Computer Vision: Shape Categorization Final Report

Prof. Dr.-Ing. Rainer Herpers

Senthilkumar Sockalingam Kathiresan, Matrikel Nr. 9030382
Yannick Weitz, Matrikel Nr. 9030385

Contents

1 Introduction

- Objective / abstract goal, problem analysis

The Task that should be solved by this Computer Vision project is based on a Task taken from the Robocup@Work rulebook. The youBot should be able to fit a 3 dimensional piece into the correct 2 dimensional cavity.
In order to fit the part into the cavity, the correct cavity has to be determined first. The whole task is testing the perception as well as the manipulation ability of the youbot.
As given by the rule book of the Robocup@work:

> single robot is used. The robot is placed by the team freely within the arena. The objective of the task is to pick the objects which are placed on one service area and make a precise-placement in the corresponding cavity at the service area with the special PPT platform (an example configuration is illustrated in Figure 3.2). The task consists of multiple grasp and place operations, possibly with base movement in be- tween, which will, however, be short. The task is finished once the objects are picked up and placed in the corresponding cavities or when the time foreseen for the run ends. Note that the placement of the object in the cavity is finished when the object is fallen into the cavity (i.e. at least some part of the object has to touch ground floor underneath the cavity). All objects to be transported in a run of a team and the corresponding cavities share the same orientation, either horizontal or vertical. This may vary between different teams and different runs.[1]

2 Material and methods

- Description of your setup and your data
- Assumptions and prerequisites
- Approach you are going to apply incl. motivation
- Plan of attack, structure /decomposition of sub steps

Prerequisites

For this task the youBot will get the command to pick up a particular part from a toolbox, from the referee unit, and then drive back to the table, where the cavities are stored. This is where the project begins. The camera of the youBot is pointing at the table. Five out of twelve plates will be placed in a cavities inside the table, arranged in a row (fig. 1). The background of the cavities inside the tiles will thus be black and the full tile will always be visible. The tiles themselves have a nearly white color and will be produced by a 3D printer. The size is fixed to $14cm^2$, with a thickness of $1cm$.
The object, that the youBot is holding is already recognized and known to the bot. There are 5 different objects available.
The cavities are always centered in the middle of the tile and will only correspond to only one particular orientation of the object.
The youBot is using a asus xtion pro live for this task. The camera will be positioned in front of the tiles, with different possible angles, always capturing all five tiles.
There will be no additional light source attached to the youBot, so shadows are possible.

General Approach

As a preprocessing step the image input has to be normalized. Since it can't be assumed that the youBot will always provide a 90° top-down view of the tiles, the input image should should be transformed using homography. The new image after this step should only show an approximate of what the row of tiles would look like from the 90° top-down view. The result will then be further decomposed into the single tiles.

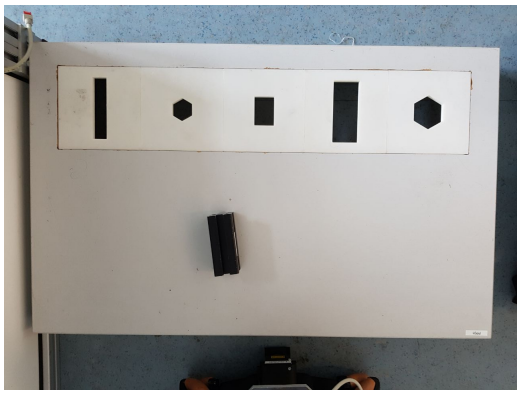

Figure 1: Example of a set of five tiles arranged like they will be in the Robocup@Work competition.

To classify the extracted tiles, a comparison between the tiles and a database is performed. The database consists of example images showing all the tiles in all possible configurations.

Data

Eight different datasets have been created for training purpose. All datasets show the same dataset specific setup from different angles. Five of the datasets consist of a collection of five tiles, placed into the table, with regular lighting, no occlusion and no other objects. The three remaining datasets have been created to test extreme conditions.
For the five regular sets, some tiles have specifically been chosen, while the rest is selected arbitrarily. The first set consists of simple shapes, where the focus is placed on differentiating between the same shape with a different size, which could give problems due to the similarity of the two shapes (fig. 2, top, most left).
The second dataset is introducing more complex shapes, that could be harder to classify and recreate while using homography (fig. 2, top, second from the left).
In the third datasets only small similar looking shapes have been chosen to test the algorithm for a number of similar shapes (fig. 2, top, second from the right).
The fourth set shows four variations of the same 2 dimensional shape, that fit to different 3 dimensional objects (fig. 2, top, most right).
The fifth set is used to determine how the algorithm works for similar shapes with different orientations (fig. 2, bottom, most left).
For the three remaining sets the focus is set to special conditions rather then testing the algorithm for the distinction between different shapes.
In the sixth dataset, the number of tiles has been reduced to four in order to test border cases (fig. 2, bottom, second from the left), the algorithm is not expected to solve this problem, it is just an additional case.
The Seventh set shows images that correspond to a problem that occurs if other areas in the image have the same color and shape of the cavities (fig. 2, bottom, second form the right).
In the last dataset, the light-source has been switched of to create a darker image (fig. 2, bottom, most right).
It can easily be seen, that each example image from the eight datasets has different areas of lighting, thus no dataset testing for shadows was necessary.

3 Approach

In order to normalize the input image and rectify the row of tiles, the first step is to crop the image and get rid of unnecessary information (fig. 3).

so at first the area that should be cropped has to be determined. Since the images are in general a lot

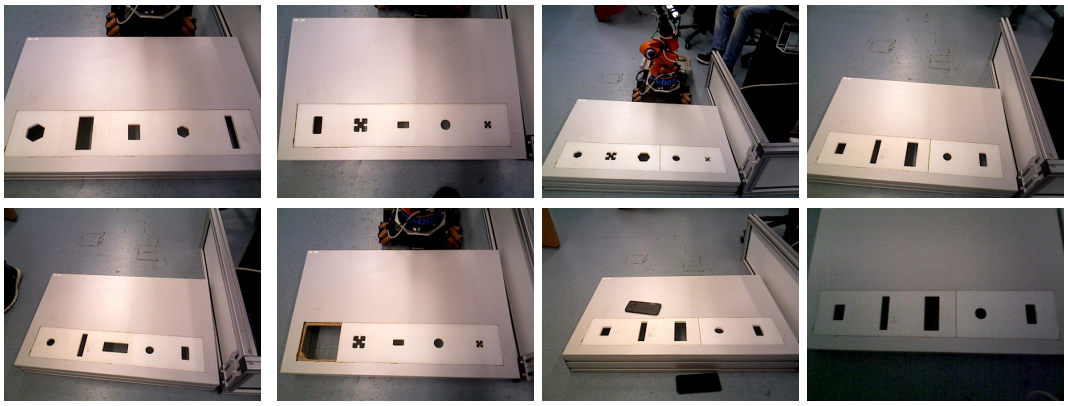

Figure 2: Example images for the eight datasets, starting with one in the top left corner, to eight in the lower right.

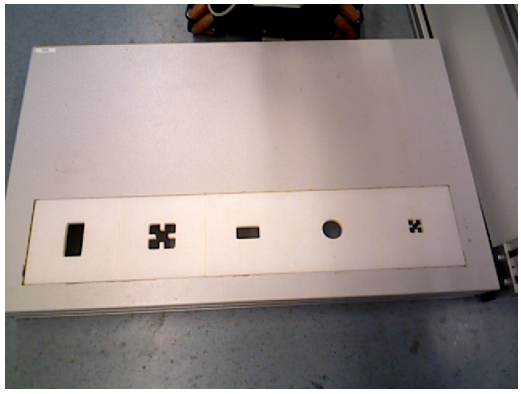

Figure 3: The Original image containing irrelevant information, like the table or small parts of the robot.

brighter than the cavities, an edge filter is applied (fig. 4). Threshold and adaptive threshold have been tested but didn't deliver a good result.

All the contours of the processed image are extracted and the pixel coordinates of the center points are

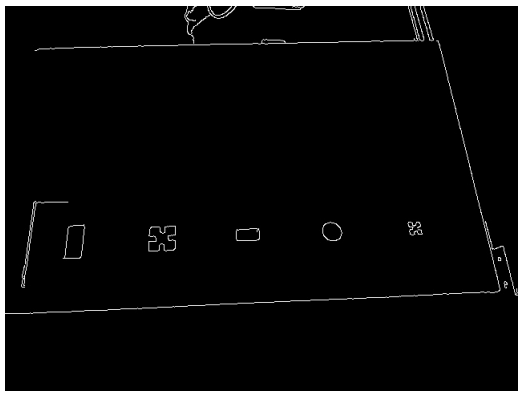

Figure 4: The image after a canny edge filter has been applied.

stored (fig. 5). Since all contours should lie in the middle of the tiles, the centers of the cavities should lie in a line consisting of five equally spaces points.

To find the five points that correspond to these centers, a line is drawn between all extracted centers from

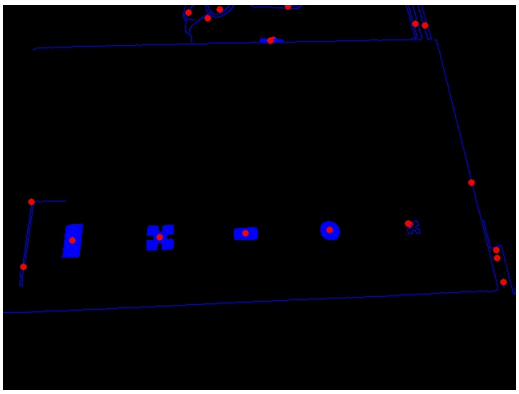

Figure 5: All the extracted contours from the edge detection image.

substep one. From all the lines where at least five points lie closely on the same line, the one is chosen where the error between the points and the line is the least (fig. 6). This has proven to give the correct line for all tested images. Since we search for the minimum distance, lines with five points are preferred over lines with more points, so by default lines with to many points get ruled out.

We know that all tiles have the same dimensions. Thus, the next step aims to find a correspondence

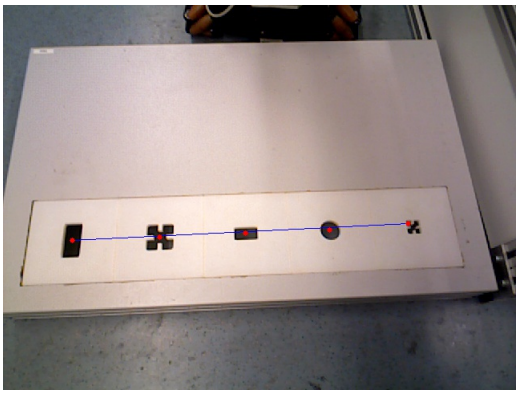

Figure 6: The five points that lie closest to a line have been extracted.

between the pixels and the distance, or pixels per centimeter. The center points of the cavities are expected to lie in the center of the tile. It follows, that the distance between two neighboring centers is equal to $14cm$.Hence, The approximate pixel equivalent to $14cm$ is calculated by getting the mean distance between two neighboring tile centers.
Now the image can be cropped by using (fig. 7):

$$cropped\,image = image[min(x) - \frac{\mu_{dist}}{2} : max(x) + \frac{\mu_{dist}}{2}, min(y) - \frac{\mu_{dist}}{2} : max(y) + \frac{\mu_{dist}}{2}] \qquad (1)$$

Where:

$\frac{\mu_{dist}}{2}$ = Pixel per 7 Centimeter, plus some additional term to compensate the angular orientation of the five tiles
$min(x)$ = Minimum x value of all center points
$max(x)$ = Maximum x value of all center points
$min(y)$ = Minimum y value of all center points
$max(y)$ = Maximum y value of all center points

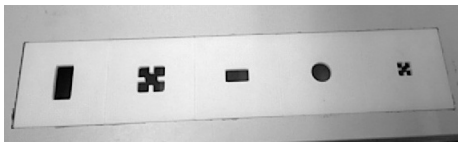

Figure 7: A lot of irrelevant information has been cropped out of the image.

To extract now the corners of the five tile row, the adaptive threshold filter is applied to the cropped image, since this has proven to give a good differentiation between the five tile row and the background. Again the contours of the image are extracted, this time with a limitation of the area, that the contour should enclose (fig. 8).

The low resolution of the image makes it hard to extract the four corners immediately. The Harris corner

Figure 8: The contour of the image that corresponds to the biggest area of the cropped image.

detection is used to extract the four corner points, since these are a little misplaced the cornerSubPix method as described in [2] is used to extract the corners more precisely(fig. 9)

The four points are used to transform the perspective of the tiles to create and artificial top-down image

Figure 9: The two extracted corners are shown, red shows the Harris corner, while green shows the corner with sub pixel precision.

containing only the five tiles (fig. 10).

The resulting image is then divided into five equal parts and the images are stored (fig. 11).

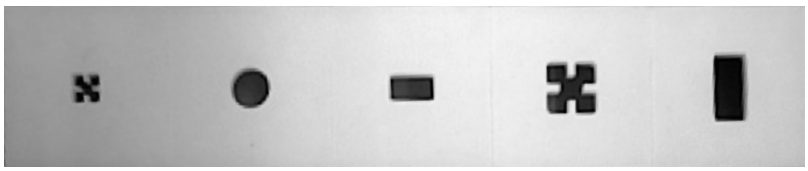

Figure 10: An artifical top-down view of the five tiles.

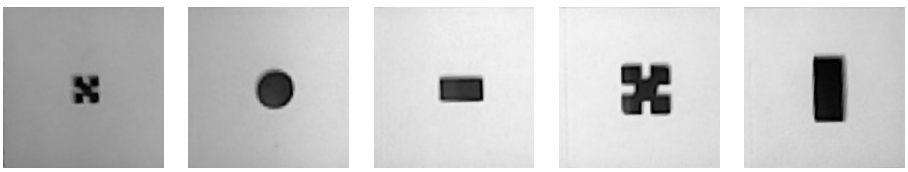

Figure 11: The five cropped individual images of the tiles.

4 Evaluation and results

- Test /Evaluation design
- Riscs

References

[1] Robocup @work rulebook, p. 35. Robocup-rulebook2017

[2] Webpage, last viewed 13.12.17 20:25, https://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html