Jigsaw Puzzle Solver Robot

Yu-Peng Hsieh, Li-Kai Chuang, Hao-Fang Cheng, Yin-Li Liu
B.S. in Electrical Engineering at National Taiwan University
B.S. in Mechanical Engineering at National Taiwan University
Advisor: Prof. Li-Chen Fu

1 Introduction

Traced back to 1964, the first algorithm describing the development of a procedure that enables a digital computer to solve *apictorial* jigsaw puzzles, i.e., puzzles in which all pieces are uniformly gray and the only available information is the shape of the pieces was proposed by Freeman and Garder [1], which successfully solved a 9-pieced jigsaw puzzle with visual information only. Since then, although numerous papers had been written about automatically solving jigsaw puzzles, there were still no published algorithms that can solve large puzzles reliably and efficiently. In [2], the algorithm proposed can handle puzzles only by shape, combining the overall strategy of previous algorithms and new techniques, such as robust fiducial points, "highest-confidence-first" search, and frequent global reoptimization of partial solutions. Other than using shapes, Sholomon et al. [3] proposed the genetic algorithm (GA)-based jigsaw puzzle solver, with a procedure of merging two "parent" solutions to an improved "child" solution by detecting, extracting, and combining correctly assembled puzzle segments, achieving faster and more accurate result than ever before. Other studies combine shape and picture features of puzzles [4][5].

As for combining puzzle-solving algorithm and robot, in 1989, an integrated vision-manipulation algorithm for robot to solve *apictorial* jigsaw puzzles was presented [6]. From Fig. 1, we can see that the robot has a force feedback so that it can slide a puzzle on top of another and terminate when the contact is lost, i.e., the edges are perfectly matched. *Solving Jigsaw Puzzles Using a Robot* [7], a book written by Burdea et al., introduced a few robotic fine assembly techniques, which are used to verify solutions from algorithms and feedback partial results. Another interesting

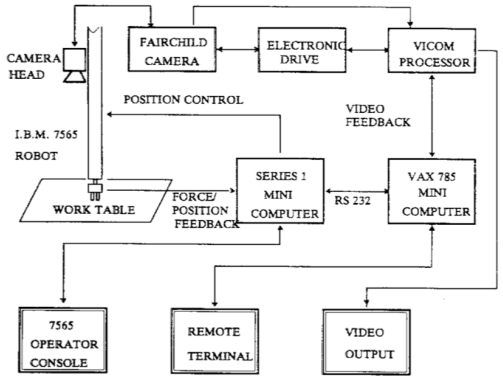

Figure 1: Experimental installation for puzzle assembly (from [6])

study used an anthropomorphic robot to learn how to solve a children's jigsaw puzzle together with a human tutor [8], and instead of choosing a straight forward engineer's approach, they decided to combine a psychologist's and a computer scientist's views.

In this report, we use our own jigsaw-puzzle-solving algorithm, design a gadget to suck up the puzzles, and command TM5-900 robot arm to assembly the puzzles.

2 Materials and Methods

The materials and methods are divided into four sections, including the puzzle-solving algorithm, the mechanical systems design, Camera Calibration, and robotic arm control. The whole project is uploaded to Github.

2.1 Puzzle-solving Algorithm

As we know about the jigsaw puzzle game, people need to reassemble the puzzle pieces back to a complete image, with the clue of image and the shape of the pieces. Therefore, the puzzle solving algorithm need to extract the pieces form the

camera photo, and detect the image and contour of each piece. The algorithm can be separated into two parts: pieces detection and puzzle solving.

2.1.1 Piece Detection

For the piece detection, first of all, we need to separate the background and the puzzle pieces. After converting the color mode of the image form RGB to HSV, we calculate the mode of each channels and set them as the background with some tolerances. Therefore, we can extract the pieces out of the original camera photo, as Fig. **??** shown below.

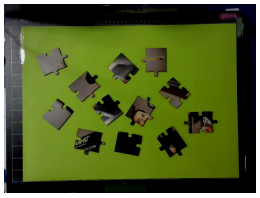

Figure 2: original figure

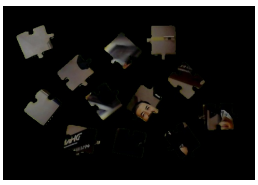

Figure 3: background removed

In order to get the center coordinate of each pieces, the next step is to detect the four corners of each pieces. We use the *findContour()* function in OpenCV to get all the possible contours, screening them with an area filter to pick up whose area is between 1% to 2% of the original image. Then we use the *MinAreaRect()* function in OpenCV to calculate the bounding boxes of all the contour candidates. However, the corners of the bounding box of whole contour is not the actual corners of each pieces. The contour might contain indent or outdent padding edges, which will cause bigger bounding box (Fig. 4). To get the actual bounding box of the piece, we design an algorithm to detect the 4 corners from the piece's contour, described in Algorithm 1.

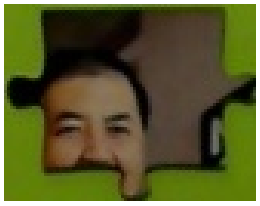

Figure 4: bounding box with indent padding edges and outdent padding edges

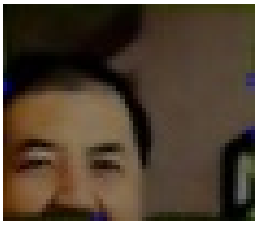

Figure 5: bounding box of actual corner

Algorithm 1 CORNER DETECTION

for $p \in \mathcal{P}$ **do**
  initialize UP=[],DOWN=[],LEFT=[],RIGHT=[]
  for $c \in Contour$ **do**
    if $c.x - boxUpperEdge.x < threshold$ **then**
      UP.push(p)
    end
    if $boxLowerEdge.x - c.x < threshold$ **then**
      DOWN.push(p)
    end
    if $c.y - boxLeftEdge.y < threshold$ **then**
      LEFT.push(p)
    end
    if $boxRightEdge.y - c.y < threshold$ **then**
      RIGHT.push(p)
    end
  end
  $upLeft \leftarrow$ point in UP with min y
  $upRight \leftarrow$ point in UP with max y
  $downLeft \leftarrow$ point in DOWN with min y
  $downRight \leftarrow$ point in DOWN with max y
end

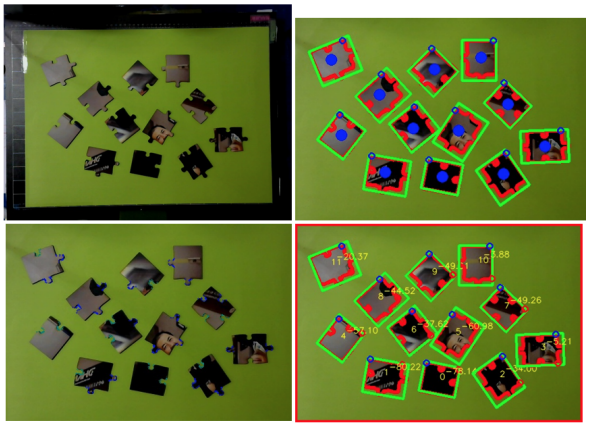

Figure 6: Webcam photo (upper left). Contour, corner and bounding boxes detected by the algorithm (upper right). Edge classification (bottom left). Final poses of pieces (bottom right)

To make the puzzle solving more easily, we want to classify the pieces' edges into three types: indent padding edge, outdent padding edge, and flat edge. The Algorithm first separates the points in contour into four classes: up, down, left, and right. For each edge, check if there exists points that are too far away from the corner's coordinate,

if exists, set the edge type to indent or outdent padding edge. Otherwise set the edge type as flat. Detailed steps are described in Algorithm 2. After following the steps above, the results are shown in order in Fig. 6.

Algorithm 2 EDGE CLASSIFICATION

for $p \in \mathcal{P}ieces$ **do**
  $x_{up} \leftarrow (upLeft.x + upRight.x)/2$
  $x_{down} \leftarrow (downLeft.x + downRight.x)/2$
  $y_{left} \leftarrow (upLeft.x + downLeft.x)/2$
  $y_{right} \leftarrow (upRight.x + downRight.x)/2$
  if $(x_{up} - minXInUP) > threshold$ **then**
    $edgeType_{up} \leftarrow outdent$
  end
  if $(maxXInUP - x_{up}) > threshold$ **then**
    $edgeType_{up} \leftarrow indent$
  end
  if $(maxXInDOWN - x_{down}) > threshold$ **then**
    $edgeType_{down} \leftarrow outdent$
  end
  if $(x_{down} - minXInDOWN) > threshold$ **then**
    $edgeType_{down} \leftarrow indent$
  end
  if $(y_{left} - minYInLEFT) > threshold$ **then**
    $edgeType_{left} \leftarrow outdent$
  end
  if $(maxYInLEFT - y_{left}) > threshold$ **then**
    $edgeType_{left} \leftarrow indent$
  end
  if $(maxYInRIGHT - y_{right}) > threshold$ **then**
    $edgeType_{right} \leftarrow outdent$
  end
  if $(y_{right} - minYInRIGHT) > threshold$ **then**
    $edgeType_{right} \leftarrow indent$
  end
end

2.1.2 Puzzle Solving

We use the *MatchTemplete()* function in OpenCV to find to which part of the original image each piece belongs, choosing the location that the similarity is highest. The types of the edges are used to double check if the matching results are correct. Because of the angles of the pieces are unknown, we match the image 4 times, each time the piece would be rotated by 90°, and we choose the rotation angle with the highest matching score as the final pose. The solving results are shown in Fig.7 to Fig. 10. After the puzzles are solved, the poses, angles and target positions of the puzzles are sent to the robot for further assembly.

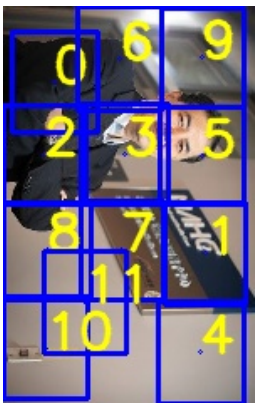

Figure 7: matching result for classical puzzle pieces

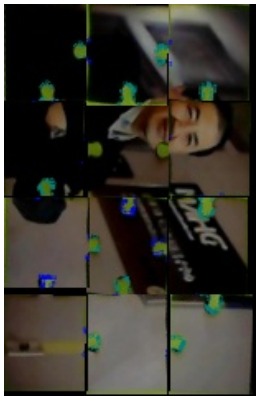

Figure 8: reconstruction image for classical puzzle pieces

Figure 9: matching result for square puzzle pieces

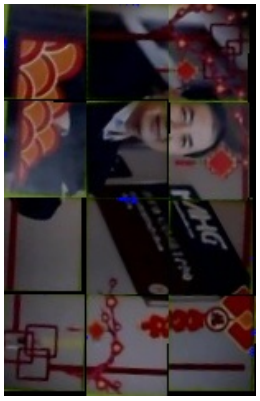

Figure 10: reconstruction image for square puzzle pieces

2.2 Mechanical Systems Design

We design a pneumatic system to pick up our puzzle, which is based on a pump and a sucker. Pump is one of the most famous machine to negative pressure environment. We choose a mini DC pump of -400 mmHg max vacuum in our system. Its max vacuum is proportional to the voltage we give. In addition to overcoming the gravity of the puzzle, we need to consider about the friction resistance along the three meters tube. This pump can meet our demand. As for the sucker, we use the industrial metal tube with a spring in it to make some clearance while sucking. A 3D-printed

adapter is made to connect the robot arm's gripper and the sucker. The sucker head is changeable so we can use different size during sucking and calibrating.

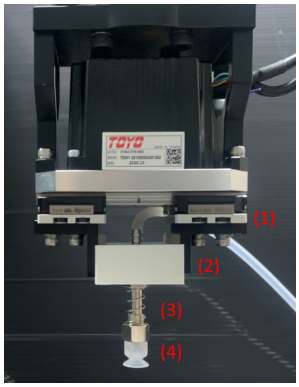

Figure 11: sucking tool, (1)gripper (2)adapter (3)metal tube (4)sucker head

The Electromechanical systems is shown in Fig. 12. First, Arduino uno board, a kind of micro controller, will receive a serial message from the computer, telling it to suck or to release the puzzle. Then, it will send a digital signal to the relay which controls the air direction in the valve. There are some voltage issues in the digital signal, so we add an additional relay on our system.

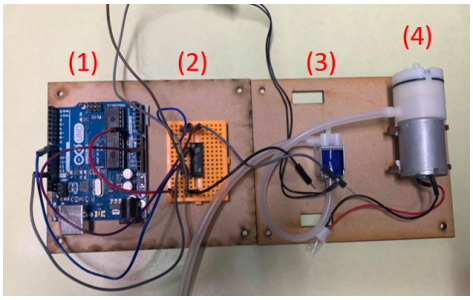

Figure 12: Electromechanical systems, (1)uno board(2)relay(3)valve(4)pump

The system can't directly put down the puzzles very precisely. Therefore, we design two suitable sets, one is the puzzles with 3mm clearance, and the other is puzzles with shapes of rectangle. These two kinds are shown in Fig. 13. To deal with the rectangle pieces, we also develop a gravity-based frame, which can solve the precision problem by making each piece slide belong the frame edge. The frame is shown in Fig. 14.

Figure 13: Puzzle sets, the left is clearance set and the right is square set.

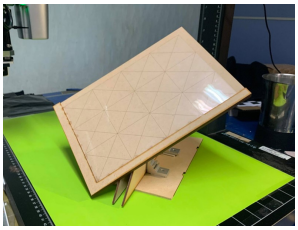

Figure 14: gravity-based frame

2.3 Camera Calibration

A robot needs to first have a sight on its target, then it takes action. In our task, we use a webcam isolated from the robotic arm to shoot a picture of the scattered pieces. Hence, a relationship between the vision and arm needs to be determined, called Camera Calibration. The next step, the intrinsic matrix and the extrinsic matrix are to be determined. The equation of Camera Calibration is shown below.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{11} & r_{13} & t_1 \\ r_{21} & r_{21} & r_{23} & t_2 \\ r_{31} & r_{31} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where (u, v) refers to as the coordinates of a 2D image, $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} r_{11} & r_{11} & r_{13} & t_1 \\ r_{21} & r_{21} & r_{23} & t_2 \\ r_{31} & r_{31} & r_{33} & t_3 \end{bmatrix}$ respectively determine the intrinsic (optical centre, scaling) and extrinsic (camera rotation and translation) properties, and (X, Y, Z) refers to as the coordinates in the 3D world.

For the intrinsic matrix, we take 20 pictures of a calibration board and make use of the library in OpenCV to find out. The extrinsic matrix is the tricky one. We let the arm go to a specified point and leave a mark. Then, we shoot a picture of it and figure out the position of the mark in the image. This way, we have a set of calibration materials, including a world point (mm) and an image point (pixel). After obtaining several sets, say 6 sets of points, we calculate the pseudo inverse to

find out the approximate solution best fitting the 6 sets of points. After obtaining both intrinsic and extrinsic matrix, we're able to transform an arbitrary point between world coordinates and image coordinates using the matrix quality (99). As a result, the accuracy is good enough in a normal task. We can transform a point from the image into world coordinate with an error of less than 1 mm.

2.4 Robotic Arm Control

We use TM5-900 as platform, a robotic arm with a reaching distance of 900 mm, to do the manipulation task. In the whole process of manipulating the robotic arm, we do not need to deeply consider all the things in the control theory, such as the rising time or the feedback of force and position, etc. The only important thing is that we have to carefully design a series of point through which the end-effector passes in order to achieve our goal. The flow graph is shown in Figure 15.

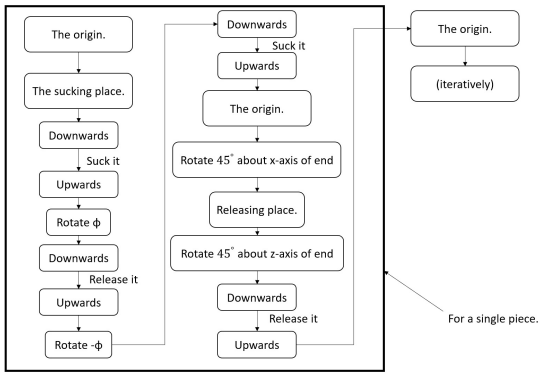

Figure 15: Flow graph of a series of point design

A point to note is that we add a "Suck, release, and suck" procedure in the workflow. Some may be confused about why we need to do a seemingly unnecessary action. However, it's the important move in our mission. Since our frame is set up with a tilted angle for the gravity assisting purpose, the pose of the robotic arm will end up in a tilted angle when solving puzzle, too. At that pose, the arm is hard to rotate its last joint, and sometimes even collides with itself. A solution to it would be rotating the last joint at the other timing, which in out task is the very first moment before sucking up a piece. As a result, we correct the orientation of the pieces beforehand on the desk (Suck, correct, release, and suck), so that when the arm is at a pose with a tilted angle, it doesn't need to rotate anymore. The colliding issue is something we had a hard time dealing with. After coming up with such a solution, to our delight, everything goes smoothly.

When controlling the robot arm, we send point information to the server. In our task, "Move_Line(TPP)" and "Move_Line(CPP)" function calls are used. We're using "Move_Line" instead of "Move_PTP" because we need to make sure the trajectory of the moving arm is a straight line in order not to collide with any other obstacles surrounding it. As for the two types of function, we use TPP and CPP, which respectively play important roles. The TPP type tells the arm to change its pose with respect to the coordinate of the current end-effector state, whereas when the arm receives information of a CPP type, it is driven with respect to the base coordinate of the robot arm. With a carefully designed series of point in two different mode TPP and CPP, we're able to transform the position and orientation of the end-effector to whatever we want it to be. This is the very interesting part for us because we get to utilize what we've learned from the Robotics lecture into practice. In our task, the coordinate transformation involves a 3-dimensional process, which is challenging and is a good opportunity to perform the knowledge we've got from the professor.

3 Results

The robot can successfully finish solving and assembling 12 puzzle pieces under 111 seconds, in which the algorithm takes about 0.25 second to solve and the robotic arm takes about 110 seconds to move. The demo videos of solving a rectangle-shaped puzzle and a normal puzzle are respectively provided in Video 1, and Video 2.

4 Conclusions and Future Work

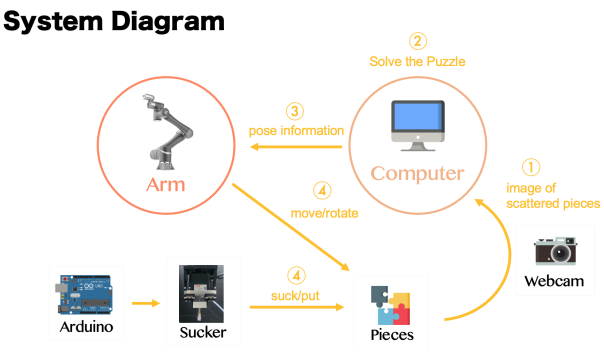

Figure 16: System diagram

As the system diagram shown in Fig. 16, in this project, we make the robot to solve puzzles. Throughout the process, we design a puzzle-solving algorithm mainly based on pictures and assisted by edge shapes, a sucker gadget controlled by Arduino Uno connected with pump, Camera Calibration according to Robotics lectures, and robot arm control on the TM5-900 platform. The robot solve the puzzles efficiently and successfully. Nevertheless, occasionally due to imbalanced lighting, minor errors may happen. Since our algorithm is mainly based on figures on the puzzles, distortions on brightness cause costs in accuracy. Therefore, if the robot arm is established in a more stable place, the results will become more robust. Moreover, our algorithm is currently supervised, which means that it requires a complete picture of the puzzle to solve, whereas there are some unsupervised algorithms that we can try out in the future to make the robot more intelligent. Last but not least, we can further add a force or a friction feedback to the robot arm as described in [6], so that it can assembly puzzle pieces more precisely and without the need of clearance margin.

References

[1] H. Freeman and L. Garder. "Apictorial Jigsaw Puzzles: The Computer Solution of a Problem in Pattern Recognition". In: *IEEE Transactions on Electronic Computers* EC-13.2 (1964), pp. 118–127. DOI: 10.1109/PGEC.1964.263781.

[2] David Goldberg, Christopher Malon, and Marshall Bern. "A global approach to automatic solution of jigsaw puzzles". In: *Comput. Geom.* 28 (Jan. 2004), pp. 165–174. DOI: 10.1145/513400.513410.

[3] Dror Sholomon, Eli David, and Nathan Netanyahu. "A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles". In: June 2013. DOI: 10.1109/CVPR.2013.231.

[4] Michail Makridis, Nikos Papamarkos, and Christodoulos Chamzas. "An Innovative Algorithm for Solving Jigsaw Puzzles Using Geometrical and Color Features". In: vol. 3773. Nov. 2005, pp. 966–976. ISBN: 978-3-540-29850-2. DOI: 10.1007/11578079_99.

[5] Kawaboongawa. *Kawaboongawa/Zolver*. URL: https://github.com/Kawaboongawa/Zolver.

[6] B. G. Burdea and H. J. Wolfson. "Solving jigsaw puzzles by a robot". In: *IEEE Transactions on Robotics and Automation* 5.6 (1989), pp. 752–764. DOI: 10.1109/70.88097.

[7] B. G. Burdea and H. J. Wolfson. "Solving jigsaw puzzles by a robot". In: *IEEE Transactions on Robotics and Automation* 5.6 (1989), pp. 752–764. DOI: 10.1109/70.88097.

[8] Catherina Burghart, Christian Gaertner, and Heinz Woern. "Cooperative Solving of a Children's Jigsaw Puzzle between Human and Robot: First Results". In: *AAAI Workshop - Technical Report* (Jan. 2006).