Exercise 3: Object Detection with SIFT and GHT

In the last exercise you implemented feature descriptors to calculate correspondences between two different images. Based on this idea, in this exercise you will go one step further and implement an object recognition algorithm for multiple instances using the Generalized Hough Transform (GHT).

Download files for exercise 3

Figure 1: Goal - Detect all the instances of an object in a scene.

Part 1: Implementation (10 points)

This time, you will not need to program your own descriptor and matching code. Instead, we will use the OpenCV library to calculate more robust SIFT descriptors and get correspondences between images. For object recognition, you will have to implement the Generalized Hough Transform. This exercise is more open in terms of specific implementation details. You will need to experiment with different clustering algorithms and how to filter bad matches, to achieve a satisfying result.

The main script only needs to be adapted to switch between scene images.

1. Generalized Hough Transform (GHT) for Object Recognition (8 points)

The general idea behind the Hough Transform is that objects are found through a voting procedure in a configuration or Hough space. If evidence for a particular object configuration is found in the image, a vote is added for this configuration. In our case such evidence is a matched feature vector from the object image to the scene image. The **location, orientation and scale** of the feature with respect to the object's coordinate system is known from the object image. If the feature is also found somewhere in the scene image, it is possible to calculate the likely location, orientation and scale of the whole object and add a "vote" for that configuration (Fig. 2a). After all matched feature vectors have voted, **clusters in the configuration space correspond to possible object instances** (Fig 2b).

In our case, the configuration (or voting) space for the object is 4-dimensional, the dimensions being x, y, the $scale$, and the $orientation$.

Our goal is to detect multiple instances of the same object. We can achieve this by applying clustering algorithms to find probable object configurations in the configuration space. We can then use the center of each cluster as the most likely object configuration. The `detect_objects` function needs to be able to detect multiple clusters and return the corresponding object configurations (x, y, $orientation$, $scale$).

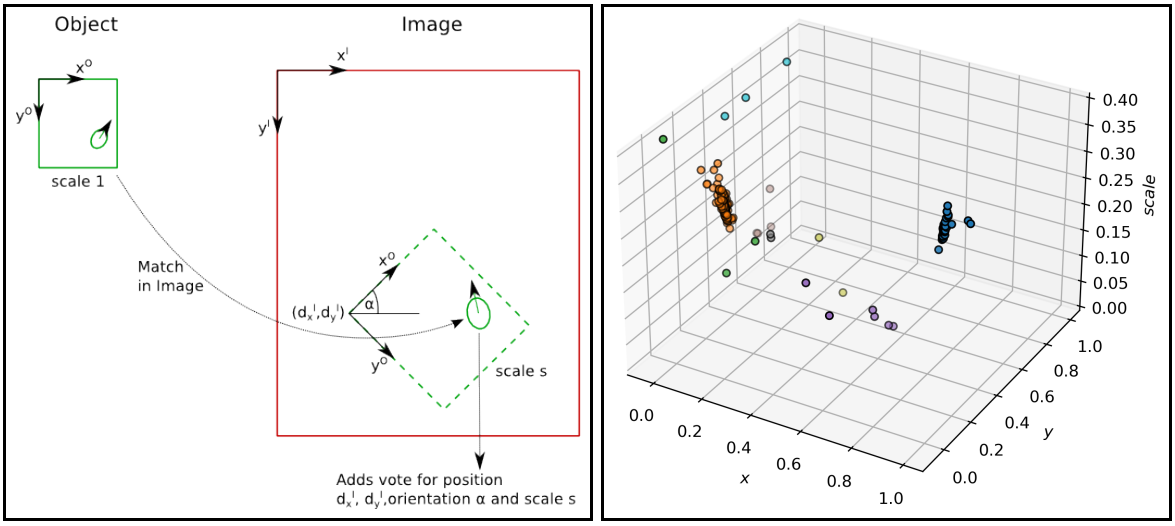

Figure 2a (left): Principal idea of the Generalized Hough Transform for object recognition. Figure 2b (right): 3-dimensional configuration space (only x, y and scale coordinates). Each point is one match and represents a possible object configuration. Different colors are different clusters. We see two big clusters representing likely object configurations.

Summary of necessary implementation of the GHT:

- Filter out bad matches we receive from SIFT. Lowe suggests in his paper to use a distance ratio between the distance of the best match and the second best match. If this

ratio falls below 0.8 the match is rejected. This would not work well in our case, since we want to detect multiple instances of the same object. For our use-case an arbitrary threshold of the match distance is fine, even though it is generally not recommended for high dimensional feature vectors. Attention: Do not finetune the threshold to exactly match our dataset (overfitting). Rather pick a clustering algorithm which can also handle bad matches in step 3. (1.5 points)

- Compute the (x, y, s, o) configuration for each match pair, using your implementation of `match_to_params` and add it to a voting space array with the shape $(number\_of\_matches, 4)$ with each row containing the (x, y, s, o) configuration of one match. The function `match_to_params` takes two matching keypoints, one in the object image and one in the scene image. It uses the relative difference in position, scale and orientation between the two keypoints to calculate the object configuration in the scene image coordinate frame. It should output the x and y coordinate of the upper left corner of the object in the scene image coordinate frame, the relative scale of the object and the orientation of the object in the scene image coordinate frame. The idea is shown in Figure 2a. (2.5 points)
- Find clusters in the voting space with ideally each cluster referring to one object configuration. You should use a clustering algorithm from 2.3. Clustering — scikit-learn 0.23.2 documentation. Read the documentation of the algorithm you choose carefully and pick sensible parameters. You should normalize the different dimensions of your configuration space for clustering to work properly. One possibility might be to use kmeans with a $k > nr\_of\_objects$ and then discard clusters depending on the number of inliers and the distance to other clusters. Note that this part of the exercise will be graded partly on whether your approach would generalise well to images outside of the set that is given. An introduction to clustering methods is given in lecture 5. (3 points)
- Extract the (x, y, s, o) configuration for the best object position estimates (the centers of the remaining clusters). (1 point)

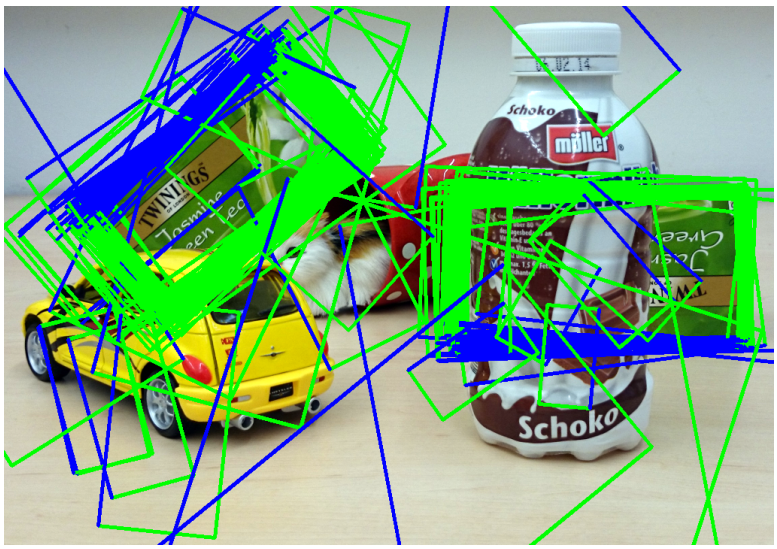

Figure 3: All possible configurations extracted from votes in the GHT. These need to be processed so that (ideally) there is one configuration per object

2. Set up Transformation Matrix (2 points)

After the configurations of the objects have been found, they should be indicated by bounding boxes placed on top of the scene image (see Figure 1 and 3). The coordinates of the rectangles are defined in the object coordinate system in the main script. To display the rectangles at the correct position in the image, the coordinates first have to be transformed to the scene image coordinate system, according to the detected translation, scale and rotation of the object. In the function `transform_points` in the file `helper_functions.py`, you have to implement this transformation. Set up the 3-dimensional transformation matrix and apply it to all input points. **Use homogeneous coordinates to be able to vectorize the transformation!** You do not have to add code to draw the rectangles, this is done by `draw_rectangles`. For more information about transformations refer to Szeliski Book 2.1.2: 2D transformations.

Useful commands

`np.matmul` or `@` : Matrix multiplication

`np.c_[array1, array2]` : Extend array1 with array2 as additional columns

`np.r_[array1, array2]` : Extend array1 with array2 as additional rows

`np.unique` : Returns the sorted unique elements of the array. With parameter `return_counts = True` : Return the number of entries for each unique element.

`plt.axes.Axes.scatter` : A scatter plot useful to analyze different clustering algorithms.

Part 2: Documentation (6 points)

- Answer each numbered question separately, with the answer labelled according question number. You do not need to include the question text.
- Include images to support your answers. These should be referenced in the answers. The document should include information about what parameter settings were used to generate images so that they can be reproduced.
- Your answers should attempt an explanation of why an effect occurs, not just describe what you see in the supporting images.
- Be precise. "It is better" or similarly vague statements without any additional information is not sufficient.

Points may be deducted if your report does not satisfy the above points.

For the experiments, you are provided with a template image of the object (object.png) and four scene images containing the object several times in more or less cluttered scenes (image1.png to image4.png).

1. Describe your method to detect peaks in the voting, and your approach to reducing the number of possible object configurations. Explain why you decided to use a specific clustering algorithm. (2 points)

2. Include the detection results for all the images, and write a short interpretation of them. Does your code work well on all images? If there are issues in any of the images, add a few sentences explaining what you think might be the cause. (2 points)

3. Describe your approach to reduce the number of matches and explain why you have chosen this approach. Change the parameters in your match reduction and show the effect in multiple images. For visualization you can either use `cv2.drawMatchesKnn` similar to how it is used in the main file or show the effects on your final results (depending on which images show the effects better). Which problems might occur with different images not in our example dataset? (2 points)

For all result images, use the same parameters and mention them in the documentation so we can reproduce your results.

Maximum 1500 words, arbitrary number of pages for result images (referenced in the text). Violating the text limit will lead to point deductions!

Assistance

Please keep to the following hierarchy whenever there are questions or problems:

1. Forum: Use the TUWEL forum to discuss your problems.
2. Email for questions help requests: machinevision@acin.tuwien.ac.at

Upload

Please upload all Python files and your documentation (pdf-format) **as one ZIP-file** via TUWEL.