Hieu Nguyen
GTID: 903185448
Email: hieu@gatech.edu
10/12/15

CS-6475 Computational Photography Assignment 7

For this feature detection assignment, I utilized a simple algorithm taken from the OpenCV documentation. This technique performed brute-force matching with ORB descriptors to find the query image in the train image. The resulting image descriptors are matched, and 10 are selected to be plotted in the output images. My test object was a box of Froot Loops breakfast cereal, which was placed in cluttered areas of my bedroom (with artificial lighting).

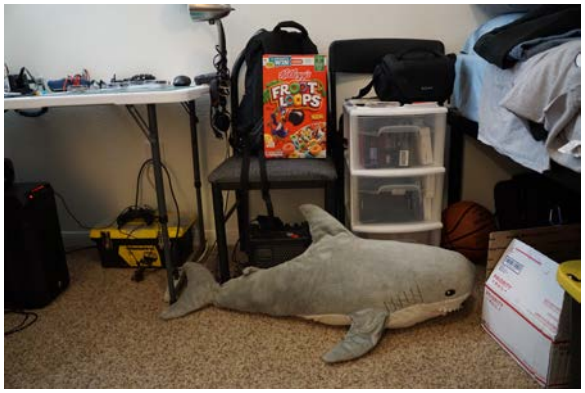
Input Images

Original

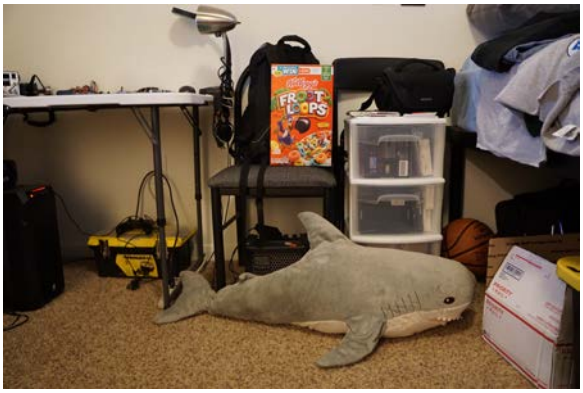
Sample

Lighting

Cool White Illumination



Warm White Illumination

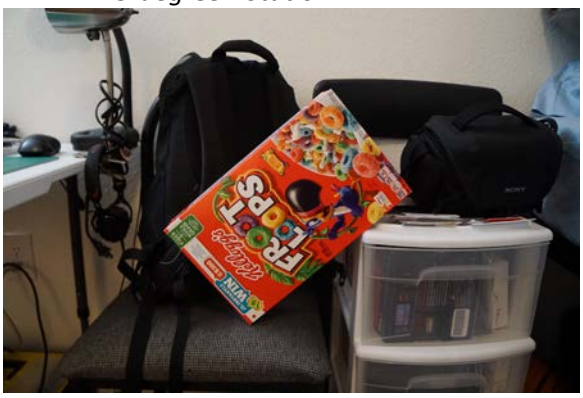


Outdoor Illumination

Rotation

45-degree Rotation

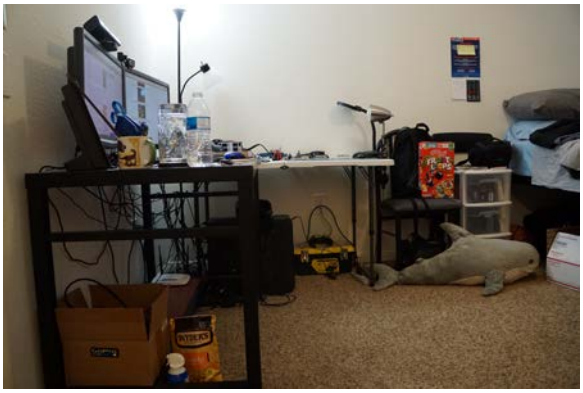
225-degree Rotation



Scale

Near

Far



Function Details

findMatchesBetweenImages()

This function finds matches between two input images by using the Brute-Force matcher techniques in OpenCV. It takes the descriptor of one feature in the first set and is matched with all other features in the second set using a distance calculation. The matches with the closest distances are taken to be the most accurate. First, I compute the keypoints and descriptors using SIFT. Next, I create a BFMatcher object and compute the matches between both images. The resulting matches structure is sorted by distance and the best 10 matches are returned for plotting.

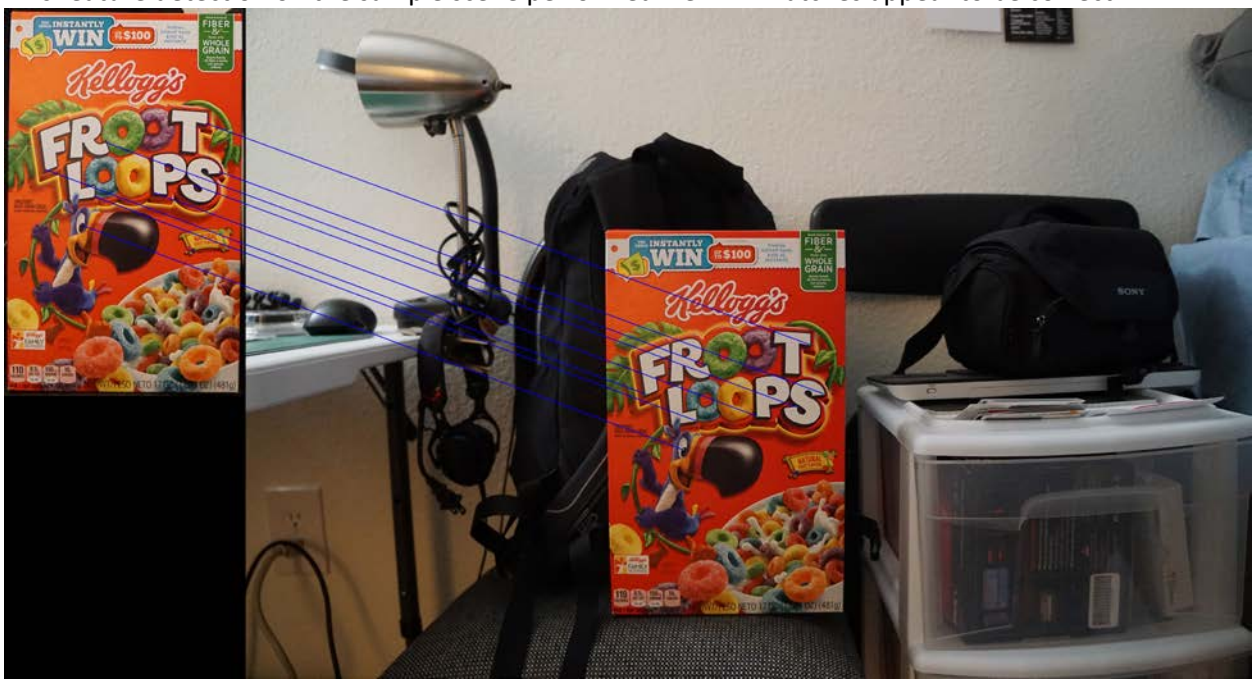
```
def findMatchesBetweenImages(image_1, image_2):
  # Compute SIFT keypoints and descriptors for both images
  orb = cv2.ORB()
  image_1_kp, image_1_desc = orb.detectAndCompute(image_1, None)
  image_2_kp, image_2_desc = orb.detectAndCompute(image_2, None)

  # Create a Brute Force Matcher, using the hamming distance
  bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

  # Compute the matches between both images
  matches = bf.match(image_1_desc, image_2_desc)

  # Sort the matches based on distance to get the best matches
  matches = sorted(matches, key=lambda x:x.distance)

  # We coded the return statement for you. You are free to modify it -- just
  # make sure the tests pass.
  return image_1_kp, image_2_kp, matches[0:10]
```

In the future, I would like to improve the robustness of this function by employing a RANSAC algorithm to remove any outliers in the matches. There are several other OpenCV algorithms to improve feature detection such as the ratio test, FLANN based matcher, and homograpy. I tried to implement some of these, but my version of OpenCV made things difficult (missing support for a few functions). I will need to play around with it more to create a better feature detection algorithm.

Output Images

Sample

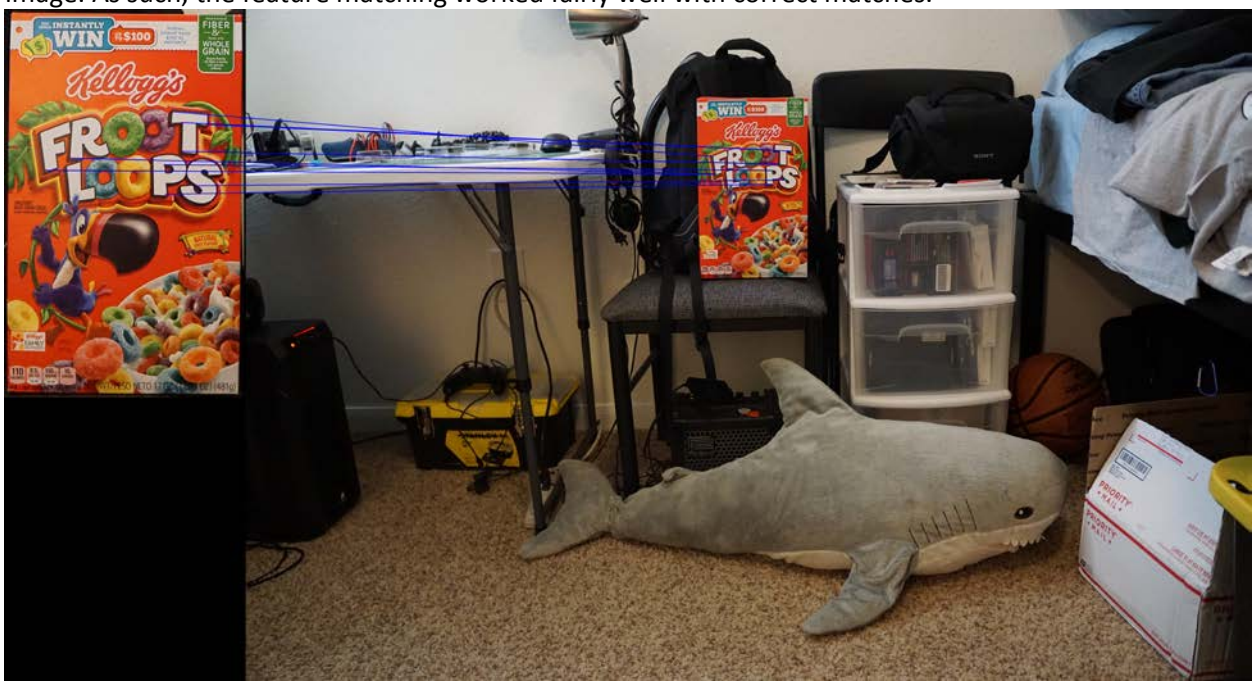
This feature detection on the sample scene performed well. All matches appear to be correct.



Lighting

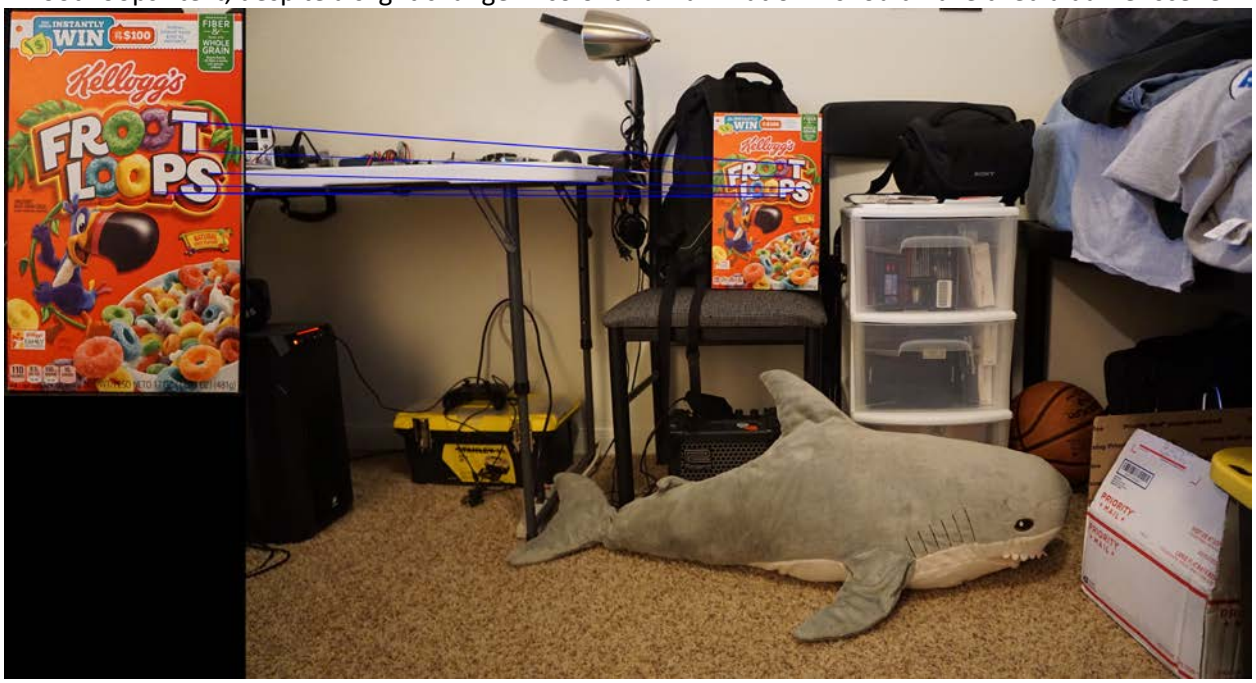
Cool White Illumination

This scene utilized a lamp with a cool white light bulb, which was the same illumination as the query image. As such, the feature matching worked fairly well with correct matches.



Warm White Illumination

This scene utilized a lamp with a warm white light bulb. The feature matching still worked well on the "Froot Loops" text, despite a slight change in color and illumination. I should have tried a darker scene.



Outdoor Illumination

This scene used natural illumination (taken outdoors) instead of artificial lighting. The result was surprising, as the output matches were very accurate. I believe this is due to the quality of the cereal box being in focus, so it is easier to eliminate objects from the blurry background.

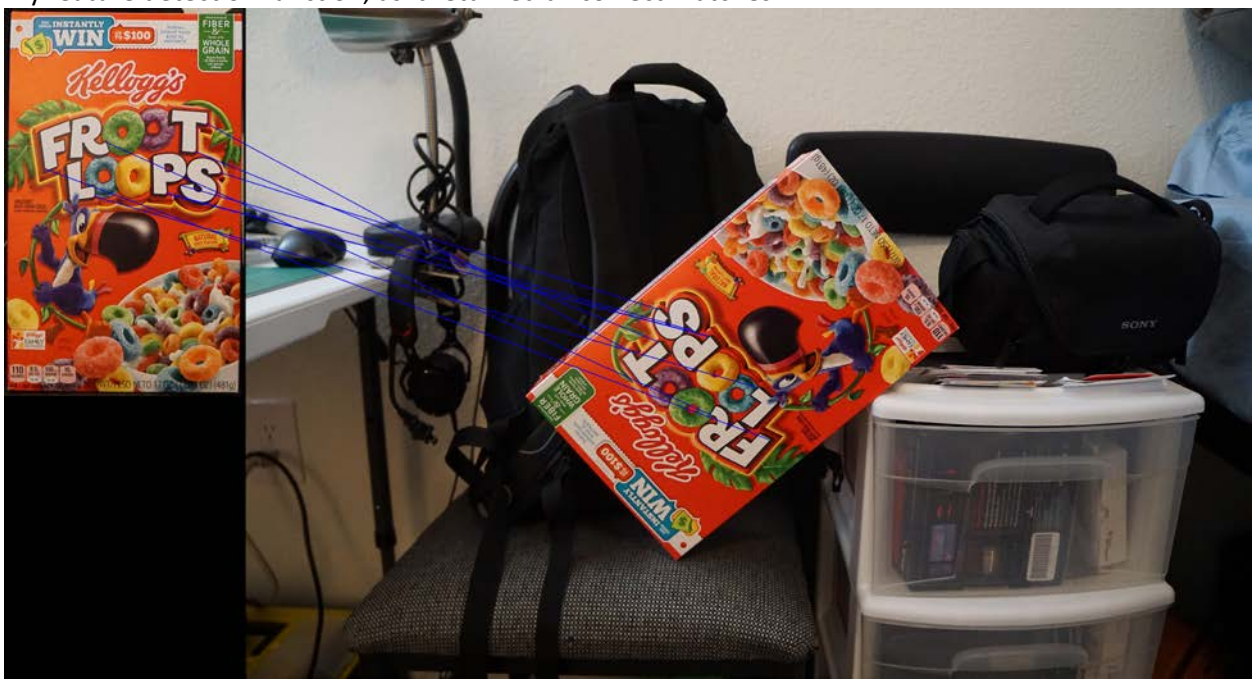
Rotation

45-degree Rotation

In this scene, I rotated the box by 45 degrees. The feature detection actually performed quite well with all correct matches.

225-degree Rotation

In this scene, I rotated the box by 225 degrees, so that it was upside down. This was not a problem for my feature detection function, as it returned all correct matches.



Scale

Near

This scene was taken very close to the box. It performed very well with all correct matches.

Far

This scene was taken far from the box and incorporated more clutter for a noisier image. This feature detection performed the worst, with zero correct matches. I believe this is due to the other objects creating a higher margin for error, producing more incorrectly matched features. I tried taking this picture multiple times, but could not get any improved results with the my current function.