Object Detection Writeup

Karhan K. Kayan

6/7/2021

In this project, I implemented object detection using feature matching, Hough Transform, RANSAC, and homography estimation. The implementation can be found in `objectdetection.ipynb`. The images are in the folders `stop` and `book` respectively. Note that I renamed some files in the stopsign folder.

1 Implementation Description

Object detection consists of three steps: Loading the files, building the database, and running detect. All the functions use only the opencv library for image processing.

- The function `load(name)` Loads the image files including the reference images and input images.
- The function `extract_correspondences(im1, im2, coeff)` takes two images and runs feature matching with SIFT desriptors on them. First the descriptors are computed, then brute force matching is run using k=2 nearest neighbors. And bad matches are eliminated using the ratio coefficient `coeff` as explained in Lowe's original paper. The coordinates, the angles, and the scales of the remaining descriptors are returned.
- The `Database` class defines the query database described in the project description. It has two methods. `match(input_img, coeff)` method takes an input image and a matching coefficient and runs `extract_correspondences` to get a feature matching between the reference image and the input image. The `query(index)` method takes the index of a keypoint and queries the database as described in the project description. It returns the top left and the bottom right coordinates of the bounding box that this keypoint would vote for in Hough transform. The way that it does is that it orients and scales the top left and bottom right corner vectors of the reference image based on the orientation angle difference and the scale ratio between the reference and input descriptors.

- `test_sift` just helps to visually see the matches between the reference and the original image.
- The `HoughTransform` class applies Hough transform to cluster the keypoints that vote for the same pose of the reference image inside the original image. Its only method is `fit(top_left_pts, bottom_right_pts)`, which takes a list of top left and bottom right corner coordinates from the database query and returns the keypoints that agree on the pose of the object (that voted for the same parameters). `who_voted` is a dictionary that keeps track of which keypoints voted for which set of parameters. `accumulator` is the standard Hough Transform accumulator. For each keypoint, the value of the accumulator that corresponds to the bounding box parameters that the keypoint votes for is increased by one. In the end, the cluster of keypoints that has the highest number of votes on the a parameter set is returned.
- The function `detect(ref_db, input_img, match_coeff, bin_size, line_thickness)` is the main function of this project. It takes a database, an input image, matching coefficient, and bin size for the Hough transform and prints the input image with the reference object detected in it. It also returns four points that identifies the quadrilateral the object is found in. The reason for using four points is because we use a projective transformation. The way that this function works is the following: First it runs feature matching on the database. Then, it gets bounding box parameters for all keypoints by querying the database. Then, it runs Hough Transform using these parameters to get a good cluster of keypoint indices. Using this cluster, it estimates a homography between these keypoints and their matching counterparts in the input image using RANSAC. Note that this is done with the opencv function `cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)`, which repedeately runs RANSAC on the set of inliers by default. Then the reference image corners are transformed using this estimated homography and the transformed points are returned, and the quadrilateral is printed.

2 Design Choices

There are two important hyperparameters that I used in this project. The first one is the matching coefficient, and the second one is the bin size for Hough Transform. The matching coefficient is used to eliminate bad matches using Lowe's ratio method. So, decreasing it gives a small number of good matches and increasing it gives a large number of unreliable matches. The bin size is used to select reliable keypoints that vote for the same pose for the object. A smaller bin size means that a smaller number of reliable keypoints will be selected. I found that for images that do not contain a good large set of SIFT descriptors, it is better to have a higher matching coefficient and a little bit lower bin size to not preemptively eliminate a lot of keypoints. Another

choice I made was when representing the parameters for the bounding boxes that the keypoints vote for. I decided to use top left and bottom right corners. I could have inclueded more parameters like the orientation of the rectangle, but since the number of matches is already small, this would result in too fine granularity for Hough Transform. Also these four parameters are easy to calculate from the orientation and scale difference of the SIFT descriptors.

3 Evaluation

Qualitatively the results are very good. It took some playing with the hyperparameters to get good results for some images, but overall object detection was accurate even in the case of occlusion, duplication, etc. A possible improvement would be choosing better hyperparameters. I found that playing the the matching coefficient and the bin size wildly changes the end result. To improve the speed, I could limit the iterations of RANSAC. For certain images where there are enough matches, it could be a good idea to include the orientation of the bounding box as a parameter for Hough Transform. Another possible improvement could be to include preprocessing in the pipeline to make the reference and input images similar to each other in terms of brightness, saturation, etc. I could also implement multiple instance recognition by considering more than one Hough Transform clusters.

4 Limitations

The most fundamental limitation of this method is that it is limited to strict instance level recognition. What this means is that small changes to the same object can cause it to be not recognized. I also found that the method is also limited when the object is at an angle to the camera. This is because the SIFT descriptor matches are not that good in that case. Another mode of failure is when the input image background is very similar to the reference object. This causes a lot of false matches to be found and can cause the system to fail. Another limitation is that my method is limited to finding a single instance of the object, though this can be remedied by considering mutliple Hough Transform clusters.