the sum of pixels that fall over the lines computed in the previous classifier (Figure 4.2f). An example is: If the mean of the pixels is over the classifier line, this returns an inner edge on the left and top side of any piece but returns an outer edge on the bottom and right side. Adding this classifier for determining edge types, we end up with green edges for flat sides, blue edges for inner and red for outer edges (Figure 4.2h).

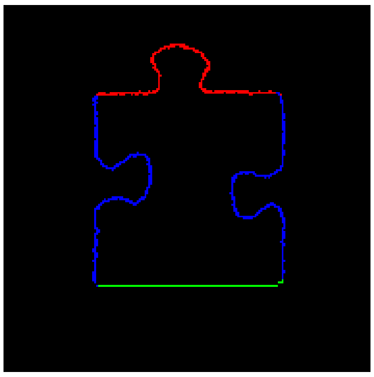


Figure 4.2h: Classified edge types

4.3 Matching of Jigsaw Pieces

After all the pieces are now enriched fully in their own separate Jigsaw piece object, we can now match each piece with other pieces, until the puzzle is solved.

4.3.1 Matching Two Jigsaw pieces

The process of matching Jigsaw pieces should always start at a corner piece and then solving all the side pieces first. After the side pieces and corner pieces are all solved, we can move inwards and repeat the process. My approach to this was by locating all corner pieces and side pieces, then finding all pieces that could possibly match with our corner pieces. Many possible pieces were found but not the best-fit. This created an issue, as our Jigsaw solver couldn't progress without this.

4.3.2 Highlighting of a possible Jigsaw piece fit

When trying to match two pieces of possible fit, this highlighting technique was implemented to clearly show the user which two edges of two pieces are being compared (Figure 4.3a). Highlighting the original Jigsaw piece was originally my idea but it did not appear too well (Appendix 2a). A possible fix for this is by giving the pixels a thicker layer of highlighting.

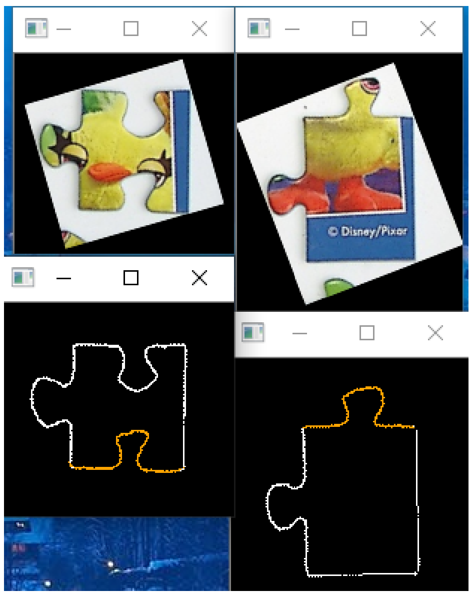


Figure 4.3a - Highlighted sides

Chapter 5

Evaluation

This chapter gives an overview of the steps taken and what changes could have been implemented to achieve better results.

The steps that are taken to achieve the end solution:

1st step - Capture a full image of a scrambled jigsaw puzzle
2nd step - Extract all jigsaw pieces from the puzzle and filter out junk noises
3rd step - Apply enhancement algorithms that characterize our jigsaw pieces
4th step - Matching, find any two pieces that have the best fit
5th step - Combinatorial, seeing if the whole picture is solved, if not backtrack

5.1 Image quality

The visualizer used in the project worked very well, it provided us the perfect lighting, scale, and quality capture of an image. It would have been nice to have worked with multiple images, this was not intended from my start point but only to work on a static Jigsaw image.

5.2 Extraction and Enrichment of Jigsaw pieces

The algorithm used to extract our Jigsaw pieces from the image worked well but turning the Jigsaw piece image into grayscale and applying CannyEdge had implemented noise into our outlined pixels. This made the matching stage of our project a very difficult task, as we couldn't use simple algorithms to compare pixel points. The noise created had increased the outlines of edges by a few pixels, making a gap between every fit. Another layer of enrichment should be applied before the Canny Edge algorithm, one that can provide better pixels of our Jigsaw pieces.

The Harris Corner and Maximum rectangle algorithm that was used to rotate the Jigsaw piece worked extremely well, it gave really accurate results and looked great during its progress stages (Figure 4.2c, Figure 4.2d).

The Classifiers to detect sides and edge types had also worked extremely well, it enriched our plain Jigsaw piece image into an object that the machine can understand. The Jigsaw piece had characteristics such as four unique sides which were all classified as an inner, outer or flat edge. HoughLinesP was used to achieve the flat edge detection, which is one of the CV2 Library's algorithms.

5.3 Matching and Combinatorial

Matching the Jigsaw pieces had a good start, logic was pushed through for corner pieces to match with sides and progress was achieved. The algorithms created for matching were solid but the pixels of the pieces were slightly off. To solve the pixel matching, we would need to first fix the Jigsaw piece's pixel accuracy. An idea was to have human interactions while the pieces were shown on screen, this would have made the project a semi-automated process.

5.4 The Jigsaw Solver's progress being displayed

Many functions were created for showing users the progress of the Jigsaw solver, although not much time had gone towards this feature, it had gone really well. It included a full merge of the Jigsaw progress (Appendix 1a). Another visualizing algorithm that went well is the Highlighted Matched Pieces that showed which two edges can match (Figure 4.3a). A better implementation could have been implemented but very little time was given to these features which were a bad choice.

Chapter 6

Conclusion

This chapter provides an overview of the entire project, how the results turned out and what else could be implemented in the future.

6.1 Results discussion

From the results of this project, it is clear that the extracted Jigsaw pieces' pixels were not clear enough for matching. This is during the process of comparing two edges and adding a comparator that returns the best fit between an edge and all possible links. As recommended in the Future Work section below, we should refine these edges at the start to ensure that no disturbances are involved during the matching phase.

It was disappointing not to be able to finish this project, it was highly complex and had more to it than I initially thought. Although not being able to finish the project, I was very happy with my results in the Extraction and Enrichment of the Jigsaw pieces and being able to create everything from scratch. This project was definitely a new challenge for me and it had a lot of problems to be solved which was very interesting and satisfying when solved.

6.2 Project Approach

This project's approach went very well at the start but complex and challenging near the end. Everything was broken down and solved at a low level from the start to finish. The entire process was very satisfying where we could see each step of the process being implemented after each state.

The project was first approached with a 2d problem (digitized image) of this project and solved fully with only the images on the jigsaw pieces and not the shape of the pieces itself. This gave me a brief overview of the problems I was yet to encounter.

A lot of my time was spent researching algorithms that could solve problems such as rotating a piece, how to detect lines and classifying the sides/edges. If there were anything different that I would do differently in my approach, it would be spending more time in developing ways of matching pieces without needing the pixels to be perfectly extracted.

Near the end of this project, time was definitely an issue. Too much time was spent working towards some complex algorithms such as piece rotation and classification.

6.3 Future Work

From the current work done in this project, there are plentiful things that could be implemented in the future. This section will discuss some of the ideas that could be made possible.

- From the outcome of this project, I would add another layer of enhancement on the puzzle to increase the accuracy of Jigsaw pieces' edges. An example of this would be refining the edges before we apply the Canny Edge algorithm to ensure we have the absolute correct pixels without any disturbances.

- Adding more Unit testing and integration tests to provide full test coverage of the system. This ensures that the whole system is tested throughout each stage of the process and that the code is ensured to be running the way it was implemented.

- Adding better user interfaces should increase the user experience and help us traverse the solving transition better in real-time. One of the Libraries that was looked into is Python Kivy, a library that creates elegant and simple interfaces.

- Automation with a robotic arm could be implemented with this project. This gives a completely new feature by solving the puzzle at first and then using an automated way to complete the Jigsaw puzzle. An idea of this would be generating the coordinate output by adding piece locations to move towards and the rotations needed. This idea could then be exported in 2d to a popular robotics library known as ROS.

- Applied Machine learning for self-learning images could also be implemented with this project. Since our project can extract jigsaw pieces from an image, this will allow us to sort jigsaw pieces by their types and have a classifier added onto it. This could possibly allow a regression algorithm to determine how these pieces differ from each other, helping us solve this puzzle with the power of machines.

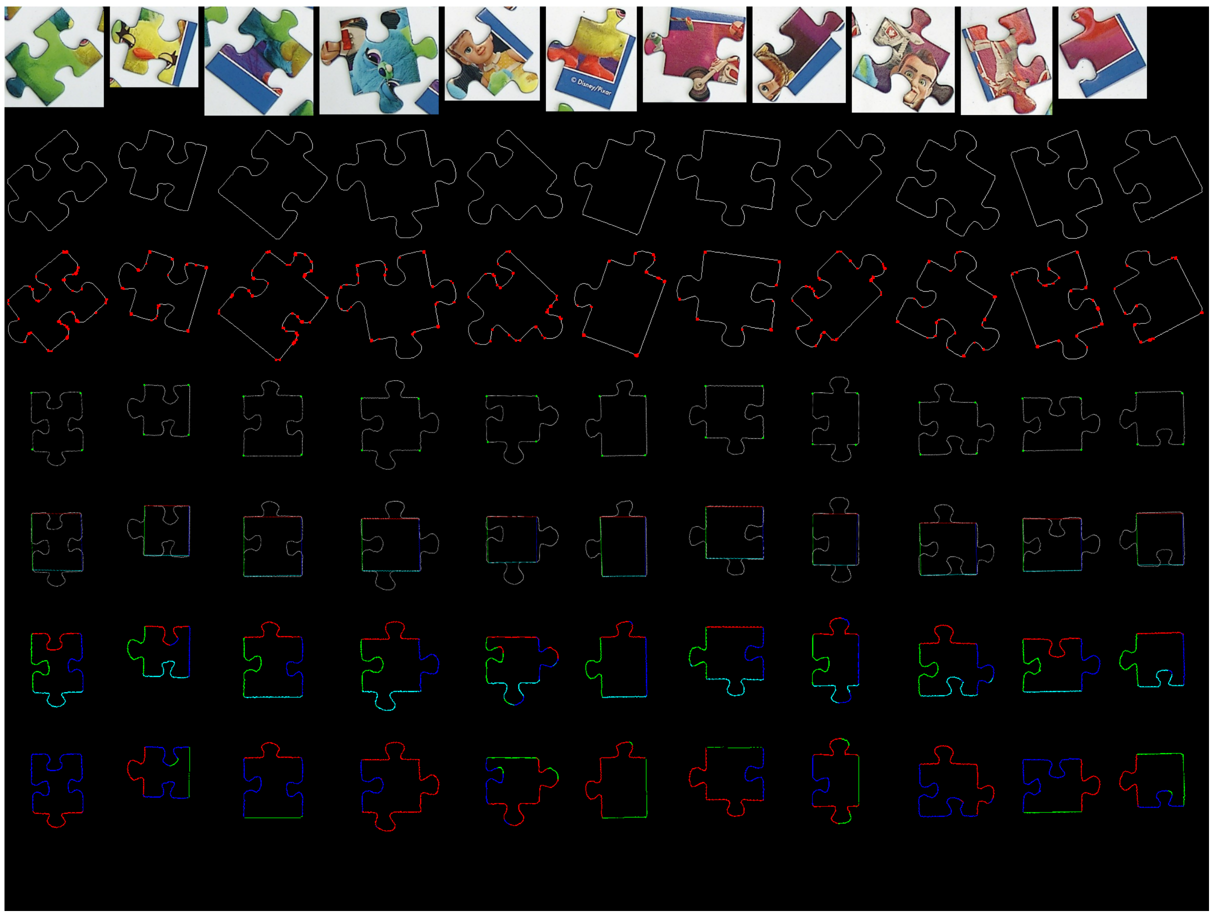
References

[1] H. Freeman, L. Gardner, Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition, IEEE Trans. Electron. Comput. 13, 118–127, 1964.
[2] H.C. da Gama Leitão, J. Stolfi, Automatic reassembly of irregular fragments, Tech. Report IC-98-06, Univ. of Campinas, 1998.
[3] S. Zachow 3D reconstruction of individual anatomy from medical image data: Segmentation and geometry processing, Zuse Institute Berlin**,** 2007.
[4] M. Levoy, K. Pulli, The Digital Michelangelo Project: 3D Scanning of Large Statues
[5]Cao, S., Liu, H., Yan, S.: Automated assembly of shredded pieces from multiple photos. In: IEEE International Conference on Multimedia and Expo, pp. 358–363, 2010.
[6] D. Goldberg A global approach to automatic solution of jigsaw puzzles, MIT Department of Mathematics, Computational Geometry 28, 165–174, 2004.
[7] Demaine, E., Demaine, M.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics 23, 195–208, 2007.
[8] T. Altman, Solving the jigsaw puzzle problem in linear time, Appl. Artificial Intelligence 3, 453–462, 1989.
[9] D. Sholomon, An Automatic Solver for Very Large Jigsaw Puzzles Using Genetic Algorithms, Bar-Ilan University, Vol. 17, No. 3, pp. 291–313, September 2016.
[10] Toyama, F., Fujiki, Y., Shoji, K., Miyamichi, J.: Assembly of puzzles using a genetic algorithm. In: IEEE International Conference on Pattern Recognition, vol. 4, pp. 389–392
[11] T. R. Nielsen, Solving jigsaw puzzles using image features, Volume 29, Issue 14, Pages 1924-1933, 2008.
[12] T. S. Cho, A probabilistic image jigsaw puzzle solver, Tel-Aviv University, 2010.
[13] OpenCV-Python, Document URL **https://pypi.org/project/opencv-python/**
[14] OpenCV-Python About URL **https://opencv.org/about/**
[15] NumPy-Python Document URL **https://numpy.org/**
[16] Python-Imaging-Library PIL URL **https://pypi.org/project/Pillow/**

Appendices

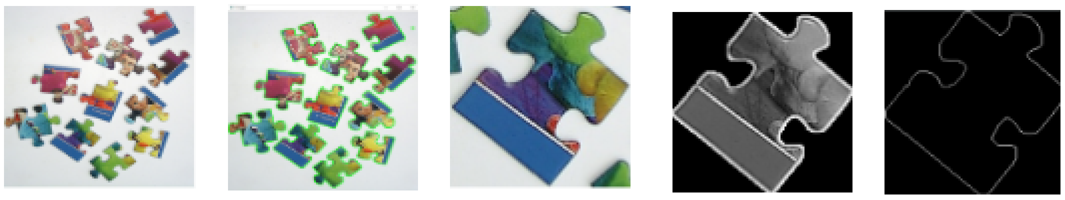
Appendix 1 Screenshots of the project implementation

After many hours of different implementations, I found this solution to be the most appealing one for users. This can tell them the whole process in 1 image. From the start image to the fully annotated and enriched image at the very end. I am very happy with this progress. The ones that came before these were a bit complex.



Appendix 1a - Full progress merged into a template

This was the previous version of the progress stages



Classifying the sides implemented new troubles for me.

We can see here that not all sides are classified perfectly, I learned from this and created better algorithms to improve the pixel matching accuracy. K-Nearest Neighbour algorithm was applied to correct each pixel that broke a certain threshold to take its neighbours' colour value.

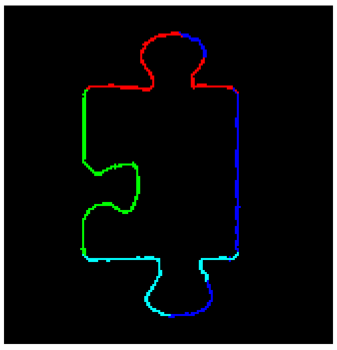
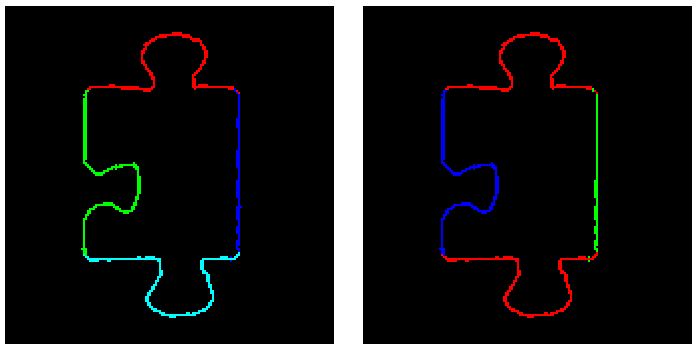
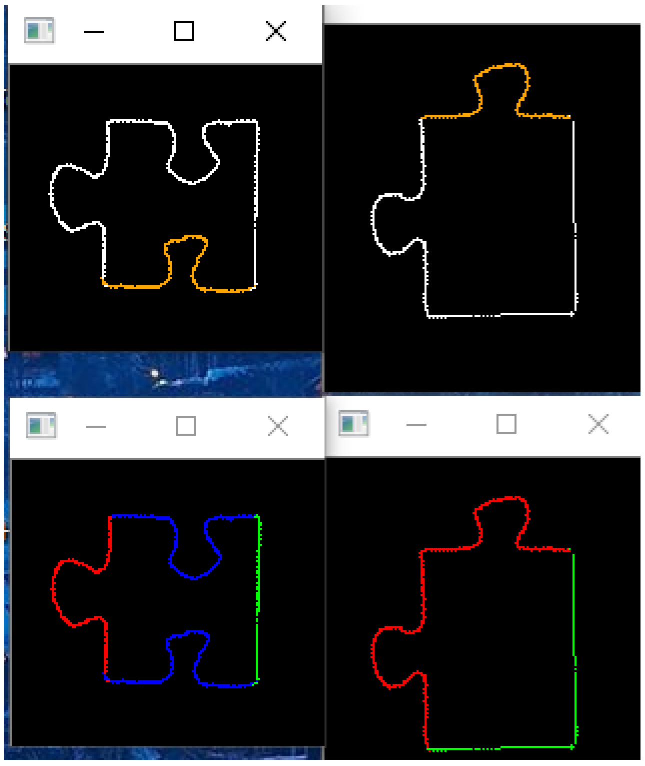
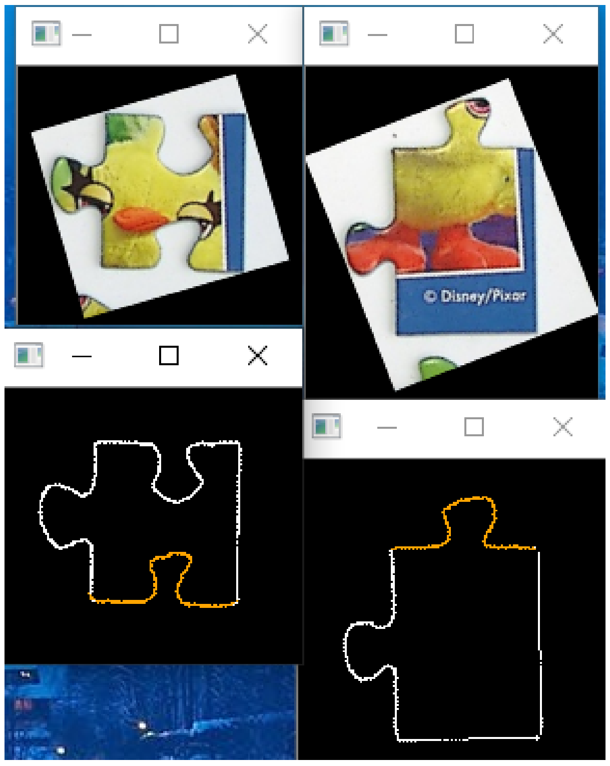


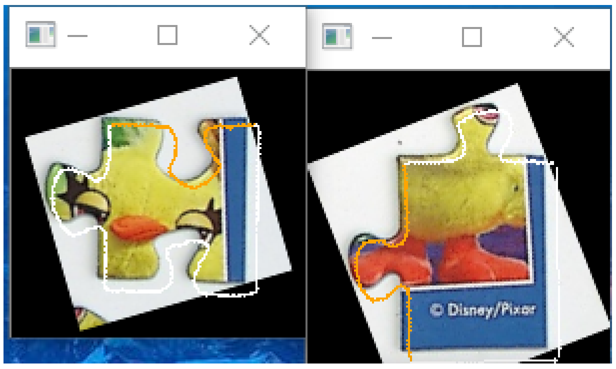
Image has some pixels off


Both classifiers after applying k nearest neighbours

Highlight and show user the matching sides



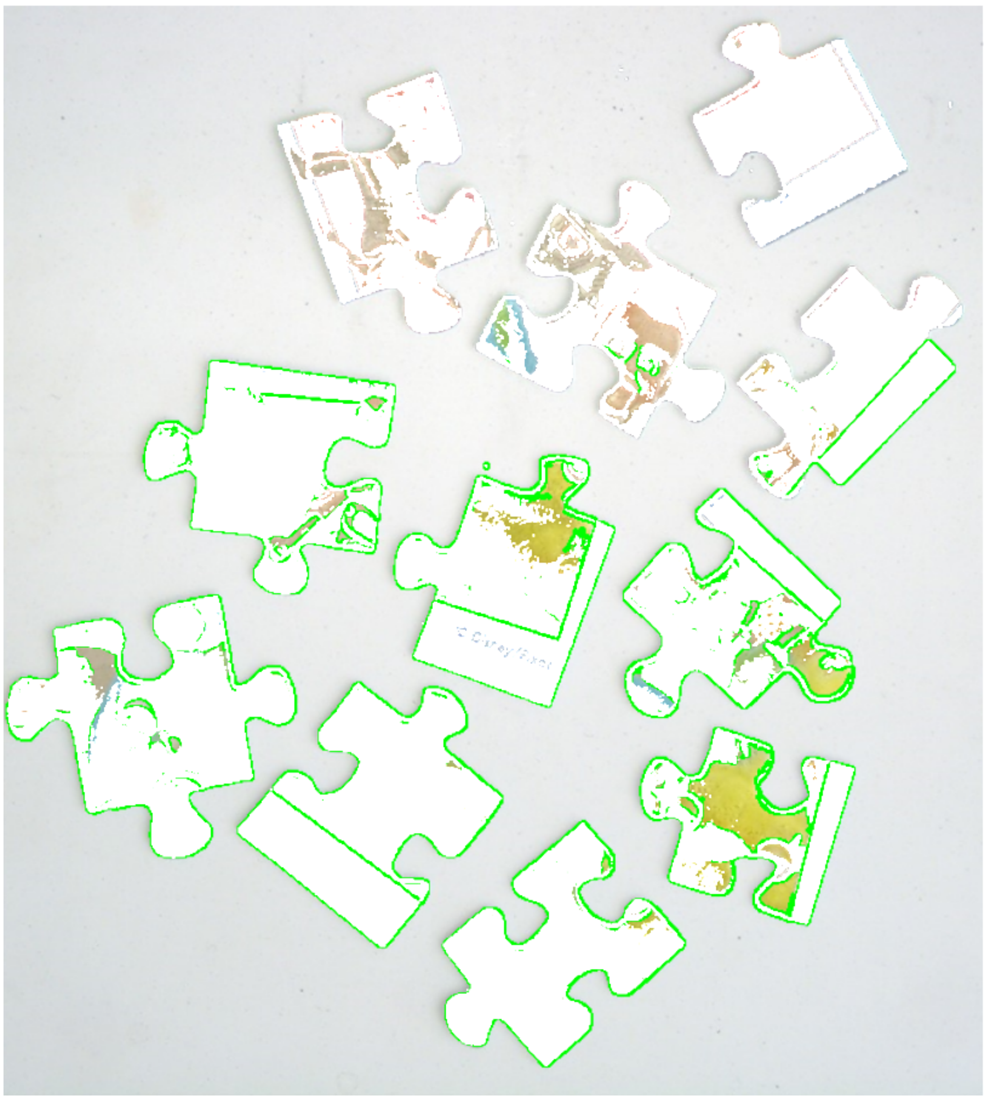




Appendix 2a - Add highlight match feature to Jigsaw image

Interesting problems that were solved

The initial masking of Jigsaw Pieces from the image



Some Classifier problems! Solved by sorting the four points in a clockwise rotation.