Kağan Cenan
28462

ELEC408/ Homework 2 / Local Feature Matching

The goal of this assignment is to create a local feature matching algorithm using techniques described in Szeliski chapter 4.1. The pipeline we suggest is a simplified version of the famous SIFT pipeline. The matching pipeline is intended to work for instance-level matching multiple views of the same physical scene. The work was divided into 3 steps:

1. Corner detection and interest points
2. Features extraction
3. Matching points

1)get_interest_point:

My get_interest_points function followed the algorithm outlined on the lecture slides/in Szeliski:

1. Change image to double values to work easily
2. Gaussian filtering to smooth image
3. Finding derivatives of image via sobel filter
4. Calculate the squares of the derivatives
5. Convolve each of the three resulting values with a larger Gaussian
6. Calculate the interest measure given in the lecture slides: *har* = g(Ix^2)g(Iy^2) - [g(IxIy)]^2 - alpha[g(Ix^2)+g(Iy^2)]
7. using ordfilt2 to find local max point
8. Checking the treshold condition to reach more accure corners
9. defining x and y for the corner points' position

2) get_features / Local Feature Creation

My implementation of get_features created SIFT-like descriptors for each detected interest point as follows:

1. Features is the array which contains histogram array of each interst points
2. For each interest point, grab a 16x16 window around the point and filter it (using the same larger Gaussian filter used to blur the squared image derivatives in step 4 of get_interest_points)
3. For each 4x4 cell in the grid, create a gradient orientation histogram with 8 row

4. Divide each of these windows into a 4x4 grid of 4x4 cells. Our feature point became at in the array (9,9) for position
5. For each pixel in each cell, I added pixel's gradient value up to direction. I divided direction into 8 each has 45 angle and for example for -150 I put it in the for row on for the first cells 8 value array part
6. normalize the feature.

3)match_features / Feature Matching

The code implements ratio test to match features. It also uses the ratio test values as confidences; low ratio means a more confident match. Matches are thresholded to have a ratio below 0.7 ; matches with a larger ratio are rejected. The algorithm is as follows:

1. For each feature in features1, compare to all features in features2 of the distance
2. Keep track of the nearest neighbor and second nearest neighbor, where the nearest neighbor is the feature
3. After comparing to all features in feature2, if the nearest neighbor of the feature being matched has a ratio greater than the threshold (.7), reject the match. Otherwise, add it to the list of matches, and add the ratio value to the accept on matrix .

Result:

As a result I found 44 good, 6 bad matches. The final result: