Jigsaw Puzzle Solving with a Robotic Arm and Computer Vision

Quentin Bishop, Vibhu Kundeti, Ethan Nguonly, and Suresh Patil
Thomas Jefferson High School for Science and Technology
Computer Systems Research Lab 2018-2019
qtip2001@gmail.com, vkundetis@gmail.com, ethannguonly@berkeley.edu, sureshspatil419@gmail.com

Abstract

In the past, computer vision algorithms and techniques have been used to computationally solve jigsaw puzzles (Allen, 2016). However, far fewer attempts have been made to implement these solutions physically in an integrated computational and hardware manner. This is in part due to the difficulty of solving a real representation of a jigsaw puzzle in software accurately enough to then be applied to a robotic component which has its own associated error. Although, this task is daunting given the level of accuracy needed to perfectly solve a jigsaw puzzle autonomously, the rewards of developing such a technology are immediately apparent. Namely, some applications of this technology could be applied to artifact restoration of ancient texts or paintings and a similar solution can be applied for medical procedures such as autonomous repair of broken bones or tissue through image analysis. Although these examples are beyond the scope of our paper, this research has tremendous potential outside of our concrete definition of the problem and is paramount to further developments concerning the physical implementation of a software solution to this problem.

Contents

1. Introduction

Puzzle solving has long been of interest to computer scientists who hope to find efficient and fully automated processes to solve puzzles. For example, Infineon, a German technology company, developed a robot which can solve a Rubik's cube in .637 seconds. Some other examples of these puzzles include the integer slider puzzle, and, the topic of our research, the jigsaw puzzle (Deamer, 2016). The jigsaw puzzle poses a unique challenge as it combines both software elements by requiring the use of computer vision to solve a 2D representation of a 3D puzzle and the hardware necessary to solve a jigsaw puzzle, namely a robotic arm. A similar problem was attempted such as the study by Stanford researchers Zanoci and Andress who were solving a digital representation of a jigsaw puzzle with all the pieces as perfect squares. However, our project expanded upon this concept by attempting to solve a physical jigsaw puzzle. Our approach was to first take a picture of an unsolved jigsaw puzzle and solve a digital representation of the puzzle in software before translating the necessary software generated movements into physical movements that would be implemented by a robotic arm. Through our research, we were able to create a robust algorithm for solving the jigsaw puzzle with a

digitally generated representation. However, our translation to robotic arm movements, although, theoretically as accurate as our software solution, was extremely sensitive to the initial conditions of setup and therefore was not as robust.

2. Methods

Our project consists of our physical materials used in our hardware setup, as well as our software which is structured into multiple processing stages.

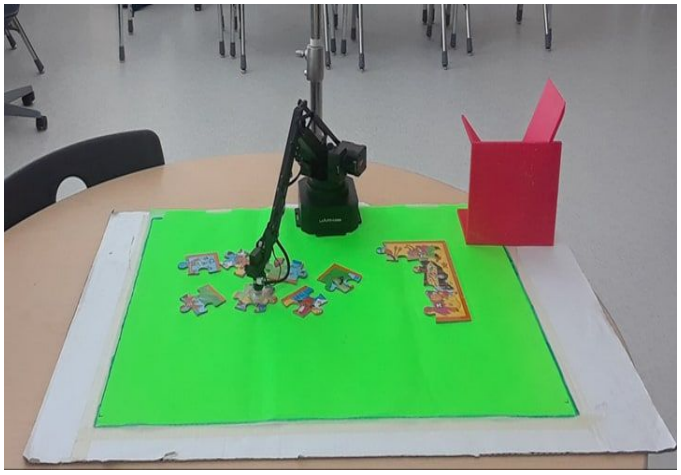

Figure 1. Overview of hardware setup

2.1. Materials

Our uArm Swift Pro has 4 degrees of freedom, a suction cup with a pump, and a range of 35 cm in a semicircle. The puzzle we used was a 3 x 4 pirate themed puzzle. Additionally, we have a puzzle piece flipping mechanism that we 3D printed, with a dual inclined plane. We also used a Samsung Galaxy S7 Edge for its high-resolution camera.

Figure 2. uArm Swift Pro

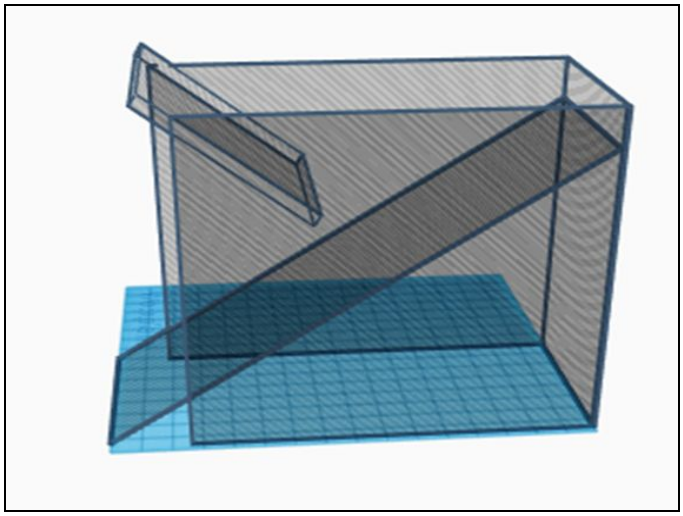

Figure 3. Double inclined plane piece flipping mechanism

2.2. Processing Stages Overview

We divided our project into three main stages which needed to work in unison in order to physically solve the puzzle: pre-processing, processing, and output.

The pre-processing stage is responsible for taking a photo and extracting it from the phone to the computer, preparing for the processing stage. Pre-processing starts with taking an overhead picture of the pieces on a fluorescent green background and retrieving the image to the main folder. A green border is then added around the image to cover the arm and any inconsistencies around the edge of the image to ensure a clean solve. The image is then thresholded to separate the puzzle pieces from the green background. The algorithm then performs contour detection and saves the piece contours in a separate file to send to the processing stage.

Figure 4. Image before pre-processing

Figure 5. Image after pre-processing

The subsequent processing stage classifies all of the pieces as either corner, edge, or center pieces. This classification is based on the number of straight sides on a piece (a corner will have two straight sides, an edge will have one straight side, and a center will have none). The algorithm then generates all of the permutations in which the pieces can fit together using a set of rules for puzzle solving (a corner piece will be next to two edge pieces, a piece with a head will fit into a piece with a hole). After generating all of these permutations, the solving algorithm runs through all of the permutations and calculates dissimilarity for each. The dissimilarity is based on the pixel color values of the two pieces along their adjacent sides. The dissimilarity is calculated using this formula.

$$\sum_{color=0}^{3} \sum_{i=0}^{len(PuzzleEdge)} (PuzzleEdge_1[i][color] - PuzzleEdge_2[i][color])^2$$

After calculating the dissimilarity for all of the permutations, the algorithm chooses the permutation that has the minimum dissimilarity. The coordinates of the piece in its initial location (pixel x and y in the initial image) and its final location (pixel x and y in the solved image) are sent to the output stage which displays a generated result.

The output stage then converts these pixel coordinates into a robotic arm input that can be sent directly to the robotic arm to pick up, rotate, and move the pieces to their final location. If a piece is detected to be upside down, the robotic arm will lift up the piece and send it to the piece flipper so that it can be placed right side up.

Figure 6. Computer generated image of a solved puzzle using our edge-pairing algorithm

The pre-processing and processing stages represent the complete solution to the digital solution of the jigsaw puzzle and the output represents a solution which includes the robot arm solving the physical jigsaw puzzle. The digital solution is currently more robust than the physical solution but further details concerning this observation will be discussed in the Results and Discussions section.

3. Puzzle Solver Technical Approach

Our technical approach included a SIFT Algorithm, edge-pairing, and an overhead camera with Android Development Bridge (ADB). Our initial approach was to use a Scale-Invariant-Feature-Transformation algorithm (SIFT).

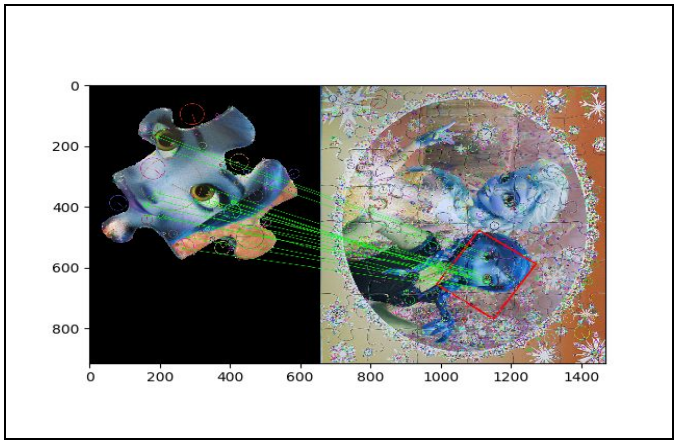

Figure 7. Individual piece matched to the solved puzzle using the SIFT algorithm

This yielded excellent results, being able to identify pieces based on key features with respect to the entire solved puzzle. Two qualities of SIFT that

lead us to look into this algorithm were that it is rotationally and scale invariant when matching. This means, using SIFT, we can match pieces to their position regardless of the individual piece's orientation and scale transformation that occurred during pre-processing.

3.1. Image Capture and Segmentation

The ADB image retrieval method works by using SuperUser authorization on the Android phone. The computer runs a python script that tells the camera to take a picture and then pulls all images from the phone's internal storage. After the ADB tool retrieves the images to our computer, it is then necessary to determine which image was last taken and accordingly the most current puzzle state. We used a *findMax* algorithm to see which image had the largest date-time stamp, accordingly identifying the current image. We then used image thresholding to exclude pixels in the image that were considered to be part of the background. This step was to ensure that the puzzle pieces could be identified, which is a crucial step before applying our solving algorithm to recognize their edges and determine the similarity between pieces.

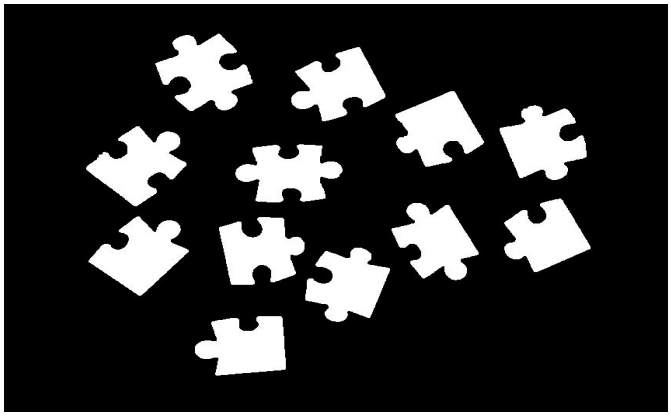

Figure 8. Removed background using image thresholding

The first step in this process was to apply a Gaussian blur to the image, this was necessary to inhibit sections of the puzzle pieces that shared the same color as the background from being removed along with the background. This is because a weighted average of surrounding pixels achieved on a puzzle piece is less likely to result in a representative NxN square of pixels on a puzzle piece matching the color of the background.

Once we have a thresholded image (Figure 8), we can distinguish the edges using canny edge detection because due to the stark contrast between the puzzle pieces (white) and the background (black) and the hysteresis values can be set at 127 (midpoint between 0-255) and for our practical purposes it would almost always be able to threshold and extract the edges from the picture. This accuracy is both promising and necessary for the subsequent steps toward our final goal.

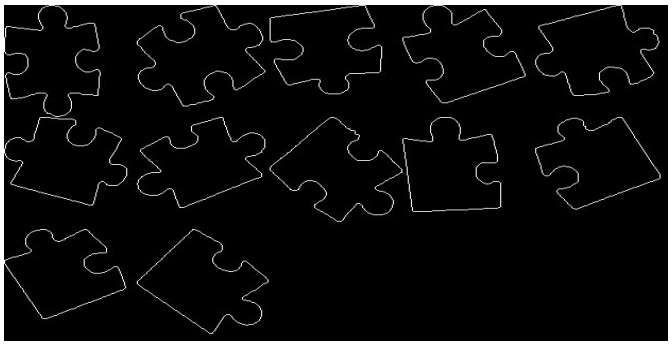

Figure 9. Identify locations of individual pieces and edges

3.2. Puzzle Piece Classification

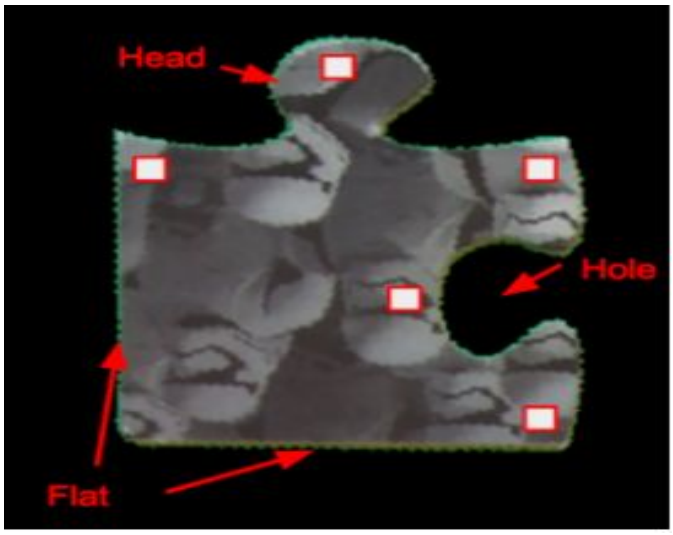

Figure 10. Classification of extrusions as heads, intrusions as holes, and straight edges as flat.

After obtaining the thresholded images with the extracted edges we tried to make sense of this visual data with the motivation of solving a jigsaw puzzle and applying this framework to our project. Therefore, we established some ground rules: pieces

would be classified as one of three types of pieces (corner pieces, edge pieces, and centerpieces). However, before we could determine what each piece was we had to analyze the specific edges and classify them as either heads (male connector), holes (female connector), or flats (no connection) of that piece and define the pieces accordingly (corners contain 2 flat and 2 connector edges, edges contain 1 flat and 3 connector edges, centers contain 4 connector edges) (Figure 9).

In order to achieve an environment that was conducive for solving we first detected the corners using Harris corner detection. However, we made the conditions for detecting corners more stringent by choosing only the top four corners that were closest to 90 degrees because otherwise, it would identify corners that were technically valid but would not be helpful in edge determination.

Once we had the corners we could draw a hypothetical line between two corner points with a certain degree of variance/error on either side. Then we would search all four edges, stopping at each corner, to determine if a pixel had left this imaginary bounding box. If a pixel does leave this bounding box and it is further away from the center of the piece than the corresponding point on the imaginary line then this edge is determined to be a head edge, if it is closer to the center than the corresponding point on this line then it is classified as hole, and if it never leaves the bounding box it is a flat edge. The variance of the bounding box was determined through trial-and-error. While these values vary by puzzle and camera setup, we found that a total variance of 12 pixels, six above and six below the line, gives us the best results for our setup.

Once we have classified each edge of every piece using this method we can then classify each whole piece using the rules explained earlier. This process provides us with the advantage of improving solving efficiency by greatly reducing our search space because we know that only corresponding head/hole pairs have any likelihood of being neighbors.

3.3. Edge-Pairing

Our edge-pairing algorithm is based on the approach Travis Allen of Stanford University used when developing a jigsaw puzzle solving algorithm (Allen, 2016). The underlying idea is that edges which are the most similar are probably ones that should fit together. The valid edge-pairs can now be determined due to the edge and piece classification that we attained in our previous section.

The algorithm first picks a corner piece at random to act as the anchor for solving the rest of the puzzle. Once this corner is picked a non-flat edge is chosen and compared against valid combinations of edges of the remainder of the pieces, dissimilarity calculations are made for each possible pair. This dissimilarity calculation is the squared difference of all the pixel values across the edge for all three given RGB values.

Once we do these dissimilarity calculations across all other edges compared to an initial target edge, the target edge is matched with another edge that shares the smallest dissimilarity value with the initial edge. Once we find this edge-pair we extract the puzzle piece from the initial image and transform it to “fit” into the corresponding edge on a solution image.

3.4. Translating Pixel Coordinates to Robotic Arm Coordinates

After the edge-pairing algorithm outputs the piece translation coordinates in pixels, the coordinates must be converted to a format that the robotic arm can use. The Swift UArm takes an x, y, and z input; since our puzzle is two dimensional and our solving all takes place in the XY plane, we only created two states for the z variable: on the board, when it is picking up or placing a piece, or 10 cm above the board, when the arm is carrying pieces from their initial to solved locations. As for the x

and y, we had to perform various scaling operations so that they could correspond to the x and y of the pixel output. The Swift UArm will accept any value for x and y, but the arm will only move if these coordinates are within range. We found limited documentation on the arm so we discovered the bounds for the x and y input through experimentation and measurement. We found that the movable range for y was approximately 0 to 300 while the range for x was approximately -350 to 350. We then measured the arm position and selected x and y values and found that the physical arm movement corresponded linearly with the x and y values inputted. With our observed measurement for arm movement and experimentation with input values, we generated linear equations that would input the x and y of the pixel coordinate of the piece and return the x and y for the robotic arm input.

3.5. Puzzle Piece Assembly

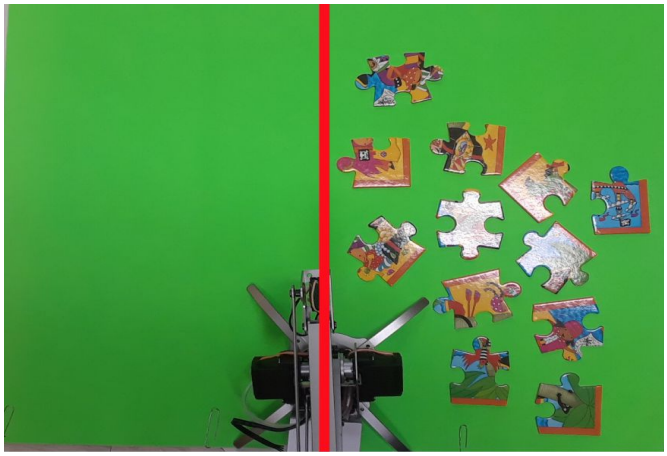

Figure 11. Partitioning of solving space into scrambled pieces (right) vs. assembly area (left)

3.5.1. First Corner

The solving algorithm starts by exporting the piece of the bottom left corner, which the robotic arm will lift and put into place in the lower left-hand side of the solving space. The rest of the solved puzzle is then placed relative to this corner piece so that all of the final positions of all other puzzle pieces will be in the range of the arm.

3.5.2 Direct Placement

The robotic arm then runs through the file of the solving algorithm output and lifts each piece from its initial position, performs the necessary piece rotation, and moves the piece to its final position. The robotic arm needs to pick up the piece in its exact center or else inaccuracies arise from the rotation.

3.5.3 . Final Assembly

After saving our intended robot arm coordinates to a text file, we then ran our robot arm program, which reads the text file to find the initial location, the final location, and the required rotation for any piece movement. The arm then moves above the initial location and enables the suction pump, the arm descends on the piece so that the suction pump forms a vacuum on the piece and lifts it back up.

4. Experimentation and Results

Throughout our research, we tested our work at various stages and experimented with changing approaches to see which methods were most accurate.

4.1 Digital Solving

Our initial edge-pairing solutions were generated in early December, where we were able to solve basic 4 piece puzzles with 100% accuracy. We increased our puzzle size to 12 and our final computational results were stellar. We were able to solve an image of any puzzle with our edge-pairing algorithm and generate an artificially solved image to show what the assembled puzzle should look like (see figure 5). Although our work was focused on solving a 12 piece puzzle, we've tested our solving algorithm on a 24 piece puzzle and found comparable success. From repeated testing and experimentation, we estimate our final solving algorithm's accuracy to be around 94%. The failures usually were a result of our piece classifier wrongly

predicting what category a piece should be, such as predicting that a corner piece is an edge piece. This miscategorization would result in failing to generate the proper permutations and would return another scrambled image of the puzzle. For these rare cases, a similar picture of the same puzzle configuration will usually fix any issues and result in a clean solution.

4.2 Physical Solving

We were unable to achieve optimal/full robot arm precision, which meant even the slight distortion from our images combined with the arm's low repeatability often resulted in inaccurate piece placement and less than perfect angle rotations. Our testing shows that the arm is not capable of moving pieces perfectly into place, even when given perfect coordinates. While puzzle pieces are generally placed in the correct general area, the pieces often do not fit perfectly into place. We're currently uncertain as to whether most of the error arises from small inaccuracies in our solving algorithm or if the arm's precision is to blame. Our current belief is that these two errors may be compounding to result in these larger solving inaccuracies.

Figure 12. Robotically solved puzzle

5. Conclusion and Future Work

Possible extensions of our research include implementing a moving cart in order to access larger puzzle spaces. Our experimentation was greatly limited by our arm's small 35 cm radius (uFactory, 2017). To avoid this complication, we could build or buy a much larger robot arm. Additionally, to reduce complexity and reduce the opportunity for error, we could eliminate the physical piece flipper and instead find an arm that has another degree of free in rotation, such as the newly released xArm, which has eight degrees of freedom. However, what would make the largest impact in terms of future work would be having a more accurate robot arm.

Puzzle solving is an extremely precise task that requires complete accuracy in order to place a piece directly into the grooves of another, especially with 100% repeatability. Additionally, we could benefit from a camera with even higher resolution as well as a better camera mount, as ours often swayed and would thus distort our images. To calculate for image distortion, we could also write a conversion system using the Pythagorean theorem, to calculate the true distance to a piece based on the angle the image is taken from.

Another opportunity for exploration is continued work implementing the SIFT algorithm. Our current algorithm emulates how a human might solve a jigsaw puzzle, we check if two edges match and place them together. Although this worked for the jigsaw puzzle we worked with, this would become exponentially slower for larger puzzles. Comparing edges between multiple times can grow to become very inefficient with larger puzzles and using an image matching algorithm like SIFT would allow for solving much larger puzzles without compromising speed, due to its O(n) nature.

6. Acknowledgments

We would like to thank our computer systems laboratory supervisors and mentors Peter Gabor and Mr. Patrick White. We would also like to acknowledge Mr. Paul Kosek, for his suggestions to use Android Development Bridge, Mr. Charles Dela

Cuesta, for allowing us to access his 3D printers, and Hitaansh Gaur, for his help on developing and 3D-printing our puzzle piece flipping mechanism.

References

Allen, Travis V. (2016). *Using Computer Vision to Solve Jigsaw Puzzles*. Stanford University, from web.stanford.edu/class/cs231a/prev_projects_2016/computer-vision-solve__1_.pdf

Deamer, K. (2016, November 11). New Record! Robot Solves Rubik's Cube in Less Than a Second.Retrievedfrom https://www.livescience.com/56828-robot-sets-rubiks-cube-world-record.html

Erell, N., & Nagar, R. (2012). *Robotic jigsaw puzzle solver.* Ben-Gurion University of the Negev, from cs.bgu.ac.il/~erelln/puzzle-solver/pre-2013-091.pdf

Nadasdi, B. (2016, February 25). Take a Photo via adb (Example). Retrieved May 23, 2019, from coderwall.com/p/3-tgjg/take-a-photo-via-adb

Zanoci, C. (2016). Making Puzzles Less Puzzling : An Automatic Jigsaw Puzzle Solver.

uFactory. (2017). *uArm Robotic Arm* Available from https://store.ufactory.cc/products/uarm