Computer Vision
Report 1
Zou Zijun – B4TB1709

Objective

To create a panoramic picture from four or more images using various theories and concepts studied during the lecture

Environment

Python 2.7.13 :: Anaconda custom (64 bit)
OpenCV 2.4.11
Ubuntu 16.04 LTS

Approach

Since every image contain some common portion with the other images. So the base image will lie either to the left or right of the next image. After capturing the first image, the movement of camera so as to capture the next image is a combination of rotation and translation. Homography is used to visualize one image with respect to another point of view. Thus, homography matrix is used to transform image plane P1 to another image plane P2.
Feature extraction from images is done by using opencv_contrib's SIFT descriptor. For matching correspondence between images, there need to be overlapping points and from these points an idea can be generated so that how the other image can be rotated or overlapped or scaled w.r.t. the first image. This process is called registration. FLANN or BFMatcher can be used for the matching purposes.
After that, when the matches between the images are confirmed, homography matrix can be calculated. This matrix uses these matching points to gauge a relative orientation transform between the two images. Now, once the Homography matrix , H is generated, stitching process is initiated. At this point, it is known how the second image will from the perspective of the first image. To transform it to fit with the first image, process of warping is used. In the program used for this report, planar warping is used to basically change the plane of field of view. Now that a warped image is obtained, the second image is added to the warped image through left or right wherever it fits.

Result

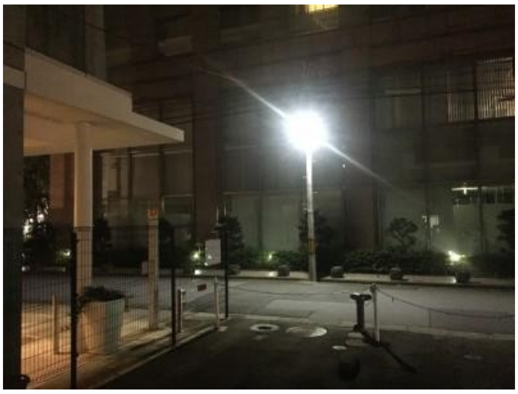

Figure 1 : first

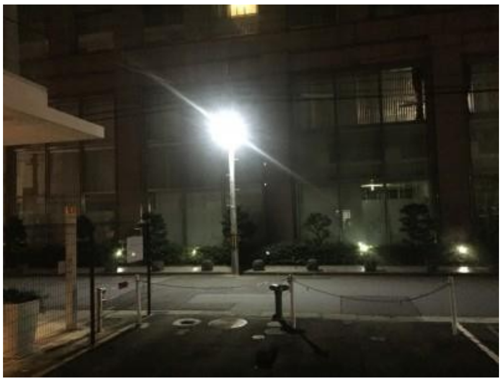

Figure 2 : second

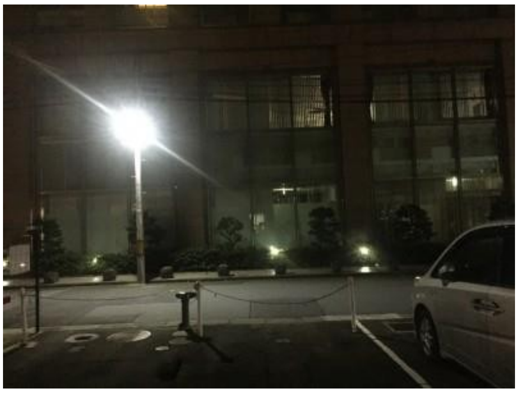

Figure 3: third

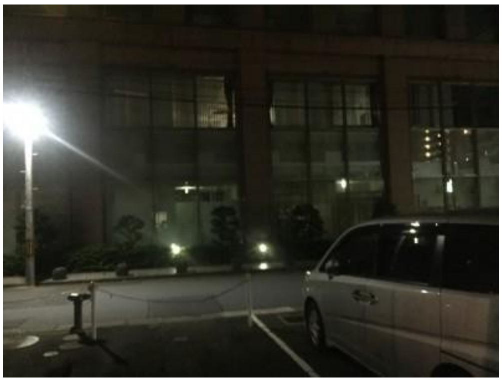

Figure 4: fourth

The Figures 1 to 4 (in the given order) are individual images taken from a phone camera outside the residence. Now, using the program given below, these images are stitched together to become a panoramic image as follows:

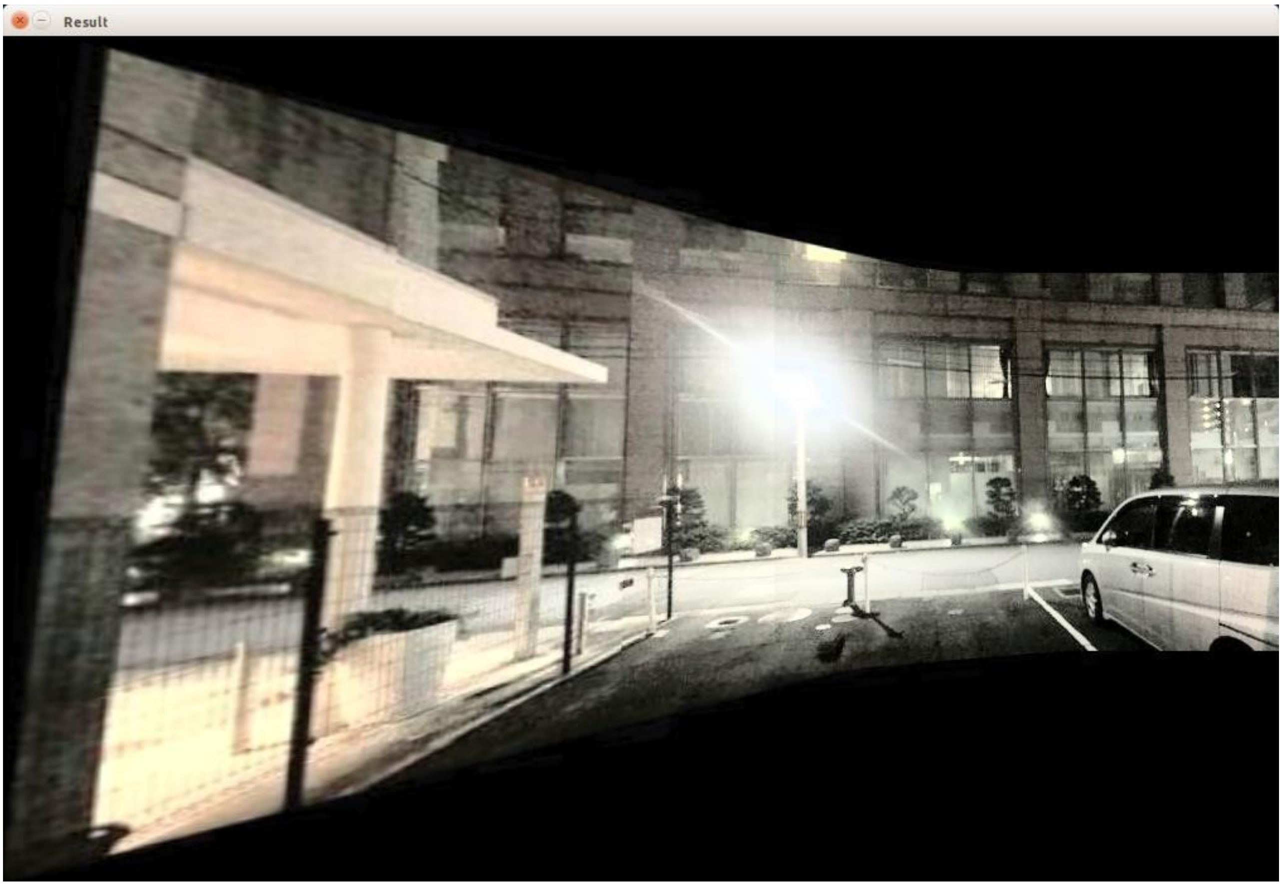

Figure 5: Final Panoramic Image

Figure 5 is the final product of the four images that were used into making a panoramic view of the street.
Since planar warping is used to fit the images with each other, the extra black region on the outside can be seen. The color and the brightness of the image is altered because of the slight changes in

color and brightness of the respective individual images. So, to compensate, color histogram was used to balance the irregularities between the images.

Program

The program below is the main program which imports another short program called “matching.py”. Functions of Final.py are as following:

- It resizes the given images into 400x300 so that they are reasonably small files which can be easily and quickly processed
- It can be used to join however many images user wants. So, it chooses one image as a center image and the other images are later warped and joined to the first image either from left or right as evident by the rightshift and leftshift function
- Function combine basically just combines the left image with the newly formed warped image together

```
#-----------------------------------------------------------------------------------------------------------
#Final.py
import numpy as np
import cv2
import sys
from matching import matching
import time

class Join:
	def_init_(self, args):
		self.path = args
		fp = open(self.path, 'r')
		filenames = [each.rstrip('\r\n') for each in fp.readlines()]
		print filenames
		self.images = [cv2.resize(cv2.imread(each),(400, 300)) for each in filenames]
		self.count = len(self.images)
		self.center_imself, self.left_list, self.right_list = [], [],None
		self.match_obj = matching()
		self.prepare_lists()

	def prepare_lists(self):
		print "Images to be processed : %d"%self.count
		self.centerIdx = self.count/2
		print "Center index image : %d"%self.centerIdx
		self.center_im = self.images[self.centerIdx]
		for i in range(self.count):
			if(i<=self.centerIdx):
				self.left_list.append(self.images[i])
			else:
				self.right_list.append(self.images[i])
		print "Image lists prepared"

	def rightshift(self):
		for each in self.right_list:
			H = self.matcher_obj.match(self.leftImage, each, 'right')
```

```
                    print "Homography :", H
                    txyz = np.dot(H, np.array([each.shape[1], each.shape[0], 1]))
                    txyz = txyz/txyz[-1]
                    dsize
=(int(txyz[0])+self.leftImage.shape[1],int(txyz[1])+self.leftImage.shape[0])
                    tmp = cv2.warpPerspective(each, H, dsize)
                    cv2.imshow("tp", tmp)
                    cv2.waitKey()
                    # tmp[:self.leftImage.shape[0], :self.leftImage.shape[1]]=self.leftImage
                    tmp = self.mix_and_match(self.leftImage, tmp)
                    print "tmp shape",tmp.shape
                    print "self.leftimage shape=", self.leftImage.shape
                    self.leftImage = tmp
            # self.showImage('left')

    def leftshift(self):
            # self.left_list = reversed(self.left_list)
            a = self.left_list[0]
            for b in self.left_list[1:]:
                    H = self.matcher_obj.match(a, b, 'left')
                    print "Homography is : ", H
                    xh = np.linalg.inv(H)
                    print "Inverse Homography :", xh
                    ds = np.dot(xh, np.array([a.shape[1], a.shape[0], 1]));
                    ds = ds/ds[-1]
                    print "final ds ", ds
                    f1 = np.dot(xh, np.array([0,0,1]))
                    f1 = f1/f1[-1]
                    xh[0][-1] += abs(f1[0])
                    xh[1][-1] += abs(f1[1])
                    ds = np.dot(xh, np.array([a.shape[1], a.shape[0], 1]))
                    y_offset = abs(int(f1[1]))
                    x_offset = abs(int(f1[0]))
                    dsize = (int(ds[0])+offsetx, int(ds[1]) + offsety)
                    print "image dsize =>", dsize
                    tmp = cv2.warpPerspective(a, xh, dsize)
                    tmp[y_offset:b.shape[0]+y_offset, x_offset:b.shape[1]+x_offset] = b
                    a = tmp

            self.leftImage = tmp


      def combine(self, leftImage, warpedImage):
            i1y, i1x = leftImage.shape[:2]
            i2y, i2x = warpedImage.shape[:2]
            print leftImage[-1,-1]

            t = time.time()
            black_l = np.where(leftImage == np.array([0,0,0]))
            black_wi = np.where(warpedImage == np.array([0,0,0]))
            print time.time() - t
            print black_l[-1]
```

```
        for i in range(0, i1x):
            for j in range(0, i1y):
                try:
                    if(np.array_equal(leftImage[j,i],np.array([0,0,0]))and
np.array_equal(warpedImage[j,i],np.array([0,0,0]))):
                        # print "BLACK"
                        # instead of just putting it with black,
                        # take average of all nearby values and avg it.
                        warpedImage[j,i] = [0, 0, 0]
                    else:
                        if(np.array_equal(warpedImage[j,i],[0,0,0])):
                            # print "PIXEL"
                            warpedImage[j,i] = leftImage[j,i]
                        else:
                            if not np.array_equal(leftImage[j,i], [0,0,0]):
                                bw, gw, rw = warpedImage[j,i]
                                bl,gl,rl = leftImage[j,i]
                                warpedImage[j, i] = [bl,gl,rl]
                except:
                    pass
        # cv2.imshow("waRPED mix", warpedImage)
        # cv2.waitKey()
        return warpedImage




    def trim_left(self):
        pass

    def showImage(self, string=None):
        if string == 'left':
            cv2.imshow("left image", self.leftImage)
            # cv2.imshow("left image", cv2.resize(self.leftImage, (400,400)))
        elif string == "right":
            cv2.imshow("right Image", self.rightImage)
        cv2.waitKey()


if__name__== '_main_':
    try:
        args = sys.argv[1]
    except:
        args = "images1.txt"
    finally:
        print "Parameters : ", args
    s = Stitch(args)
    s.leftshift()
    # s.showImage('left')
    s.rightshift()
    print "done"
```

```
        cv2.imwrite("final.jpg", s.leftImage)
        print "image is created"
      cv2.destroyAllWindows()
#---------------------------------------------------------------------------------------------------------
```

Now the program below is used for matching the common points on the images

- This program uses FLANN provided by opencv to match points across the images to help with setting the orientation of the second image with respect to the first one

```
#-----------------------------------------------------------------------------------------------------------
#matching.py
import cv2
import numpy as np

class matching:
        def_init_(self):
                self.surf = cv2.xfeatures2d.SURF_create()
                FLANN_INDEX_KDTREE = 0
                index_params = dict(algorithm=0, trees=5)
                search_params = dict(checks=50)
                self.flann = cv2.FlannBasedMatcher(index_params, search_params)

        def match(self, i1, i2, direction=None):
                imageSet1 = self.getSURFFeatures(i1)
                imageSet2 = self.getSURFFeatures(i2)
                print "Direction : ", direction
                matches =
                        self.flann.knnMatch( im
                        ageSet2['des'],
                        imageSet1['des'],
                        k=2
                        )
                good = []
                for i , (m, n) in enumerate(matches):
                        if m.distance < 0.7*n.distance:
                                good.append((m.trainIdx, m.queryIdx))

                if len(good) > 4:
                        pointsCurrent = imageSet2['kp']
                        pointsPrevious = imageSet1['kp']

                        matchedPointsCurrent =
                                np.float32( [pointsCurrent[i].pt for (__, i) in
                                good]
                        )
                        matchedPointsPrev =
                                np.float32( [pointsPrevious[i].pt for (i, __)
                                in good]
                                )

                        H, s = cv2.findHomography(matchedPointsCurrent,  matchedPointsPrev,
cv2.RANSAC, 4)
                        return H
                return None
```

```
	def getSURFFeatures(self, im):
		gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
		kp, des = self.surf.detectAndCompute(gray, None)
	    return {'kp':kp, 'des':des}
#------------------------------------------------------------------------------------------------------------------------
```