Pentominoes Puzzle solving with Kuka Miiwa

Diogo Marques[1][83631] and Marco Silva[2][84770]

[1] University of Aveiro, Aveiro, PT d.marques@ua.pt
[2] University of Aveiro, Aveiro, PT masilva@ua.pt

Abstract. This document approaches the implementation of an algorithm to solve a Puzzle constituted by Pentominoes, that is, pieces made with 5 cubes each. The solution is split in multiple stages, starting by localizing the play area and where the pieces are, followed by classifying each piece and its grab point and, finally, the execution of the puzzle solution.

Keywords: Intelligent robotics · Computer Vision · Kuka Miiwa · ROS

1 Introduction

A pentomino is a object made of 5 equal-sized cubes connected edge-to-edge. When rotations and reflections are not considered to be distinct shapes, there are 12 different free pentominoes. When reflections are considered distinct, there are 18 one-sided pentominoes. In this document the goal is to demonstrate how a solver is implemented in the Kuka Miiwa robot, so that it can assembly a puzzle using the available pentominoes that the it detects on the table using its cameras. The solution is approached with 3 different phases: location of pieces and play frame on the table, recognition and categorization of each piece and, finally, assembly the puzzle based on the given solution. Some changes were made to the interface used to control the arm, as well as, some other constraints that are approached in more detail in section 3.

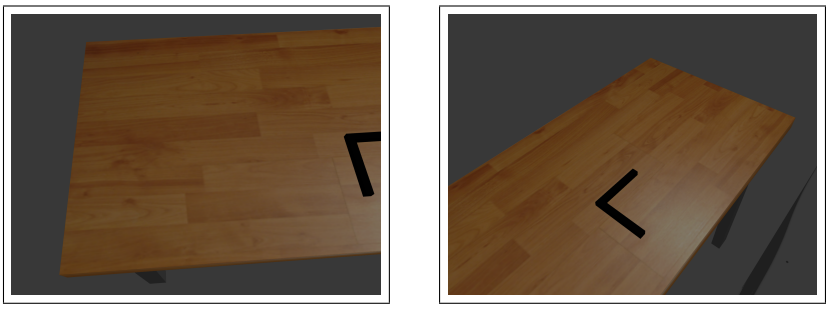
2 Implementation

The implementation was divided into 3 distinct phases in order to make the solution more modular, so it is easier to make changes in specific parts without interfering with the others, making testing also faster. The three phases are the location of the pieces and the play frame on the table, after we have the position of all the pieces we move on to categorizing them, where we get the type of piece and a pose to grab it. After this, the third phase consists of reading a solution to the puzzle and completing it.

2.1 Phase 1 - Location of pieces and play frame on the table

The first step to solve the puzzle is to know where the pieces are located on the table. To accomplish this, the templates from Figure 1 are required. Initially

the Pan Tilt camera is set to position so that the image captured by the camera matches one of the templates. After getting the image from the camera, the template is subtracted from it which allows for an easier way of detecting the pieces on the table. This step of detecting the localization is split in two since the camera can't capture the whole table in a single shot. Because of that, the camera is set to match firstly the right template, Figure 1b, and then it will look for pieces on the other side of the table, moving to the position that match the left template, Figure 1a. One should note that, given the camera position on the base frame, the detection of pieces that are very close will only count as one piece. One example of this situation is present in Figure 3, where the F and N pentominoes are detected as one. This problem is solved, later, using the TCP camera on the arm, that will capture a top view image of a given point. All the points obtained are just pixels in the captured image. In order to obtain 3D world coordinates relative to the robot base frame, initially this pixel coordinates are converted to 3D coordinates relative to the Pan Tilt frame and, after that, transformed to the base frame.

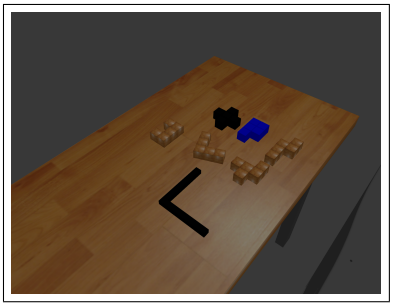


(a) Table Left Template (b) Table Right Template

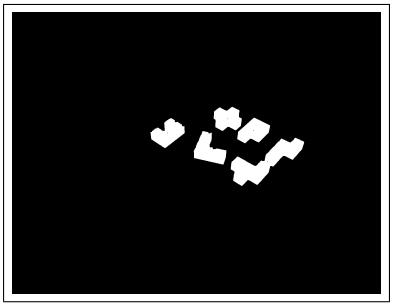
Fig. 1: Templates required to detect pieces location

After detecting where the pieces are located on the right side of the table and before moving the camera to match the left side template, it is necessary to locate where the play area will be, that is, where the final puzzle assembly will be done, since that changing the camera orientation will change it's Pinhole Camera Model and so, the conversion to 3D coordinates would be wrong. To obtain this, the template from Figure 1b is used. With this image, and the appropriate OpenCV[1] filters, the black L can be isolated from the rest of the image and one can determine the location of its inner corner. Again, this coordinates are just related to the image so they need to be converted to 3D coordinates and then transformed to the robot base frame. With this point, and as is explained in subsection 2.3, one can obtain each piece drop position in the play area.

Finally, with the TCP camera attached to the arm, a top view image, like the one in Figure 3a, from each location, as said previously, is captured. Using

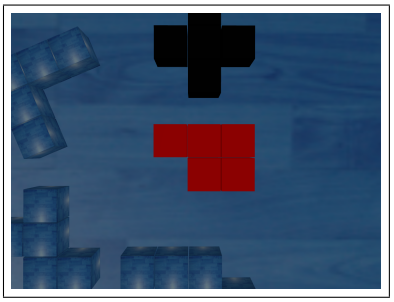


(a) Image from Pan Tilt camera

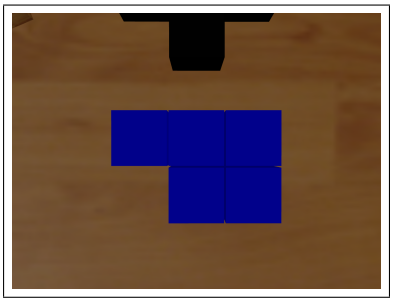


(b) Result after OpenCV filters (F and N pieces are considered one)

Fig. 2: Capture and Result from Pan Tilt piece location



(a) Top View image from TCP Camera



(b) Closer view image

Fig. 3: Capture and Result from Pan Tilt piece location

the openCV[1] library the minimum rectangle that contains each detected piece contour in the top view image is obtained, and its center is saved. This point is, then, converted to 3D world coordinates related to the robot base frame and is used in phase 2, subsection 2.2, to capture a image like the one from Figure 3b with the TCP camera so that each piece can be classified. Given that the top view image will most of the times contain many pieces, after saving every center point for every piece in all the top view images, this group of saved points is filtered in order to remove the closest ones, that is, the ones that belong to the same piece.

2.2 Phase 2 - Recognize and categorize each piece

After knowing the position of each piece on the table, we can proceed to its categorization. In this process the robot arm goes to each previously collected point, but this time with a default height in order to capture an image where the piece always has the same dimensions. If the planning of the movement to the desired position results in an error, other angles are tested until success is achieved or it is concluded that the arm cannot move to this position. In the last case the robot will continue with the recognition of the remaining pieces.

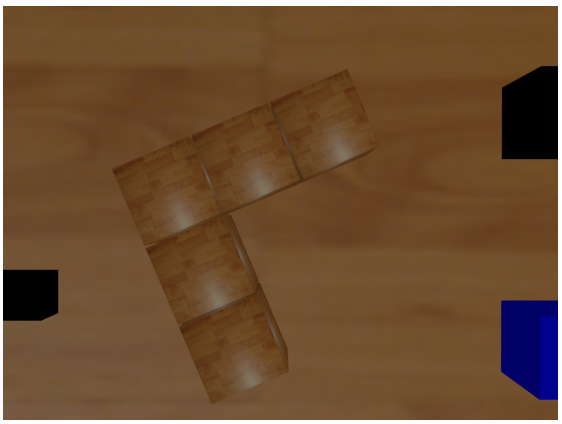


Fig. 4: V-piece image taken from the TCP camera

After obtaining the image centered on the piece, as in Figure 4, a recognition algorithm is used to categorize the piece and also obtain the point and angle at which it should be grasped when performing the puzzle. This algorithm is mainly based on the openCV[1] library and where filters are first applied so that it is easier to get the contours of the piece and then only the points of the vertices of the piece. After having only the points of the contours that correspond to the vertices, they are compared with some attributes of the pieces' templates and from the number of vertices, the perimeter of the piece and the area of the

minimum rectangle that contains the points, it is possible to categorize the piece. After knowing the type of the piece and its unique characteristics, the rotation angle of the piece and the gripping point are calculated. For this whole process a default template of the piece, like the one in Figure 5, and the transformation vector with reference to the center of the piece for the grab point are needed. The algorithm is able to distinguish 7 pentominoes (N, F, P, L, X, U and L), to recognize more would require adding the template, the vector for the point to grab and a section of code to detect the correct angle of the piece. Note that if the robot is unable to categorize the piece in the current pose the robot will rotate as previously stated. As in the previous phase, subsection 2.1, this points and angles are just related to the image so they need to be converted to 3D coordinates, then transformed to the robot base frame and do the same with the angle.

At the end of this process we already have the information about the pieces on the table and where to grab them, and can proceed to the puzzle solving phase.

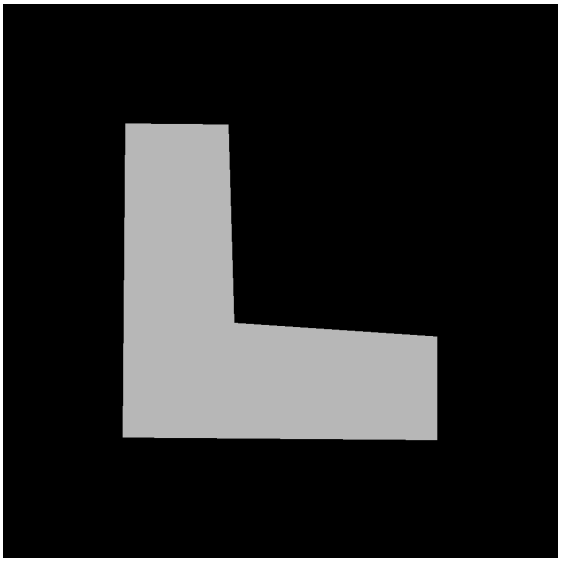


Fig. 5: V-piece template

2.3 Phase 3 - Grab the pieces and solve the puzzle

In the last phase the robot has the job of solving the puzzle with the solution provided, with the format shown in the Figure 6. The first step at this stage will be to take the solution file and extract the position and rotation of the pieces

from it. For this process it is also necessary to have a template text file that corresponds to each piece and which cube is grabbed by the robot.

With the position where the pieces need to be in the solution and the information taken in the other phases the robot has everything to solve the puzzle. The robot would move to the position of each piece and place it in the correct pose and grab the pentomino. With the piece grasped, it now goes to the position calculated from the solution and the position of the game frame and drop the piece. Does this for all the pieces according to their distance to the frame, and at the end, consider the puzzle finished. One should note that the gripper sometimes doesn't attach successfully the piece and so the arm will consider the puzzle finished even though one or more pieces weren't grabbed with success.

3 Constraints that need to be met

In order to the arm assemble the puzzle some constraints were defined *a priori* which will be explained in this section.

There are two static templates of the table, used as explained in subsection 2.1, and for that reason, any movement of the table will result in a bad detection of the pieces location, since the template without the pieces won't match the image from the Pan Tilt camera.

The arm is attached to a base platform that is static. Given the table size, the arm won't be able to reach every corner of table, mostly the left side of it, and so, for a successful assembly all the pieces must be on the right side of the table.

There is a play area were the puzzle will be assembled. According to the provided solution, this area can be longer vertically, horizontally or have the same length horizontally and vertically. With this in mind, the arrangement of the pieces must be done in order that none of them are inside of this play area.

There are two templates for each piece obtained *a priori*, one in an image format and one in a text format that provides which cube is to be grabbed with the arm with an angle of 0 radians. This text template is used, as well, to determine the final rotation of a piece.

The pieces are considered to be on the same side as the final solution, that is, there are no pieces flipped in any form. However their orientation is not important.

All the pieces are separated enough so that the top view camera can clearly isolate each one.

Finally the last constraint is that the solution of the puzzle is given in a text file, with a matrix format as in Figure 6.

4 Results

With the three phases implemented the robot is able to solve any puzzle that meets the constraints mentioned in section 3. The solution of the puzzle can use

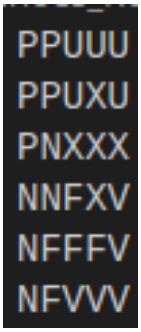
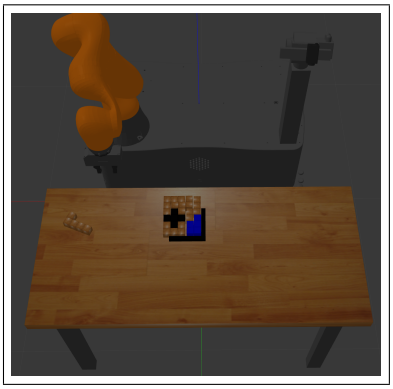
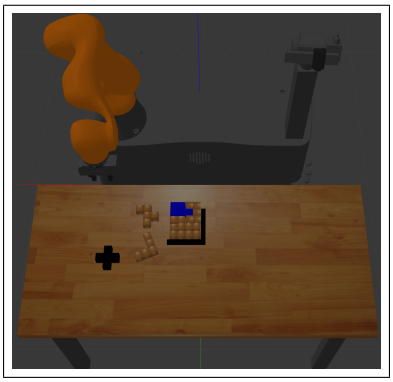


Fig. 6: Solution of a puzzle

a subset of the pieces present on the table, with the robot categorizing them all. So several puzzles with varying dimensions were tested, for example, 5x4 using 4 pieces and 6x5 using 6 pieces demonstrating the capabilities of the robot. The main reason the robot fails to build the puzzle even though the constraints are met is because it cannot grasp the piece even though it is between the handles. In Figures 7a and 7b it is possible to observe the puzzle solved by the robot with the sizes mentioned before and using different pieces.



(a) 4 Pieces Solution



(b) 6 Pieces Solution

Fig. 7: Puzzle assembly results

4.1 Notes to consider

The arm_interface.cpp file that allows one to send instructions to the arm joints has a little bug that was corrected using the IF condition shown in Figure 8.

This condition prevents the access to an empty vector when the desired pose can't be calculated.

The *move_msg.request.position* value in the gripper_interface.cpp GripperInterface::grasp function was changed from 0.0395 to 0.03515. After many simulations, it was concluded that the movement of attaching a piece was more successful with this value.

Lastly, manually changing the pieces position in the simulator, that is, grabbing a piece and putting it in other position, may cause interferences and with the other pieces and the solver may not be able to grab a piece and solve the entire puzzle, mainly because of the grasp failures as well. A good example of interferences caused by moving pieces is shown in Figure 9, where some pieces are tilt (like the N or U pentominoes) just because other were moved.

```
206     if(gik.response.joint_position.values.size() <= 0){
207         ROS_ERROR("ArmInterface: Joint Position Values Vector size: %d", gik.response.joint_position.values.size());
208         return false;
209     }
210     /* Now move the arm */
211     // adjust the gripper offset directly in the joint
212     gik.response.joint_position.values[6] -= M_PI*0.25;
213     gik.response.joint_position.values[6] = angles::normalize_angle(gik.response.joint_position.values[6]);
214     return moveJoints(gik.response.joint_position.values, velocity, blocking);
```

Fig. 8: If Condition that corrects the bug in arm_interface.cpp

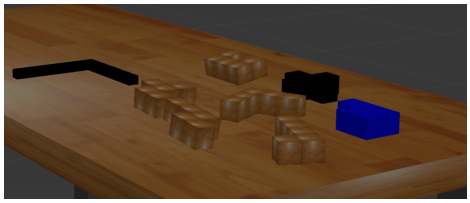


Fig. 9: Tilt pieces caused by manual position change

5 Future Work

Given the constraints from section 3, there is some future work that can be done, such as, the ability to detect the pieces location and the play area without the use of templates, or move the base platform in the cases that the arm can't reach a piece. Another improvement would be the ability to detect all the existing pentominoes and to be able to flip them if needed. The final evolution would be the ability for the robot to assemble the puzzle without guidelines, that is, without having access to the solution.

References

1. Bradski, G. (2000). The OpenCV Library. Dr. Dobbx27;s Journal of Software Tools.