Computer Vision P Set 2

Bayard Walsh

January 2024

1 findinterestpoints

I decided to use the Harris Corner Detector as mentioned in class. First I start by setting up the appropriate terms for calculating R by demonising the squared gradients dx, dy (found by Sobel). Note that the scale parameter passed here is used for a wider window in the Gaussian function, which is why I chose Gaussian parameters 1,3,5 (as sigma must be odd). Next, I loop through all features, calculating R for any y,x pair in the image, and saving any instances with R greater than 0. It could have been faster to calculate R beforehand as its own grid rather than calculating R for each x,y instance, however, I found the performance was higher with this approach. Next, I zeroed out any values from coords in the mask (if a mask is applied). Afterward, I apply max thresholding for the 8 values around any points still of interest and remove any non-local maximum. Finally, I take the maxpoint highest points still left in our candidate grid (by finding the threshold where the top x percent of the grid is 200), and return those with their coordinates as xs, ys, scores.

2 extractfeatures

I decided to use the scale factor as an aspect of the spatial width within binning, and this is multiplied by 3, so the base width is 3. After some variable setup, I calculate the magnitude and theta grids for the image. Next for each point, I grab the window around that point for both the magnitude and theta grids which vary depending on the given scale. Next, I sum the gradients within that window, reduce the sums over the threshold by a factor of a fifth use the magnitude sum and thetas to calculate a histogram over the $-\pi$, π space. This is repeated for every point and then saved as a list of feature descriptors for each input N.

3 matchfeatures

For match features, I implemented a feature-matching algorithm similar to the Nearest Neighbor Distance Ratio mentioned in class. Implementation is simple

and involves iterating through all features in feat0, and then subtracting the corresponding scores from feat1 and normalizing to get the norms. From there I sorted the norms and used the ratios of the two best norms to determine the score for a given feature, by adding the scores of the highest features for score0 and I score1 and dividing by the norm of the first and second feature from feats0, and saved the scores and matches as outputs. In terms of a very small edge case, I was encountering some cases where either of the first two norms would be 0, and to avoid dividing by 0 errors I set the ratio to 1.0 here, because 0 indicates a very closely correlated distance, so a higher ratio is appropriate. Note that these errors were infrequent, so this modification shouldn't affect the matching behavior much.

4 houghvotes

I started by subtracting the differences in x and y between the two interest point sets, saved in tx, ty, and then I saved the widest dimensions for each of tx, ty, which I then used to calculate the edges of the linear space. From there, for the two linear spaces I add the score from scores to the best fitting bin in the votes grid. In each iteration, I save the highest vote grid value amount and bin parameters and then return those as the most likely translated direction for each x,y. In terms of finding the discrete space, I found 30 to be a good trade-off in terms of performance between the cup and car through trial and error. With a linear space of 34 performance increased by a few percent by cup but decreased for car.

5 objectdetection

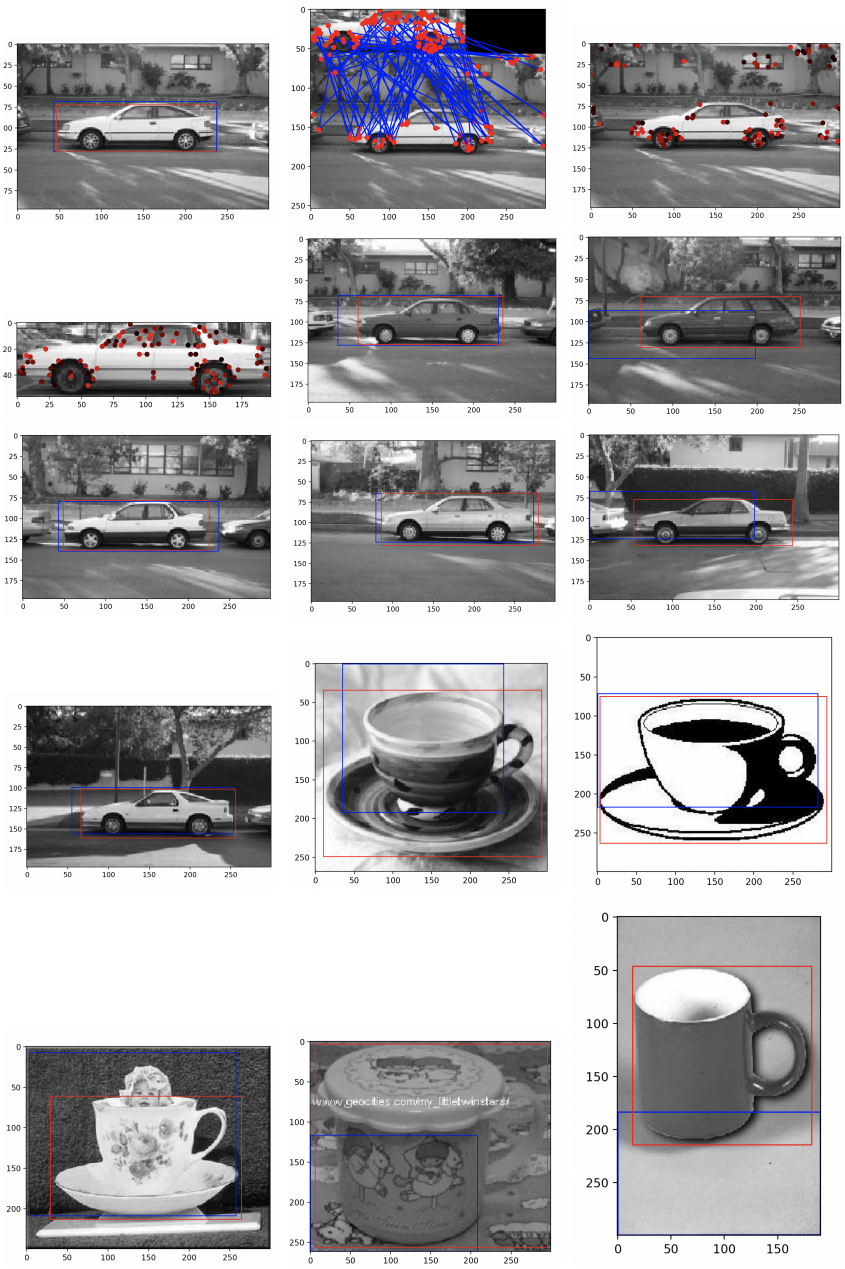
I started by finding the feature points + extracting features from the testimg. After some trail and error I determined that increasing the max points for the image increased performance, so I opted to use 400 instead of 200 points. Next I iterate through the template images and masks, as well as with a variety of scales (1,3,5) (note again that odd numbers are needed because of my implementation of using scalar for Gaussian denoising), and consider each image + mask pair with each scale. Here I extract the interest points and then the features from those points, and then run match features and hough votes compared to the initial object. Here I'm using the standard 200 points for each feature to save on runtime a bit. I save the dimensions of the image and the translation of the shape that had the most votes and then form the box based on the shapes of the best fitting translation vector in relation to the shape of the overall image after iterating through each image. This approach for picking the best box worked well, though for most of the images my detector picked sigma = 1.0, suggesting that more narrow sigma works well for simple object detection.

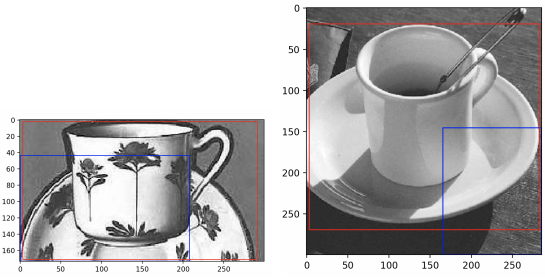
6 Performance Analysis

Single Scale Version Results:
0th datacar image IOU 0.8241141023121765
1th datacar image IOU 0.9156626506024096
2th datacar image IOU 0.36086793249935456
3th datacar image IOU 0.8742256801625455
4th datacar image IOU 0.8760430968044874
5th datacar image IOU 0.45660374071137994
6th datacar image IOU 0.8274790827068302
class datacar, average IOU 0.7335708979713119, total running time 78.36491775512695s
0th datacup image IOU 0.4870152709792772
1th datacup image IOU 0.7063867481619276
2th datacup image IOU 0.6348082076053548
3th datacup image IOU 0.3775281939958702
4th datacup image IOU 0.28848255263349604
5th datacup image IOU 0.5308073323453625
6th datacup image IOU 0.19969058735094936
class datacup, average IOU 0.4606741275817 4825, total running time 194.4525923728943s

MultiScale Version Results:
0th datacar image IOU 0.8241141023121765
1th datacar image IOU 0.9156626506024096
2th datacar image IOU 0.36086793249935456
3th datacar image IOU 0.8742256801625455
4th datacar image IOU 0.8760430968044874
5th datacar image IOU 0.45660374071137994
6th datacar image IOU 0.8274790827068302
class datacar, average IOU 0.7335708979713119, total running time 187.71479320526123s
0th datacup image IOU 0.4870152709792772
1th datacup image IOU 0.7063867481619276
2th datacup image IOU 0.6348082076053548
3th datacup image IOU 0.3775281939958702
4th datacup image IOU 0.1181257088597577
5th datacup image IOU 0.5308073323453625
6th datacup image IOU 0.19969058735094936
class datacup, average IOU 0.4363374356140714, total running time 599.9521670341492s

7 Generated Images from Testing




0
20
40
60
80
100
120
140
160
0
50
100
150
200
250
0
50
100
150
200
250
0
50
100
150
200
250