get_keypoints

```
sift = cv2.SIFT_create(nfeatures=num_keypoints)
keypoints, descriptors = sift.detectAndCompute(image=image, mask=None)
```

Using SIFT to get keypoints from images.

get_matches

```
bf = cv2.BFMatcher_create(crossCheck=True)
matches = bf.match(descriptors1, descriptors2)
matches = sorted(matches, key=lambda x: x.distance)
```

Perform feature matching and sort keypoint matches in ascending order according to distance.

Convert_lines_pts_to_lines

```
for i in range(len(lines)):
    lines[i] = np.cross(pts[i], other_pts[i])
    lines[i] = lines[i] / lines[i][-1]
```

For every pair of start/end point, get the line joining them using cross product. Then scale line to the [a, b, 1] representation.

get_line_intersections

```
for i in range(len(lines)):
    intersections[i] = np.cross(lines[i], other_lines[i])
    intersections[i] = intersections[i] / intersections[i][-1]
```

For every pair of virtual/real lines, get the intersection using cross product. Then scale intersection point to [u, v, 1] representation.

get_line_crossings

```
v_lines = np.ones_like(r_lines)
for i in range(len(v_lines)):
    v_lines[i] = v_line
line_intersections = get_line_intersections(v_lines, r_lines)
for i, pt in enumerate(line_intersections):
    prod = (pt[0] - r_start_pts[i][0]) * (r_end_pts[i][0] - r_start_pts[i]
[0]) + (pt[1] - r_start_pts[i][1]) * (r_end_pts[i][1] - r_start_pts[i][1])
    sqrlen = (r_end_pts[i][0] - r_start_pts[i][0]) * (r_end_pts[i][0] - r_
start_pts[i][0]) + (r_end_pts[i][1] - r_start_pts[i][1]) *
(r_end_pts[i][1] - r_start_pts[i][1])
    if (prod >= 0 and prod <= sqrlen):
            line_crossings = np.concatenate((line_crossings, pt))
    line_crossings = np.reshape(line_crossings, (-1, 3))
```

Get line intersections between virtual line and real lines. Check if intersection is within the real line segment and intersection point to line_crossing if it is. Then reshape line_crossing to [M, 3] representation.

get_cross_ratios

```
def det(x, y):
    return (x[0] * y[1]) - (x[1] * y[0])

for i in range(len(cross_ratios)):
    cross_ratios[i] = (det(a[i], b[i]) * det(c[i], d[i])) / \
            (det(a[i], c[i]) * det(b[i], d[i]))
```

Following formula given in lab1.pdf. Get the cross ratios of a, b, c, d points.