In [1]:

```
#program setup

# imports
import cv2
import numpy as np
from matplotlib import pyplot as plt

# ransac package
from skimage.measure import ransac
from skimage.transform import ProjectiveTransform, AffineTransform

import time

# generates matches between 2 given images using the following method:
# complete step 2: using sift to detect local features in an image
# complete step 3: knn tree and ratio testing to select good points
# complete step 4: use ransac inorder to detect inliers in the two images
# it then returns these inlier points
def getMatches(base_img, new_img):

    # step 2 use SIFT on the images
    # initialise sift object
    sift = cv2.xfeatures2d.SIFT_create()

    # run sift
    kp1, des1 = sift.detectAndCompute(base_img, None)
    kp2, des2 = sift.detectAndCompute(new_img, None)

    # set parameters
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks = 50)

    # step 3 use KNN
    # initialise KNN object
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    # run KNN tree
    matches = flann.knnMatch(des1,des2,k=2)

    # ratio test to gather good points
    good = []
    for m, n in matches:
        if m.distance < 0.7 * n.distance:
            good.append(m)

    # add these good points too both point holders
    base_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1, 2)
    new_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1, 2)

    # run Ransac to find inliers
    model, inliers = ransac((base_pts, new_pts), AffineTransform, min_samples=4, resi

    n_inliers = np.sum(inliers)

    inlier_keypoints_base = [cv2.KeyPoint(point[0], point[1], 1) for point in base_pts
    inlier_keypoints_new = [cv2.KeyPoint(point[0], point[1], 1) for point in new_pts[

    d_matches = [cv2.DMatch(idx, idx, 1) for idx in range(n_inliers)]

    base_pts = np.float32([ inlier_keypoints_base[m.queryIdx].pt for m in d_matches ]
    new_pts = np.float32([ inlier_keypoints_new[m.trainIdx].pt for m in d_matches ]).

    # return the inlier points
    return base_pts, new_pts

# takes in two images and their inlier points, finds the homography required to fix
# the destination picture and warps it accordingly while stitching in the source imag
def stitchImages(base_pts, new_pts, base_img, new_img):

    # find homography
    H, masked = cv2.findHomography(new_pts, base_pts, cv2.RANSAC, 5.0)

    # warp and stitch image
    stitched = cv2.warpPerspective(new_img,H,((new_img.shape[1] + base_img.shape[1]),

    # copy new image
    stitched[0:base_img.shape[0], 0:base_img.shape[1]] = base_img #stitched image

    # return the new image
    return stitched

# this function should remove any black borders which maybe present
def removeBorder(img):

    # generate threshold
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    not_needed,thresh = cv2.threshold(gray,1,255,cv2.THRESH_BINARY)

    # find contours
    contours,hierarchy = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_S

    # make rectangles
    x,y,w,h = cv2.boundingRect(contours[0])

    # crop the image
    crop = img[y:y+h,x:x+w]

    # return the cropped image
    return crop
```

In [2]:

```
# load in images
imgs = []

for x in range(1, 6):
    imgs.append(cv2.imread("img_"+str(x)+".jpg", cv2.COLOR_RGBA2BGRA))

    # use the first image as the base
base = imgs.pop(0).copy()

# write the original for comparison
cv2.imwrite("original.jpg", base)
```

Out[2]:

```
True
```

In [3]:

```
count = 1

# try adding all the images to it
while len(imgs) != 0:

    # get new image
    new_img = imgs.pop(0).copy()

    # get the matching points
    base_points, new_points = getMatches(base, new_img)

    # match check
    if (base_points.shape[0] > 10):

        print("matches found" + str(base_points.shape))

        # stitch the image
        stitched = stitchImages(base_points, new_points, base, new_img)

        # remove the border
        base = removeBorder(stitched)

        # save progress
        cv2.imwrite("addition"+str(count)+".jpg", base)

        count+=1

    else:

        print("image rejected as insufficent matches were found")

# save output
cv2.imwrite("output.jpg", base)

print("the end has been reached")
```

```
matches found(285, 2)
matches found(271, 2)
image rejected as insufficent matches were found
matches found(115, 2)
the end has been reached
```