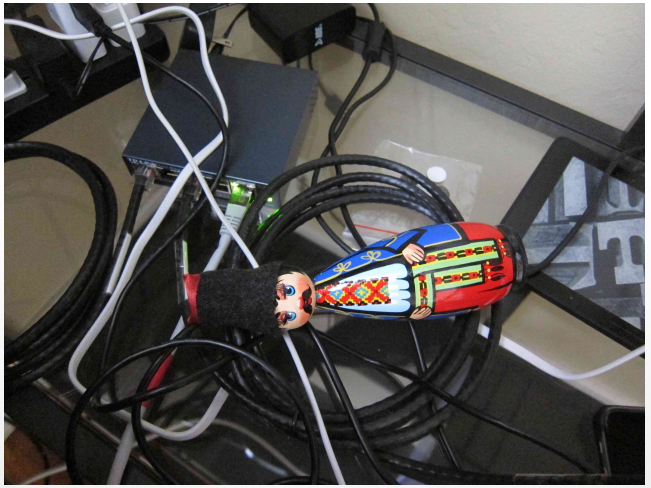
Computational Photography
Assignment #7: Feature Detection & Matching

Ram Subramanian
Fall 2016

Sample

Lighting

Template



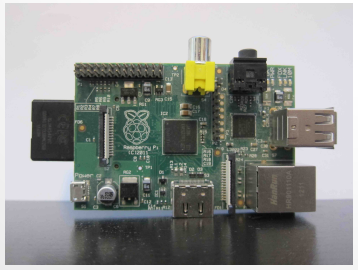
Rotation

Scale

Description of Object

I ended up choosing the doll as my subject because of the distinct patterns painted on it, which would provide for ample and well-defined feature points. It also produced near perfect results in all of the test cases.

I tried a few other objects as my subject – a bare raspberry pi and a pattern on my kindle. They had good feature matches, but ultimately since we only take the top 10 matches by hamming distance, these were not selected and I ended up having very poor results.



Description of Scenes

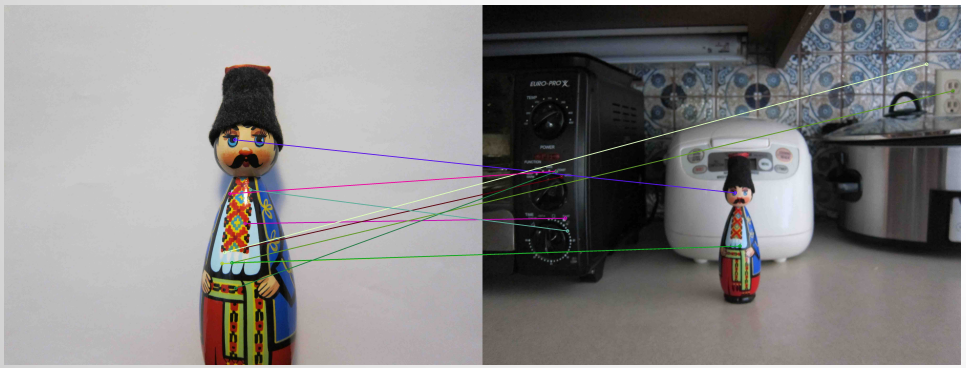
My scenes mainly consist of the shelf in my room containing various objects such as books, cables, gadgets, ethernet switch, my kindle, etc. I also use the tiling in my kitchen to provide a background for one of the tests, due to its patterns.

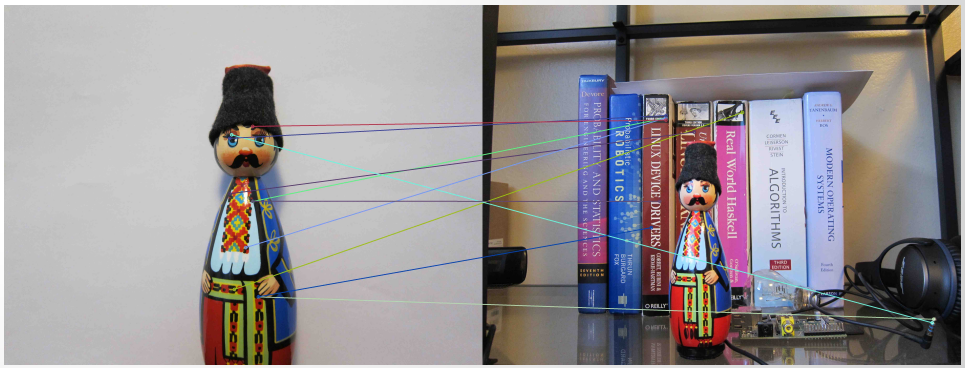
Some other scenes I tried:

- against my piano and sheet music: too plain of a scene
- against my desk: again no sharp contrasts or interesting features
- in my garage: ultimately didn't work out well
- in the pool: for the blur tinting, doll was simply too small in the picture (should have bought a gopro :/)

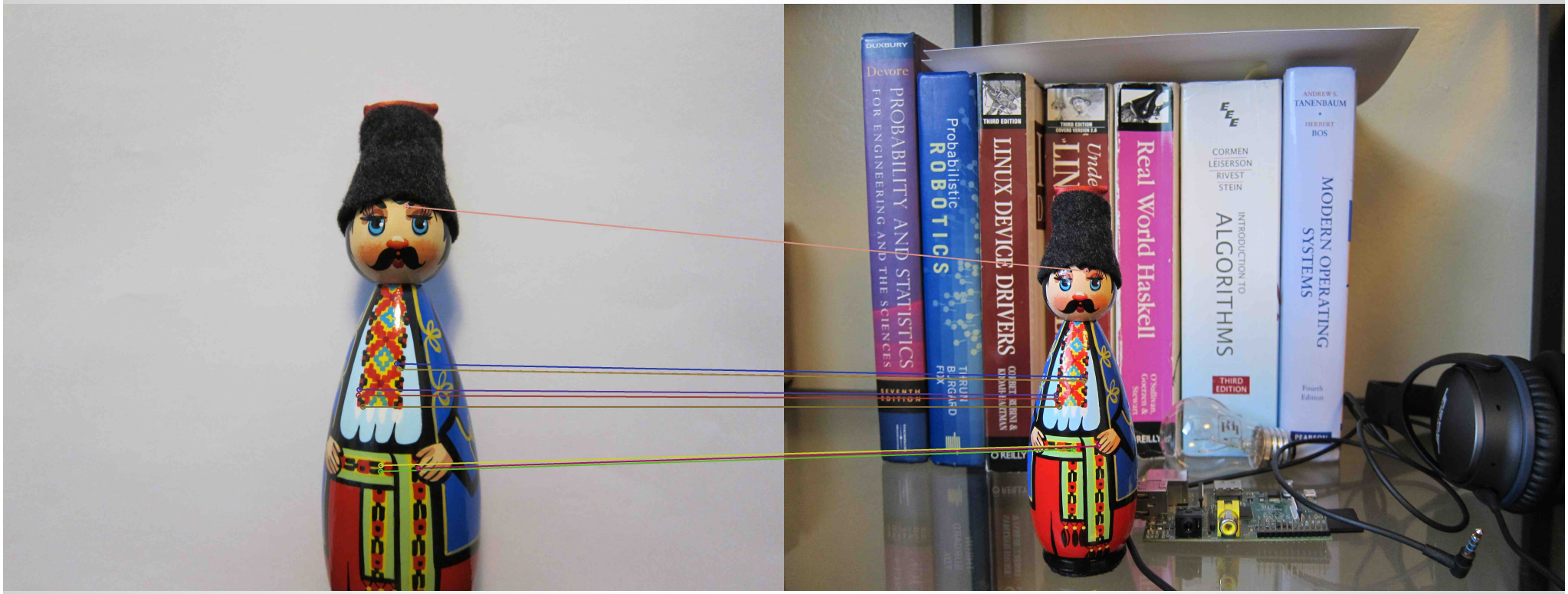
Making the Basic Setup Work

I had to re-take pretty much all tests multiple times to get good results and bizzarely, some of them gave perfects results, despite there only being minor differences (see below)!





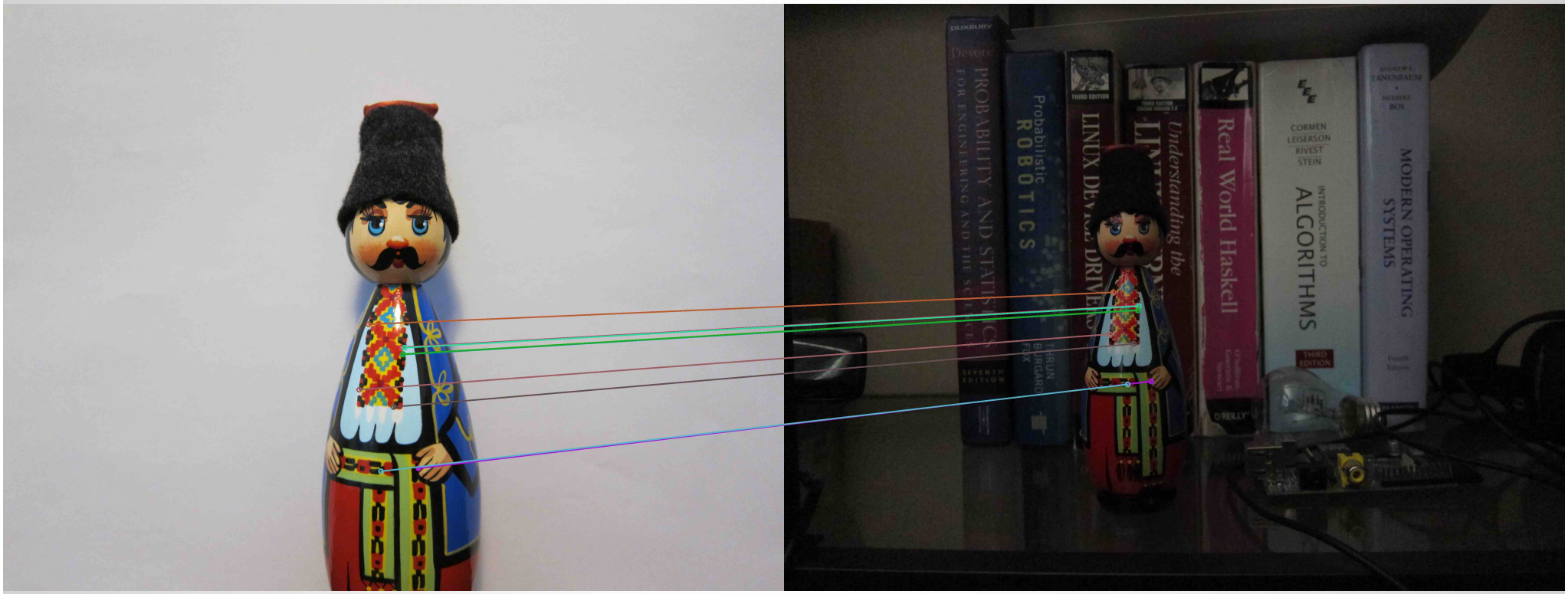
SAMPLE



Pretty good proof-of-concept!
Although, two of the matches are so close to each other, it's hard to tell apart.

SCORE: 10/10

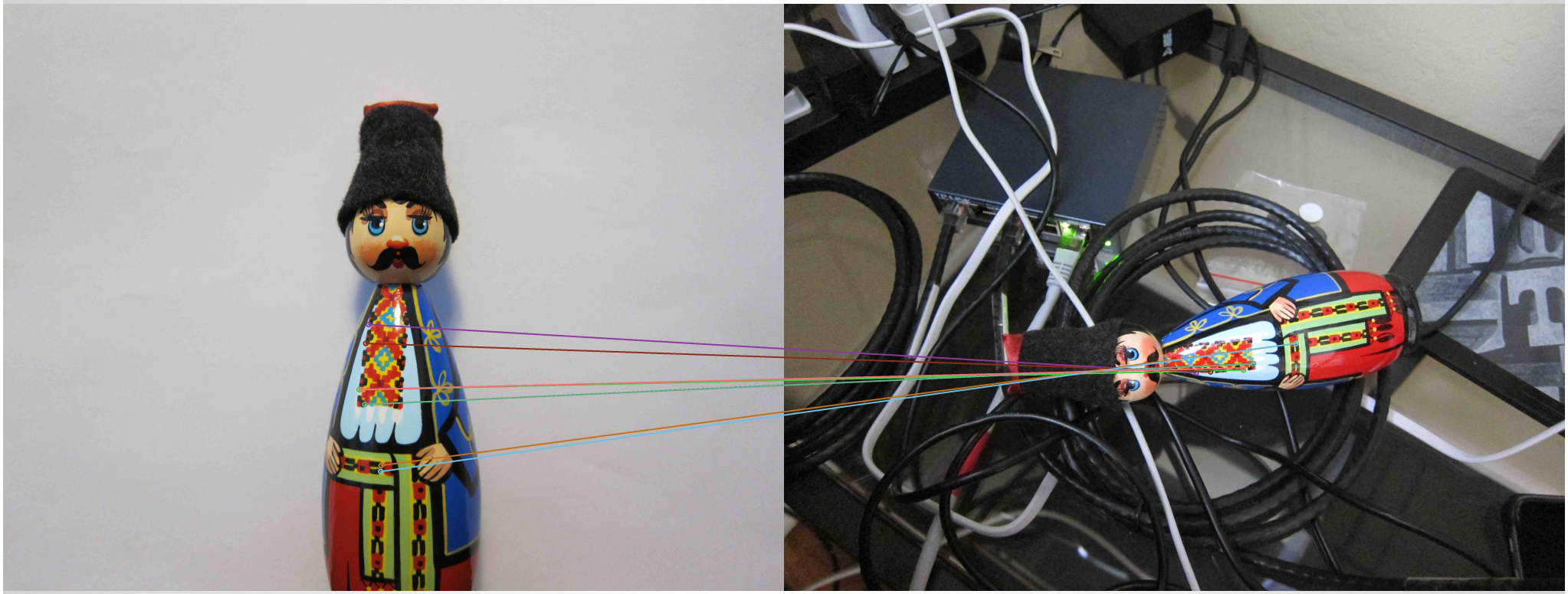
LIGHTING



Quite interesting because another similar picture for lighting produced quite subpar results (included in the previous slides).
(3 overlaps here)

SCORE: 10/10

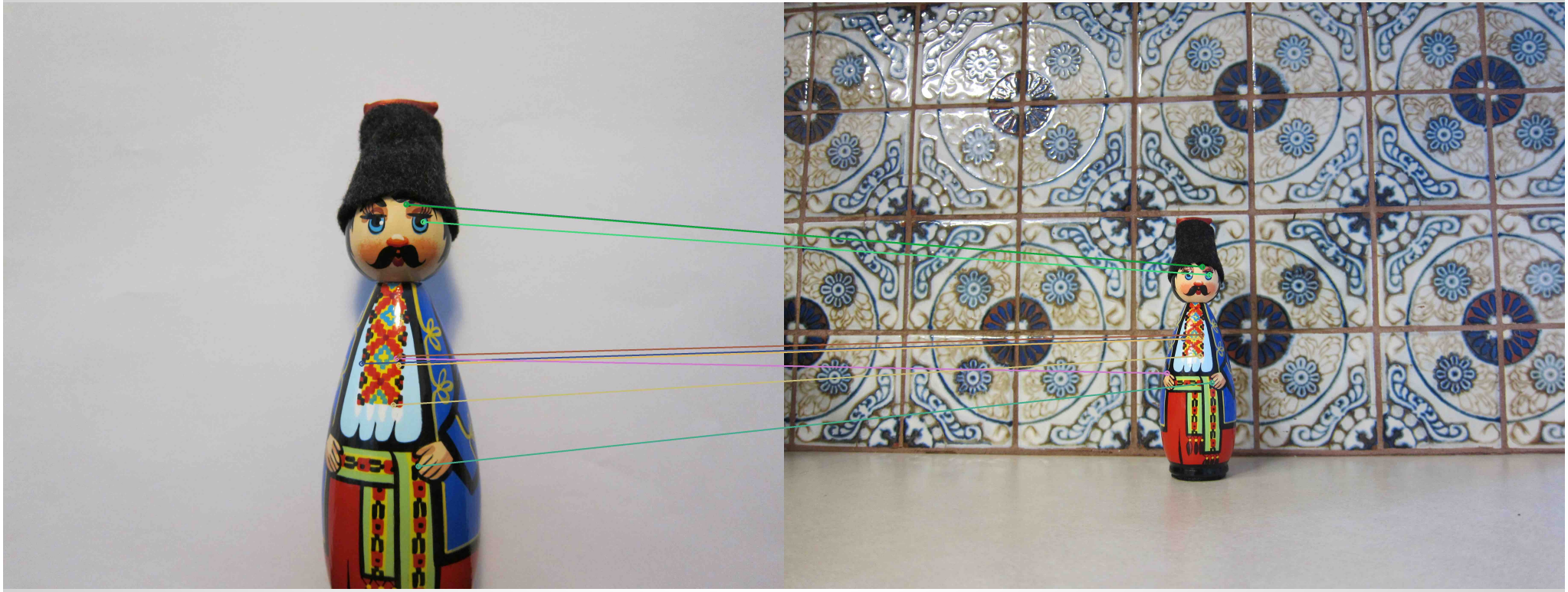
ROTATION



Quite good, although the background could have a few more features
(3 overlaps)

SCORE: 10/10

SCALE



(Comments about your results can go here -- did your algorithm miss any matches? Why do you think these features were matched incorrectly?)

SCORE: 10/10

Summary of Results

Match Set	Correct Matches (out of 10)	Parameter Notes
Sample	10	Camera translation (!)
Lighting	10	Shutter speed (n: 1/50 o:1/80)
Rotation	10	
Scale	10	Background

Using this table or another means, summarize your results. Record any parameters that were changed to generate your results, giving their original and revised settings.

Function: findMatchesBetweenImages

```
orb = cv2.ORB()
kp1, des1 = orb.detectAndCompute(image_1, None)
kp2, des2 = orb.detectAndCompute(image_2, None)
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1, des2)
matches = sorted(matches, key = lambda x:x.distance)
return kp1, kp2, matches[:10]
```

In order to find feature points (FAST algo) and their descriptions (BRIEF algo), I use the built-in ORB() class and its function detectAndCompute() to find the above mentioned items for each image. I then use the brute-force matcher to obtain a “match” of features from one image to another along with a inverse-scores of how close they are to each other (hamming distance). The matches with the lowest 10 scores are returned.

Internally, ORB find co-ordinates of patches in each image that would make good features (i.e. patches with strong corners), finds the Harris-score of each and calculates the BRIEF descriptors for 500 patches with the most harris-scores.

Function: drawMatches

```
(h1, w1) = image_1.shape[:2]
(h2, w2) = image_2.shape[:2]
output = np.zeros((max(h1, h2), w1+w2) + image_1.shape[2:])
output[:h1, :w1] = image_1
output[:h2, w1:] = image_2
for match in matches:
        (p1x, p1y) = image_1_keypoints[match.queryIdx].pt
        (p2x, p2y) = image_2_keypoints[match.trainIdx].pt
        pt1 = (int(p1x), int(p1y))
        pt2 = (int(p2x + w1), int(p2y))
        rand = np.random.randint
        color = (rand(0, 255), rand(0, 255), rand(0, 255))
        cv2.circle(output, pt1, 10, color, 5)
        cv2.circle(output, pt2, 10, color, 5)
        cv2.line(output, pt1, pt2, color, 5, lineType=cv2.CV_AA)
return output
```

To draw matches given the keypoints for both images and the matche sfrom one to the other, I first create an empty output image of the required size. I then copy each of the images onto the output image such that they are shown side by side. Then, for every match in the matches array, I compute the co-ordinates of the keypoints in each of the images (which means the x-cordinate needs to be added to the width of the image on the left) and then I simply draw circles at each of the points and a line between them. Repeat for the rest of the matches.

Discussion of Results

For the scale test against various kitchen equipment, most of the features were matched to the convention oven in the background rather than the doll itself. I suppose this is because the strong corners on the black with white lettering oven had better scores than the corners on the doll itself – and this is shown because the real matches are found by ORB, but filtered out by the ‘top 10’ rule.

This is consistent with one of the most cited drawbacks of ORB – that it is not robust to scale changes.

I also tried to use RANSAC algorithm to select good matches instead of the hamming distance, but I could not reliably get this to work because I couldn’t get good transformation matrices (my guess is due to the wildly varying background) :/

Bibliography & Sources

1. Opencv docs (ORB implementation, canvas drawing functions).
2. Wikipedia (to read up on ORB, FAST and BRIEF algorithms).