Assignment 2

Bhavi Dhingra

2018201058

Image Mosaicing

In [1]:

```
import cv2
import imutils
import numpy as np
import ipdb
```

In [2]:

```
class ImageMosaicing:
    def __init__(self):
        self.BLACK_LIMIT=20

    def trim(self, frame):
        #crop top
        if not np.sum(frame[0][:self.BLACK_LIMIT]):
            return self.trim(frame[1:])
        #crop bottom
        elif not np.sum(frame[-1][:self.BLACK_LIMIT]):
            return self.trim(frame[:-2])
        #crop left
        elif not np.sum(frame[:self.BLACK_LIMIT,0]):
            return self.trim(frame[:,1:])
        #crop right
        elif not np.sum(frame[:self.BLACK_LIMIT,-1]):
            return self.trim(frame[:,:-2])
        return frame

    def trim_top_left(self, frame):
        if not np.sum(frame[0][-self.BLACK_LIMIT:]):
            return self.trim_top_left(frame[1:])
        return frame

    def trim_bottom_right(self, frame):
        if not np.sum(frame[-1][-self.BLACK_LIMIT:]):
            return self.trim_top_left(frame[:-2])
        return frame

    def make_panaroma(self, image_folder_path, _id, num_images, ext):
        images = []
        for i in range(num_images):
            image = cv2.imread(image_folder_path + "/img" + str(_id) + "_" + str(i+
            images.append(image)

        result = image_mosaic.stitch_images(images)
        cv2.imwrite("./result/result" + str(_id) + ext, result)


    def stitch_images(self, images, ratio=0.75, reprojThresh=4.0, showMatches=False
        for i in range (len(images)-1):
            image1, image2 = images[i], images[i+1]
            image1 = imutils.resize(image1, width=1000)
            image2 = imutils.resize(image2, width=1000)

            # detect keypoints and extract local invariant descriptors
            (key_points1, features1) = self.detectAndDescribe(image1)
            (key_points2, features2) = self.detectAndDescribe(image2)

            # match features between the two images
            M = self.matchKeypoints(key_points2, key_points1,
                                    features2, features1, ratio, reprojThresh)

            if M is None:
                return None

            # apply a perspective warp to stitch the images together
            (matches, H, status) = M
            result = cv2.warpPerspective(image2, H,
                                         (image2.shape[1] + image2.shape[1], image1
            result[0:image1.shape[0], 0:image1.shape[1]] = image1

            # trim the result image to remove the black regions
            result = self.trim(result)
            result = self.trim_top_left(result)
            result = self.trim_bottom_right(result)
            images[i+1] = result

        return images[-1]



    def detectAndDescribe(self, image):
        descriptor = cv2.xfeatures2d.SIFT_create()
        (key_points, features) = descriptor.detectAndCompute(image, None)

        # convert the keypoints from KeyPoint objects to NumPy array
        key_points = np.float32([kp.pt for kp in key_points])

        return (key_points, features)


    def matchKeypoints (self, key_points1, key_points2, features1, features2, ratio
        matcher = cv2.DescriptorMatcher_create("BruteForce")
        rawMatches = matcher.knnMatch(features1, features2, 2)
        matches = []

        for m in rawMatches:
            # ensure the distance is within a certain ratio of each other
            # (i.e. Lowe's ratio test)
            if len(m) == 2 and m[0].distance < m[1].distance * ratio:
                matches.append((m[0].trainIdx, m[0].queryIdx))

        # computing a homography requires at least 4 matches
        if len(matches) > 4:
            points1 = np.float32([key_points1[i] for (_, i) in matches])
            points2 = np.float32([key_points2[i] for (i, _) in matches])

            (H, status) = cv2.findHomography(points1, points2, cv2.RANSAC, reprojTh
            return (matches, H, status)

        return None
```

In [3]:

```
image_mosaic = ImageMosaicing()
```

Input Images:

https://drive.google.com/open?id=19714aq7YICoiqjhV_XbSv9Hsdf0-gR9g (https://drive.google.com/open?id=19714aq7YICoiqjhV_XbSv9Hsdf0-gR9g)

img1_1.jpg to img1_4.jpg

In [4]:

```
image_mosaic.make_panaroma("./image_mosaicing", 1, 4, ".jpg")
```

img2_1.png to img2_6.png

In [5]:

```
image_mosaic.make_panaroma("./image_mosaicing", 2, 6, ".png")
```

img3_1.png to img3_2.png

In [6]:

```
image_mosaic.make_panaroma("./image_mosaicing", 3, 2, ".png")
```

img4_1.jpg to img4_2.jpg

In [7]:

```
image_mosaic.make_panaroma("./image_mosaicing", 4, 2, ".jpg")
```

img5_1.jpg to img5_4.png

In [8]:

```
image_mosaic.make_panaroma("./image_mosaicing", 5, 4, ".jpg")
```

Own Camera Images

In [9]:

```
image_mosaic.make_panaroma("./image_mosaicing", 6, 3, ".jpg")
```

Output Images:

https://drive.google.com/open?id=16APukMI4QP0sTb18rAcjDI_6pt71o0s- (https://drive.google.com/open?id=16APukMI4QP0sTb18rAcjDI_6pt71o0s-)